International Journal of Applied Operational Research
Vol. 6, No. 1, pp. 13-23, Winter 2016

Journal homepage: ijorlu.liau.ac.ir

An Approach to Increasing Reliability Using Syndrome Extension

H. Hamidi*

Received: 20 July 2015**;** **Accepted:** 10 December 2015

Abstract Computational errors in numerical data processing may be detected efficiently by using parity values associated with real number codes, even when inherent round off errors are allowed in addition to failure disruptions. This paper examines correcting turbo codes by straightforward application of an algorithm derived for finite-field codes, modified to operate over any field. There are syndromes associated with Turbo code words corrupted by large and small error values that relate directly to the induced errors. Hence, after determining the location of a few large errors if possible, the syndromes may be extended so as to yield these large corrupting errors along with approximations to the prevalent low-level errors. The techniques proposed in this paper can be employed in correcting erroneously processed data caused by soft errors.

Keywords: Security, Reliability, Syndrome, Error value.

1 Introduction

Turbo codes coupled with detection and correction procedures have found applications by some researchers in channel and source coding systems [1-5]. Several error-correcting methods for turbo codes were examined in these papers. Most error correction approaches addressed a few large infrequent errors in combination with frequent low-level noise values. In communication and source coding configurations, error correction is necessary because the original data being conveyed are not readily available, unlike computer processing where preprocessed data are temporarily preserved and available for re-computation. The impulsive noise situations also arise naturally in high-performance processing where “soft errors” [6] cause a few infrequent large values to appear in processed outputs. The extended syndromes are subjected to an inverse DFT exposing the large error values for cleansing code words, ultimately revealing the encoded data.

The important method of code combination known as product, this product method produces excellent codes. Very low-rate Convolutional codes can be constructed by taking products of binary Convolutional codes and block repetition codes. Example of codes in this class includes turbo codes [1]. For the first time, in this paper we present methods for employ turbo codes into systematic forms. Security techniques are most effective when applying a systematic form. The redundancy necessary for the security method is commonly defined by real number codes, generally of the block type [2-6]. Security introduced by K.H.Huang and Abraham [2], was at first devoted to matrix operations on systolic arrays. It has been used to

* **Corresponding Author. (✉)**
E-mail: h_hamidi@kntu.ac.ir (H. Hamidi)

H. Hamidi
Information Technology Engineering Group, Department of Industrial Engineering, K.N. Toosi University of Technology, Tehran, Iran.

[Downloaded from ijaor.ir on 2026-11-28]

reduce redundant hardware. Security technique is distinctive by three characteristics: (a) Encoding the input sequence, (b) Plan again of the algorithm to act on the encoded input sequence, (c) Distribution of the redundant computational steps among the individual computational units in order to adventure maximum parallelism. The input sequences are encoded in the form of error detecting or correcting codes. The modified algorithm operates on the encoded data and produces encoded data output, from which useful information can be recovered very easily. Obviously, the modified algorithm will take more time to operate on the encoded data when compared to the original algorithm; this time redundant must not be excessive. Security for arithmetic and numerical processing operations is based on linear codes. G. Bosilca et al. [7] for high-performance computing, propose a new security method based on a parity check coding. In [8] is the application of Low Density Parity Check based security, it compare and analyses the use of LDPC to classical Reed-Solomon codes with regards to different fault models. But, [8] did not provide a method for constructing LDPC codes algebraically and systematically, such as RS and BCH codes are constructed, and LDPC encoding is very complex due to the miss of appropriate structure. Security methodologies used in [9] present parity values dictated by a real convolutional code for protecting linear processing systems. Paper [10] introduces a class of Convolutional codes which is called burst-correcting Convolutional codes; these codes provide error detection in a continuous mode using the same computational resources as the algorithm progresses. Redinbo [11] presented a method to Wavelet Codes into systematic forms for Algorithm-Based Fault Tolerance applications. This method employ high-rate wavelet codes along with low-redundancy which use continuous checking attributes to detect the errors, in this paper since their descriptions are at the algorithm level can be applied in hardware or software. But, this technique is suited to image processing and data compression applications and is not a general method. Also, other constraint is on burst-error due to computational load high relatively. Moreover, there is onerous analytical approach to exact measures of the detection performances of the security technique applying wavelet codes.

We make the following contributions in this paper: In section 2, we discuss our block diagram of the security technique. In section 3, we propose the usage of codes, turbo codes for security technique. In section 4, we discuss the simulations and results. In section 5, discussion of the conclusions.

