Early Detection and Trellis Splicing: Reduced-Complexity Soft Iterative Decoding

Working draft document — please do not distribute.

Brendan J. Frey
Department of Computer Science, University of Toronto
6 King's College Road, Toronto, Canada M5S 3H5
frey@cs.utoronto.ca

Frank R. Kschischang
Department of Electrical and Computer Engineering, University of Toronto
10 King's College Road, Toronto, Canada M5S 3G4
frank@comm.utoronto.ca

August 29, 1996

Abstract

The excellent bit error rate performance of new soft iterative decoding algorithms (*eg.*, turbo-codes) is achieved at the expense of a computationally burdensome iterative decoding procedure. In this paper, we present a new method called *early detection* that can be used to reduce the computational complexity of a variety of soft iterative decoding methods. Using a confidence criterion, some information symbols, state variables and codeword symbols are detected early on in the iterative decoding procedure, leading to a reduction in the computational complexity of further processing. After presenting a general Markov random field framework (*eg.*, the Tanner graph) for *compound* codes, we use this framework to show how early detection leads to computational savings. We then present an easily implemented instance of this algorithm, called *trellis splicing*, that can be used with turbo-codes. For a simulated turbo-code system, at low BERs we obtain a reduction in computational complexity of over a factor of four relative to conventional turbo-decoding, without any increase in BER.

1 Introduction

The impressive bit error rate performance of the *turbo-decoding* algorithm introduced by Berrou *et al.* [1] has led to an explosion of interest in the theory and application of a much more general class of error-correction decoding algorithms called *soft iterative decoding*. In this paper, we present a simple method for reducing the computational complexity of soft iterative decoders. The new method stems from our observation that the values of certain information symbols, state variables and codeword symbols can often be ascertained early on in the decoding process. By identifying such candidates and detecting them, we reduce the computational complexity of subsequent decoding. We refer to this general method as *early detection*.

Early detection can be applied in conjunction with a variety of soft iterative decoding algorithms. In general, these algorithms are used to decode *compound* codes (aka. "parallel concatenated codes"[1]). 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." In our case, the compound code is comprised of *constituent codes*. For a given codeword, all of the constituent codewords are generated from the same information block. As a consequence, they are interdependent and must be decoded interactively. The idea is that the error-correcting capabilities of the entire compound code far exceeds the capabilities of the constituent codes. Usually, but not always, each constituent code is highly structured. The compound code can contain many constituent codes which are short, or only a few constituent codes which are long. Obviously, extremely short constituent codes cannot be highly structured. On the other hand, the compound code presented by Berrou *et al.* [1] consists of two very long convolutional constituent codes[2].

[1] As many researchers have pointed out, the terms "parallel" and "concatenated" are misleading. Any code can be thought of as parallel in the sense of separate generator matrix rows. The term "concatenated" is loaded in the information theory literature; *eg.*, Forney used it long ago to refer not to codewords, but to concatenated encoders and decoders [2].

[2] We take the term *turbo-code* to refer to a compound code comprised of convolutional constituent codes.

Given a received word, the soft iterative decoder first extracts the constituent words and then operates in *cycles*. During each cycle, the decoder processes a single constituent code, using the soft decisions made by the previous cycles to bias the result. As pointed out by Wiberg *et al.* [3] and MacKay and Neal [4], the key ideas of soft iterative decoding are present in Gallager's 1963 work on low-density parity-check codes [5]. In 1981, Tanner derived some lower bounds on rate and minimum distance for similar "recursive" codes [6]. A large body of interesting work that ranges from the very practical to the very theoretical has recently emerged [3, 4, 7–15].

Here is an outline of this paper. In Section 2, we present a view of compound codes as Markov random fields [6, 16, 17] and show that Tanner graphs [6, 18]) are a special case of Markov random fields. We then present the general *probability propagation* decoding algorithm which was first derived by Gallager [5] and then later in the expert systems literature by Pearl [19]. The graphical model framework is a useful tool for understanding the probabilistic mechanics of soft iterative decoding as well as the procedure and computational implications of early detection. Readers that are intimately familiar with soft iterative decoding and readers that are primarily interested in turbo-codes may wish to skip this somewhat tutorial section on graphical models.

In Section 3, we first give an intuition for early detection using a turbo-code example. Then, we present a specific method called *thresholded early detection* that is used to ascertain which symbols or state variables to detect early. We also discuss what types of criteria are appropriate for terminating the iterative decoding procedure, when early detection is used.

In Section 4, we show how, in general, the compound code graph is simplified when an information symbol, a state variable or a codeword symbol is early-detected. It turns out that each of these types of variable leads to a different degree of simplification for a given graph. To conclude this section, we relate the simplification of the compound code graph to the computational reduction for subsequent decoding. The following sections can be easily understood without the general ideas presented in this section, so readers that are primarily interested in turbo-codes may wish to skip this section.

In Section 5, we consider information symbol early detection for trellises, and derive the *trellis splicing* algorithm, which is well-suited to the turbo-decoder. We analyse the additional computational complexity and memory required by trellis splicing and for processing the resulting spliced trellis. It turns out that the additional complexity and memory requirements are minor. Also, the software algorithm for performing trellis splicing is quite simple.

In Section 6, we present results for thresholded early detection of information symbols in a turbo-code system, using trellis splicing to simplify the compound code graph. Here, we find that for a specified BER there is an optimal threshold that will lead to the greatest reduction in computational complexity.

Finally, in Section 7, we comment on the usefulness of thresholded early detection over the parameter ranges that we have explored. We describe some future research directions meant to find the ideal conditions for early detection.

2 Graphical Models and Soft Iterative Decoding

In the error-correction field, we are interested in finding an information symbol sequence that maximizes the *a posteriori* distribution over information symbols given the observed channel output. For example, if X_k is the random variable representing an information symbol and $\mathbf{y}$ is the observed sequence of received signals, we can minimize the rate of information symbol errors by choosing the x_k that maximizes $P(X_k = x_k|\mathbf{y})$. Alternatively, if $\mathbf{X}$ is the random variable representing an information symbol sequence, we can minimize the rate of information *sequence* errors (word errors) by choosing the $\mathbf{x}$ that maximizes $P(\mathbf{X} = \mathbf{x}|\mathbf{y})$. Many codes are very naturally described in graphical models [3–6, 20–22], and in fact every block code can be represented graphically [23, 24]. A graphical model consists of a graph which depicts dependencies between information symbols, state variables and codeword symbols, in addition to local probabilistic information (*eg.*, signal likelihoods). Not only is the graphical model formalism a useful way to enforce code structure, but it also implies a computational paradigm for encoding and decoding. For example, soft iterative decoding is fundamentally related to the method of *probability propagation* [19, 25] which is an algorithm used for computing marginal probabilities (*eg.*, $P(X_k = x_k|\mathbf{y})$) using graphical models (see textbook references [17, 26]).

Gallager [5] and Pearl [19] have independently shown that probability propagation is an *exact* algorithm for computing conditional probabilities when the graphical model is *singly-connected*. A singly-connected graph is one that has only a single path connecting any two nodes (otherwise, the graph is multiply-connected). For example, if we associate a graph node with the state variable of a trellis at each time step, then the entire trellis forms a Markov chain which is singly connected. An algorithm for computing $P(X_k = x_k|\mathbf{y})$ on a Markov chain was developed by Baum and Petrie [27] as well as by Bahl *et al.* [28] (who cite [27]). The former authors called their procedure the *forward-backward* algorithm. Although there has recently been a tendency to refer to this algorithm as the "BCJR" algorithm (after the initials of the latter authors), we prefer to use the original name since it is more descriptive. In any case, it turns out that this algorithm is identical to the algorithm that results when probability propagation is applied to a trellis. Furthermore, it turns out that soft iterative decoding is probability propagation applied

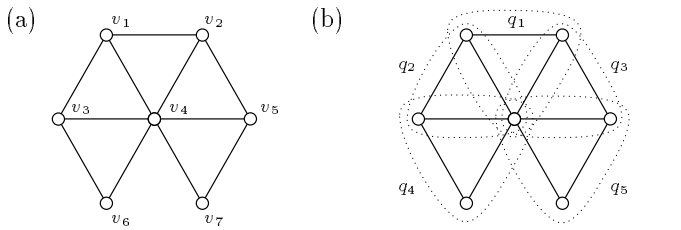

Figure 1: (a) The graph for a Markov random field (MRF). (b) The cliques for this graph are shown by dotted loops.

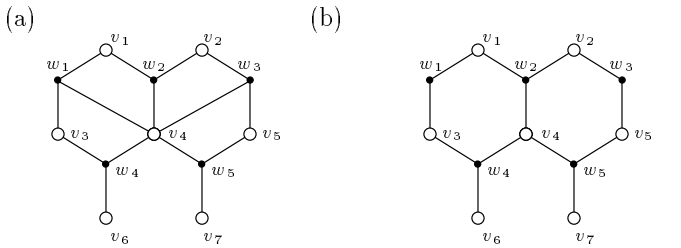

Figure 2: (a) A *Tanner graph* that is equivalent to the MRF graph shown in Fig. 1. (b) A Tanner graph that specifies more detailed structure than the more general MRF graph can illustrate.

to multiply-connected graphical models that describe compound codes. Because probability propagation is only approximate in this case, the entire propagation procedure must be iterated until some termination criterion is satisfied. As an example of a graphical model that is *not* singly-connected, consider the convolutional turbo-code of Berrou *et al.* [1], which has a trellis corresponding to each of two convolutional constituent codes. Two adjacent sections in one trellis are connected directly in that trellis as well as indirectly via the information symbols through the other trellis.

