Turbo Decoding as an Instance of Pearl's "Belief Propagation" Algorithm

Robert J. McEliece, *Fellow, IEEE,* David J. C. MacKay, and Jung-Fu Cheng

***Abstract*—In this paper, we will describe the close connection between the now celebrated iterative turbo decoding algorithm of Berrou *et al.* and an algorithm that has been well known in the artificial intelligence community for a decade, but which is relatively unknown to information theorists: Pearl's *belief propagation* algorithm. We shall see that if Pearl's algorithm is applied to the "belief network" of a parallel concatenation of two or more codes, the turbo decoding algorithm immediately results. Unfortunately, however, this belief diagram has loops, and Pearl only proved that his algorithm works when there are no loops, so an explanation of the excellent experimental performance of turbo decoding is still lacking. However, we shall also show that Pearl's algorithm can be used to routinely derive previously known iterative, but suboptimal, decoding algorithms for a number of other error-control systems, including Gallager's low-density parity-check codes, serially concatenated codes, and product codes. Thus, belief propagation provides a very attractive general methodology for devising low-complexity iterative decoding algorithms for hybrid coded systems.**

***Index Terms*—Belief propagation, error-correcting codes, iterative decoding, Pearl's Algorithm, probabilistic inference, turbo codes.**

I. Introduction and Summary

TURBO codes, which were introduced in 1993 by Berrou *et al.* [10], are the most exciting and potentially important development in coding theory in many years. Many of the structural properties of turbo codes have now been put on a firm theoretical footing [7], [18], [20], [21], [27], [45], and several innovative variations on the turbo theme have appeared [5], [8], [9], [12], [27], [48].

What is still lacking, however, is a satisfactory theoretical explanation of why the turbo decoding algorithm performs as well as it does. While we cannot yet announce a solution to this problem, we believe that the answer may come from a close study of *Pearl's belief propagation algorithm*, which is largely unknown to information theorists, but well known in the artificial intelligence community. (The first mention of belief propagation in a communications paper, and indeed the paper that motivated this one, is that of MacKay and Neal [37]. See also [38] and [39].)

In this paper, we will review the turbo decoding algorithm as originally expounded by Berrou *et al.* [10], but which was perhaps explained more lucidly in [3], [18], or [50]. We will then describe Pearl's algorithm, first in its natural "AI" setting, and then show that if it is applied to the "belief network" of a turbo code, the turbo decoding algorithm immediately results. Unfortunately, however, this belief network has loops, and Pearl's algorithm only gives exact answers when there are no loops, so the existing body of knowledge about Pearl's algorithm does not solve the central problem of turbo decoding. Still, it is interesting and suggestive that Pearl's algorithm yields the turbo decoding algorithm so easily. Furthermore, we shall show that Pearl's algorithm can also be used to derive effective iterative decoding algorithms for a number of other error-control systems, including Gallager's low-density parity-check codes, the recently introduced low-density generator matrix codes, serially concatenated codes, and product codes. Some of these "BP" decoding algorithms agree with the ones previously derived by ad hoc methods, and some are new, but all prove to be remarkably effective. In short, belief propagation provides an attractive general method for devising low-complexity iterative decoding algorithms for hybrid coded systems. This is the message of the paper. (A similar message is given in the paper by Kschischang and Frey [33] in this issue.)

Here is an outline of the paper. In Section II, we derive some simple but important results about, and introduce some compact notation for, "optimal symbol decision" decoding algorithms. In Section III, we define what we mean by a turbo code, and review the turbo decoding algorithm. Our definitions are deliberately more general than what has previously appeared in the literature. In particular, our transmitted information is not binary, but rather comes from a q-ary alphabet, which means that we must deal with q-ary probability distributions instead of the traditional "log-likelihood ratios." Furthermore, the reader may be surprised to find no discussion of "interleavers," which are an essential component of all turbo-coding systems. This is because, as we will articulate fully in our concluding remarks, we believe that the interleaver's contribution is to make the turbo code a "good" code, but it has nothing directly to do with the fact that the turbo decoding algorithm is a good approximation to an optimal decoder. In Section IV, we change gears, and give a tutorial overview of the general probabilistic inference problem, with special reference to *Bayesian belief networks*. In Section V,

Manuscript received September 27, 1996; revised May 3, 1997. This work was supported by NSF Grant NCR-9505975, AFOSR Grant 5F49620-97-1-0313, and a grant from Qualcomm, Inc. A portion of R. J. McEliece's contribution was done while he was visiting the Sony Corporation in Tokyo. The collaboration between D. J. C. MacKay and R. J. McEliece was begun at, and partially supported by, the Newton Institute for Mathematical Sciences, Cambridge, U.K.

R. J. McEliece is with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA.

D. J. C. MacKay is with the Cavendish Laboratory, Department of Physics, Darwin College, Cambridge University, Cambridge CB3 OHE U.K.

J.-F. Cheng is with Salomon Brothers Inc., New York, NY 10048 USA.

Publisher Item Identifier S 0733-8716(98)00170-X.

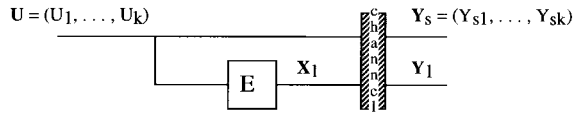


Fig. 1. Codeword $\boldsymbol{X} = (\boldsymbol{U}, \boldsymbol{X}_1)$ is transmitted over a memoryless channel and received as $\boldsymbol{Y} = (\boldsymbol{Y}_s, \boldsymbol{Y}_1)$.

we describe Pearl's BP algorithm, which can be defined on any belief network, and which gives an exact solution to the probabilistic inference problem when the belief network has no loops. In Section VI, we show that the turbo decoding algorithm follows from a routine application of Pearl's algorithm to the appropriate (loopy) belief network. In Section VII, we briefly sketch some other decoding algorithms that can be derived from BP considerations. Finally, in Section VIII, we summarize our findings and venture some conclusions.

II. Preliminaries

In this section, we will describe a general class of q-ary systematic encoders, and derive the optimal *symbol-by-symbol* decoding rule for a memoryless channel.

Let $\boldsymbol{U} = (U_1, \cdots, U_k)$ be a k-dimensional random vector of independent, but not necessarily equiprobable, symbols from a q-letter alphabet A, with $\Pr\{U_i = a\} = \pi_i(a)$, for $a \in A$. The vector $\boldsymbol{U}$ represents information to be transmitted reliably over an unreliable channel. We suppose that $\boldsymbol{U}$ is *encoded systematically*, i.e., mapped into a codeword $\boldsymbol{X}$ of the form

$$\boldsymbol{X} = (\boldsymbol{U}, \boldsymbol{X}_1) \tag{2.1}$$

where $\boldsymbol{U}$ is the "systematic" part and $\boldsymbol{X}_1$ is the "nonsystematic" part of the codeword $\boldsymbol{X}$. In the rest of the paper, we will sometimes call $\boldsymbol{X}_1$ a *codeword fragment*.

We assume that the codeword $\boldsymbol{X}$ is transmitted over a noisy channel with transition probabilities $p(\boldsymbol{y}|\boldsymbol{x}) \stackrel{\text{def}}{=} \Pr\{\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x}\}$, and received as $\boldsymbol{Y} = (\boldsymbol{Y}_s, \boldsymbol{Y}_1)$, where $\boldsymbol{Y}_s$ is the portion of $\boldsymbol{Y}$ corresponding to the systematic part of the codeword $\boldsymbol{U}$, and $\boldsymbol{Y}_1$ is the portion corresponding to the codeword fragment $\boldsymbol{X}_1$. We assume further that the channel is memoryless, which implies that the conditional density factors according to the rule

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{x}) &= p(\boldsymbol{y}_s, \boldsymbol{y}_1|\boldsymbol{u}, \boldsymbol{x}_1) \\ &= p(\boldsymbol{y}_s|\boldsymbol{u}) p(\boldsymbol{y}_1|\boldsymbol{x}_1) \qquad (2.2) \\ &= \left(\prod_{i=1}^{k} p(y_{si}|u_i) \right) \cdot p(\boldsymbol{y}_1|\boldsymbol{x}_1) \qquad (2.3) \end{aligned}$$

where y_{si} denotes the ith component of $\boldsymbol{y}_s$. The situation is depicted in Fig. 1.

The *decoding problem* is to "infer" the values of the hidden variables U_i based on the "evidence," viz. the observed values $\boldsymbol{y}_s$ and $\boldsymbol{y}_1$ of the variables $\boldsymbol{Y}_s$ and $\boldsymbol{Y}_1$. The *optimal decision*, i.e., the one that minimizes the probability of inferring an incorrect value for U_i, is the one based on the conditional probability, or "belief," that the information symbol in question has a given value

$$\mathrm{BEL}_i(a) \stackrel{\text{def}}{=} \Pr\{U_i = a|\boldsymbol{Y}_s = \boldsymbol{y}_s, \boldsymbol{Y}_1 = \boldsymbol{y}_1\}. \tag{2.4}$$

(A communication theorist would use the term "*a posteriori* probability," rather than "belief.") If a_0 is such that $\mathrm{BEL}_i(a_0) > \mathrm{BEL}_i(a)$, for all $a \neq a_0$, the decoder infers that $U_i = a_0$. The following straightforward computation is central to our results. In this computation, and for the rest of the paper, we will use Pearl's α *notation* [44].

Definition 2.1: If $\boldsymbol{x} = (x_1, \cdots, x_m)$ and $\boldsymbol{y} = (y_1, \cdots, y_m)$ are vectors of nonnegative real numbers, the notation

$$\boldsymbol{x} = \alpha \boldsymbol{y}$$

means that $x_i = y_i / (\Sigma_{k=1}^{m} \; y_k)$, for $i = 1, \cdots, m$. In other words, $\boldsymbol{x}$ is a probability vector whose components are proportional to those of $\boldsymbol{y}$. (If $f(x)$ and $g(x)$ are nonnegative real-valued functions defined on a finite set, the notation $f(x) = \alpha g(x)$ is defined similarly.)

