In *Proceedings of the* 34$^{\text{th}}$ *Allerton Conference*, Champaign-Urbana, Illinois, 1996.

Probability Propagation and Iterative Decoding

Brendan J. Frey and **Frank R. Kschischang**
Department of Electrical and Computer Engineering
University of Toronto
brendan@comm.utoronto.ca and frank@comm.utoronto.ca

Abstract

In this paper, we present a unified *graphical model* framework for describing codes and deriving iterative decoding algorithms. We illustrate how the following systems can be described using graphical models: turbo-codes, serially-concatenated convolutional codes, frame-oriented turbo-codes, low-density parity-check codes, product codes, and convolutional codes on channels with memory. Recently proposed iterative decoding algorithms (*e.g.*, turbo-decoding) can be viewed as a simple message passing procedure on the graphical model. This framework provides the means to derive new iterative decoders for new codes quite easily.

1 Introduction

In this paper, we seek to unify various themes that underlie much of the recent work on iterative decoding and turbo-codes. 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 and his co-workers (see *e.g.*, [1]). Interestingly, Gallager's algorithm [2] 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 mostly forgotten in the interim. Recently, Wiberg, Loeliger and Kötter [3] 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 decoding techniques, much as the code trellis is an appropriate model in which to describe and study "conventional" maximum-likelihood soft-decision decoding using the Viterbi algorithm. MacKay and Neal in [4] 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, we first present a simple example of message passing in a "spy network." Although simplistic, this example illustrates many of the important properties of probability propagation algorithms. Next, we review graphical code models, including 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.

We then describe the probability propagation algorithm. In the case of singly-connected graphs, probability propagation is an exact algorithm that allows one to compute the *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. However, in some cases (such as turbo-decoding) empirical results have shown great promise for the algorithm.

Finally, we discuss various applications of these results for the decoding of "compound codes," which we think of as codes 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. Examples of compound codes include turbo-codes [5], serially concatenated convolutional codes [6], frame-oriented turbo-codes [7], low-density parity-check codes [2], and iteratively decoded product codes [8, 9].

2 A Simple "Spy Network"

We now give a simple example of a distributed algorithm that computes a result via local message passing. This example, though simplistic, serves to illustrate many of the important properties of probability propagation algorithms.

In the tradition of spy novels, imagine a "spy network" (or corporate hierarchy) arranged in a tree, like the one shown in Fig. 1(a). Label the root vertex with 'HQ', for "headquarters." The subordinates of a given vertex v (or "controller") is the set of vertices reached by a directed edge from v. In a spy network, a vertex and its subordinates form a "cell" and only the controller has knowledge of the number vertices ("spies") in a given cell.

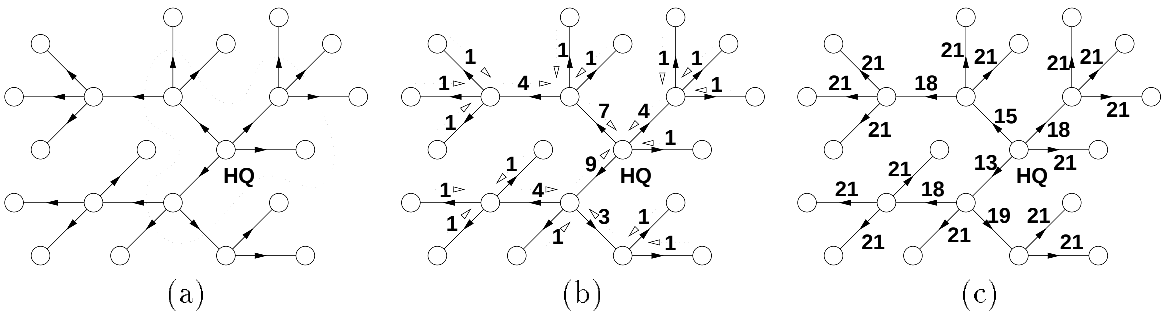

Fig. 1: Message passing in a "spy network." (a) Arrows are directed from controllers to subordinates. (b) Head counts are combined at controller vertices and propagate to the "controller's controller." (c) All members of the network receive the total head count by further message propagation.