2.1 Markov Random Fields

A Markov random field (MRF) consists of a set of *variable* vertices V; a set of undirected edges E, such that the vertex pair (u, v) is in E if and only if vertex u is connected to vertex v; a set of random variables $\mathbf{X}$ with one element $X_v \in F_v$ for each variable vertex v (F_v is the field of X_v); and a set of *potential functions* which define a joint distribution over the random variables. An example of the graph for a MRF is shown in Fig. 1a. A *clique* is a maximal complete subgraph of the MRF; all of the cliques for the above MRF are shown by dotted loops in Fig. 1b. Each clique q in the complete set of cliques Q has a subset of vertices V_q. Given a nonnegative potential function for each clique $q \in Q$,

$$\psi_q^{\mathrm{MRF}}(\{X_v : v \in V_q\}) : \bigotimes_{v \in V_q} F_v \to \mathbb{R},$$

the joint distribution over the random variables is given by

$$P^{\mathrm{MRF}}(\mathbf{X}) \overset{\mathbf{X}}{\propto} \prod_{q \in Q} \psi_q^{\mathrm{MRF}}(\{X_v : v \in V_q\}),$$

where $\overset{\mathbf{X}}{\propto}$ means that the constant of proportionality is *independent* of $\mathbf{X}$. For the MRF shown in Fig. 1, we have

$$\begin{aligned} P^{\mathrm{MRF}}(\mathbf{X}) \overset{\mathbf{X}}{\propto}\ & \psi_{q_1}^{\mathrm{MRF}}(X_{v_1}, X_{v_2}, X_{v_4}) \psi_{q_2}^{\mathrm{MRF}}(X_{v_1}, X_{v_3}, X_{v_4}) \\ & \cdot \psi_{q_3}^{\mathrm{MRF}}(X_{v_2}, X_{v_4}, X_{v_5}) \psi_{q_4}^{\mathrm{MRF}}(X_{v_3}, X_{v_4}, X_{v_6}) \\ & \cdot \psi_{q_5}^{\mathrm{MRF}}(X_{v_4}, X_{v_5}, X_{v_7}). \end{aligned} \quad (1)$$

A MRF can be used to describe a code in a variety of ways. If each random variable X_v corresponds to a codeword symbol, then the potentials provide a highly local (in graph topology) description of the code. For example, the MRF discussed above can be used to specify a code that satisfies even parity locally in $GF(2)$:

$$\begin{aligned} \psi_{q_1}^{\mathrm{MRF}}(X_{v_1}, X_{v_2}, X_{v_4}) &= X_{v_1} + X_{v_2} + X_{v_4} + 1 \\ \psi_{q_2}^{\mathrm{MRF}}(X_{v_1}, X_{v_3}, X_{v_4}) &= X_{v_1} + X_{v_3} + 1 \\ \psi_{q_3}^{\mathrm{MRF}}(X_{v_2}, X_{v_4}, X_{v_5}) &= X_{v_2} + X_{v_5} + 1 \\ \psi_{q_4}^{\mathrm{MRF}}(X_{v_3}, X_{v_4}, X_{v_6}) &= X_{v_3} + X_{v_4} + X_{v_6} + 1 \\ \psi_{q_5}^{\mathrm{MRF}}(X_{v_4}, X_{v_5}, X_{v_7}) &= X_{v_4} + X_{v_5} + X_{v_7} + 1, \end{aligned} \quad (2)$$

From (1) it is easy to see that the MRF places equal probability mass on all codewords that satisfy the local parity equations described above.

2.2 Tanner Graphs Are MRFs

For the MRF code with the potentials given in (2), $\psi_{q_2}^{\mathrm{MRF}}(X_{v_1}, X_{v_3}, X_{v_4})$ does not depend on X_{v_4} (*ie.*, X_{v_4} is not included in the parity check equation). However, this fact is *not* represented graphically in the MRF graph shown in Fig. 1. A special type of *bipartite* MRF, called a *Tanner graph* [6, 29], can be used to make the dependency structure more explicit[3]. For the Tanner graph, we augment the original set of MRF variable vertices V with a disjoint set of vertices W, which we call *clique vertices*. For each clique q in the original MRF, there is one clique vertex in the Tanner graph, which is connected to all of the variable vertices that ψ_q depends on. The Tanner graph for the general MRF graph shown in Fig. 1a is illustrated in Fig. 2a, where variable vertices in V are shown as large unfilled discs and clique vertices in W are shown as small filled discs. Each clique vertex w is connected to all of the variable vertices V_q in the corresponding clique, q. For the more specific potentials given in (2), we get a Tanner graph

[3] A *bipartite* graph is one with two nonoverlapping sets of vertices such that each vertex is not connected to any of the vertices in the same set.

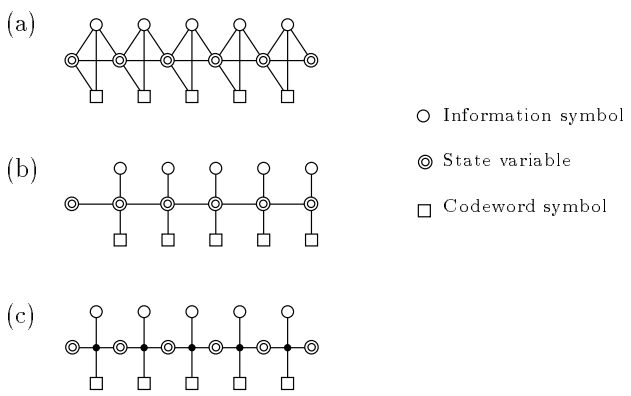

Figure 3: (a) The minimal-memory MRF for a trellis. (b) A simpler extended-memory trellis MRF. (c) The Tanner graph for a trellis.

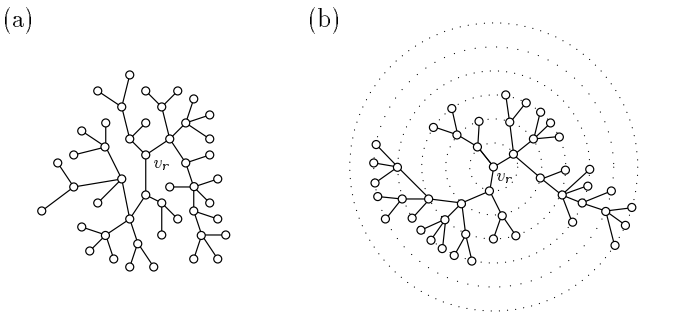

Figure 4: (a) The connectivity of a singly-connected MRF. (b) Vertices are ordered by selecting a root v_r, laying out the vertices concentrically, and then ordering them radially.

with fewer edges, as shown in Fig. 2b. For example, this graph shows that the potential for the clique associated with clique vertex w_1 does not depend on X_{v_4}.

Since each clique vertex in a Tanner graph represents the variables on which the corresponding potential depends, it is natural to associate these variables with the clique vertex. Formally, each clique vertex w has a corresponding vector random variable Z_w whose elements are the random variables for the vertices to which the clique vertex connects. For the potentials given in (2), we have

$$\begin{aligned} Z_{w_1} &= (X_{v_1}, X_{v_3}) \\ Z_{w_2} &= (X_{v_1}, X_{v_2}, X_{v_4}) \\ Z_{w_3} &= (X_{v_2}, X_{v_5}) \\ Z_{w_4} &= (X_{v_3}, X_{v_4}, X_{v_6}) \\ Z_{w_5} &= (X_{v_4}, X_{v_5}, X_{v_7}). \end{aligned}$$

Since Z_w contains the relevant random variables for all of clique vertex w's neighbors, in a Tanner graph we need only specify one potential for each clique vertex in the graph:

$$P^{\text{TG}}(\mathbf{X}) \overset{\mathbf{X}}{\propto} \prod_{w \in W} \psi_w^{\text{TG}}(Z_w).$$

As another example, consider the MRF for a minimal-memory trellis, as shown in Fig. 3a. We use unfilled discs to represent information symbols, unfilled squares to represent codeword symbols, and a pair of unfilled nested discs to represent state variables. The last symbol was introduced by Wiberg [18]. Each codeword symbol depends on the previous state and the current information symbol. Each state also depends on the previous state and the current information symbol. By extending the state memory, we can draw a simpler trellis MRF, as shown in Fig. 3b. The Tanner graph shown in Fig. 3c shows how the current information symbol, the current codeword symbol, the previous state variable and the next state variable must all satisfy a set of local constraints.

