To appear in *IEEE J. Selected Areas in Communication.*
Draft of August 13, 1997 11:06 a.m.

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

Robert J. McEliece
California Institute of Technology

David J. C. MacKay
Cambridge University

Jung-Fu Cheng
California Institute of Technology

Abstract.

In this paper we will describe the close connection between the now celebrated iterative turbo decoding algorithm of Berrou, Glavieux, and Thitimajshima, 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.

* This work was supported by NSF grant no. NCR-9505975, AFOSR grant no. 5F49620-97-1-0313, and by a grant from Qualcomm, Inc. A portion of McEliece's contribution was done while he was visiting the Sony corporation in Tokyo. The collaboration between MacKay and McEliece was begun at, and partially supported by, the Newton Institute for Mathematical Sciences, Cambridge, England.

1. Introduction and Summary.

"Turbo codes," which were introduced in 1993 by Berrou, Glavieux, and Thitimajshima [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 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 2 we derive some simple but important results about, and introduce some compact notation for, "optimal symbol decision" decoding algorithms. In Section 3 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 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 4, we change gears and give a tutorial overview of the general probabilistic inference problem, with special

reference to *Bayesian belief networks*. In Section 5 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 6, we show that the turbo decoding algorithm follows from a routine application of Pearl's algorithm to the appropriate (loopy) belief network. In Section 7 we briefly sketch some other decoding algorithms that can be derived from BP considerations. Finally in Section 8 we summarize our findings and venture some conclusions.

2. 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 $\boldsymbol{U} = (U_1, \ldots, U_k)$ 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 $p(\boldsymbol{y} \mid \boldsymbol{x}) \stackrel{\text{def}}{=} \Pr\{\boldsymbol{Y} = \boldsymbol{y} \mid \boldsymbol{X} = \boldsymbol{x}\}$, 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

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

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

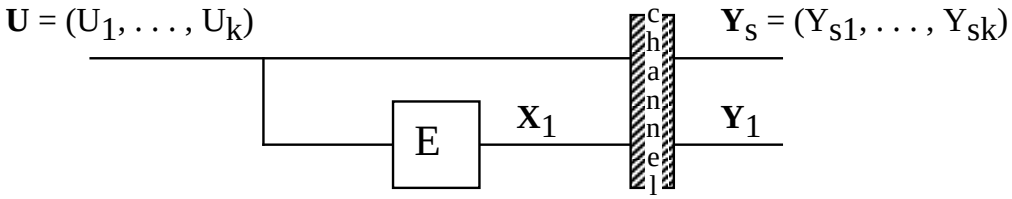

Figure 1. The codeword $\boldsymbol{X} = (\boldsymbol{U}, \boldsymbol{X}_1)$ is transmitted over a memoryless channel and received as $\boldsymbol{Y} = (\boldsymbol{Y}_s, \boldsymbol{Y}_1)$.

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:

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

(A communication theorist would use the term "a posteriori probability," rather than "belief.") If a_0 is such that $\text{BEL}_i(a_0) > \text{BEL}_i(a)$, 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].

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

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

means that $x_i = y_i/(\sum_{k=1}^{m} y_k)$, for $i = 1, \ldots, m$. 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.)

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

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

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

$$\begin{aligned} \Pr\{U_i = a \mid \boldsymbol{Y} = \boldsymbol{y}\} &= \frac{\Pr\{\boldsymbol{Y} = \boldsymbol{y}, U_i = a\}}{\Pr\{\boldsymbol{Y} = \boldsymbol{y}\}} \\ &= \alpha\ \Pr\{\boldsymbol{Y} = \boldsymbol{y}, U_i = a\} \qquad \text{(using the } \alpha\text{-notation)} \\ &= \alpha \sum_{\mathrm{u}\,:u_i=a} p(\boldsymbol{y}, \mathrm{u}) \\ &= \alpha \sum_{\mathrm{u}\,:u_i=a} p(\boldsymbol{y} \mid \mathrm{u}) \cdot p(\boldsymbol{u}) \end{aligned}$$

[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 1.

$$= \alpha \sum_{\mathrm{u}:u_i=a} p(\boldsymbol{y}_1 \mid \boldsymbol{x}_1) p(\boldsymbol{y}_s \mid \boldsymbol{u}) \cdot \prod_{j=1}^{k} \pi_j(u_j) \qquad \text{by (2.2)}$$

$$= \alpha \sum_{\mathrm{u}:u_i=a} p(\boldsymbol{y}_1 \mid \boldsymbol{x}_1) \cdot \prod_{j=1}^{k} \lambda_j(u_j)\pi_j(u_j) \qquad \text{by (2.3)}$$

$$= \alpha\ \lambda_i(a)\pi_i(a) \sum_{\mathrm{u}:u_i=a} p(\boldsymbol{y}_1 \mid \boldsymbol{x}_1) \prod_{\substack{j=1 \\ j\neq i}}^{k} \lambda_j(u_j)\pi_j(u_j).$$

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 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 5, where we shall use it to describe Pearl's algorithm—see Table 5.1, line 6.)

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

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

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 $\boldsymbol{g}' = (g'_1, \ldots, g'_k)$, where g'_i is defined by

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

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* $\boldsymbol{h} = \boldsymbol{f}\boldsymbol{g}$ as a simple componentwise product, i.e., $\boldsymbol{h} = (h_1, \ldots, h_k)$, 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

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

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

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.