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.**

***Index Terms*— Concatenated coding, decoding, graph theory, iterative methods, product 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 *et al.* [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] and [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. McEliece *et al.* [19] 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 *et al.* 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.

Manuscript received September 27, 1996; revised May 8, 1997 and July 29, 1997.

F. R. Kschischang is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont., Canada, on leave at the Massachusetts Institute of Technology, Cambridge, MA 02138 USA.

B. J. Frey is with the Beckman Institute, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA.

Publisher Item Identifier S 0733-8716(98)00225-X.

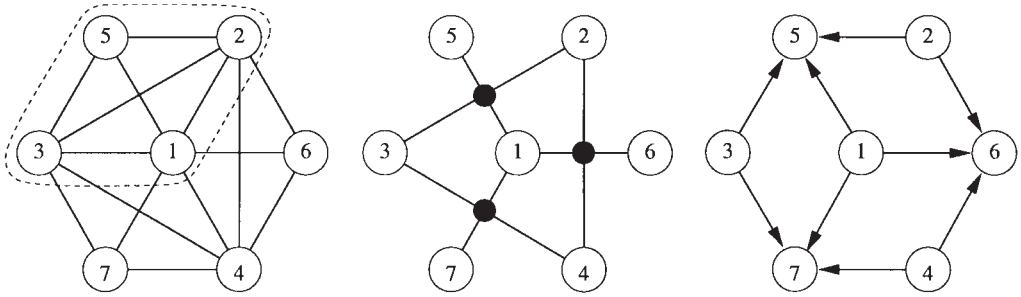

Fig. 1. Graphical models for the (7, 4) Hamming code. (a) Markov random field with a maximal clique indicated. (b) Tanner graph. (c) Bayesian network.

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.

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 $U = \{v_1, \cdots, v_N\}$ of random variables with joint probability distribution $p(v_1, \cdots, v_N)$, 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 $p(v_1, \cdots, v_n)$ 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 nonneighboring variables in the graph, given the values of its immediate neighbors. MRF's 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 $\psi_q : \prod_{v \in V_q} S_v \to \mathbb{R}^+ \cup \{0\}$, the joint probability mass (or density) function over $V = \{v_1, \cdots, v_N\}$ is given by

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

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 nonmaximal 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 nonstrictly 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 MRF's 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 $v_1, \cdots, v_7$, 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 $U = \{v_1, \cdots, v_N\}$, 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

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.

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 nonnegative 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

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

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 MRF's. 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 MRF's 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 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

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

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,

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

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 $P(v_N|v_1, v_2, \cdots, v_{N-1})$ 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

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

The first four factors express the prior probabilities of $v_1, \cdots, v_4$, 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 toward 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 parentless 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 $a(v) \neq \varnothing$

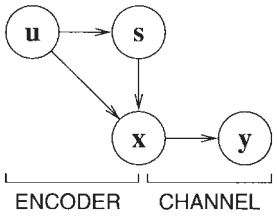

Fig. 2. General Bayesian network for channel coding.

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 among the information symbols $\boldsymbol{u}$, the encoder state variables $\boldsymbol{s}$ (if there are any), the transmitted codeword symbols $\boldsymbol{x}$, and the received signals $\boldsymbol{y}$ 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

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

Usually, $P(\boldsymbol{u})$ is a uniform distribution and $P(\boldsymbol{s}|\boldsymbol{u})$ and $P(\boldsymbol{x}|\boldsymbol{u},\boldsymbol{s})$ are deterministic (i.e., all probability mass is placed on a single outcome). The *channel likelihood* $p(\boldsymbol{y}|\boldsymbol{x})$ expresses the noise and intersymbol 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 $P(\boldsymbol{y}|\boldsymbol{x}) = \prod_k P(y_{k1}|x_{k1})P(y_{k2}|x_{k2})$ 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.

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 this global function. Using 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 $\{A_k\colon\ k \in I\}$ 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 $\boldsymbol{x} \in W$ and nonempty J, we denote by $\boldsymbol{x}|_J$ the image of $\boldsymbol{x}$ under this projection. We denote the complement of J relative to I as J^c. By abuse of notation, we equate the pair $(\boldsymbol{x}|_J, \boldsymbol{x}|_{J^c})$ with $\boldsymbol{x}$, although formally, some reordering of coordinates may be necessary for this equality strictly to hold.

A function $Z\colon W \to R$ 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* $\boldsymbol{y}$, which are fixed in any instance of distributed marginalization. In this case, we write $Z(\boldsymbol{x};\boldsymbol{y})$ for the value the function assumes at configuration $\boldsymbol{x}$. 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 $\boldsymbol{y}$.

