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

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

Abstract—**The excellent bit error rate performance of new iterative decoding algorithms (*e.g.*, turbodecoding) is achieved at the expense of a computationally burdensome decoding procedure. In this paper, we present a method called *early-detection* that can be used to reduce the computational complexity of a variety of iterative decoders. Using a confidence criterion, some information symbols, state variables and codeword symbols are detected early on during decoding. In this way, the computational complexity of further processing is reduced with a controllable increase in BER. We present an easily implemented instance of this algorithm, called *trellis splicing*, that can be used with turbodecoding. For a simulated system of this type, we obtain a reduction in computational complexity of up to a factor of four, relative to a turbodecoder that obtains the same increase in BER by performing fewer iterations.**

I. Introduction

EXPLODING INTEREST in the theory and application of iterative decoding algorithms was ignited by the impressive bit error rate performance of the *turbodecoding* algorithm [1–16]. In this paper, we present a simple method for reducing the computational complexity of such iterative decoders. The new method stems from our observation that the values of certain information symbols, state variables and codeword symbols often can be ascertained reliably early on in the decoding process. By identifying such candidates and detecting them early (*i.e.*, clamping them early on in the decoding process) we reduce the computational complexity of subsequent decoding iterations. We refer to this general method as *early-detection*.

One way to view early-detection is as a refinement of a block-oriented stopping criterion used to terminate the iterative process in iterative decoders. For example, Hagenauer *et al.* [9] proposed monitoring the relative entropy between the set of information bit reliabilities for the current iteration and the previous iteration. When the change in this relative entropy falls below a threshold, the iterative decoding process is terminated. The basic idea is that iterative decoding should stop when the decoder's reliability values are stable. Block-oriented stopping criteria lead to iterative decoders that are more efficient than fixed-complexity iterative decoders, since the stopping criteria effectively allows the decoder to spend more iterations on "tough" blocks, and fewer iterations on "easy" blocks.

Taking this reasoning one step further, we believe that in some cases, *parts of the codeword* may be more easily decoded than other parts. Although different parts of a codeword are usually inter-dependent, for particular noise patterns the coupling between parts may be weak. In these cases, it makes sense that the decoder should spend more computations on "tough" parts, and fewer computations on "easy" parts. During decoding, those parts that are deemed to be successfully decoded are clamped. Decoding computations are then focussed on the remaining parts.

For example, Fig. 1 shows how the reliabilities for a randomly selected subset of information bits *within the same block* evolve during iterative decoding of a turbocode. A large positive value of the decoder log-probability ratio indicates that the decoder is quite confident of the value of the information bit, and that this value is correct. A large negative value (none shown) of the log-probability ratio indicates that the decoder is quite confident of the value of the information bit, and that this value is *wrong*. These curves were produced by simulating the transmission of a single codeword over an $E_b/N_0 = 0.2$ dB AWGN channel, using a binary rate 1/3 unpunctured turbocode with 10,000 information bits, identical constituent encoders $(21, 37)_{\mathrm{octal}}$, and a randomly drawn permuter. As evidenced by the horizontal spread of these curves, the decoder is correctly confident of many information bits before it has sorted out the values of other information bits. By detecting some of the well-determined bits early, computations can be refocussed on decoding the less well-determined bits.

The notion of revisiting a decoding operation after "pinning" some of the variables has been used before to *improve BER performance*. The most common application is for decoding the serial concatenation of a Reed-Solomon outer code with a convolutional inner code. For practical purposes, the Reed-Solomon decoder either outputs an error-free codeword segment or flags the segment as a decoding failure. After the convolutional code has been decoded and its output decoded by the Reed-Solomon decoder, the codeword segments that are practically known to be error-free can be fed back to the convolutional decoder and used to pin certain trellis states for a second round of improved decoding. By using this approach, substantial coding gains have been reported by Lee [17], Collins [18], and Hagenauer *et al.* [19].

