Iterative Decoding of Compound Codes by Probability Propagation in Graphical Models

Frank R. Kschischang, *Member, IEEE* and Brendan J. Frey

Abstract—**We present a unified graphical model framework for describing compound codes and deriving iterative decoding algorithms. After reviewing a variety of graphical models (Markov random fields, Tanner graphs, and Bayesian networks), we derive a general distributed marginalization algorithm for functions described by factor graphs. From this general algorithm, Pearl's belief propagation algorithm is easily derived as a special case. We point out that recently developed iterative decoding algorithms for various codes, including "turbo decoding" of parallel-concatenated convolutional codes, may be viewed as probability propagation in a graphical model of the code. We focus on Bayesian network descriptions of codes, which give a natural input/state/output/channel description of a code and channel, and we indicate how iterative decoders can be developed for parallel- and serially-concatenated coding systems, product codes, and low-density parity-check codes.**

I. Introduction

COMPOUND codes are codes composed of a collection of interacting constituent codes, each of which can be decoded tractably. In this paper, we describe various graphical models that can be used not only to describe a wide variety of compound codes, but also to derive a variety of iterative decoding algorithms for these codes. Prominent among compound codes are the *turbo codes* introduced by Berrou, Glavieux and Thitimajshima [1], in which the constituent convolutional codes interact in "parallel concatenation" through an interleaver. It is probably fair to say that the near-capacity error-rate performance of turbo codes has sparked much of the current interest in iterative decoding techniques, as evidenced by this special issue. Other examples of compound codes include classical serially concatenated codes [2] (see also [3, 4]), Gallager's low-density parity-check codes [5], and various product codes [6, 7].

In [8, 9], we observed that iterative decoding algorithms developed for these compound codes are often instances of probability propagation algorithms that operate in a graphical model of the code. These algorithms have been developed in the past decade in the expert systems literature, most notably by Pearl [10] and Lauritzen and Spiegelhalter [11]. (See [12–14] for textbook treatments on probability or "belief" propagation algorithms, and [15] for an extensive treatment of graphical models.)

The first to connect Pearl's "belief propagation" algorithm with coding were MacKay and Neal [16–18], who showed that Gallager's 35-year-old algorithm [5] for decoding low-density parity-check codes is essentially an instance of Pearl's algorithm. Extensive simulation results of MacKay and Neal show that Gallager codes can perform nearly as well as turbo codes, indicating that we probably "sailed" much closer to capacity 35 years ago than might have been appreciated in the interim. MacKay, McEliece and Cheng [19, this issue] have also independently observed that turbo decoding is an instance of "belief" propagation. They provide a description of Pearl's algorithm, and make explicit the connection to the basic turbo decoding algorithm described in [1].

Recently, and independently of developments in the expert systems literature, Wiberg, Loeliger and Kötter in [20] and Wiberg in his doctoral dissertation [21], have refocused attention on graphical models for codes. They show that a type of graphical model called a "Tanner graph" (first introduced by Tanner [22] to describe a generalization of Gallager codes) provides a natural setting in which to describe and study iterative soft-decision decoding techniques, much as the code trellis [23] is an appropriate model in which to describe and study "conventional" maximum-likelihood soft-decision decoding using the Viterbi algorithm. Forney [24] gives a nice description of the history of various "two-way" algorithms and their connections with coding theory.

In this paper we seek to unify this recent work by developing a graphical model framework that can be used to describe a broad class of compound codes and derive corresponding iterative decoding algorithms. In Section II, we review and relate various graphical models, such as Markov random fields, Tanner graphs, and Bayesian networks. These graphs all support the basic probability propagation algorithm, which is developed in Section III in the general setting of a "factor graph," and in Section IV for the specific case of a Bayesian network.

Given a graphical code model, probability propagation can be used to compute the conditional probability of a message symbol given the observed channel output. For richly connected graphs containing cycles, *exact* probability propagation becomes computationally infeasible, in which case iterative decoding can still yield excellent results. The basic iterative decoding algorithm proceeds as if no cycles were present in the graph, with no guarantee that the computed "conditional probabilities" are close to the correct values, or that they even converge! Nevertheless, the excellent performance of turbo codes and Gallager codes is testimony to the efficacy of these iterative decoding procedures.

In Section V we describe Bayesian network models for a variety of compound codes, and describe how probability propagation can be used to decode these codes. As it is a straightforward exercise to develop Bayesian network models for many coding schemes such as multilevel codes and coset codes, and also for channels more general than the usual memoryless channels, we believe that there are many possibilities for application of iterative decoding techniques beyond what has been described in the literature to date.

Manuscript received Sept. 27, 1996; revised July 28, 1997.

F. R. Kschischang is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario M5S 3G4, Canada. B. J. Frey was with the Department of Electrical and Computer Engineering, University of Toronto. He is now with the Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, 405 North Mathews Avenue, Urbana IL 61801.

Publisher Item Identifier S 0733-8716(98) 00xxx-x.

II. GRAPHICAL CODE MODELS

In this section we present several graph-based models that can be used to describe the conditional dependence structure in codes and channels. Given a set $U = \{v_1, \ldots, v_N\}$ of random variables with joint probability distribution $p(v_1, \ldots, v_N)$, a graphical model attempts to capture the conditional dependency structure inherent in this distribution, essentially by expressing how the distribution factors as a product of "local functions" (e.g., conditional probabilities) involving various subsets of U. Graphical models are useful for describing the structure of codes, and are the key to "probability propagation" and iterative decoding.

A. Markov Random Fields

A Markov random field (see, e.g., [25]) is a graphical model based on an undirected graph $G = (V, E)$ in which each vertex corresponds to a random variable, i.e., an element of U. Denote by $n(v)$ the neighbors of $v \in V$, i.e., the set of vertices of V connected to v by a single edge of E. The graph G is a *Markov random field* (MRF) if the distribution $p(v_1, \ldots, v_n)$ satisfies the local Markov property: $(\forall v \in V)\ \ p(v|V\backslash\{v\}) = p(v|n(v))$. In other words, G is an MRF if every variable v is independent of non-neighboring variables in the graph, given the values of its immediate neighbors. MRFs are well developed in statistics, and have been used in a variety of applications (see, e.g., [25–28]).

The joint probability mass (or density) function for the vertices of a MRF G is often expressed in terms of a Gibbs *potential function* defined on the maximal cliques of G. A *clique* in G is a collection of vertices which are all pairwise neighbors, and such a clique is *maximal* if it is not properly contained in any other clique. Corresponding to each clique q in the set of maximal cliques Q is a collection of vertices V_q that are contained in q. Denote by S_v the sample space for the random variable v. Given a nonnegative potential function for each clique $q \in Q$, i.e., a function $\psi_q : \prod_{v \in V_q} S_v \to \mathbb{R}^+ \cup \{0\}$, the joint probability mass (or density) function over $V = \{v_1, \ldots, v_N\}$ is given by

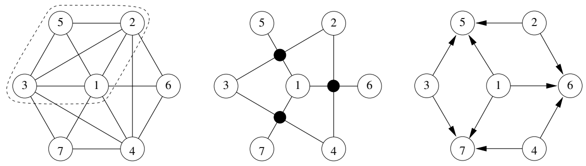
$$p(v_1, \ldots, v_N) = Z^{-1} \prod_{q \in Q} \psi_q(\{v \in V_q\}), \tag{1}$$

where Z^{-1} is a normalizing constant, assuming that the product in (1) is not everywhere zero. It is possible to define an MRF in terms of potential functions defined over all cliques in G, not just the maximal cliques, but any potential function defined over a non-maximal clique q can be absorbed into the potential function defined over the maximal clique containing q.

From the structure of the potential functions, it is a straightforward exercise (see, e.g., [25]) to show that the resulting probability distribution satisfies the local Markov property. Indeed, every strictly positive MRF can be expressed in terms of a Gibbs potential, although the proof of this result (given, e.g., in [26, ch. 1]) is less straightforward. Lauritzen [15, pp. 37–38] gives an example due to Moussouris of a non-strictly positive MRF satisfying the local Markov property for which it is impossible to express the joint distribution as a product of potentials as in (1).

