

# Concatenated Tree Codes: A Low-Complexity, High-Performance Approach

Li Ping, *Member, IEEE*, and Keying Y. Wu

**Abstract**—This paper is concerned with a family of concatenated tree (CT) codes. CT codes are special low-density parity-check (LDPC) codes consisting of several trees with large spans. They can also be regarded as special turbo codes with hybrid recursive/nonrecursive parts and multiple constituent codes. CT codes are decodable by the belief-propagation algorithm. They combine many advantages of LDPC and turbo codes, such as low decoding cost, fast convergence speed, and good performance.

**Index Terms**—Bayesian networks, graph codes, iterative decoding, low-density parity-check (LDPC) codes, multidimensional concatenated codes, Tanner graphs, turbo codes.

## I. INTRODUCTION

LOW-DENSITY parity-check (LDPC) codes [1]–[10] and turbo codes [11]–[15] are similar in many respects. They both belong to the general class of powerful concatenated codes employing pseudo-random encoders and iterative decoders. For very long interleaver lengths (say, interleaver lengths  $>10000$ ), it has been reported [8], [16] that irregular LDPC codes can outperform turbo codes. For shorter lengths, turbo codes appear to have better performance [8].

For an LDPC code, each constituent code [a single-parity-check (SPC) code] typically involves only several bits. Reliable information builds up relatively slowly during the iterative decoding process, so global convergence is slow. On the other hand, each constituent code in a turbo code (a recursive convolutional code) involves all the information bits. Decoding information spreads very fast in a turbo decoder, which leads to relatively fast convergence speed. However, a turbo decoder based on the Bahl–Cocker–Jelinek–Raviv (BCJR) algorithm [17], [18] typically requires more memory and more operations per iteration than an LDPC decoder.

It has been demonstrated that increasing the number of constituent codes is an efficient technique for low-complexity turbo-type code design [12], [15]. However, more constituent codes imply lower rate. Although heavy puncturing can be used to compensate for the rate loss, the resultant performance loss may offset the benefit. Thus the schemes considered in

[12], [15] are most suitable for relatively low rates (say, less than  $1/3$ ).

This paper is concerned with a family of concatenated tree (CT) codes, improved over those in [20]–[23]. Graphically, a CT code can be seen as a special LDPC code consisting of several trees with large spans. Locally optimal soft-in/soft-out decoding can be carried out over an entire constituent tree code (which represents a large portion of an overall CT code). Consequently, decoding convergence is fast for CT codes. For interleaver lengths less than 10 000, CT codes also demonstrate better performance than common LDPC codes. CT codes are most suitable for medium to high rates (say, greater than  $1/3$ ), since tree codes constructed from SPC codes inherently have high rates.

CT codes are also closely related to turbo codes. Some good CT codes can be seen as the turbo-type concatenation of unconventional two-state convolutional codes with both recursive and nonrecursive parts. By using multiple constituent codes and properly balancing the recursive and nonrecursive parts, the performance loss due to the use of very simple constituent codes can be recovered. CT codes can achieve performance comparable to that of standard turbo codes. However, a CT decoder using the belief-propagation algorithm is much simpler (in terms of both operations per iteration and memory usage) than a turbo decoder using the BCJR algorithm. These two types of decoders have similar convergence speeds. Consequently, the overall complexity of a CT decoder is considerably lower than that of a turbo decoder.

This paper is structured as follows. In Section II, we will derive a graphical model for the local encoder and a turbo-type global encoder for the overall CT code. Section III is concerned with parallel and serial decoding strategies for CT codes. We will present some design principles and performance comparisons for CT codes in Section IV. We will compare CT codes with turbo codes and LDPC codes in Section V.

## II. CT CODES

### A. Tanner Graphs and Tree Codes

The Tanner graph [2], [3] for a linear binary code consists of *variable nodes* and *check nodes* as in Fig. 1. A variable node, denoted by a circle, represents a bit in the codeword. A check node, denoted by a square, represents a parity-check constraint. The variable nodes connected to a given check node form an SPC code. In this paper, we assume that the code is systematic. We will use white and black circles to represent information and parity bits, respectively. Parity bits can be punctured for rate

Manuscript received December 14, 1999; revised September 1, 2000. This work was supported in part by City University of Hong Kong under Strategic Research Grant 7001044 and under grant from Motorola. The material in this paper was presented at the International Symposium on Turbo Codes, Brest, France, September 2000.

L. Ping is with the Department of Electronic Engineering, City University of Hong Kong, Kowloon, Hong Kong (e-mail: eeliping@cityu.edu.hk).

K. Y. Wu is with the National Key Lab of ISN, Xidian University, Xi'an, Shaanxi, China (e-mail: ymwang@ns2.xidian.edu.cn).

Communicated by G. D. Forney, Jr., Guest Editor.

Publisher Item Identifier S 0018-9448(01)00719-2.

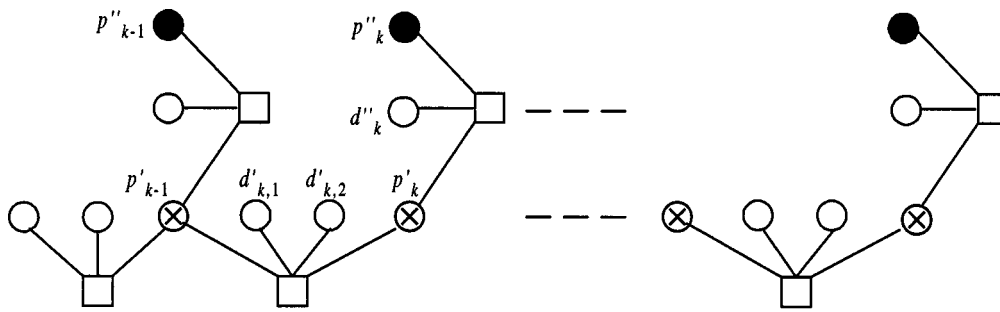


Fig. 1. The Tanner graph for a binary linear code. White, black, and crossed circles represent information, unpunctured parity, and punctured parity bits, respectively. Square blocks represent parity-check constraints.

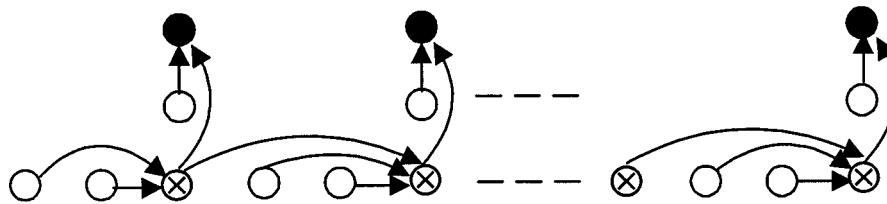


Fig. 2. The Bayesian network for the same code as in Fig. 1. White, black, and crossed circles represent information, unpunctured parity, and punctured parity bits, respectively.

adjustment, as denoted by the crossed circles “ $\otimes$ ” in Fig. 1. A punctured bit may be considered to be a hidden or state variable.

