Revisiting a High-Performance Serially-Concatenated Code

Yang Yang

May 31, 2004

Abstract

This paper aims to verify the performance of a serially concatenated code proposed by Stephan ten Brink [1]. The code combines an outer rate one-half repetition code with an inner rate one convolutional code, separated by a permuter. The code is claimed to reach a bit error rate under 10^{-5} by 0.28 dB.

1 Introduction

The premise of soft iterative decoding is that given the probability that each encoded bit sent across a channel was a certain value, similar information can be calculated for the original message bits. This process, known as belief propogation, has proven to be useful in a variety of decoding schemes, and often has an intuitive foundation in graph theory. The implementation of the convolutional code incorporated in the proposed concatenated code, described later, is one example of this.

The success of such codes is often judged by their ability to reach low bit error rates (fraction of incorrect bits after decoding) across channels with capacities near the theoretical limit given by Shannon [2]. For codes with rate one-half (i.e. encoding doubles the length of message words), this limit is a signal-to-noise ratio (SNR) of 0.19 dB for the Gaussian noise channel. Stephan ten Brink claims that the bit error rate of the proposed code is less than 10^{-5} at 0.28 dB.

The proposed code is formed by concatenating a repetition code, a random permutation, and a convolutional code. This is a variation on the popular serially-concatenated codes that use two convolutional codes [3]. In the decoding process, the repetition and convolutional decoders can be seen as modules which take estimates for each bit and use the structure of the corresponding encoder to calculate each bit's extrinsic estimates. The two modules pass these probability estimates back and forth across the permuter, ideally improving on the estimates with each iteration. Since a bit error can negatively affect the decoding of a neighboring bit, the motivation of the permuter is to isolate such errors for a more robust code.

2 Encoding

2.1 Repetition Code

The repetition code is a simple code that is not particularly successful alone. It merely repeats each message bit a set number of times. The repetition code in this case has rate 1/2, so each bit appears twice in a row in the output codeword.

2.2 Permuter

The permuter, also known as an interleaver, changes the order of the repetition encoder output according to a set permutation. Randomly generated permutations have been effective here, but in any case the permutation must be agreed upon by both the encoder and the decoder [3]. Clearly, the permuter has rate 1.

2.3 Convolutional Code

The bulk of the encoding process lies in the convolutional code, a code described by both [3] and [4] and that has many variations. The convolutional encoder consists of a set of delay registers, initialized to 0, which determine the state of the machine. Since this encoder has three delays, there are $2^3 = 8$ possible machine states. At discrete points in time, an input bit is sent to the encoder, each delay both sends a previously stored bit and receives a new bit, and a set number of output bits is produced. In the given convolutional encoder, the output is the parity of the delays, the delays simply shift bits rightward, and the first delay takes on a new value that is the sum of the input bit and the current output. For each input bit, one output bit is produced, so this convolution code has rate 1.

2.4 Doping

Often the practice of omitting a small number of output bits from the convolutional encoder, called puncturing, is used to increase the code rate [4]. Stephan ten Brink proposes a variation of puncturing called "doping" that leaves the rate unchanged. Instead of output bits being omitted, they are replaced by the corresponding input bits coming directly from the permuter. The motivation for doping is explained by the EXIT chart technique [1]. This paper follows ten Brink's terminology where a "systematic bit" is an output bit of the permuter that takes the place of its corresponding convolutional "coded bit" in the codeword. The ratio of systematic bits to coded bits is small; in this case, the doping ratio is $1 : 100$. Although no particular distribution of systematic bits is specified, the decoder needs to know this distribution along with the permuter, so a uniform distribution is most easily implemented.

The overall rate of the concatenated code is the product of the rates of the components. In this case, the code rate is $(1/2) \times (1) \times (1) = 1/2$.

3 Channel

The channel across which the encoded bits are sent is the Gaussian channel, which adds Gaussian white noise. The SNR varies between 0 and 1. As is common in soft iterative decoding, the implementation of the channel outputs the more useful probability that the sent bit was a certain value instead of the actual value received.