Our application of "pinning", which we call "early-detection", is meant to *decrease decoding complexity*, but not improve BER performance or improve coding gain. For example, turbocodes do not have component decoders with large minimum distance, so there is no way to be practically certain that an early-detected variable is correct. When applied to some types of iterative decoders such as turbodecoders, early-detection actually worsens the BER performance. However, we

Manuscript received September 15, 1996; revised May 2, 1997.

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

Publisher Item Identifier S 0733-8716(98) 00162-0.

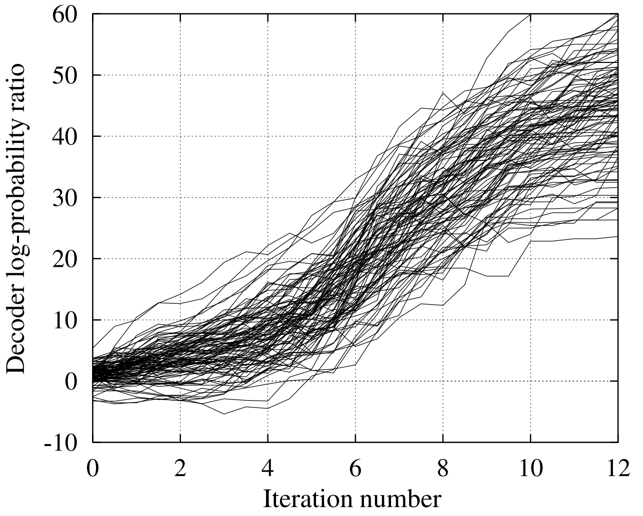


Fig. 1. A plot of the log-probability ratio versus iteration number, for the correct value of each information bit in a randomly selected set of 100 bits *within the same block* of 10,000 bits.

are interested in using early-detection to reduce the complexity of iterative decoders in a way that leads to a smaller increase in BER compared to other techniques, such as performing fewer decoding iterations. Our method can be applied in conjunction with a variety of iterative decoding algorithms, including the decoders for turbocodes [14–16], serially-concatenated convolutional codes [14, 15], product codes [14, 15], and Gallager's low-density parity-check codes [11, 14].

In Sec. II we trace the soft decision reliabilities of many information bits during turbodecoding. Based on these simulations, we speculate that identifying candidates for early-detection through the use of a log-probability ratio threshold is more efficient than higher order methods, such as monitoring the change in the log-probability ratios.

In Sec. III, we describe the computational consequences for subsequent decoding in a variety of iterative decoders, when an information symbol, codeword symbol or state variable is detected early. By viewing iterative decoding as message passing in the "Bayesian network" that describes a code [12, 14–16, 20], we are able to obtain an approximate general formula for the computational complexity of a single iteration of decoding.

In Secs. IV and V we describe the *trellis splicing* algorithm and present results for thresholded early-detection of information symbols in a turbocode system. Here, we find that for a specified BER there is an optimal threshold that will lead to the greatest reduction in computational complexity, relative to simply performing fewer iterations of decoding without early-detection.

II. EARLY-DETECTION CRITERIA

As discussed in the next section, the computational complexity of an iteration decreases with the number of early-detected variables. So, in order to obtain the greatest reduction in computational complexity, the decoder should early-detect many variables as possible. However, an overly aggressive early-detection criterion will lead to a high rate of *erroneous* decisions, spoiling the BER performance. In addition to this constraint, the early-detection criterion should be relatively simple, so that the overhead of ascertaining which variables ought to be early-detected does not overshadow the reduction in the computational complexity of subsequent iterative decoding. In this section, we explore criteria that use the soft decision reliabilities in order to ascertain whether or not an early-detection should occur.

The soft decisions used for iterative decoding can be represented as log-probability ratios that approximate the true *a posteriori* log-probability ratios. The log-probability ratio for an information symbol, state variable, or codeword symbol Z at iteration i given the channel output $\mathbf{y}$ is

$$\hat{L}_Z^i(z) = \log \frac{\hat{P}^i(Z = z|\mathbf{Y} = \mathbf{y})}{\hat{P}^i(Z \neq z|\mathbf{Y} = \mathbf{y})}, \tag{1}$$

where $\hat{P}^i(Z|\mathbf{Y} = \mathbf{y})$ is the approximation to the *a posteriori* distribution $P(Z|\mathbf{Y} = \mathbf{y})$ produced at iteration i. We will let i be fractional when the meaning is clear. For example, in a turbodecoder with two constituent codes, $i = 0.5$ refers to quantities produced by processing the first constituent code for the first time. Equivalently, $\hat{L}_Z^i(z)$ can be viewed as a decision accompanied by a reliability value:

$$\hat{z}^i = \text{argmax}_z \hat{L}_Z^i(z), \quad \hat{R}_Z^i = |\hat{L}_Z^i(\hat{z}^i)|.$$

