Design and Evaluation of High Performance Error-Correcting Codes

1. Introduction

This document details the work completed for the EE447 Code Design Challenge (CDC) of Fall Semester 2002. Several different error-correcting codes were designed, simulated, and evaluated based on their error-correcting capabilities and required processing time. Part 2 of this paper describes the details of the CDC. Part 3 describes the process of designing a rate 1/3 Turbo Code (TC) for communication over a binary input additive white Gaussian noise (BIAWGN) channel. The design process for a rate 1/7 Repeat Accumulate (RA) and rate 1/7 repetition code for communication over a binary symmetric channel (BSC) is given in Part 4. Part 5 gives details on running the Matlab m-files used for this project. A summary of the work completed for this project is given in Part 6.

2. Code Design Challenge

The purpose of the CDC was to design error-correcting codes (ECC) of various rates for use on several different types of communication channels. Four possible communication scenarios were presented to the class for consideration. Students worked in teams of 3 or 4 and each team was required to design a code for two of the four scenarios listed below in Table 1.

Table 1 – Communication Scenarios

Scenario	Channel	Rate
A	BSC	$\geq 1/7$
B	BSC	$\geq 5/6$
C	AWGN	$\geq 1/3$
D	ISI	$> 1/2$

The goal in designing these error-correcting codes was to minimize the signal-to-noise ratio (SNR) at two fixed bit error rates (BER), 10^{-3} and 10^{-5}, while also minimizing processing time.

3. Rate 1/3 Turbo Code Design

The rules of the CDC allowed teams to use existing software written by other authors as long as the work was properly referenced. Because of this, existing Matlab m-files were downloaded and used a starting point for the Turbo Code design [1]. These m-files were then modified in order to comply with the CDC requirements. Lines of additional comments were also added within the source code to describe decoding and encoding functions more clearly. A final version of the software used to simulate the Turbo Code system was archived and posted on Blackboard.

Having obtained the basic building blocks for simulating TC's, a systematic simulation process was begun to determine the effect of various code parameters on code performance and running time. The following sections describe the simulations, simulation results, and conclusions reached from this process. A block diagram of the final design can be seen in Figures 6 and 7.

3.1 – Interleaver Length

The probability of bit error (P_b) for a TC is inversely related to interleaver length as given by the following equation presented in class [2]:

$$P_b \leq \frac{1}{N} e^{-Rd_{\text{free, eff}} \frac{E_b}{N_0}}$$

Table 2 – Interleaver Length

Interleaver Length	P_b	bits/s	% Change P_b	% Change bits/s
128	2.01e-2	247.2	--	--
256	1.03e-2	242.7	48.66%	1.85%
512	6.32e-3	233.4	19.97%	4.06%
1024	4.24e-3	237.5	10.35%	-1.85%
2048	3.41e-3	205.4	4.12%	16.25%
4096	2.99e-3	178.0	2.05%	18.51%
8192	2.89e-3	114.6	0.50%	76.89%
16384	2.17e-3	62.77	3.60%	178.05%

Therefore, for everything else equal, a TC with a long interleaver length will outperform a TC with a short interleaver length in terms of BER. However, this performance improvement comes at a cost since it was found that processing time increases with interleaver length.

With these tradeoffs in mind, simulations were performed to find good performing codes with reasonable processing time. The results of these simulations are listed in Table 2. All of simulations listed in Table 2 were for an AWGN channel with 1dB SNR, generator matrices $G_1 = 7$, $G_2 = 5$, MAP decoding with 3 iterations, and 131,072 total transmitted bits. Processing time was measured by recording the average number of bits encoded and decoded per second (bits/s). This was accomplished by using the Matlab "tic" and "toc" functions. The percent change listed is the percent difference between the current and previous interleaver lengths. This percent difference was plotted in Figure 2. From this figure it was determined that a random interleaver of length 4096 would be used.

3.2 – Recursive Systematic Convolutional Code

After determining to use an interleaver of length 4096, several different generator matrices for the recursive systematic convolutional encoders were experimented with. Table 3 lists the simulations performed and results obtained for this process. From these simulations it was decided tat $G_1 = 13$, $G_2 = 15$ would be used for the RSC's.

