Product Accumulate Codes: A Class of Capacity-Approaching, Low Complexity Codes

Jing Li, Krishna R. Narayanan* and Costas N. Georgiades
Department of Electrical Engineering
Texas A&M University, College Station, TX 77843
Email: \{jingli, krishna, georghia\}@ee.tamu.edu

Abstract

We propose a novel class of provably good codes which are a serial concatenation of a single-parity check turbo product code (TPC/SPC), an interleaver and a rate-1 recursive convolutional code. The proposed codes, termed product accumulate (PA) codes, are linear time encodable and linear time decodable. We show that the product code by itself is not a “good” code; however, a product accumulate code is a “good” code both in the maximum-likelihood (ML) sense and under iterative decoding. Two message-passing decoding algorithms are proposed and it is shown that a particular update schedule for these message-passing algorithms is equivalent to conventional turbo decoding of the serial concatenated code, but with significantly lower complexity. Tight upper bounds on the ML performance using Divsalar’s simple bound and thresholds under density evolution show that these codes are capable of performance within a few tenths of a dB away from the Shannon limit. Simulation results confirm these claims and show that these codes provide performance similar to turbo codes but with significantly less decoding complexity and without an error floor. Hence, we propose PA codes as a class of prospective codes with good performance, low decoding complexity and regular structure, uniformly for all rates greater than or equal to 1/2.

*Corresponding author.
1 Introduction and Outline of the Paper

We propose a novel class of provably "good" codes \(^1\) which are referred to as product accumulate (PA) codes. The proposed codes are shown to possess many desirable properties, including close-to-capacity performance, low decoding complexity, regular structure and easy rate adaptivity uniformly for all rates higher than and including 1/2.

The work was initiated by the search for good, high-rate codes which permit soft-decision and soft-output decoding. Several applications require the use of (soft-decodable) high rate codes. Some widely used high-rate codes are Reed-Solomon (RS) codes, punctured convolutional codes, turbo codes and low density parity check (LDPC) codes. Until very recently, soft decision decoding of RS codes has been a major computational problem. Recent developments are yet to be benchmarked to know the exact performance of soft decision decoding of RS codes. In order to obtain good performance from high-rate punctured convolutional codes and turbo codes, convolutional codes usually must be of long constraint length, making the decoding complexity rather high. Low density parity check codes, on the other hand, provide good performance at possibly lower complexity; however, the encoding complexity is \(O(N^2)\) (\(N\) is the codeword length) and, moreover, may require explicit storage of a generator matrix. Further, good high-rate LDPC codes are difficult to construct for short block lengths.

In an effort to construct good, simple, soft-decodable, high-rate codes, we investigated single-parity check turbo product codes (TPC/SPC) which can be soft decoded by a message-passing algorithm (also referred to as the sum-product algorithm). TPC/SPC codes have recently been investigated for potential application on high-density magnetic recording channels and have demonstrated encouraging performance via a turbo approach [3]. Since the TPC/SPC itself is not a "good" code, we consider the concatenation of a rate-1 inner code (differential encoder or accumulator) with the TPC/SPC code through an interleaver. Through analysis and simulations we find this class of codes to be remarkably good in bit error rate (BER) performance at high code rates \((R \geq 0.7)\) when used with an iterative message-passing decoding algorithm. We show that the performance of these codes can be further improved by replacing the block interleaver in the conventional TPC/SPC outer code with a random interleaver. We will refer to such codes as PA-I codes. Clearly, when the outer code is a conventional TPC/SPC code (with a block interleaver), it is a special case of the proposed PA-I codes and we will refer to those as PA-II codes.

To facilitate understanding the structure and potential of the proposed codes, we compute tight upper bounds on their performance using the bounding technique developed by Divsalar [4]. We also study the graph structure of these codes. Thresholds are computed using density evolution (DE) [5] and shown to be within a few tenths of a dB from the Shannon limit for all rates \(R \geq 1/2\). By studying the graph structure, a message-passing (sum-product) decoding algorithm and its low-complexity approximation, a

\(^1\)A "good" code is defined as a code for which there exists a threshold above which an arbitrarily low error rate can be achieved as block size goes to infinity [1].
min-sum algorithm, can be developed to iteratively decode the outer and the inner codes. We show that a particular update schedule for this algorithm when applied to the graph of the inner code results in optimal decoding of the inner code $1/(1 + D)$. That is, the sum-product algorithm applied to the decoding of $1/(1 + D)$ is equivalent to the Bahl, Jelinek, Cocke and Raviv (BCJR) algorithm [22] (optimal in the a posteriori probability (APP) sense) and the min-sum algorithm is equivalent to the Max-log-MAP algorithm. However, the message-passing algorithm can be implemented with significantly lower complexity than the BCJR equivalents. Simulation results with long block lengths confirm the thresholds and simulations with moderate block lengths show that performance close to turbo codes can be achieved with significantly lower complexity.

As such, we propose the class of product accumulate codes as a prospective class which not only enjoys good-performance, low-complexity and soft-decodability, but also maintains a simple and regular structure uniformly for all block sizes and for all rates above $1/2$. This regular structure as well as the ease in construction is a particularly appealing property in practical implementation and in applications that require rate adaptivity.

A brief background on TPC/SPC codes is presented in Section 2, followed by description of PA codes in Section 3. The decoding of PA codes is discussed in Section 4; in particular, a graph-based sum-product algorithm is described and shown to be optimal for inner rate-1 convolutional codes, yet with very low complexity. Section 5 analyzes in detail some properties of PA codes, including upper bounds on the performance under ML decoding and thresholds of the codes under iterative decoding. Section 6 discusses an algebraic construction which is useful in practical implementation. Section 7 presents simulation results. Section 8 compares the proposed codes with other good codes proposed recently. Conclusions and future work are discussed in Section 9.

2 Background on TPC/SPC Codes

Since the motivation for the proposed product accumulate codes stems from TPC/SPC codes, it is desirable to first discuss TPC/SPC codes.

2.1 TPC/SPC Code Structure and Properties

A turbo product code (TPC) [6] is composed of a multi-dimensional array of codewords from linear block codes, such as parity check codes, Hamming codes and BCH codes. Particularly of interest is the simplest type of TPC codes, namely, single-parity check turbo product codes (TPC/SPC), due to their simplicity and high rate. An $s$-dimensional ($s$-D) turbo product code $C$ formed from component codes $C_1 \sim (n_1, k_1, d_1), C_2 \sim (n_2, k_2, d_2), \ldots, C_s \sim (n_s, k_s, d_s)$ has parameters $(n_1 n_2 \cdots n_s, k_1 k_2 \cdots k_s, d_1 d_2 \cdots d_s)$, where $n$, $k$ and $d$ are the codeword length, user data block length and the minimum distance of the code, respectively, and its generator matrix is the Kronecker product of the generator matrices of the component codes: $G = G_1 \otimes G_2 \otimes \cdots \otimes G_s$. Since high-rates are of interest, we restrict our attention to 2-D TPC/SPC
codes in this work. The encoding of a 2-D TPC code is straight-forward: The data bits are placed in a $K_1 \times K_2$ block and each row is first encoded using $\mathcal{C}_1$, (parity $P_1$ is added, see Fig. 1). Then, each column is encoded using $\mathcal{C}_2$, (parity $P_2$ and $P_p$ are added). The order of encoding is unimportant, and all rows will be valid codewords of $\mathcal{C}_1$ (referred to as the row code) and all columns will be valid codewords of $\mathcal{C}_2$ (referred to as the column code). The encoding algorithm can be easily extended to the multidimensional case. In the general case, a TPC code may or may not have “parity-on-parity” bits ($P_p$ in Fig. 1). A TPC code without parity-on-parity is essentially a parallel concatenation with a block interleaver within the encoder, and a TPC code with parity-on-parity is a serial concatenation with a block interleaver within the encoder.

The decoding of TPC codes takes an iterative approach based on the soft-in soft-out (SISO) decoders for each of its component codes. Decoding of TPC codes is generally via the Chase algorithm [7], a controlled-search procedure. However, with single-parity check component codes, decoding can be handled in a simpler and more efficient manner. The observation that a TPC/SPC code can be effectively viewed as a type of structured LDPC code [3] [8] where each row in each dimension satisfies a check, leads to a convenient adoption of the message-passing algorithm (or the sum-product algorithm) from LDPC codes. Since each bit is expressed as the modulo-2 sum of the rest of the bits in the check, this message-passing decoding algorithm is in fact an extension of replication decoding [9]. The exact decoding algorithm can be found in Appendix I. The simple and regular structure of a TPC/SPC code makes it possible to analyze the code properties. In particular, the weight spectrum of a 2-D TPC/SPC code with parameter $C \sim (n_1n_2, (n_1-1)(n_2-1))$ can be calculated by the following equation [9]:

$$A(h) = 2^{-n_1} \sum_{a=0}^{n_1} \binom{n_1}{a} \left[ \sum_{m=0, m \text{ even}}^{n_1} P_m(a; n_1) h^m \right]^{n_2},$$

(1)

where

$$P_m(a; n) = \sum_{k=0}^{m} (-1)^k \binom{a}{k} \binom{n - a}{m - k}.$$  

(2)

As expected, $A(h)$ is symmetric in $n_1$ and $n_2$. It has been shown that the weight distribution of TPC/SPC codes asymptotically approaches that of a random code if the dimension of the code and the lengths of all component codes go to infinity [9]. However, increasing the dimension decreases the code rate and is therefore not of interest in the design of high-rate codes.

2.2 A TPC/SPC Code by Itself Is Not a “Good” Code

A “good” code is defined as a code for which there exists a threshold above which an arbitrarily low error rate can be achieved as block size goes to infinity [1]. Examples of good codes include low density parity check codes, turbo codes and other serial/parallel concatenated codes. Unfortunately a TPC/SPC code by itself is not a good code. To see this, note that an $s$-dimensional TPC/SPC code always has minimum distance $2^s$ irrespective of the block size. Assuming maximum-likelihood decoding, the lower bound on the
word error rate (WER) is:
\[
P_w(e) \geq Q \left( \frac{2^{r+1}RE_b}{N_0} \right),
\]
where \( R \) is the code rate. Obviously, the lower bound is not a function of block size. In other words, unless the dimensionality of a TPC/SPC code, \( s \), goes to infinity, its WER performance is always bounded away from zero independent of the block size. Hence, a TPC/SPC code alone is not good in the ML sense.