2 Security Schemes

For error correction purposes, redundancy must be inserted in some form and, using the security, turbo parity codes will be employed. A systematic form of turbo codes is especially profitable in the security detection plan because no redundant transformations are needed to achieve the processed data after the detection operations. To achieve fault detection and correction properties of turbo code in data processing with the minimum additional computations, we propose the block diagram in Fig 1. This figure summarizes a security technique employing a systematic turbo code to define the parity values. The k is the basic block size of the input data, and n is block size of the output data, new data samples are accepted and (n - k) new parity values produced.

[Downloaded from ijaor.ir on 2026-11-28]

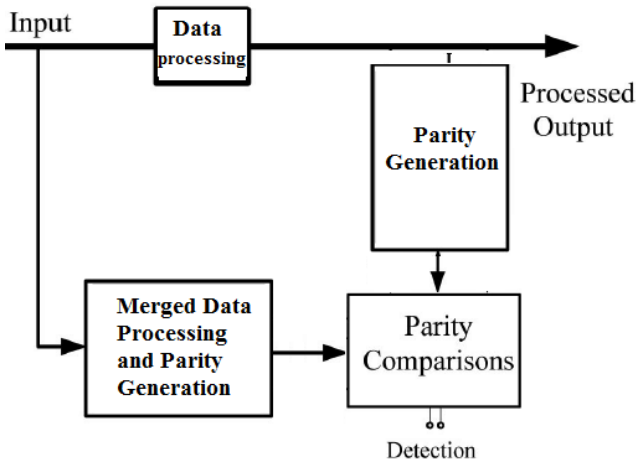


Fig. 1 Block diagram of the security technique

The upper way, Fig. 1, is the Process data flow which passes through the process block (data processing block) and then fed to the turbo encoder (parity regeneration) to make parity values. On the other hand, the comparable parity values are generated efficiently and directly from the inputs, parity and processing combined, without producing the original outputs. The security method detects errors whenever these two parity values do not compare within a tolerance threshold. The difference in the comparable two parity values, which are computed in different ways, is called the syndrome; the syndrome sequence is a stream of zero or near zero values. The turbo code's structure is designed to produce distinct syndromes for a large class of errors appearing in the processing outputs. Fig. 1 employs turbo code parity in detecting and correcting processing errors.

3 Turbo Code

3.1 Turbo Encoder

Early Most of the researchers tend to focus on methods are structured, such as RM and BCH codes with very strong algebraic structures, or topological, such as Convolutional codes [12]. Anyway, structures does not always result in the best distance properties for code, and can be produced very complex operations. Special types of Convolutional codes, called recursive systematic Convolutional codes (RSC), are used as the building blocks of a turbo code encoder, Fig.2 (a). The basic turbo code encoder is built using two identical recursive systematic Convolutional (RSC) codes with parallel concatenation [1]. The two component encoders are separated by an interleaver, only one of the systematic outputs from the two component encoders is used, because the systematic output from the other component encoder is just a permuted version of the chosen systematic output, Fig. 2(b). A turbo code encoder with two component codes is shown in the Fig. 2(b).

[Downloaded from ijaor.ir on 2026-11-28]