Although Tanner graphs may be more explicit about dependencies than equivalent MRFs (cf. Fig. 2b and Fig. 1a), the Tanner graph in general is just a special case of the MRF. So, many results and algorithms that have been developed for MRFs are applicable to codes based on Tanner graphs. In particular, various algorithms that can be used for *decoding* have been developed, including Markov chain Monte Carlo [30, 31], mean field methods [4, 32, 33], the generalized Viterbi algorithm [6, 18] and probability propagation [5, 19, 26]. The last of these algorithms turns out to be the underlying method for the turbo-decoding algorithm and other soft iterative decoding algorithms. Before presenting the method of probability propagation, we describe singly-connected MRFs for which the method is exact.

2.3 Singly-Connected MRFs

The graph for a singly-connected MRF is restricted so that there is only one path between any two vertices (cf. Fig. 4a). It will be useful to think of the vertices as having an ancestral ordering that is specified by arbitrarily selecting a vertex as the root v_r and then laying out the remaining vertices concentrically as shown in Fig. 4b. Vertices that are equidistant from the root are ordered arbitrarily. We use the notation $v < u$ to mean that vertex v is earlier in this ancestral ordering than vertex u. In this ordering, each vertex $v \neq v_r$ is directly connected to a single vertex ρ_v that is closer to the root, called the *parent* of v. The set of all vertices that have the same parent v are called the *children* of v; N_v is the number of children of vertex v. In a Tanner graph, it is obvious that both the parent and the children of a vertex are in the opposite vertex set. In a singly-connected MRF, each edge corresponds to a clique, so the potentials are defined for all $u, v \in V : (u, v) \in E$:

$$\psi_{uv}^{\text{MRF}}(X_u, X_v) : F_u \times F_v \to \mathbb{R}.$$

Using the ordering described above, the joint distribution over the entire set of random variables $\mathbf{X}$ is given by

$$P^{\text{MRF}}(\mathbf{X}) \overset{\mathbf{X}}{\propto} \psi^{\text{MRF}}_{v_r}(X_{v_r}) \prod_{\substack{u \in V: \\ u \neq v_r}} \psi^{\text{MRF}}_{u\rho_u}(X_u, X_{\rho_u}), \qquad (3)$$

where the leading term, $\psi^{\text{MRF}}_{v_r}(X_{v_r})$, is a potential particular to the root. Using Bayes' rule, the joint distribution can also be written as a product of radially conditioned probabilities:

$$P^{\text{MRF}}(\mathbf{X}) = P^{\text{MRF}}(X_{v_r}) \prod_{\substack{u \in V: \\ u \neq v_r}} P^{\text{MRF}}(X_u | \{X_v : v < u\}). \qquad (4)$$

Identifying terms in the products of (3) and (4), we find that $P^{\text{MRF}}(X_u|\{X_v : v < u\})) \overset{X_u}{\propto} \psi^{\text{MRF}}_{u\rho_u}(X_u, X_{\rho_u})$. This potential only depends on X_u and X_{ρ_u}, and so $P^{\text{MRF}}(X_u|\{X_v : v < u\}) = P^{\text{MRF}}(X_u|X_{\rho_u})$. This means that if the value of the random variable associated with vertex v is known, then the distribution over the descendents of v does not depend on the random variables in any other part of the graph. This also allows us to express the local potentials as follows:

$$P^{\text{MRF}}(X_{v_r}) \overset{X_{v_r}}{\propto} \psi^{\text{MRF}}_{v_r}(X_{v_r}), \text{ and}$$
$$P^{\text{MRF}}(X_u|X_{\rho_u}) \overset{X_u}{\propto} \psi^{\text{MRF}}_{u\rho_u}(X_u, X_{\rho_u}), \forall\, u \in V : u \neq v_r.$$

So, as an alternative to the set of potential functions, a singly-connected MRF can be specified by listing one local radially conditioned probability distribution for each edge.

For a Tanner graph, the random variable associated with each clique vertex consists of the random variables associated with the vertices to which it connects. As a result, for edges where a clique vertex is the parent, the conditional probability is trivially unity, since

$$P^{\text{TG}}(X_v|Z_{\rho_v}) = P^{\text{TG}}(X_v|X_v, X_{\rho_{\rho_v}}) = 1.$$

So, for a Tanner graph, we need only specify one local radially conditioned probability distribution for each edge where the child is a clique vertex.

For example, consider the Tanner graph shown in Fig. 5a, where $Z_{w_1} = (X_{v_1}, X_{v_2}, X_{v_r})$, $Z_{w_2} = (X_{v_3}, X_{v_5}, X_{v_r})$, $Z_{w_3} = (X_{v_4}, X_{v_6}, X_{v_r})$, and $Z_{w_4} = (X_{v_7}, X_{v_8}, X_{v_r})$. To specify a code that satisfies even parity locally, we select a root node v_r as shown and then write out one conditional distribution for each edge where

(a) (b)

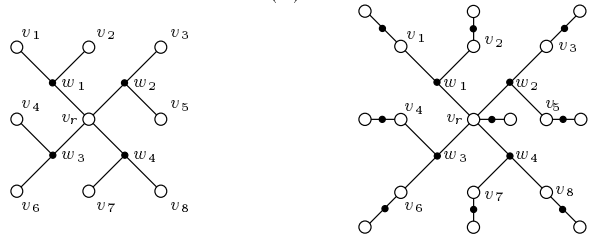

Figure 5: (a) A singly-connected Tanner graph that describes a code. (b) The Tanner graph used for decoding. The extra variable vertices correspond to the received noise-correpted codeword symbols and the extra clique vertices model the channel noise (assumed memoryless in this case).

the child is a clique vertex:

$$\begin{aligned}
P^{\text{TG}}(X_{v_r}) &= 1 \\
P^{\text{TG}}(Z_{w_1}|X_{v_r}) &= P^{\text{TG}}(X_{v_1}, X_{v_2}, X_{v_r}|X_{v_r}) \\
&= P^{\text{TG}}(X_{v_1}, X_{v_2}|X_{v_r}) \\
&\overset{X_{v_1}, X_{v_2}}{\propto} X_{v_1} + X_{v_2} + X_{v_r} + 1 \\
P^{\text{TG}}(Z_{w_2}|X_{v_r}) &\overset{X_{v_3}, X_{v_5}}{\propto} X_{v_3} + X_{v_5} + X_{v_r} + 1 \\
P^{\text{TG}}(Z_{w_3}|X_{v_r}) &\overset{X_{v_4}, X_{v_6}}{\propto} X_{v_4} + X_{v_6} + X_{v_r} + 1 \\
P^{\text{TG}}(Z_{w_4}|X_{v_r}) &\overset{X_{v_7}, X_{v_8}}{\propto} X_{v_7} + X_{v_8} + X_{v_r} + 1.
\end{aligned}$$

Now that we have developed the above probabilistic interpretation for the singly-connected MRF, we can easily show how to exactly decode a noise-corrupted codeword that was produced by a singly-connected MRF or Tanner graph.

2.4 Encoding and Decoding: Probability Propagation in Singly-Connected MRFs

Suppose that in the Tanner graph of Fig. 5a, random variables X_{v_1}, X_{v_3}, X_{v_3} and X_{v_8} (corresponding to the corner vertices) form the systematic part of a $(9, 4)$ code. In order to encode this systematic component, we must fix these variables to the source values x_{v_1}, x_{v_3}, x_{v_6} and x_{v_8} and then somehow determine a valid configuration for the remaining variables X_{v_2}, X_{v_4}, X_{v_r}, X_{v_5}, and X_{v_7}. Although in the present case we may accomplish this by inspection, for more complex graphs we will need a low-complexity automated procedure. We have a similar problem for decoding when we receive $\mathbf{y}$, a noise-corrupted version of $\mathbf{X}$. The Tanner graph for the decoding problem is shown in Fig. 5b, where extra variable vertices represent the random variables for the received signals. Taking v' as the vertex corresponding to the received signal for the codeword symbol corresponding to v, the local conditional distributions

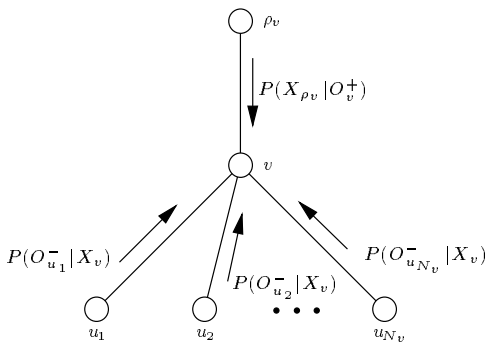

Figure 6: A fragment of a MRF illustrates the local computations used for probability propagation.

$P(Y_{v_{k'}}|X_{v_k})$ for the received signals are specified according to a channel model. Here, we are interested in symbol-wise MAP, so we fix the values of $\mathbf{Y}$ to the received values $\mathbf{y}$ and then determine an $\mathbf{x}$ that maximizes the marginal *a posteriori* distributions, $P(X_{v_k} = x_{v_k}|\mathbf{Y} = \mathbf{y})$. Again, we need a low-complexity automated procedure for computing these distributions.