We generalize Pearl's distributed soldier counting example [1, p. 155] in the following way. Suppose that HQ wishes to obtain a head count for the entire network. An obvious way that HQ could accomplish this, even without knowledge of the network topology, would be by "delegating" responsibility to the subordinates, each of whom could delegate responsibility to their subordinates, and so on, until "count requests" have propagated as indicated by the arrows of Fig. 1(a) to the leaf vertices.

After receiving head counts from her subordinates, each controller would then sum them, including a count for herself. This head count would then be passed "up" to the controller's controller, and so on, until the head count messages arrive at HQ, who would then be able to ascertain the total network head count (see Fig. 1(b)).

Indeed, every network vertex — not just HQ — could be apprised of the total head count by a further round of message propagation. One 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).

In general, 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 $N_{w\mathbf{A}v}$. (The notation $N_{v\mathbf{A}w}$ is to be read "the network that v articulates for w.") In statistical terms, v must articulate for w a sufficient statistic, which essentially turns out to be a conditional probability in the case of probability propagation in singly-connected Bayesian networks.

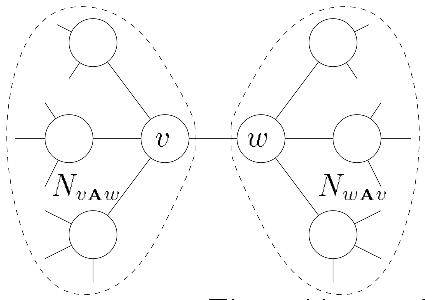

Figure 2. The idea of articulation.

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.

3 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. A graphical model consists of a set of random variables $\{v_1, \ldots, v_N\}$, a graph where each vertex corresponds to a variable, and a probability distribution $P(v_1, \ldots, v_N)$ whose conditional dependency "structure" can be derived from the graph. These graphical models will be useful for describing the structure of codes, and are the key to "probability propagation" and iterative decoding.

3.1 Markov Random Fields

Markov random fields (MRFs) are graphical models based on undirected graphs (see textbook reference [10]). If $n(v_i)$ are the neighbors of vertex v_i, then the the distribution $P(\cdot)$ must satisfy the local Markov property:

$$p(v_i|\{v_j\}_{\forall j \neq i}) = p(v_i|n(v_i)). \tag{1}$$

In a MRF, each variable v_i is independent of non-neighbouring vertices in the graph, given the values of its immediate neighbours. The joint distribution is commonly expressed in terms of a Gibbs *potential function* defined on the maximal cliques (*i.e.*, maximal complete subgraphs). Corresponding to each clique q in the set of maximal cliques Q is a collection of vertices V_q. Given a nonnegative potential function $\psi_q(\{v_i \in V_q\})$ for each clique $q \in Q$, the joint distribution over $\{v_1, \ldots, v_N\}$ is given by

$$P(v_1, \ldots, v_N) = Z^{-1} \prod_{q \in Q} \psi_q(\{v_i \in V_q\}), \tag{2}$$

where Z^{-1} is a normalizing constant.

To illustrate how MRFs can be used to describe codes, consider the MRF with seven binary variables shown in Fig. 3a. There are four maximal cliques: $q_1 = \{v_1, v_2, v_5, v_6\}$ (dashed loop), $q_2 = \{v_1, v_4, v_6, v_7\}$, $q_3 = \{v_1, v_3, v_5, v_7\}$, and $q_4 = \{v_1, v_5, v_6, v_7\}$. From (2), $P(v_1, \dots, v_7) = Z^{-1} \psi_{q_1}(v_1, v_2, v_5, v_6)\ \psi_{q_2}(v_1, v_4, v_6, v_7)\ \psi_{q_3}(v_1, v_3, v_5, v_7)\ \psi_{q_4}(v_1, v_5, v_6, v_7)$. This MRF can be used to describe a Hamming code by setting $\psi_{q_4} = 1$ and by letting the first three potentials be even parity indicator functions. That is, $\psi_q(x_1, x_2, x_3, x_4) = 1$ if its variables have even parity and 0 otherwise. The MRF places equal probability on all codewords that satisfy even parity in cliques q_1, q_2 and q_3.