Figure 2 – Interleaver Length

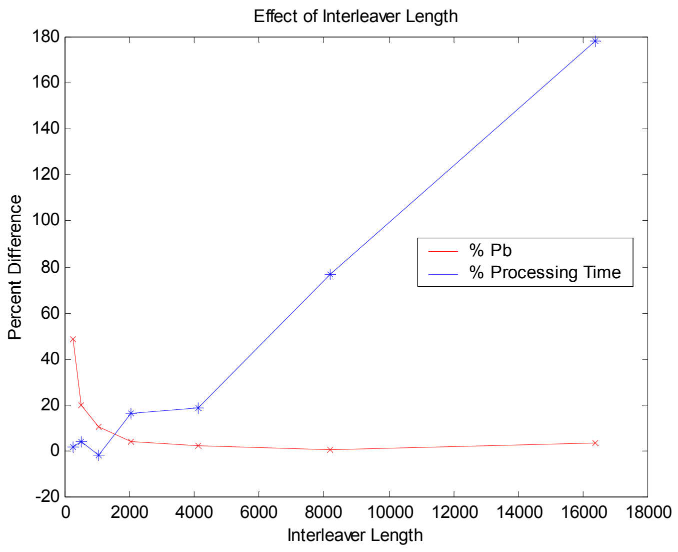


Table 3 – Interleaver Length

Constraint Length	G_1	G2	P_b	bits/s
3	7	5	2.27e-3	181.7
4	13	15	2.90e-4	93.77
5	31	17	2.00e-4	22.34

3.3 – Decoding Iterations

As a final step in designing a rate 1/3 Turbo Code, numerous simulations were performed with varying number of MAP decoding iterations. The effect of decoding iterations on code performance can be seen in Figure 3. The effect of decoding iterations on processing rate (run-time) can be seen in Figure 4. From these simulations it was decided that 5 decoding iterations would be used. Having determined the final parameter of our rate 1/3 Turbo code, a final simulation was performed. The results of this simulation are listed in Table 4. The final performance curve for this error-correcting code is plotted in Figure 5. Block diagrams of the encoder and decoder structures can be seen in Figures 6 and 7.