The degree of a node is the number of edges connected to it. The degree of a variable node is thus the number of parity checks involving the variable. For example, in Fig. 1, the information nodes, the unpunctured parity nodes and the punctured parity nodes have degrees 1, 1, and 3, respectively. In this paper, we only consider codes with low nodal degrees which belong to the general category of LDPC codes. If a Tanner graph does not contain any loop, like the one in Fig. 1, it is called a *tree*. The corresponding code is called a *tree code*.

**B. Bayesian Networks**

A *Bayesian network* [6] as in Fig. 2 is an alternative graphical representation of a code. It is convenient to use a Bayesian network to illustrate the encoding procedure of a tree code. Here, every node represents a bit in the code. If a node has no incoming edges (a white circle), it is an information node whose value can be independently assigned. Otherwise, the node is the parity check of the starting nodes of its incoming edges. A tree code can be easily encoded following the edge directions in the Bayesian network. In contrast, with general LDPC codes, encoding can be a complicated process.

**C. CT Codes**

Simple tree codes can be concatenated in a turbo-type manner [11], [12], [20], [21], as in Fig. 3. The parallel encoder in Fig. 3 consists of  $M$  local tree encoders. Let

$$\mathbf{D}^{(m)} = \pi^{(m)}(\mathbf{D}), \quad m = 0, 1, \dots, M - 1$$

denote  $M$  interleaved versions of an information sequence  $\mathbf{D}$ . For each  $\mathbf{D}^{(m)}$ , the  $m$ th local encoder generates a tree code defined as above with the parity sequence  $\mathbf{p}^{(m)}$ . The overall codeword is formed by  $\{\mathbf{D}, \mathbf{p}^{(0)}, \dots, \mathbf{p}^{(M-1)}\}$ . The pair  $\{\mathbf{D}, \mathbf{p}^{(m)}\}$

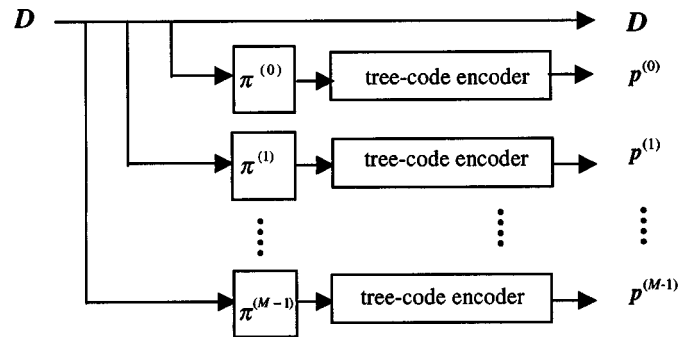


Fig. 3. A CT encoder with  $M$  constituent codes.

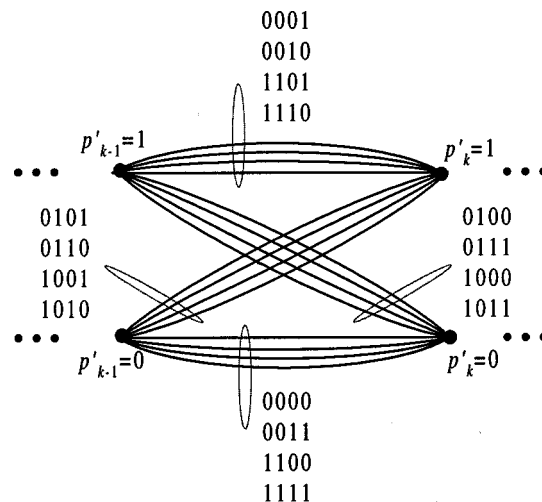


Fig. 4. The trellis representation of the tree code in Fig. 1. Here each state represents a valid value of  $p'_k$  and each branch label represents a valid value of  $\{d'_{k,1}, d'_{k,2}, d''_k, p''_k\}$ .

is referred to as the  $m$ th constituent code. The resultant code is called a CT code, which can still be seen as an LDPC code for

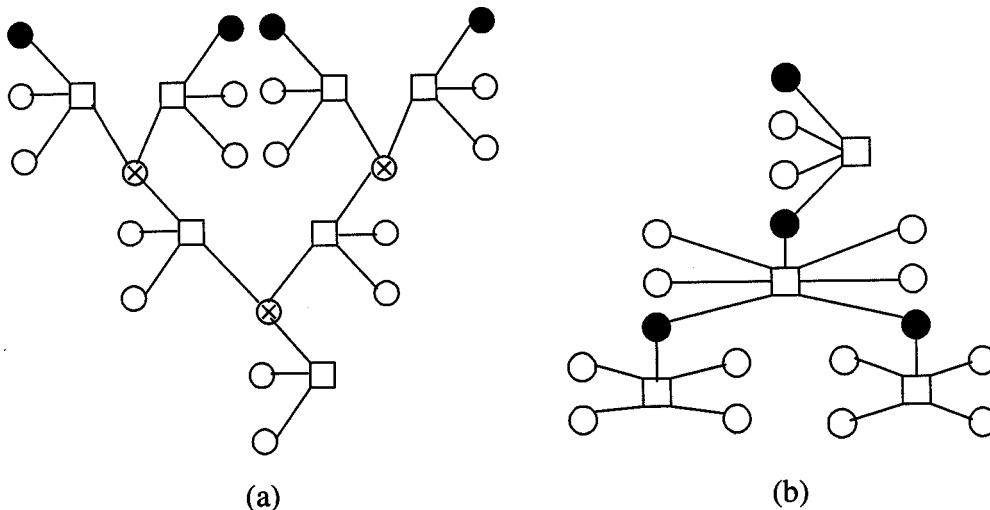


Fig. 5. Two examples of tree codes. They can both be encoded from the bottom upward. They can also be extended indefinitely following the same patterns.