3.2 Tanner Graphs

Tanner graphs were introduced in [11] 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.* [3]. 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. 3b. Each check vertex q in the set of check vertices Q is shown as a filled circle. A check vertex ensures that its set of neighbours satisfies even parity in a valid configuration.

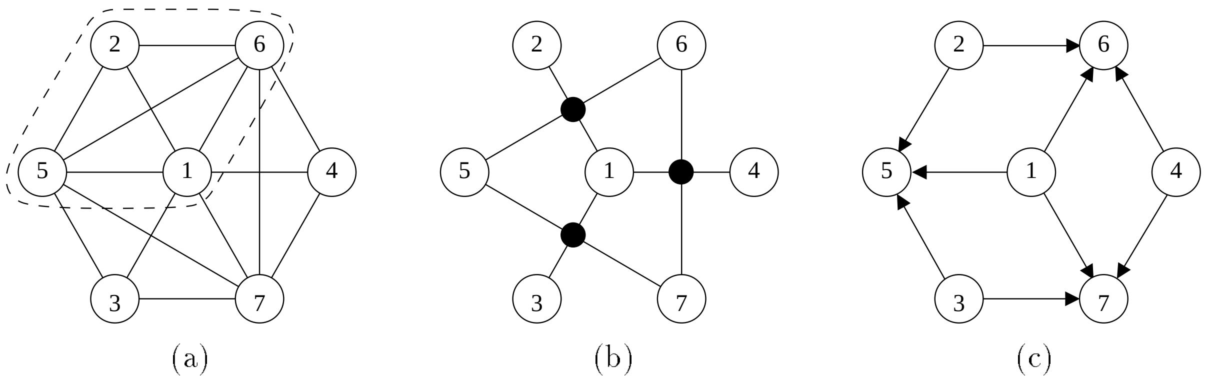

Fig. 3: Graphical models for the (7,4) Hamming code: (a) a Markov random field, with a maximal clique indicated; (b) a Tanner graph; and (c) a Bayesian network.

We see that the check vertices have a similar role in a Tanner graph as the maximal cliques have in a Markov random field. In general, for each check vertex q with neighbours $n(q)$, we can associate a non-negative real-valued potential function $\psi_q(\{v_i \in n(q)\})$. We then write a probability distribution over the variables as

$$P(v_1, \dots, v_N) = Z^{-1} \prod_{q \in Q} \psi_q(\{v_i \in n(q)\}), \tag{3}$$

where Z^{-1} is a normalizing constant. Of course, (3) is analogous to (2).

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. 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. However, Tanner graphs may be more specific about dependencies than MRFs. Different Tanner graphs may map to the same MRF; *e.g.*, the graph in Fig. 3b with an additional check vertex connected to v_1, v_5, v_6 and v_7 will also map to the MRF in Fig. 3a.

3.3 Bayesian Networks

We now introduce Bayesian networks that, unlike Markov random fields and Tanner graphs, have *directed acyclic* graphical representations [1]. A directed acyclic graph is one where there are no graph cycles when the edge directions are followed. (There may be cycles when the edge directions are ignored.) Let the *parents* $a(v_i)$ of vertex v_i be the set of vertices that have directed edges connecting *to* v_i. For a Bayesian network, the joint distribution can be written

$$P(v_1, \ldots, v_N) = \prod_{i=1}^{N} P(v_i | a(v_i)), \tag{4}$$

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, $P(v_1, \ldots, v_N) = P(v_1)P(v_2|v_1)P(v_3|v_1, v_2)\ldots P(v_N|v_1, v_2, \ldots, v_{N-1})$. So, we can pick any ordering of the variables and then condition each variable on all previous variables. However, this trivial network does not capture any useful probabilistic structure. The last factor, $P(v_N|v_1, v_2, \ldots, v_{N-1})$, contains all N variables and so is really just as complicated as the full joint distribution.