Figure 3 – Effect of Decoding Iterations

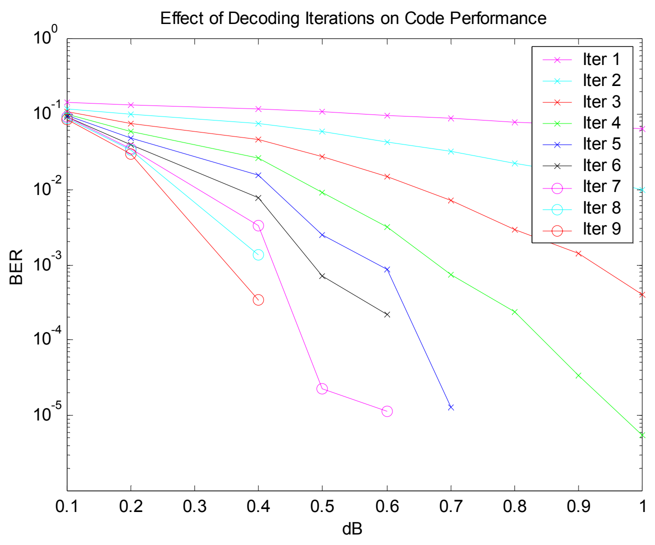


Figure 4 – Effect of Decoding Iterations

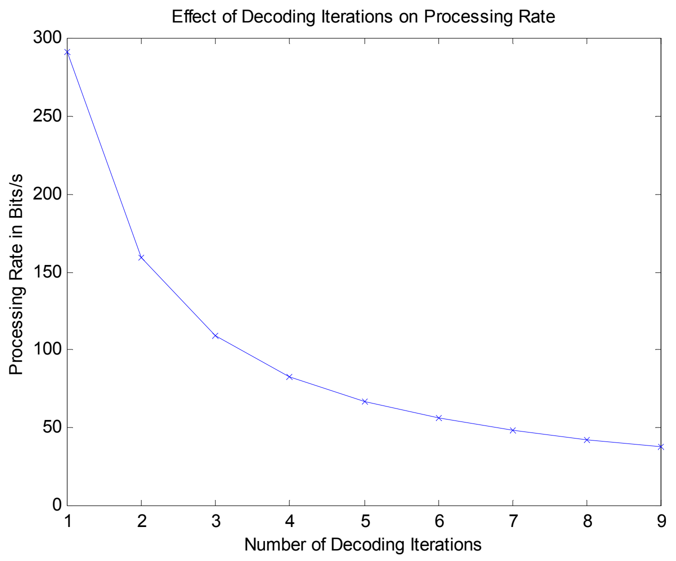


Table 4 – Turbo Code Final Simulation Results

Eb/N0 (db)	0.1	0.2	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Bits Sent	32,744	32,744	65,488	130,976	261,952	392,928	654,880	916,832	1,178,784
Errors	3009	1113	756	383	130	13	4	0	2
P_b	9.1895e-2	3.3991e-2	1.1544e-2	2.9242e-3	4.9627e-4	3.3085e-5	6.1080e-6	0.0000	1.6967e-6
Bits/s	63.82	62.37	63.75	62.01	59.52	68.62	69.05	65.46	58.15

Figure 5 – Turbo Code Final Performance Curve

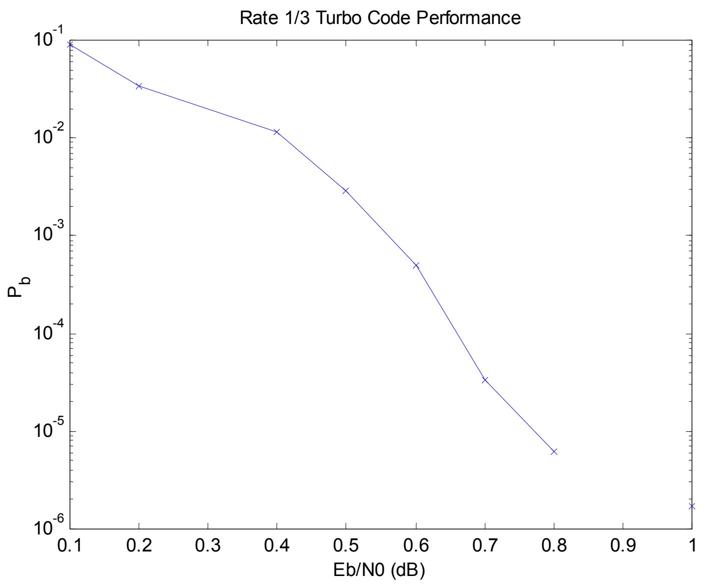


Figure 6 – Turbo Code Encoder Block Diagram

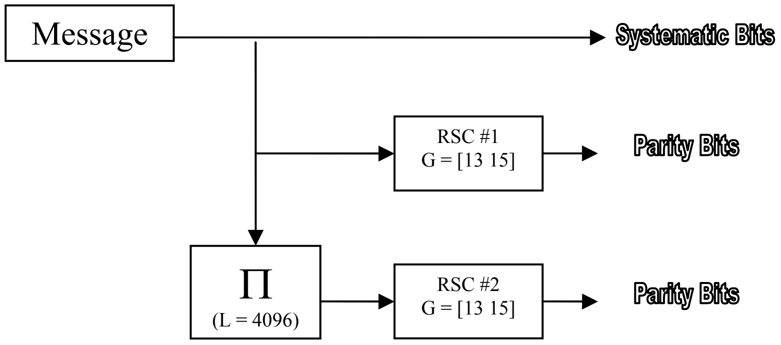
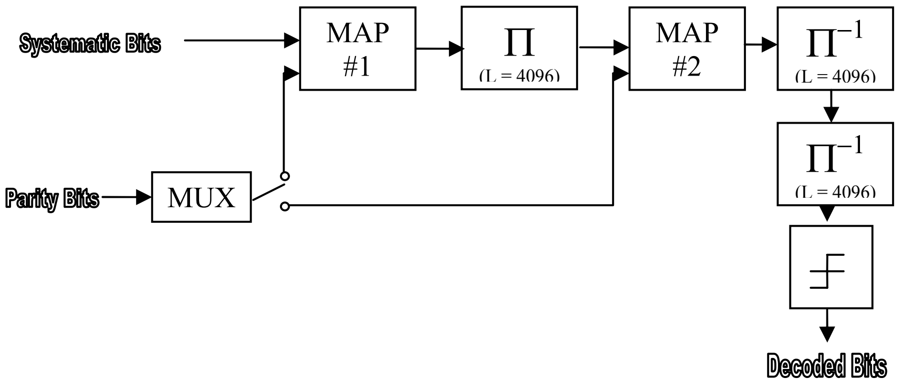