Lemma 2.2: If the likelihood $p(y_{si}|u_i)$[1] is denoted by $\lambda_i(u_i)$, then the belief $\mathrm{BEL}_i(a)$ defined in (2.4) is given by

$$\begin{aligned} \mathrm{BEL}_i(a) &= \alpha \sum_{\boldsymbol{u}:\, u_i = a} p(\boldsymbol{y}_1|\boldsymbol{x}_1) \prod_{j=1}^{k} \lambda_j(u_j) \pi_j(u_j) \\ &= \alpha \lambda_i(a) \pi_i(a) \sum_{\boldsymbol{u}:\, u_i = a} p(\boldsymbol{y}_1|\boldsymbol{x}_1) \prod_{\substack{j=1 \\ j \neq i}}^{k} \lambda_j(u_j) \pi_j(u_j). \end{aligned} \tag{2.5}$$

Proof: We have, by the definition (2.4), $\mathrm{BEL}_i(a) = \Pr\{U_i = a|\boldsymbol{Y} = \boldsymbol{y}\}$. Then

$$\begin{aligned} &\Pr\{U_i = a|\boldsymbol{Y} = \boldsymbol{y}\} \\ &\quad = \frac{\Pr\{\boldsymbol{Y} = \boldsymbol{y}, U_i = a\}}{\Pr\{\boldsymbol{Y} = \boldsymbol{y}\}} \\ &\quad = \alpha \Pr\{\boldsymbol{Y} = \boldsymbol{y}, U_i = a\} \quad \text{(using the } \alpha \text{ notation)} \\ &\quad = \alpha \sum_{\boldsymbol{u}:\, u_i = a} p(\boldsymbol{y}, \boldsymbol{u}) \\ &\quad = \alpha \sum_{\boldsymbol{u}:\, u_i = a} p(\boldsymbol{y}|\boldsymbol{u}) \cdot p(\boldsymbol{u}) \\ &\quad = \alpha \sum_{\boldsymbol{u}:\, u_i = a} p(\boldsymbol{y}_1|\boldsymbol{x}_1) p(\boldsymbol{y}_s|\boldsymbol{u}) \cdot \prod_{j=1}^{k} \pi_j(u_j) \quad \text{by (2.2)} \\ &\quad = \alpha \sum_{\boldsymbol{u}:\, u_i = a} p(\boldsymbol{y}_1|\boldsymbol{x}_1) \cdot \prod_{j=1}^{k} \lambda_j(u_j) \pi_j(u_j) \quad \text{by (2.3)} \\ &\quad = \alpha \lambda_i(a) \pi_i(a) \sum_{\boldsymbol{u}:\, u_i = a} p(\boldsymbol{y}_1|\boldsymbol{x}_1) \prod_{\substack{j=1 \\ j \neq i}}^{k} \lambda_j(u_j) \pi_j(u_j). \end{aligned}$$

The last two lines of the above calculation are the assertions of the lemma. ∎

We see from (2.5) that $\mathrm{BEL}_i(a)$ is the product of three terms. The first term, $\lambda_i(a)$, might be called the *systematic evidence* term. The second term, $\pi_i(a)$, takes into account the *a priori* distribution of U_i. Note that the effect of the systematic evidence is, in effect, to change the prior distribution of U_i from $\pi_i(a)$ to $\alpha \pi_i(a) \lambda_i(a)$. The third term, which is more

[1] If the encoder is not systematic, i.e., if the uncoded information symbols U_i are not transmitted, these likelihoods should all be set equal to one.

TABLE I
UPDATE RULES FOR PEARL'S ALGORITHM (HERE, $\langle \boldsymbol{v} \rangle = v_1 v_2 \cdots v_n$, IF $\boldsymbol{v} = (v_1, \cdots, v_n)$ IS A VECTOR OF REAL NUMBERS)

	quantity (at X)	Type	Update Rule
1.	$\mu_X(\boldsymbol{u})$	$A_{U_1} \times \cdots \times A_{U_M} \to R^M$	$< \boldsymbol{\pi}_{\boldsymbol{U},X}(\boldsymbol{u}) >$ (1 if X has no parents)
2.	$\lambda_X(x)$	$A_X \to R$	$< \boldsymbol{\lambda}_{\boldsymbol{Y},X}(x) >$ (1 if X has no children)
3.	$\pi_X(x)$	$A_X \to R$	$\sum_{\boldsymbol{u}} p(x\|\boldsymbol{u})\mu_X(\boldsymbol{u})$ ($p(x)$ if X has no parents)
4.	$\gamma_X(\boldsymbol{u})$	$A_{U_1} \times \cdots \times A_{U_M} \to R$	$\sum_x \lambda_X(x)p(x\|\boldsymbol{u})$
5.	$\mathrm{BEL}_X(x)$	$A_X \to R$	$\alpha \cdot \lambda_X(x)\pi_X(x)$
6.	$\boldsymbol{\lambda}_{X,\boldsymbol{U}}(\boldsymbol{u})$	$A_{U_1} \times \cdots \times A_{U_M} \to R^M$	$\boldsymbol{\pi}_{\boldsymbol{U},X} \circ \gamma_X$
7.	$\pi_{X,Y_j}(x)$	$A_X \to R^N$	$\pi_X(x) \cdot \prod_{\substack{i=1 \\ i \neq j}}^{N} \lambda_{Y_i,X}(x)$

complicated, takes into account the geometry of the code. Following [10], we will call this term the *extrinsic* term, and denote it by $E_i(a)$. The extrinsic term is so important to what follows that we shall introduce a special notation for it. (This notation will also prove useful in Section V, where we shall use it to describe Pearl's algorithm—see Table I, line 6.)

Thus, let $A_1, \cdots, A_k$ be finite alphabets, let $\boldsymbol{U} \subseteq A_1 \times \cdots \times A_k$, and let R denote the set of real numbers. Let $\boldsymbol{g} = (g_1, \cdots, g_k)$ be a function mapping $\boldsymbol{U}$ into R^k. In other words, $\boldsymbol{g}$ is a vector of k real-valued functions, and if $\boldsymbol{u} = (u_1, \cdots, u_k) \in \boldsymbol{U}$, then

$$\boldsymbol{g}(\boldsymbol{u}) = (g_1(u_1), \cdots, g_k(u_k)).$$

Now, suppose that $K(\boldsymbol{u})$ is a real-valued function defined on the set $\boldsymbol{U}$, which we call a *kernel*. The K *transform of* $\boldsymbol{g}$ is the vector $\boldsymbol{g}' = (g_1', \cdots, g_k')$, where g_i' is defined by

$$g_i'(a) = \sum_{\boldsymbol{u}:\, u_i = a} K(\boldsymbol{u}) \prod_{\substack{j=1 \\ j \neq i}}^{k} g_j(u_j). \tag{2.6}$$

We summarize (2.6) by writing

$$\boldsymbol{g}' = \boldsymbol{g} \circ K. \tag{2.7}$$

Next, if $\boldsymbol{f}$ and $\boldsymbol{g}$ are vector-valued functions as above, we define their *adjacent product* $\boldsymbol{h} = \boldsymbol{fg}$ as a simple componentwise product, i.e., $\boldsymbol{h} = (h_1, \cdots, h_k)$, where

$$h_i(a) = f_i(a)g_i(a). \tag{2.8}$$

Using the circle and adjacent notation,[2] we can express the result of Lemma 2.2 compactly. To do so, we take $\boldsymbol{U} = A^k$, and define a kernel $p(\boldsymbol{u})$ as

$$p(\boldsymbol{u}) \stackrel{\text{def}}{=} p(\boldsymbol{y}_1|\boldsymbol{x}_1)$$

where the codeword fragment $\boldsymbol{x}_1 = \boldsymbol{x}_1(\boldsymbol{u})$ is a deterministic function of $\boldsymbol{u}$. Then Lemma 2.2 can be summarized as follows:

$$\mathbf{BEL} = \alpha \boldsymbol{\lambda}\boldsymbol{\pi}(\boldsymbol{\lambda}\boldsymbol{\pi} \circ p) \tag{2.9}$$

where $\boldsymbol{\lambda}(\boldsymbol{u}) = (\lambda_1(u_1), \cdots, \lambda_k(u_k))$ and $\boldsymbol{\pi}(\boldsymbol{u}) = (\pi_1(u_1), \cdots, \pi_k(u_k))$.

III. SYSTEMATIC PARALLEL CONCATENATED (TURBO) CODES

In this section, we will define what we mean by a turbo code, and present a general version of the turbo decoding algorithm.

With the same setup as in Section II, suppose we have two systematic encodings of $\boldsymbol{U}$

$$\mathcal{C}_1 \colon \boldsymbol{U} \to (\boldsymbol{U}, \boldsymbol{X}_1)$$
$$\mathcal{C}_2 \colon \boldsymbol{U} \to (\boldsymbol{U}, \boldsymbol{X}_2).$$

One way to combine $\mathcal{C}_1$ and $\mathcal{C}_2$ into a single code is via the mapping

$$\mathcal{C} \colon \boldsymbol{U} \to \boldsymbol{X} = (\boldsymbol{U}, \boldsymbol{X}_1, \boldsymbol{X}_2)$$

which is called the *parallel concatenation* of $\mathcal{C}_1$ and $\mathcal{C}_2$, or the *turbo code* formed by combining $\mathcal{C}_1$ and $\mathcal{C}_2$.

Once again, we assume that the codeword $\boldsymbol{X}$ is transmitted through a noisy channel with transition probabilities $p(\boldsymbol{y}|\boldsymbol{x})$. It is received as $\boldsymbol{Y} = (\boldsymbol{Y}_s, \boldsymbol{Y}_1, \boldsymbol{Y}_2)$, where $\boldsymbol{Y}_s$ is the component of $\boldsymbol{Y}$ corresponding to $\boldsymbol{U}, \boldsymbol{Y}_1$ is the component of $\boldsymbol{Y}$ corresponding to $\boldsymbol{X}_1$, and $\boldsymbol{Y}_2$ is the component of $\boldsymbol{Y}$ corresponding to $\boldsymbol{X}_2$. We assume again that the channel is memoryless, which implies that the conditional density factors according to the rule

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{x}) &= p(\boldsymbol{y}_s, \boldsymbol{y}_1, \boldsymbol{y}_2|\boldsymbol{u}, \boldsymbol{x}_1, \boldsymbol{x}_2) \\ &= p(\boldsymbol{y}_s|\boldsymbol{u})p(\boldsymbol{y}_1|\boldsymbol{x}_1)p(\boldsymbol{y}_2|\boldsymbol{x}_2) \qquad (3.1) \\ &= \left(\prod_{i=1}^{k} p(y_{si}|u_i) \right) p(\boldsymbol{y}_1|\boldsymbol{x}_1)p(\boldsymbol{y}_2|\boldsymbol{x}_2). \qquad (3.2) \end{aligned}$$

The situation is as depicted in Fig. 2.

[2] We assume that "adjacent" takes precedence over "circle" in order to minimize the use of parentheses.

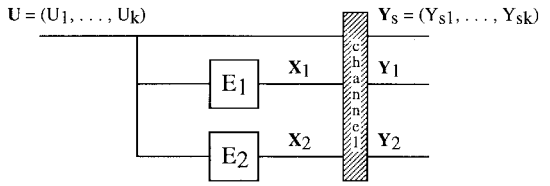


Fig. 2. Generic "turbo code." The codeword $\boldsymbol{X} = (\boldsymbol{U}, \boldsymbol{X}_1, \boldsymbol{X}_2)$ is transmitted over a memoryless channel and received as $\boldsymbol{Y} = (\boldsymbol{Y}_s, \boldsymbol{Y}_1, \boldsymbol{Y}_1)$.