A Bayesian network for the Hamming code described above is shown in Fig. 3c. The joint distribution is obtained from (4), using parent-child relationships: $P(v_1, \ldots, v_7) = P(v_1)P(v_2)P(v_3)P(v_4)P(v_5|v_1, v_2, v_3)P(v_6|v_1, v_2, v_4)P(v_7|v_1, v_3, v_4)$. The first four factors express the prior probabilities of $v_1, \ldots, v_4$. 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.

3.4 Advantages of Bayesian Networks over MRFs and Tanner Graphs

The most significant advantage of the Bayesian network is that it explicitly represents causality. By inspecting the directed edges, it is easy to see which variables directly influence others. Consequently, compared to the MRF and Tanner graph representations, for a given code, it is easier to derive an encoding algorithm from the Bayesian network. For example, in Fig. 3c, we simply pick values for the systematic root 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. The MRF and Tanner graphs do not specify such a temporal construction for the code. The explicit representation of causality is also useful for modelling physical effects, such as channel noise. For example, in a channel with memory, the current received signal depends on the corresponding transmitted codeword symbol *and earlier ones*. Whereas the MRF and Tanner graph cannot explicitly represent this temporal causality, the Bayesian network can (as we shall see later). As described in the next section, 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. The Bayesian network for channel coding in general is shown in Fig. 4a. 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.