4 Decoding

The decoding algorithm takes as input the probability estimates for each codeword bit from the Gaussian channel, and starts by applying the convolutional decoder; recall that the decoder must

be able to differentiate systematic bits from coded bits. This produces estimates for the permuted bits, which are then fed through the inverse permuter and sent to the repetition decoder. At this point, the results of the repetition decoder can either be used as final guesses for the original message bits or be sent back through to permuter. For the second and subsequent iterations, the convolutional decoder combines the estimates from both the permuted repetition decoder results and the channel. Thus, this process is soft in that probability estimates rather than hard binary bits are propogated, and iterative in that the propogation loop can continue indefinitely.

4.1 Convolutional Decoder

Note that this section is described in greater detail by Fan [4]. The convolutional decoder normally takes as input estimates for both the systematic bits and the coded bits along with preset information about the finite automaton that was used. In this case, the estimates for the systematic bits come from the permuted results of the repetition decoder (for the first iteration, this information is null; that is, the probabilities are 0.5) and the occasional systematic estimate from the channel. These two probabilities are combined as if for a rate one-half repetition decoder:

$$p = \frac{p_1 p_2}{p_1 p_2 + (1 - p_1)(1 - p_2)} \tag{1}$$

Note that when one of the estimates is 0.5, the total information is that of the other estimate. Naturally, this formula will reappear in the repetition decoder component.

For each transmitted codeword, the estimates for the coded bits are constant; they come for the most part directly from the channel. Whenever the channel provides an estimate for a systematic bit, the information about the corresponding coded bit is set to null (i.e. probability that the coded bit was 1 is 0.5).

The decoding algorithm for a convolutional code involves the belief propogation of messages containing probabilities for the state of the finite automaton, sent forward and backward in time. The principle behind this is that at any point in time, the message traveling forward contains the combined information of all systematic and coded estimates before it, and the backward message contains information from later estimates. Thus, a certain input bit's extrinsic probability, taking into account all other estimates except for its own, can be found by combining the forward and backward messages with the corresponding output bit estimate.

A state change of the finite automaton can be described by a bipartite graph called a trellis consisting of the initial and final states as vertices and all possible transitions from the former to the latter as edges. Propogating messages forward in time involves calculating the probability of each final state, given the probability of each initial state and the estimates for the input and output bits corresponding to this transition. Specifically, if $s_0 \dots s_7$ are the states, S is the set of ordered pairs (i, j) describing valid transitions $s_i \to s_j$, and $w_{i \to j}$ and $v_{i \to j}$ are respectively the input and output values of such a transition, then at time t

$$p_t(s_j) = c_t \sum_{(i,j) \in S} p_{t-1}(s_j)\ p_{t-1}(w_{i \to j})\ p_{t-1}(v_{i \to j})$$

where c_t is a normalizing constant such that $\sum_{j=0}^{7} p_t(s_j) = 1$. The initial condition for the forward direction is $p_0(s_0) = 1$. Thus, the message-passing process is fast because each of the above values, other than c_t, is immediately known. Passing messages backward given $p_{t+1}(s)$, $p_{t+1}(w)$, and

$p_{t+1}(v)$ is nearly identical, with the exception that the initial conditions contain no information, i.e. $p_n(s) = 1/8$.

Finally, after messages have been passed both ways, the extrinsic probability of the input bit is given by

$$p_t(w = w_0) = c'_t \sum_{\substack{(i,j) \in S \\ w_{i\to j} = w_0}} p_t(v_{i\to j}) \overrightarrow{p}_t(s_i) \overleftarrow{p}_{t+1}(s_j)$$

where c'_t normalizes $p_t(w = 0)$ and $p_t(w = 1)$, and $\overrightarrow{p}$ and $\overleftarrow{p}$ are the forward and backward messages, respectively. In this case, $p_t(w = 1)$ is the information sent to the inverse permuter.