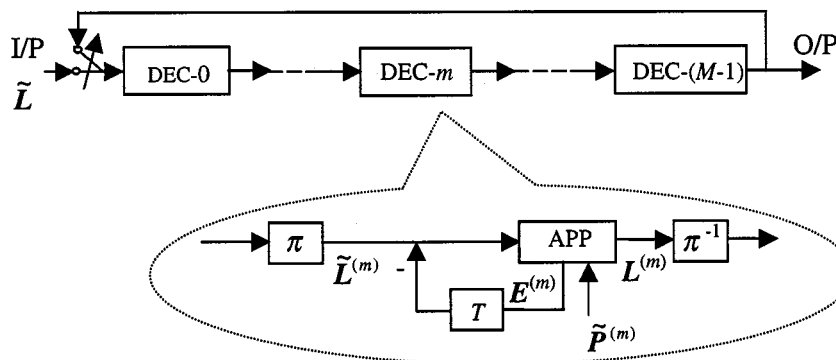


Fig. 6. The global CT decoder: “ $T$ ” denotes a delay of one iteration and “ $\pi$ ” denotes an interleaver.

small  $M$ . The degrees of the information nodes in the overall CT code are increased by  $M$  times from those in the constituent codes. In general, there will be loops in a CT code. It is known that good LDPC codes generally have loopy graphs [1]–[8].

#### D. Connection with Turbo Codes

Some tree codes have simple trellis representations. For example, the two-state trellis shown in Fig. 4 represents the code in Fig. 1. Here, the state variables at depth  $k$  are the two valid values of the punctured bit  $p'_k$ . The branch labels are the valid values of the coded bits in the segment  $\{d''_{k,1}, d''_{k,2}, d''_k, p''_k\}$ .

The trellis representation indicates a connection between some tree codes and convolutional codes. Note that the code in Fig. 4 is different from a conventional convolutional code in two respects. First, the trellis contains parallel branches. For example, let  $p'_{k-1} = 0$ , then four combinations of  $\{d''_{k,1}, d''_{k,2}, d''_k, p''_k\}$  can lead to  $p'_k = 0$ , namely

$$\{0, 0, 0, 0\}, \{0, 0, 1, 1\}, \{1, 1, 0, 0\}, \text{ and } \{1, 1, 1, 1\}.$$

Second, this particular code contains both recursive and nonrecursive parts, as will be elaborated in Section IV-A.

Many tree codes may not have simple trellis representations. Fig. 5 illustrates two such examples.

### III. BELIEF-PROPAGATION DECODING OF CT CODES

#### A. Parallel Decoder

Being Tanner graph codes, CT codes can be decoded by belief-propagation decoders [1], [3]–[6]. With a parallel realization, messages are exchanged only among neighboring nodes during each iteration [1], [4], [6]. We will show later that a parallel decoder has relatively slow convergence speed.

#### B. Serial Decoder

Due to the special structure of CT codes, a serial turbo-type decoder [20] as in Fig. 6 can be employed. It consists of  $M$  looped local decoders, each responsible for one constituent code. Between two local decoders, log-likelihood ratios (LLRs) of information bits are exchanged during decoding. Due to the tree structure, the local decoder can be implemented by the belief-propagation algorithm based on the two-way schedule [6]. It can achieve a locally optimal solution for each constituent code within each decoding and is suitable for a serial processor.

A detailed discussion of various global decoder structures for general concatenated codes can be found in [12], [24]. There may be a convergence problem for such decoders (see [12]). From our observation, this problem becomes very serious for parallel concatenated codes with  $M \geq 4$ . A damping method

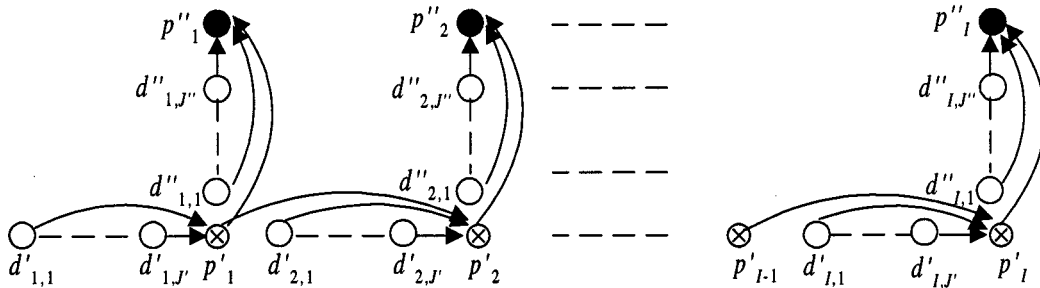


Fig. 7. The constituent tree code used in the simulation.  $J'$  ( $J''$ ) is the number of information bits involved in each horizontal (vertical) segment.

using a factor  $\alpha_1^{(m)}$  together with a simulated annealing technique have been suggested in [12] to treat this problem. However, empirically, we find that the serial decoder of Fig. 6 offers a simple as well as very robust solution without the need of damping factors.

The variables involved in Fig. 6 are

- $\tilde{\mathbf{L}}$  initial *a priori* LLR vector for the information bits;
- $\tilde{\mathbf{L}}^{(m)}$  *a priori* LLR vector for the information bits in the  $m$ th constituent code;
- $\mathbf{L}^{(m)}$  *a posteriori* LLR vector for the information bits in the  $m$ th constituent code;
- $\tilde{\mathbf{P}}^{(m)}$  *a priori* LLR vector for the parity bits in the  $m$ th constituent code;
- $\mathbf{E}^{(m)}$  extrinsic LLR vector for the  $m$ th constituent code, defined by [11]

$$\mathbf{E}^{(m)} = \mathbf{L}^{(m)} - \left( \tilde{\mathbf{L}}^{(m)} - \mathbf{E}_{\text{last iteration}}^{(m)} \right)$$

where  $\mathbf{E}_{\text{last iteration}}^{(m)}$  is the extrinsic LLR vector for the  $m$ th constituent code generated in the last iteration (with zero initial values).

At the start,  $\tilde{\mathbf{L}}$  is fed into DEC-0. The  $M$  local *a posteriori* probability (APP) decoders operate successively. The *a posteriori* LLR vector  $\mathbf{L}^{(m-1)}$  from DEC- $(m-1)$  is used, after appropriate interleaving, as the *a priori* LLR vector  $\tilde{\mathbf{L}}^{(m)}$  for DEC- $m$ .  $\mathbf{L}^{(M-1)}$  is fed back to DEC-0 for the second iteration. The process is then repeated iteratively. For the first iteration

$$\tilde{\mathbf{L}}^{(m)} = \tilde{\mathbf{L}} + \mathbf{E}^{(1)} + \dots + \mathbf{E}^{(m-1)} \quad (1)$$

and after the first iteration

$$\tilde{\mathbf{L}}^{(m)} = \tilde{\mathbf{L}} + \underbrace{\mathbf{E}^{(m)} + \mathbf{E}^{(m+1)} + \dots + \mathbf{E}^{(M-1)}}_{\text{from the previous iteration}} + \underbrace{\mathbf{E}^{(0)} + \mathbf{E}^{(1)} + \dots + \mathbf{E}^{(m-1)}}_{\text{from the current iteration}}. \quad (2)$$

Notice that  $\mathbf{E}^{(m)}$  is generated by DEC- $m$ , so it should be prevented from circulating back to DEC- $m$  again. This is realized by the subtraction of delayed  $\mathbf{E}^{(m)}$  from  $\tilde{\mathbf{L}}^{(m)}$  shown in Fig. 6.