In an effort to improve the performance of TPC/SPC codes, some attempts have been made to increase its minimum distance by carefully adding more parity checks by increasing the dimensionality \([10] [11]\). However, adding dimensionality obviously reduces code rate. Further, for any TPC/SPC code of a given dimensionality, the minimum distance is fixed and does not improve with block size. In other words, except for the asymptotic case where \( s \to \infty \), multi-dimensional TPC/SPC codes are still not “good” codes. Moreover, when \( s \to \infty, R \to 0 \), and, hence, this case is not of interest here.

In this paper we take a different approach in improving the performance of TPC/SPC codes, which is to group several blocks of TPC/SPC codewords together, interleave them and further encode them with a rate-1 recursive convolutional code (or an accumulator). The resulting serial concatenation brings a significant improvement to TPC/SPC codes in their fundamental structural properties, for, as will be explained in later sections, the resulting serial concatenated code now becomes a “good” code with linear encoding and decoding complexity. Furthermore, we will discuss a modification to the interleaving scheme within the TPC/SPC code which results in a better code structure.

3 Structure of the Proposed Product Accumulate Codes

3.1 Proposed Code Structure

Fig. 2 shows the overall structure of a high-rate product accumulate code. It comprises a 2-D TPC/SPC outer code of rate \((t/(t+1))^2\) \((t/(t+1)\) in each dimension) and a random interleaver, a rate-1 recursive convolutional inner code of the form \(1/(1+D)\) (also known as the accumulator). For a TPC/SPC code of length \((t+1)^2\), \( P \) such codewords are taken and interleaved using a random interleaver of size \( N = P(t+1)^2 \) (denoted by \( \pi \) in Fig. 2), and further encoded using a rate-1 recursive convolutional encoder to form a codeword of length \( N \). Thus, the resulting code is a \((P(t+1)^2,P \ell^2)\) code with rate \((t/(t+1))^2\). The random interleaver works to break up the correlation between the messages (extrinsic information) and, in conjunction with the recursive inner code, to map low-weight error events to high-weight error events, which results in a good distance spectrum.

Since \( P \) codewords of the TPC/SPC code are interleaved together before passing through the accumulator, the input block length (and, hence, latency) is \( P \ell^2 \) information bits. In a conventional TPC/SPC code the data is placed in a \( t \times t \) block before encoding and, hence, a conventional TPC/SPC code can be considered as a concatenated code with a block interleaver within the TPC/SPC encoder. Therefore, the structure
of the outer code (Fig. 2) is equivalent to \(Pt^2\) information bits being interleaved using \(P\) separate block interleavers, further interleaved using a random interleaver (\(\pi\) in Fig. 2) and then encoded by a rate-1 inner code. Instead of this structure, it is possible to replace the interleaver within the TPC/SPC with one random interleaver of size \(Pt^2\) bits (denoted by \(\pi_1\) in Fig. 3). When a random interleaver is used, it is no longer trivial to add parity-on-parity bits and, hence, we do not use parity-on-parity bits. The structure of the resulting code is shown in Fig. 3. To be precise, the resulting code (i.e., our proposed product accumulate code) is a serial concatenation of an outer code, an interleaver and a rate-1 inner code. The outer code itself is a parallel concatenation of two \((t + 1, t)\) single parity check codes with a pseudo-random interleaver between the two parallel branches. (For notational convenience, we still refer to the outer code as a TPC/SPC code, and those that use block interleavers within the TPC/SPC code as conventional TPC/SPC codes.) For an input of \(K = Pt^2\) bits, the outer codewords (without parity-on-parity) are of length \(P(t^2 + 2t)\) and, therefore, the rate of the code is \(t/(t + 2)\). Note that with the random interleaver, an input length \(K = Pt\) bits can be encoded in \(P(t + 2)\) bits and, hence, the input block length should be a multiple of \(t\) and not necessarily \(t^2\). We refer to this structure as PA-I codes and is shown in Fig. 3. The code using the conventional TPC/SPC code as the outer code (introduced in the previous paragraph) can be thought of as a special case of this PA-I structure where the interleaver in the parallel concatenation is constrained to be \(P\) separate block interleavers and has parity on the parity bits (with a resulting rate of \((t/(t + 1))^2\) rather than \(t/(t + 2)\)). We refer to this special case as PA-II codes and is shown in Fig. 2.

The idea of concatenating an outer code and an interleaver with a rate-1 recursive inner code, particularly of the form of \(1/(1 + D)\), to achieve coding gains (interleaving gain) without reducing the overall code rate is widely recognized [12] [13] [14]. For low rate codes (rate-1/2 or less), convolutional codes and even very simple repetition codes [15] are good outer code candidates to provide satisfactory performance. However, the construction of very high rate codes based on this concept poses a problem. The key problem here is that, from Benedetto et al's results [16] [17], the outer code must have a minimum distance of at least 3 in order to obtain an interleaving gain. To obtain good high-rate convolutional codes through puncturing, and in particular to maintain a \(d_{\min}\) of 3 after puncturing, the original convolutional codes must have fairly long constraint length, which makes decoding computationally inefficient. On the other hand, 2-dimensional single-parity check turbo product codes (TPC/SPC) possess many nice properties for a concatenated high-rate coding structure, such as high rate, simplicity, and the availability of an efficient soft decoding algorithm. PA-II codes have outer codes with \(d_{\min} = 4\) for any code rate and, hence, an interleaving gain is achieved. In Section 5.2, we show that although the outer code of PA-I codes has \(d_{\min} = 2\), an interleaving gain still exists for PA-I codes.

In the sections below, we will perform a comprehensive analysis and evaluation of the proposed product accumulate codes. The focus is on PA-I codes since they are the more general case and since they generally achieve better performance than PA-II codes. However, the special case of PA-II codes is separately discussed. This is because PA-II codes are simpler to analyze and implement than PA-I codes (a block interleaver is
usually easier to implement in hardware than a random interleaver). Further, as we will show later, at rates $R > 0.7$, PA-II codes are sufficient to achieve near-capacity performance, alleviating the need for another random interleaver in the outer TPC/SPC.

4 Iterative Decoding of PA Codes

The turbo principle is used to iteratively decode the serially concatenated system, where soft extrinsic information in log-likelihood ratio (log-LLR) form is exchanged between the inner and outer decoders. The extrinsic information from one sub-decoder is used as a priori information by the other sub-decoder. The decoding of the outer TPC/SPC code is done using a message-passing algorithm similar to that of LDPC codes, as described previously. The inner rate-1 convolutional code is typically decoded using a 2-state BCJR algorithm, which generates the extrinsic information for bit $x_i$ in the $k_{th}$ turbo iteration, denoted $L_e^{(k)}(x_i)$. The outer decoder uses $L_e^{(k)}(x_i)$ as a priori information and produces extrinsic information $L_e^{(k)}(x_i)$. However, a more computationally efficient approach is to use message-passing decoding directly on the graph of the product accumulate code including the inner code, whose sub-graph has no cycles.

It has been recognized that the message-passing algorithm is an instance of Pearl’s belief propagation which converges to the optimal solution if the operating graph is cycle-free. The basic idea of probability inference decoding is implied in Tanner’s pioneering work in 1981 [18], and later studied by Wiberg [19], Frey [20], Forney et al. [21], as it gained success in decoding LDPC codes. However, little has been reported for practical application on convolutional codes. This is because the code graph of a convolutional code is in general complex and involves many cycles which either make the message flow hard to track or make the algorithm ineffective (due to the significant amount of correlation in the messages caused by the cycles). Nevertheless, for the specific case of the $1/(1 + D)$ code, a cycle-free Tanner graph presenting the relation of $y_i = x_i \oplus y_{i-1}$ ($\oplus$ denotes modulo-2 addition) can be constructed, using which the message flow can be conveniently traced. We describe below in detail how the message-passing algorithm can be efficiently applied to the $1/(1 + D)$ code. Recently, message-passing on the graph structure of a $1/(1 + D)$ inner code has been used with irregular repeat accumulate (IRA) codes by Jin, Khandekar and McEliece [15] and by Divsalar et al. [4] to analyze 2-state codes. Here, we derive a message-passing algorithm from the BCJR algorithm and show the relationship between the two. Specifically, we show that a serial update in the graph (rather than the parallel update as used in [15] and [4]) is equivalent to the BCJR algorithm, but has an order of magnitude lower complexity. Similarly, we show that the low-complexity approximation, the min-sum update on the graph, is equivalent to the Max-log-MAP algorithm.

4.1 The Message-Passing Algorithm

As shown in Fig. 4(a), the combination of the outer code, the interleaver and the inner code can be represented using one graph which contains bit nodes (representing the actual bits) and check-nodes (representing a constraint such that connecting bit nodes should sum up (modulo-2) to zero). Fig. 4(b) illustrates how
messages evolve within $1/(1 + D)$ code. The outgoing message along an edge should contain information from all other sources except the incoming message from this edge. For example, the LLR of bit $y_i$ consists of $L_{ch}(y_i)$ from the channel, $L_{e_f}(y_i)$ from check $i$ and $L_{e_b}(y_i)$ from check $i + 1$ (Fig. 4(b)). When $y_i$ participates in check $i$, to calculate the extrinsic $L_e(x_i)$ for $x_i$, the information content $L_{e_f}(y_i)$, which was obtained from check $i$ previously, should not be passed back to check $i$. Similar rules hold for all other bits. Hence the extrinsic messages sent out at bit $x_i$ at the $k_{th}$ turbo iteration, $L_e^{(k)}(x_i)$, should be computed as:

$$L_e^{(k)}(x_i) = \left( L_{ch}(y_{i-1}) + L_{e_f}^{(k)}(y_{i-1}) \right) \oplus \left( L_{ch}(y_i) + L_{e_b}^{(k)}(y_i) \right),$$

where $L_{ch}(y_i) = \log \frac{p(r_i|y_i=1)}{p(r_i|y_i=0)}$ denotes the message (LLR) obtained from the channel ($r_i$ is the received signal corresponding to the coded bit $y_i$), $L_{e_f}(y_i)$ and $L_{e_b}(y_i)$ denote the (extrinsic) messages passed “forward” and “backward” to bit $y_i$ from the sequence of bits/checks before and after the $i_{th}$ position, respectively. Superscript $(k)$ denotes the $k_{th}$ turbo iteration between the inner and outer decoders (as opposed to the local iterations in the decoding of the outer TPC/SPC code) and subscript $i$ denotes the $i_{th}$ bit/check. Operation $\oplus$ refers to a “check” operation. It can be shown that if $\alpha$ and $\beta$ are the LLRs passed along an incoming edge into a $\oplus$ operation, then the outgoing extrinsic information is given by:

$$\gamma = \alpha \oplus \beta \iff \gamma = \log \frac{1 + e^{\alpha + \beta}}{e^\alpha + e^\beta},$$

$$\iff \gamma = 2 \tanh^{-1} \left( \frac{\tanh \frac{\alpha}{2} \cdot \tanh \frac{\beta}{2}}{2} \right).$$