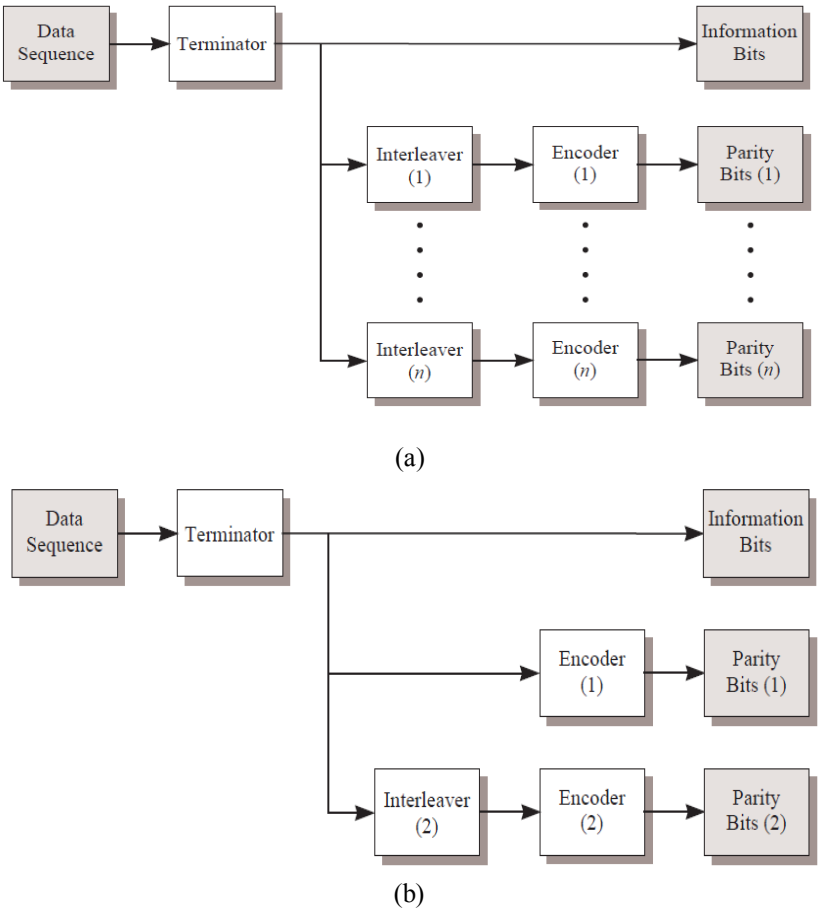


Fig. 2 (a) Block diagram of the generalized turbo encoder, (b) Block diagram of the Two-Component Turbo Encoder

In the general case, the code consists of two parts: the un-coded input values and a set of parity sequences generated by passing interleaved versions of the information bits through Convolutional encoders. Typically, the encoders used are Recursive Systematic encoders; also, in most turbo codes the encoders used are the same (making the Turbo code symmetric), and two sets of parity values are used, one which is generated from the non-interleaved data sequence, and one which is generated from an interleaved sequence. This structure is shown schematically in Fig. 2(b). The parity values are usually punctured in order to raise the code rate to, $R=k/n$, 1/2. The data sequence may or may not be terminated, usually depending on the kind of interleaver used. We will assume that the input sequence contains k input values and is represented by $X^{(0)}$.

The number of data carrying symbols is denoted by K, implying that there are $(N\text{-}K)$ parity symbols associated with an (N, K) turbo code. The minimum Hamming distance of an (N, K) turbo code is $(N\text{-}K+1)$ indicating an error-correcting capability of

$$[(N\text{-}K+1)/2]. \quad (1)$$

The turbo is defined through an $N.N$ matrix Γ and it is identified by selecting (N-K) consecutive rows of Γ as a parity check matrix H, $(N\text{-}K)\cdot N$. A complex vector v, $N\cdot 1$, is a codeword if and only if $\underline{0} = H\underline{v}$; $\underline{0}$, $(N\text{-}K)\cdot 1$ all zeros.

[Downloaded from ijaor.ir on 2026-11-28]

The processed, transmitted or stored data are contained in a codeword r_1 which may be affected by errors due to computer operations, noisy transmission or storage. The disruptive influences may impact all N components of r_1 , and also just a few values drastically. The corruption may be modeled by adding random vectors to r_1 .The observed, corrupted codeword will be denoted by $\underline{\rho}$, $N \cdot 1$. The corrupted codeword is defined through

$$\rho = r_1 + r_2 + r_3 \tag{2}$$

The N components of r_3 represent low-level noise while those in r_2 are a few nonzero components modeling the impulsive noise, the infrequent, large disruptive values. The low-level noises in r_3 are Gaussian with uncorrelated components, zero-mean and small variances σ^2 . On the other hand, the components of r_2 obey a Gaussian process in which each element r_{2i} is nonzero with small probability p and accordingly zero with probability (1-p).
When a nonzero element r_{2i} appears, its numerical value follows a zero-mean, larger variance σ^2, Gaussian law. The components of r_2 and r_3 are assumed statistically independent among and between elements. The first step towards correction is detecting errors, particularly large errors modeled in vector r_2 . The syndromes

$$S = (S_1, S_2, ..., S_{N-K})^T,$$

$$S = H(r_2 + r_3) \tag{3}$$

associated with the parity check matrix H (and forward transform) provide a window on the errors appearing in an observed possibly corrupted codeword $\underline{\rho}$. Note

$$H.r_1 = 0 \tag{4}$$

A reasonable test for determining if any nonzero components of r_2 have occurred is to examine the mean-squared value of each component of S. If any computed mean square syndrome value exceeds a threshold E , an error detection is declared.