By Lemma 2.2, the optimal decisions for the turbo code are based on the beliefs

$$\begin{aligned}\mathrm{BEL}_i(a) &= \alpha \sum_{\boldsymbol{u}: u_i = a} p(\boldsymbol{y}_1|\boldsymbol{x}_1) p(\boldsymbol{y}_2|\boldsymbol{x}_2) \prod_{j=1}^{k} \lambda_j(u_j)\pi_j(u_j) \\ &= \alpha\lambda_i(a)\pi_i(a) \sum_{\boldsymbol{u}: u_i = a} p(\boldsymbol{y}_1|\boldsymbol{x}_1) \\ &\quad \cdot p(\boldsymbol{y}_2|\boldsymbol{x}_2) \prod_{\substack{j=1 \\ j\neq i}}^{k} \lambda_j(u_j)\pi_j(u_j). \end{aligned} \tag{3.3}$$

For simplicity, and in accordance with engineering practice, from now on we will assume that the *a priori* probability density of the U_i's is uniform, i.e., $\boldsymbol{\pi} = (\alpha\mathbf{1}, \cdots, \alpha\mathbf{1})$. With this assumption, using the notation introduced in Section II, (3.3) becomes[3]

$$\mathbf{BEL} = \alpha\boldsymbol{\lambda}(\boldsymbol{\lambda} \circ p_1 p_2) \tag{3.4}$$

where the kernels p_1 and p_2 are defined by

$$\begin{aligned} p_1(\boldsymbol{u}) &= p(\boldsymbol{y}_1|\boldsymbol{x}_1) \\ p_2(\boldsymbol{u}) &= p(\boldsymbol{y}_2|\boldsymbol{x}_2). \end{aligned} \tag{3.5}$$

The celebrated "turbo decoding algorithm" [10], [50], [3] is an iterative approximation to the optimal beliefs in (3.3) or (3.4), whose performance, while demonstrably suboptimal [41], has nevertheless proved to be "nearly optimal" in an impressive array of experiments. The heart of the turbo algorithm is an iteratively defined sequence $\boldsymbol{\pi}^{(m)}$ of product probability densities on A^k defined by

$$\boldsymbol{\pi}^{(0)} = (\alpha\mathbf{1}, \cdots, \alpha\mathbf{1}) \tag{3.6}$$

i.e., $\boldsymbol{\pi}^{(0)}$ is a list of k uniform densities on A, and for $m \geq 1$

$$\boldsymbol{\pi}^{(m)} = \begin{cases} \alpha\boldsymbol{\lambda}\boldsymbol{\pi}^{(m-1)} \circ p_1, & \text{if } m \text{ is odd} \\ \alpha\boldsymbol{\lambda}\boldsymbol{\pi}^{(m-1)} \circ p_2, & \text{if } m \text{ is even.} \end{cases} \tag{3.7}$$

Then the *mth turbo belief vector* is defined by

$$\mathbf{BEL}^{(m)} = \alpha\boldsymbol{\lambda}\boldsymbol{\pi}^{(m)}\boldsymbol{\pi}^{(m-1)}. \tag{3.8}$$

The general form of (3.7) is shown in Fig. 3.

In a "practical" decoder, the decision about the information bits is usually made after a fixed number of iterations. (The hope that the limit of (3.8) will exist is, in general, a vain one since, in [41], several examples of nonconvergence are given.) If the decision is made after m iterations, the mth *turbo decision* is defined as

$$\hat{U}_i^{(m)} = \arg \max_{a \in A} \mathrm{BEL}_i^{(m)}(a). \tag{3.9}$$

[3] As we observed earlier, the effect of $\boldsymbol{\lambda}$ is to change the prior distribution from $\boldsymbol{\pi}$ to $\boldsymbol{\lambda}\boldsymbol{\pi}$. It follows that if there is a nonuniform prior $\boldsymbol{\pi}$, it can be accounted for by replacing every occurrence of "$\boldsymbol{\lambda}$" in our formulas with $\boldsymbol{\lambda}\boldsymbol{\pi}$.

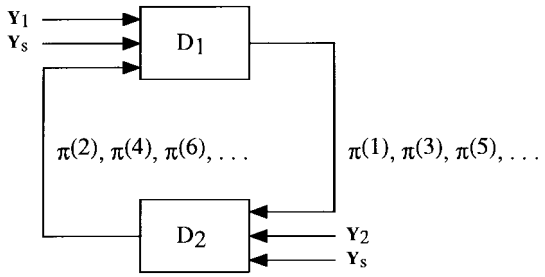


Fig. 3. Block diagram of turbo decoding procedure.

We conclude this section by observing that, as we have stated it, the turbo algorithm [(3.7) and (3.9)] does not appear to be significantly simpler than the optimal algorithm (3.4) since (for example) $(\boldsymbol{\lambda} \circ p_1)$ is not, in general, much easier to compute than $(\boldsymbol{\lambda} \circ p_1 p_2)$. The following theorem, and the discussion that follows, shed light on this problem.

Theorem 3.1: If the components of $\boldsymbol{U}$ are assumed to be independent, with $\Pr\{U_i = u_i\} = \pi_i^{(m-1)}(u_i)$, for $i = 1, \cdots, k$, then

$$\begin{aligned} \pi_i^{(m)}(a) &= \alpha \frac{\Pr\{U_i = a|\boldsymbol{Y}_s, \boldsymbol{Y}_1\}}{\lambda_i(a)\pi_i^{(m-1)}(a)}, \quad \text{if } m \text{ is odd} \\ &= \alpha \frac{\Pr\{U_i = a|\boldsymbol{Y}_s, \boldsymbol{Y}_2\}}{\lambda_i(a)\pi_i^{(m-1)}(a)}, \quad \text{if } m \text{ is even.} \end{aligned} \tag{3.10}$$

Proof: We consider the case m odd, the proof for even m being essentially the same. By reasoning similar to that in Lemma 2.2, we find that

$$\begin{aligned} &\Pr\{U_i = a|\boldsymbol{Y}_s, \boldsymbol{Y}_1\} \\ &\quad = \sum_{\boldsymbol{u}: u_i = a} p(\boldsymbol{y}_1|\boldsymbol{u}) \prod_{j=1}^{k} \lambda_j(u_j)\pi_j^{(m-1)}(u_j). \end{aligned} \tag{3.11}$$

If we divide both sides of (3.11) by $\lambda_i(a)\pi_i^{(m-1)}(a)$, we obtain

$$\begin{aligned} \frac{\Pr\{U_i = a|\boldsymbol{Y}_s, \boldsymbol{Y}_1\}}{\lambda_i(a)\pi_i^{(m-1)}(a)} &= \sum_{\boldsymbol{u}: u_i = a} p(\boldsymbol{y}_1|\boldsymbol{u}) \prod_{\substack{j=1 \\ j\neq i}}^{k} \lambda_j(u_j)\pi_j^{(m-1)}(u_j) \\ &= \boldsymbol{\lambda}\boldsymbol{\pi}^{(m-1)} \circ p_1. \end{aligned} \tag{3.12}$$

Since by (3.7), $\boldsymbol{\pi}^{(m)} = \alpha\boldsymbol{\lambda}\boldsymbol{\pi}^{(m-1)} \circ p_1$, the theorem follows. ■

The significance of Theorem 3.1 is that it tells us that the appropriate components of the vectors $\boldsymbol{\pi}^{(m)}$ can be computed by a decoder for $\mathcal{C}_1$ (or $\mathcal{C}_2$) which is capable of computing the probabilities $\Pr\{U_i = a|\boldsymbol{Y}_s, \boldsymbol{Y}_1\}$, based on an observation of the noisy codeword $\boldsymbol{Y} = (\boldsymbol{Y}_s, \boldsymbol{Y}_1)$, i.e., an optimal "soft" symbol decision decoder. The ith component of the message passed to the second decoder module is then

$$\pi_i^{(m)}(a) = \frac{\Pr\{U_i = a|\boldsymbol{Y}_s, \boldsymbol{Y}_1\}}{\lambda_i(a)\pi_i^{(m-1)}(a)} \tag{3.13}$$

which is the "extrinsic information" referred to earlier.

One of the keys to the success of turbo codes is to use component codes $\mathcal{C}_1$ and $\mathcal{C}_2$ for which a *low-complexity* soft bit decision algorithm exists. For example, the BCJR or "APP" decoding algorithm [4] provides such an algorithm for any code, block or convolutional, that can be represented by a trellis.[4]

As far as is known, a code with a low-complexity optimal decoding algorithm cannot achieve high performance, which means that individually, the codes $\mathcal{C}_1$ and $\mathcal{C}_2$ must be relatively weak. The brilliant innovation of Berrou *et al.* [10] was to devise a code of the type shown in Fig. 2, in which the individual codes $\mathcal{C}_1$ and $\mathcal{C}_2$ are indeed relatively weak (but have a low-complexity decoding algorithm), in such a way that that the overall code is very powerful. Roughly speaking, they accomplished this by making the encoder E_2 identical to E_1, except for a random permutation (accomplished by the "interleaver") of the inputs. (The encoders were short-constraint-length systematic convolutional encoders with feedback.) However, since it is the object of this paper to study the decoding algorithm without regard to the resulting performance, we shall not discuss the constructive aspect of turbo codes further.

IV. Background on Probabilistic Inference, Bayesian Belief Networks, and Pearl's Algorithm

In this section, we will give a tutorial overview of the so-called *probabilistic inference problem* of the artificial intelligence community, as well as a brief discussion of Pearl's algorithm, which solves the probabilistic inference problem in many important special cases.

Thus, let $\boldsymbol{X} = \{X_1, X_2, \cdots, X_N\}$[5] be a set of N discrete random variables, where X_i assumes values in the finite alphabet A_i. The joint density function

$$p(\boldsymbol{x}) = p(x_1, x_2, \cdots, x_N) \stackrel{\text{def}}{=} \Pr\{X_1 = x_1, \cdots, X_N = x_N\}$$

is then a mapping from $A_1 \times \cdots \times A_N$ into the set of real numbers R. We assume that the marginal densities $p(x_i) \stackrel{\text{def}}{=} \Pr\{X_i = x_i\}$ are also known. The marginal density function $p(x_i)$ represents our *a priori* "belief" about the random variable X_i. Now, suppose that one or more of these random variables is measured or "observed." This means that there is a subset $J \subseteq \{1, 2, \cdots, N\}$ (the evidence set) such that, for all $j \in J$, the random variable X_j is known to have a particular value, say a_j. The *evidence* is then defined to be the event

$$\mathcal{E} = \{X_j = a_j : j \in J\}.$$

The fundamental *probabilistic inference problem* is to compute the *updated beliefs*, i.e., the *a posteriori* or conditional probabilities $p(X_i|\mathcal{E})$, for all $i \notin J$.

The brute force approach to computing $p(X_i|\mathcal{E})$ is to sum over all of the terms of $p(\boldsymbol{x})$ which do not involve either i or J. To simplify notation, we assume $i = 1$, and $J = \{m+1, \cdots, N\}$. Then we have

$$p(X_1 = a|\mathcal{E}) = \alpha \sum_{x_2, \cdots, x_m} p(a, x_2, \cdots, x_m, a_{m+1}, \cdots, a_N). \quad (4.1)$$

If X_i can assume q_i different values, then computing the sum in (4.1) for each possible value of a requires $q_1 q_2 \cdots q_m$ additions, which is impractical unless m and the q_i's are very small numbers.

The idea behind the "Bayesian belief network" approach [28], [51] to this inference problem is to exploit any "partial independencies" which may exist among the X_i's to simplify belief updating. The simplest case of this is when the random variables $X_1, \cdots, X_N$ are mutually independent, in which case the work in (4.1) can be avoided altogether since an observation of one such variable cannot affect our belief in another. More generally, the partial independencies can be described by a *directed acyclic graph*, or DAG.

A DAG is a finite, directed graph, in which there are no directed cycles. For example, Fig. 4 shows a DAG with five vertices and five edges. Let us agree that if there is a directed edge $a \to b$, then a will be called a "parent" of b, and b will be called a "child" of a. If the set of parents of a vertex v is denoted by $\mathrm{pa}(v)$, then we can describe the graph of Fig. 4 as follows:

$$\begin{aligned} \mathrm{pa}(v_1) &= \emptyset \\ \mathrm{pa}(v_2) &= \emptyset \\ \mathrm{pa}(v_3) &= \{v_1\} \\ \mathrm{pa}(v_4) &= \{v_1, v_2\} \\ \mathrm{pa}(v_5) &= \{v_3, v_4\}. \end{aligned} \quad (4.2)$$

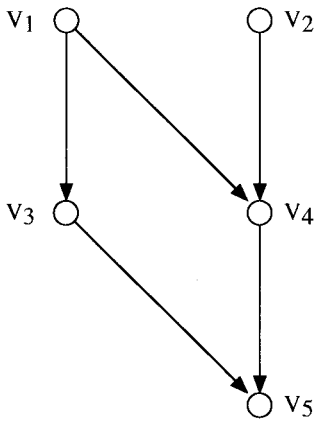


Fig. 4. Simple example of a DAG which represents a five-variable directed Markov field [see (4.4)]. This DAG is "loopy," with the vertices v_1, v_3, v_4, and v_5 forming a loop.