Messages $L_{e_f}(y_i)$ and $L_{e_b}(y_i)$ correspond to a forward and backward pass, respectively, along the code graph. As illustrated in Fig. 4(c,d), at the $k_{th}$ turbo iteration (between the inner and outer decoder), they can be calculated as:

$$L_{e_f}^{(k)}(y_{i}) = L_{o}^{(k-1)}(x_i) \oplus \left( L_{ch}(y_{i-1}) + L_{e_f}^{(k)}(y_{i-1}) \right), \quad 2 \leq i \leq N,$$

$$L_{e_b}^{(k)}(y_{i}) = L_{o}^{(k-1)}(x_{i+1}) \oplus \left( L_{ch}(y_{i+1}) + L_{e_b}^{(k)}(y_{i+1}) \right), \quad 1 \leq i \leq N - 1,$$

where $L_{o}^{(k-1)}(x_i)$ is the message received from the outer TPC/SPC code in the $(k - 1)_{th}$ turbo iteration (between inner and outer codes). Clearly, $L_{o}^{(0)}(x_i) = 0, \forall i$, since in the first turbo iteration the inner code gets no information from the outer code. The boundary conditions are:

$$L_{e_f}^{(k)}(y_{1}) = L_{o}^{(k-1)}(x_{1}) \oplus \infty = L_{o}^{(k-1)}(x_{1}), \quad k \geq 1,$$

$$L_{e_b}^{(k)}(y_{N}) = 0, \quad k \geq 1.$$

From the above computation, it can be seen that the outbound message at the present time instance $i$, $L_{e}^{(k)}(x_i)$, has utilized all dependence among the past and future (through $L_{e_f}^{(k)}(x_{i-1})$ and $L_{e_b}^{(k)}(x_{i})$) without any looping back of the same information.

**Lemma 1:** The aforementioned message-passing (sum-product) decoder is identical to the BCJR algorithm for the $1/(1 + D)$ inner code.
Proof: Interested readers are referred to [22][23][24] for a basic introduction of the BCJR algorithm. We use $x_t$, $y_t$, $s_t$, $r_t$ to represent data bit, coded bit, (binary) modulated bit (signals to be transmitted over the channel) and received bit (noise corrupted), respectively. Their relations are illustrated as follows:

$$y_t = y_{t-1} \oplus x_t \quad \text{BPSK + noise}$$
$$x_t \in \{0, 1\} \implies y_t \in \{0, 1\} \implies s_t \in \{\pm 1\} \implies r_t$$

The following definitions and notations are needed in the discussion:

- $\Pr(S_t = m)$ — probability the decoder is in state $m$ at time instance $t$, $(m \in \{0, 1\}$ in a 2-state case).
- $r_t^j \triangleq (r_t, r_{t+1}, \ldots, r_j)$ — received sequence.
- $\alpha_t(m) \triangleq \Pr(S_t = m, r_t^j)$ — forward path metric.
- $\beta_t(m) \triangleq \Pr(r_t^N | S_t = m)$ — backward path metric.
- $\gamma_t(m', m) \triangleq \Pr(S_t = m, r_t | S_{t-1} = m')$ — branch metric.
- $\Lambda_t \triangleq \log \frac{\Pr(x_t = 0 | r_t^N)}{\Pr(x_t = 1 | r_t^N)}$ — output LLR of bit $x_t$.

The branch metric of the $1/(1 + D)$ code is given by (see trellis in Fig. 5):

$$\gamma_t(0,0) = \Pr(x_t = 0) \Pr(r_t | y_t = 0),$$
$$\gamma_t(0,1) = \Pr(x_t = 1) \Pr(r_t | y_t = 1),$$
$$\gamma_t(1,0) = \Pr(x_t = 1) \Pr(r_t | y_t = 0),$$
$$\gamma_t(1,1) = \Pr(x_t = 0) \Pr(r_t | y_t = 1).$$

Consider the ratio $\alpha_t(0) / \alpha_t(1)$ in the forward recursion:

$$\frac{\alpha_t(0)}{\alpha_t(1)} = \frac{\alpha_{t-1}(0) \gamma_t(0,0) + \alpha_{t-1}(1) \gamma_t(1,0)}{\alpha_{t-1}(0) \gamma_t(0,1) + \alpha_{t-1}(1) \gamma_t(1,1)}. \quad (16)$$

Substituting (12)-(15) in (16), and dividing both the numerator and denominator by $\alpha_{t-1}(1) \Pr(x_t = 1) \Pr(r_t | y_t = 1)$, we get:

$$\frac{\alpha_t(0)}{\alpha_t(1)} = \left( \frac{\alpha_{t-1}(0) \Pr(x_t = 0) + 1}{\alpha_{t-1}(0) \Pr(x_t = 1)} \right) \cdot \frac{\Pr(r_t | y_t = 0)}{\Pr(r_t | y_t = 1)}. \quad (17)$$

Define:

$$\tilde{\alpha}_t \triangleq \log \frac{\alpha_t(0)}{\alpha_t(1)},$$
$$L_{ch}(y_t) \triangleq \log \frac{\Pr(r_t | y_t = 0)}{\Pr(r_t | y_t = 1)},$$
$$L_o(x_t) \triangleq \log \frac{\Pr(x_t = 0)}{\Pr(x_t = 1)}.$$

Taking logarithms on both sides of (17), we have:

$$\tilde{\alpha}_t = \log \frac{\tilde{\alpha}_{t-1} \cdot e^{L_o(x_t)} + \frac{1}{e^{L_o(x_t)}}}{e^{L_o(x_t)} + e^{L_o(x_t)}} + L_{ch}(y_t),$$
$$= \left( \tilde{\alpha}_{t-1} \oplus L_o(x_t) \right) + L_{ch}(y_t). \quad (21)$$
Likewise, in the backward recursion we have:

\[
\bar{\delta}_t \doteq \log \frac{\beta_t(0)}{\beta_t(1)},
\]

\[
= \log \frac{\gamma_{t+1}(0,0)\beta_{t+1}(0) + \gamma_{t+1}(0,1)\beta_{t+1}(1)}{\gamma_{t+1}(1,0)\beta_{t+1}(0) + \gamma_{t+1}(1,1)\beta_{t+1}(1)}
\]

\[
= \log \frac{\Pr(r_t=0|y_t=0)}{\Pr(r_t=1|y_t=1)} \cdot \frac{\beta_{t+1}(0)}{\beta_{t+1}(1)} + 1
\]

\[
= \log \frac{e^{L_e(x_{t+1})} \cdot e^{L_{ch}(y_{t+1})} + \bar{\delta}_{t+1}}{e^{L_e(x_{t+1})} + e^{L_{ch}(y_{t+1})} + \bar{\delta}_{t+1}},
\]

\[
= L_0(x_{t+1}) \oplus (L_{ch}(y_{t+1}) + \bar{\delta}_{t+1}). \tag{22}
\]

Finally we compute the output (extrinsic) information using:

\[
\Lambda_t = \log \frac{\Pr(x_t=0|Y_1^N)}{\Pr(x_t=1|Y_1^N)}
\]

\[
= \log \frac{\sum_{m} \sum_{m'} \alpha_{t-1}(m') \gamma_t(x_t=0,m',m) \beta_t(m)}{\sum_{m} \sum_{m'} \alpha_{t-1}(m') \gamma_t(x_t=1,m',m) \beta_t(m)}
\]

\[
= \log \frac{\alpha_{t-1}(0) \Pr(r_t|y_t=0) \beta_t(0) + \alpha_{t-1}(1) \Pr(r_t|y_t=1) \beta_t(1)}{\alpha_{t-1}(0) \Pr(r_t|y_t=1) \beta_t(1) + \alpha_{t-1}(1) \Pr(r_t|y_t=0) \beta_t(0)} \tag{23}
\]

Dividing the numerator and denominator in (23) by \(\alpha_{t-1}(1) \Pr(r_t|y_t=1) \beta_t(1)\), we obtain:

\[
\Lambda_t = \log \frac{e^{\sigma_{t-1}} \cdot e^{L_{ch}(y_t)} + \bar{\delta}_t + 1}{e^{\sigma_{t-1}} + e^{L_{ch}(y_t)} + \bar{\delta}_t}
\]

\[
= \sigma_{t-1} \oplus (L_{ch}(y_t) + \bar{\delta}_t). \tag{24}
\]

It can then be seen that the message-passing algorithm described in the previous section can be derived from the BCJR algorithm where \(\sigma_t = L_{ch}(y_t) + L_{e_f}(y_t)\), \(\bar{\delta}_t = L_{ch}(y_t)\) and \(\Lambda_t = L_{e}(x_t)\). For clarity, Tab. 1 summarizes the above results from the BCJR algorithm and compares them with the message-passing algorithm described in the previous section. The key advantage however is that the message-passing decoding obviates the need to compute \(\log(e^\alpha + e^\beta)\) and the need to explicitly normalize at each step. Instead, a single operation \(\log \tanh(\frac{\alpha}{2})\) is used which can be implemented using table lookup. As such, a significant amount of complexity is saved. Hence, the message-passing decoding of \(1/(1 + D)\) presents an efficient alternative of the conventional BCJR algorithm. \(\Box\)