$$|S_i|^2 \geq E \tag{5}$$

The threshold E will be taken as a multiple M times the square-root of the statistical mean-squared expectation when there are only low-level errors in r_2;

$$E = M.\sqrt{N.\sigma^2} \tag{6}$$

3.2 Iterative Turbo Code Decoder

In this paper, the turbo code decoder is based on a modified Viterbi algorithm that includes reliability values to improve decoding performance. The Viterbi algorithm produces the majority logic (ML) output value for Convolutional codes. This algorithm provides optimal sequence estimation for one stage Convolutional codes. For concatenated Convolutional

[Downloaded from ijaor.ir on 2026-11-28]

codes, there are two main disadvantages to conventional Viterbi decoders. First, the inner Viterbi decoder produces bursts of bit errors which reduce the performance of the outer Viterbi decoders [13-15]. Second, the inner Viterbi decoder produces hard decision outputs which prevent the outer Viterbi decoders from deriving the advantage of soft decisions [16-19]. Both of these drawbacks can be reduced and the performance of the overall concatenated decoder can be improved if the Viterbi decoders are able to produce reliability (soft-output) values [20]. The reliability values are passed on to subsequent Viterbi decoders as apriority information to improve decoding performance. The Viterbi algorithm was modified to output bit reliability information [22]. The soft-output Viterbi algorithm (SOVA) computes the reliability of the input values as a log-likelihood ratio (LLR),

$$\Lambda(\mathrm{X}^{(0)}) = \log\left(\frac{\Pr[\mathrm{X}^{(0)}=1 \mid \bar{\mathrm{r}}]}{\Pr[\mathrm{X}^{(0)}=0 \mid \bar{\mathrm{r}}]}\right) \tag{7}$$

where $\bar{r}$ denotes the received sequence. In an iterative decoding procedure, the output information provided by $\Lambda_{i,e}(\mathrm{X}^{(0)})$ can be fed back to the decoder as a priori probability for a second round of decoding. The output LLR can be written as, Let $\mathrm{D}_i^{(j)}$ be the set of branches connecting state $S_{i-1}^{(l')}$ to state $S_i^{(l)}$ such that the associated information bit $\mathrm{X}^{(0)}$=j, with j$\in${0, l}.

$$\Lambda_{i,e}(\mathrm{X}^{(0)}) = \log\left(\frac{\sum_{(l,l')\in D_i(0)} \zeta_{i-1}(l')\gamma_i(l,l')\eta_i(l)}{\sum_{(l,l')\in D_i(1)} \zeta_{i-1}(l')\gamma_i(l,l')\eta_i(l)}\right) \tag{8}$$

where $\zeta_{i-1}(l')$, $\eta_i(l)$ and $\gamma_i(l,l')$ are given by

$$\zeta_i(l) = \Pr\{S_i^{(l)}, \mathrm{r}'_i\} \tag{9}$$

$$\eta_i(l) = \Pr\{\mathrm{r}'_i \mid S_i^{(l)}\} \tag{10}$$

$$\gamma_i(l,l') = \delta_{ij}(l.l')\exp\left(\frac{E}{N_0}\sum_{m=1}^{n-1} r_{i,m}x_{i,m}\right) \tag{11}$$

where

$$\delta_{ij}(l.l') = \begin{cases} 1 & ,if\ (l',l) \in D_i^{(j)} \\ 0 & else \end{cases} \tag{12}$$

This output LLR, for an information position *i*, does not contain any variable directly related to $\mathrm{X}_i^{(0)}$, for *i* = 1, 2, . . ., N. It should be noted that because of the assumption that encoding is systematic, and therefore the sum in the modified branch metric (11) starts at *m* = 1. In the

[Downloaded from ijaor.ir on 2026-11-28]

more general case of a two-dimensional product coding scheme, the first decoder produces $\Lambda^{(1)}{}_{i,e}(X^{(0)})$ which is given to the second decoder as a priori probability $\Lambda^{(2)}{}_{i,e}(X^{(0)})$ to be used in the computation of the LLR of input values $X_i^{(0)}$. In other words, the output information provides a soft output that involves only reliabilities that are not directly related to the information symbol $X_i^{(0)}$. The operation of the Turbo decoder is shown in Fig. 3, [21-22], each iteration consists of two phases, one decoding phase per component decoder. First phase, In the first decoding iteration the soft-in soft-out decoder for the first component code computes the a posteriori LLR ,(8).This decoder computes the extrinsic information for each information symbol, $\Lambda^{(1)}{}_{i,e}(X^{(0)})$, on the basis of the part of the received sequence that corresponds to the parity symbols, $\bar{r}_{p1}$, and sends the result to the second decoder.