If G is a DAG, and if $\boldsymbol{X}$ is a set of random variables in one-to-one correspondence with the vertices of G, the joint density function $p(\boldsymbol{x})$ is said to *factor according to* G if

$$p(x_1, \cdots, x_N) = \prod_{i=1}^{N} p(x_i|\mathrm{pa}(x_i)) \quad (4.3)$$

where $\mathrm{pa}(x_i)$ denotes a value assignment for the parents of X_i. For example, a five-variable density function $p(x_1, \cdots, x_5)$

[4] As we shall see in Section IV, the BCJR algorithm itself, and the many variations of it, are themselves special cases of Pearl's algorithm. In this application, the algorithm is provably exact since the corresponding "belief" diagram has no loops.

[5] We have already used upper case X's to denote codeword components, for example, (2.1). We use upper case X's here to denote arbitrary random variables, and hope no confusion will occur.

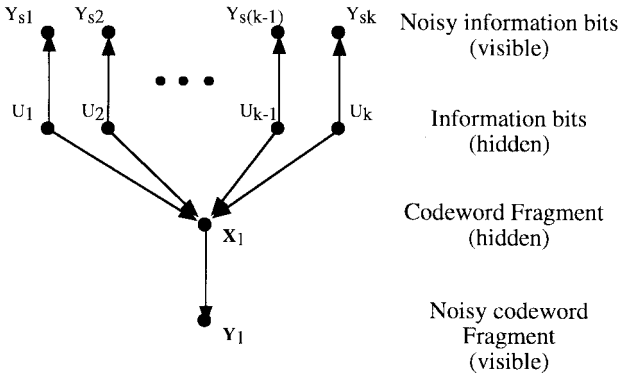


Fig. 5. Bayesian network interpretation of the decoding problem.

factors according to the graph of Fig. 4 if

$$p(x_1,x_2,x_3,x_4,x_5) = p(x_1)p(x_2)p(x_3|x_1)p(x_4|x_1,x_2)p(x_5|x_3,x_4). \quad (4.4)$$

A set of random variables $\boldsymbol{X}$ whose density functions factor according to a given DAG is called a *directed Markov field* [35], [32], [65]. For example, if G is a directed chain, then $\boldsymbol{X}$ is an ordinary Markov chain. A DAG, together with the associated random variables $\boldsymbol{X}$, is called a *Bayesian belief network*, or *Bayesian network* for short [28].

At this point, we observe that the general coding framework of Fig. 1 can be represented as the Bayesian network shown in Fig. 5. From the decoder's viewpoint, the observed noisy information bits Y_{si} are probabilistic functions of the hidden information bits U_i. Similarly, the observed noisy codeword fragment $\boldsymbol{Y}_1$ is a probabilistic function of the codeword $\boldsymbol{X}_1$, which in turn is a deterministic function of the hidden input bits. (Fig. 5 implies that the information bits U_i are independent.) The decoder's problem is thus to infer the values of the hidden variables U_i based on the evidence variables $(Y_{s1},\cdots,Y_{sk})$ and $\boldsymbol{Y}_1$.

Bayesian networks can sometimes lead to considerable simplifications of the probabilistic inference problem. The most important of these simplifications, for our purposes, is Pearl's *belief propagation* algorithm. In the 1980's, Kim and Pearl [31], [42]–[44] showed that if the DAG is a "tree," i.e., if there are no loops,[6] then there are efficient distributed algorithms for solving the inference problem. If all of the alphabets A_i have the same size q, Pearl's algorithm solves the inference problem on trees with $O(Nq^e)$ computations, where e is the maximum number of parents of any vertex, rather than $O(q^m)$, where m is the number of unknown random variables, which is required by the brute-force method. The efficiency of belief propagation on trees stands in sharp contrast to the situation for general DAG's since, in 1990, Cooper [16] showed that the inference problem in general DAG's is NP hard. (See also [17] and [53] for more on the NP hardness of probabilistic inference in Bayesian networks.)

Since the network in Fig. 5 is a tree, Pearl's algorithm will apply. However, the result is uninteresting: Pearl's algorithm applied to this Bayesian network merely gives an alternative derivation of Lemma 2.2.

[6] A "loop" is a cycle in the underlying undirected graph. For example, in the DAG of Fig. 4, $v_1 \to v_4 \to v_5 \to v_3 \to v_1$ is a loop.

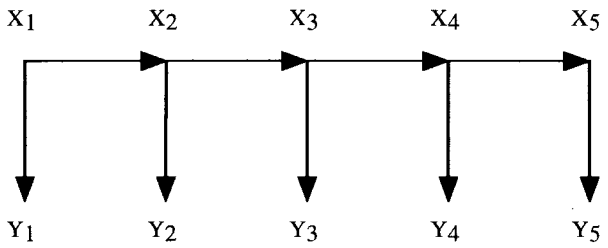


Fig. 6. Bayesian network for the "hidden Markov chain" problem. Here, $X_1,\cdots,X_N$ form a Markov chain, and $Y_1,\cdots,Y_N$ are noisy versions of $X_1,\cdots,X_N$. The problem is to compute the conditional probabilities of the hidden variables X_i based in the "evidence" variables Y_i.

A more profitable application of Pearl's algorithm is to the classic "hidden Markov chain" inference problem, where the appropriate Bayesian network is shown in Fig. 6. Here, the result is a linear-time exact solution which is functionally identical to the celebrated "forward–backward algorithm" discovered in the 1960's and 1970's.[7]

For us, the important feature of Pearl's BP algorithm is that it can be defined for an arbitrary DAG which is not necessarily a tree, even though there is no guarantee that the algorithm will perform a useful calculation if there are loops in the DAG. We believe that the key to the success of turbo codes, and a potentially important research area for the AI community, is the experimentally observed fact that Pearl's algorithm works "approximately" for some loopy, i.e., nontree DAG's.[8] We shall explain the connection between turbo codes and BP in Section VI, after first describing the BP algorithm in detail in Section V. For now, as a preview of coming attractions, we present Fig. 7, which is a loopy Bayesian network appropriate for the turbo decoding problem.[9]

V. Detailed Description of Pearl's Algorithm

In this section, we will give a detailed functional description of Pearl's algorithm as described in [44, Ch. 4].

[7] The forward–backward algorithm has a long and convoluted history that merits the attention of a science historian. It seems to have first appeared in the unclassified literature in two independent 1966 publications [6], [11]. Soon afterwards, it appeared in papers on MAP detection of digital sequences in the presence of intersymbol interference [23]. It appeared explicitly as an algorithm for tracking the states of a Markov chain in the early 1970's [40], [4] (see also the survey papers [47] and [49]). A similar algorithm (in "min-sum" form) appeared in a 1971 paper on equalization [62]. The algorithm was connected to the optimization literature in 1987 [63]. All of this activity appears to have been completely independent of the developments in AI that led to Pearl's algorithm!

[8] There is an "exact" inference algorithm for an arbitrary DAG, developed by Lauritzen and Spiegelhalter [34], which solves the inference problem with $O(N_c q^J)$ computations, where N_c is the number of cliques in the undirected triangulated "moralized" graph G_m which can be derived from G, and J is the maximum number of vertices in any clique in G_m. However, this proves not to be helpful in the turbo decoding problem since the appropriate DAG produces moralized graphs with huge cliques. For example, the turbo codes in [10] have an associated G_m with a clique of size 16 384.

[9] Our Fig. 7 should be compared to Wiberg [67, Fig. 2.5], which describes the "Tanner graph" of a turbo code. The figures are similar, but there is a key difference. Wiberg incorporates the turbo code's interleaver, citing it (the interleaver) as necessary for ensuring that there are no short cycles in the graph. In our Fig. 7, on the other hand, there are many short cycles. It is our belief the presence of short cycles does not, at least in many cases, compromise the performance of the decoding algorithm, although it may degrade the quality of the code. We will expand on these remarks at the conclusion of the paper.

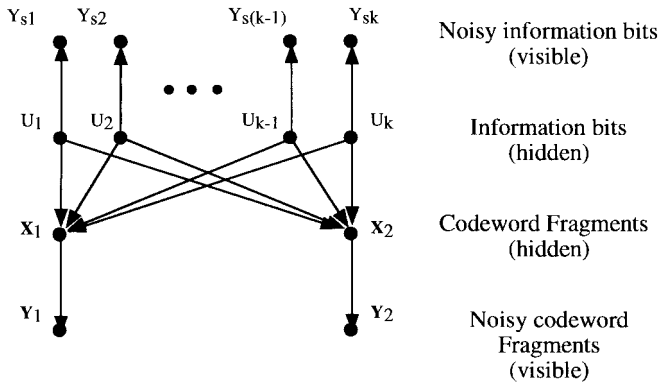


Fig. 7. Bayesian network interpretation of the turbo decoding problem. Note the presence of many loops, i.e., $U_1 \rightarrow \boldsymbol{X}_2 \rightarrow U_2 \rightarrow \boldsymbol{X}_1 \rightarrow U_1$.

Pearl's belief propagation algorithm is a decentralized "message-passing" algorithm, in which there is a processor associated with each vertex of G. Each processor can communicate only with its parents and children. Furthermore, the processor associated with a variable X is assumed to "know" the conditional density function $p(x|\boldsymbol{u}) \stackrel{\text{def}}{=} \Pr\{X = x|U_1 = u_1, \cdots, U_M = u_M\}$, where $U_1, \cdots, U_M$ are the parents of X. (If X has no parents, this knowledge is assumed to be the marginal density function $p(x) \stackrel{\text{def}}{=} \Pr\{X = x\}$.) Thus, the "local environment" of a node X is as shown in Fig. 8(a).

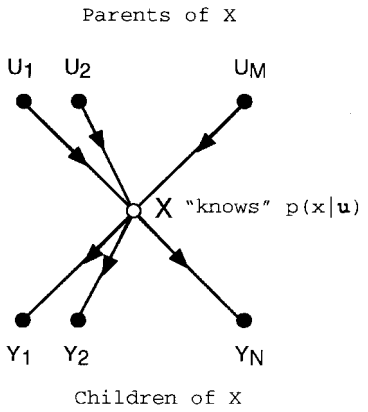
When a processor is activated, it "reads" the messages received from each of its parents and children, updates its belief based on these messages, and then sends new messages back to its parents and children.

The message a node X receives from its parent U_i, denoted $\pi_{U_i,X}(u_i)$, is in the form of a list of probabilities ("π" for "probability"), one for each value $u_i \in A_{U_i}$. Informally, $\pi_{U_i,X}(u_i)$ is the probability of the event $U_i = u_i$, conditioned on the evidence in the tree already "known" to U_i. Similarly, the message X receives from its child Y_j, denoted $\lambda_{Y_j,X}(x)$, is in the form of a list of nonnegative real numbers (likelihoods: "λ" for "likelihood"), one for each value of $x \in A_X$. Informally, $\lambda_{Y_j,X}(x)$ is the probability of the evidence Y_j "knows," conditioned on the event $X = x$. For simplicity, we adopt a vector notation for these incoming messages

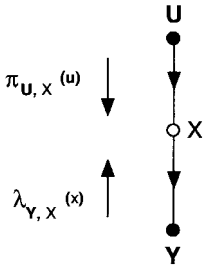
$$\begin{aligned}\boldsymbol{\pi}_{\boldsymbol{U},X}(\boldsymbol{u}) &\stackrel{\text{def}}{=} (\pi_{U_1,X}(u_1), \cdots, \pi_{U_M,X}(u_M)) \\ \boldsymbol{\lambda}_{\boldsymbol{Y},X}(x) &\stackrel{\text{def}}{=} (\lambda_{Y_1,X}(x), \cdots, \lambda_{Y_N,X}(x)). \end{aligned} \tag{5.1}$$

The situation is summarized in Fig. 8(b).