The message-passing algorithm used by Jin et al [15] and Divsalar et al [4] is a parallel version the sequential update of \(L^{(k)}_{e_f}\) and \(L^{(k)}_{ch}\) in (7) and (8):

\[
L^{(k)}_{e_f}(y_t) = L^{(k-1)}_0(x_t) \oplus (L_{ch}(y_{t+1}) + L^{(k-1)}_{e_f}(y_{t+1})). \tag{25}
\]

\[
L^{(k)}_{ch}(y_t) = L^{(k-1)}_0(x_{t+1}) \oplus (L_{ch}(y_{t+1}) + L^{(k-1)}_{ch}(y_{t+1})). \tag{26}
\]

Obviously, since the parallel version uses the information from the last iteration rather than the most recent, the convergence may be a little slower. But for practical block sizes and for moderate decoding times, simulations have shown that the compromise in performance is only about 0.1 dB after 15 to 30 iterations.
Table 1: Summary of Sum-Product and MAP Decoding

<table>
<thead>
<tr>
<th></th>
<th>BCJR</th>
<th>Sum-product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>$\tilde{\alpha}<em>t = (\tilde{\alpha}</em>{t-1} \boxplus L_o(x_t)) + L_{ch}(y_t)$</td>
<td>$L_{ef}(y_t) = (L_{ef}(y_{t-1}) + L_{ch}(y_{t-1})) \boxplus L_o(x_t)$</td>
</tr>
<tr>
<td>Backward</td>
<td>$\tilde{\beta}<em>t = (\tilde{\beta}</em>{t+1} + L_{ch}(y_{t+1})) \boxplus L_o(x_{t+1})$</td>
<td>$L_{eb}(y_t) = (L_{eb}(y_{t+1}) + L_{ch}(y_{t+1})) \boxplus L_o(x_{t+1})$</td>
</tr>
<tr>
<td>Extrinsic LLR</td>
<td>$\Delta_t = \tilde{\alpha}_{t-1} \boxplus (\tilde{\beta}<em>t + L</em>{ch}(y_t))$</td>
<td>$L_e(x_t) = (L_{ef}(y_{t-1}) + L_{ch}(y_{t-1})) \boxplus (L_{eb}(y_t) + L_{ch}(y_t))$</td>
</tr>
</tbody>
</table>

4.2 The Min-Sum Algorithm

The main complexity in the decoder comes from the $\boxplus$ operation in both the outer TPC/SPC and inner $1/(1+D)$ decoding. Each turbo iteration (composed of one round of $1/(1+D)$ decoding followed by one round of TPC/SPC decoding) requires at least 5 $\boxplus$ operations per coded bit. A straightforward implementation of $\boxplus$ may require as many as 1 addition and 3 table lookups (assuming $\log(\tanh(\cdot))$ and $\log(\tanh^{-1}(\cdot))$ are implemented via table lookups). Although this is already lower complexity than turbo codes, it is possible and highly practical to further reduce the complexity with a slight compromise in performance. Just like the Max-log-MAP algorithm of turbo codes, the $\boxplus$ operation has a similar approximation:

$$\gamma = \alpha \boxplus \beta,$$
$$= 2 \tanh^{-1} \left( \tanh \frac{\alpha}{2} \cdot \tanh \frac{\beta}{2} \right),$$
$$= \log \frac{1 + e^{\alpha+\beta}}{e^{\alpha} + e^{\beta}},$$
$$= \text{sign}(\alpha) \cdot \text{sign}(\beta) \cdot \min(|\alpha|, |\beta|) + \log \frac{1 + e^{-|\alpha+\beta|}}{1 + e^{-|\alpha| - |\beta|}},$$
$$\approx \text{sign}(\alpha) \cdot \text{sign}(\beta) \cdot \min(|\alpha|, |\beta|). \quad (27)$$

If the approximation in (27) is used, i.e., a mere “min” operation is used instead of $\boxplus$, then a considerable reduction in complexity is achieved, and the message-passing algorithm, or the sum-product algorithm, is then reduced to the min-sum algorithm.

**Lemma 2**: Min-sum decoding of $1/(1+D)$ is equivalent to Max-log-MAP decoding.

**Proof**: Now that we have shown the equivalence of the sum-product decoding and the MAP decoding for the code $1/(1+D)$, it is straightforward to show the equivalence of min-sum decoding to max-log-MAP decoding. Max-log-MAP decoding is an approximation of the MAP or log-MAP, where the calculation of $\log(e^\alpha + e^\beta)$ is approximated as:

$$\log(e^\alpha + e^\beta) \approx \max(\alpha, \beta). \quad (28)$$

Simulation results show that the best performance/complexity gain is achieved with only one local iteration of TPC/SPC decoding in each turbo iteration between the inner and outer decoders.
Table 2: Decoding Complexity in Operations Per Data-bit Per Iteration (R is code rate)

<table>
<thead>
<tr>
<th>Code</th>
<th>outer TPC/SPC</th>
<th>inner $1/(1 + D)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoding Algorithm</td>
<td>sum-product</td>
<td>min-sum</td>
</tr>
<tr>
<td>Additions</td>
<td>$5 + \frac{1}{R}$</td>
<td>2</td>
</tr>
<tr>
<td>Min/Max</td>
<td>$5 - \frac{1}{R}$</td>
<td>$(5 \cdot 2 - 2)/R$</td>
</tr>
<tr>
<td>Table Lookup</td>
<td>$2 + \frac{2}{R}$</td>
<td>$(5 \cdot 2 - 2)/R$</td>
</tr>
</tbody>
</table>

Likewise, the only difference between the min-sum algorithm and the message-passing algorithm is that a simple “min” operation is used instead of $\oplus$ operation. This approximation is in fact a direct derivation of (28):

$$
\gamma = \alpha \oplus \beta, \\
= \log(e^\alpha + e^{\alpha+\beta}) - \log(e^\alpha + e^\beta), \\
\approx \max(0, \alpha + \beta) - \max(\alpha, \beta), \\
= \text{sign}(\alpha) \cdot \text{sign}(\beta) \cdot \min(\alpha, \beta).
$$

It thus follows that the min-sum algorithm is a computationally efficient realization of the Max-log-MAP algorithm for the decoding of $1/(1 + D)$ codes. □

5 Properties of Product Accumulate Codes

Before going through numerical results, we first show some properties of product accumulate codes to facilitate the understanding of their performance. The proposed product accumulate codes possess the following properties [2]:

(i) Property I: They are linear time encodable and linear time decodable.

(ii) Property II: They are “good” under ML decoding, which assures good performance asymptotically.

(iii) Property III: They are “good” under the practical iterative decoding approach.

5.1 Encoding and Decoding Complexity

The encoding and decoding complexity for PA codes is linear in the codeword length. The encoding process involves only a parity check in each dimension (see Section 2.1), interleaving and encoding by a rate-1 inner code (see Fig. 2), all of which require linear complexity in the block length. The decoding complexity is proportional to the number of iterations of the outer TPC/SPC code and the inner convolutional code, both of which have linear decoding complexity.
Tab. 2 summarizes the complexity of different decoding strategies for the inner and outer codes. We assume that in sum-product decoding, $\log \tanh(\frac{d}{2})$ is implemented using table lookup. The complexity of the log-MAP and Max-log-MAP algorithms is evaluated using [24] (based on the conventional implementation of the BCJR algorithm). As can be seen, the sum-product and min-sum decoding of $1/(1+D)$ require only about $1/6$ and $1/8$ the complexity of their BCJR equivalents, respectively. For a rate $1/2$ PA code, message-passing decoding requires about 33 operations per data bit per iteration, while min-sum decoding requires only about 15 operations; both are significantly less than the number of operations involved in a turbo code.

5.2 Performance under ML Decoding

In the ML-based analysis of PA codes, we first quantify the interleaving gain and then derive a tight upper bound on the word error performance. We show that under maximum likelihood decoding, the probability of word error is proportional to $P^{-1}$ for large $E_b/N_0$, where $P$ is the number of TPC/SPC codewords concatenated before interleaving. Further, we show that these codes can perform close to capacity limits by computing thresholds for these codes based on the tight upper bound on the word error rate (WER) due to Divsalar [4].

(A) Interleaving Gain

From the results of Benedetto et al [16] and Divsalar, Jin and McEliece [17], we know that for a general serial concatenated system with recursive inner code, there exists a threshold $\gamma$ such that for any $E_b/N_0 \geq \gamma$, the asymptotic word error rate is upper bounded by:

$$P_{w}^{UB} = O \left( N^{-\left[ \frac{d_m}{2}\right]} \right),$$

where $d_m$ is the minimum distance of the outer code and $N$ is the interleaver size. Whereas this result offers a useful guideline in quantifying the interleaving gain, one must be careful in interpreting it for PA codes.

The result in (30) indicates that if the minimum distance of the outer code is at least 3, then an interleaving gain can be obtained. However, the outer codewords of PA codes (with random interleavers) have minimum distance of only 2. On the other hand, if $S$-random interleavers are used such that bits within distance $S$ are mapped to at least distance $S$ apart, then the outer codewords are guaranteed to have a minimum distance of at least 3 as long as $S \geq t$. Since a block interleaver can be viewed as a structured $S$-random interleaver, it follows that interleaving gain exists for PA-II codes. Below we show that although the minimum distance of the outer codewords is only 2 over the ensemble of interleavers, an interleaving gain still exists for PA codes with random interleavers (PA-I codes). Since from (30) outer codewords of weight 3 or more will lead to an interleaver gain, we focus the investigation on weight-2 outer codewords only and show that the number vanishes as $P$ increases. The all-zero sequence is used as the reference since the code is linear.

It is convenient to employ the uniform interleaver which represents the average behavior of the ensemble of codes. Let $A_{w,j}^{(j)}$, $j = 1, 2$, denote the input output weight enumerator (IOWE) of the $j$th SPC branch code
(parallelly concatenated in the outer code). The IOWE of the outer codewords, $A_{w,h}^o$, averaged over the
code ensemble is given as:

$$A_{w,h}^o = \sum_{h_1} A_{w,h_1}^{(1)} A_{w,h-h_1}^{(2)} \left( \frac{K}{w} \right),$$

(31)

where $K = P_t$ is the input sequence length.