To illustrate how MRFs can be used to describe codes, consider the MRF with seven binary variables shown in Fig. 1(a). There are four maximal cliques: $q_1 = \{1, 2, 3, 5\}$ (dashed loop), $q_2 = \{1, 2, 4, 6\}$, $q_3 = \{1, 3, 4, 7\}$, and $q_4 = \{1, 2, 3, 4\}$. From (1), the joint probability distribution for $v_1, \ldots, v_7$, can be written as a product of Gibbs potential functions defined over the variable subsets indicated by these four cliques. This MRF can be used to describe a Hamming code by setting $\psi_{q_4} = 1$ (which is equivalent to neglecting q_4) and by letting the first three potentials be even parity indicator functions. That is, $\psi_q(\cdot) = 1$ if its arguments form a configuration with even parity and 0 otherwise. The MRF places a uniform probability distribution on all configurations that satisfy even parity in cliques q_1, q_2 and q_3, and zero probability on configurations not satisfying these parity relations.

While the potential functions chosen in this example define a linear code, it is clear that such potential functions can be used to determine a code satisfying any set of local check conditions. In particular, given a set of variables $U = \{v_1, \ldots, v_N\}$, let Q be a collection of subsets of U. Corresponding to each element E of Q, a *local check condition* enforces structure on the variables contained in E, by restricting the values that these variables can assume. (For example, the check condition could enforce even parity, as in the example above.) By defining an indicator function for each local check condition that assumes the value unity for valid configurations and zero for invalid configurations, and by defining a graph in which each element of Q forms a clique, an MRF description that assigns a uniform probability distribution over the valid configurations is obtained, provided that at least one valid configuration exists. As we shall see, a Tanner graph is another way to represent the same local check structure.



(a) (b) (c)

Fig. 1. Graphical models for the (7,4) Hamming code: (a) a Markov random field, with a maximal clique indicated; (b) a Tanner graph; and (c) a Bayesian network.

B. Tanner Graphs

Tanner graphs were introduced in [22] for the construction of good long error-correcting codes in terms of shorter codes. Our treatment follows the slightly different presentation of Wiberg, *et al.* [20].

A Tanner graph is a *bipartite* graph representation for a check structure, similar to the one described above. In such a graph, there are two types of vertices corresponding to the variables and the "checks", respectively, with no edges connecting vertices of the same type. For example, a Tanner graph corresponding to the Hamming code described above is shown in Fig. 1(b). Each check vertex q in the set of check vertices Q is shown as a filled circle. In this case, a check vertex ensures that its set of neighbors satisfies even parity in a valid configuration.

We see that the check vertices play precisely the same role in a Tanner graph as do the maximal cliques in an MRF. In general, for each check vertex q with neighbors $n(q)$, we can associate a non-negative real-valued potential function $\psi_q(\{v \in n(q)\})$, that assigns positive potential only to valid configurations of its arguments. We then write a probability distribution over the variables as

$$p(v_1, \ldots, v_N) = Z^{-1} \prod_{q \in Q} \psi_q(\{v \in n(q)\}), \tag{2}$$

where Z^{-1} is a normalizing constant. Of course, (2) is analogous to (1).

An MRF can be converted to a Tanner graph by introducing a check vertex for each maximal clique, with edges connecting that check vertex to each variable in the clique. The potential function assigned to the check vertex would be the same as that assigned to the clique.

A Tanner graph can be converted to an MRF by eliminating the check vertices and forming cliques from all variables originally connected to the same check vertex. The potential associated with the clique would be the same as that assigned to the check vertex. It is possible that some new cliques may be formed in this process, which are not associated with a check vertex of the Tanner graph. A unit potential is assigned to these "induced" cliques. Different Tanner graphs may map to the same MRF, hence Tanner graphs may be more specific about dependencies than MRFs. For example, the graph in Fig. 1(b) with an additional check vertex connected to v_1, v_2, v_3 and v_4 will also map to the MRF in Fig. 1(a).

C. Bayesian Networks

We now introduce Bayesian networks that, unlike MRFs and Tanner graphs, are *directed acyclic* graphs [12]. A directed acyclic graph is one where there are no graph cycles when the edge directions are followed (though there may be cycles when the edge directions are ignored). As in an MRF, a random variable is associated with each graph vertex. Given a directed graph $G = (V, E)$, let the *parents* (or direct ancestors) $a(v)$ of vertex v be the set of vertices of V that have directed edges connecting *to* v. For a Bayesian network, the joint probability distribution can be written

$$p(v_1, \ldots, v_N) = \prod_{i=1}^{N} p(v_i | a(v_i)), \tag{3}$$

where, if $a(v_i) = \varnothing$ (*i.e.*, v_i has no parents) then we take $p(v_i|\varnothing) = p(v_i)$.

Every distribution can be described by a Bayesian network, since by the chain rule of probability,

$$\begin{aligned} p(v_1, \ldots, v_N) &= p(v_1)p(v_2|v_1)p(v_3|v_1, v_2) \times \cdots \\ &\quad \times p(v_N|v_1, v_2, \ldots, v_{N-1}). \end{aligned}$$

It follows that we can pick any ordering of the variables and then condition each variable on all variables that precede it. However, this trivial network does not capture any useful probabilistic structure because the last factor, $P(v_N|v_1, v_2, \ldots, v_{N-1})$, contains all N variables and so is really just as complicated as the full joint distribution.

A Bayesian network for the Hamming code described above is shown in Fig. 1(c). The joint distribution is obtained from (3) using parent-child relationships:

$$\begin{aligned} P(v_1, \ldots, v_7) &= P(v_1)P(v_2)P(v_3)P(v_4)P(v_5|v_1, v_2, v_3) \\ &\quad \times P(v_6|v_1, v_2, v_4)P(v_7|v_1, v_3, v_4). \end{aligned}$$

The first four factors express the prior probabilities of $v_1, \ldots, v_4$, while the last three factors capture the parity checks: e.g., $P(v_5|v_1, v_2, v_3) = 1$ if v_1, v_2, v_3, and v_5 have even parity and 0 otherwise.

A Tanner graph (and by extension, an MRF) can be converted into a Bayesian network simply by directing edges towards the check vertices. A binary $\{0, 1\}$ indicator random variable is introduced at each check site q_i such that $p(q_i|a(q_i)) = 1$ only if the random variables in the set $a(q_i)$ satisfy the constraint checked by the corresponding vertex in the Tanner graph.

A potential advantage of Bayesian networks is that the directed edges (arrows) can be used to model causality explicitly. By inspecting the arrows in such models, it is easy to determine which variables directly influence others. This often makes it possible to *simulate* the network, i.e., draw a configuration of variables consistent with the distribution specified by the network. One simply draws a configuration for variables having no parents, consistent with the (prior) distribution affecting those variables. Once a configuration has been drawn for all parents $a(v)$ of a variable v, a configuration for v can be drawn consistent with the conditional probability $p(v|a(v))$. For example, in Fig. 1(c), we simply pick values for the parent-less vertices v_1, v_2, v_3, and v_4 and then determine the remainder of the codeword, v_5, v_6, and v_7. This explicit representation of causality is also useful for modeling physical effects, such as channel noise and intersymbol interference.

It should be noted that simulating a Bayesian network can become a hard problem when variables v for which $a(v) \neq \varnothing$ are required to take on a specific value, i.e., when some child variables are "clamped." For example, drawing a configuration of variables consistent with the observed output of a channel is essentially as hard as (or harder than) decoding. Similarly, when a Tanner graph is converted into a Bayesian network in the manner described above, it may be difficult to draw a valid configuration of the variables, as all indicator variables have a nonempty set of parents and all are required to take on the value one.