Probability propagation is an exact method for computing the distribution each variable in a singly-connected MRF, given the values for an observed subset of variables[4]. The method consists of highly local computations that occur at each vertex. We assume that the values for some subset of random variables are given (*eg.*, observed channel output), and we refer to these collectively as O. Consider an unobserved vertex v, with parent ρ_v and children $u_1, \ldots, u_{N_v}$ as shown in Fig. 6. Let $O_{u_k}^-$ refer to the observed random variables associated with vertex u_k or those vertices that are connected to u_k from *below*. Let O_v^+ refer to the observed random variables associated with those vertices that are connected to v from *above*, not including v itself, which is unobserved by assumption. Suppose that the distributions $P(X_{\rho_v}|O_v^+)$ and $P(O_{u_k}^-|X_v), \forall k$ are available at vertex v as shown in Fig. 6. (These can be thought of as local messages being passed from above and below.) Using the conditional distributions $P(X_v|X_{\rho_v})$ and $P(X_{u_k}|X_v)$, which are part of the MRF specification, we can easily compute the probability for X_v conditioned on O_v^+:

$$P(X_v|O_v^+) = \sum_{X_{\rho_v}} P(X_v|X_{\rho v}) P(X_{\rho_v}|O_{\rho_v}^+). \tag{5}$$

Using the fact that $O_v^- = (O_{u_1}^-, O_{u_2}^-, \ldots, O_{u_{N_v}}^-)$, the likelihood of O_v^- conditioned on X_v is computed from

$$\begin{aligned} P(O_v^-|X_v) &= P(O_{u_1}^-, O_{u_2}^-, \ldots, O_{u_{N_v}}^-|X_v) \\ &= \prod_k P(O_{u_k}^-|X_v). \end{aligned} \tag{6}$$

[4] The method of probability propagation is also applicable to other types of graphical models, including Bayesian networks [17].

Finally, the distribution over X_v conditioned on all the observed random variables is computed by

$$\begin{aligned} P(X_v|O) &= P(X_v|O_v^-, O_v^+) \overset{X_v}{\propto} P(X_v, O_v^-|O_v^+) \\ &= P(X_v|O_v^+) P(O_v^-|X_v). \end{aligned} \tag{7}$$

So, given the probability "messages" shown in Fig. 6, equations (5), (6), and (7) can be used to compute the marginal distribution for X_v conditioned on the given random variables.

It is easy to see how vertex v can now produce appropriate outgoing probability messages. The probability $P(O_v^-|X_{\rho_v})$, which is passed up to ρ_v, is computed from

$$P(O_v^-|X_{\rho_v}) = \sum_{X_v} P(O_v^-|X_v) P(X_v|X_{\rho_v}). \tag{8}$$

The probability $P(X_v|O_{u_k}^+)$, which is passed down to each u_k, must take into account not only the observations connected to v from above, but also the observations connected from below to $u_{k'}, \forall k' \neq k$. It is computed from

$$\begin{aligned} P(X_v|O_{u_k}^+) &= P(X_v|O_v^+, \{O_{u_{k'}}^- : k' \neq k\}) \\ &\overset{X_v}{\propto} P(X_v, \{O_{u_{k'}}^- : k' \neq k\}|O_v^+) \\ &= P(X_v|O_v^+) P(\{O_{u_{k'}}^- : k' \neq k\}|X_v) \\ &= P(X_v|O_v^+) \prod_{k' \neq k} P(O_{u_{k'}}^-|X_v). \end{aligned} \tag{9}$$

To compute this top-down message, we must first have available the bottom-up messages for the other children.

Using these rules for computing local probabilities, it is possible to compute $P(X_v|O)$ for each unobserved vertex v in the MRF. Without loss of generality, we assume that the observed random variables correspond to leaf vertices. (If the random variable corresponding to a nonleaf vertex v is observed, then the distribution over its descendents does not depend on the random variables in any other part of the graph. Then, probabilities can be propagated in a forest of independent trees.) Suppose that in Fig. 6, X_{u_1} is known to have the value x_{u_1}. The probability message $P(O_{u_1}^-|X_v)$ is then equal to $P(X_{u_1} = x_{u_1}|X_v)$, which can be determined directly from the MRF specification. We continue to propagate these probabilities towards the root, level by level, storing them as we go (see the concentric rings in Fig. 4b). Once the root has been reached, we begin to propagate probabilities back out to the leaves, again storing them as we go. Finally, we compute the marginals for each unobserved vertex, using (7). This two-pass procedure can be thought of as a generalization of the forward-backward algorithm.

As with the forward-backward algorithm, the computational complexity of probability propagation is reasonably low. The number of operations needed to compute (5) and (8) scales with $F_v F_{\rho_v}$. For (6), the number of operations scales with $F_v N_v$, where N_v is the number of children of

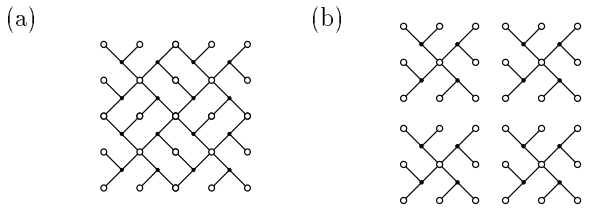

Figure 7: (a) The MRF for a compound code. (b) The MRFs for the corresponding four constituent codes.

vertex v. For (9), the number of operations scales with N_v^2.

If an observed leaf vertex in a MRF has only a single edge (which is always the case for a singly-connected MRF), then the vertex will contribute a single constant message to the propagation procedure. In the above example, where vertex u_1 is observed, the probability distribution $P(X_{u_1} = x_{u_1}|X_v)$ is propagated upwards from u_1. This message stays constant during the two-pass procedure and so the vertex u_1 can be left out of the graph for the purpose of illustrating the probability propagation dynamics. Often, we will draw the MRF without the observed leaf vertices with the understanding that the constant distributions that they contribute are taken into account while propagating probabilities. As an example, for the Tanner graph shown in Fig. 5b, we will usually leave out the vertices v_k' and simply present the Tanner graph shown in Fig. 5a, with the understanding that each vertex v_k takes into account any directly related observations.

2.5 MRFs for Compound Codes

We define a *compound* graphical code to be one which can be described by a set of singly-connected constituent MRFs, which are joined together so that the total MRF is not singly-connected. The MRF for an example of such a compound code is shown in Fig. 7a. This example illustrates that there is often not a unique decomposition of a compound code into its constituent codes. However, usually the compound code is designed using well-known constituent codes. For this example, the compound code was constructed from four of the same constituent codes of the type shown in Fig. 5a. The compound code is shown broken into singly-connected pieces in Fig. 7b.

A compound code which has recently indicated great promise at low SNRs (at least for moderate BERs) is the turbo-code. The MRF for the turbo-code is shown in Fig. 8a. This compound code consists of two trellises that are connected at the information symbol vertices. The decomposed code is shown in Fig. 8b. Each of the constituent MRFs are joined to the information symbol vertices using a different pseudo-random ordering.

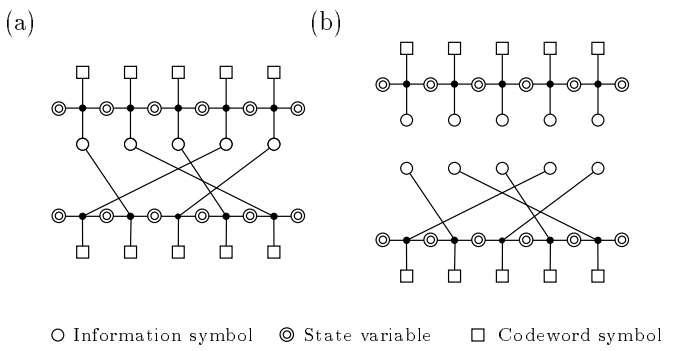

Figure 8: (a) The MRF for the turbo-code compound code. (b) The constituent MRFs for the turbo-code.

2.6 Probability Propagation for Compound Codes

Since each constituent MRF is singly-connected, probabilities for each random variable given the observed random variables can be efficiently computed *exactly* within each constituent MRF. However, because the compound MRF is multiply-connected, probability propagation on the compound MRF is only an approximate algorithm for computing these probabilities. As we see it, the general idea of soft iterative decoding is to make use of the efficient probability propagation algorithm for each constituent MRF. The overall decoding procedure consists of a series of processing cycles. In each cycle, probabilities are propagated across a particular constituent MRF, producing a set of probability messages. These messages can be used to compute 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 messages produced by the previous cycle when processing the next constituent MRF. Usually, the constituent codes are processed in order and one pass through all of the codes is called an *iteration*. Because the compound code MRF is multiply-connected, the overall propagation procedure never terminates. Instead, this cyclic procedure is allowed to iterate until some termination criterion is satisfied (*eg.*, 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.