### C. Decoding Complexity

The complexity of the belief-propagation decoding algorithm is about  $6Nt$  multiplications and  $5Nt$  additions per iteration,

where  $N$  is the total length of the code (including all the constituent codes) and  $t$  is the average degree of nodes, (see appendix). This applies (approximately) to both parallel and serial decoders. Let  $R$  be the code rate. The normalized decoding complexity is  $6Nt/NR = 6t/R$  multiplications and  $5t/R$  additions per information bit per iteration. For small  $t$  and with similar performance and convergence speeds, the complexity of a CT decoder is much lower than that of a turbo decoder; see Section V. Some additional remarks are as follows.

- The lookup table technique [18] applies equally to both turbo and CT codes.
- The decoding complexity of CT codes decreases at higher rates.
- It can be shown that the memory usage of a belief-propagation decoder is generally much lower than that of a turbo decoder.

### IV. SOME DESIGN PRINCIPLES

Except where specified otherwise, the discussion in this section is carried out based on the tree structure in Fig. 7, since it is regular and easy to analyze. The rate of the code is  $R = (J' + J'') / (J' + J'' + 1)$ . For the concatenation of  $M$  constituent codes, the overall rate is  $R = (J' + J'') / (J' + J'' + M)$ . Some other tree structures will be briefly compared at the end of this section.

We will use  $R$  for rate,  $M$  for the number of constituent codes,  $L$  for interleaver length, and  $It$  for iteration number. As explained below, the bit-error rate (BER) bound analysis for CT codes is a complicated issue that is still under investigation. Most of our discussion will be heuristic.

#### A. Recursive and Nonrecursive Parts

We divide the information nodes into *recursive* and *nonrecursive* ones. For a nonrecursive node, such as  $d''_{k,j}$  in Fig. 7, changing its value will affect a fixed number of parity nodes. On the other hand, changing the value of a recursive node, such as  $d'_{k,j}$  in Fig. 7, will affect an indefinite number of parity nodes, depending on code length. The sets of  $\{d''_{k,j}\}$  and  $\{d'_{k,j}\}$  are called the nonrecursive and recursive parts, respectively.

In terms of graph theory, the recursive nodes belong to directed paths that grow with the graph length. Along such a path, an information sequence with finite weight may produce a codeword with infinite weight (when the code length approaches infinity). Recall that a standard turbo code is the concatenation of two recursive convolutional codes. A

nonrecursive code tends to link low parity weights with low information weights. For a recursive code, such combinations are reduced. With random interleavers, the resultant codewords are more like those in random codes and, as is well known, random codes can approach the channel capacity. For CT codes, a similar principle applies. Furthermore, we find that a hybrid recursive and nonrecursive structure can lead to improved performance for CT codes, as discussed below.

### B. A Design Criterion for CT Codes

According to [11], [13], the recursive encoders play a fundamental role in turbo codes. If at least two recursive constituent codes are used which are separated by random interleavers, multiplicities of low-weight codewords are greatly reduced after concatenation. This effect is referred to as the interleaving gain. As pointed out in [13], the interleaving gain is critical to good performance of turbo codes. Take a weight-2 information sequence as an example and consider constituent encoders with only recursive parts (such as for an ordinary turbo code). The parity weight generated by such an input word is approximately proportional to the distance between the positions of the two information ones. With at least two recursive constituent codes and random interleavers, the probability that the two information ones are close to each other in all encoders is small. Based on the same principle, to achieve good performance, every information bit in a CT code should appear in the recursive part at least twice.

### C. Interleaver Design

Suppose that one random interleaver per constituent code is used for a CT code. There is a possibility that some information bits may appear in the nonrecursive part for all the constituent codes, which is not preferred according to Section IV-B. The following interleaver design can avoid such an event. For convenience, assume that  $a' = J' \times M / (J' + J'')$  and  $a'' = J'' \times M / (J' + J'')$  are both integers for the code in Fig. 7. Also assume that  $M$  divides interleaver length  $L$ . Divide the information array  $\mathbf{D}$  into  $M$  subsets of equal size

$$\mathbf{D} = \{ \mathbf{D}^{(0)}, \mathbf{D}^{(1)}, \dots, \mathbf{D}^{(M-1)} \}. \quad (3)$$

Define  $\langle x \rangle = x$  modulo  $M$ . For the  $m$ th constituent code, define

$$\mathbf{D}' = \{ \mathbf{D}^{\langle(m)\rangle}, \mathbf{D}^{\langle(m+1)\rangle}, \dots, \mathbf{D}^{\langle(m+a'-1)\rangle} \}$$

and let  $\mathbf{D}''$  be the complement of  $\mathbf{D}'$  in  $\mathbf{D}$ . Two independent random interleavers are applied to  $\mathbf{D}'$  and  $\mathbf{D}''$ , respectively, to obtain the recursive and nonrecursive parts. In this way, every bit will appear in  $\mathbf{D}'$  and  $\mathbf{D}''$  exactly  $a'$  and  $a''$  times, respectively. This interleaver design is adopted throughout the discussions below.

In [13], an elegant technique has been developed to generate the distance spectra of turbo codes based on the assumption that a ‘‘uniform’’ interleaver is used for each constituent code. However, as mentioned above, two independent interleavers should

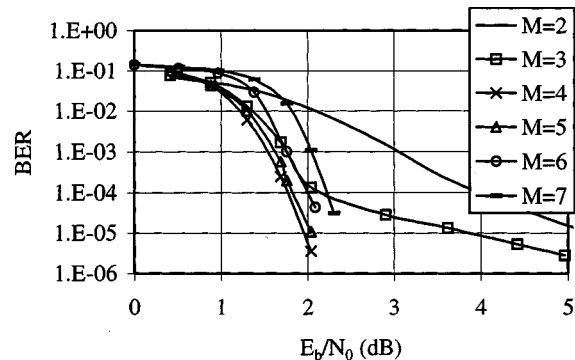


Fig. 8. Performance comparison for the CT codes based on Fig. 7. Rate  $R = 1/2$ , interleaver length  $L = 1024$ , iteration number  $It = 18$ ,  $(J', J'') = (2, 0), (3, 0), (4, 0), (5, 0), (6, 0)$ , and  $(7, 0)$  for  $M = 2, 3, 4, 5, 6$ , and  $7$ , respectively.

be used separately for the recursive and nonrecursive parts to ensure good performance for a CT code. Therefore, the formulas in [13] are not directly applicable here, although the principles are still useful.