In coding, the relationships between the information symbols $\mathbf{u}$, the encoder state variables $\mathbf{s}$ (if there are any), the transmitted codeword symbols $\mathbf{x}$, and the received signals $\mathbf{y}$ completely define the encoding and decoding problem for a given code. Without loss of generality, these relationships can be expressed probabilistically and depicted pictorially using graphical models. The Bayesian network for channel coding in general is shown in Fig. 2. By inspection of the network, the joint distribution is

$$P(\mathbf{u}, \mathbf{s}, \mathbf{x}, \mathbf{y}) = P(\mathbf{u})P(\mathbf{s}|\mathbf{u})P(\mathbf{x}|\mathbf{u}, \mathbf{s})p(\mathbf{y}|\mathbf{x}).$$

Usually, $P(\mathbf{u})$ is a uniform distribution and $P(\mathbf{s}|\mathbf{u})$ and $P(\mathbf{x}|\mathbf{u}, \mathbf{s})$ are deterministic (i.e., all probability mass is placed

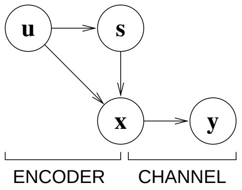
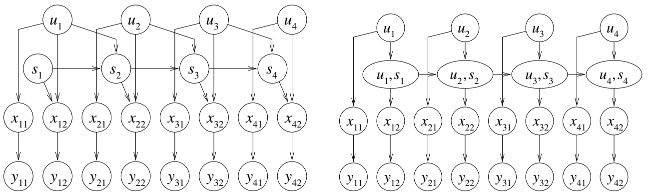


Fig. 2. The general Bayesian network for channel coding.

on a single outcome). The *channel likelihood* $p(\mathbf{y}|\mathbf{x})$ expresses the noise and inter-symbol interference introduced by the channel.

Fig. 3(a) shows the Bayesian network for a systematic convolutional code with a memoryless channel. The systematic codeword symbols x_{k1} are simply copies of the information symbols u_k. The other codeword symbols are outputs of the encoder; x_{k2} depends on u_k *and* state s_k. By inspecting the parents of the received signals, we find that $P(\mathbf{y}|\mathbf{x}) = \prod_k P(y_{k1}|x_{k1})P(y_{k2}|x_{k2})$ which expresses the absence of memory in the channel. Fig. 3(b) shows a cycle-free network for the same code, obtained by grouping information and state variables together. This eliminates undirected cycles at the expense of increasing the complexity of some of the network variables.



(a) (b)

Fig. 3. (a) The Bayesian network for a systematic convolutional code and a memoryless channel. (b) A cycle-free connected network for the same code and channel.

Further examples of Bayesian networks for codes will be discussed in Section V. In the next section, we will describe the basic distributed marginalization algorithm that will form the basis for iterative decoding.

III. A FRAMEWORK FOR DISTRIBUTED MARGINALIZATION

In this section we develop the basic "probability propagation" algorithm that can be used to compute marginal probabilities in graphical models, given some observations. A common feature of the graphical models described in the previous section is that they can be used to describe a "global" joint probability distribution as a product of "local" functions. The computation of a conditional probability then amounts essentially to a "marginalization" of the this global function. Using the structure of the local functions, it may be possible to greatly simplify this computation, as we now show. A derivation along similar lines has also been carried out recently by Aji and McEliece [29], who also develop an algorithm for "information distribution" on a graph.

A. Notation

We begin by introducing some notation. Let I be a finite index set, and let $\{A_k : k \in I\}$ be a collection of finite sets called *symbol alphabets*, indexed by I. The *configuration space* W is defined as the Cartesian product of symbol alphabets, $W = \prod_{k\in I} A_k$, and elements of W are called configurations. For $J \subset I$, let $W|_J$ denote the projection of W onto the coordinates indexed by J, so that $W|_J = \prod_{k\in J} A_k$, which is taken to be empty when J is empty. For a configuration $\mathbf{x} \in W$, and nonempty J, we denote by $\mathbf{x}|_J$ the image of $\mathbf{x}$ under this projection. We denote the complement of J relative to I as J^c. By abuse of notation, we equate the pair $(\mathbf{x}|_J, \mathbf{x}|_{J^c})$ with $\mathbf{x}$, though formally some re-ordering of coordinates may be necessary for this equality strictly to hold.

A function $Z : W \to R$ over the set of configurations is said to be a *global function*. Initially we assume that the codomain R is the set of real numbers, but later we will allow R to be an arbitrary commutative semiring [21, 29–31].

It will often be useful to introduce *families* of global functions, indexed by a set of finite-dimensional real-valued *parameters* $\mathbf{y}$, which are fixed in any instance of distributed marginalization. In this case, we write $Z(\mathbf{x}; \mathbf{y})$ for the value the function assumes at configuration $\mathbf{x}$. Introducing such parameters allows us to take into account the influence of continuous-valued variables such as channel outputs. However, for notational convenience, we will sometimes omit the explicit dependence on $\mathbf{y}$.

For a set $J \subset I$, we define the *marginal function* $Z_J : W|_J \to R$ with respect to J as

$$Z_J(\mathbf{x}|_J) = \sum_{\mathbf{x}|_{J^c} \in W|_{J^c}} Z(\mathbf{x}|_J, \mathbf{x}|_{J^c}).$$

In other words, the value of the marginal function with respect to J at the point $\mathbf{x}|_J$ is obtained by summing the global function over all configurations that *agree* with $\mathbf{x}|_J$ in the coordinates indexed by J. Any variable *not* indexed by J is said to be *marginalized out* in Z_J. Note that $Z_\varnothing$ is the constant obtained by summing over all configurations of variables, while $Z_I = Z$. We have chosen the symbol Z for the global function, as we view $Z_\varnothing$ as a "*Zustandssumme*" (a sum-over-states), i.e., a partition function as in statistical physics (see, e.g., [32, p. 13]).

If the function Z is the joint probability mass function of a collection of random variables indexed by I, then Z_A is the marginal joint probability mass function for the random variables indexed by A, and $Z_\varnothing = 1$. Re-introducing the parameter $\mathbf{y}$, suppose the function $Z(\mathbf{x}; \mathbf{y})$ is the conditional joint probability mass function of a collection of random variables given the observation of continuous-valued random vector $\mathbf{y}$. Then the marginal functions represent conditional probability mass functions. For example, $Z_{\{i\}}(\mathbf{x}|_{\{i\}}; \mathbf{y}) = P(x_i|\mathbf{y})$, the conditional probability mass function for x_i given the observed value of $\mathbf{y}$. Such formulations will often be useful in decoding problems, when the continuous-valued output of a noise channel is observed.

When $|I|$, the number of arguments of Z, is small, we will sometimes use a modified notation for the marginal functions. We replace an argument x_i of Z with a '+' sign to indicate that the corresponding variable is to be summed over, i.e., marginalized out. Thus if $|I| = 4$, $Z(x_1, +, +, +) = Z_{\{1\}}(x_1)$,

$Z(x_1,+,x_3,+) = Z_{\{1,3\}}(x_1,x_3)$, $Z(+,+,+,+) = Z_\varnothing$, and so on.

It will often be useful to marginalize some variables while holding other variables constant, for example, in the case of computing a conditional probability mass function given that some variables are observed. Since the key operation in the computation of a marginal function or in the computation of a conditional probability is *marginalization*, we shall focus attention on developing efficient algorithms for this operation.

B. *Local Functions and Factor Graphs*

The key to efficient marginalization is to take into account any structure that the global function Z possesses. Suppose that Z is "separable," i.e., that Z can be written as the product of a number of *local functions*, each a function of the variables contained in a subset of I. More precisely, let $A_1,\ldots,A_N$ be a collection of nonempty subsets of I, and suppose

$$Z(\mathbf{x}) = \prod_{j=1}^{N} \psi_j(\mathbf{x}|_{A_j}). \tag{4}$$

The functions $\psi_j : W|_{A_j} \to R$ are called local functions.