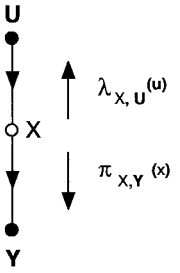
After X has been activated, the message that X passes to its child Y_j, denoted $\pi_{X,Y_j}(x)$, is a list of probabilities, one for each value of x. Roughly speaking, $\pi_{X,Y_j}(x)$ is the probability of the event $X = x$, given the evidence in the tree already "known" to X, which now includes any new evidence which may have been contained in the incoming messages. Similarly, the message that X passes to its parent U_i, denoted $\lambda_{X,U_i}(u_i)$, is the probability of the evidence it now knows about, given



(a)



(b)



(c)

Fig. 8. Summary of Pearl's algorithm. (Boldface symbols denote random vectors; ordinary symbols represent random variables.)

the event $U_i = u_i$. Again, we adopt a vector notation

$$\begin{aligned}\boldsymbol{\lambda}_{X,\boldsymbol{U}}(\boldsymbol{u}) &\stackrel{\text{def}}{=} (\lambda_{X,U_1}(u_1), \cdots, \lambda_{X,U_M}(u_M)) \\ \boldsymbol{\pi}_{X,\boldsymbol{Y}}(x) &\stackrel{\text{def}}{=} (\pi_{X,Y_1}(x), \cdots, \pi_{X,Y_N}(x)). \end{aligned} \tag{5.2}$$

This situation is summarized in Fig. 8(c).

Additionally, each node of the graph keeps track of a number of other quantities

$$\begin{aligned}\mu_X(\boldsymbol{u}) &: A_{U_1} \times \cdots \times A_{U_M} \rightarrow R \\ \lambda_X(x) &: A_X \rightarrow R \\ \pi_X(x) &: A_X \rightarrow R \\ \gamma_X(\boldsymbol{u}) &: A_{U_1} \times \cdots \times A_{U_M} \rightarrow R \\ \text{BEL}_X(x) &: A_X \rightarrow R. \end{aligned}$$

TABLE II
INITIALIZATION RULES FOR PEARL'S ALGORITHM

	quantity (at X)	initially (evid.) $(x = x_0)$	initially (non. evid.)
1.	$\mu_X(\boldsymbol{u})$	$--$	$--$
2.	$\lambda_X(x)$	$--$	$--$
3.	$\pi_X(x)$	$\delta(x, x_0)^*$	$\begin{cases} p(x)^* & \text{if } X \text{ is a source node} \\ -- & \text{otherwise} \end{cases}$
4.	$\gamma_X(\boldsymbol{u})$	$p(x_0\|\boldsymbol{u})$	1
5.	$\mathrm{BEL}_X(x)$	$\delta(x, x_0)^*$	$\begin{cases} p(x) & \text{if } X \text{ is a source node} \\ -- & \text{otherwise} \end{cases}$
6.	$\boldsymbol{\lambda}_{X,\boldsymbol{U}}(\boldsymbol{u})$	$\begin{cases} p(x_0\|u)^* & \text{if } M = 1 \\ -- & \text{otherwise} \end{cases}$	1
7.	$\boldsymbol{\pi}_{X,\boldsymbol{Y}}(x)$	$\delta(x, x_0)^*$	$\begin{cases} p(x) & \text{if } X \text{ is a source node} \\ -- & \text{otherwise} \end{cases}$

*Once initialized, these quantities never change.

The quantities $\mu_X(\boldsymbol{u}), \lambda_X(x), \pi_X(x)$, and $\gamma_X(\boldsymbol{u})$ have no particular intrinsic significance, but the quantity $\mathrm{BEL}_X(x)$ is the heart of the algorithm since, when the algorithm terminates, $\mathrm{BEL}_X(x)$ gives the value of the desired conditional probability $\Pr\{X = x|\mathcal{E}\}$.

Here, then, is a complete description of Pearl's algorithm. When the node X is activated, it "reads" its incoming messages $\boldsymbol{\pi}_{\boldsymbol{U},X}(\boldsymbol{u})$ and $\boldsymbol{\lambda}_{\boldsymbol{Y},X}(x)$, and updates $\mu_X(\boldsymbol{u}), \lambda_X(x), \pi_X(x), \gamma_X(\boldsymbol{u}), \mathrm{BEL}_X(x), \boldsymbol{\lambda}_{X,\boldsymbol{U}}(\boldsymbol{u})$ and $\boldsymbol{\pi}_{X,\boldsymbol{Y}}(x)$, in that order, using the update rules in Table I and the initial values given in Table II. (In Table I, we use the notation $\langle \boldsymbol{v} \rangle = v_1 v_2 \cdots v_n$ if $\boldsymbol{v} = (v_1, \cdots, v_n)$ is a vector of real numbers.) A node can be activated only if all of its incoming messages exist. Otherwise, the order of node activation is arbitrary. Pearl proved that if the DAG is a tree, then after a number of iterations at most equal to the diameter of the tree, each node will have correctly computed its "belief," i.e., the probability of the associated random variable, conditioned on all of the evidence in the tree, and no further changes in the beliefs will occur. If the network is not a tree, the algorithm has no definite termination point, but in practice, the termination rule chosen is either to stop after a predetermined number of iterations, or else to stop when the computed beliefs cease to change significantly.

VI. TURBO DECODING AS AN INSTANCE OF BP

In this section, we will show formally that if Pearl's BP algorithm is applied to the belief network of Fig. 7, the result is an algorithm which is identical to the "turbo decoding" algorithm described in Section III. More precisely, we will show that if the network of Fig. 7 is initialized using the rules of Table II, and if the nodes are updated (using Table I) in the order $\boldsymbol{U}, X_1, \boldsymbol{U}, X_2, \boldsymbol{U}, X_1, \cdots$, the results are summarized in Table III. In particular, the sequence of "beliefs" in the information symbols $\boldsymbol{U}$ will be

$$\alpha\boldsymbol{\lambda}, \alpha\boldsymbol{\lambda}\boldsymbol{\pi}^{(1)}\boldsymbol{\pi}^{(0)}, \alpha\boldsymbol{\lambda}\boldsymbol{\pi}^{(2)}\boldsymbol{\pi}^{(1)}, \alpha\boldsymbol{\lambda}\boldsymbol{\pi}^{(3)}\boldsymbol{\pi}^{(2)}, \cdots$$

in agreement with (3.8).

TABLE III
PEARL'S ALGORITHM APPLIED TO THE BELIEF NETWORK OF FIG. 7
(NODES ARE ACTIVATED IN THE ORDER SHOWN IN THE FIRST COLUMN)

node activated	$\mathbf{BEL}_{\boldsymbol{U}}$	$\boldsymbol{\pi}_{\boldsymbol{U},X_1}$	$\boldsymbol{\pi}_{\boldsymbol{U},X_2}$	$\boldsymbol{\lambda}_{X_1,\mathrm{U}}$	$\boldsymbol{\lambda}_{X_2,\mathrm{U}}$
(initial conditions)				$\mathbf{1}$	$\mathbf{1}$
$\boldsymbol{U}$	$\alpha\,\boldsymbol{\lambda}$	$\alpha\,\boldsymbol{\lambda}$	$\alpha\,\boldsymbol{\lambda}$	"	"
X_1	"	"	"	$\boldsymbol{\pi}^{(1)}$	"
$\boldsymbol{U}$	$\alpha\,\boldsymbol{\lambda}\boldsymbol{\pi}^{(1)}\boldsymbol{\pi}^{(0)}$	"	$\alpha\,\boldsymbol{\lambda}\boldsymbol{\pi}^{(1)}$	"	"
X_2	"	"	"	"	$\boldsymbol{\pi}^{(2)}$
$\boldsymbol{U}$	$\alpha\,\boldsymbol{\lambda}\boldsymbol{\pi}^{(2)}\boldsymbol{\pi}^{(1)}$	$\alpha\,\boldsymbol{\lambda}\boldsymbol{\pi}^{(2)}$	"	"	"
X_1	"	"	"	$\boldsymbol{\pi}^{(3)}$	"
$\boldsymbol{U}$	$\alpha\,\boldsymbol{\lambda}\boldsymbol{\pi}^{(3)}\boldsymbol{\pi}^{(2)}$	"	$\alpha\,\boldsymbol{\lambda}\boldsymbol{\pi}^{(3)}$	"	"
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Let us now verify the entries in Table III. First, we discuss the necessary initializations. Because U_i is a source node (i.e., it has no parents), and since we are assuming that the prior distribution on the U_i's is independent and uniform, by line 3 in Table II, the quantity $\pi_{U_i}(u_i)$ is permanently set as follows:

$$\pi_{U_i}(u_i) = \alpha\mathbf{1} \quad \text{(permanent)}. \tag{6.1}$$

Since the Y_{si}'s are "direct evidence" nodes (i.e., evidence nodes which have only one parent), by line 6 of Table II, the message that Y_{si} sends the U_i is permanently set as follows:

$$\lambda_{Y_{si},U_i}(u_i) = p(y_{si}|u_i) = \lambda_i(u_i) \quad \text{(permanent)} \tag{6.2}$$

Since the nodes X_1 and X_2 are not evidence nodes, by line 6 of Table II, the messages that they send to the U_i's are

initially set as follows:

$$\lambda_{X_j,U_i}(u_i) = 1 \quad \text{(temporary)} \tag{6.3}$$

which appears (in vector notation) in line 1 of Table III.

Now, we simultaneously activate the nodes $U_1, \cdots, U_k$. Since U_i is a source node, it is not necessary to evaluate μ_{U_i}, γ_{U_i}, or the λ messages. By line 2 of Table I

$$\begin{aligned}\lambda_{U_i}(u_i) &= \lambda_{Y_{si},U_i}(u_i) \cdot \lambda_{X_1,U_i}(u_i) \cdot \lambda_{X_2,U_i}(u_i) \quad (6.4)\\ &= \lambda_i(u_i) \cdot 1 \cdot 1 \quad \text{[by (6.2) and (6.3)]}\\ &= \lambda_i(u_i). \quad (6.5)\end{aligned}$$

Similarly, by line 5 of Table I

$$\begin{aligned}\mathrm{BEL}_{U_i}(u_i) &= \alpha\lambda_{U_i}(u_i) \cdot \pi_{U_i}(u_i) \quad (6.6)\\ &= \alpha\lambda_i(u_i) \cdot 1 \quad \text{[by (6.1) and (6.5)]}\\ &= \alpha\lambda_i(u_i). \quad (6.7)\end{aligned}$$

In vector notation, (6.7) is equivalent to

$$\mathbf{BEL} = \alpha\boldsymbol{\lambda}$$

which appears in line 2 of Table III.