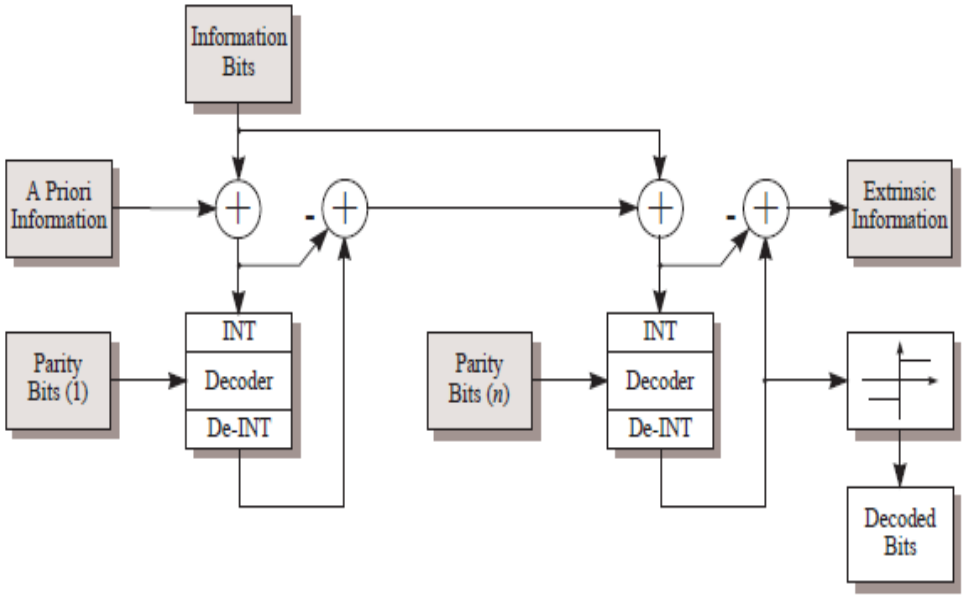


Fig. 3 Two-Component Turbo Decoder

In the second phase of the first decoding iteration, the permuted (or interleaved) output information from the first decoder is used as a priori LLR, $\pi . \Lambda^{(1)}{}_{i,e}(X^{(0)})$. Extrinsic information $\Lambda^{(2)}{}_{i,e}(X^{(0)})$ is computed on the basis of the part of the received sequence that corresponds to the parity values of the second component code, $\bar{r}_{p2}$, thus conclude the first decoding iteration. At this point, a decision can be made on an information symbol, on the basis of its a posteriori LLR $\Lambda_{i,e}(X^{(0)})$. In subsequent iterations, the first decoder uses the de interleaved extrinsic information from the second decoder, $\pi^{-1} . \Lambda^{(2)}{}_{i,e}(X^{(0)})$, as a priori LLR for the computation of the soft-output (the a posteriori LLR), $\Lambda_{i,e}(X^{(0)})$. This procedure can be repeated until either a stopping criterion is met [23-24] or a maximum number of iterations is performed. It should be noted that making decisions on the information symbols after the first decoder saves one deinterleaver.

[Downloaded from ijaor.ir on 2026-11-28]

4 Simulations and Results

A detailed stochastic analysis of the correcting methodology espoused in the previous section seems quite intractable. Extensive simulations employing MatLab scripts are used to evaluate the probability of error and mean-squared error performances of a few selected turbo codes, all triple error-correcting type of high rate, relatively long. The scripts allow separating error influences that impact error-correcting procedures. The sum of squares of the syndromes guides the detection of possible errors. The algorithm follows the methods given earlier including adjusting the roots of the error-modeling polynomial, the nonlinear step. The MatLab code makes comparisons of possibly corrected code words and counts any improper corrections or undetected small errors. All turbo codes simulated are of the turbo codes so that the results are similar to types in other earlier papers, even though most of them were of much shorter length. The probability of code word error curves are shown in Fig. 4 for various large excursion probabilities p and for two quite high standard deviations of the low level noise; $\sigma = 10^{-4}$ and $\sigma = 10^{-2}$. The theoretical value of no coding employing N=25 data symbols, which is $(1-(1-p)^N)$, is displayed in the upper part of each plot. The upper panel of Fig. 4 for shows a coding gain for the (32, 25) and (39, 32) codes while at the higher values of p, the overload effects of long codes are demonstrated, particularly for the (50, 43) code.