For example, suppose that X_1, X_2, X_3 are random variables forming a Markov chain (in that order) given a specific observation $\mathbf{Y} = \mathbf{y}$. (For example, these random variables might represent the state sequence of a convolutional code in successive time intervals, and $\mathbf{Y}$ might represent the corresponding channel output.) The conditional joint probability mass function can be written as

$$p(x_1,x_2,x_3|\mathbf{y}) = p(x_1|\mathbf{y})p(x_2|x_1,\mathbf{y})p(x_3|x_2,\mathbf{y}).$$

Translating to the notation of this section, and observing that a conditional probability mass function $p(x_{i+1}|x_i,\mathbf{y})$ is essentially a function of two variables (since $\mathbf{y}$ is a constant) we write

$$Z(x_1,x_2,x_3) = \psi_1(x_1)\psi_2(x_1,x_2)\psi_3(x_2,x_3). \tag{5}$$

We will have occasion to consider products of local functions. For example, in (5) the product of $\psi_2(x_1,x_2)$ and $\psi_3(x_2,x_3)$ is a function of three variables that we denote $\psi_2\psi_3(x_1,x_2,x_3)$. We will also apply the '+'-sign notation to local functions and their products.

It will be useful to display a particular factorization of the global function by means of a bipartite graph graph called a *factor graph*. Suppose $Z(\mathbf{x})$ factors as in (4). A factor graph $G = (V,E)$ is a bipartite graph with vertex set $V = I \cup \{A_j : 1 \le j \le N\}$. The only edges in E are those that connect a vertex $i \in I$ to a vertex A_j if and only if $i \in A_j$, i.e., $E = \{\{i, A_j\} : i \in A_j\}$. In words, each vertex of a factor graph G corresponds to either a variable or a local function. An edge joins a variable x to a local function ψ if and only if x appears as an argument of ψ. For example, Fig. 4 shows the factor graph corresponding to the Markov chain (5). Note that a factor graph is essentially a generalization of a Tanner graph, in which local "checks" involving the incident variables have been replaced with local functions involving the incident variables.

It is a straightforward exercise to convert the various graphical models described in Section II into a factor graph representation. A Markov random field G that expresses a Gibbs potential function yields a factor graph with one local function vertex for every maximal clique, i.e., a local function vertex for every factor in (1). A Tanner graph directly yields a factor graph by associating with each check vertex a binary indicator function that indicates whether the local check condition is satisfied. More generally, each factor of (2) can be associated with a local function vertex, as in the MRF case. Finally, a Bayesian network is converted into a factor graph by introducing a local function vertex for every factor of (3), and a variable vertex for each variable. Clearly the local function vertex associated with $p(v_i|a(v_i))$ is connected by an edge to the variable vertices in the set $\{v_i\} \cup a(v_i)$. Thus, a Bayesian network with N variables yields a factor graph with $2N$ vertices.

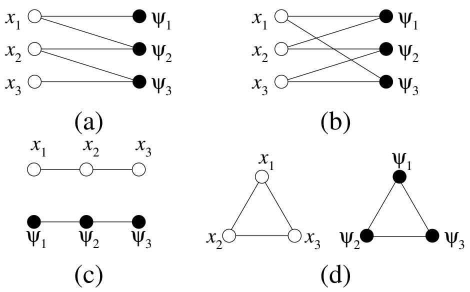


Fig. 4. Factor graphs for (a) a Markov chain, and (b) a loopy example, and their corresponding second higher power graphs (c) and (d), omitting self-loops.

C. *Marginalization by Message Passing*

Our aim is to derive a graph-based algorithm for computing the marginal functions $Z_{\{i\}}(\mathbf{x}|_{\{i\}})$ for all $i \in I$. The functions ψ_i are called "local" functions because we assume that access to these functions is local to a specific vertex in the graph. Knowledge of local functions (or functions derived from local functions) can be propagated to non-local vertices by "message passing" along the edges of the graph. The "messages" are descriptions of (possibly marginalized) local function products. To illustrate what we mean by this, and to motivate the general situation in which this "graph-based message-passing" paradigm is made more precise, we consider the specific case where $Z(x_1,x_2,x_3)$ is defined in (5).

Consider the computation of $Z(x_1,+,+) = p(x_1|y)$. We write

$$\begin{aligned} Z(x_1,+,+) &= \sum_{x_2}\sum_{x_3} \psi_1(x_1)\psi_2(x_1,x_2)\psi_3(x_2,x_3) \\ &= \psi_1(x_1)\underbrace{\sum_{x_2}\psi_2(x_1,x_2)\underbrace{\sum_{x_3}\psi_3(x_2,x_3)}_{\psi_3(x_2,+)}}_{\psi_2\psi_3(x_1,+,+)}, \end{aligned} \tag{6}$$

where we have identified the various factors that need to be computed to obtain $Z(x_1,+,+)$. Our primary observation is that $Z(x_1,+,+)$ can be computed knowing just $\psi_1(x_1)$ and $\psi_2\psi_3(x_1,+,+)$. The latter factor can be computed knowing just $\psi_2(x_1,x_2)$ and $\psi_3(x_2,+)$.

Analyzing the computation of the remaining marginal functions in the same manner, we find that

$$Z(+, x_2, +) = \psi_3(x_2, +) \sum_{x_1 \in S_1} \psi_1(x_1)\psi_2(x_1, x_2) \tag{7}$$

$$Z(+, +, x_3) = \sum_{x_2 \in S_2} \psi_1\psi_2(+, x_2)\psi_3(x_2, x_3) \tag{8}$$

Examining (6)–(8), we see that all marginal functions can be computed recursively from a chain of local function products, which we view as *messages* passed between the vertices of the "propagation tree" shown in Fig. 5. Comparing with (6)–(8), we observe that the information passed to the vertex associated with ψ_i, $1 \leq i \leq 3$, is precisely that needed to compute the marginal function for x_i, and so we choose that vertex as a "fusion site" for the variable x_i.

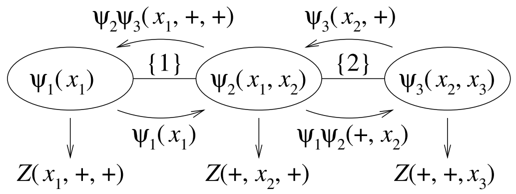


Fig. 5. Computation of marginal functions by message passing. The messages are descriptions of (possibly marginalized) local function products, and are passed along the graph edges as indicated by the arrows. (The arrows themselves are not part of the graph.)

We consider now the general case. Let $G = (V, E)$ be a factor graph, describing the way in which the global function $Z(\mathbf{x})$ factors as a product of local functions as in (4). Consider the second higher power graph G^2, defined as having the same vertex set as G, but with edges connecting two vertices if and only if there is a path of length two between the vertices in G. Self loops are ignored. Since G is bipartite, G^2 will always split into at least two disconnected components, G_v^2 having vertices associated with variables and G_f^2 having vertices associated with local functions. For example, Fig. 4(c) and (d) shows the second higher power graphs associated with the factor graphs shown in (a) and (b).

Observe that in G_v^2, two variables will be joined by an edge if they are both arguments of the same local function, so all arguments of a particular local function form a clique. When the local functions of the factor graph correspond to Gibbs potential functions, then G_v^2 is the corresponding Markov random field. In other words, an MRF can be recovered as a component of the second higher power graph (omitting self-loops) of the corresponding factor graph.

Consider now G_f^2, which we call the *propagation graph* corresponding to G. We assume that G_f^2 consists of a single connected component; if not, marginalization can be carried out independently in the various components. The vertices of G_f^2 correspond to local functions. Two vertices are joined by an edge for each argument that the corresponding local functions have in common, though we will collapse multiple edges between two vertices to a single edge. A description of the general message-passing algorithm is simplified by imagining that a vertex is an active processing element, capable of receiving and transmitting messages (i.e., marginalized local function products) along the edges incident on the vertex and capable of performing computations involving messages and the local function associated with the vertex.