4.2 Repetition Decoder

The repetition decoder receives from the inverse permuter a pair of estimates for each message bit. When passing extrinsic estimates back through the permuter, the information for each bit comes solely from the other estimate for the same original message bit. Thus, the repetition decoder merely swaps the two probabilities for a message bit. However, at the final stage when posterior probabilities are wanted, equation (1) combines the two probabilities.

5 Results

5.1 Revising the Claim

The main results relevent to ten Brink's claim are shown in Figure 1. When the experiment was simulated for codelength 2×10^4, 100 iterations, and a 1:100 doping ratio, the results were poorly matched with ten Brink's original results. The bit error rate is markedly higher for channels with SNR between 0.4 and 0.8 dB. The original results show a steady decrease in BER on this interval, but the new results show what ten Brink calls a "turbo cliff," or a rapid drop in BER, at 0.8 dB SNR.

The results for the simulations with codelength 2×10^5, 100 iterations, and with codelength 1×10^6, 300 iterations were closer to ten Brink's. Both of these performed significantly better than the classic parallel concatenated codes, or Turbo codes, given by ten Brink for comparison. However, the turbo cliffs for these parameters occured at high SNR's than claimed. In particular, the 0.28 dB turbo cliff exhibited by the latter code actually occured at 0.32 dB.

5.2 Varied Doping Ratios

The original choice to use a doping ratio of 1:100 [1] seems arbitrary. Thus, the simulation was performed with doping ratios 1:n for $n = 5$, 10, 20, 50, 100, and 200, holding all other parameters constant (codelength 2×10^4 and 100 iterations). When n is low, there are too many systematic bits and much of the redundancy of the finite automaton is forfeit. When n is high, ten Brink's EXIT chart argument is convincing; with too few systematic estimates, the convolutional decoder's transfer characteristic intercepts the y-axis near 0, so that starting the iterative process is difficult [1]. Thus, there is an optimal set or range of doping ratios. For the given parameters, Figure 2 shows that a ratio of 1:50 results in the best performance. Simulating with finer values of n is

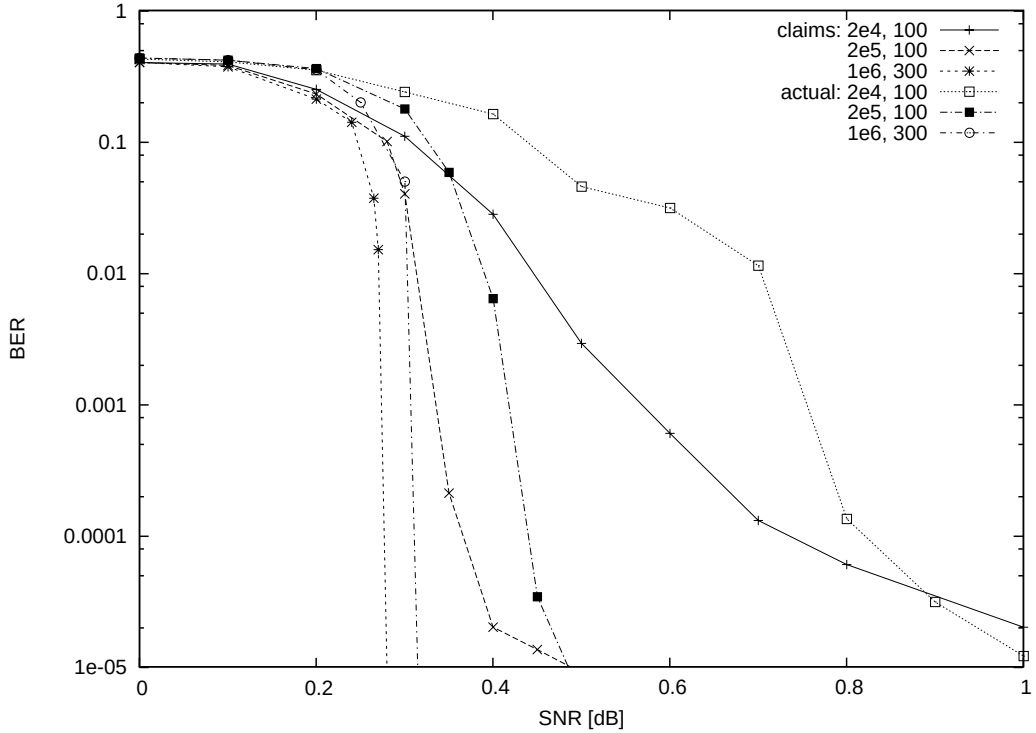

Figure 1: Comparison of claimed and actual bit error rates for various codelengths and iterations