Define the input output weight transfer probability (IOWTP) of the $j_h$ branch code, $P_{w,h}^{(j)}$, as the probability
that a particular input sequence of weight $w$ is mapped to an output sequence of weight $h$:

$$P_{w,h}^{(j)} = \frac{A_{w,h}^{(j)}}{\left( \frac{K}{w} \right)}, \quad j = 1, 2.$$

(32)

Substituting (32) in (31), we get:

$$A_{w,h}^o = \sum_{h_1} A_{w,h_1}^{(1)} P_{w,h-h_1}^{(2)}.$$

(33)

For each branch where $P (t+1, t)$ SPC codewords are combined, the IOWE function is given as (assuming
even parity check):

$$A_{w,h}^{SPC} = \left( 1 + \left( \frac{t}{2} \right) w^2 h^2 + \left( \frac{t}{3} \right) w^3 h^4 + \left( \frac{t}{4} \right) w^4 h^4 + \cdots \right)^P,$$

(34)

$$= \left( \sum_{j=0}^t \left( \frac{t}{j} \right) w^j h^{2\lceil j/2 \rceil} \right)^P,$$

(35)

where the coefficient of the term $w^u h^v$ denotes the number of codewords with input weight $u$ and output
weight $v$. Using (35), we can compute the IOWEs of the first SPC branch code, denoted as $A_{u,v}^{(1)}$ ($= A_{w,v}^{SPC}$).

For the second branch of the SPC code, since only parity bits are transmitted, $A_{u,v}^{(2)} = A_{u,v+u}^{(1)}$.

With a little computation, it is easy to see that the number of weight-2 outer codewords is given by:

$$A_{h-2}^o = \sum_w A_{w,h-2}^o,$$

(36)

$$= P \left( \frac{t}{2} \right) \left( \frac{P_{t}^{(j)}}{P_{t}^{(2)}} \right),$$

(37)

$$= O(t^2),$$

(38)

where the last equation assumes a large $P$ (i.e. large block size). Equation (38) shows that the number of
weight-2 outer codewords is a function of a single parameter, $t$, which is related only to the rate of SPC
codes and not the block length. Now considering the serial concatenation of the outer codewords with the
inner $1/(1 + D)$ code, the overall output weight enumerator (OWE), $A_{h}^{PA}$, is:

$$A_{h=s}^{PA} = \sum_{h'} A_{h'} A_{h',h}^{1/(1+D)} \left( \frac{\binom{h}{h'}}{\binom{h}{h'}} \right),$$

(39)

$$= \sum_{h'} \sum_w A_{w,h'} A_{w,h'}^{1/(1+D)} \left( \frac{\binom{h}{h'}}{\binom{h}{h'}} \right),$$

(40)
where \( A_{w,h}^{0} \) is the OWE of the outer code, and the IOWE of the \( 1/(1 + D) \) code is given by [17]:

\[
A_{w,h}^{1/(1+D)} = \binom{N - h}{\lfloor w/2 \rfloor} \binom{h - 1}{\lceil w/2 \rceil - 1}.
\] (41)

In particular, the number of weight-\( s \) PA codewords produced by weight-2 outer codewords (for small-s), denoted as \( A_{h-s}^{PA2} \), is:

\[
A_{h-s}^{PA2} = \frac{(t - 1)^2 N - s}{2},
\] (42)

\[
= O(tP^{-1}),
\] (43)

where \( N = P(t + 2) \) is the PA codeword length. This indicates that the number of small weight \( s \) codewords of the overall PA code due to weight-2 outer codewords (caused by weight-2 input sequences) vanishes as \( P \) increases. When the input weight is greater than 2, the outer codeword always has weight greater than 2 and, hence, an interleaving gain can be guaranteed. Hence, an interleaving gain exists for PA codes and it is proportional to \( P \).

(B) Upper bounds

To further shed insight into the asymptotic performance \( (N \to \infty) \) of PA codes under ML decoding, we compute thresholds for this class of codes based on the bounding technique recently proposed by Divsalar [4]. The threshold here refers to the capacity of the codes under ML decoding, i.e. the minimum \( E_b/N_0 \) for which the probability of error decreases exponentially in \( N \) and, hence, tends to zero as \( N \to \infty \).

Among the various bounding techniques developed, the union bound is the most popular but it is fairly loose above the cutoff rate. Tighter and more complicated bounds include the tangential sphere bound by Poltyrev [25], the Viterbi and Viterbi bound [26], Duman-Salehi bound [27], the Hughes bound [28]. These new tight bounds are essentially based on the bounding techniques developed by Gallager [29]:

\[
\Pr(\text{error}) \leq \Pr(\text{error}, \bar{y} \in \mathcal{R}) + \Pr(\bar{y} \notin \mathcal{R}),
\] (44)

where \( \bar{y} \) is the received codeword (matched filter samples of the noise-corrected codeword), and \( \mathcal{R} \) is a region in the observed space around the transmitted codeword. To get a tight bound, the above methods usually require optimization and integration to determine a meaningful \( \mathcal{R} \).

Recently, Divsalar developed a simple bound on error probability over AWGN channels [4]. The bound is also based on (44), but a simple closed-form expression is derived and shown that the computed minimum SNR threshold can serve as a tight upper bound on the ML capacity of nonrandom codes. The simple bound is the tightest closed-from bound developed so far. It is also shown that, as block size goes to infinity, this simple bound is equivalent to the tangential sphere bound [4]. Below we apply this simple bounding technique to the analysis of PA codes.

We first quote and summarize the main results of [4]. Define the spectral shape of a code, \( \gamma_N(\delta) \), as the normalized weight distribution averaged over the code ensemble \( \mathcal{C}_N \):

\[
\gamma_N(\delta) = \frac{1}{N} \ln(A_{h-\lfloor\delta N\rfloor}), \quad 0 < \delta < 1,
\] (45)
where $N$ is the code length, $A_h$ is the (average) output weight enumerator of the code. Further, define the ensemble spectral shape as:

$$\gamma(\delta) \triangleq \lim_{N \to \infty} r_N(\delta), \quad 0 < \delta < 1.$$  \hfill (46)

It can be shown that the probability of word error can be upper bounded by [4]:

$$P_w(e) \leq \sum_h e^{-NE(E_b/N_0,h)},$$  \hfill (47)

where

$$E(c, h) = -\gamma(\delta) + \frac{1}{2} \ln \left( [1 + (1 - \beta)e^{2\gamma(\delta)}] + \frac{\delta \beta}{1 - \delta(1 - \delta)} \right),$$  \hfill (48)

where

$$\beta = \sqrt{c \left( \frac{1 - \delta}{\delta} - 1 \right) - \frac{1 - \delta}{\delta} (1 + c)}.$$  \hfill (49)

The threshold $C_{ML}^*$ is defined as the minimum $E_b/N_0$ such that $E(E_b/N_0, h)$ is positive for all $h$ and, hence, for all $E_b/N_0 \geq C_{ML}^*$, $P_w(e) \to 0$ as $N \to \infty$. The threshold can be computed as [4]:

$$C_{ML}^* = \frac{1}{R} \max_{0 < \delta \leq (1 - R)} c_0(\delta),$$  \hfill (50)

where $R$ is the code rate. For the simple bound, $c_0(\delta)$ is given by:

**Simple:** $c_0(\delta) = \frac{1 - \delta}{2\delta} \left( 1 - e^{-2\gamma(\delta)} \right).$  \hfill (51)

Similar forms are also derived for Viterbi-Viterbi, Hughes, and Union bounds [4]:

**Viterbi:** $c_0(\delta) = \frac{1 - \delta}{\delta} \gamma(\delta),$  \hfill (52)

**Hughes:** $c_0(\delta) = \frac{1 - e^{-2\gamma(\delta)}}{2\delta},$  \hfill (53)

**Union:** $c_0(\delta) = \frac{\gamma(\delta)}{\delta}.$  \hfill (54)

Since the above bounds are based on the ensemble spectral shape $r(\delta)$, they serve as the asymptotic performance limit (i.e. $N \to \infty$) of the code ensemble assuming ML decoding.

There is no simple closed form expression for the ensemble spectral shape of PA codes. However, the spectral shape can be computed to a good accuracy numerically since the component codes of the concatenation are single parity check codes. Specifically, using (33), (40) and (45) we can compute the spectral shape of PA codes, which is a function of $N, P, t$. We approximate the ensemble spectral shape by choosing a large $N$. Whenever possible, input output weight transfer probability, $P_{w,h}$, should be used instead of input output weight enumerator, $A_{w,h}$, to eliminate numerical overflow. The bounds for GPA codes are computed and plotted in Fig. 6 (for clarity, only the simple bound and the union bound are shown). For comparison, also shown are the bounds for random codes and the Shannon limit. Several things can be observed: (1) the simple bounds of PA codes are very close to those of the random codes, indicating that PA codes have good distance spectrum; (2) the higher the rate, the tighter the bound is, indicating that GPA codes are likely more advantageous at high rates than low rates (as opposed to repeat accumulate codes).
The implication of the above analysis is that PA codes are capable of performance a few tenths of a dB away from the capacity limit with ML decoding. However, since there does not exist a computationally feasible ML decoder, it is desirable to investigate the iterative decoding process in order to give a more meaningful evaluation of the code performance with a practical decoder.

5.3 Performance under Iterative Decoding

In this section we show that product accumulate codes are “good” codes also in the iterative sense. We show this by computing a threshold (minimum $E_b/N_0$) for this class of codes such that when the channel signal-to-noise ratio (SNR) is higher than the threshold the error probability goes to zero (with infinite block size). We compute this threshold by means of density evolution (DE). Density evolution has been shown to be a very powerful tool in the analysis and design of LDPC codes [5] [30] [31] [32]. Following the message-passing decoding on the code graph (Fig. 4), we will explain how the DE procedure can be applied to compute the thresholds for PA codes. By examining the distribution of the messages passed within and in-between the subdecoders, we are able to determine the fraction of incorrect messages (extrinsic messages of the wrong sign). The basic idea is that if the fraction of incorrect messages goes to zero with the increase of the number of iterations, then the decoding procedure will eventually converge to the correct codeword.