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

In order to determine an appropriate early-detection criterion, we simulated the transmission of 100 blocks from a rate 1/3 unpunctured turbocode with 10,000 information bits, identical constituent encoders $(21, 37)_{\text{octal}}$, and a randomly drawn permuter. We used binary signalling over an additive white Gaussian noise (AWGN) channel with $E_b/N_0 = 0.2$ dB. To speed up decoding, our forward-backward algorithm was implemented using a linear interpolation approximation to the function $\log(1 + \exp(\cdot))$. Also, our decoder did not weight the "extrinsic information" by the reliability variances as was originally suggested by Berrou *et al.* [21]. (We found that this weighting operation is not necessary at BER greater than 10^{-6}.) Fig. 2 shows a plot of the log-probability ratio versus iteration number for the correct value of a randomly positioned information bit in each of the 100 blocks. In contrast to Fig. 1, this figure shows the variability in log-probability ratio convergence rate from block to block.

It appears from Fig. 2 that the only simple criterion that a decoder can use without introducing too many early-detection errors is a simple threshold. Higher order criteria, such as the change in $\hat{L}_{U_k}(u_k)$, would produce too many erroneous early-detections. Although the relative entropy from one iteration to the next was successfully used in [9] as a block-oriented termination criterion, the same rule would not work at the more refined symbol-oriented level of early-detection. The decoder remains undecided on some variables for many iterations (up

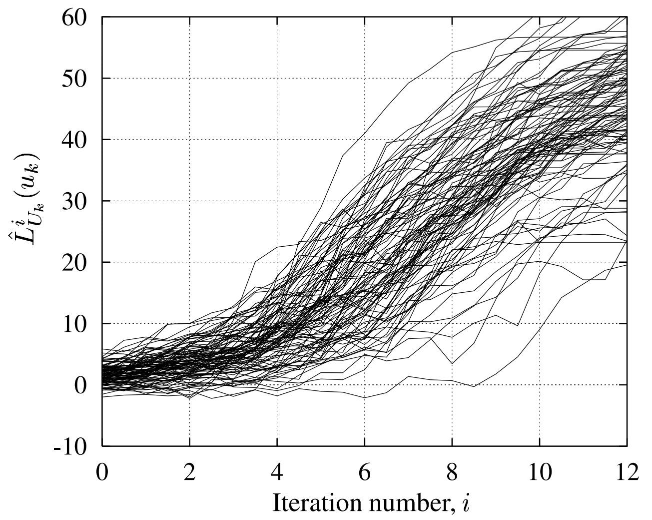


Fig. 2. A plot of the log-probability ratio versus iteration number, for the correct value of a randomly positioned information bit in each of 100 decoded turbocode blocks.

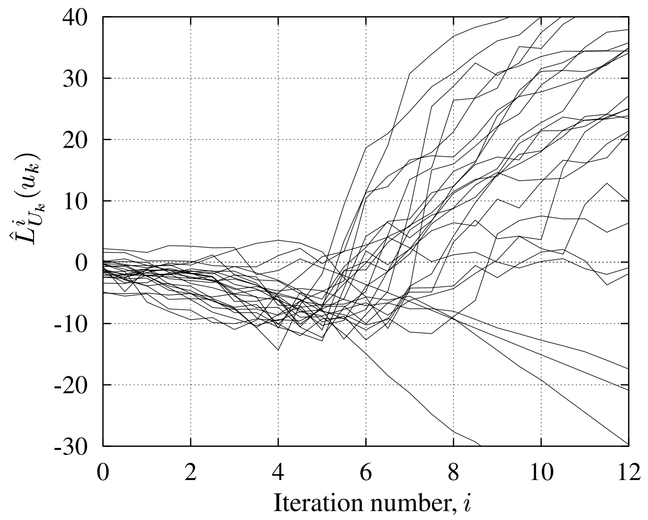


Fig. 3. A plot of the log-probability ratio versus iteration number, for the correct value of each of 25 information bits from 25 randomly selected blocks in which the log-probability ratio dropped below -10.0 during decoding.