### D. Selection of the Number of Constituent Codes

Fig. 8 examines the impact of the number of constituent tree codes on CT codes. Performance improves as the number of constituent codes increases up to  $M = 4$ , and then starts to decline. Similar observations have been made for other parameters, such as different choices of the  $(J', J'')$  pair or the interleaver length  $L$ . Notice that, after concatenating  $M$  tree codes with the structure in Fig. 7, the degree of an information node is simply  $M$ . Define  $J = J' + J''$ . With  $R = 1/2$ , we have  $M = J$ , and the average degree  $t$  of the variable nodes (including the punctured nodes and considering all the constituent codes) is

$$t = \frac{(J+4)M}{J+2M} = \frac{M+4}{3}. \quad (4)$$

For  $M = 2, 3, 4, 5, 6, 7$ , and we have  $t = 2, 2.3, 2.7, 3, 3.3$ , and  $3.7$ , respectively. The values for  $M = 4, 5$ , and  $6$  are close to the common choice of  $t = 3$  for LDPC codes [4].

We conjecture that more constituent codes may lead to a lower error floor but this is difficult to verify using simulation. We are currently trying to find bounds.

Also notice that  $t$  increases with  $M$  in (4). From the appendix, the decoding complexity is proportional to  $t$ . Therefore, more constituent codes imply higher decoding cost.

In view of the above discussion, we will fix  $M = 4$  below.

### E. Relative Sizes of Recursive and Nonrecursive Parts

We use the codes with  $R = 1/2$  and  $M = 4$  as examples. In this case,  $M = J' + J''$  and  $(J', J'')$  can be chosen as  $(4, 0), (3, 1), (2, 2), (1, 3)$ , and  $(0, 4)$ . The corresponding codes will be distinguished by the  $(J', J'')$  pair below.

Clearly, the  $(0, 4)$  and  $(1, 3)$  codes are not good choices, since every information bit appears in the recursive part fewer than two times in these two codes (see Section IV-B and Fig. 9). We

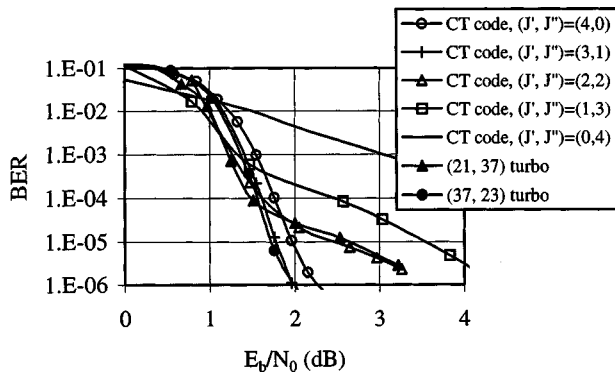


Fig. 9. Performance comparison for different CT codes with  $R = 1/2$ ,  $M = 4$ ,  $L = 1024$ , and  $It = 18$ . Performance of two punctured turbo codes with  $R = 1/2$ ,  $M = 2$ ,  $L = 1024$ , and  $It = 18$  are included as references. Their generators are  $(21, 37)$  and  $(37, 23)$ . The  $(37, 23)$  turbo code has a low error floor [25].

will only compare  $(4, 0)$ ,  $(3, 1)$ , and  $(2, 2)$  codes. For these codes, the codewords with minimum distance are all generated by weight-2 information sequences. (Note that a weight-1 information sequence will generate at least one nonzero parity bit per constituent code.) For the  $(4, 0)$  CT code, the minimum distance is 2, corresponding to the event that the two ones (more precisely, their corresponding nodes in the Tanner graph) are connected to a common check node in every constituent tree code and produce zero parity. This is referred to as the zero-parity event. Similarly, it can be shown that the  $(2, 2)$  code also has a minimum distance of 2. Notice that here we assume random interleavers. The probability of the zero-parity event is small but not zero.

For the  $(3, 1)$  CT code, every information bit will appear in the nonrecursive part once (using the interleaver design of Section IV-C). Two parity ones will be produced when two information ones are in the nonrecursive part. Thus the minimum distance of the  $(3, 1)$  code is 4. This implies an advantage compared with the  $(4, 0)$  and  $(2, 2)$  CT codes, which can be seen from the simulation results shown in Fig. 9. The performance of the  $(3, 1)$  CT code is better than that of the  $(4, 0)$  CT code by about 0.3 dB at BER equal to  $10^{-5}$ . The other choices of  $(J', J'')$  pairs clearly result in performances with worse error floors.

For the punctured  $(37, 23)$  and  $(21, 37)$  turbo codes [11], [25] compared in Fig. 9, the minimum distances are 5 and 4, respectively. This implies that for very high  $E_b/N_0$ , the  $(37, 23)$  turbo code may have an advantage. However, for LDPC/turbo-type concatenated codes, performance is also strongly affected by the multiplicities of codewords with small weights, in addition to the minimum distance. (Recall that an ideal random code can have codewords with very small weights and very small multiplicities.) A detailed discussion of this issue can be found in [13], [14] for turbo codes. From [14], increasing the number of constituent codes can further reduce the multiplicities of codewords with small weights. This explains the good performance of the CT codes that use more constituent codes than the turbo codes, although the former have lower minimum distances. As seen from Fig. 9, the  $(3, 1)$  CT code and the  $(37, 23)$  turbo code have very similar performance over a quite wide range. It is also

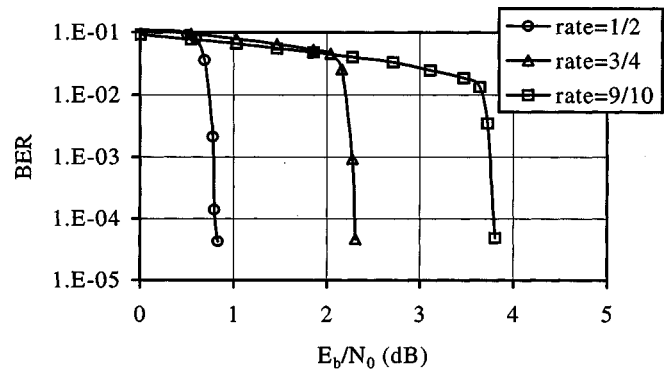


Fig. 10. Simulated performance for the CT codes with  $M = 4$ ,  $L = 65536$ ,  $It = 18$ , and  $(J', J'') = (3, 1)$  for  $R = 1/2$ ,  $(J', J'') = (9, 3)$  for  $R = 3/4$ ,  $(J', J'') = (27, 9)$  for  $R = 9/10$ . Theoretical limits:  $(R, E_b/N_0) = (1/2, 0.2 \text{ dB})$ ,  $(3/4, 1.63 \text{ dB})$ , and  $(9/10, 3.2 \text{ dB})$ .

interesting that the  $(2, 2)$  CT code and the  $(21, 37)$  turbo code have very similar performance.