We now describe a general distributed marginalization algorithm that operates in a *tree* spanning the propagation graph; we refer to this spanning tree as a *propagation tree*. Given a factor graph G, we must

1. specify a spanning tree T for G_f^2, and
2. identify a "fusion vertex" in T for each marginal function to be computed.

Note that T is in general quite *different* from the graphical model (MRF, Tanner graph, or Bayesian network) from which T is derived. In general it may be a difficult problem to choose T optimally so as, e.g., to minimize overall computational complexity. For now we assume that T is chosen in some (arbitrary) manner.

For a given T, we say that a variable x_i is *involved* at a vertex of T if x_i is an argument of the corresponding local function. Let e be an edge of T. We say that a given variable x_i must be *carried* over e if e is part of a path that joins any vertex in which x_i is involved with the fusion vertex for x_i. In essence, x_i must be carried over an edge of the subtree of T that *spans* the fusion vertex for x_i and the other vertices in which x_i is involved. Outside of this subtree, only marginal knowledge of x_i is needed, and hence x_i can be marginalized out. Given a propagation tree T, and an assignment of fusion vertices, it is easy to determine which variables must be carried over any given edge in T. (For example, each edge of the trees shown in Figs. 5 and 6 is labeled with the indices of the variables to be carried over that edge.)

The size of the messages sent over an edge is greatly influenced by the number of variables that must be carried over the edge, and by the number of possible values that each such variable can assume. A simplistic measure of complexity associated with an edge is its *thickness*, defined as the number of variables that are to be carried over that edge. (A more useful measure would be the product of the sizes of the symbol alphabets corresponding to these variables, or the size of a minimal description of the corresponding local function product.) It may be desirable to find a propagation tree and an assignment of fusion vertices so that the maximum thickness is minimized, but we conjecture that this is a hard problem in general. (Given a propagation tree, the maximum thickness is minimized if the fusion vertex for a variable x_i is a vertex in the subtree of T that spans the vertices in which x_i is involved, so the problem is to find a suitable propagation tree.)

To illustrate that it is not always possible to achieve unit maximum thickness, consider the global function $Z(x_1, x_2, x_3) = \psi_1(x_1, x_2)\psi_2(x_2, x_3)\psi_3(x_1, x_3)$. The factor graph and its second higher power graph are shown in Fig. 4(b) and (d). By symmetry, there is essentially only one propagation tree for this function, as shown in Fig. 6. Numbering the vertices in the figure from 1 to 3 (left to right), we choose vertex i as a fusion vertex for x_i. We observe that x_1 must be carried over both of the edges in the propagation tree, while x_2 and x_3 each need to be carried over only one edge, as indicated in Fig. 6. The thickness of each edge is two, and no assignment of fusion vertices can reduce this number.

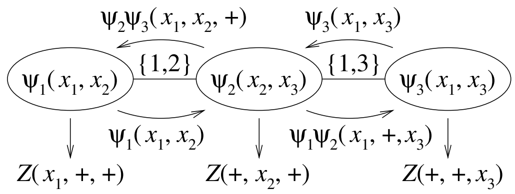


Fig. 6. Another example of marginalization by message passing. Sets of indices identifying the variables carried along each edge are shown.

D. Marginalization Algorithms

The fundamental idea behind marginalization algorithms is to compute the product of local functions (i.e., messages) marginalizing out unnecessary variables. We now describe two versions of a general message-passing algorithm that can be used to compute marginal functions. In the first version (the two-way schedule) messages are passed once in each direction along each edge in the propagation tree. This is the minimum possible, and for this reason, the two-way schedule is best suited for serial implementation.

The Two-Way Schedule
Let T be a propagation tree. We allow each vertex of T to be in one of way two states: "inbound" or "outbound." Initially all vertices are placed in the inbound state.
1. (The inbound state.) At a vertex in the inbound state, wait until messages have arrived on all edges but one, and call this remaining edge the "prime" edge. Compute the product of all incoming messages with the local function, marginalize out any variables *not* to be carried on the prime edge, and send the result on the prime edge. Toggle to the "outbound" state.
2. (The outbound state.) At a vertex in the outbound state, wait until a message arrives on the prime edge. For each non-prime edge e, compute the product of all incoming messages with the local function, *except* the message on e. Marginalize out any variables not to be carried on e and pass the result on e. If this vertex is a fusion vertex, the desired marginal function(s) can be computed as the product of all incoming messages with the local function, marginalizing out any undesired variables.

Observe that only a single edge is incident on any leaf vertex, and so leaf vertices are not required to "wait" in the inbound state. Thus message passing is initiated at the leaf vertices. Messages propagate from the leaves to the interior of the graph, and then back towards the leaves. The algorithm terminates when all leaf vertices have *received* a message. Since each edge is used to convey exactly one "inbound" message and exactly one "outbound" message, the total number of messages transmitted is twice the number of propagation tree edges, namely $2(N-1)$.

In the second version of the algorithm (the flooding schedule), nodes do not necessarily "wait" before passing messages, and hence messages may pass more than once in a given direction along an edge in the propagation tree. The flooding schedule is better suited for parallel implementation, since more than the minimum number of messages will usually be passed.

The Flooding Schedule
1. (Initialization) At any collection of vertices, carry out the "flooding" procedure described in the next step, with a unit message arriving on a fictitious edge.
2. (Flooding) When a message is received on any *any* edge e at a vertex, define this message as the "current" message for e, designate e as "incoming" and all other edges as "outgoing." For each outgoing edge e' compute the product of the local function with all current messages *except* the current message on e', and marginalize out any variables not to be carried on e', passing the result on e'. For a (leaf) vertex with no outgoing edges, simply "absorb" the message received.
3. (Marginalization) The algorithm terminates when no vertices have more messages to pass, at which point marginalization at each fusion vertex in the same way as in the two-way schedule. Indeed, marginalization can take place at any time during the previous step to yield a "current estimate" of the marginal function.

The basic principle behind the flooding schedule is that the receipt of a message at a vertex "triggers" that vertex to send messages on all other edges. At a leaf vertex, there is no "other" edge, so a received message is absorbed. Since the propagation tree has no cycles, eventually all messages are absorbed at the leaf vertices.

It is clear that both the two-way schedule and the flooding schedule are versions of the same algorithm, the difference being only the order in which messages are propagated. Indeed, one can also devise hybrid schedules. When a message arrives on an edge at a vertex, that message effectively creates "pending" messages on all other edges incident on that vertex. These pending messages need not be sent immediately; indeed, the pending messages may change to reflect the content of other messages arriving at the vertex. In general, a wide variety of such schedules are possible, of which the two-way and flooding schedules represent two extremes. In our simulations of the flooding schedule for a turbo decoder [33], we observed that, compared with the standard message-passing schedule, several orders of magnitude *more* messages are passed, but if the messages are passed concurrently, then several orders of magnitude *fewer* time-steps are required to achieve a given decoding performance.

Note that, regardless of the message passing schedule, message passing in a tree is guaranteed to converge to a state in which no messages are pending. At this point marginalization at each fusion vertex can be carried out, with exactly the same result for each different message-passing schedule.

E. Generalization to Commutative Semirings

Up to now we have made the assumption that the global function Z is real-valued. However, the only properties of the reals of which we have made use are essentially the commutativity of multiplication and the distribution of multiplication over addition. Hence, the distributed marginalization algorithm described will work over any commutative semiring (see, e.g., [30, Sec. 3.2] and [21, 29, 31]). For example, replacing the real-valued product operation with summation, and the summation operation with the max operator will yield a generalization of the Viterbi algorithm equivalent to Pearl's "belief revision" algorithm [12, Sec. 5.3] and the "min-sum" algorithm described by Wiberg, *et al.* [20]. We conjecture that many of the distributed algorithms used in data networks for routing, network topology determination, etc., such as the distributed Bellman-Ford algorithm (see, e.g., [34, Ch. 5]) are instances of this general marginalization algorithm in an appropriately defined semiring.