The message U_i sends to X_1 is, according to line 7, Table I

$$\begin{aligned}\pi_{U_i,X_1}(u_i) &= \alpha\pi_{U_i}(u_i) \cdot \lambda_{Y_{si},U_i}(u_i) \cdot \lambda_{X_2,U_i}(u_i) \quad (6.8)\\ &= \alpha 1 \cdot \lambda_i(u_i) \cdot 1 \quad \text{[by (6.1), (6.2), (6.3)]}\\ &= \alpha\lambda_i(u_i) \quad (6.9)\end{aligned}$$

for $i = 1, \cdots, k$. In vector notation, (6.9) becomes

$$\boldsymbol{\pi}_{\boldsymbol{U},X_1} = \alpha\boldsymbol{\lambda} \tag{6.10}$$

which also appears in line 2 of Table III. A similar calculation gives $\boldsymbol{\pi}_{\boldsymbol{U},X_2} = \alpha\boldsymbol{\lambda}$, which again appears in line 2 of Table III.

Next, we update X_1. The quantities $\mu_{X_1}(\boldsymbol{u}), \mathrm{BEL}_{X_1}(x_1)$, and $\boldsymbol{\pi}_{X_1,\boldsymbol{U}}$ are not required since we do not update the evidence node Y_1. Since Y_1 is an evidence node, by line 6, Table II, the message $\lambda_{Y_1,X_1}(x_1)$ is permanently fixed as $p(y_1|x_1)$. Thus, by line 2, Table I, $\lambda_{X_1}(x_1)$ is also fixed

$$\lambda_{X_1}(x_1) = p(y_1|x_1) \quad \text{(permanent)}. \tag{6.11}$$

Next, we compute $\gamma_{X_1}(\boldsymbol{u})$, using line 4 of Table I:

$$\gamma_{X_1}(\boldsymbol{u}) = \sum_{x_1} p(y_1|x_1)p(x_1|\boldsymbol{u}).$$

Since X_1 is a deterministic function of $\boldsymbol{U}$, if follows that $p(x_1|\boldsymbol{u})$ is equal to 1 for that value of $\boldsymbol{u}$ that produces the code fragment x_1, i.e.,

$$\begin{aligned}\gamma_{X_1}(\boldsymbol{u}) &= p(y_1|x_1(\boldsymbol{u}))\\ &= p_1(\boldsymbol{u}) \quad \text{(permanent)} \quad (6.12)\end{aligned}$$

where in 6.12 we have used the definition (3.5). Finally, we update the messages $\boldsymbol{\lambda}_{X_1,\boldsymbol{U}}$, using line 6 of Table I

$$\begin{aligned}\boldsymbol{\lambda}_{X_1,\boldsymbol{U}} &= \boldsymbol{\pi}_{\boldsymbol{U},X_1} \circ \gamma_{X_1}\\ &= \alpha\boldsymbol{\lambda} \circ p_1 \quad \text{[by (6.10) and (6.12)]}\\ &= \alpha(\boldsymbol{\lambda}\boldsymbol{\pi}^{(0)} \circ p_1) \quad \text{[by (3.6)]}\\ &= \boldsymbol{\pi}^{(1)} \quad \text{[by (3.7)]} \quad (6.13)\end{aligned}$$

which appears in line 3 of Table III.

Now, we update $\boldsymbol{U}$ again, using the definition (6.4), and the previous values given in (6.2), (6.13), and (6.3)

$$\lambda_{U_i}(u_i) = \lambda_i(u_i)\pi^{(1)}(u_i). \tag{6.14}$$

TABLE IV
PEARL'S ALGORITHM APPLIED IN A SLIGHTLY DIFFERENT WAY TO THE BELIEF NETWORK OF FIG. 7 (NODES ARE ACTIVATED IN THE ORDER SHOWN IN THE FIRST COLUMN)

node activated	$\mathbf{BEL}_{\boldsymbol{U}}$	$\boldsymbol{\pi}_{\boldsymbol{U},X_1}$	$\boldsymbol{\pi}_{\boldsymbol{U},X_2}$	$\boldsymbol{\lambda}_{X_1,\mathrm{U}}$	$\boldsymbol{\lambda}_{X_2,\mathrm{U}}$
(initially)				$\mathbf{1}$	$\mathbf{1}$
$\boldsymbol{U}$	$\alpha\,\boldsymbol{\lambda}$	$\alpha\,\boldsymbol{\lambda}$	$\alpha\,\boldsymbol{\lambda}$	"	"
X_1, X_2	"	"	"	$\boldsymbol{\pi}^{(1)}$	$\tilde{\boldsymbol{\pi}}^{(1)}$
$\boldsymbol{U}$	$\alpha\,\boldsymbol{\lambda}\boldsymbol{\pi}^{(1)}\tilde{\boldsymbol{\pi}}^{(1)}$	$\alpha\boldsymbol{\lambda}\tilde{\boldsymbol{\pi}}^{(1)}$	$\alpha\,\boldsymbol{\lambda}\boldsymbol{\pi}^{(1)}$	"	"
X_1, X_2	"	"	"	$\tilde{\boldsymbol{\pi}}^{(2)}$	$\boldsymbol{\pi}^{(2)}$
$\boldsymbol{U}$	$\alpha\,\boldsymbol{\lambda}\boldsymbol{\pi}^{(2)}\tilde{\boldsymbol{\pi}}^{(2)}$	$\alpha\,\boldsymbol{\lambda}\boldsymbol{\pi}^{(2)}$	$\alpha\,\boldsymbol{\lambda}\tilde{\boldsymbol{\pi}}^{(2)}$	"	"
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Similarly, using the definition (6.6), and the previous values in (6.4) and (6.1)

$$\mathrm{BEL}_i(u_i) = \alpha\lambda_i(u_i)\pi^{(1)}(u_i)$$

which, in vector notation, is

$$\begin{aligned}\mathbf{BEL}_{\boldsymbol{U}} &= \alpha\boldsymbol{\lambda}\boldsymbol{\pi}^{(1)}\\ &= \alpha\boldsymbol{\lambda}\boldsymbol{\pi}^{(1)}\boldsymbol{\pi}^{(0)}\end{aligned}$$

in agreement with line 4 of Table III.

Next, we update $\boldsymbol{\pi}_{\boldsymbol{U},X_1}$ and $\boldsymbol{\pi}_{\boldsymbol{U},X_2}$

$$\begin{aligned}\pi_{U_i,X_1}(u_i) &= \alpha\pi_{U_i}(u_i) \cdot \lambda_{Y_{si},U_i}(u_i) \cdot \lambda_{X_2,U_i}(u_i) \quad \text{[by (6.8)]}\\ &= \alpha 1 \cdot \lambda_i(u_i) \cdot 1 \quad \text{[by (6.1), (6.2), (6.3)]}\\ &= \alpha\lambda_i(u_i) \quad (6.15)\end{aligned}$$

and

$$\begin{aligned}\pi_{U_i,X_2}(u_i) &= \alpha\pi_{U_i}(u_i) \cdot \lambda_{Y_{si},U_i}(u_i) \cdot \lambda_{X_1,U_i}(u_i) \quad \text{[like (6.8)]}\\ &= \alpha 1 \cdot \lambda_i(u_i) \cdot \pi^{(1)}(u_i) \quad \text{[by (6.1), (6.2), (6.4)]}\\ &= \alpha\lambda_i(u_i) \cdot \pi^{(1)}(u_i). \quad (6.16)\end{aligned}$$

The values (6.15) and (6.16) are the ones given in line 4 of Table III. It is now a matter of routine to verify that the rest of the values given in Table III are correct.

The order in which we chose to update the nodes in Fig. 7 was arbitrary, and other orders give different algorithms. For example, it is easy to verify that the update order $\boldsymbol{U}, \boldsymbol{X}, \boldsymbol{U}, \boldsymbol{X}, \cdots$ yields the results in Table IV, where the sequences $\boldsymbol{\pi}^{(m)}$ and $\boldsymbol{\pi}t^{(m)}$ are defined by

$$\boldsymbol{\pi}^{(0)} = \tilde{\boldsymbol{\pi}}^{(0)} = (\alpha\mathbf{1}, \cdots, \alpha\mathbf{1})$$

and

$$\boldsymbol{\pi}^{(m)} = \begin{cases} \alpha\boldsymbol{\lambda}\boldsymbol{\pi}^{(m-1)} \circ p_1, & \text{if } m \text{ is odd} \\ \alpha\boldsymbol{\lambda}\boldsymbol{\pi}^{(m-1)} \circ p_2, & \text{if } m \text{ is even} \end{cases} \tag{6.17}$$

$$\tilde{\boldsymbol{p}}^{(m)} = \begin{cases} \alpha\boldsymbol{\lambda}\tilde{\boldsymbol{p}}^{(m-1)} \circ p_1, & \text{if } m \text{ is even} \\ \alpha\boldsymbol{\lambda}\tilde{\boldsymbol{p}}^{(m-1)} \circ p_2, & \text{if } m \text{ is odd.} \end{cases} \tag{6.18}$$

It would be interesting to experiment with this alternative version of the turbo decoding algorithm. (This "parallel update" rule is, in fact, the rule used to derived the decoding algorithm for multiple turbo codes, as discussed in Section VII.)

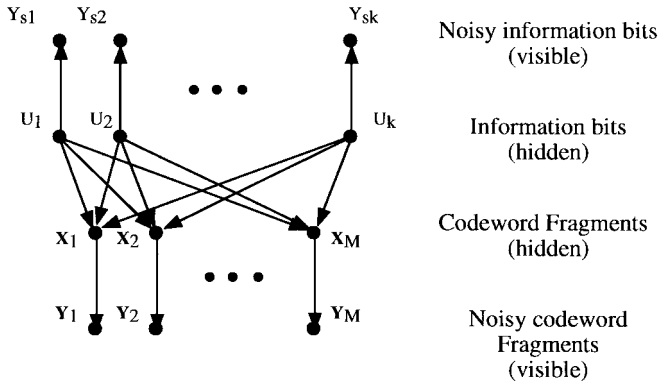


Fig. 9. Belief network appropriate for decoding a "multiple" turbo code, in which there are M code fragments.

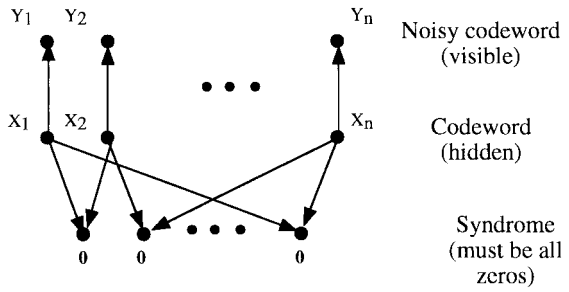


Fig. 10. Belief network for decoding a Gallager "low-density parity-check" code.

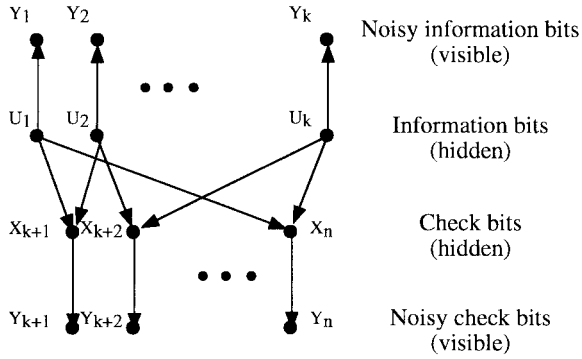


Fig. 11. Belief network for decoding systematic, low-density generator matrix codes.

VII. Other Decoding Algorithms Derived from Belief Propagation

As we have seen in Sections IV and V, Pearl's algorithm can be applied to any belief network, not just to one like Fig. 7. It is a fruitful exercise to apply Pearl's algorithm to the belief networks of a variety of hybrid coding schemes, to see what results. In this section, we will briefly outline (without proofs) what we have discovered along these lines.