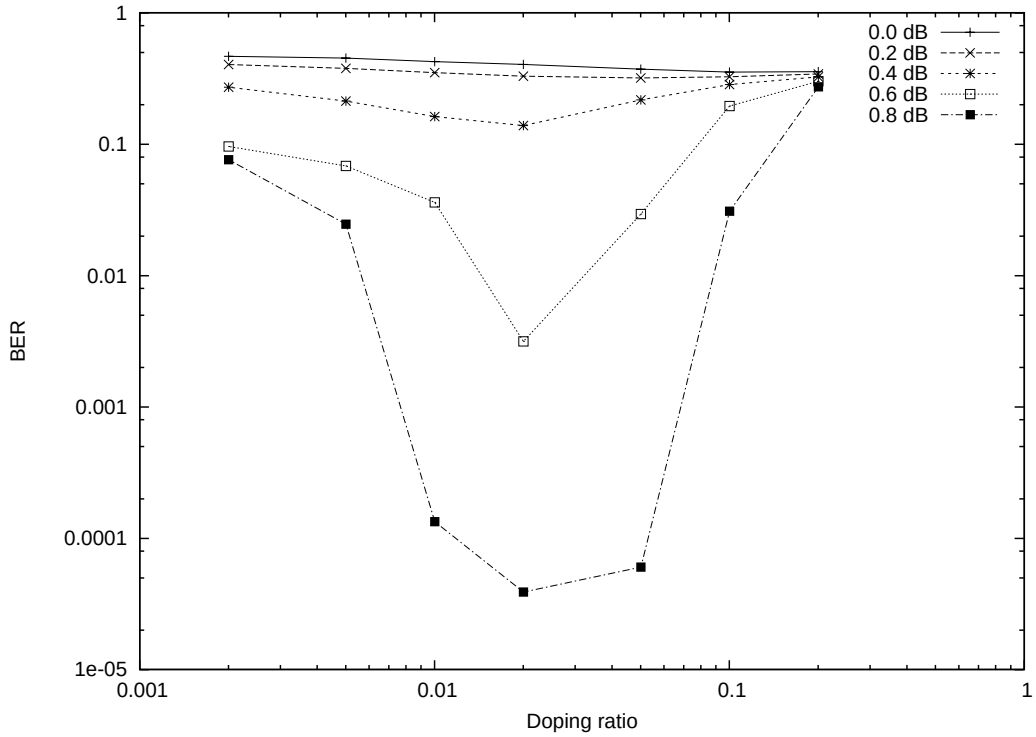

Figure 2: Bit error rates for codelength 2e4, 100 iterations, at various doping ratios

likely to improve on this value. In addition, the advantage of a well-chosen doping ratio is more significant at higher SNR's.