IV. Probability Propagation in Bayesian Networks

We now specialize the general message-passing algorithm described in the previous section to the special case of Bayesian networks. We begin with the special case of a Bayesian network that (neglecting the direction of edges) has no cycles, i.e., is a tree.

A. The Cycle-Free Case

Let B be a Bayesian network on L random variables, $X_1, \ldots, X_L$. By definition of a Bayesian network (3), the joint probability mass function of these random variables can be written as $p(x_1, \ldots, x_L) = \prod_{i=1}^{L} p(x_i|a(X_i))$. In other words, the joint probability mass function can be written as a product of L local functions, each representing the conditional probability mass function for a variable given its parents.

In this subsection, we assume that B forms a tree, i.e., that it contains no undirected cycles. In this case, it is easy to see that the corresponding factor graph G will also form a tree, as will the propagation graph G_f^2. Indeed, in this case G_f^2 is isomorphic to an undirected version of B, in which the vertex corresponding to the variable X_i is associated with the local function $p(x_i|a(X_i))$. We choose this vertex as the fusion site for X_i. For example, Fig. 7(a) shows a simple cycle-free Bayesian network B. The corresponding factor graph and propagation tree are shown in Fig. 7(b) and (c), respectively.

For simplicity, we refer to the vertex of the propagation tree into which X_i translates as vertex x_i. If X_j is a child (parent) of X_i in the Bayesian network, then we refer to x_j as a child (parent) of x_i in the propagation tree, even though the edges of the propagation tree are undirected.

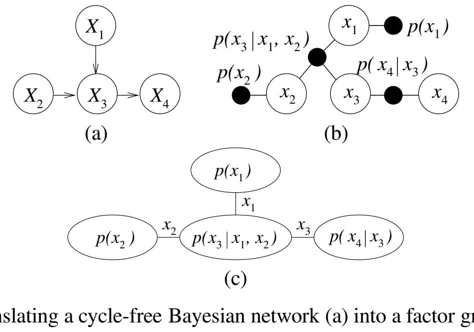


Fig. 7. Translating a cycle-free Bayesian network (a) into a factor graph (b) and a propagation tree (c).

Observe that a variable x_i in the propagation network is involved with vertex x_i, and all of the children of this vertex. Thus, in the propagation network, x_i must be carried along all edges connecting vertex x_i to its children, but these are the *only* edges along which x_i must be carried. Since a given edge connects precisely one parent to one child, precisely one variable must be carried along that edge, namely the parent variable. In other words, a cycle-free Bayesian network yields a propagation tree of unit thickness. This observation is the key to Pearl's "belief propagation" algorithm.

We now determine the messages that must be propagated. Following Pearl [12], we denote child-to-parent messages as λ messages, and parent-to-child messages as π messages, so that, e.g., $\lambda_{x_i}(x_j)$ and $\pi_{x_k}(x_i)$, are, respectively, messages transmitted from vertex x_i to its parent x_j and to its child x_k. Note that a message is always a function of the parent variable.

Consider a vertex x_i, and suppose the children of x_i are indexed by the set K, and the parents $a(x_i)$ of x_i are indexed by the set J, written explicitly as $J = \{j_1, j_2, \ldots, j_{|J|}\}$.

For $k \in K$, the message sent by x_i to its child x_k is given by

$$\pi_{x_k}(x_i) = \prod_{k' \in K \setminus \{k\}} \lambda_{x_{k'}}(x_i) \sum_{x_{j_1}} \cdots \sum_{x_{j_{|J|}}} \prod_{j \in J} \pi_{x_i}(x_j) p(x_i|a(x_i)).$$

For $j \in J$, the message sent by x_i to its parent x_j is given by

$$\lambda_{x_i}(x_j) = \sum_{x_i} \Bigg[\prod_{k \in K} \lambda_{x_k}(x_i) \times \underbrace{\sum_{x_{j_1}} \cdots \sum_{x_{j_{|J|}}}}_{\text{omit } x_j} \prod_{j' \in J \setminus \{j\}} \pi_{x_i}(x_{j'}) p(x_i|a(x_i)) \Bigg].$$

The conditional probability mass function for x_i, given the set of messages received (which we denote by $\mathbf{O}$ for "observations") is given by

$$p(x_i|\mathbf{O}) = \prod_{k \in K} \lambda_{x_k}(x_i) \times \sum_{x_{j_1}} \cdots \sum_{x_{j_{|J|}}} \prod_{j \in J} \pi_{x_i}(x_j) p(x_i|a(x_i)).$$

Note that

$$\pi_{x_k}(x_i) = p(x_i|\mathbf{O}) / \lambda_{x_k}(x_i). \qquad (9)$$

Although these expressions may at first glance seem complicated, they are really a simple application of the general propagation rule, which states that an outgoing message sent on edge e is computed as the product of all incoming messages (except that on edge e) with the local function, with any variables not to be carried on e marginalized out. For Bayesian networks common in coding applications, these propagation updates are quite simple.

In applications of Bayesian networks, it will often be useful to include variables that are continuous-valued so as to model, for example, channel outputs. While strictly speaking continuous-valued variables do not enter the framework as we have described it (except through the parameter $\mathbf{y}$), we will allow our Bayesian networks to have continuous-valued variables, provided that such variables are observed and that the corresponding vertices have no children. Since y is restricted to be childless, the problems of describing a continuous conditional density function (needed for a π message sent from y) is avoided. In our diagrams of Bayesian networks, continuous-valued observed vertices will be shown as filled circles.

The complexity of probability propagation in a Bayesian network (using the two-way schedule) depends on the manner in which the messages are represented. Assuming that each message is a vector of values of size given by the size of the parent's sample space, the computational complexity can be estimated as follows. Denote the set of children of a vertex v as $d(v)$ and let $|d(v)|$ denote the size of this set. Recall that the parents of v are denoted $a(v)$. Let $|v, a(v)|$ denote the number of nonzero entries in the table corresponding to the local function at v, and let $|v|$ denote the size of the sample space for v.

Each λ message sent to a parent of v requires the computation of the product of the incoming λ messages, which requires on the order of $|v| \cdot |d(v)|$ operations, together with $|a(v)| \cdot |v, a(v)|$ operations, since we must sum over $|v, a(v)|$ nonzero products of $|a(v)|$ factors to marginalize the product of the incoming π messages. Each π message sent to a child of v can be computed from (9) using approximately $|d(v)| \cdot |v|$ operations. The total number of operations at v is approximately $2|v| \cdot |d(v)| + |a(v)|^2|v, a(v)|$, which is usually dominated by the second term. We conclude, therefore, that the total number of operations χ performed by the two-way schedule for a Bayesian network with vertex set V scales as $\chi = \mathcal{O}(\sum_{v \in V} |a(v)|^2|v, a(v)|)$.

B. Probability Propagation in Bayesian Networks with Cycles

The tactic of deriving a propagation tree from the structure of the Bayesian network itself does not apply to Bayesian networks with (undirected) cycles, because clearly the Bayesian network does not form a tree. In this subsection we show with a small example that exact probability propagation is possible, but only at the expense of greater than unit thickness of the propagation tree. While this may be an acceptable tradeoff for simple networks, it may not be acceptable in complicated networks.

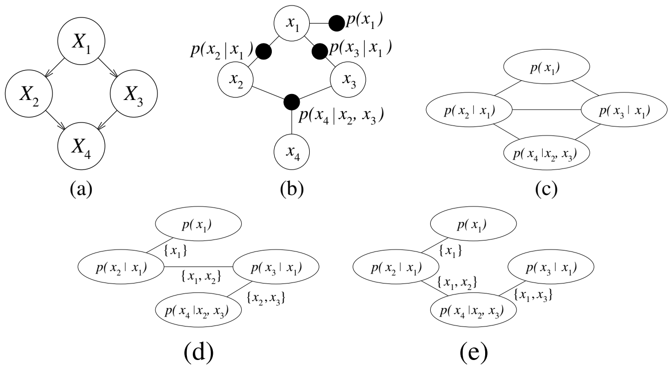


Fig. 8. In a Bayesian network with cycles (a), with factor graph (b), and propagation graph (c), the corresponding propagation trees must have maximum thickness at least two. Two examples are shown in (d) and (e).