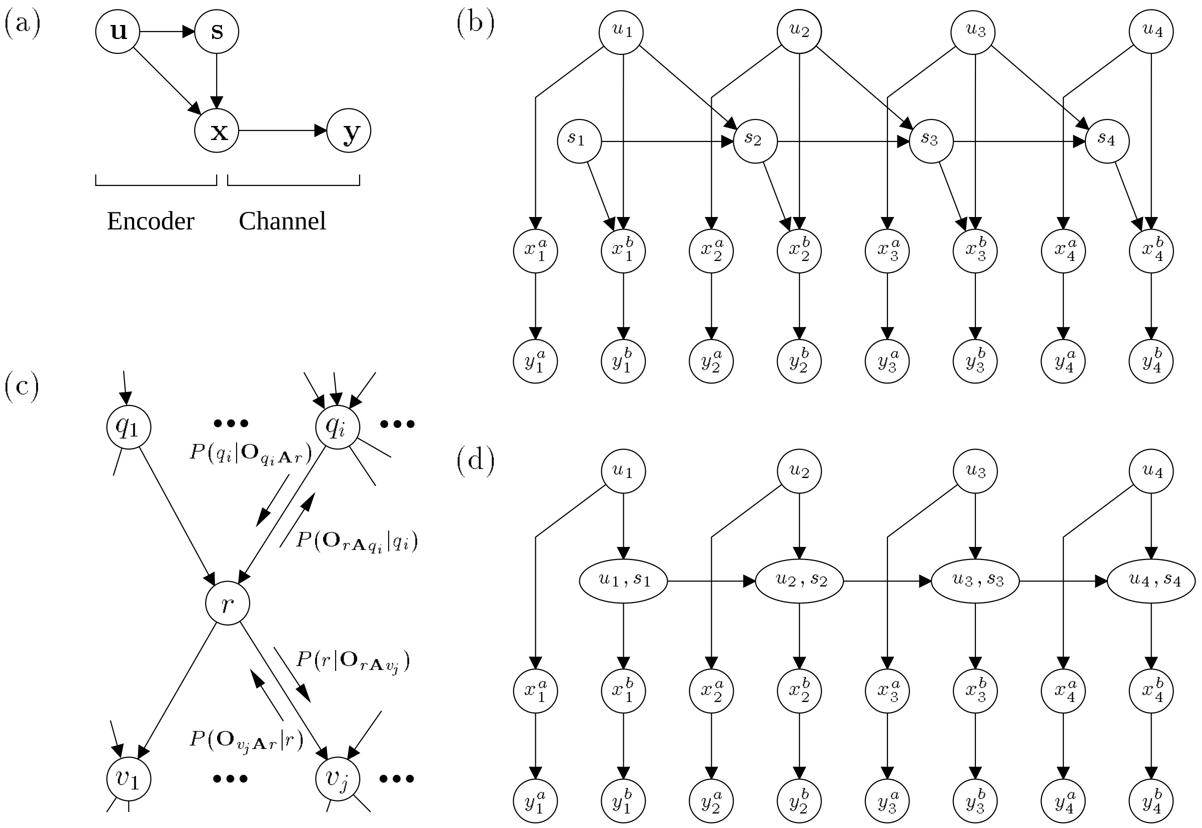

Fig. 4: (a) The Bayesian network for channel coding. (b) The Bayesian network for a systematic convolutional code and a memoryless channel. (c) A network fragment shows the parents q_i and the children v_j of variable r in addition to some of the incoming and outgoing probability messages. (d) A singly-connected network for the convolutional code from (b).

Fig. 4b shows the Bayesian network for a systematic convolutional code with a memoryless channel. The systematic codeword symbols are denoted by "a"; the symbols denoted by "b" are outputs of the convolver, where 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.

4 Probability Propagation

In this next section, we describe the basic probability propagation algorithm that forms the basis for iterative decoding. 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 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.

Fig. 4c shows a network fragment consisting of a variable r, its parents $\{q_i\}_{\forall i}$, and its children $\{v_j\}_{\forall j}$. Some of the other variables 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. 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.

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

$$P(r|\mathbf{O}) \overset{r}{\propto} \Big[\prod_{\forall j} P(\mathbf{O}_{v_j\mathbf{A}r}|r)\Big] \sum_{\{q_i\}_{\forall i}} \Big[P(r|\{q_i\}_{\forall i}) \prod_{\forall k} P(q_k|\mathbf{O}_{q_k\mathbf{A}r})\Big], \tag{5}$$

where the symbol "$\overset{r}{\propto}$" indicates equality up to a constant that is independent of r. (See [1] for derivations.) In order to get the actual value of $P(r|\mathbf{O})$, the quantity computed from the right hand side of (5) 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 $\{q_i\}_{\forall i}$ of the parents of r.

The outgoing parent-child messages $\{P(r|\mathbf{O}_{r\mathbf{A}v_j})\}_{\forall j}$ are computed as follows:

$$P(r|\mathbf{O}_{r\mathbf{A}v_j}) \overset{r}{\propto} P(r|\mathbf{O})/P(\mathbf{O}_{v_j\mathbf{A}r}|r). \tag{6}$$

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.

Up to a scaling factor, the outgoing child-parent messages $\{P(\mathbf{O}_{r\mathbf{A}q_i}|q_i)\}_{\forall i}$ are

$$P(\mathbf{O}_{r\mathbf{A}q_i}|q_i) \overset{q_i}{\propto} \sum_r \left(\Big[\prod_{\forall j} P(\mathbf{O}_{v_j\mathbf{A}r}|r)\Big] \sum_{\{q_k\}_{\forall k\neq i}} \Big[P(r|\{q_k\}_{\forall k}) \prod_{\forall l\neq i} P(q_l|\mathbf{O}_{q_l\mathbf{A}r})\Big] \right). \tag{7}$$

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 (5) and (6), which can be normalized.

These update equations provide a consistent way to propagate conditional probability messages across a singly-connected network. 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 Buffered Messages

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 (5), (6) and (7). 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, as shown in Fig. 5.

