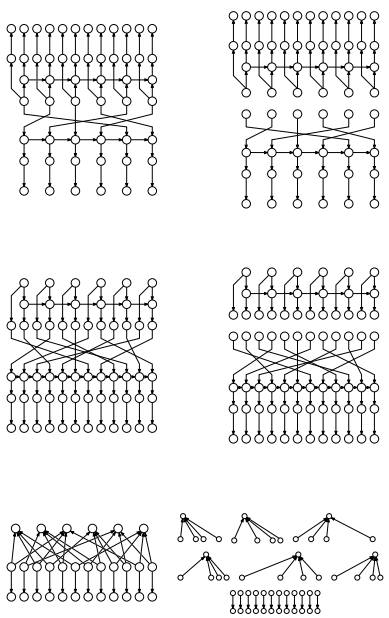
Iterative Decoding of Compound Codes by Probability Propagation in Graphical Models

Frank R. Kschischang
Department of Electrical and Computer Engineering
University of Toronto
frank@comm.utoronto.ca

Brendan J. Frey
Department of Computer Science
University of Toronto
frey@cs.utoronto.ca

September 26, 1996
Revision 1.3



Iterative Decoding of Compound Codes by Probability Propagation in Graphical Models

Frank R. Kschischang
Department of Electrical and Computer Engineering
University of Toronto
frank@comm.utoronto.ca

Brendan J. Frey
Department of Computer Science
University of Toronto
frey@cs.utoronto.ca

September 27, 1996
Revision 1.3

Abstract

This paper is a tutorial on graphical code models and iterative decoding. After reviewing the conditional dependence structure of probability distributions using graphical models (Markov random fields, Tanner graphs, Bayesian networks), this tutorial paper describes the probability propagation algorithm that can be used in such models. Probability propagation is a well-known technique developed over the past decade in the expert systems literature, although the essential elements of the technique were already present in Gallager's decoding algorithm for low-density parity check codes. We show that more 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 indicate how iterative decoders can be developed for parallel- and serial-concatenated coding systems, product codes, and low-density parity-check codes.

1 Introduction

Prompted by the success of the "turbo-codes" introduced by Berrou, Glavieux and Thitimajshima [1], and as evidenced by this special issue, there has been an explosion of recent interest in applying iterative soft-decision decoding techniques and attempting to explain their excellent performance. In this tutorial paper, we seek to unify various themes that underlie much of this recent work.

We observe that the basic computational paradigm underlying these recently developed techniques appears to be "probability propagation in graphical models," a technique developed in the past decade in the expert systems literature, most notably by Pearl [2] and his co-workers (see [3, 4] for textbook treatments). Interestingly, Gallager's algorithm [5] for decoding low-density parity-check codes is an example of an iterative decoding algorithm that also is essentially an instance of "probability propagation" published over 30 years ago, but largely forgotten in the interim, apart from the work of Tanner [6]. Recently, however, Wiberg, Loeliger and Kötter in [7] and Wiberg in his doctoral dissertation [8], and independently of developments in the expert systems literature, have refocussed attention on graphical models for codes. They show that such graphical models provide a natural setting in which to describe and study iterative soft-decision decoding techniques, much as the code trellis [9] is an appropriate model in which to describe and study "conventional" maximum-likelihood soft-decision decoding using the Viterbi algorithm. MacKay and Neal in [10, 11] have also observed that iterative decoding is also essentially an instance of probability or "belief" propagation in a graphical model.

To motivate the general computational paradigm—which is essentially the same paradigm that underpins dataflow machine architectures (e.g., [12, 13, 14])—we start, in Section 2, with a simple example of message passing in a "spy network." We observe that in a singly-connected network, messages passed from a vertex v to a vertex w are intended to "articulate" the state of the sub-network $N_{v\mathbf{A}w}$ reached from w only through v. This notion of "articulation," coupled with conditional independence (or "Markov property") implied by graphical models for codes provide a foundation for the development of probability propagation algorithms.

In Section 3, we review (and make connections among) various graphical code models such as Markov random fields, Tanner graphs, and Bayesian networks. These graphs have the basic Markov property that only a local conditional probability update is needed to articulate the state of a sub-network.

This Markov property then leads naturally to the probability propagation algorithm developed in Section 4. In the case of singly-connected graphs, probability propagation is an exact algorithm that allows one to compute *a posteriori* probability of any random variable in the graph, given what has been observed. When the graph has cycles, iterative probability propagation results, with no guarantee of convergence, but, in some cases (notably the case of turbo-codes), with excellent decoding performance in soft-iterative decoding applications.

In Section 5 we then discuss various applications of these results for the decoding of "compound codes." Webster's 7th dictionary defines *compound* as a "macroscopically homogeneous substance consisting of atoms or ions of two or more different elements in definite proportions, and usually having properties unlike those of its constituent elements." A *compound code* is composed of *constituent codes*, each of which can be decoded tractably by probability propagation on a subgraph of the graph that describes the compound code. For a given codeword, the constituent codewords are generated from the same information block; consequently, the constituent codewords are interdependent and must be decoded interactively, an observation that leads naturally to iterative decoding. Examples of compound codes include the turbo-codes of Berrou *et al.* [1], Gallager's low-density parity-check codes [5], "serial-concatenated" codes [15, 16], and product codes for which iterative decoding algorithms have been developed [17, 18]. 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.

2 Computation by Data Flow

In this section we present a simple example of distributed computation using localized message passing. This computational paradigm underlies so-called dataflow machines [12, 13, 14]. We observe that in a singly connected network, the role of a vertex v is to "articulate" for a neighbour w the "state" of the subnetwork $N_{v\mathbf{A}w}$ reached from w through v. The notation $N_{v\mathbf{A}w}$ is to be read: "the network that v articulates for w."

2.1 A Simple "Spy Network"

In the tradition of spy novels, imagine a "spy network" (or corporate hierarchy) in which the spies are arranged in *cells*. Each cell has a "controller" and a number of "subordinates" and (let us suppose) only the controller knows the identities of the subordinates in any given cell. The network is organized so that the subordinates of a cell can themselves control subcells, subordinates of subcells can control sub-subcells, and so on, in tree-like fashion.

A spy network can be represented as a directed graph with vertices corresponding to spies. For example, Fig. 1 illustrates one possible spy network. Edges in the network are directed from each cell controller to the subordinates in the cell. For example, the dotted lines in Fig. 1(a) enclose the cell with controller vertex labelled "HQ" (which stands for "headquarters"). We say that a controller c "directly controls" his subordinates, and "indirectly controls" any network member that can be reached by a directed path from c. We assume that a given network member can have at most one controller, and that a controller does not indirectly control himself, so that there are no cycles (directed or undirected) in the

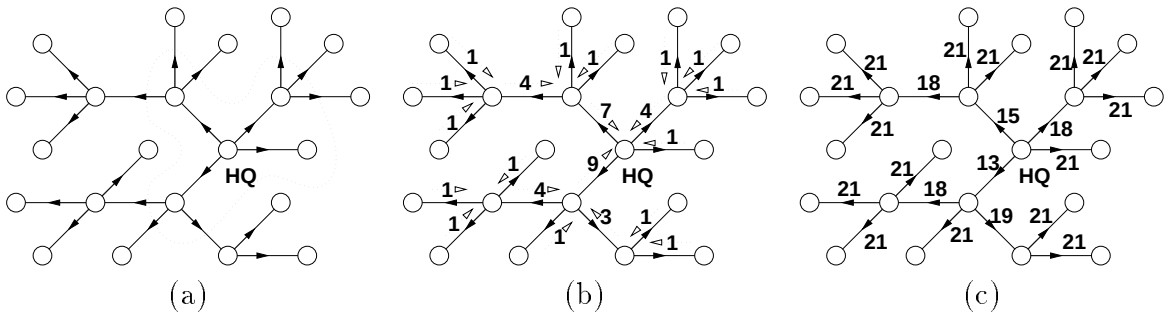


Figure 1: Message passing in a "spy network." (a) Arrows are directed from controllers to subordinates, showing the direction of propagation of "count" messages. (b) Head counts are combined at controller vertices and propagate to the "controller's controller" in the direction *opposite* to the arrows. (c) All members of the network are apprised of the total head count by further message propagation starting from HQ.

corresponding graph. In other words, the graph is a tree.