The analysis of product accumulate codes involves computation of the probability density function (pdf) of the message flow within the outer decoder, the inner decoder and in-between the two. However, the unconstrained DE procedure is quite complex since the pdf that evolves with iterations may not have a closed-form expression and one has to keep track of an infinite dimensional vector (pdf). Hence, density evolution takes a numeric approach. It is worth mentioning that a simplified approximation can be made by assuming that the messages passed in each step follow Gaussian distributions. This Gaussian assumption trades a little accuracy for a considerable reduction in computational complexity when combined with the consistency condition which states that the distribution $f$ of messages $w$ passed in each step satisfies $f(w) = f(-w)e^w$ [32]. Here, to preserve the accuracy, we perform the exact density evolution (with quantization).

Exploiting the linearity of the code, we assume the all-zero codeword is transmitted. It is convenient to use log likelihood ratios as messages to examine the decoding process. The threshold, which serves as the practical capacity limit for a given code (given rate and decoding strategy), is thus formulated as:

$$C_{\text{iterative}} = \inf_{\text{SNR}} \left\{ \text{SNR} : \lim_{k \to \infty} \lim_{N \to \infty} \int_{-\infty}^{0} f_{L_k^{(w)}(x)} \, dx \to \infty \right\},$$

(55)

where $f_{L_k^{(w)}(x)}$ is the pdf of the messages (extrinsic information) evaluated at the output of the outer decoder, (note due to the i.i.d. assumption, we have dropped the dependence $i$ on $x_i$,) superscript $(k)$ denotes the $k_{th}$ iteration between the outer and inner decoder, and $N$ is the block size. Before we describe how DE is performed numerically for the case of PA codes, we first discretize messages. Let $Q(w)$ denote the quantization operation on message $w$ with a desired quantization interval (accuracy) $\Delta$.

A. Message Flow within the Outer Decoder
The outer code of the general product codes (PA-I) consists of 2 parallel concatenated branches where each branch is formed of \(P\) blocks of \((t + 1, t)\) SPC codewords. The outer code (alone) can also be considered as a special case of LDPC codes whose parity check matrix has 2\(P\) rows with uniform row weight of \((t + 1)\), and \(p(t + 2)\) columns with \(\frac{p}{p}2\) percent of the columns having weight 2 and the rest weight 1. Therefore, the exact decoding algorithm for LDPC codes can be applied to the outer code. However, for a more efficient convergence, we could make use of the fact that the checks in the outer code can be divided into two groups (corresponding to the upper and lower branch, respectively) such that the corresponding sub-graph (Tanner graph) of each group is cycle-free. It thus leads to a serial message-passing mode where each group of checks take turns to update (as opposed to the parallel update of all checks in LDPC codes).

The fundamental element in the decoding of the outer code is the decoding of SPC codes. Consider the upper branch. Suppose data bits \(d_{i,1}, d_{i,2}, \cdots, d_{i,t}\) and parity bit \(p_i\) participate in the \(i\)th SPC codeword \((1 \leq i \leq P)\). Then the messages (extrinsic information) for each bit obtained from this check (during the \(k\)th turbo iteration and \(l\)th local iteration) are:

\[
\text{data bit: } L_{e1}^{(k,l)}(d_{i,j}) = \left( \sum_{1 \leq k \leq t, k \neq j} \left( L_{e1}^{(k)}(d_{i,k}) + L_{e2}^{(k-1)}(d_{i,k}) \right) \right) \odot L_{e1}^{(k)}(p_i),
\]

\(
\iff \text{tanh} \frac{L_{e1}^{(k,l)}(d_{i,j})}{2} = \left( \prod_{1 \leq k \leq t, k \neq j} \text{tanh} \frac{L_{e1}^{(k)}(d_{i,k}) + L_{e2}^{(k-1)}(d_{i,k})}{2} \right) \cdot \text{tanh} \frac{L_{e1}^{(k)}(p_i)}{2}, (56)
\)

\[
\text{parity bit: } L_{e1}^{(k,l-1)}(p_i) = \left( \sum_{1 \leq k \leq t} \left( L_{e1}^{(k)}(d_{i,k}) + L_{e2}^{(k-1)}(d_{i,k}) \right) \right),
\]

\(
\iff \text{tanh} \frac{L_{e1}^{(k,l-1)}(p_i)}{2} = \left( \prod_{1 \leq k \leq t} \text{tanh} \frac{L_{e1}^{(k)}(d_{i,k}) + L_{e2}^{(k-1)}(d_{i,k})}{2} \right), (57)
\)

where \(L_{e}^{(\cdot)}\) denotes the messages (a priori information) received from the inner code, \(L_{e1}^{(\cdot)}\) denotes the messages (extrinsic information) obtained from the upper SPC branch to be passed to the lower branch and \(L_{e2}^{(\cdot)}\) denotes the messages to be passed from the lower branch to the upper branch. After interleaving, similar operations of (56) and (57) are performed within the lower branch. We assume \(L_{e1}^{(\cdot)}\) and \(L_{e2}^{(\cdot)}\) to be independent and identically distributed (i.i.d) random variables and drop the dependence on \(i\) and \(j\).

We use superscript \((k, l)\) to denote the \(k\)th turbo iteration between the outer decoder and inner decoder and the \(l\)th iteration within the outer decoder (local iterations). For independent messages to add together, the resulting pdf of the sum is the discrete convolution of the component pdf’s. This calculation can be efficiently implemented using an fast Fourier transform (FFT). For the tanh operation on messages, define:

\[
\gamma = \alpha \oplus \beta \triangleq Q \left( 2 \text{tanh}^{-1} \left( \frac{\alpha}{2} \text{tanh} \frac{\beta}{2} \right) \right), (58)
\]

where \(\alpha\), \(\beta\) and \(\gamma\) are quantified messages. The pdf \(f_{\gamma}\) of \(\gamma\) can be computed using:

\[
\gamma[i, k] = \sum_{(i,j): k = i \oplus j \Delta} f_{\alpha}[i] \cdot f_{\beta}[j], (59)
\]
To simplify the notation, we denote this operation (59) as:

$$f_{\gamma} = \mathcal{R}(f_{\alpha}, f_{\beta}).$$

(60)

In particular, using induction on the above equation, we can denote:

$$\mathcal{R}^k(f_{\alpha}) \triangleq \mathcal{R}(f_{\alpha}, (\mathcal{R}(f_{\alpha}, \cdots, \mathcal{R}(f_{\alpha}, f_{\alpha}) \cdots))).$$

(61)

It then follows from (56), (57) and (61) that the pdf of the extrinsic messages obtained from the upper branch \(f_{L_{e1}}(\cdot)\), and the lower branch, \(f_{L_{e2}}(\cdot)\), are given by:

**Upper branch: data bit:**

$$f_{L_{e1,d}}^{(k,t)} = \mathcal{R} \left( f_{L_{e,d}}^{(k)}, \mathcal{R}^{(t-1)} \left( f_{L_{e,d}}^{(k)} * f_{L_{e2,d}}^{(k,t-1)} \right) \right),$$

(62)

**parity bit:**

$$f_{L_{e1,p}}^{(k,t)} = \mathcal{R}^{t} \left( f_{L_{e,d}}^{(k)} * f_{L_{e2,d}}^{(k,t-1)} \right),$$

(63)

**Lower branch: data bit:**

$$f_{L_{e2,d}}^{(k,t)} = \mathcal{R} \left( f_{L_{e,d}}^{(k)}, \mathcal{R}^{(t-1)} \left( f_{L_{e,d}}^{(k)} * f_{L_{e1,d}}^{(k,t)} \right) \right),$$

(64)

**parity bit:**

$$f_{L_{e2,p}}^{(k,t)} = \mathcal{R}^{t} \left( f_{L_{e,d}}^{(k)} * f_{L_{e1,d}}^{(k,t)} \right),$$

(65)

where \(f_{L_e}^{(k)}(\cdot)\) denotes the pdf of the messages \(f_{L_e}^{(k)}(\cdot)\) from the inner \(1/(1+D)\) code in the \(k\)th turbo iteration, \(f_{L_{e1}}^{(k,t)}(\cdot)\) and \(f_{L_{e2}}^{(k,t)}(\cdot)\) denote the pdf’s of the extrinsic information from the upper and lower branch of the outer code, \(L_{e1}^{(k,t)}(\cdot)\) and \(L_{e2}^{(k,t)}(\cdot)\), respectively, and * denotes the discrete convolution. The subscript \(d\) refers to a data bit and \(p\) refers to the parity bit.

Since the systematic bits (data) and the parity bits of the outer code are treated the same in inner \(1/(1+D)\) code, we have \(f_{L_{e,d}}^{(k)} = f_{L_{e,p}}^{(k)} = f_{L_{e,v}}^{(k)}\), where \(f_{L_{e,v}}^{(k)}\) is pdf of the extrinsic information \(L_{e,v}\) obtained from \(1/(1+D)\) (also refer to the next section for a detailed explanation). For PA-I codes, the local iterations within the outer code only involve the exchange of messages associated with data bits (as can be seen from the above equations). After \(L\) local iterations, the messages the outer code passes along to the inner code include those of data bits \(L_{e1}(d)\) and \(L_{e2}(d)\) and parity bits \(L_{e1}(p)\) and \(L_{e2}(p)\), which thus leads to a mixed message density with a fraction \(t/(t+1)\) having pdf \((f_{L_{e1,d}} * f_{L_{e2,d}})\) and equal fractions \(1/(2t+2)\) having mean \(f_{L_{e1,p}}\) and \(f_{L_{e2,p}}\) respectively (note these fractions are from the edge perspective in the bipartite code graph of the outer code). This will in turn serve as the pdf of the a priori information, \(f_{L_{e,v}}^{(k+1)}\), to the inner decoder.

A similar serial update procedure can also be used with PA-II codes, and the message-passing analysis is much the same. For a PA-II code with conventional \((K_1 + 1, K_1) \times (K_2 + 1, K_2)\) TPC/SPC codes (with block interleavers and parity-on-parity bits) as the outer code, the means of the extrinsic messages associated with row code and column code, \(L_{e1}(\cdot)\) and \(L_{e2}(\cdot)\), can be computed using (also refer to Appendix I for the decoding algorithm of TPC/SPC codes):

$$f_{L_{e1}}^{(k,t)} = \mathcal{R} K_1 \left( f_{L_{e}}^{(k)} * f_{L_{e2}}^{(k,t-1)} \right),$$

(66)

$$f_{L_{e2}}^{(k,t)} = \mathcal{R} K_2 \left( f_{L_{e}}^{(k)} * f_{L_{e1}}^{(k,t)} \right).$$

(67)
Unlike PA-I codes, the data bits and the parity bits are treated exactly the same in the outer code of PA-II codes. Hence, the pdf of the messages passing along to the inner $1/(1 + D)$ decoder is given by $(f_{L_{e_1}}^{(k, L)} \ast f_{L_{e_2}}^{(k, L)})$ after $L$ rounds of local iterations.