Consider the Bayesian network shown in Fig. 8(a). Two possible propagation trees for this Bayesian network are shown in Figs. 8(d) and (e), each which has thickness two. For this Bayesian network, it is not possible to achieve a propagation tree of unit thickness.

V. Compound Codes

In Section I, we defined a *compound code* to be one which can be described by the interaction between constituent codes, each of which is tractably decodable on its own. Graphically, each constituent code is represented by a cycle-free constituent Bayesian network. These constituent networks share some variables, so that taken as a whole, the total Bayesian network is not cycle-free. In general, there does not exist a unique decomposition of a compound code into its constituent codes, but the compound code is usually designed using well-known constituent codes.

A. Bayesian Networks for Some Known Codes

The Bayesian network for a systematic rate 1/3 compound turbo code is shown in Fig. 9(a), along with its cycle-free constituent networks. This compound code consists of two chain-type networks that are connected using a different ordering of the information symbol vertices. Whereas the upper chain directly uses the information sequence $\mathbf{u}$, the lower chain uses a permuted sequence, obtained by applying an interleaver. The systematic codeword component has been included with the upper constituent code. Wiberg, *et al.* [20, 21] were probably the first to describe turbo codes using this type of graphical model.

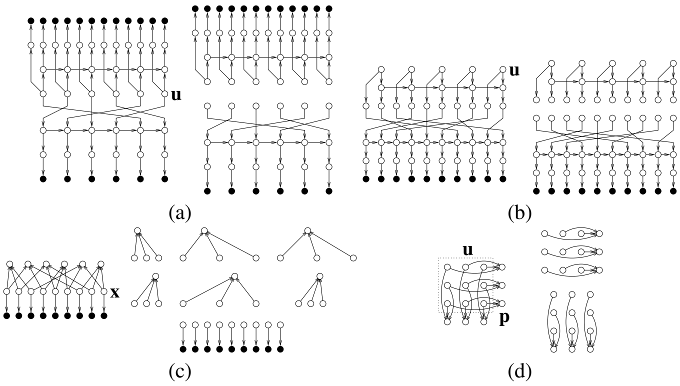


Fig. 9. Bayesian networks for three compound codes and their constituent codes: (a) A systematic rate 1/3 turbo code; (b) A rate 1/2 "serially concatenated" convolutional code; (c) A low-density parity check code; (d) a (15,9) product code.

A "serially concatenated" convolutional compound code was proposed by Benedetto and Montorsi [3]. Their system is essentially the same as Forney's concatenated codes [2], with convolutional inner and outer codes. The Bayesian network for a non-systematic rate 1/2 compound code of this sort is shown in Fig. 9(b), along with its two cycle-free constituent networks. Notice that the output of the outer convolutional code (including the systematic part) is used as the input to the inner convolutional code, via an interleaver. Only the output of the inner convolutional code is transmitted over the channel.

Fig. 9(c) shows an example of a Gallager low-density parity check code [5]. This code consists of parity check restrictions on subsets of the codeword symbols, denoted by $\mathbf{x}$. (The mapping from information blocks to codewords is not directly specified.) Each variable in the row of variables above the codeword symbol row is a binary indicator, which takes on the nonzero value only for configurations of its parents that have even parity. The channel outputs are indicated below the row of codeword symbols. MacKay and Neal [16] were the first to describe Gallager's codes using Bayesian networks.

Fig. 9(d) shows an example of a simple product code, in which a single parity bit checks the parity of each row and column of the information bit array (enclosed in the dotted box). The channel output symbols are not shown.

B. Iterative Decoding: Probability Propagation for Compound Codes

Since each constituent Bayesian network in a compound code is cycle-free, probabilities for each random variable given the observed random variables can be efficiently computed *exactly* within each constituent network. However, because the compound network has cycles, probability propagation is only an approximate algorithm for computing these probabilities. As we see it, the general idea of iterative decoding is to make use of the efficient probability propagation algorithm for each constituent network, while either ignoring the cycles or somehow taking them into account. This graphical framework unifies several iterative decoding algorithms. The turbo decoding algorithm, the "separable MAP filter" algorithm [6], the new iterative decoding algorithm [4] used for decoding "serially concatenated" convolutional codes, and, as first pointed out by MacKay and Neal [16], Gallager's algorithm for decoding low-density parity check codes [5] are all a form of probability propagation in the compound code networks shown in Fig. 9.

The overall decoding procedure essentially consists of applying probability propagation while ignoring the graph cycles. The procedure can be broken down into a series of processing cycles. In each cycle, probabilities are propagated across a particular constituent network, producing current estimates of the distributions over information symbols, state variables and codeword symbols, given the observed channel output. The next cycle then uses the probability estimates produced by the previous cycle when processing the next constituent network. Usually, the constituent codes are processed in order and one pass through all of the codes is called an *iteration*. An iteration essentially consists of propagating probabilities across the network as if it were cycle-free, stopping when each vertex has been processed *once*. In fact, because the compound code network has cycles, the propagation procedure actually never self-terminates. Usually the cyclic procedure is allowed to iterate until some termination criterion is satisfied. Then, the information symbols are detected, usually in the fashion of the maximum *a posteriori* (MAP) symbol probability decoding rule.

C. Turbo Decoding: Probability Propagation for Turbo Codes

For example, as shown explicitly in [19, this issue], this probability propagation algorithm, when applied to turbo codes, *is* the standard turbo decoding algorithm. The turbo decoding algorithm uses the forward-backward algorithm [35,36] (or an approximation to it) to process each constituent trellis. The algorithm uses "extrinsic information" [1, 7] produced by the previous step, when processing the next trellis. This is the information that is passed from one trellis to the other through the information symbols. In probability propagation terminology, extrinsic information is the set of parent-child probability messages that are passed down from the information symbols to one constituent network, in response to the child-parent messages received from the other network.

Fig. 10 shows the message passing dynamics for a simplified turbo code Bayesian network. When the channel output is observed (as shown by the filled discs), messages propagate up to the state vertices of both constituent networks, creating pending messages on the incident edges (as indicated by black dots in the figure). These messages are the codeword symbol likelihoods as determined by the channel model and the channel output. Each constituent network is processed one at a time in the manner of the forward-backward algorithm. The information symbols receive probability messages from the constituent network just processed. In the case of the single code, the information symbols are each connected by a single edge and so propagation terminates. In this case, however, a set of messages are passed to the other constituent network (these messages are the "extrinsic information".) Since there will always be messages pending, the overall procedure repeats the basic "iteration" shown in the pictures outlined by the dashed polygon.

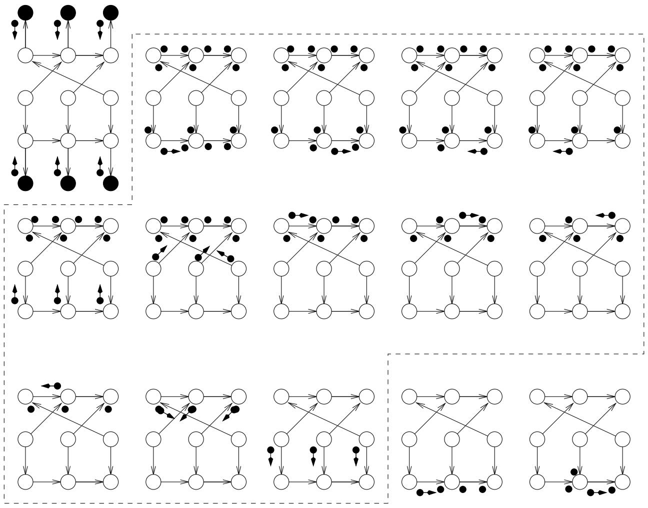


Fig. 10. Probability propagation in the Bayesian network for a turbo code. The dashed polygon encloses the steps in a single iteration. Black dots without arrows represent pending messages.