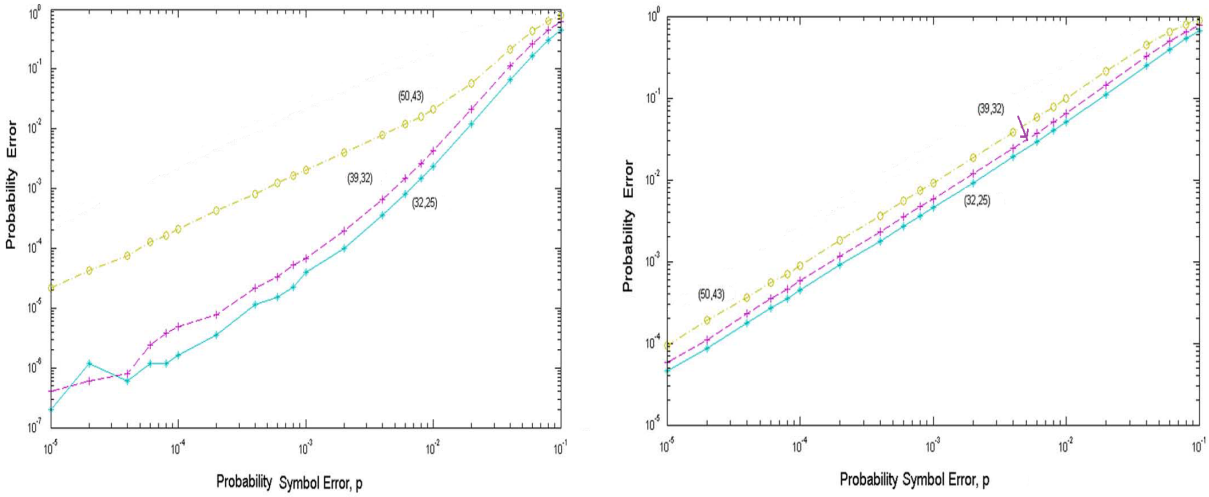


Fig. 4 Probability of code word error for three turbo type codes. (a) Low level noise, standard deviation $\sigma = 10^{-4}$. (b) Low level noise, standard deviation $\sigma = 10^{-2}$.

The curves are not smooth at the lower end of the p range because fewer errors are encountered for averaging purposes. When the low level noise becomes more serve with standard deviation $\sigma = 10^{-2}$, all codes show degradation for the larger values of p. The mean-squared errors between input code word symbols and those in the corrected code words are computed for high deviations of the low level noise with $\sigma = 10^{-4}$. Those results are shown in Fig. 5. The effects of overloaded code word correction by the Bernoulli-Gaussian model is visible for the longest code (50, 43). The impact of low level noise on the recursive extension process is examined in the simulations by determining sample means and variances of the low level processes and their extensions. The simulations easily isolate the low level noise processes and their sample means and variances are evaluated. Both the input processes and the output of the extension operations should have zero means. Fig. 6 shows the average mean-squared error values for the coded and unprotected systems. The scales are both

[Downloaded from ijaor.ir on 2026-11-28]

logarithmic. There is very little change in the error levels when activity probabilities change 0.005 from to 0.05. However, the improvement in the average mean-squared error for the coded system varies from three to four orders of magnitude better than the unprotected levels.

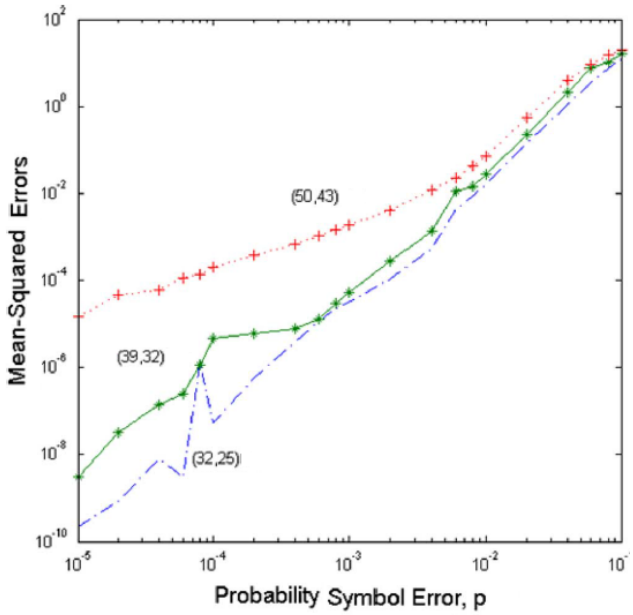


Fig. 5 Probability of codeword error for three codes.