We generalize Pearl's distributed soldier counting example [3, p. 155] in the following way. Suppose that HQ wishes to obtain a head count for the entire network. HQ could accomplish this by "delegating" responsibility to the subordinates, with each subordinate responsible for providing an accurate head count for the subcell (if it exists) that they control. A subordinate who is not a controller would respond to such a request with a unit head count, whereas a subordinate who *is* a controller would delegate further. This process would continue, until count requests have propagated as indicated by the arrows of Fig. 1(a) and reached all leaf vertices.

Having propagated "count" instructions, each controller would wait for a head count to arrive from each of her subordinates. The controller would "fuse" (i.e., add up) the head counts received, including a count for herself, and then pass this head count to her own controller. Head counts would propagate in this manner (*opposite* the arrows in the original graph, as shown in Fig. 1(b)) until HQ receives counts from all subordinates, and obtains a head count for the entire network.

Indeed, every network vertex—not just HQ—could be apprised of the total head count by a further round of message propagation. For one thing, HQ could propagate the total itself as a message that propagates along the same paths as the "count" message of Fig. 1(a). Another approach would be to let each controller (starting with HQ) inform each subordinate of the total number of vertices in the network that lie in the complement of the network accounted for by that subordinate. In other words, each controller would "articulate" the state of the network for each subordinate, as shown in Fig. 1(c). This type of message propagation will be important for probability propagation and iterative decoding.

Notice that the graph of Fig. 1 is singly-connected, i.e., the graph has no *undirected cycles*

(i.e., cycles where the edge directions are ignored). In contrast, suppose some vertex v is a direct subordinate of its controller's controller. It is quite clear that the message passing algorithm, as described here, would not yield valid results in the presence of such undirected cycles. The controller of v's controller will double-count vertex v's head count because of the direct and indirect paths. This is the sort of problem that makes iterative decoding inexact.

As a final observation, it is important to note that this message passing process does not need to be initiated by HQ. For example, when a new cell is formed by any subordinate, a head count message could be generated that would propagate throughout the network. The main principle is one of "articulation:" as long as a vertex correctly articulates the state of the network in the complement of that to which a neighbour is connected, the neighbour will get a correct view of the network, and can update its state accordingly, and apprise its own neighbours of the new network state.

2.2 Data Fusion, Data Fission

The computational paradigm of message passing in graphs is not particular to probability propagation in Bayesian networks or Markov random fields, but rather is the principle underlying so-called "dataflow" machines [12, 13, 14] which have been proposed as post von Neumann computer architectures. Kung [19, p. 299] suggests that "wavefront arrays" are the union of systolic arrays with the dataflow computational paradigm. Perhaps one of the greatest strengths of the dataflow concept is the potential for efficient VLSI implementations. The dataflow concept is also inherently "object-oriented" so one should be able to achieve simple and elegant software implementations with an object-oriented programming language.

In the dataflow paradigm, computing elements correspond to graph vertices. A given computing element must essentially perform two tasks: data fusion (computing a function that depends on messages arriving at one or more "input lines", along with internal state) and data fission (distributing data via messages on one or more "output lines", along with updating the internal state). Fusion/fission is usually triggered in a computing element by the arrival of messages on one or more input lines, and in general a computing element may be triggered by more than one criterion.

In a singly-connected graph, the role of a vertex v is to articulate for a neighbour w the "state" of the sub-network $N_{v\mathbf{A}w}$ reached from w through v, as shown in Fig. 2. Reciprocally, w articulates for v the "state" of the sub-network $N_{w\mathbf{A}v}$. This paradigm will be computationally most effective when the size of the message passed in such articulation is relatively small. In other words, it would not be useful in most cases for v to tell w everything about $N_{v\mathbf{A}w}$, but rather only that part of $N_{v\mathbf{A}w}$ *relevant* for w's computational task. In statistical terms, v must articulate for w a sufficient statistic. In probability propagation, these messages turn out to be conditional probabilities. As we shall describe in Section 3, the structure of the graph will imply a conditional independence structure (or Markov property) that will make the conditional probabilities sufficient statistics in the special case when the graph is

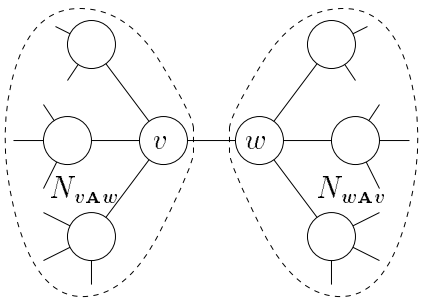


Figure 2: In a singly-connected network, vertex v articulates for vertex w the "state" of the sub-network $N_{v\mathbf{A}w}$ reached from w through v. Similarly, w articulates for v the state of $N_{w\mathbf{A}v}$.

singly connected (i.e., free of undirected cycles). The nature of the messages passed will be described in detail in Section 4.

3 Probability Graphs and Code Models

In this section we describe several graph-based models that can be used to model the conditional dependence structure in a collection of random variables. These graphical models will be useful for describing the structure of codes, and are the key to "probability propagation" and iterative decoding. We begin with a brief review of some graph-theoretic terminology.

3.1 Some Terminology from Graph Theory

A *graph* is the pair $G = (V, E)$ where V is a nonempty set of *vertices* and $E \subset \{\{a, b\} : a, b \in V\}$ is a collection of unordered pairs of vertices called *edges*. An edge $\{a, b\}$ is said to be incident on vertices a and b. A path of length n in G from vertex v_0 to vertex v_n is a sequence of edges of the form $\{\{v_0, v_1\}, \{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}\}$. A cycle is a path from a vertex to itself. A graph G is said to be *connected* if if there is at least one path in G from each vertex to every other vertex. A graph G is said to be *singly-connected* if there is exactly one path from each vertex to every other vertex. A singly-connected graph is a connected graph with no cycles.

The set $n(v)$ of *neighbours* of a vertex v in $G = (V, E)$ is the set of vertices that are reached from v by a path of unit length. Clearly if v is a neighbour of w then w is a neighbour of v, i.e., v and w are neighbours. A *clique* in a graph (V, E) is a collection $C \subset V$ of two or more vertices such that any pair of vertices drawn from C are neighbours. A *maximal clique* is a clique that is not properly contained in another clique.

A directed graph (*digraph*) is the pair $G = (V, E)$ where V is a nonempty set of vertices and $E \subset V \times V$ is a set of *directed edges* or *arcs*. An arc $e = (v_1, v_2) \in E$ has *initial vertex* $i(e) \stackrel{\text{def}}{=} v_1$ and *final vertex* $f(e) \stackrel{\text{def}}{=} v_2$. A *directed path* of length n in G from vertex v_0 to vertex v_n is a sequence of arcs of the form $\{(v_0, v_1), (v_0, v_2), \ldots, (v_{n-1}, v_n)\}$. A *cycle* is a directed path from a vertex to itself. A *DAG* (directed acyclic graph) is a digraph with no cycles.

A vertex v_1 is a *parent* of vertex v_2 (or v_2 is a *child* of v_1) in a digraph $G = (V, E)$ if $(v_1, v_2) \in E$. A vertex v_1 is an *ancestor* of a vertex v_2 (or v_2 is a *descendant* of v_1) if there is a directed path in G from v_1 to v_2. Given a vertex v, the set of v's parents is denoted $a(v)$, the set of v's ancestors is denoted $A(v)$, the set of v's children is denoted $d(v)$, and the set of v's descendants is denoted $D(v)$. In a DAG (V, E), for all $v \in V$, $A(v) \cap D(v) = \emptyset$, i.e., v's ancestors and descendants are disjoint.

Every digraph $D = (V, E)$ "induces" a graph $G = (V, E')$, where E' is the image of E under the function that "forgets" direction, i.e., the map that takes the arc (a, b) to the edge $\{a, b\}$. A digraph D is said to be singly-connected if the graph induced from D is singly-connected.

We define a "probability graph" as a graph (or digraph) in which the vertices are considered to be random variables. The interpretation of the edges (or arcs) in probability graphs will vary, depending on the type of graph, but in general, an edge joining two graph vertices will imply some type of probabilistic dependency between the corresponding random variables.