5.3 Varied Iteration Lengths

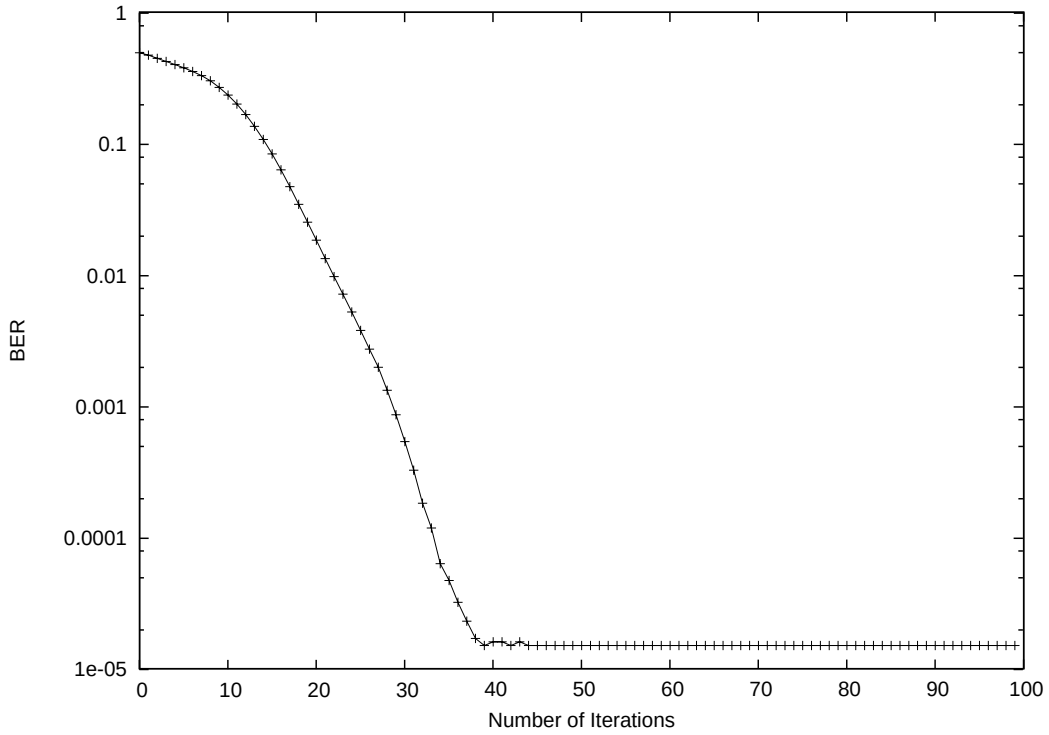

Figure 3: Progress of bit error rate for decoding 2e4 bits at 1 dB

Lastly, the number of iterations used in the decoding process before outputting a final guess can be varied. This is equivalent to tracking the progress of decoder at various points in time. Figure 3 shows the result of this investigation for codelength 2×10^4 at SNR of 1 dB. Although the code reaches a successful BER of 1.5×10^{-5} by 100 iterations, it could have stopped after 40 iterations for the same BER. Thus, no improvement on the BER is possible after a certain number of iterations, and subsequent calculations are unproductive; knowledge of this point is useful especially in time-critical situations.

6 Conclusion

6.1 Summary

Although the results of this experiment did not accurately verify Stephan ten Brink's proposal, it is evident that this particular serial concatenation of repetition and convolutional codes performs well in comparison to Turbo codes (Figure 1), exhibiting a turbo cliff at 0.32 dB instead of the claimed 0.28 dB. This small discrepancy might be due to differences in implementing systematic doping; it is possible that another distribution of systematic bits among the codeword will improve

the code's performance. Using a different doping ratio (closer to 1:50) can also achieve lower BER (Figure 2). Finally, the efficiency of the decoding process can be increased if it is known that the BER will not improve with additional iterations past a certain point (Figure 3).

6.2 Source Code

The experiment was written in C++ and compiled with GCC 3.3.1 for GNU/Linux. The relevent source code is available at `http://mit.edu/y_yang/Public/18.413` along with large versions of the graphs.

6.3 Acknowledgment

This work was carried out in the Error-Correcting Codes Laboratory course at MIT, under the guidance of Professor D. A. Spielman.

References

[1] S. ten Brink, "A rate one-half code for approaching the Shannon limit by 0.1dB," *IEE Electronics Letters*, vol. 36, no. 15, pp. 1293–1294, July 2000.

[2] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, July, Oct. 1948.

[3] D. A. Spielman, "Error-Correcting Codes Laboratory" Lecture Series. MIT, Cambridge. 2 Feb. 2004 – 15 Apr. 2004.

[4] J. L. Fan, *Constrained Coding and Soft Iterative Decoding*. Kluwer, Boston, 2001.