Fig. 9 shows the flowchart for soft iterative decoding. In general, there are J constituent codes indexed by $j \in \{0, \ldots, J-1\}$: $C_0, \ldots, C_j, \ldots, C_{J-1}$. For cycle i, the MRF for constituent code $C_{m(i)}$ is used, where $m()$ is an integer mapping that specifies which constituent code to use for a given cycle number. Usually, the constituent codes are processed in order; *eg.*, $m(i) = i \bmod J$.

The above procedure produces soft decisions at the end of each cycle for the information symbols, state variables,

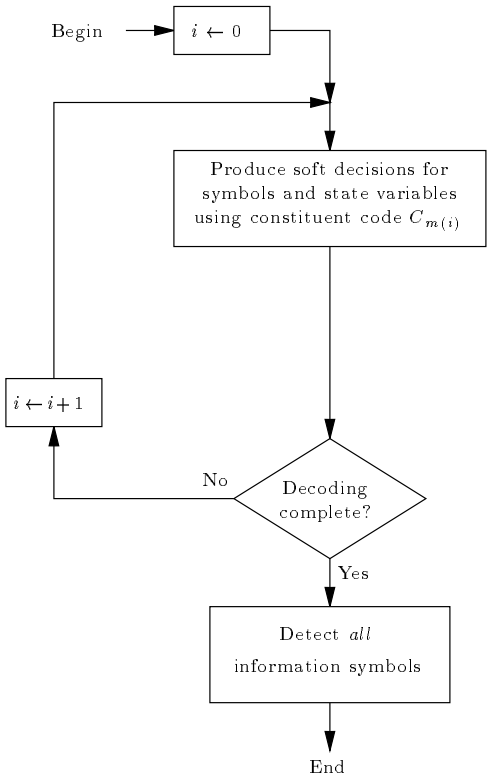

Figure 9: The flowchart for soft iterative decoding using multiple constituent codes. For example, in [1] each constituent code consists of a convolutional code produced using a different permutation of the information symbols.

and codeword symbols. A soft decision at cycle i for random variable X_v takes the form of a log-odds value, $\tilde{L}_v^i(x_v')$, defined for each possible value of $x_v' \in F_v$. $\tilde{L}_v^i(x_v')$ is an approximation to the *a posteriori* log-odds for random variable X_v, given the observed channel output. If $\mathbf{y} = (\mathbf{y}_0, \ldots, \mathbf{y}_{J-1})$ is the total received signal sequence (including each received signal sequence $\mathbf{y}_j$ for constituent code C_j), then

$$\tilde{L}_v^i(x_v') = \log \frac{\tilde{P}^i(X_v = x_v'|\mathbf{y})}{\tilde{P}^i(X_v \neq x_v'|\mathbf{y})}, \qquad (10)$$

where $\tilde{P}^i(X_v|\mathbf{y})$ is the approximation to the *a posteriori* distribution over X_v produced at cycle i. Equivalently, the log-odds can be viewed as a decision accompanied by a reliability value:

$$\tilde{x}_v^i = \text{argmax}_{x_v'} \tilde{L}_v^i(x_v'), \quad \tilde{R}_v^i = |\tilde{L}_v^i(\tilde{x}_v^i)|.$$

Once the iterative decoding procedure is complete, the current information symbol decisions are used as estimates of the information symbol values.

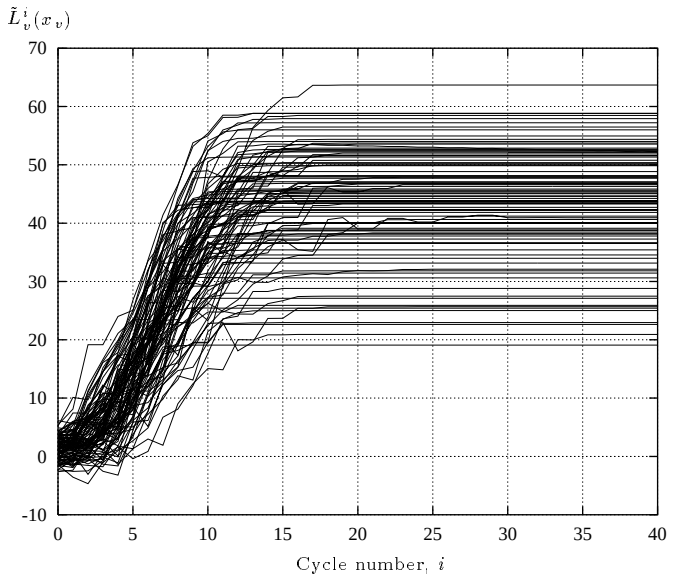

Figure 10: A plot of the log-odds versus decoding cycle number, for the correct value of each information bit in an unbiased sample of 100 cases (the code is described in Section 3).

3 Early Detection

Often, after only a relatively small number of cycles, many of the information symbols, state variables and codeword symbols can be detected with a low rate of error. Consider a systematic rate 1/3 turbo-coding system using two constituent convolutional codes with identical encoder transfer functions, $(21)_{\text{octal}}/(37)_{\text{octal}}$. The binary information block length is 1000 bits and there is a random interleaver at the input to the second encoder. The system uses binary signaling over an $E_b/N_0 = 0.8$ dB AWGN channel. Fig. 10 shows how the approximate *a posteriori* log-odds (10) changes during decoding, for each information bit in an unbiased sample of 100 cases. The log-odds values for the correct information bit values are shown, so that a positive value indicates the bit is being correctly decoded, whereas a negative value indicates the bit is being erroneously decoded. Values at $i = 0$ are equal to the signal likelihood log-odds for the uncoded systematic component. Approximately 21% of these initially have the wrong sign, indicating that threshold detection for the uncoded information block would result in a BER of 0.21. The log-odds for all cases shown eventually go positive during decoding. Most of the information bits shown can be correctly detected after only a few cycles and all of the bits shown can be correctly detected after 6 cycles.

In general, knowing the values of some information symbols, state variables, and codewords symbols reduces the computational requirements for making soft decisions for the remaining symbols. When the value of a random vari-

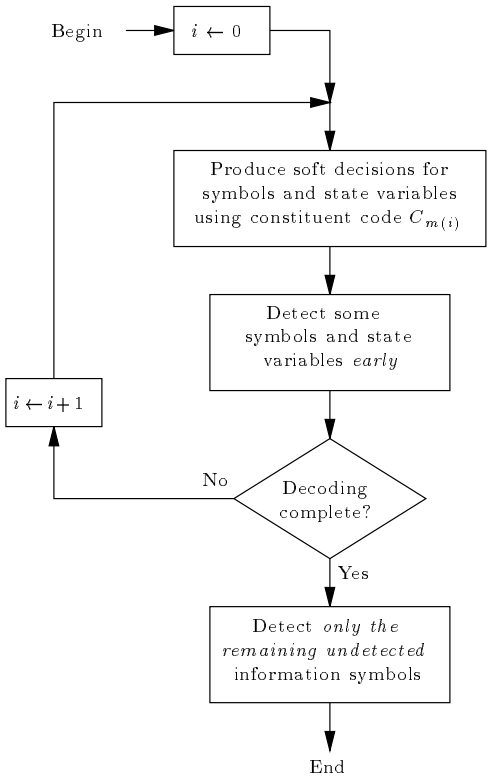

Figure 11: The flowchart for soft iterative decoding with early detection. At the end of each soft decoding cycle, some of the information symbols, state variables and codeword symbols are detected so that the computational demands of later cycles are reduced.

able is known, the associated vertex in the MRF can be removed from the code graph, simplifying further probability propagation. So, overall computational complexity can be reduced by correctly detecting information symbols, state variables and codeword symbols early on in the iterative process. Of course, it is not possible to identify with certainty a large fraction of these variables that can be correctly detected. The hope is that if the number of erroneous detections can be kept extremely low, an acceptable final information symbol error rate can be obtained with reduced computational complexity. We call any such scheme *early detection*. Fig. 11 shows a flowchart for this method. After each constituent code is processed, some information symbols, state variables and codeword symbols are detected early according to an early detection criterion. In general, the criterion for early detection of random variable X_v at cycle i depends on all previously computed log-odds values, $\tilde{L}_{v'}^{i'}, \forall v' \in V, \forall i' < i$. Note that although the flowchart in Fig. 11 allows for early detection after each constituent code is processed, some criteria may restrict early detection to occur at the end of each iteration (*ie.*, when $i \equiv 0 \bmod J$).

Since early detection will inevitably lead to some errors in the early-detected random variables, the early detection criterion must be designed with a desired information symbol error rate in mind. Obviously, the criterion should not give rise to an early-detected *information* symbol error rate that is *higher* than the desired information symbol error rate. More subtle, however, is the issue of what consequences an erroneously early-detected variable may have. It is conceivable that erroneous early-detection in the first few cycles can lead to a catastrophic mode of failure. Whereas it may be possible for the soft iterative decoding algorithm to recover from a log-odds value that has the wrong sign, once the corresponding random variable is detected and its corresponding vertex removed from the MRF, there is no possibility of recovery. However, we have found that for practical purposes this effect (if it exists at all) is not prohibitive.