Let $V = \{V_1, V_2, \ldots, V_N\}$ denote a set of random variables where V_i takes on values from the set S_i. These random variables can be described by their joint probability mass (or density) function

$$P_{V_1,\ldots,V_N}(v_1, \ldots, v_N)$$

for $v_i \in S_i$. For simplicity of notation, we denote this function as $P(v_1, v_2, \ldots, v_N)$. Likewise, we will write $P(v_2|v_1)$ for the conditional probability mass (or density) function $P_{V_2|V_1}(v_2|v_1)$, etc. This simplified notation is commonly adopted in information theory (see, e.g., [20, 21]).

The "chain rule" of probability states that

$$P(v_1, \ldots, v_N) = P(v_1)P(v_2|v_1)P(v_3|v_1, v_2) \cdots P(v_N|v_1, \ldots, v_{N-1}). \tag{1}$$

This chain rule follows easily from the definition of conditional conditional probability, and will become important when various graphical models are introduced.

A probability mass function $P(v_1, \ldots, v_N)$ is said to be "strictly positive" if it is nonzero everywhere.

3.2 Markov Random Fields

Markov random fields are graphical models based on undirected graphs [22]. A probability graph $G = (V, E)$ is said to be a *Markov random field* (MRF) if,

$$(\forall v \in V) \;\; p(v|V\backslash\{v\}) = p(v|n(v)). \tag{2}$$

In other words, a probability graph is an MRF if every variable v is independent of non-neighbouring vertices in the graph, given the values of its immediate neighbours. Markov random fields are well developed in the mathematics literature, and have been used in a variety of applications (see, e.g., [22, 23, 24]). Here we will develop a Markov random field approach for codes, which later we will connect with the Tanner graph framework of [7, 8].

The joint probability mass (or density) function for the vertices of a Markov random field G is commonly expressed in terms of a Gibbs *potential function* defined on the maximal cliques of $G = (V, E)$. Let Q denote the collection of maximal cliques in G. Corresponding to each such clique q is a collection of vertices V_q that are contained in q. Given a nonnegative potential function for each clique $q \in Q$, i.e., a function

$$\psi_q : \bigotimes_{v \in V_q} S_v \to \mathbb{R},$$

the joint probability mass (or density) function over $V = \{v_1, \ldots, v_N\}$ is given by

$$p(v_1, \ldots, v_N) = Z^{-1} \prod_{q \in Q} \psi_q(\{v \in V_q\}), \tag{3}$$

where Z^{-1} is a normalizing constant, assuming that the product in (3) is not everywhere zero.

From the structure of the potential functions, it is a straightforward exercise (see, e.g., [22]) to show that the resulting probability distribution satisfies the definition (2) of a Markov random field. Indeed, every strictly positive MRF can be expressed in terms of a Gibbs potential, although the proof of this result (given, e.g., in [23, ch. 1]) is less straightforward.

To illustrate how MRFs can be used to describe codes, consider the MRF with seven vertices shown in Fig. 3. There are three maximal cliques: $q_1 = \{v_1, v_2, v_3, v_4\}$, $q_2 = \{v_2, v_4, v_5, v_7\}$, and $q_3 = \{v_3, v_4, v_6, v_7\}$. Hence from (3),

$$p(v_1, \ldots, v_7) = Z^{-1} \psi_{q_1}(v_1, v_2, v_3, v_4) \psi_{q_2}(v_2, v_4, v_5, v_7) \psi_{q_3}(v_3, v_4, v_6, v_7). \tag{4}$$

Assuming that $v_i \in \{0, 1\}$, this Markov random field can be used to describe a Hamming code by defining a real-valued function $\mathrm{ev}(x_1, x_2, x_3, x_4)$ that takes on the value 1 if $x_1 + x_2 + x_3 + x_4 \equiv 0$ modulo 2 and 0 otherwise. Then defining

$$\begin{aligned} \psi_{q_1}(v_1, v_2, v_3, v_4) &= \mathrm{ev}(v_1, v_2, v_3, v_4), \\ \psi_{q_2}(v_2, v_4, v_5, v_7) &= \mathrm{ev}(v_2, v_4, v_5, v_7), \\ \psi_{q_3}(v_3, v_4, v_6, v_7) &= \mathrm{ev}(v_3, v_4, v_6, v_7), \end{aligned}$$

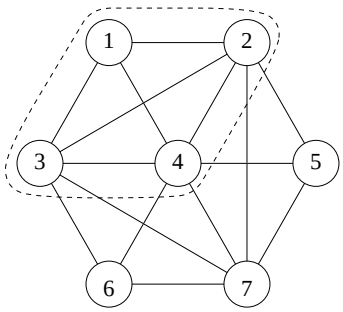


Figure 3: The graph for a Markov random field. The maximal clique $\{v_1, v_2, v_3, v_4\}$ is indicated.

from (4) it is easy to see that the MRF places equal probability on all codewords that satisfy even parity in each of the maximal cliques defined.

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 $V = \{v_1, \ldots, v_N\}$, let Q be a collection of subsets of V. 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 maximal clique, a Markov random field description for the given code is obtained.

As we shall see, a Tanner graph is another way to represent the same local check structure.

3.3 Tanner Graphs

Tanner graphs were introduced in [6]—though in a slightly different way—in the construction of good long error-correcting codes in terms of shorter codes. Our treatment follows that of Wiberg, et al. [7, 8].

A Tanner graph essentially is a graphical model that describes a check structure for a system. For a finite set N of sites and a collection $\{A_s\}_{s \in N}$ of alphabets, let $W = \prod_{s \in N} A_s$ denote a "configuration space." A system is the triple (N, W, B), where $B \subset W$ is a *behaviour*. A check structure for a system (N, W, B) is a collection Q of subsets of N, and corresponding collection of "local behaviours" (of B restricted to each $E \in Q$) such that any configuration $x \in W$ is a behaviour if and only if x_E, the restriction of x to E, is a valid local behaviour for every $E \in Q$.

A Tanner graph is a *bipartite* graph representation for such a check structure. In such a

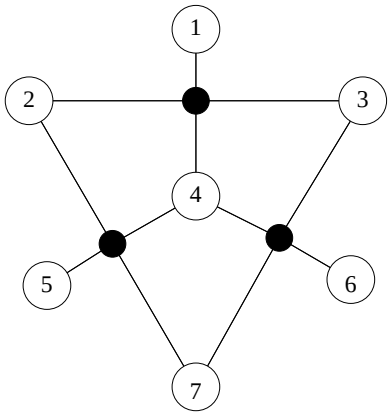


Figure 4: A Tanner graph for the (7,4) binary Hamming code.

graph, there are two types of vertices corresponding to the sites and the checks, respectively, with no edges connecting vertices of the same type. For example, the Tanner graph corresponding to the same Hamming code example as in the previous subsection is shown in Fig. 4. The check vertices are shown as filled circles, and each such vertex "checks" that the set of neighbours satisfies even parity in a valid configuration. We denote the site vertices as V_s and the check vertices as V_c.

We see that the check vertices have precisely the same role in a Tanner graph as the maximal cliques have in a Markov random field. In general, for each check vertex $v \in V_c$ with neighbours $n(v)$, we can associate a non-negative real-valued potential function $\psi_v(n(v))$. We then write a probability distribution over the elements of V_s as

$$p(\{v \in V_s\}) = Z^{-1} \prod_{v \in V_c} \psi_v(\{v \in n(v)\}), \tag{5}$$

where Z^{-1} is a normalizing constant, assuming that the product in (5) is not everywhere zero. Of course (5) is directly analogous to (3).

An MRF G can be converted to a Tanner graph simply by introducing a check vertex for each maximal clique in G, with edges connecting that check vertex to each clique element. Conversely, a Tanner graph can be converted to an MRF by eliminating the check vertices, and then forming cliques from all sites originally connected to the same check vertex. Generally, however, the correspondence between the graphs is not one-to-one, i.e., two different Tanner graphs may map to the same MRF graph, as shown in Fig. 5.