• *Multiple Turbo Codes:* As we have defined them, turbo codes involve only two encodings of the information, as shown in Fig. 2. However, several researchers (e.g., [19]) have experimented with three or more parallel encodings. If there are M parallel encodings, the appropriate belief network is as shown in Fig. 9. Applying the BP algorithm to this belief network, with the update order $\boldsymbol{U}, \boldsymbol{X}, \boldsymbol{U}, \boldsymbol{X}, \cdots$, we obtain a generalized turbo decoding algorithm which is identical to the one employed successfully in [19].

• *Gallager's Low-Density Parity-Check Codes:* The earliest suboptimal iterative decoding algorithm is that of Gallager, who devised it as a method of decoding his "low-density parity-check" codes [25], [26]. This algorithm was later generalized and elaborated upon by Tanner [61] and Wiberg [67]. But as MacKay and Neal [37]–[39] have pointed out, in the first citation of belief propagation by coding theorists, Gallager's algorithm is a special kind of BP, with Fig. 10 as the appropriate belief network. [In Fig. 10, $\boldsymbol{X} = (X_1, \cdots, X_n)$ is a codeword which satisfies the parity-check equations $H\boldsymbol{X} = \boldsymbol{0}$. $\boldsymbol{Y} = (Y_1, \cdots, Y_n)$ is a noisy version of $\boldsymbol{X}$. The "syndrome" $\boldsymbol{S} = (S_1, \cdots, S_r)$ is defined as $\boldsymbol{S} = H\boldsymbol{X}$, which is perpetually "observed" to be $(0, \cdots, 0)$]. Although LDPC codes had largely been forgotten by coding theorists until their rediscovery by MacKay and Neal, simulations of Gallager's original decoding algorithm made with powerful modern computers show that their performance is remarkably good, in many cases rivaling that of turbo codes. More recently, Sipser and Spielman [57], [60] have replaced the "random" parity-check martrices of Gallager and MacKay–Neal with deterministic parity-check matrices with desirable properties, based on "expander" graphs, and have obtained even stronger results.

• *Low-Density Generator Matrix Codes:* Recently, Cheng and McEliece have experimented with BP decoding on certain systematic linear block codes with low-density *generator matrices* [13]. (This same class of codes appeared earlier in a paper by MacKay [36] in a study of modulo-2 arithmetic inference problems, and in a paper by by Spielman [60] in connection with "error reduction.") The decoding algorithm devised by Cheng and McEliece was adapted from the one described in the MacKay–Neal paper cited above, and the results were quite good, especially at high rates. More recently, Cheng [14], [15] used some of these same ideas to construct a class of block codes which yield some remarkably efficient multilevel coded modulations. Fig. 11 shows the belief network for low-density generator matrix codes used by McEliece and Cheng.

• *Serially Concatenated Codes:* We have defined a turbo code to be the parallel concatenation of two or more components codes. However, as originally defined by Forney [22], concatenation is a *serial* operation. Recently, several researchers [8], [9] have investigated the performance of serially concatenated codes, with turbo-style decoding. This is a nontrivial variation on the original turbo decoding idea, and the iterative decoding algorithms in [8] and [9] differ so significantly from the original Berrou *et al.* algorithm that they must be considered an original invention. Still, these decoding algorithms can be derived routinely from a BP viewpoint, using the network of Fig. 12. Here, $\boldsymbol{U}$ is the information to be encoded, $\boldsymbol{X}$ is the outer (first) encoding, $\boldsymbol{Y}$ is the inner (second) encoding, and $\boldsymbol{X}$ is the noisy version of $\boldsymbol{Y}$.

• *Product Codes:* A number of researchers have been successful with turbo-style decoding of product codes in two or more dimensions [46], [48], [54], [27]. In a product code, the

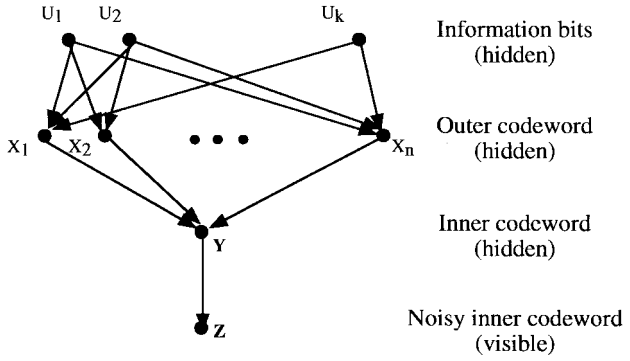


Fig. 12. Belief network for decoding a pair of serially concatenated codes.

information is arranged in an M-dimensional array, and then encoded separately in each dimension. Thus, the appropriate belief network is like the ones in Figs. 7 and 9 (a product code, is, by definition, systematic). We have experimented with "BP" decoding of product codes, and obtained results similar to those in the cited references. However, in this case, it appears that the BP algorithms differ in some small details from turbo-style decoding, and we are currently investigating this phenomenon.

- *"Tail-Biting" Convolutional Codes:* The class of "tail-biting" convolutional codes introduced by Solomon and van Tilborg [56] is a natural candidate for BP decoding. Briefly, a tail-biting convolutional code is a block code formed by truncating the trellis of a conventional convolutional code and then pasting the ends of the trellis together. If the parent convolutional code is an (n, k) code, and if the truncation depth is N, the resulting tail-biting code is an (Nn, Nk) block code.

In Fig. 13, we show a belief diagram for a tail-biting code where the truncation depth is $N = 5$. Assuming as above that the parent convolutional code is an (n, k) code, then in Fig. 13, the U_i's are k-bit information words, and the X_i's are n-bit codeword segments. The Y_i's are the observed noisy versions of the X_i's. The nodes intermediate between the information words and the codeword segments are pairs of encoder states. For a given encoder state pair (S_{i-2}, S_{i-1}) and information word U_i, the encoder rules (deterministically) produce the next pair of codeword states (S_{i-1}, S_i) and the next codeword segment X_i. If it were not for the "tail-biting" edge from (S_{n-1}, S_n) to (S_n, S_0), this belief net would be without loops and would represent an ordinary convolutional code. If, then, the BP algorithm were applied, the result would be identical to the BCJR APP decoding algorithm.[10]

If we were to apply Pearl's algorithm to the belief diagram of Fig. 13, we would obtain an iterative decoding algorithm for the tail-biting code. To our knowledge, no one has done exactly that, but Wiberg [67] has applied his algorithm to the Tanner graph of a tail-biting code with good success, and functionally, these two approaches yield virtually identical algorithms. Forney [24] has also discussed the iterative decoding of tail-biting codes using the Tanner–Wiberg approach.

[10] In this connection, we should note that Wiberg [67] has observed that his algorithm, when applied to a Tanner graph similar to Fig. 13 (less the tail-biting edge), also implies the BCJR algorithm. The "min-sum" form of Wiberg's algorithm, when applied to the same graph, is closely related to Viterbi's algorithm. Incidentally, there is a "min-sum" version of Pearl's algorithm described in [44, Ch. 5], called "belief revision," which does the same thing.

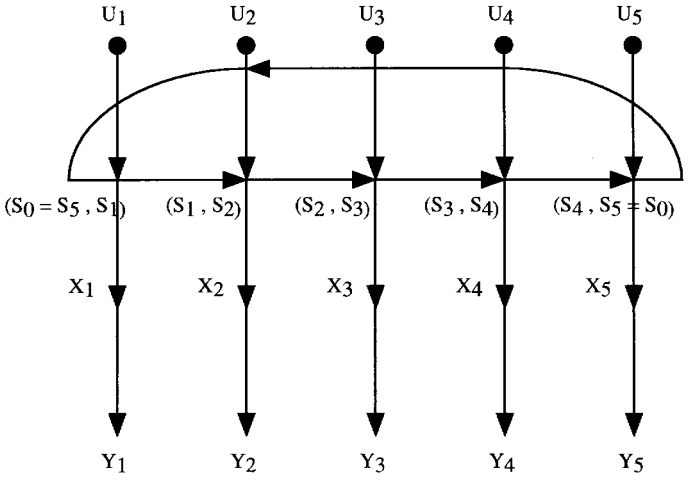


Fig. 13. Belief network for decoding a tail-biting convolutional code, illustrated for a truncation length of $N = 5$.

VIII. Concluding Remarks

We have shown that Pearl's algorithm provides a systematic method for devising low-complexity, suboptimal iterative decoding algorithms for a wide variety of error-control systems. Although there is as yet no guarantee that these algorithms will give useful results, the great body of experimental work done in the "turbo-code" literature suggests that the performance is likely to be very good.

One of the most interesting historical aspects of the turbo decoding problem is how often in the past inventors of decoding algorithms have hit upon a "BP"-like algorithm. The earliest, almost clairvoyant, occurrence is in the papers of Gallager [25], [26]. Later, Tanner [61], realizing the importance of Gallager's construction, made an important generalization of low-density parity check codes, and of Gallager's iterative decoding algorithm. With hindsight, especially in view of the recent work of Wiberg [67], it is now evident that both Viterbi's algorithm [64], [23] and the BCJR algorithm [4] can be viewed as a kind of belief propagation. Indeed, Wiberg [66], [67] has generalized Gallager's algorithm still further, to the point that it now resembles Pearl's algorithm very closely. (In particular, Wiberg shows that his algorithm can be adapted to produce both the Gallager–Tanner algorithm and the turbo decoding algorithm.) Finally, having noticed the similarity between the Gallager–Tanner–Wiberg algorithm and Pearl's algorithm, Aji and McEliece [1], [2], relying heavily on the post-Pearl improvements and simplifications in the BP algorithm [29], [30], [52], [58], [59] have devised a simple algorithm for distributing information on a graph that is a simultaneous generalization of both algorithms, and which includes several other classic algorithms, including Viterbi's algorithm (which is already subsumed by Wiberg's algorithm in "min-sum" form) and the FFT. It is natural to predict that this algorithm or one of its close relatives will soon

become a standard tool for scientists in communications, signal processing, and related fields.

We conclude with our view of "why" turbo coding is so successful. We believe that there are two, separable, essential contributing factors.

First: The presence of the pseudorandom interleavers between the component codes ensures that the resulting overall code behaves very much like a long random code, and by Shannon's theorems, a long random code is likely to be "good" in the sense of having the potential, with optimal decoding, to achieve performance near channel capacity. But optimal decoding would be impossibly complex. This brings us to the second essential factor.

Second: We believe that there are general undiscovered theorems about the performance of belief propagation algorithms on loopy DAG's. These theorems, which may have nothing directly to do with coding or decoding, will show that in some sense BP "converges with high probability to a near-optimum value" of the desired belief on a class of loopy DAG's that includes most or all of the diagrams in Figs. 7, 9, and 10–13 of this paper. If such theorems exist, they will no doubt find applications in realms far beyond information theory.

ACKNOWLEDGMENT

The authors wish to thank P. Smyth for apprising them about the "post-Pearl" developments in the belief propagation algorithm, and one of the referees for supplying them with much of the history of the forward–backward algorithm that appears in Section IV.

REFERENCES

[1] S. Aji and R. J. McEliece, "A general algorithm for distributing information on a graph," in *Proc. 1997 IEEE Int. Symp. Inform. Theory*, Ulm, Germany, June 1997, p. 6.

[2] ———, "The generalized distributive law," in *Proc. 4th Int. Symp. Commun. Theory Appl.*, Ambleside, U.K., July 1997, pp. 135–146. Revised version available from http://www.systems.caltech.edu/EE/Faculty/rjm.