For a thorough performance analysis, especially in the error floor range, a BER bound technique such as the one developed in [13], [14] for turbo codes can be employed. However, as mentioned above, the uniform interleaver assumption introduced in [13] is not applicable to CT codes using the two-interleaver technique outlined in Section IV-C. We are now investigating the issue of generating a weight-distribution function for general CT codes.

#### F. High-Rate CT Codes

Fig. 10 shows the performance of CT codes with different rates, which are all about a half decibel away from the theoretical limits. Based on the discussion in Section III-C, the decoding costs of CT codes decrease for higher rate. This is in contrast to punctured turbo codes, for which decoding complexity remains the same regardless of the puncturing rate [26]. In the limiting case of  $R \rightarrow 1$ , the complexity ratio between a CT code and a 16-state turbo code [26] is about a fraction of 10. (Note that for  $R \rightarrow 1$ ,  $t \rightarrow 4$  for a CT code with  $M = 4$  constituent codes.)

#### G. Other Tree Structures

CT codes can also be generated using more general tree structures than the one in Fig. 7. Furthermore, different structures can be used for different constituent codes. Fig. 11 includes the simulation results for some CT codes based on the tree structures of Fig. 5. Some of these are quite close to (but not better than) the best ones in Fig. 9.

### V. COMPARISONS WITH TURBO CODES AND LDPC CODES

Fig. 12 is a comparison of CT codes with turbo codes with different interleaver lengths. All the other parameters are fixed as for those in Fig. 9. We only include the  $(3, 1)$  and  $(2, 2)$  CT codes for clarity. It is seen that the performance curves of the  $(2, 2)$  and  $(3, 1)$  CT codes are, respectively, quite close to those of the  $(21, 37)$  and  $(37, 23)$  turbo codes.

For the CT codes considered, the ratio between the total numbers of edges and the total number of information nodes is eight

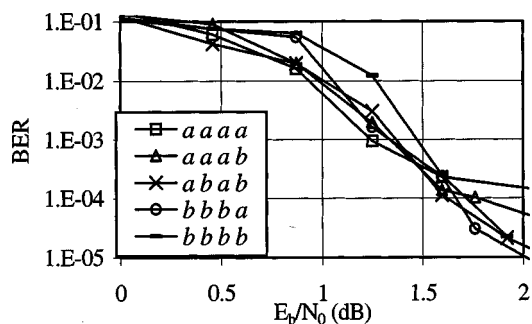


Fig. 11. Simulated performance for some CT codes based on the tree structures of Fig. 5. Here we use “a” and “b” to indicate the tree structures in Fig. 5(a) and (b), respectively. For example, “aaab” means the structure of Fig. 5(a) is used in three constituent codes and the structure of Fig. 5(b) in one constituent code, etc.  $R = 1/2$ ,  $L = 1022$ , and  $It = 18$ .

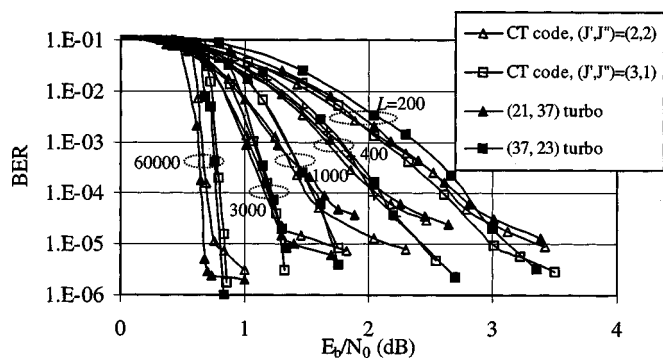


Fig. 12. Simulated performance of CT and turbo codes with  $R = 1/2$ ,  $It = 18$ ,  $L = 200, 400, 1000, 3000$ , and  $60000$ , with  $M = 4$  for the CT codes and  $M = 2$  for the turbo codes.

(using the Tanner graph, not the Bayesian network, to count degree). There is no operation required for the edges connected to the unpunctured parity nodes of degree 1. Considering this, and using the results of the appendix, the decoding cost of a CT code is about 42 multiplications and 35 additions per information bit per iteration. This is about six times lower than the decoding complexity of 16-state turbo codes. (The latter is about 256 multiplications and 192 additions per information bit per iteration, including normalization.)

Fig. 13 compares the convergence properties of the (3, 1) CT code and the (21, 37) turbo code, both with  $R = 1/2$  and  $L = 200$ . The convergence speeds of the two codes are quite similar.

Fig. 14 is a comparison between CT codes and some LDPC codes reported in the literature, including the repeat-accumulate (RA) code. An RA code consists of an inner repetition code, a random interleaver, and an outer two-state convolutional code. This is somehow similar to the CT code based on Fig. 7 with  $J'' = 0$ , except that the RA code employs a single interleaver for the whole code (which makes it easier to analyze).

The performance curves of the LDPC codes and the RA code are taken from the respective references listed below. Some details for the code structures compared in Fig. 14 are as follows.

- RA1024: the RA code in [19],  $L = 1024$ ,  $R = 1/3$ ,  $It = 30$ .

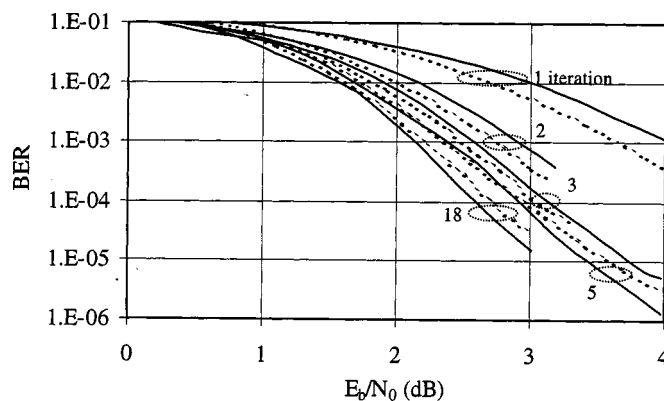


Fig. 13. Convergence comparison for CT and turbo codes with  $R = 1/2$ ,  $L = 200$  and  $It = 1, 2, 3, 5, 18$ . The parameters of the CT code are  $M = 4$  and  $(J', J'') = (3, 1)$ . The parameters of the turbo code are  $M = 2$  and the generator = (21, 37). The solid lines are for the CT code and the dashed lines are for the turbo code.