Figure 7 – Turbo Code Decoder Block Diagram

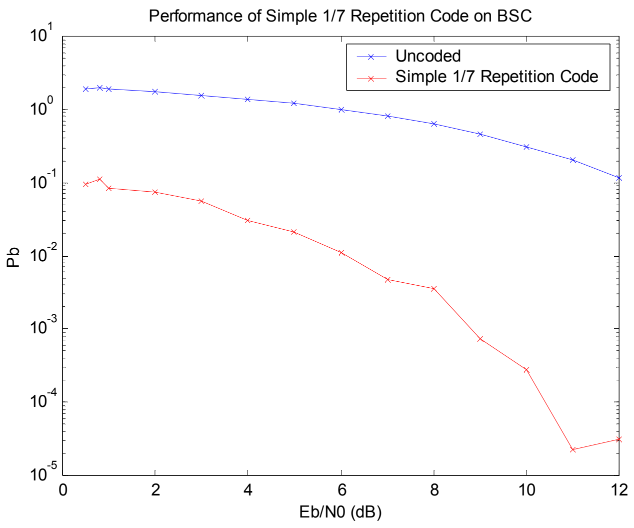


4. Rate 1/7 Repeat Accumulate and Repetition Code

A simple rate 1/7 Repetition Code was first designed for communication over a binary symmetric channel (BSC). The performance of this coding scheme compared to uncoded data can be seen in Figure 8. While it's performance is mediocre at best, it does offer much improvement over uncoded transmission and has a very high processing rate of approximately 400 bits/s.

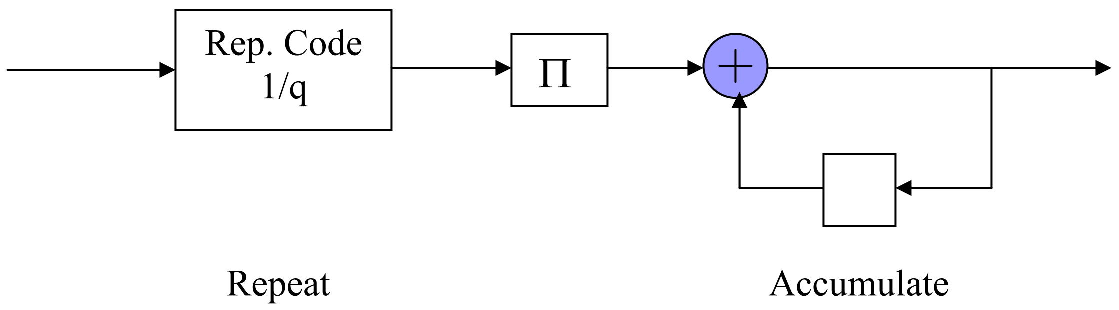
In hope of improving performance over that of the simple Repetition Code, a rate 1/7 Repeat Accumulate code was designed. RA Codes are low-complexity "turbo-like" codes introduced by Divsalar, Jin and McEliece. The outer code is a rate 1/q repetition code and the inner code is a interleaved rate-1 convolutional code with transfer