3.1 Thresholded Early Detection

The simplest criterion for early detection is to compare the reliability value of each information symbol, state variable, and codeword symbol with a static threshold. We prespecify a threshold θ_v for each random variable X_v. (Often, it makes sense to use one threshold for all information symbols, a second threshold for all state variables, and a third threshold for all codeword symbols.) If at cycle i, $\tilde{R}_v^i \geq \theta_v$, then we detect random variable X_v early and set its value to $\hat{x}_v = \tilde{x}_v^i$. We refer to this method of early detection as *thresholded early detection*. Setting the thresholds to a very large positive value effectively causes the algorithm to behave like standard soft iterative decoding. On the other hand, setting the thresholds to zero corresponds either to detection based on a single constituent code or to a single soft iterative decoding iteration, depending on whether the threshold is applied at each cycle or only after each complete iteration.

For the turbo-code system described above, Fig. 12 shows an unbiased sample of 10 cases for which the approximate information bit log-odds drop *below* -10.0 during decoding. Five of these cases are degenerate in the sense that the log-odds converge to values with the wrong sign. However, in the other five cases, after dropping below -10.0, the log-odds rise above 0.0 and even eventually above 10.0. If a threshold of $\theta = 10.0$ were used, all of these bits would be erroneously early-detected within the first 6 cycles. We are currently exploring early detection criteria that avoid early detection of the nondegenerate cases.

An obvious way to minimize the chance of erroneous early detection is to use very large thresholds. However, if the threshold is set too high, the fraction of variables that are early-detected will be negligible. In this case, the computational complexity of further iterations will not be significantly reduced. Fig. 13 shows the fraction of information bits *not* early-detected as a function of threshold, in each of the first 10 decoding cycles, for the turbo-decoding system described above. For a given threshold,

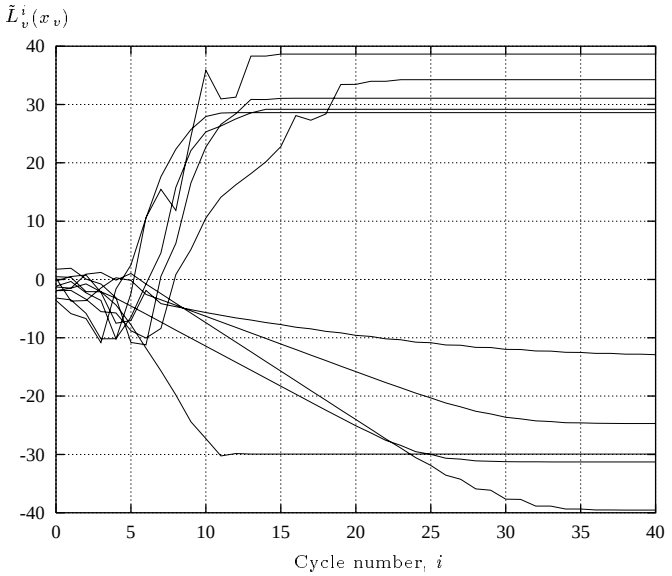

Figure 12: A plot of log-odds versus decoding cycle number, for the correct value of each information bit in an unbiased sample of 10 cases where the log-odds drop below -10.0 during decoding (the code is described in Section 3).

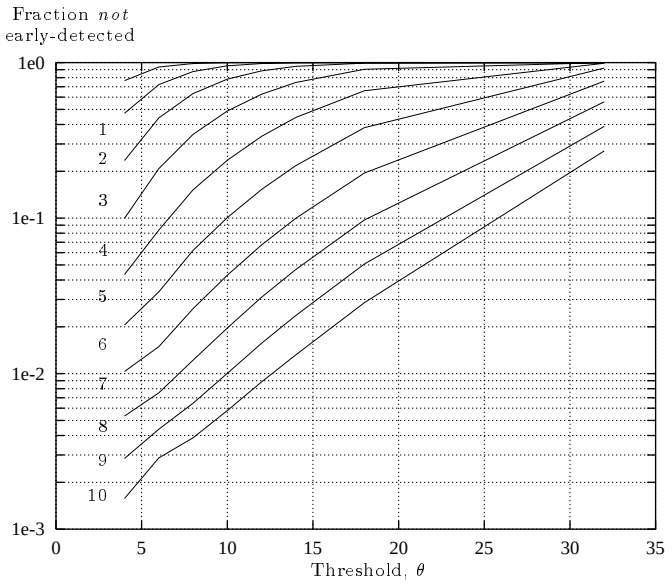

Figure 13: The fraction of information bits *not* early-detected in each of the first 10 decoding cycles as a function of threshold (the code is described in Section 3).

as decoding proceeds, more bits are early-detected. However, for higher thresholds very few bits are early-detected. Evidently, a tradeoff must be made between complexity reduction and the final BER.

3.2 Termination Criterion

A termination criterion is used to determine when to interrupt the iterative procedure and detect the remaining information symbols. Berrou *et al.* [1] simply execute a predetermined number of cycles. In their case, this criterion implies that a fixed number of computations are performed for each information symbol sequence. When early detection is used, however, the number of computations per cycle varies from sequence to sequence and also during decoding. In this case it makes sense to simply terminate the procedure once a predefined number of computations have taken place.

We are currently exploring a more sophisticated termination criterion that depends on the log-likelihood of the observed channel output, given the current estimate of the information symbol sequence. Because of the asymptotic equipartition property [34], for reasonably long sequences this log-likelihood gives an accurate measure of the quality of the current information symbol sequence estimate. The iterative procedure can be terminated when this log-likelihood rises above a threshold, or when a prespecified computation limit is exceeded.

4 Early Detection in MRFs

**** Put parts of this paragraph later **** In this section, we show how in general, the constituent MRFs are simplified when an information symbol, a state variable, or a codeword symbol is early-detected. That is, after early-detection, fewer computations are needed to propagate probabilities across each constituent MRF. Also, we show how different types of vertex lead to different reductions in MRF complexity for a given compound MRF. For example, in Berrou *et. al.*'s turbo-codes [1], early-detection of an information symbol gives the greatest amount of simplification whereas early-detection of a codeword symbol gives the least amount of simplification. In contrast, for Gallager's low density parity check codes [5], there is no essential distinction between information symbols and parity check symbols, so early detection of a symbol of either type leads to the same amount of graph simplification.

If a random variable X_v is early-detected, then the subsequent probability messages that vertex v sends to its parent and children are fixed. So, we need no longer compute (5), (6), (9), and (8) for vertex v. An example is shown in Fig. 14a. However, the above computations correspond to an assumption that the MRF is singly-connected. In fact, for multiply-connected MRFs, early-detection can save a great deal more computation. Fig. 14b shows how early-detecting an internal vertex for this MRF converts it from a multiply-connected MRF to a singly-connected MRF, for which probability propagation is exact. Fig. 14c shows that early-detecting an internal vertex can also lead to a limited reduction in computational complexity.

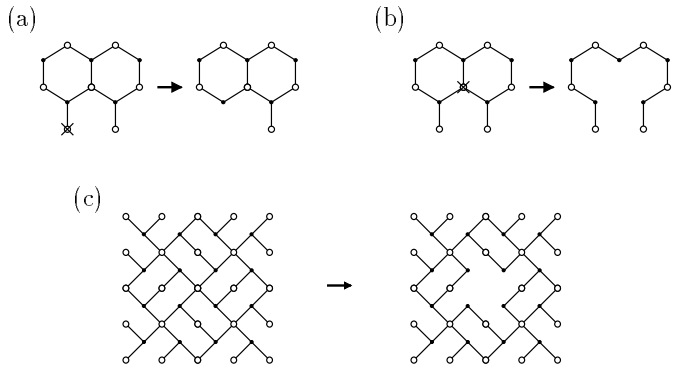

Figure 14: (a) Early-detection of a leaf vertex leads to little simplification of the multiply-connected MRF. (b) Early-detection of an internal vertex in this case transforms the system from a multiply-connected MRF to a signly-connected MRF. (c) Early-detection of an internal vertex leads to moderate simplification.

4.1 MRF Splicing

In the end, when decoding we are interested in the distributions over information symbols, given the channel output. We are not directly interested in the distributions over the state variables. In many cases, it is possible to "splice out" a state variable.

**** Show how state variables can sometimes be "spliced out" ****

5 Early Detection for Turbo-Codes: Trellis Splicing

In this section, we illustrate how early detection applied to turbo-codes can be used to reduce the overall computational complexity of decoding. For turbo-codes, each constituent MRF is a trellis that is processed using a special case of the probability propagation algorithm, called the forward-backward algorithm [27, 28] (see Section 2). This algorithm computes the *a posteriori* information bit probabilities given the channel output, using *a priori* information bit probabilities. The forward-backward algorithm can be viewed simply as a combination of probabilistic "flows" [24] computed in the forward direction and in the backward direction.