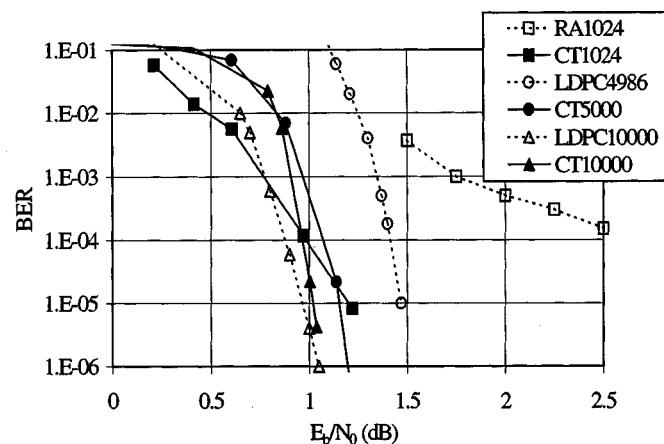


Fig. 14. Simulated performance of various codes decodable by the belief-propagation algorithm. All the CT codes have four constituent codes with the structure shown in Fig. 7.

- CT1024: a hybrid (2, 0)-(1, 1) CT code [each tree code consisting of alternative (2, 0) and (1, 1) sections],  $L = 1024$ ,  $R = 1/3$ ,  $It = 18$ .
- LDPC4986: the LDPC code in [9],  $L = 4986$ ,  $R = 1/2$ ,  $It \leq 50$ .
- CT5000: the (3, 1) CT code,  $L = 5000$ ,  $R = 1/2$ ,  $It = 18$ .
- LDPC10000: the LDPC code in [8],  $L = 10000$ ,  $R = 1/2$ .
- CT10000: the (3, 1) CT code,  $L = 10000$ ,  $R = 1/2$ ,  $It = 18$ .

From Fig. 14, it is seen that the CT codes outperform the RA code and the LDPC code for interleaver lengths of 1024 and about 5000. For an interleaver length of 10000, the LDPC code has better performance [8]. All the codes compared in Fig. 14 are decodable by the belief-propagation algorithm. However, for serial processing, the CT codes exhibit faster convergence due to the use of the two-way schedule (see Fig. 15).

The influence of different decoder structures is also an interesting issue. Fig. 15 compares the serial and parallel decoders (see Sections III-A and III-B) for the same CT code. The serial

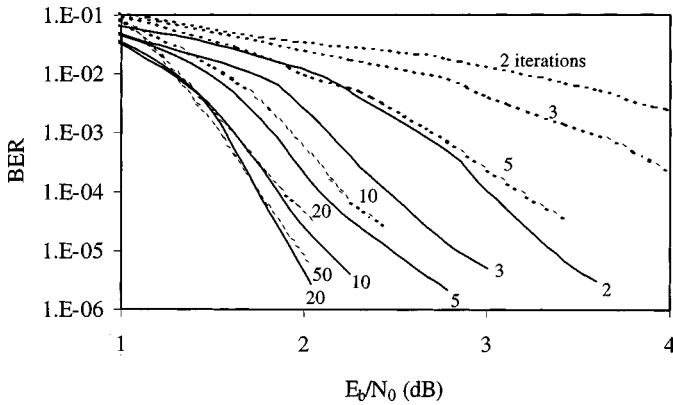


Fig. 15. Comparison of the convergence speeds for serial and parallel decoders. The same CT code (based on Fig. 7) is used with  $M = 4$ ,  $(J', J'') = (4, 0)$ , and  $L = 1024$ . The solid lines are for the serial decoder in Fig. 4, and the dashed lines for the parallel decoder based on [4].

decoder demonstrates better convergence speed as expected. This implies that, for a serial processor, the convergence speed of a general LDPC code may also be improved if it can be decomposed into several trees with large spans (so that the two-way schedule can be applied).

## VI. CONCLUSION

In summary, CT codes can be encoded straightforwardly. They are decodable by the belief-propagation algorithm like common LDPC codes, but exhibit faster convergence speed and better performance for short interleaver length. They achieve performance comparable to turbo codes, but with significantly lower decoding cost.

For further research, the capacity analysis of general CT codes is still an issue. Some tree structures have simple trellis representations (see Fig. 4), based on which their distance spectra can be generated. For codes with only a recursive part and employing one interleaver per constituent code, the uniform interleaver assumption of [13] is applicable. A BER bound analysis for this special case can be found in [23]. However, for general CT codes with both recursive and nonrecursive parts, the uniform interleaver assumption is not applicable if the interleaver design is based on the technique (two random interleavers per constituent code) outlined in Section IV-C. This complicates the problem of analyzing the weight distribution function. We will discuss this issue elsewhere.

## APPENDIX

### THE BELIEF-PROPAGATION DECODING ALGORITHM [4], [6], AND [27]

Consider a Tanner graph. Denote by  $\{c_m\}$  and  $\{v_n\}$  the check and variable nodes, respectively. Denote by  $(m, n)$  the edge between  $c_m$  and  $v_n$ . Let  $E(x)$  be the set of edges connected to a node  $x$  and let  $E(x) \setminus e \in E(x)$  excluding  $e$ . Let  $f_n^0$  and  $f_n^1$  be the *a priori* probabilities of  $v_n$  being 0 and 1, respectively, based on the observation of an individual channel output. Define  $f_n = f_n^0/f_n^1$ .

For every edge  $(m, n)$ , two messages are defined below. It is easy to verify that  $q_{m,n}^0$  and  $q_{m,n}^1$  in the following are equivalent

to the variables used in [4] bearing the same names. They are initialized as  $q_{m,n}^0 = f_n^0$  and  $q_{m,n}^1 = f_n^1$ .