B. Message Flow within the inner $1/(1 + D)$ Decoder

Similar to the treatment of TPC/SPC codes, we assume that messages (LLRs) are i.i.d. for the $1/(1 + D)$ code. From (25) and (26), it is obvious that for sufficiently long sequences, messages $L_{e_1}(y)$ and $L_{e_2}(y)$ follow the same distribution. Note that we are somewhat abusing the notation here by dropping the dependence on $i$, which denotes the transmission at the $i$th epoch. This is because on a memoryless channel the pdfs of $L_{e_1}(y_i)$ and $L_{e_2}(y_i)$ are independent of $i$. Further as can be seen from the message-passing algorithm, the forward and the backward passes are symmetric and, hence, for large block sizes, $L_{e_1}(y)$ and $L_{e_2}(y)$ follow the same pdfs. Thus we drop the subscript and use $L_{e}(y)$ to represent both $L_{e_1}(y)$ and $L_{e_2}(y)$. It was verified by simulations that the serial (see (7) and (8)) and parallel (see (25) and (26)) modes do not differ in performance significantly (only about 0.1 dB as shown in Fig. 17), especially with sufficient number of turbo iterations. It is convenient to use the parallel mode for analysis here. Hence messages (LLRs) as formulated in (4) and (25), (26) have their pdfs evolve as:

$$f_{L_{e,x}}^{(k)} = R^2 (f_{L_{e,y}}^{(k)} \ast f_{L_{e,y}}^{(k)}),$$

(68)

where

$$f_{L_{e,y}}^{(k)} = R \{ f_{L_{e,y}}^{(k-1)} \ast f_{L_{e,y}}^{(k-1)} \}.$$

(69)

The initial conditions are $f_{L_{e,y}}^{(0)} = \mathcal{N}(2/\sigma^2, 4/\sigma^2)$ (Gaussian distribution of mean $2/\sigma^2$ and variance $4/\sigma^2$) and $f_{L_{e,x}}^{(0)} = \delta(x)$ (Kronecker delta function).

The message flow between the inner and outer codes is straight-forward. The pdf of the outbound message, $f_{L_{e,x}}^{(k)}$ in (68), becomes the pdf of the a priori information, $f_{L_{e,x}}^{(k)}$ and $f_{L_{e,x}}^{(k)}$ in (62) - (65) (PA-I code) and $f_{L_{e}}^{(k)}$ in (66) and (67) (PA-II code). Likewise, the pdf of the extrinsic information from the outer TPC/SPC code, 

$$\left( \frac{1}{L_{e_1}} f_{L_{e_1},d}^{(k, L)} \ast f_{L_{e_2},d}^{(k, L)} \right) \ast \frac{1}{L_{e_2}} f_{L_{e_1},p}^{(k, L)} \ast \frac{1}{L_{e_2}} f_{L_{e_2},p}^{(k, L)}$$

for PA-I codes and $(f_{L_{e_1}}^{(k, L)} \ast f_{L_{e_2}}^{(k, L)})$ for PA-II codes, becomes the pdf of a priori information, $f_{L_{e,x}}^{(k+1)}$ in (69), for inner the $1/(1 + D)$ code.

It should be noted that the assumption that all messages passed are independent and identically distributed is required for the derivation of (56) and (57) for PA-I codes. However, the same cannot be directly used to analyze PA-II codes. Due to the use of the random interleaver in the PA-I code structure, it is reasonable to assume that the neighborhood of each node is tree-like. However, in the case of PA-II codes, when a block interleaver is used in the TPC/SPC code, length-8 cycles are unavoidable (even when $N \rightarrow \infty$). Hence, partial independence holds only when the message flow in the decoding has not closed a length-8 cycle. In other words, the number of times (66) and (67) can be applied consecutively is strictly limited to be no more than $\log_2 \frac{8}{2} = 2$, before messages need to be passed out to the $1/(1 + D)$ decoder. In fact, due to the serial update, even 2 local iterations will incur the looping of the same message [8] and, hence,
we take $L = 1$ for analysis. Further, during every global iteration $(k)$, the extrinsic messages within the TPC/SPC code generated in the previous iterations, $L_{c1}^{(k-1, L)}$ and $L_{c2}^{(k-1, L)}$, should not be used again since this represents correlated information. Due to the above reasons, the resulting thresholds are upper bounds (pessimistic case).

Fig. 7 shows the thresholds for PA-I codes for several rates $R \geq 0.5$. It can be seen that the thresholds are within 0.6 dB from the Shannon limit for BPSK on an AWGN channel. The thresholds are closer as the rate increases suggesting that these codes are better at higher rates. The thresholds for PA-II codes are shown in Fig. 8. The plotted thresholds in Fig. 8 are a lower bound on the capacity (upper bound on the thresholds) since only one iteration is performed in the outer TPC/SPC decoding in each turbo iteration (i.e. $L = 1$ in (67)) [8]. Note that at high rates ($R > 0.7$), the capacity of product accumulate codes (both PA-I and PA-II) is within 0.5 dB from the Shannon limit. However, at lower rates the gap becomes larger especially for PA-II codes. Simulation results for fairly long block sizes are also shown in both Fig. 7 and Fig. 8. A block size of $K = 64$K data bits was used for $R = 1/2$ and for the higher rates $K = 16$K was used and a BER of $10^{-5}$ is taken as reference. It can be seen that the simulation results are quite close to the thresholds. This shows that PA-I codes and PA-II codes are both capable of good performance at high rates, however, at lower rates PA-I codes are significantly better.

6 Algebraic Interleaver

Observe that a rate-$K/N$ PA-I code involves two random interleavers of sizes $K$ and $N$, where $K$ and $N$ are the user data block size and codeword block size, respectively. Interleaving and deinterleaving using look-up tables can be quite inefficient in hardware and, hence, we study the performance of PA codes under algebraic interleaving. That is, we use interleavers where the interleaving pattern can be generated on the fly without having to store the interleaving pattern. We consider congruential sequence generated according to [33]:

$$A_{n+1} = (a \cdot A_n + b) \mod N. \quad (70)$$

To assure that this generates a maximal length sequence from 0 to $N-1$, parameters $a$ and $b$ need to satisfy:

1. $a < N$, $b < N$, $b$ be relatively prime to $N$;
2. $(a-1)$ be a multiple of $p$, for every prime $p$ dividing $N$;
3. particularly, $(a-1)$ be a multiple of 4 if $N$ is a multiple of 4. It is also desirable though not essential that $a$ be relatively prime to $N$;

We consider such an interleaver for both the interleavers in the proposed code. This can also be considered as an algebraic design of the code graph since the graph structure can be directly specified by the interleaving sequence. Hence, given an $N$ and $t$, the choice of $a$ and $b$ completely specifies the code graph and, hence, the encoding and decoding operations.

Another direct benefit of using algebraic interleavers is that it allows great flexibility for PA codes to change code rates as well as code length. With LDPC codes, however, it is not easy to change code lengths nor code rates using one encoder/decoder structure. Although LDPC codes can be defined with a bit/check degree profile and a random interleaver (see Fig. 18), encoding requires the availability of the generator
matrix. In other words, with LDPC codes, for each code rate and code length, not only does the code structure (connections between bits and checks) need to be devised specifically, but the generator matrix needs to be stored individually. Although possible, it requires special treatment to accommodate several rates/block sizes in one LDPC encoder/decoder pair.

7 Simulation Results of PA Codes

Performance of PA-I codes at medium rate: Fig. 9 shows the performance of a rate-1/2 PA-I code of data block size 64K, 4K and 1K, respectively. As can be seen, the larger the block size, the steeper the performance curve, which clearly depicts the interleaving gain phenomenon. For comparison purpose, the performance of a (2K,1K) turbo code from [15] and the most recently reported irregular repeat accumulate (IRA) codes [15] of the same parameters are also shown. As can be seen, (2K, 1K) PA-I codes perform as well as the turbo codes at BER of $10^{-5}$ with no error floors. From Tab. 2, we can see that the decoding complexity of rate-1/2 PA codes with 30 iterations is approximately 1/16 that of a 16-state turbo code with 8 iterations. It is also important to note that the complexity savings are higher as the rate increases, since the decoding complexity of punctured turbo codes does not reduce with increasing rate, whereas the decoding complexity of PA codes is inversely proportional to the rate. It should also be noted that the curve of PA-I codes is somewhat steeper than that of turbo codes or irregular repeat accumulate codes, and therefore may outperform them at lower BERs.

Performance of PA-I codes at high rate: As indicated by both ML-based and iterative-based analysis, product accumulate codes are most advantageous at high rates. Fig. 10 compares the performance of a rate-3/4 PA-I code at $15_{th}$ and $20_{th}$ iteration with a 16-state turbo code of polynomials $(37,23)_6$ at $4_{th}$ iteration. Data block size is $K = 1002$ for both codes. Clearly, while PA-I code is comparable to turbo code (Fig. 9) at rate-1/2, it significantly outperforms turbo codes at rate-3/4 (much steeper curves and no error floors). Further, PA-I code at $15_{th}$ and $20_{th}$ iteration requires only about 23% and 30% the complexity of the turbo code at $4_{th}$ iteration, respectively. Hence, PA codes are suitable for applications requiring high-rates with the advantages of low-complexity, good BER performance and no error floors.

Performance of PA-II codes: Fig. 11 plots the bit error rate performance of PA-II codes at high rates. The codes simulated have rates 0.88, 0.94 and 0.97, which are formed from $(16,15)^2$, $(32,31)^2$ and $(64,63)^2$ outer 2-D TPC/SPC codes, respectively. Since interleaving gain is directly proportional to the number of TPC/SPC blocks in a codeword, several TPC/SPC blocks may be combined to achieve a large effective block size when needed. Corresponding threshold bounds calculated by density evolution are also shown. Two things can be immediately seen from the plot: (1) product accumulate codes demonstrate a significant performance improvement than plain TPC/SPC codes (Fig. 13). A 1 dB gain is achieved for rate-0.97 codes, while as much as a 3 dB gain is achieved for rate-0.88 codes; (2) with a data block size of $K = 16K$, the performance of PA-II codes is within 0.3 dB from the capacity bound at BER of $10^{-5}$, showing a very good fit. All curves shown are after 15 turbo iterations. Although not plotted here, a reduction from 15 to 8
iterations incurs only about 0.1 dB loss.