3.4 Bayesian Networks

We now introduce Bayesian networks that, unlike Markov random fields and Tanner graphs, are probability graphs that are DAGs. Let $G = (V, A)$ be a probability graph that is a DAG,

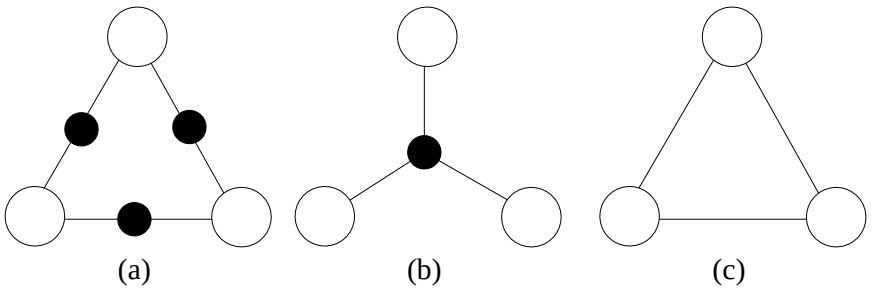


Figure 5: Two different Tanner graphs (a) and (b) that lead to the same MRF graph (c).

where $V = \{v_1, \ldots, v_N\}$. Then G is said to be a *Bayesian network* for the joint probability mass (or density) function $p(v_1, \ldots, v_N)$ if

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

where, in the notation of Section 3.1, $a(v_i)$ is the set of parents of v_i in G, and we take $p(v_i|\emptyset) = p(v_i)$ in the case where v_i has no parents in G.

It follows from the chain rule (1) that corresponding to every joint probability distribution there is a Bayesian network. For example, if $N = 4$, we obtain the trivial Bayesian network shown in Fig. 6.

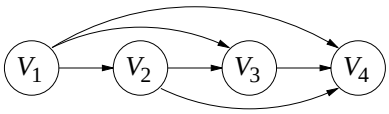


Figure 6: The trivial Bayes network with four vertices.

A Bayesian network G for p is said to be *minimal* if it fails to be a Bayesian network for p upon the deletion of any arc. A minimal Bayesian network can always be obtained from the trivial Bayesian network by deleting arcs. If $p(v_i|a(v_i)) = p(v_i|a(v_i)\backslash w$ for any vertex v_i and $w \in a(V_i)$, then the arc from w to v_i can be deleted. It is important to note, however, that the minimal Bayesian network obtained in this manner is quite sensitive to the ordering of the random variables from which the trivial Bayesian network is constructed. The problem of finding a Bayesian Network with the minimal number of edges is known to be NP hard.

Although different ordering of the random variables will lead to different Bayesian network representations, it is often useful to order the variables so that arrows point from "causes" to "effects." For example, if a particular variable v is a function of one or more other variables, it is often convenient to order the variables so that v appears later than the variables upon

which v depends. As described in the next subsection, for channel coding applications, it will usually be convenient to arrange the variables roughly in the order of input, state, codeword, channel output.

3.5 Bayesian Networks for Channel Coding

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. 7a. 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 on a single outcome). The *channel likelihood* $P(\mathbf{y}|\mathbf{x})$ expresses the noise and inter-symbol interference introduced by the channel.

Fig. 7b shows the Bayesian network for a systematic convolutional code with a memoryless channel. The systematic codeword symbols are denoted by a superscript "a" and are simply copies of the information symbols. The codeword symbols denoted by "b" are outputs of the encoder; x_k^b depends on u_k *and* state s_k. By inspecting the parents of the received signals, we find that

$$P(\mathbf{y}|\mathbf{x}) = \prod_k P(y_k^a|x_k^a)P(y_k^b|x_k^b),$$

which expresses the absence of memory in the channel. Fig. 7c shows a singly-connected network for the same code. By grouping information and state variables together, we eliminate all undirected cycles, at the expense of increasing the complexity of the state variables.

Further examples of Bayesian networks for codes will be discussed in Section 5. In the next section, we will describe the basic probability propagation algorithm that will form the basis for iterative decoding.

4 Probability Propagation

In this section, we describe how to propagate probabilistic information across a singly-connected Bayesian network in response to observing some of the variables. As an example of how probability propagation is useful, suppose we observe the channel output in the Bayesian network for a code; we would then like to compute the *a posteriori* information symbol probabilities in order to perform MAP decoding. Probability propagation is a

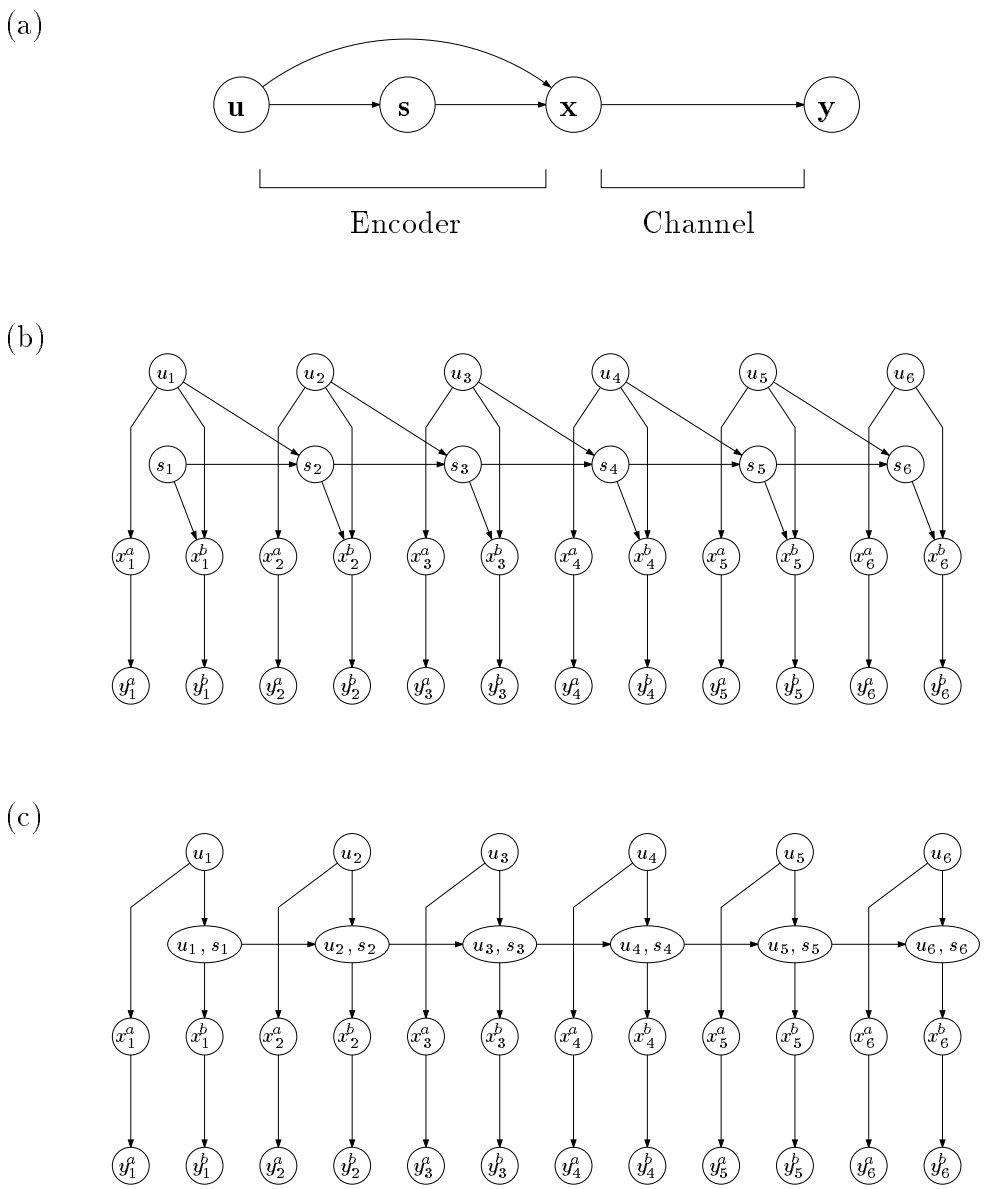


Figure 7: (a) The Bayesian network for channel coding. (b) The Bayesian network for a systematic convolutional code and a memoryless channel. (c) A singly-connected network for the convolutional code in (b).

method of propagating the effects of one or more observations across a Bayesian network so that each unobserved node can compute the probability for its variable *conditioned* on the observed variables.