4.2 Message Creation and Absorption

Suppose that in Fig. 4c, 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, $P(r|\mathbf{O}) = \delta(r, r')$, where the indicator function $\delta(\cdot)$ satisfies $\delta(r, r') = 1$ if $r = r'$ and 0 otherwise. It follows from (6) that the parent-child messages $\{P(r|\mathbf{O}_{r\mathbf{A}v_j})\}_{\forall j}$ can be computed from

$$P(r|\mathbf{O}_{r\mathbf{A}v_j}) = \delta(r, r'). \tag{8}$$

In order to determine the newly created child-parent messages $\{P(\mathbf{O}_{r\mathbf{A}q_i}|q_i)\}_{\forall i}$, we pretend that r is not observed and imagine a fictitious child node z that *is* observed and to which r is connected deterministically: $P(z|r) = \delta(z, r)$. The authentic observation is equivalent to the fictitious observation, $z = r'$. When determining the child-parent messages from (7), the term $\prod_j P(\mathbf{O}_{v_j\mathbf{A}r}|r)$ will now include the fictitious child z. So, this term will be zero for all but the case $r = r'$ and the child-parent messages are

$$P(\mathbf{O}_{r\mathbf{A}q_i}|q_i) \overset{q_i}{\propto} \sum_{q_k : k\neq i} P(r = r'|\{q_m\}) \prod_{n\neq i} P(q_n|\mathbf{O}_{q_n\mathbf{A}r}). \tag{9}$$

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_j, new messages are sent to all other neighbors, but *not* back to v_j. 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 $P(\mathbf{O}_{r\mathbf{A}q_i}|q_i) = P(\varnothing|q_i) = 1$, for all q_i. Also, $P(r|\mathbf{O}_{r\mathbf{A}v_j}) = P(r|\varnothing) = P(r) = \sum_{\{q_i\}_{\forall i}} P(r|\{q_i\}_{\forall i}) \prod_j P(q_j)$. 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 (BCJR) Algorithm

As a familiar example of probability propagation, consider the Bayesian network for a convolutional code. The multiply-connected network shown in Fig. 4b can be converted to the singly-connected network shown in Fig. 4d. By grouping information and state variables together, we eliminate all undirected cycles, at the expense of increasing the complexity of the state variables. The *forward-backward algorithm* (a.k.a. the "BCJR algorithm") for computing $P(u_k|\mathbf{y})$ for such a Markov-type model was first developed by Baum and Petrie [12]. 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. 4d is shown by the first picture in Fig. 5. (We have included the codeword symbols with the state variables, for the sake of graphical simplicity.) When the channel output is observed (crossed circles), 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 passed *forward* along the chain. Then, a series of

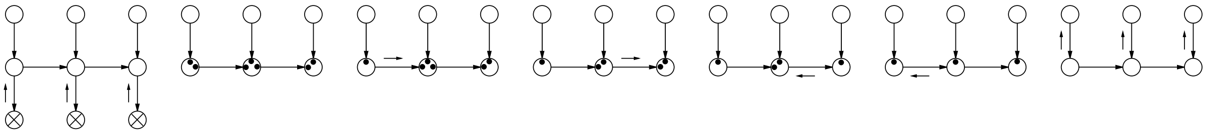

Fig. 5: Probability propagation in the Bayesian network for a convolutional code.

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 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.

5.1 Bayesian Networks for Some Known Codes

The Bayesian network for a systematic rate 1/3 compound turbo-code is shown in Fig. 6a, 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, the lower chain uses a permuted version of the information sequence.

A serially concatenated convolutional compound code [6] is shown in Fig. 6b. This system is essentially the same as Forney's concatenated codes [13], except that a convolutional code is used for both the inner and outer codes, instead of the more popular Reed-Solomon outer code.

A frame-oriented turbo-code [7] is shown in Fig. 6c. This code was designed to eliminate the need for trellis termination when using short block lengths.