**PA-I v.s. PA-II:** As expected, PA-I codes tend to perform better than PA-II codes at higher rates also, but the difference is relatively small, since PA-II codes are already very close to the Shannon limit at high rates. Fig. 12 compares the performance of PA-I and PA-II codes at rate-0.94. As can be seen, the difference seems more noticeable at a moderate size (16K) than either very short block size (1K) where the random interleaver is too short to play a significant role or very long block size (64K) where the performance is already close to the threshold.

**Performance of the min-sum decoding:** Fig. 14 compares the performance of a rate-0.5 PA-I code with the sum-product decoding to the low-complexity min-sum decoding. Performance at 5, 10, 15, 20 iterations is evaluated. At all these iterations, the min-sum decoding incurs only about 0.2 dB loss, while saving more than half the complexity. Hence, the min-sum algorithm provides a good tradeoff between performance and complexity, and is thus very appealing for simple, low-cost systems. Fig. 15 compares the performance of PA-I codes using the min-sum decoding to the performance of a serial concatenated convolutional codes (SCCC or serial turbo) of the same code rate and block size. The serial turbo code is composed of an outer 4-state and an inner 2-state convolutional code. It is interesting to see that even with the low-complexity min-sum decoding, the PA-I code still outperforms the SCCC code. Comparing the performance of the PA-I code (using min-sum decoding) at 15th iteration with that of the serial turbo at 4th iteration, we see that a 0.4 dB performance gain is achieved at BER of $10^{-5}$ with only about 60% of the complexity, which is impressive (see Tab. 2 for a complexity analysis).

**Algebraic interleaver v.s. S-random interleaver:** Fig. 16 compares the performance of a rate-0.5 PA-I code with S-random interleavers and algebraic interleavers. Interestingly, replacing S-random interleavers with algebraic interleavers results in hardly any performance degradation. Since the length of algebraic interleavers can be conveniently changed, using algebraic interleavers can lend another degree of flexibility to PA codes.

**Serial sum-product decoding v.s. parallel sum-product decoding:** Fig. 17 compares the performance of a rate-0.5 PA-I code with serial sum-product decoding and parallel sum-product decoding. We see that the difference between the two approaches is about 0.1 dB at bit error rate of $10^{-5}$. Hence, parallel sum-product decoding serves as a good candidate for hardware implementation.

8 **Comparison to Other Related Codes**

Graphical representation of codes has shed great insight into the understanding of many codes [18] [19] [20] [21], including turbo codes, LDPC codes and (irregular) repeat accumulate (RA/IRA) codes [15] [17]. This section revisits PA codes from the graph perspective for a comparison and unification of PA codes and other capacity-approaching codes.

The Tanner graph structure shown in Fig. 18 reveals that PA codes are essentially LDPC codes with two levels of checks instead of one as in conventional LDPC codes, small circles denote bits and small boxes
denotes checks). However, the encoding complexity of PA codes is linear and the encoder is easy to implement since it does not require explicit storage of a generator matrix.

Repeat-accumulate codes [17], and their improved version, irregular repeat-accumulate codes [15], are a class of very nice codes, which are linear time encodable and provide near capacity performance. A careful study of the code graph shows an intrinsic connection between the structure of the proposed PA codes and RA/IRA codes, although our initial motivation for PA codes are from the encouraging performance of TPC/SPC codes over partial response channels in magnetic recording systems [3]. Fig. 18 presents the Tanner graphs of proposed PA codes and RA/IRA codes. One difference is that a PA code is non-systematic whereas the systematic bits are transmitted explicitly in RA/IRA codes. One advantage of PA codes is their regular code structure which facilitates implementation. Further, the rate of the code can be easily changed at the transmitter and receiver, since the structure of the codes is identical for all rates. For IRA codes, a different irregularity pattern and associated graphs have to be designed and stored for different rates. Similarly, the length of PA codes can be easily changed if algebraic interleavers are used since the structure remains unchanged with length.

Recently, concatenated tree (CT) codes were shown to be a good lower complexity alternative to turbo codes in [34]. Simulation results for rates 1/2, 0.7 and 0.88 show that the BER performance and decoding complexity of PA codes are similar to those of concatenated tree codes. In fact, the decoding complexity seems to be slightly lower for PA codes. Again, the advantage is in the ease of rate change at the transmitter and receiver. It should also be noted that the performance of PA codes is also on par with finite-geometry LDPC based codes proposed by Shou, Lin and Fossorier [35].

9 Conclusion and Future Work

A class of interleaved serially concatenated codes called product accumulate codes has been constructed and shown to possess good BER performance, linear encoding/decoding complexity as well as an efficient soft-decoding algorithm. The proposed codes consist of an outer 2-D TPC/SPC code, a random interleaver, and a rate-1 recursive convolutional inner code of the form 1/(1+D). The main advantages of the proposed codes are very low decoding complexity compared to turbo codes especially for high rates, good BER performance and ease of implementation. Through analysis and simulations, we show this class of proposed codes perform well for almost all rates $R \geq 1/2$ and for long and short block sizes alike. Further there are no error floors observable.

Future work on PA codes includes extension to rates below 1/2 by packing more levels of single-parity check codes in the outer TPC/SPC codes. Irregularity can be introduced to the code structure to further improve performance; however, one advantage of PA codes seems to be regular and simple structure which makes implementation easier. Future work will also involve extending to fading channels and multilevel modulation.
Appendix I: Decoding algorithm for TPC/SPC codes

Assuming even-parity check codes, BPSK modulation (0 → +1, 1 → −1) and AWGN channels, a 2-D TPC/SPC code formed from \((N_1, N_1 - 1) \otimes (N_2, N_2 - 1)\) \((N_2\) rows and \(N_1\) columns) has the following SISO decoding algorithm (Tab. 3), where \(r_{i,j}\) denotes the bits received from the channel, \(L_{i,j}\) denotes the a priori information (obtained from the channel or the inner code in a concatenated scheme), \(LLR_{i,j}\) denotes the log-likelihood ratio, and \(Le_{i,j}^{(1)}\) and \(Le_{i,j}^{(2)}\) denotes the extrinsic information associated with component code \(C_1\) and \(C_2\) respectively.

<table>
<thead>
<tr>
<th>Initialization:</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (i = 1) to (N_2), for (j = 1) to (N_1),</td>
</tr>
<tr>
<td>(L_{i,j} = \frac{2}{\sigma} r_{i,j};)</td>
</tr>
<tr>
<td>(Le_{i,j}^{(1)} = Le_{i,j}^{(2)} = 0,)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iterations:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoding row code (C_1): for (i = 1) to (N_2), for (j = 1) to (N_1),</td>
</tr>
<tr>
<td>(Le_{i,j}^{(1)} = 2 \tanh^{-1}(\prod_{1 \leq t \leq N_1, t \neq j} \tanh \left( \frac{L_{i,t} + Le_{i,t}^{(1)}}{2} \right)),)</td>
</tr>
<tr>
<td>Decoding column code (C_2): for (j = 1) to (N_1), for (i = 1) to (N_2),</td>
</tr>
<tr>
<td>(Le_{i,j}^{(2)} = 2 \tanh^{-1}(\prod_{1 \leq t \leq N_2, t \neq i} \tanh \left( \frac{L_{t,j} + Le_{t,j}^{(1)}}{2} \right)),)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Soft output and decision:</th>
</tr>
</thead>
<tbody>
<tr>
<td>for (i = 1) to (N_2), for (j = 1) to (N_1),</td>
</tr>
<tr>
<td>(LLR_{i,j} = L_{i,j} + Le_{i,j}^{(1)} + Le_{i,j}^{(2)};)</td>
</tr>
<tr>
<td>(\hat{s}<em>{i,j} = LLR</em>{i,j} &gt; 0 \ ? \ 0 : 1;)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration stop criteria:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success: All rows and columns add up (modulo-2) to 0.</td>
</tr>
<tr>
<td>Fail: A max number of iteration is reached.</td>
</tr>
</tbody>
</table>

Table 3: Decoding algorithm for 2-D TPC/SPC codes

References


Figure 1: Structure of TPC codes

Figure 2: System model for product accumulate (PA-II) codes

Figure 3: System model for product accumulate (PA-I) codes
Forward pass in $1/(1+D)$:

- $x_i$ is the input bit to $1/(1+D)$.
- $y_i$ is the observation from the channel.
- $L_{i,1}(y_i)$ and $L_{i,0}(y_i)$ are the forward messages.

Backward pass in $1/(1+D)$:

- $x_i$ is the input bit to $1/(1+D)$.
- $y_i$ is the observation from the channel.
- $L_{i,1}(y_i)$ and $L_{i,0}(y_i)$ are the backward messages.

Figure 4: Code Graph and Message-Passing Decoding for $1/(1 + D)$

Trellis of $1/(1 + D)$ Code

Figure 5: Trellis of $1/(1 + D)$ Code
Figure 6: The union bound and the simple bound of product accumulate codes (PA-I)

Figure 7: Thresholds for PA-I codes (simulations are evaluated at BER $10^{-5}$)
Figure 8: Thresholds for PA-II codes (simulations are evaluated at BER $10^{-5}$)

Figure 9: Performance of PA-I codes at rate-1/2
Figure 10: Performance of PA-I codes at rate-3/4

Figure 11: Performance of PA-II codes at high rates
Figure 12: Comparison of PA-I and PA-II codes at high rates

Figure 13: Performance of plain TPC/SPC codes over AWGN channels
Figure 14: Sum-product decoding vs min-sum decoding

Figure 15: Min-sum decoding of PA-I code
Figure 16: Algebraic Interleaver vs S-random Interleaver

Figure 17: Serial sum-product decoding vs parallel sum-product decoding

Figure 18: Tanner Graphs of PA Codes, LDPC Codes and RA Codes