[3] J. Andersen, "The TURBO coding scheme," unpublished manuscript distributed at *1994 IEEE Int. Symp. Inform. Theory*, Trondheim, Norway, June 1994.

[4] L. R. 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.

[5] A. S. Barbulescu and S. S. Pietrobon, "Interleaver design for three dimensional turbo-codes," in *Proc. 1995 IEEE Int. Symp. Inform. Theory*, Whistler, B.C., Canada, Sept. 1995, p. 37.

[6] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state Markov chains," *Ann. Math. Statist.*, vol. 37, pp. 1554–1563, 1966.

[7] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 409–428, Mar. 1996.

[8] ———, "Serial concatenation of block and convolutional codes," *Electron. Lett.*, vol. 32, pp. 887–888, May 1996.

[9] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *JPL TDA Progr. Rep.*, vol. 42-126, Aug. 1996.

[10] G. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding: Turbo codes," in *Proc. 1993 Int. Conf. Commun.*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[11] R. W. Chang and J. C. Hancock, "On receiver structures for channels having memory," *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 463–468, Oct. 1966.

[12] J.-F. Cheng and R. J. McEliece, "Unit memory Hamming turbo codes," in *Proc. 1995 IEEE Int. Symp. Inform. Theory*, Whistler, B.C., Canada, Sept. 1995, p. 33.

[13] ———, "Near capacity codecs for the Gaussian channel based on low-density generator matrices," submitted to 1996 Allerton Conf.

[14] J.-F. Cheng, "On the construction of efficient multilevel coded modulations," submitted to the 1997 IEEE Int. Symp. Inform. Theory.

[15] ———, "Iterative decoding," Ph.D dissertation, Caltech, Pasadena, CA, Mar. 1997.

[16] G. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks," *Artif. Intell.*, vol. 42, pp. 393–405, 1990.

[17] P. Dagum and M. Luby, "Approximating probabilistic inference in Bayesian belief networks is NP-hard," *Artif. Intell.*, vol. 60, pp. 141–153, 1993.

[18] D. Divsalar and F. Pollara, "Turbo codes for deep-space communications," *TDA Progr. Rep.*, vol. 42-120, pp. 29–39, Feb. 15, 1995.

[19] ———, "Multiple turbo codes for deep-space communications," *TDA Progr. Rep.* vol. 42-121, pp. 66–77, May 15, 1995.

[20] D. Divsalar, S. Dolinar, R. J. McEliece, and F. Pollara, "Transfer function bounds on the performance of turbo codes," *TDA Progr. Rep.* vol. 42-122, pp. 44–55, July 15, 1995.

[21] D. Divsalar and R. J. McEliece, "Effective free distance of turbo-codes," *Electron. Lett.*, vol. 32, pp. 445–446, Feb. 1996.

[22] G. D. Forney, Jr., *Concatenated Codes*. Cambridge, MA: MIT Press, 1966.

[23] ———, "The Viterbi algorithm," *Proc. IEEE*, vol. 63, pp. 268–278, Mar. 1973.

[24] ———, "The forward-backward algorithm" in *Proc. 34th Allerton Conf. Commun., Contr., Computing*, Allerton, IL, Oct. 1996.

[25] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[26] ———, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[27] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory* vol. 42, pp. 429–445, Mar. 1996.

[28] D. Heckerman and M. P. Wellman, "Bayesian networks," *Commun. ACM*, vol. 38, pp. 27–30, 1995.

[29] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen, "Bayesian updating in recursive graphical models by local computations," *Computational Statist. Quart.*, vol. 4, pp. 269–282, 1990.

[30] F. V. Jensen, *An Introduction to Bayesian Networks*. New York: Springer-Verlag, 1996.

[31] J. H. Kim and J. Pearl, "A computational model for combined causal and diagnostic reasoning in inference systems," in *Proc. 8th Int. Joint Conf. AI (IJCAI83)*, Karlsruhe, Germany, pp. 190–193.

[32] R. Kindermann and J. L. Snell, *Markov Random Fields and their Applications*. Providence, RI: American Mathematical Society, 1980.

[33] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models, this issue, pp. 219–230.

[34] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *J. Roy. Statist. Soc., Ser. B*, vol. 50, pp. 157–224, 1988.

[35] S. L. Lauritzen, A. P. Dawid, B. N. Larsen, and H.-G. Leimer, "Independence properties of directed Markov fields," *Networks*, vol. 20, pp. 491–505, 1990.

[36] D. J. C. MacKay, "A free energy minimization framework for inference problems in modulo 2 arithmetic," in *Fast Software Encryption*, B. Preneel, Ed. Berlin, Germany: Springer-Verlag Lecture Notes in Computer Science, vol. 1008, 1995, pp. 179–195.

[37] D. J. C. MacKay and R. Neal, "Good codes based on very sparse matrices," in *Proc. 5th IMA Conf. Cryptography and Coding*, C. Boyd, Ed. Berlin, Germany: Springer Lecture Notes in Computer Science, vol. 1025, 1995, pp. 100–111.

[38] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," submitted to *IEEE Trans. Inform. Theory.* Preprint available from http://wol.ra.phy.cam.ac.uk.

[39] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 32, pp. 1645–1646, Aug. 1996. Reprinted in *Electron. Lett.*, vol. 33, pp. 457–458, Mar. 1997.

[40] P. L. McAdam, L. Welch, and C. Weber, "M.A.P. bit decoding of convolutional codes," in *Abstr. Papers, 1972 IEEE Int. Symp. Inform. Theory*, Asilomar, CA, Jan. 1972, p. 90.

[41] R. J. McEliece, E. R. Rodemich, and J-F. Cheng, "The turbo decision algorithm," in *Proc. 33rd Allerton Conf. Commun., Contr., Computing*, Oct. 1995, pp. 366–379.

[42] J. Pearl, "Reverend Bayes on inference engines: A distributed hierarchical approach," in *Proc. Conf. Nat. Conf. AI*, Pittsburgh, PA, 1982, pp. 133–136.

[43] _____, "Fusion, propagation, and structuring in belief networks," *Artif. Intell.*, vol. 29, pp. 241–288, 1986.
[44] _____, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.
[45] L. C. Perez, J. Seghers, and D. J. Costello, Jr., " A distance spectrum interpretation of turbo codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1698–1709, Nov. 1996.
[46] A. Picart and R. Pyndiah, "Performance of turbo decoded product codes used in multilevel coding," in *Proc. IEEE ICC'96*, Dallas, TX, June 1996.
[47] A. M. Poritz, "Hidden Markov models: A guided tour," in *Proc. 1988 IEEE Int. Conf. Acoust., Speech, Signal Processing*. New York: IEEE Press, vol. 1, pp. 7–13.
[48] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near optimum decoding of product codes," in *Proc. IEEE GLOBECOM'94*, San Francisco, CA, Nov. 1994, vol. 1, pp. 339–343.
[49] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, pp. 257–285, 1989.
[50] P. Robertson, "Illuminating the structure of code and decoder of parallel and concatenated recursive systematic (Turbo) codes," in *Proc. IEEE GLOBECOM 1994*, pp. 1298–1303.
[51] R. D. Shachter, "Probabilistic inference and influence diagrams," *Oper. Res.*, vol. 36, pp. 589–604, 1988.
[52] G. R. Shafer and P. P. Shenoy, "Probability propagation," *Ann. Mat. Artif. Intell.*, vol. 2, pp. 327–352, 1990.
[53] S. E. Shimony, "Finding MAPS for belief networks is NP-hard," *Artif. Intell.*, vol. 68, pp. 399–410, 1994.
[54] J. Seghers, "On the free distance of turbo codes and related product codes," Ph.D. dissertation, Swiss Fed. Inst. Technol., Zurich, Switzerland, Aug. 1995, Final Rep., Diploma Project SS 1995.
[55] P. Smyth, D. Heckerman, and M. Jordan, "Probabilistic independence networks for hidden Markov probability models," *Neural Computation*, accepted for publication.
[56] G. Solomon and H. C. A. van Tilborg, "A connection between block and convolutional codes," *SIAM J. Appl. Math.*, vol. 37, pp. 358–369, Oct. 1979.
[57] M. Sipser and D. A. Spielman, "Expander codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1710–1722, Nov. 1996.
[58] D. J. Spiegelhalter and S. L. Lauritzen, "Sequential updating of conditional probabilities on directed graphical structures," *Networks*, vol. 20, pp. 579–605, 1990.
[59] D. J. Spiegelhalter, A. P. Dawid, S. L. Lauritzen, and R. G. Cowell, "Bayesian analysis in expert systems," *Statist. Sci.*, vol. 8, pp. 219–283, 1993.
[60] D. A. Spielman, "Linear-time encodable and decodable error-corecting codes." *IEEE Trans. Inform. Theory*, vol. 42, pp. 1723–1731, Nov. 1996.
[61] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 533–547, Sept. 1981.
[62] G. Ungerboeck, "Nonlinear equalization of binary signals in Gaussian noise," *IEEE Trans. Commun. Technol.*, vol. COM-19, pp. 1128, Dec. 1971.
[63] S. Verdu and H. V. Poor, "Abstract dynamic programming models under commutativity conditions," *SIAM J. Contr. Optimiz.*, vol. 25, pp. 990–1006, July 1987.
[64] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, Apr. 1967.
[65] J. Whittaker, *Graphical Models in Applied Multivariate Statistics*. Chichester, U.K.: Wiley, 1990.
[66] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *Europ. Trans. Telecommun.* vol. 6, pp. 513–526, Sept.–Oct. 1995.
[67] N. Wiberg, "Codes and decoding on general graphs," Linköping Studies in Sci. and Technol., dissertations no. 440. Linköping, Sweden, 1996.

Robert J. McEliece (M'70–SM'81–F'84) was born in Washington, DC, in 1942. He received the B.S. and Ph.D. degrees in mathematics from the California Institute of Technology, Pasadena, in 1964 and 1967, respectively, and attended Trinity College, Cambridge University, U.K., during 1964–1965.

From 1963 to 1978, he was employed by the California Institute of Technology's Jet Propulsion Laboratory, where he was Supervisor of the Information Processing Group from 1971 to 1978. From 1978 to 1982, he was a Professor of Mathematics and Research Professor at the Coordinated Science Laboratory, University of Illinois, Urbana–Champaign. Since 1982, he has been on the faculty at Caltech, where he is now the Allen E. Puckett Professor of Electrical Engineering. Since 1990, he has also served as Executive Officer for Electrical Engineering at Caltech. He has been a regular consultant in the Communications Research Section of Caltech's Jet Propulsion Laboratory since 1978. His research interests include deep-space communication, communication networks, coding theory, and discrete mathematics.

David J. C. MacKay was born in Stoke on Trent, U.K., on April 22, 1967. Following his education at Newcastle-under-Lyme School and Trinity College, Cambridge, he received the Ph.D. degree in computation and neural systems from the California Institute of Technology, Pasadena, in 1991.

He is now a Lecturer in the Department of Physics, Cambridge University and a Fellow of Darwin College, Cambridge. His interests include the construction and implementation of hierarchical Bayesian models that discover patterns in data, the development of probablistic methods for neural networks, and the design and decoding of error correcting codes.

Jung-Fu Cheng was born in Taipei, Taiwan, in March 1969. He received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1991 and 1993, respectively, and the Ph.D. degree in electrical engineering with a subject minor in social science from the California Institute of Technology, Pasadena, in 1997.

His academic research interests focused on coding and communications theory. Since July 1997, he has been employed as a Research Analyst in the Fixed Income Research Department of Salomon Brothers, Inc., New York, NY.