Fig. 6d shows a low-density parity check code [2]. This code consists of parity check restrictions (upper vertices) on subsets of the codeword symbols (middle vertices). MacKay and Neal [4] were the first to describe Gallager's codes using Bayesian networks.

Fig. 6e shows a systematic product code. The distribution for the parity symbol in each row (column) is conditioned on the information symbols in its row (column), and is nonzero only for even parity row-wise (column-wise) configurations.

The Bayesian network for a convolutional code on a channel with 1st order memory is shown in Fig. 6f. In this case, the *codeword symbols* are permuted before transmission.

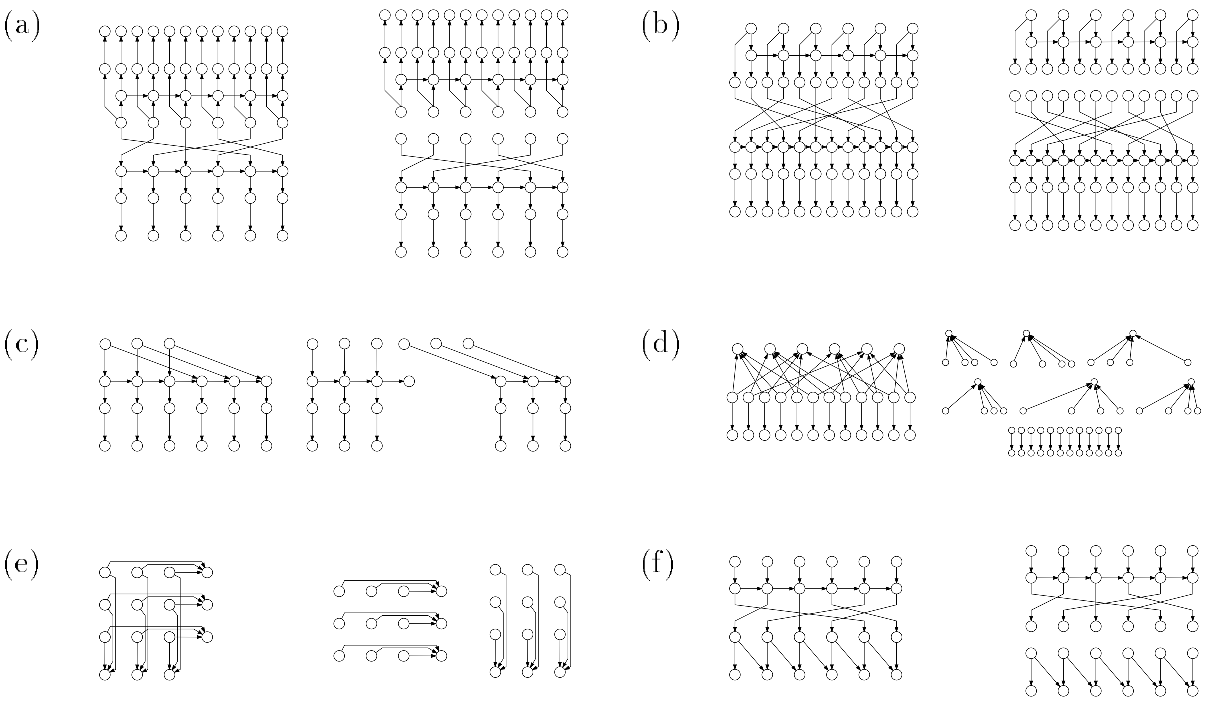

Fig. 6: The Bayesian networks for several coding systems: (a) a systematic turbo-code; (b) a nonsystematic serially concatenated convolutional code; (c) a nonsystematic frame-oriented turbo-code; (d) a low-density parity check code; (e) a product code; and (f) a convolutional code on a channel with 1st order memory, with permuted codeword symbols.