Figure 8 shows a network fragment consisting of a variable r, its parents $\{q_i\}$, and its children $\{v_j\}$. (We define $\{v_j\}$ as the set of variables $\{v_1, v_2, \ldots\}$.) Some of the variables

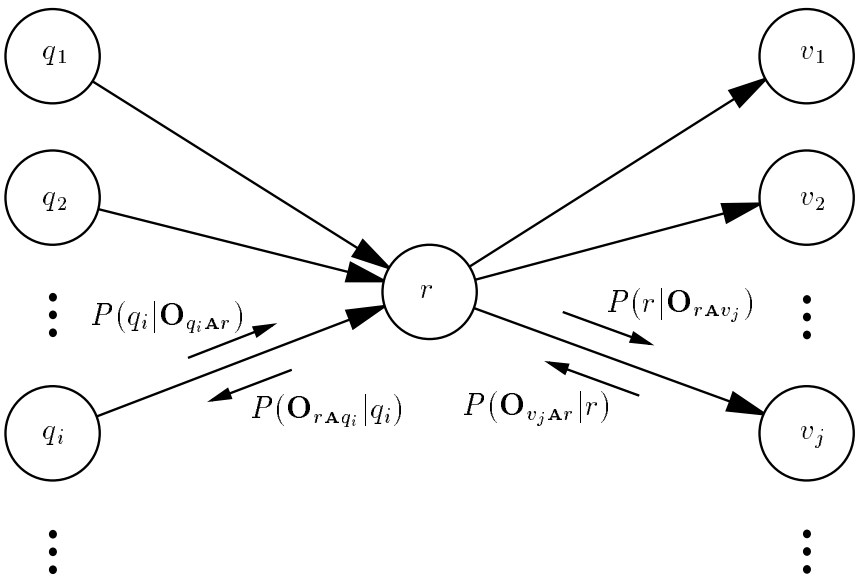


Figure 8: A network fragment shows the parents q_i and the children v_j of variable r in addition to some of the incoming and outgoing probability messages.

not shown have been observed and we wish to determine the effect of these observations on the probability of r; that is, we want to compute $P(r|\mathbf{O})$ where $\mathbf{O}$ is the set of observations. (We assume that r itself is not observed.) Since the network is singly-connected, each parent and child of r articulates the observations in the respective subnetwork. Let $\mathbf{O}_{v_j\mathbf{A}r}$ be the observations that v_j articulates for r. Similarly, let $\mathbf{O}_{q_i\mathbf{A}r}$ be the observations that q_i articulates for r. Then, the total set of observations $\mathbf{O}$ can be written as $\{\mathbf{O}_{q_i\mathbf{A}r}\}, \{\mathbf{O}_{v_j\mathbf{A}r}\}$.

To begin with, assume that the probability messages entering r, $P(q_i|\mathbf{O}_{q_i\mathbf{A}r})$ and $P(\mathbf{O}_{v_j\mathbf{A}r}|r)$, have been computed. Given these messages and the network specification $P(r|\{q_m\})$, we can compute $P(r|\mathbf{O})$ using the following formula:

$$P(r|\mathbf{O}) \overset{r}{\propto} \prod_j P(\mathbf{O}_{v_j\mathbf{A}r}|r) \sum_{q_i} P(r|\{q_m\}) \prod_k P(q_k|\mathbf{O}_{q_k\mathbf{A}r}), \tag{6}$$

where the symbol "$\overset{r}{\propto}$" indicates equality up to a constant that is independent of r. In order to get the actual value of $P(r|\mathbf{O})$, the quantity computed from the right hand side of (6) must be scaled so that $\sum_r P(r|\mathbf{O}) = 1$. The most significant computational burden in obtaining $P(r|\mathbf{O})$ from the incoming messages is usually the summation over all the possible configurations $\{q_m\}$ of the parents of r.

The outgoing parent-child message $P(r|\mathbf{O}_{r\mathbf{A}v_j})$ is computed as follows:

$$P(r|\mathbf{O}_{r\mathbf{A}v_j}) \overset{r}{\propto} P(r|\mathbf{O})/P(\mathbf{O}_{v_j\mathbf{A}r}|r). \tag{7}$$

Again, this message can be easily normalized. Notice that once $P(r|\mathbf{O})$ has been computed, the computation of this message is very simple.

The outgoing child-parent message $P(\mathbf{O}_{r\mathbf{A}q_i}|q_i)$ is computed, up to a scaling factor, as follows:

$$P(\mathbf{O}_{r\mathbf{A}q_i}|q_i) \overset{q_i}{\propto} \sum_r \prod_j P(\mathbf{O}_{v_j\mathbf{A}r}|r) \sum_{q_k:k\neq i} P(r|\{q_m\}) \prod_{n\neq i} P(q_n|\mathbf{O}_{q_n\mathbf{A}r}). \tag{8}$$

In this case, since q_i is the condition for $P(\mathbf{O}_{r\mathbf{A}q_i}|q_i)$, this probability can't be easily normalized. (To normalize it would require a summation over all the possible observation configurations $\mathbf{O}_{r\mathbf{A}q_i}$ that r articulates for q_i.) However, it turns out that these child-parent messages can be used in un-normalized form. They are used to compute (6) and (7), which can be normalized.

These update equations provide a consistent way to propagate conditional probability messages across the network. Notice that the above derivations rely on the fact that the network is singly-connected. When the network is *not* singly-connected, the propagation rules can still be applied, but there is no guarantee as to the quality of the results. Soon, we shall see that this is exactly what turbo-decoding in its simplest form does.

4.1 Messages in Waiting

The above update rules do not prescribe a *schedule* for passing the messages. When r receives a message from one of its children, must r immediately and in parallel propagate messages to its other children and its parents? It turns out that when the network is singly-connected, the propagation schedule does not matter, as long as all of the messages are eventually *absorbed*, as described below. Consequently, it is often useful to buffer messages. For example, if we know that r will soon receive messages from all the other children, we can pass fewer messages in total by waiting for those others to arrive before applying the update rules (6), (7) and (8). Pictorially, when a message arrives on an edge, but is not propagated on to the other neighbors, we draw a small dot on each of the other edges. These dots can be turned into arrows (indicating a message is being passed) at any time.

4.2 Message Creation and Absorption

Suppose that in Fig. 8, r is observed and has the value r'. This observation will cause new messages to be sent to the parents and the children of r. Obviously,

$$P(r|\mathbf{O}) = \begin{cases} 1, & \text{if } r = r' \\ 0, & \text{if } r \neq r' \end{cases} . \tag{9}$$

It follows from (7) that the only r for which the the parent-child message $P(r|\mathbf{O}_{r\mathbf{A}v_j})$ is nonzero is $r = r'$. So, we set the message to

$$P(r|\mathbf{O}_{r\mathbf{A}v_j}) = \begin{cases} 1, & \text{if } r = r' \\ 0, & \text{if } r \neq r' \end{cases} . \tag{10}$$

In order to determine the newly created child-parent message $P(\mathbf{O}_{r\mathbf{A}q_i}|q_i)$, we pretend that r is not observed and imagine a fictitious child node z that *is* observed and to which r is connected deterministically:

$$P(z|r) = \begin{cases} 1, & \text{if } z = r \\ 0, & \text{if } z \neq r \end{cases} . \tag{11}$$

The old observation is equivalent to the fictitious observation, $z = r'$. When determining the child-parent messages from (8), the term $\prod_j P(\mathbf{O}_{v_j\mathbf{A}r}|r)$ will now include the fictitious child z. Consequently, this term will be zero for all but the case $r = r'$. The child-parent message is thus

$$P(\mathbf{O}_{r\mathbf{A}q_i}|q_i) \overset{q_i}{\propto} \sum_{q_k : k \neq i} P(r = r'|\{q_m\}) \prod_{n \neq i} P(q_n|\mathbf{O}_{q_n\mathbf{A}r}). \tag{12}$$

Once created, these messages will continue to propagate across the network, creating new ones as they go. As mentioned above, it turns out that this process eventually terminates. When r receives a message from one of its neighbors v, new messages are sent to the other neighbors, but *not* back to v. So, if r is a vertex that is connected by a single edge, the received message will be absorbed.