VI. Conclusions

In this paper we have attempted to unify various recently developed themes in iterative decoding. We have reviewed graphical codes models, including Markov random fields, Tanner graphs and Bayesian networks, all of which encode the "local" probabilistic structure of codes and channels.

We have developed a distributed marginalization algorithm in the general setting of a factor graph. Given a function of discrete variables that can be written as a product of local potential functions, marginalization can be carried out by a message-passing procedure in a propagation tree, derived from the second higher power of the factor graph. The "thickness" of an edge in this tree is equal to the number of variables that must be carried over this edge to perform exact marginalization. In a cycle-free Bayesian network, the network itself—when used as a propagation tree—achieves a maximum edge thickness of unity. This observation is the key to Pearl's "belief" or probability propagation algorithm, which computes the *a posteriori* distribution *exactly* in a cycle-free Bayesian network.

For compound codes, however, the Bayesian networks are not cycle-free. Nevertheless, the networks are broken into tractable subnetworks, each describing a constituent code and in which probability propagation can be applied. Iterating over these constituent decoders can result in excellent decoding performance in practice, as demonstrated by Berrou, *et al.* [1]. We have shown that many recently proposed iterative decoders can be described as message passing in a graphical code model.

In general, it is a straightforward exercise to develop Bayesian network models for many coding schemes, such as multilevel codes and coset codes, and also for channels more general than the usual memoryless channels. We believe that there are many possibilities for application of iterative decoding techniques beyond what has been described in the literature to date.

ACKNOWLEDGMENTS

We are grateful for useful comments on an earlier version of this paper made by the reviewers and by G. D. Forney, Jr. and H.-A. Loeliger. The concept of factor graphs as a generalization of Tanner graphs was devised by a group at ISIT '97 in Ulm that included the authors, H.-A. Loeliger, D. J. C. MacKay, N. Wiberg, and R. M. Tanner. We are grateful for discussions on this topic with S. M. Aji, G. D. Forney, Jr., R. Kötter, and R. J. McEliece.

REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. IEEE Int. Conf. Commun. (ICC)*, (Geneva, Switzerland), pp. 1064–1070, 1993.

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

[3] S. Benedetto and G. Montorsi, "Serial concatenation of block and convolutional codes," *Electronics Letters*, vol. 32, pp. 887–888, 1996.

[4] S. Benedetto and G. Montorsi, "Iterative decoding of serially concatenated convolutional codes," *Electronics Letters*, vol. 32, pp. 1186–1188, 1996.

[5] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: M.I.T. Press, 1963.

[6] J. Lodge, R. Young, P. Hoeher, and J. Hagenauer, "Separable MAP 'filters' for the decoding of product and concatenated codes," in *Proceedings of IEEE International Conference on Communications*, pp. 1740–1745, 1993.

[7] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, pp. 429–445, 1996.

[8] B. J. Frey and F. R. Kschischang, "Probability propagation and iterative decoding," in *Proc. 34th Annual Allerton Conf. on Communication, Control, and Computing*, (Allerton House, Monticello, Illinois), pp. 482–493, October 1996.

[9] B. J. Frey, *Bayesian Networks for Pattern Classification, Data Compression and Channel Coding*. Toronto, Canada: Department of Electrical and Computer Engineering, University of Toronto, 1997. Doctoral dissertation available at `http://www.cs.utoronto.ca/~frey`.

[10] J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artificial Intelligence*, vol. 29, pp. 241–288, 1986.

[11] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *J. of the Royal Statistical Society, Series B*, vol. 50, pp. 157–224, 1988.

[12] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA: Morgan Kaufmann, 1988. Revised second printing.

[13] R. E. Neapolitan, *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. Toronto: John Wiley & Sons, 1990.

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

[15] S. L. Lauritzen, *Graphical Models*. Oxford University Press, 1996.

[16] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *Cryptography and Coding. 5th IMA Conference* (C. Boyd, ed.), no. 1025 in Lecture Notes in Computer Science, pp. 100–111, Berlin Germany: Springer, 1995.

[17] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 18, pp. 1645–1646, 1996. Reprinted in vol. 33, March 1997, pp. 457–458.

[18] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices." Submitted to *IEEE Transactions on Information Theory*, 1997.

[19] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, "Turbo decoding as an instance of Pearl's 'belief propagation' algorithm." To appear in *IEEE J. Selected Areas in Commun.*, Jan. 1998.

[20] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *European Trans. on Telecommun.*, vol. 6, pp. 513–525, Sep./Oct. 1995.

[21] N. Wiberg, *Codes and Decoding on General Graphs*. PhD thesis, Linköping University, Sweden, 1996.

[22] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. on Inform. Theory*, vol. IT-27, pp. 533–547, Sept. 1981.

[23] G. D. Forney, Jr., "Trellises old and new," in *Communications and Cryptography: Two Sides of One Tapestry* (R. E. Blahut, D. J. Costello, Jr., U. Maurer, and T. Mittelholzer, eds.), pp. 115–128, Kluwer Academic Publishers, 1994.

[24] G. D. Forney, Jr., "The forward-backward algorithm," in *Proc. 34th Annual Allerton Conf. on Communication, Control, and Computing*, (Allerton House, Monticello, Illinois), pp. 432–446, October 1996.

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

[26] C. J. Preston, *Gibbs States on Countable Sets*. Cambridge University Press, 1974.

[27] V. Isham, "An introduction to spatial point processes and Markov random fields," *Int. Stat. Rev.*, vol. 49, pp. 21–43, 1981.

[28] G. E. Hinton and T. J. Sejnowski, "Learning and relearning in Boltzmann machines," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D. E. Rumelhart and J. L. McClelland, eds.), vol. I, pp. 282–317, Cambridge MA.: MIT Press, 1986.

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

[30] S. Verdú and H. V. Poor, "Abstract dynamic programming models under commutativity conditions," *SIAM J. on Control and Optimization*, vol. 25, pp. 990–1006, July 1987.

[31] R. J. McEliece, "On the BJCR trellis for linear block codes," *IEEE Transactions on Information Theory*, vol. 42, pp. 1072–1092, July 1996.

[32] E. Schrödinger, *Statistical Thermodynamics*. Cambridge University Press, 1962.

[33] B. J. Frey, F. R. Kschischang, and P. G. Gulak, "Concurrent turbo-decoding," in *Proc. 1997 IEEE Int. Symp. on Inform. Theory*, (Ulm, Germany), p. 431, 1997.

[34] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, New Jersey: Prentice-Hall, 2nd ed., 1992.

[35] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state Markov chains," *Annals of Mathematical Statistics*, vol. 37, pp. 1559–1563, 1966.

[36] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, vol. 20, pp. 284–287, 1974.

Frank R. Kschischang is an Associate Professor in the Department of Electrical and Computer Engineering at the University of Toronto. During the 1997/98 academic year, he is a Visiting Scientist at the Massachusetts Institute of Technology. He received his B.A.Sc. with honors in electrical engineering from the University of British Columbia in 1985, and his M.A.Sc. and Ph.D. in electrical engineering from the University of Toronto in 1988 and 1991, respectively. Dr. Kschischang's research interests are focussed on the area of coding techniques, primarily on soft-decision decoding algorithms, trellis structure of codes, and iterative decoders. He teaches graduate courses in information theory, coding and communication theory, and currently is an IEEE TRANSACTIONS ON INFORMATION THEORY Associate Editor for Coding Theory.

Brendan Frey received his Ph.D. in electrical and computer engineering from the University of Toronto in 1997, where he was a member of the Neural Networks Research Group and was an NSERC 1967 Science and Engineering Scholar. Currently he is a Fellow of the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign. He researches interesting and approximate solutions to hard problems in digital communication and machine learning. In particular, he is interested in applying graphical models and their associated algorithms in these areas.