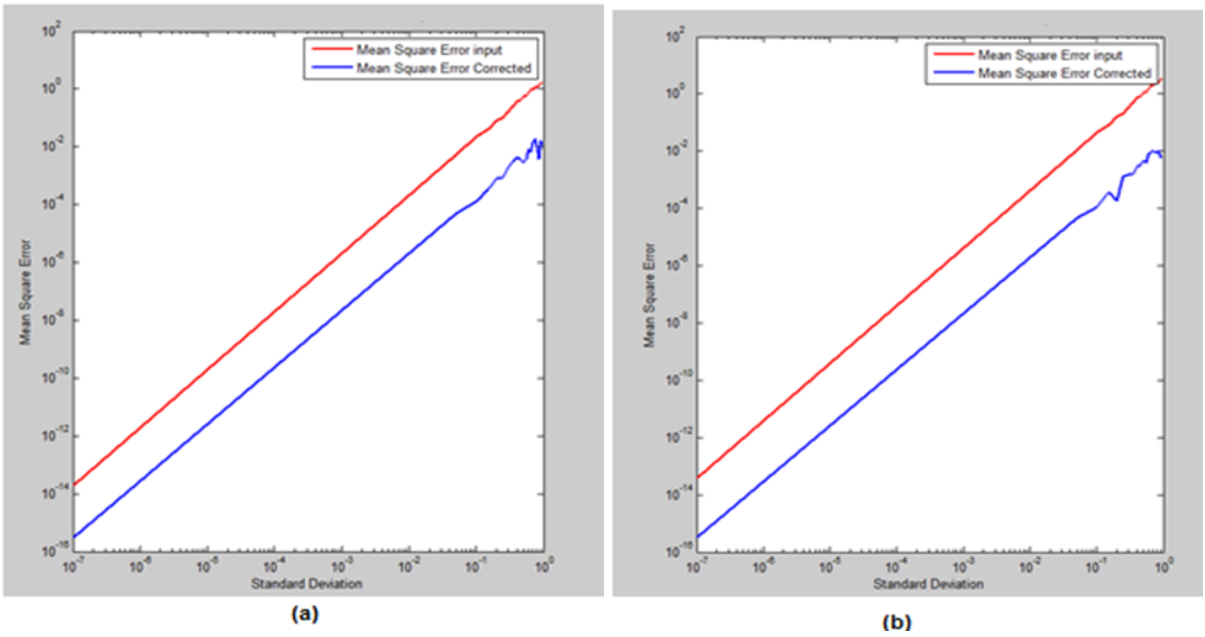


Fig. 6 Sample average mean-squared error real number (71, 64) code (a) Average mean-squared error for activity factor 0.005. (b) Average Mean-squared error for activity factor 0.05

[Downloaded from ijaor.ir on 2026-11-28]

5 Conclusion

Straightforward error correction of turbo codes employing for large error location followed by syndrome extension is a reasonable approach for high rate codes facing small, large error probabilities. This leads to error value computations by extending the starting syndromes through a recursive evaluation, a linear feedback shift register type computation. The low level interferences perturb the error location procedures more than the extension operations. Simulations show that the sample variances of the low level noise effects riding on the true error value evaluations have small variances, one or two orders above those of the noise variances alone. Such decoding methods can be appropriate for certain error regions when employing turbo codes.

For future work, the problem of interleaver design for punctured codes also necessitates an appropriate interleaver optimisation. Some work has already been done on punctured codes, particularly indicating the usefulness of an odd-even interleaver structure. Modification of the simulated annealing algorithm to create odd-even interleavers is trivial1, and should provide a preliminary design strategy. The effect of trellis termination on punctured codes needs to be investigated, particularly for small block sizes.