4.3 Network Initialization

Before propagating the effects of observations across the network, we must produce initial messages that correspond to the *a priori* status of the network. We obtain the initial child-parent messages by noting that $\mathbf{O} = \varnothing$ and so

$$P(\mathbf{O}_{r\mathbf{A}q_i}|q_i) = P(\varnothing|q_i) = 1, \tag{13}$$

for all q_i. Also,

$$P(r|\mathbf{O}_{r\mathbf{A}v_j}) = P(r|\varnothing) = P(r) = \sum_{q_i} P(r|\{q_m\}) \prod_j P(q_j). \tag{14}$$

So, if r is a root node, it simply passes its marginal probability $P(r)$ to its children. If r is not a root node, it uses the marginal probabilities of its parents to compute $P(r)$. These messages propagate from parent to child until each variable has an initial probability.

4.4 The Forward-Backward Algorithm

As a familiar example of probability propagation, consider the Bayesian network for a convolutional code, as shown in Fig. 7c. The *forward-backward algorithm* for computing $P(u_k|\mathbf{y})$ for such a Markov-type model was developed by Baum and Petrie [25] as well as by Bahl *et. al.* [26] (who cite [25]). It turns out that this algorithm is identical to the algorithm that results when probability propagation is applied to the corresponding Bayesian network. A simpler, nonsystematic version of the network in Fig. 7c is shown by the first picture in Fig. 9. (We have included the codeword symbols with the state variables, for the sake of graphical simplicity.) When the channel output is observed (as shown by the filled discs),

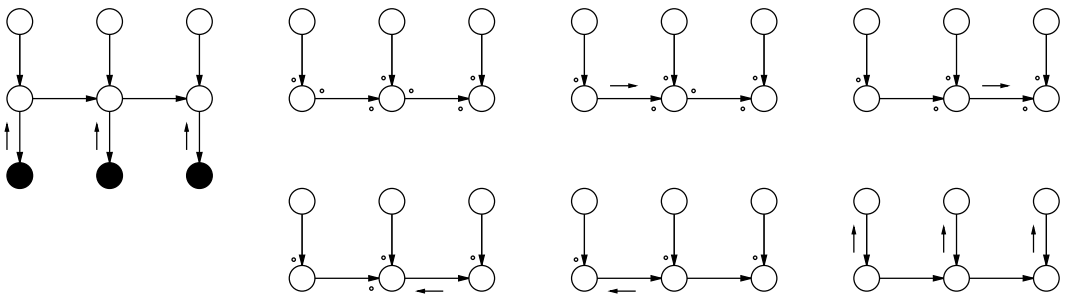


Figure 9: Probability propagation in the Bayesian network for a convolutional code.

messages propagate up to the state vertices. These messages are the codeword symbol likelihoods as determined by the channel model $P(y_k|x_k)$ and the channel output $\mathbf{y}$. In order to minimize the total number of messages passed, the messages are buffered as shown. A series of parent-child messages are then passed *forward* along the chain. Then, a series of child-parent messages are passed *backward* along the chain. These two types of message are exactly the same as the "α"s and "β"s in the standard forward-backward algorithm. Finally, the information symbols receive probability messages that allow them to compute $P(u_k|\mathbf{y})$.

Although this example does not really improve our understanding of decoding a single convolutional code, it does illustrate how algorithms for decoding on Bayesian networks can be easily derived using the probability propagation idea. In the following section, we will

present the Bayesian networks for several recent codes, including turbo-codes, and illustrate that the new iterative decoding algorithms (*e.g.*, turbo-decoding) are instances of probability propagation (albeit, very clever ones) applied to networks that are *not* singly-connected.

5 Compound Codes

We define a *compound code* to be one which can be described by a set of constituent codes, each of which is tractably decodable on its own. Graphically, each constituent code is represented by a singly-connected constituent Bayesian network. These constituent networks share some variables, so that taken as a whole, the total Bayesian network is not singly-connected. In general, there does not exist a unique decomposition of a compound code into its constituent codes. However, usually the compound code is designed using well-known constituent codes.

5.1 Bayesian Networks for Some Known Codes

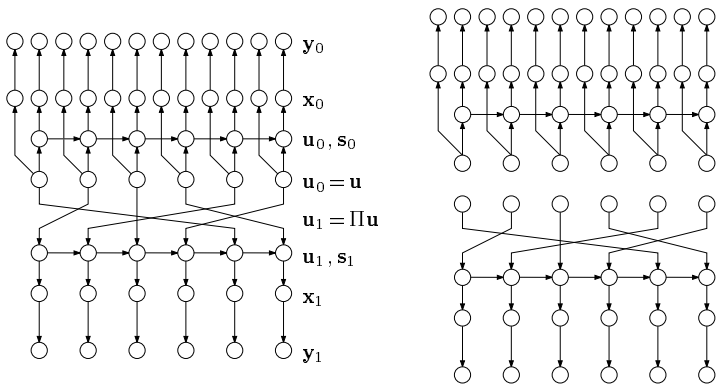
The Bayesian network for a systematic rate 1/3 compound turbo-code is shown in Fig. 10a, along with its singly-connected constituent networks. This compound code consists of two chain-type networks that are connected using a different ordering of the information symbol vertices. Whereas the upper chain directly uses the information sequence $\mathbf{u}$, the lower chain uses a permuted sequence, obtained by applying an interleaver matrix Π: $\mathbf{u}_1 = \Pi\mathbf{u}$. The systematic codeword component has been included with the upper constituent code.

A "serially concatenated" convolutional compound code was recently proposed by Benedetto and Montorsi [15]. Their system is essentially the same as Forney's concatenated codes [27], except that they use convolutional codes for both the inner and outer codes, instead of the more popular Reed-Solomon outer code. The Bayesian network for a non-systematic rate 1/2 compound code of this sort is shown in Fig. 10b, along with its two singly-connected constituent networks. Notice that the output of the first convolutional code (including the systematic part) is used as the input to the second convolutional code, via an interleaver. Only the output of the second convolutional code is transmitted over the channel.

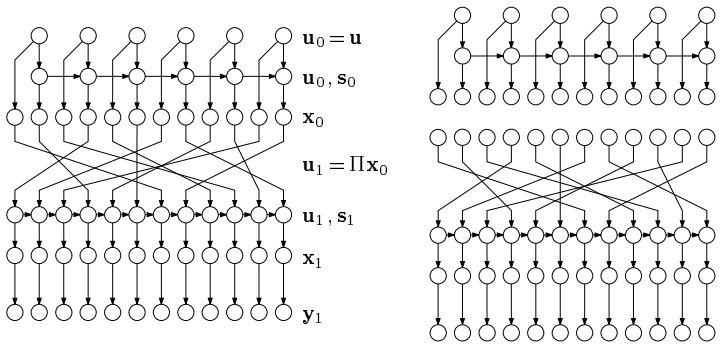
Fig. 10c shows an example of Gallager's low-density parity check codes [5]. This code consists of parity check restrictions on subsets of the codeword symbols. (The mapping from information blocks to codewords is not directly specified.) Each q-variable is a dummy variable whose conditional probability is nonzero only for configurations of its parents that have even parity. MacKay and Neal [10] were the first to describe Gallager's codes using Bayesian networks.

Finally, Fig. 11 shows an example of a systematic product code.

(a)



(b)



(c)

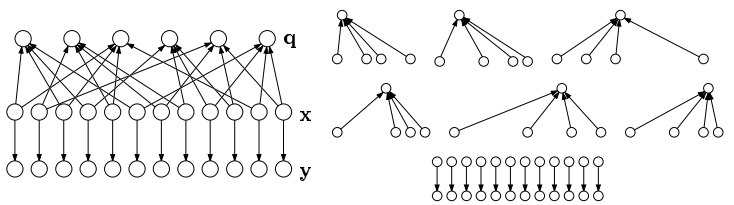


Figure 10: The Bayesian networks for three compound codes and their constituent codes. (a) A systematic rate 1/3 turbo-code. (b) A rate 1/2 "serially concatenated" convolutional code. (c) A low-density parity check code.