Consider the simple two-state trellis shown in Fig. 15a, that corresponds, say, to constituent code C_j. Instead of labelling the vertices "v", we will let X_k be the random variable for the information bit in the kth section of the trellis, and S_k be the random variable for the state at the beginning of the kth section of the trellis. The edge in the kth section of the trellis that leaves state $s_k \in \{0,1\}$ in response to information bit $x_k \in \{0,1\}$ has an associated branch metric, $\gamma_k^{x_k}(s_k)$. These metrics are determined from the received signals and the *a priori* probabilities regarding the transmitted information bit values. (For example, in a systematic code, the likelihoods for the noisy received information bits can be included in the *a priori* probabilities.) If $P(Y_{jk} = y_{jk}|X_k = x_k, S_k = s_k)$ is the likelihood function for the kth received signal of constituent code C_j, and $P(X_k = x_k)$ is the *a priori* probability for information bit x_k, then

$$\begin{aligned}\gamma_k^{x_k}(s_k) &= P(X_k = x_k) \\ &\quad \cdot P(Y_{jk} = y_{jk}|X_k = x_k, S_k = s_k).\end{aligned}$$

The forward pass consists of computing the flows from these metrics in the forward direction, determining in the end a flow value $\alpha_k(s_k)$ for each state s_k at each section $k, k = 0 \ldots K$:

$$\begin{aligned}\alpha_{k+1}(0) &= \gamma_k^0(0)\alpha_k(0) + \gamma_k^1(1)\alpha_k(1), \\ \alpha_{k+1}(1) &= \gamma_k^1(0)\alpha_k(0) + \gamma_k^0(1)\alpha_k(1).\end{aligned}$$

The flow values at the beginning of the trellis, $\alpha_0(s_0)$ are set according to *a priori* knowledge. For example, if we know (as we often do) that the encoder initialized the trellis in state $s_0 = 0$, we set $\alpha_0(0) = 1$ and $\alpha_0(s_0) = 0$ for $s_0 \neq 0$. The backward pass simply consists of a flow computation in the reverse direction in order to obtain a flow value $\beta_k(s_k)$ for each state at each section:

$$\begin{aligned}\beta_k(0) &= \gamma_k^0(0)\beta_{k+1}(0) + \gamma_k^1(0)\beta_{k+1}(1), \\ \beta_k(1) &= \gamma_k^1(1)\beta_{k+1}(0) + \gamma_k^0(1)\beta_{k+1}(1).\end{aligned}$$

Again, if we know that the encoder used trellis termination so that the final state is $s_K = 0$, we set $\beta_K(0) = 1$ and $\beta_K(s_K) = 0$ for $s_K \neq 0$. The flows are combined to obtain the *a posteriori* odds that each information bit is 1 versus 0, given *only* the received signal sequence for constituent code C_j:

$$\frac{P(X_k = 1|\mathbf{y}_j)}{P(X_k = 0|\mathbf{y}_j)} =$$

$$\frac{\alpha_k(0)\gamma_k^1(0)\beta_{k+1}(1) + \alpha_k(1)\gamma_k^1(1)\beta_{k+1}(0)}{\alpha_k(0)\gamma_k^0(0)\beta_{k+1}(0) + \alpha_k(1)\gamma_k^0(1)\beta_{k+1}(1)},$$

where $\mathbf{y}_j$ is the received signal sequence for constituent code C_j.

The computational cost of each section in the forward-backward algorithm thus consists of the time spent computing the αs and βs for each state, as well as the time spent computing the *a posteriori* odds. Although there

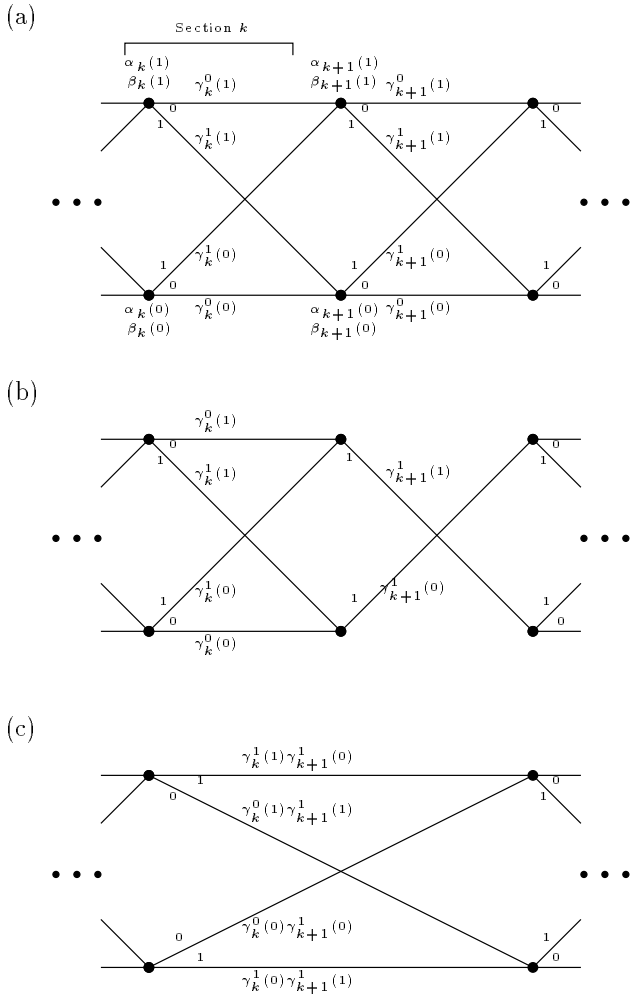

Figure 15: Trellis splicing. (a) shows a two-state trellis with edges accompanied by information bit labels and metrics and with nodes accompanied by flows. (b) and (c): If we know that information bit $k+1$ has a value of 1, we can cut the corresponding section out of the trellis and splice the trellis back together, introducing new information bit labels and new metrics for the connecting edges.

are various useful techniques and approximations for decreasing this cost [13, 15], we will define it as our basic computational unit, and refer to it as a trellis *section operation*.

Suppose that according to some early detection criterion, we decide that the value of information bit X_{k+1} is 1. (Here, we will consider early-detection for information symbols only.) As a consequence, the trellis simplifies to the one shown in Fig. 15b. The trellis can be simplified further by multiplying out the path metrics, giving the trellis shown in Fig. 15c. Note that not only have the path metrics changed, but also the transitions now correspond to different information bit values. In general, portions of the trellis corresponding to early-detected information bits can be cut away, and the remaining segments spliced together with new path metrics and new information bit edge labels. If the values of b information bits are known, the spliced trellis will be b sections shorter, leading to a computational savings of b section operations for each future forward-backward sweep.

In order to implement trellis splicing, an integer array must be used to determine the state transitions, $(s_k, x_k) \longrightarrow s_{k+1}$. Whereas in the original trellis this mapping is very regular, after trellis splicing it is usually not. For example, the information bits associated with the outgoing edges of the kth state in Fig. 15c have *opposite* values compared to those in Fig. 15a. The use of this array slightly increases the computational complexity of each section operation. Also, the array must be modified each time a section is cut away. However, both of these computational costs are insignificant compared to the cost of the basic section operation. In the implementation of trellis splicing used for the experiments presented in Section 6, we found that the percentage of cpu time spent on trellis splicing was less than 6%. The integer array also requires extra memory. However, the total memory requirement actually *decreases* while using trellis splicing. When a single section is cut away, the memory liberated by the elimination of γs, αs and βs more than makes up for the extra integer array memory introduced. Moreover, if sections adjacent to the first are cut away, the transition array is simply modified, so that the memory associated with the γs, αs and βs of the adjacent sections is completely recovered.

6 Preliminary Results

We have simulated trellis splicing results for the turbo-decoding system described in Section 3, using thresholded early detection applied at the end of each cycle. Fig. 16 shows plots of BER versus number of section operations per information bit decoded, for thresholds of 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0 and 18.0. The curve for turbo-decoding *without* trellis splicing is also shown. The iterative termination criterion was that a specific number of cycles were performed for each block. The resulting number of errors and number of section operations were then averaged over block transmissions. In order to average out the effects of catastrophic *block* failures (*ie.*, failure modes where a large fraction of the information block is incorrectly decoded), we simulated the transmission of 200 000 information blocks for each threshold.

As expected, for a given threshold, early detection leads to a BER floor as a result of erroneously early-detected information bits. This effect highlights the trade-off that is being made. A desired BER must be specified beforehand. A threshold is then chosen to maximize the fraction of early-detected bits while maintaining the overall

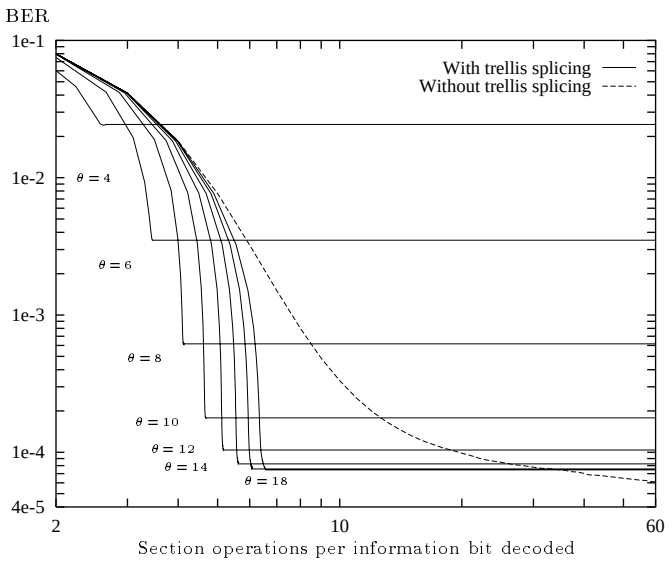

Figure 16: BER performance turbo-decoding with and without thresholded early detection (the code is described in Section 3). In order to average out the effects of block failures, we simulated the transmission of 200 000 information blocks.