Figure 8 - Repetition Code Performance



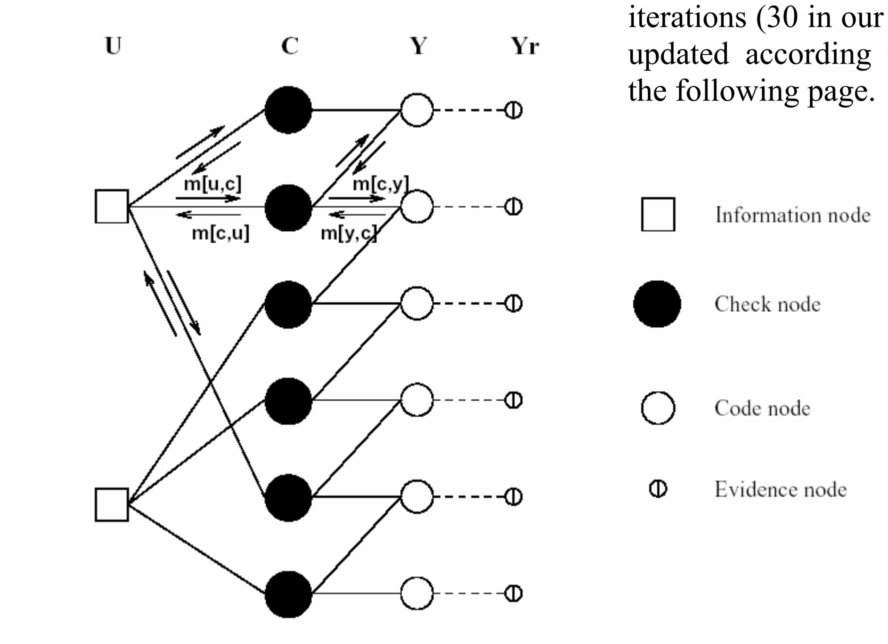
function 1/(1+D), as shown below in Figure 9.

Figure 9 – RA Encoder



Iterative decoding of RA codes can be done using a simple message passing decoding algorithm called the sum-product (belief propagation algorithm). Messages are sent between nodes on a Tanner Graph such as the one in Figure 10. Figure 10 shows a Tanner Graph of a repetition 3, length 2 RA code, with permutation $\pi = (1,2,5,3,4,6)$ [4].

Figure 10 – Tanner Graph of RA Code



Messages are passed between nodes as described in [4]. However, due to some errors found in that reference, the correct algorithm is presented here in its entirety.

Message Passing Algorithm

Initialize all messages to be zero. The messages are updated continuously K times, where K is the number of iterations (30 in our case). Messages are updated according to the algorithm on the following page.

Message Passing Algorithm (cont).

1. Update m[y,c]

$$m[y,c] = \begin{cases} B(y) & \text{if } y = y_{qn} \\ B(y) + m[c',y] & \text{otherwise,where } (c',y) \in E \text{ and } c' \neq c \end{cases}$$

2. Update m[u,c]

$$m[u,c] = \sum\nolimits_{c'} m[c',u] \text{ where } (c',u) \in E \text{ and } c' \neq c$$

3. Update at check nodes, m[c,y] and m[c,u]

$$m[c,y] = \begin{cases} m[u,c] & \text{if } c = c_1 \text{ where } (u,c) \in E \text{ and } u \in U \\ -2a\tanh(\tanh(m[u,c]/2) * \tanh(m[y',c]/2)) & \text{otherwise,where } (y',c),(u,c) \in E \text{ and } y' \neq y \in Y \end{cases}$$

$$m[c,u] = \begin{cases} m[y,c] & \text{if } c = c_1 \text{ where } (y,c) \in E \text{ and } y \in Y \\ -2a\tanh(\tanh(m[y,c]/2) * \tanh(m[y',c]/2)) & \text{otherwise,where } (y',c),(y,c) \in E \text{ and } y' \neq y \in Y \end{cases}$$