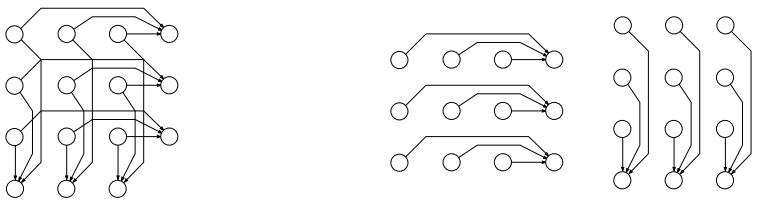


Figure 11: The Bayesian networks for a (15,9) product code and its constituent codes.

5.2 Soft Iterative Decoding: Probability Propagation for Compound Codes

Since each constituent Bayesian network in a compound code is singly-connected, probabilities for each random variable given the observed random variables can be efficiently computed *exactly* within each constituent network. However, because the compound network is multiply-connected, 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 using, *e.g.*, annealing [1, 28]. This graphical framework unifies several iterative decoding algorithms. The turbo-decoding algorithm, the "separable MAP filter" algorithm [17], the new iterative decoding algorithm [16] used for decoding "serially concatenated" convolutional codes, and 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. 10 and Fig. 11.

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 singly-connected, stopping when each vertex has been processed *once*. In fact, because the compound code network is multiply-connected, the propagation procedure actually never self-terminates. Usually the cyclic procedure is allowed to iterate until some termination criterion is satisfied (*e.g.*, a specific number of iterations have occurred). Then, the information symbols are detected, usually in the fashion of the maximum *a posteriori* (MAP) symbol probability decoding rule.

5.3 Turbo-Decoding: Probability Propagation for Turbo-Codes

We have implemented a probability propagation algorithm for decoding on the Bayesian network for turbo codes (Fig. 10a). This algorithm produces the same results as the standard turbo-decoding algorithm (without "annealing"), indicating empirically that indeed turbo-decoding is probability propagation. In fact, it is straightforward to prove that turbo-decoding is probability propagation for this compound network. For the sake of simplicity, we will avoid the mathematical details. The turbo-decoding algorithm uses the forward-backward algorithm (or an approximation to it) to process each constituent trellis. The algorithm uses "extrinsic information" [1] 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 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. 12 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. 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. A series of parent-child messages are passed *forward* along one constituent network. Then, a series of child-parent messages are passed *backward* along the same constituent network. Next, the information symbols receive probability messages from the constituent network just processed. In the case of the single code (Fig. 9), 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".) The overall procedure is cyclical and repeats over and over as shown by the pictures outlined by a dashed polygon.

6 Conclusions

In this tutorial paper we have attempted to unify various recently developed themes in iterative decoding. We have reviewed various graphical models such as Markov random fields, Tanner graphs and Bayesian networks, all of which encode the "local" probabilistic structure of codes and channels. We have emphasized Bayesian networks here, because the basic input/state/output/channel description gives rise both to useful encoder structures and useful decoding algorithms.

We have described the by now "well-known" probability propagation algorithm that can be used to compute the *exact* posterior conditional probability distribution for any variable in a singly-connected Bayesian network, given a set of observations. Unfortunately, Bayesian net-

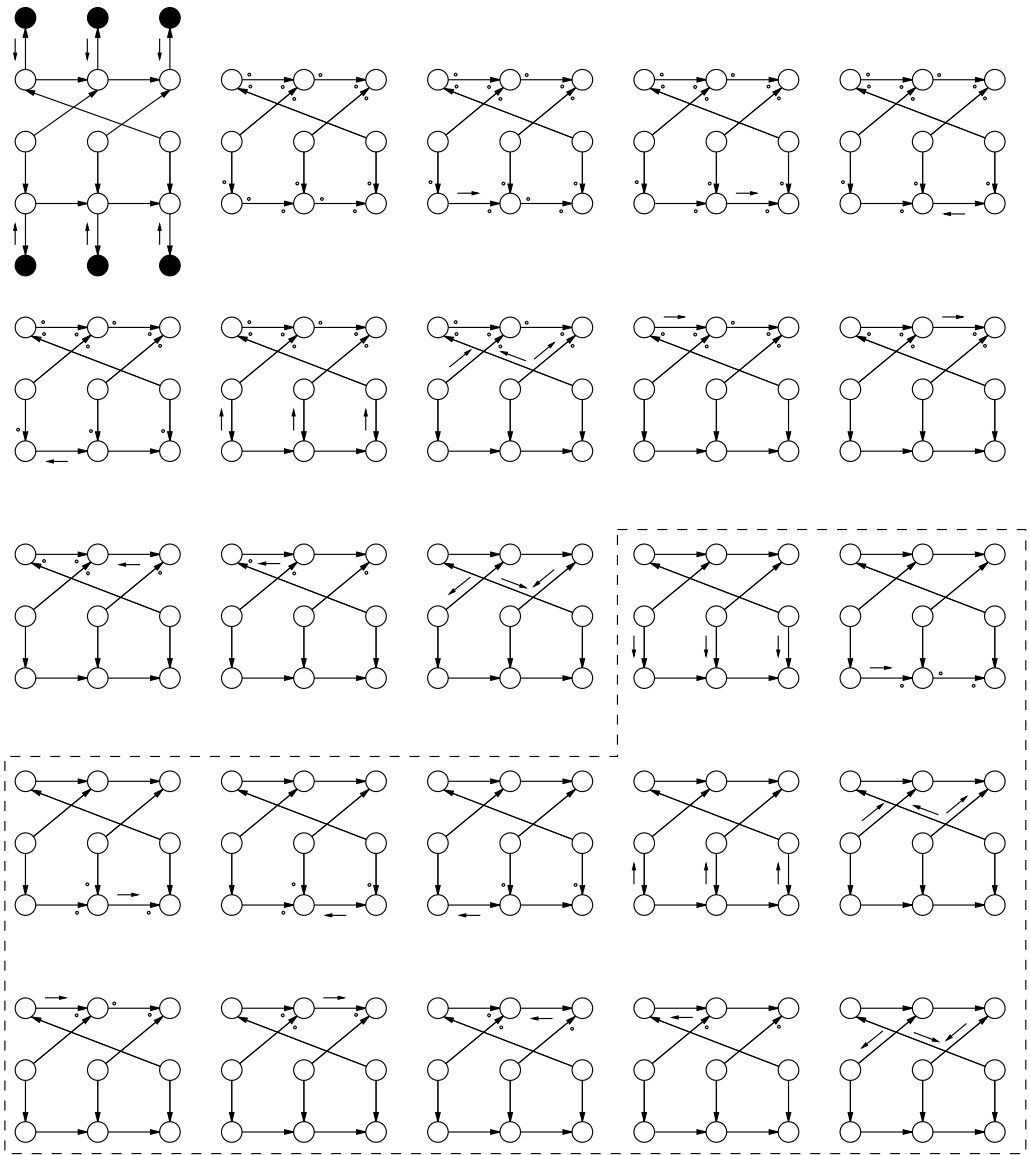


Figure 12: Probability propagation in the Bayesian network for a turbo-code.

works for many practical codes are not singly-connected. For "compound codes," however, the Bayesian networks can be 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 algorithms devised for iterative decoding of turbo-codes, low-density parity-check codes, product-codes, serially-concatenated codes, etc., all can be interpreted essentially as probability propagation algorithms.

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.

Acknowledgment

We are grateful to P. G. Gulak for useful discussions.

A Derivations for Probability Propagation

Probability propagation is in general only exact for singly-connected Bayesian networks. In such a network, articulated observations can be considered to be local to the articulating vertex. For example, when considering the effect that child vertex v_j will have on r, the observations in $\mathbf{O}_{v_j \mathbf{A} r}$ can be considered as part of v_j. This follows from the fact that since the network is singly-connected, these observations can only influence r through v_j. We use this fact and several others to derive the probability messages. (Also, see [3] for the derivations.)