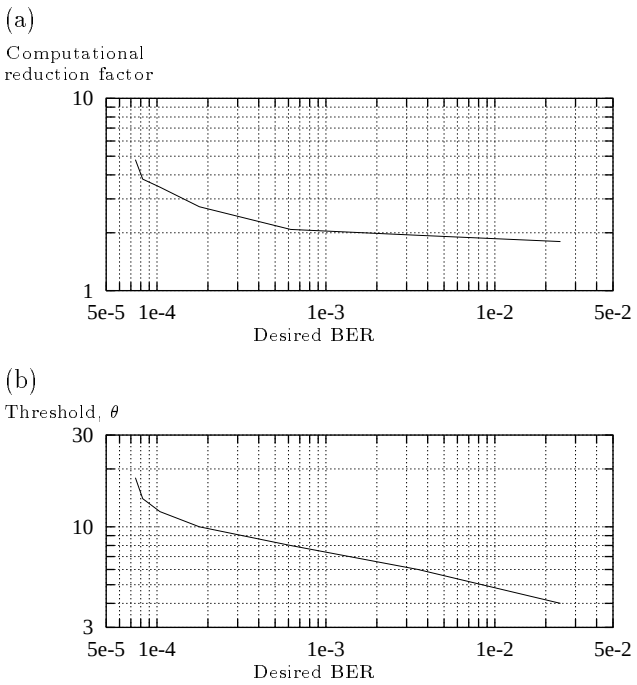

Figure 17: (a) Achievable computational reduction versus BER for the code described in Section 3. (b) Threshold values that give the reductions plotted in (a).

BER at this level. For a prespecified BER, the computational complexity of decoding can be reduced the most by using the threshold that corresponds to the curve in Fig. 16 that bottoms out at the prespecified BER. Thus, the locus of points corresponding to the knees of the curves gives the achievable BER-complexity performances. The locus of points described above is interpolated in Fig. 17 which shows the computational reduction factor and the threshold versus the designed BER. For a specified BER, the achievable computational reduction can be determined from Fig. 17a, and this reduction can be obtained using the threshold determined from Fig. 17b. This performance is subject to the specific system under study, the method of early detection (thresholded), and also the termination criteria used to stop the iterative procedure. We expect that further experiments will lead to higher computational reduction factors.

7 Conclusions and Further Research

In this paper, we have introduced a method called *early detection* that can be used in conjunction with soft iterative decoding. In the new method, information symbols are detected early on in decoding in order to reduce the computational complexity of later processing. For the case of convolutional constituent codes, early detection allows trellis sections to be removed. The resulting *spliced* trellis is shorter and so can be processed more quickly during later processing. As an instance of early detection, we presented *thresholded early detection* and simulated results for this method applied to soft iterative decoding with convolutional constituent codes (turbo-codes). The new method leads to a reduction in computational complexity of over a factor of four in a specific operating regime. Early detection may be applied using a wide variety of constituent codes. The method is not much more difficult to implement than standard soft iterative decoding, but leads to an overall decrease in the computational complexity of decoding.

We believe that greater reductions in computational complexity can be achieved than those presented in Section 6. Currently, we are exploring different regimes (*eg.*, SNR, block length, encoder polynomials, type of constituent code, $m()$ map, *etc.*) in order to determine under what conditions early detection gives optimum performance. Specifically, the early detection criterion and the iterative termination criterion stand to be improved. For example, it may be possible to assign a *risk factor* to each information symbol (*eg.*, based on the length of the minimum code graph cycle of which the information symbol is part of). Each information symbol can have a different threshold that is determined from the risk factor. Also, there may be a useful way to adapt the threshold *during* decoding. The results in Section 6 were obtained using an iterative termination criterion that simply required that a specific number of cycles be performed for each block.

It seems that a more sophisticated criterion based on the log-likelihood of the received signals might give better performance.

The early detection scheme also provides insight into how to isolate the information symbols that are leading the soft iterative decoding procedure astray. If for a given block there are relatively few of these symbols, it may be possible to do *exact* decoding for the troublesome symbols.

We are also investigating session-level issues, such as how the early detection and iterative termination criteria can be set for a particular session. For example, some type of preamble sent at the beginning of a session may be used to determine the criteria. It may be useful to adjust the criteria *during* a session.

Finally, we are exploring the general issue of how to make statistically significant conclusions. We have noted earlier in this paper the possible presence of catastrophic modes of failure (this is a general problem, not one that is just relevant to early detection). In such a mode of failure, a significant fraction (say 10%) of the information symbols are in error. To date, we have dealt with this by simulating the transmission of a very large number of blocks. However, it may be possible to use a more sophisticated error model. Alternatively, paired statistical tests may be useful for comparing two or more results.

8 Acknowledgement

We thank Glenn Gulak, David MacKay and Radford Neal for helpful comments on an earlier draft of this paper. Brendan Frey also thanks his PhD. supervisor, Geoffrey Hinton, for tolerating this detour into another state.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *Proceedings of the IEEE International Conference on Communications*, 1993.

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

[3] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *European Transactions on Telecommunications*, vol. 6, pp. 513–525, 1995.

[4] 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.

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

[6] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, pp. 533–547, 1981.

[7] G. Battail, C. Berrou, and A. Glavieux, "Pseudo-random recursive convolutional coding for near-capacity performance," in *Proceedings of GLOBECOM'93*, 1993.

[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. D. Anderson, "The TURBO-coding scheme," in *Proceedings of ISIT'94*, 1994.

[10] J. E. M. Nillson and R. Kötter, "Iterative decoding of product code constructions," in *Proceedings of ISITA'94*, 1994.

[11] P. Jung and M. Nasshan, "Dependence of the error performance of turbo-codes on the interleaver structure in short frame transmission systems," *Electronics Letters*, vol. 30, pp. 287–288, 1994.

[12] D. Divsalar and F. Pollara, "Turbo-codes for PCS applications," in *Proceedings of ICC'95*, pp. 54–59, 1995.

[13] 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.

[14] S. Benedetto and G. Montorsi, "Unveiling turbo-codes: Some results on parallel concatenated coding schemes," *IEEE Transactions on Information Theory*, vol. 42, pp. 409–428, 1996.

[15] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-ouput decoding algorithms in iterative decoding of parallel concatenated convolutional codes." Submitted to *IEEE International Conference on Communications*, 1996.

[16] R. Kinderman and J. L. Snell, *Markov Random Fields and Their Applications*. Providence USA.: American Mathematical Society, 1980.

[17] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*. San Mateo CA.: Morgan Kaufmann, 1988.

[18] N. Wiberg, *Codes and Decoding on General Graphs*. Linköping Sweden: Department of Electrical Engineering, Linköping University, 1996. Doctoral dissertation.

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

[20] N. Alon, J. Bruck, J. Naor, M. Naor, and R. M. Roth, "Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs," *IEEE Transactions on Information Theory*, March 1992.

[21] M. Sipser and D. A. Spielman, "Expander codes." To appear in *IEEE Transactions on Information Theory*, 1996.

[22] D. A. Spielman, "Linear-time encodable and decodable error-correcting codes." To appear in *IEEE Transactions on Information Theory*, 1996.

[23] F. R. Kschischang and V. Sorokine, "On the trellis structure of block codes," *IEEE Transactions on Information Theory*, vol. 41, pp. 1924–1937, 1995.

[24] R. J. McEliece, "On the BJCR trellis for linear block codes," *IEEE Transactions on Information Theory*, vol. 42, 1996.

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

[26] R. E. Neapolitan, *Probabilistic Reasoning in Expert Systems*. New York NY.: John Wiley & Sons, 1990.

[27] 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.

[28] 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.

[29] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *European Transactions on Telecommunications*, vol. 6, pp. 513–525, 1995.

[30] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721–741, 1984.

[31] G. G. Potamianos and J. K. Goutsias, "Partition function estimation of Gibbs random field images using Monte Carlo simulations," *IEEE Transactions on Information Theory*, vol. 39, pp. 1322–1332, 1993.

[32] D. Chandler, *Introduction to Modern Statistical Mechanics*. New York NY.: Oxford University Press, 1987.

[33] J. Zhang, "The mean field theory in EM procedures for blind Markov random field image restoration," *IEEE Transactions on Image Processing*, vol. 2, pp. 27–40, 1993.

[34] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York NY.: John Wiley & Sons, 1991.