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

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

In other words, the value of the marginal function with respect to J at the point $\boldsymbol{x}|_J$ is obtained by summing the global function over all configurations that *agree* with $\boldsymbol{x}|_J$ in the coordinates indexed by J. Any variable *not* indexed by J is said to be *marginalized out* in Z_J. Note that $Z_\emptyset$ 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 $Z_\emptyset$ 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 $Z_\emptyset = 1$. Reintroducing the parameter $\boldsymbol{y}$, suppose the function $Z(\boldsymbol{x};\boldsymbol{y})$ is the conditional joint probability mass function of a collection of random variables given the observation of continuous-valued random vector $\boldsymbol{y}$. Then the marginal functions represent conditional probability mass functions. For example, $Z_{\{i\}}(\boldsymbol{x}|_{\{i\}};\boldsymbol{y}) = P(x_i|\boldsymbol{y})$, the conditional probability mass function for x_i given the observed value of $\boldsymbol{y}$. Such formulations will often be useful in decoding problems, when the continuous-valued output of

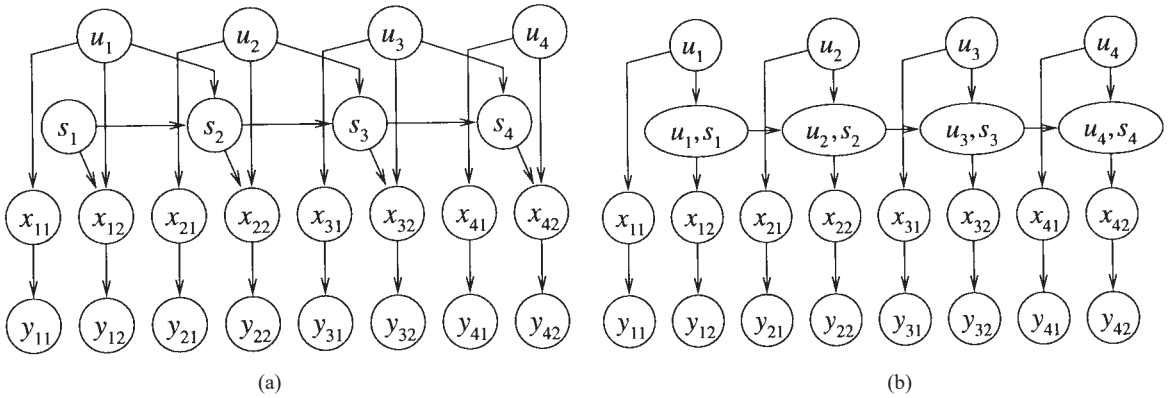

Fig. 3. (a) Bayesian network for a systematic convolutional code and a memoryless channel. (b) Cycle-free connected network for the same code and channel.

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 $A_1, \cdots, A_N$ be a collection of nonempty subsets of I, and suppose

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

The functions ψ_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 $\boldsymbol{Y} = \boldsymbol{y}$. (For example, these random variables might represent the state sequence of a convolutional code in successive time intervals, and $\boldsymbol{Y}$ might represent the corresponding channel output.) The conditional joint probability mass function can be written as

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

Translating to the notation of this section, and observing that a conditional probability mass function $p(x_{i+1}|x_i,\boldsymbol{y})$ is essentially a function of two variables (since $\boldsymbol{y}$ 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 $\psi_2\psi_3(x_1,x_2,x_3)$. We will also apply the "+"-sign notation to local functions and their products.

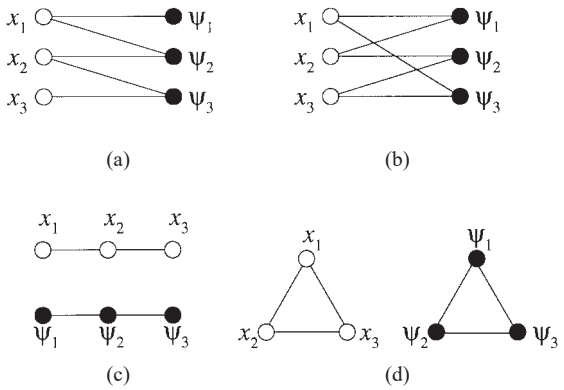

Fig. 4. Factor graphs for (a) a Markov chain, (b) a loopy example, and their corresponding second higher power graphs, (c) and (d), omitting self-loops.

It will be useful to display a particular factorization of the global function by means of a bipartite graph called a *factor graph*. Suppose $Z(\boldsymbol{x})$ factors as in (4). A factor graph $G = (V,E)$ is a bipartite graph with vertex set $V = I \cup \{A_j\colon 1 \le j \le N\}$. 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., $E = \{\{i,A_j\}\colon i \in A_j\}$. 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

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.