A.1 Derivation of $P(r|\mathbf{O})$

Noting that $\mathbf{O} = \{\mathbf{O}_{q_i \mathbf{A} r}\}, \{\mathbf{O}_{v_j \mathbf{A} r}\}$, we break $P(r|\mathbf{O})$ into a product of two pieces, using Bayes rule:

$$\begin{aligned} P(r|\mathbf{O}) = P(r|\{\mathbf{O}_{q_i \mathbf{A} r}\}, \{\mathbf{O}_{v_j \mathbf{A} r}\}) &= \frac{P(r, \{\mathbf{O}_{v_j \mathbf{A} r}\}|\{\mathbf{O}_{q_i \mathbf{A} r}\})}{P(\{\mathbf{O}_{v_j \mathbf{A} r}\}|\{\mathbf{O}_{q_i \mathbf{A} r}\})} \\ &\overset{r}{\propto} P(r, \{\mathbf{O}_{v_j \mathbf{A} r}\}|\{\mathbf{O}_{q_i \mathbf{A} r}\}) = P(r|\{\mathbf{O}_{q_i \mathbf{A} r}\}) P(\{\mathbf{O}_{v_j \mathbf{A} r}\}|r, \{\mathbf{O}_{q_i \mathbf{A} r}\}). \end{aligned} \tag{15}$$

Now, as indicated by the conditional dependencies depicted in the Bayesian network, given r, the observations $\{\mathbf{O}_{v_j \mathbf{A} r}\}$ are independent of the observations $\{\mathbf{O}_{q_i \mathbf{A} r}\}$. (Note that this

would not be so if there were a path not including r from an ancestor of r to a descendent of r.) So, $P(\{\mathbf{O}_{v_j \mathbf{A} r}\}|r, \{\mathbf{O}_{q_i \mathbf{A} r}\}) = P(\{\mathbf{O}_{v_j \mathbf{A} r}\}|r)$, and

$$P(r|\mathbf{O}) = P(r|\{\mathbf{O}_{q_i \mathbf{A} r}\}) P(\{\mathbf{O}_{v_j \mathbf{A} r}\}|r). \tag{16}$$

Since the graph is singly-connected, the sets of observations $\mathbf{O}_{v_j \mathbf{A} r}$ do not overlap, so the second factor can be broken into a product as follows:

$$P(\{\mathbf{O}_{v_j \mathbf{A} r}\}|r) = \prod_j P(\mathbf{O}_{v_j \mathbf{A} r}|r). \tag{17}$$

Finally, the first factor is expressed in terms of messages and the network specification as follows:

$$\begin{aligned} P(r|\{\mathbf{O}_{q_i \mathbf{A} r}\}) &= \sum_{q_i} P(r|\{q_i\}) P(\{q_i\}|\{\mathbf{O}_{q_i \mathbf{A} r}\}) \\ &= \sum_{q_i} P(r|\{q_i\}) \prod_k P(q_k|\mathbf{O}_{q_k \mathbf{A} r}). \end{aligned} \tag{18}$$

where the first step is simply an integration over the parent variables, and the second step follows from the fact that the parents of r cannot be connected in any way but through r, so $P(\{q_i\}|\{\mathbf{O}_{q_i \mathbf{A} r}\}) = \prod_k P(q_k|\mathbf{O}_{q_k \mathbf{A} r})$. Combining (16), (17) and (18) gives the formula for computing $P(r|\mathbf{O})$ in (6).

A.2 Derivation of $P(\mathbf{O}_{r \mathbf{A} q_i}|q_i)$

First, note that $\mathbf{O}_{r \mathbf{A} q_i} = \{\mathbf{O}_{v_j \mathbf{A} r}\}, \{\mathbf{O}_{q_k \mathbf{A} r}\}_{k \neq i}$, so

$$\begin{aligned} P(\mathbf{O}_{r \mathbf{A} q_i}|q_i) &= P(\{\mathbf{O}_{v_j \mathbf{A} r}\}, \{\mathbf{O}_{q_k \mathbf{A} r}\}_{k \neq i}|q_i) = \sum_r P(r, \{\mathbf{O}_{v_j \mathbf{A} r}\}, \{\mathbf{O}_{q_k \mathbf{A} r}\}_{k \neq i}|q_i) \\ &= \sum_r P(\{\mathbf{O}_{v_j \mathbf{A} r}\}|r) P(r, \{\mathbf{O}_{q_k \mathbf{A} r}\}_{k \neq i}|q_i) \\ &= \sum_r \prod_j P(\mathbf{O}_{v_j \mathbf{A} r}|r) \sum_{q_k : k \neq i} P(r, \{q_m\}, \{\mathbf{O}_{q_k \mathbf{A} r}\}_{k \neq i}|q_i) \\ &= \sum_r \prod_j P(\mathbf{O}_{v_j \mathbf{A} r}|r) \sum_{q_k : k \neq i} P(r|\{q_m\}) P(\{q_m\}_{m \neq i}|\{\mathbf{O}_{q_k \mathbf{A} r}\}_{k \neq i}) P(\{\mathbf{O}_{q_k \mathbf{A} r}\}_{k \neq i}) \\ &\overset{q_i}{\propto} P(\mathbf{O}_{r \mathbf{A} q_i}|q_i) \overset{q_i}{\propto} \sum_r \prod_j P(\mathbf{O}_{v_j \mathbf{A} r}|r) \sum_{q_k : k \neq i} P(r|\{q_m\}) \prod_{n \neq i} P(q_n|\mathbf{O}_{q_n \mathbf{A} r}). \end{aligned} \tag{19}$$

In the above steps, to get the child-parent message sent to q_i, we marginalized over r and the other parents $\{q_k\}_{k \neq i}$, while using the probabilistic structure implied by the Bayesian network.

A.3 Derivation of $P(r|\mathbf{O}_{r\mathbf{A}v_j})$

First, note that $\mathbf{O}_{r\mathbf{A}v_j} = \{\mathbf{O}_{q_i\mathbf{A}r}\}, \{\mathbf{O}_{v_k\mathbf{A}r}\}_{k\neq j}$, so

$$\begin{aligned} P(r|\mathbf{O}_{r\mathbf{A}v_j}) &= P(r|\{\mathbf{O}_{q_i\mathbf{A}r}\}, \{\mathbf{O}_{v_k\mathbf{A}r}\}_{k\neq j}) \overset{r}{\propto} P(r, \{\mathbf{O}_{v_k\mathbf{A}r}\}_{k\neq j}|\{\mathbf{O}_{q_i\mathbf{A}r}) \\ &= P(\{\mathbf{O}_{v_k\mathbf{A}r}\}_{k\neq j}|r)P(r|\{\mathbf{O}_{q_i\mathbf{A}r}) = \frac{P(\{\mathbf{O}_{v_k\mathbf{A}r}\}|r)P(r|\{\mathbf{O}_{q_i\mathbf{A}r})}{P(\{\mathbf{O}_{v_j\mathbf{A}r}\}|r)} \\ &= P(r|\mathbf{O})/P(\{\mathbf{O}_{v_j\mathbf{A}r}\}|r). \end{aligned} \tag{20}$$

Notice that the above derivations rely on the fact that the network is singly-connected. When the network is *not* singly- connected, the propagation rules can still be applied, but there is no guarantee as to the quality of the results. Soon, we shall see that this is exactly what turbo-decoding in its simplest form does.

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

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

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

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

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

[7] 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.

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

[9] 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.

[10] 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.

[11] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices." Submitted to *IEEE Transactions on Information Theory*, 1996.

[12] Arvind and K. P. Gostelow, "A computer capable of exchanging processing elements for time," in *Proc. IFIP Congr. 77*, pp. 849–854, 1977.

[13] J. B. Dennis, "The varieties of data flow computers," in *First IEEE Int. Conf. Data Flow Comput.*, pp. 430–431, 1979.

[14] P. C. Treleaven, "Exploiting program concurrency in computing systems," *Computer*, vol. 12, pp. 42–49, Jan. 1979.

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

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

[17] 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.

[18] 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.

[19] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[20] R. G. Gallager, *Information Theory and Reliable Communication*. Wiley, 1968.

[21] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley and Sons, 1991.

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

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

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

[25] 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.

[26] 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.

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

[28] D. Divsalar and F. Pollara, "Turbo-codes for PCS applications," in *Proceedings of the International Conference on Communications*, pp. 54–59, 1995.