References

1. Berlekamp, E. R. (1962). A Class of Convolution Codes. Information and Control, 6, 1-13.
2. Berrou, C., Glavieux, A., Thitimajshima, P., (1993). Near Shannon limit error-correcting coding and decoding: Turbo-codes. In Proceedings of the IEEE International Conference on Communications, 2, 1064–1070.
3. Hagenauer, J., Hoher, F., (1989). A Viterbi Algorithm with Soft-Decision Outputs and Its Applications. Proc. 1989 IEEE Global Teleconim Con & (GLOBECOM'89). 47.1.1- 47.1.7, Dallas, Texas.
4. Chen, Z., (2008). Extending Algorithm-based Fault Tolerance to Tolerate Fail-stop Failures in High Performance Distributed Environments. Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium, DPDNS'08 Workshop, Miami, FL, USA, April 14-18.
5. Costello, D., Lin, S., (2004). Error Control Coding Fundamentals and Applications. 2nd edition, Pearson Education Inc., NJ, U.S.A.
6. El Gamal, H., Hammons, A. R., Jr. (2001). Analyzing the turbo decoder using the Gaussian approximation. IEEE Transactions on Information Theory, 47, 671–686.
7. Bosilca, G., Delmas, R., Dongarra, J., Langou, J., (2009). Algorithm-based fault tolerance applied to high performance computing. Journal of Parallel and Distributed Computing, Elsevier, 69(4), 410-416.
8. Hagenauer, J., Offer, E., Papke, L., (1996). Iterative decoding of binary block and convolutional codes. IEEE Trans. Inform. Theory, 42, 429-445.
9. Hamidi, H., Vafaei, A., Monadjemi, A. H., (2009). Algorithm based fault tolerant and checkpointing for high performance computing systems, J. Applied Sci., 9, 3947-3956.
10. Hamidi, H., Vafaei, A., Monadjemi, S. A., (2012). Analysis and Evaluation of a New Algorithm Based Fault Tolerance for Computing Systems. International Journal of Grid and High Performance Computing (IJGHPC), 4(1), 37-51. doi:10.4018/jghpc.2012010103
11. Redinbo, G. R. (2010). Wavelet Codes for Algorithm-Based Fault Tolerance Applications. IEEE Transactions on Dependable and Secure Computing, 7, 3, pp. 315-328.
12. Huang, K. H., Abraham, J. A., (1984). Algorithm-based fault tolerance for matrix operations, IEEE Transactions on Computers, C-33, 518-528.
13. Massey, J. L., (1965). Implementation of Burst-Correcting Convolutional Codes. IEEE Trans. Information Theory, 11, 416-422.

[Downloaded from ijaor.ir on 2026-11-28]

14. Morelos-Zaragoza, R. H., (2006). The Art of Error Correcting Coding. 2nd Edition, John Wiley & Sons, ISBN: 0470015586.
15. Proakis, J. G., (2001). Digital Communications. 4th Edition. New York: McGraw Hill.
16. Redinbo, G. R., (1998). Generalized Algorithm-Based Fault Tolerance: Error Correction via Kalman Estimation. IEEE Transactions ON Computers, 47(6).
17. Redinbo, G. R., (2003). Failure-Detecting Arithmetic Convolutional Codes and an Iterative Correcting Strategy. IEEE Transactions on Computers, 52(11), 1434-1442.
18. Hamidi, H., Vafaei, A., Monadjemi, S. A., (2011). A framework for ABFT techniques in the design of fault-tolerant computing systems. EURASIP Journal on Advances in Signal Processing 2011:1, 90. Online publication date: 1-Jan-2011.
19. Roche,T., Cunche, M., Roch, J. L., (2009). Algorithm-Based Fault Tolerance Applied to P2P Computing Networks. ap2ps, 2009 First International Conference on Advances in P2P Systems, 144-149.
20. Sadjadpour, H., Sloane, N., Salehi, M., Nebe, G., (2001). Interleaver design for turbo codes. IEEE Journal on Selected Areas in Communications, 19, 831-837.
21. Snasel, V., Platos, J., Kromer, P., Ouddane, N., (2008). Genetic Algorithms Searching for Turbo Code Interleaver and Solving Linear Ordering Problem. Cisim, 2008 7th Computer Information Systems and Industrial Management Applications, 71-77.
22. Viterbi, A. J., Omura, J. K., (1985). Principles of Digital Communication and Coding. Mc-Grawhill , 2-nd Print.
23. Wu, P. H. Y., (2001). On the complexity of turbo decoding algorithms. In Proceedings of the IEEE Vehicular Technology Conference-Spring, 2, 1439–1443.
24. Zhang, C. N., Liu, X. W., (2009). An algorithm based mesh check-sum fault tolerant scheme for stream ciphers. International Journal of Communication Networks and Distributed Systems, 3(3), 217-233.

[Downloaded from ijaor.ir on 2026-11-28]