4. After K iterations

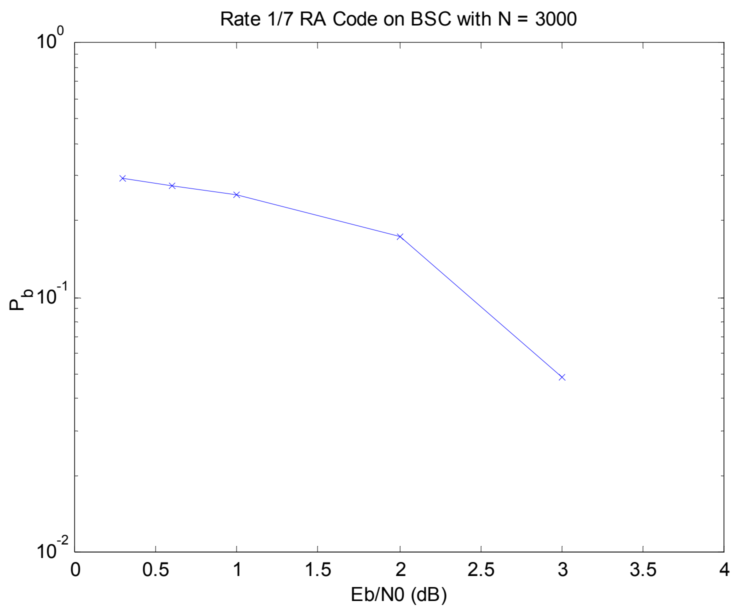
$s(u) = \sum_{c} m[u,c]$ for every $u \in U$ is calculated. The bit u is decoded as shown below:

$$u = \begin{cases} 1 & \text{if } s(u) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

End of Message Passing Algorithm.

Using this decoding technique several simulations were performed. The performance curve for the rate 1/7 Repeat Accumulate code on a BSC channel can be seen in Figure 11. Simulations are still being performed to obtain more performance results.

Figure 11 – Rate 1/7 RA Code Performance



5. Software Instructions

Listed below are instructions for running the simulations presented in this paper.

5.1 - Turbo Code Simulations

1) Download awgn_turbo.zip from Blackboard.
2) Unzip to a Matlab subdirectory.
3) Edit parameters `EbN0db` and `num_bits_vector` in turbo_sys_main.m to specify the number of bits to transmit at each value of $E_b/N0$.
4) Run compile_m from the Matlab command line to compile all m-files to C code.
5) Start a DOS window and run turbo_sys_main.exe.
6) Simulations results will be automatically saved in MAT files for each simulated $E_b/N0$.

5.2 – Repetition Code Simulations

1) Download rep_code.zip from Blackboard.
2) Unzip to a Matlab subdirectory.
3) Edit parameters `SNR_vector` and `N_vector` in rep_code.m to specify the number of bits to transmit at each value of $E_b/N0$.
4) Run rep_code.m from the Matlab command line.

5.3 - Repeat Accumulate Simulations

1) Download bsc_ra.zip.
2) Unzip to a Matlab subdirectory.
3) Edit parameters `N` and `EbN0db` `in` ra_encoder_demo.m.m to specify the number of bits to transmit at each value of $E_b/N0$.
4) Run ra_encoder_demo.m from the Matlab command line.

6. Performance Summary

Several error correcting codes were designed and simulated in this project. A summary of code performance for the points of interest in this project are presented in Table 5. The processing rates listed are for a 2.2 MHz Pentium 4 processor with 1GB of RAM. Further simulations are being performed to obtain better simulations results and fill in some remaining items on the table.

Table 5 – Performance Summary

Code	Turbo	Repetition Code	RA
Channel	AWGN	BSC	BSC
Proc Rate (bits/s)	64	400	--
$E_b/N0$ @ 10^{-3}	0.56dB	9dB	--
$E_b/N0$ @ 10^{-5}	0.77dB	--	--

7. References

[1] The Turbo Code m-files developed in this project were based on original work by Yufei Wu and were obtained from http://www.ee.vt.edu/~yufei/turbo.html.

[2] Power Point Slides, "Turbo Codes", Dr. Weeks.

[3] Channel models used in this project were provided by Dr. Weeks of the University of Missouri at Rolla.

[4] The belief propagation algorithm and figures used for Repeat Accumulate decoding was based on the algorithm presented in "Analysis and Design of Turbo Like Codes" by Hui Jin. It can be obtained at http://etd.caltech.edu/etd/available/etd-08222001-151244/.