**Message from a check node  $c_m$ :**

$$r_{m,n} = \frac{1 + \prod_{(m,n') \in E(c_m) \setminus (m,n)} (q_{m,n'}^0 - q_{m,n'}^1)}{1 - \prod_{(m,n') \in E(c_m) \setminus (m,n)} (q_{m,n'}^0 - q_{m,n'}^1)}. \quad (5)$$

**Message from a variable node  $v_n$ :**

$$s_{m,n} = f_n \prod_{(m',n) \in E(v_n) \setminus (m,n)} r_{m',n}. \quad (6a)$$

Also update

$$q_{m,n}^0 = \frac{s_{m,n}}{1 + s_{m,n}} \quad \text{and} \quad q_{m,n}^1 = 1 - q_{m,n}^0. \quad (6b)$$

Consider the evaluation of (5). For every  $c_m$ , we first compute

$$\Delta_m = \prod_{(m,n') \in E(c_m)} (q_{m,n'}^0 - q_{m,n'}^1). \quad (7)$$

The product in (5) can then be generated by  $\Delta_m / (q_{m,n'}^0 - q_{m,n'}^1)$  for every  $(m, n')$  in  $E(c_m)$ . With this technique, (5) costs (approximately) three additions and three multiplications per edge.

Similarly, for (6), let

$$R_n = f_n \prod_{(m',n) \in E(v_n)} r_{m',n}. \quad (8)$$

Then  $s_{m,n} = R_n / r_{m,n}$  for every  $(m, n)$  in  $E(v_n)$ . It is easy to verify that (6a) and (6b) cost three multiplications and two additions per edge. Incidentally,  $R_n$  is the *a posteriori* likelihood ratio for  $v_n$  and its logarithm is the LLR used in Section III-B.

In belief-propagation decoding, (5) and (6) are evaluated iteratively. With the parallel (flooding) schedule, one iteration consists of evaluating (5) concurrently for all the check nodes and then evaluating (6) concurrently for all the variable nodes. With the serial (two-way) schedule, (5) and (6) are evaluated alternately following a proper sequential order within an iteration, according to the tree structure. For a detailed discussion of the two methods, see [6] and [27]. In both cases, each of the two messages above is generated once for every edge. Let  $N$  be the total number of variable nodes (i.e., the code length) and  $t$  the average variable nodal degree. Based on the analysis above, the total cost (for either parallel or serial schedule) is thus  $6Nt$  multiplications and  $5Nt$  additions per iteration.

As a final note, the algorithm above is based on the manipulation of the difference between two probability values. Such an approach is sensitive to quantization. A modified method is discussed in [28] to overcome the difficulty by operating on likelihood ratios. The latter requires considerably fewer quantization bits and so lower decoding cost. The two methods are equivalent with perfect arithmetic precision and have similar operation counts.

## ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers for the detailed comments and corrections from . They would also like to acknowledge help from W. K. Leung and J. K. S. Ho.

## REFERENCES

- [1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.
- [2] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 533–547, Sept. 1981.
- [3] N. Wiberg, H. A. Loeliger, and R. Kotter, "Codes and iterative decoding on general graphs," *Euro. Trans. Telecommun.*, vol. 6, pp. 513–526, Sept. 1995.
- [4] D. J. C. McKay and R. M. Neal, "Near Shannon limit performance of low-density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.
- [5] R. J. McEliece, D. J. C. MacKay, and J. F. Cheng, "Turbo decoding as an instance of Pearl's 'belief propagation' algorithm," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 140–152, Feb. 1998.
- [6] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 219–230, Feb. 1998.
- [7] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs and belief propagation," in *Proc. IEEE Int. Symp. Information Theory (ISIT'98)*, Cambridge, MA, Aug. 1998, p. 117.
- [8] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of provably good low-density parity check codes," *IEEE Trans. Inform. Theory*, to be published.
- [9] D. J. C. MacKay, S. T. Wilson, and C. D. Matthew, "Comparison of constructions of irregular Gallager codes," *IEEE Trans. Commun.*, vol. 47, pp. 1449–1454, Oct. 1999.
- [10] I. Kanter and D. Saad, "Error-correcting codes that nearly saturate Shannon's bound," *Phys. Rev. Lett.*, vol. 83, no. 13, pp. 2660–2663, Sept. 1999.
- [11] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE Int. Communications Conf. (ICC'93)*, vol. 2, 1993, pp. 1064–1070.
- [12] D. Divsalar and F. Pollara, "Multiple turbo codes," in *Proc. IEEE MILCOM'95*, vol. 1, 1995, pp. 279–285.
- [13] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated decoding schemes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 409–428, Mar. 1996.
- [14] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Analysis, design, and iterative decoding of double serially concatenated codes with interleavers," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 231–244, Feb. 1998.
- [15] C. Berrou, M. Jézéquel, and C. Douillard, "Multidimensional turbo codes," in *IEEE Int. Workshop Information Theory*, Metsovo, Greece, June 1999.
- [16] S.-Y. Chung, G. D. Forney Jr., T. J. Richardson, and R. Urbanke, "On the design of low density parity-check codes within 0.0051 dB of the Shannon limit," *IEEE Commun. Lett.*, to be published.
- [17] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.
- [18] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. IEEE Int. Communications Conf. (ICC'95)*, June 1995, pp. 1009–1013.
- [19] D. Divsalar, H. Jin, and R. McEliece, "Coding theorems for 'turbo-like' codes," in *Proc. 1998 Allerton Conf.*, 1998, pp. 201–210.
- [20] L. Ping, S. Chan, and K. L. Yeung, "Iterative decoding of multi-dimensional concatenated single parity check codes," in *Proc. IEEE Int. Communications Conf. (ICC'98)*, June 1998, pp. 131–135.
- [21] L. Ping, "Modified turbo codes with low decoding complexity," *Electron. Lett.*, vol. 34, no. 23, pp. 2228–2229, Nov. 1998.
- [22] L. Ping, W. K. Leung, and N. Phamdo, "Low density parity check codes with semi-random parity check matrix," *IEE Electron. Lett.*, vol. 35, no. 1, pp. 38–39, Jan. 1999.
- [23] L. Ping, X. L. Huang, and N. Phamdo, "Zigzag codes and concatenated zigzag codes," *IEEE Trans. Inform. Theory*, submitted for publication.
- [24] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-input soft-output modules for the construction and distributed iterative decoding of code networks," *Euro. Trans. Telecommun.*, vol. 9, no. 2, pp. 157–172, Mar.–Apr. 1998.
- [25] S. Benedetto, R. Garelo, and G. Montorsi, "A search for good convolutional codes to be used in the construction of turbo codes," *IEEE Trans. Commun.*, vol. 46, pp. 1101–1105, Sept. 1998.
- [26] Ö. F. Açıkel and W. E. Ryan, "Punctured turbo-codes for BPSK/QPSK channels," *IEEE Trans. Commun.*, vol. 47, pp. 1315–1323, Sept. 1999.
- [27] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, pp. 498–519, Feb. 2001.
- [28] L. Ping and W. K. Leung, "Decoding low density parity check codes with finite quantization bits," *IEEE Commun. Lett.*, vol. 4, pp. 62–64, Feb. 2000.