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 approximately taking them into account. This graphical framework unifies several iterative decoding algorithms. The iterative decoding algorithms for turbo-codes, serially concatenated convolutional codes, frame-oriented turbo-codes, low-density parity-check codes, product codes, and inter-symbol interference systems can all be viewed as probability propagation in the networks shown in Fig. 6.

The overall decoding procedure essentially consists of applying probability propagation while ignoring the graph cycles. The procedure is usually 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 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), before applying the MAP symbol 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 (*e.g.*, Fig. 6a). This algorithm produces the same results as the standard turbo-decoding algorithm, 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" [5, 9] 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.

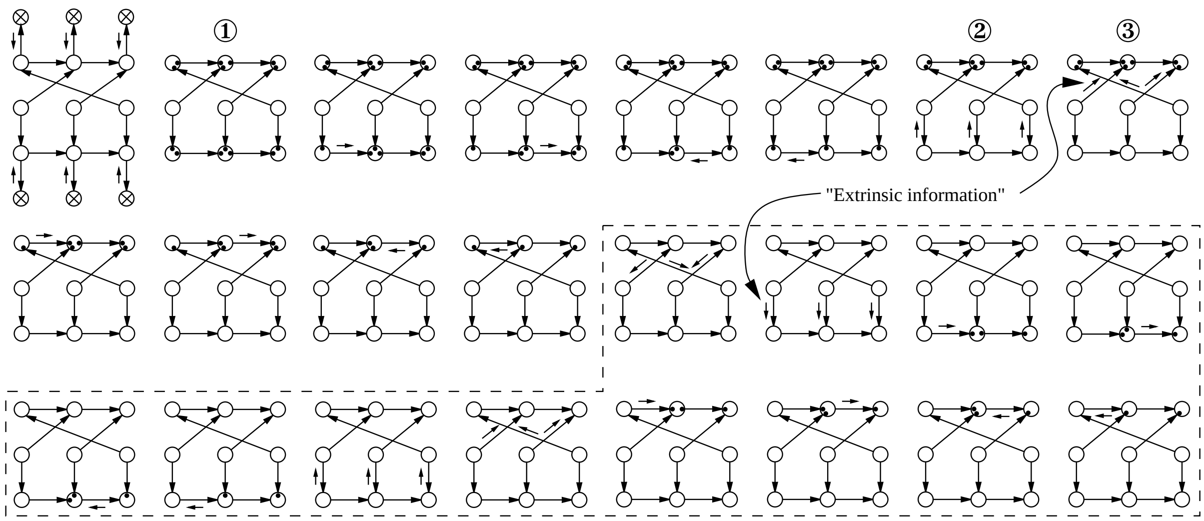

Fig. 7: Probability propagation in the Bayesian network for a turbo-code.

Fig. 7 shows the message passing dynamics for a simplified turbo-code Bayesian network. When the channel output is observed (as shown by the crossed circles), messages propagate up to the state vertices of both constituent networks. One constituent network is processed in the manner of the forward backward algorithm, ① to ②. The information symbol vertices receive probability messages from the constituent network just processed. In the case of the single code (Fig. 5), the information symbols are each connected by a single edge and so propagation terminates. In this case, however, a set of messages are then passed to the other constituent network ③ (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

We have reviewed graphical code models, including 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 to useful encoder structures, easily implemented physical models, and useful decoding algorithms.

We have described the probability propagation algorithm that computes the *a posteriori* distributions *exactly* in a singly-connected Bayesian network. Bayesian networks for many 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, as demonstrated by Berrou, *et al.* [5]. We have shown that many recently proposed iterative decoders can be described as message passing on a code graph.

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.

We appreciate the discussions we had with Glenn Gulak, David MacKay, Radford Neal and Niclas Wiberg.

References

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

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

[3] 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.

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

[5] 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.

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

[7] C. Berrou and M. Jezequel, "Frame-oriented convolutional turbo-codes," *Electronics Letters*, vol. 32, pp. 1362–1364, 1996.

[8] 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.

[9] 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.

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

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

[12] 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.

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