to $i = 8.5$ for one curve in Fig. 2), and consequently $\hat{L}_{U_k}(u_k)$ does not change much for those variables. However, eventually the decoder finds a consistent codeword segment and then the log-probability ratios for the related information bits change drastically.

For the turbocode system described above, Fig. 3 shows 25 randomly selected cases (each from a different block) for which the log-probability ratios drop *below* -10.0 during decoding. These traces show that the decoder can become incorrectly confident of the value of an information bit, but then with further iterations become correctly confident. By using a threshold of 15.0 for early-detection, all of the bits that the decoder correctly decodes as $i \to \infty$ can be detected early and correctly (the four curves that are between -5.0 and 10.0 at $i = 12$ eventually rise above 15.0). On the other hand, if the change in the log-probability ratio is used for early-detection, many of the bits that the decoder correctly decodes as $i \to \infty$ would be incorrectly detected early at the values for i where the curves stop falling and begin to rise. That is, the change in $\hat{L}^i_{U_k}(u_k)$ is close to zero at the iteration where the decoder begins to *correct* the bit. Higher order criteria may actually help in this case (by detecting that a curve is turning around), but it appears the data is too noisy for this approach to be successful. Also, higher-order criteria increase the computational overhead of early-detection.

III. Reduction in Complexity Due to Early-Detection

As pointed out by Wiberg [22] and MacKay and Neal [11], the key ideas of iterative decoding using soft decisions were present in Gallager's work in the early 1960's on low-density parity-check codes [23]. Given the channel output, an iterative decoder processes each constituent code (one at a time or in parallel) while using soft decisions made by previous iterations to bias the results. The appropriate way to take into account the soft outputs from previous iterations can be derived to first order using Bayes' rule [9]. In fact, this procedure has been well-established for over a decade as the "probability propagation" algorithm for processing "Bayesian networks" used in the area of artificial intelligence [12,14,20]. For an overview of how graphical models such as Bayesian networks can be used to describe codes and to derive iterative decoders, see the article by Kschischang and Frey in this issue [15]. See [14] for a comprehensive book on this subject. For an in-depth derivation showing that turbodecoding is in fact probability propagation in a Bayesian network, see the article by McEliece, *et al.* in this issue [16]. In this section, we show how early-detection reduces decoding complexity by using an approximate general formula for the computational complexity of a single iteration of decoding. This formula is approximately valid for the iterative decoders for turbocodes, serially-concatenated convolutional codes and product codes. For a given code, this framework indicates which variables ought to be given priority for early-detection, in order to save the most computations.

Using the symbolic and graphical notation introduced in the companion article in this issue [15], the Bayesian networks for a variety of codes are shown in the first column of pictures in Fig. 4. (The channel output variables are not shown — their likelihoods are to be included as "bias" effects on the state variables, codeword bits, and information bits (where applicable) during decoding.) Each vertex in a Bayesian network represents a random variable in a probability model, and the set of vertices that have directed edges connecting to variable Z_i are called the *parents* $\mathbf{A}_i$ of Z_i. The joint distribution over all variables $\mathbf{Z} = (Z_1, \ldots, Z_N)$ in a Bayesian network can be expressed by the following product:

$$P_{\mathbf{Z}}(\mathbf{z}) = \prod_{i=1}^{N} P_{Z_i|\mathbf{A}_i}(z_i|\mathbf{a}_i). \quad (2)$$

Each term $P_{Z_i|\mathbf{A}_i}(z_i|\mathbf{a}_i)$ gives the conditional probability for Z_i given the values of its parents. In the Bayesian network for

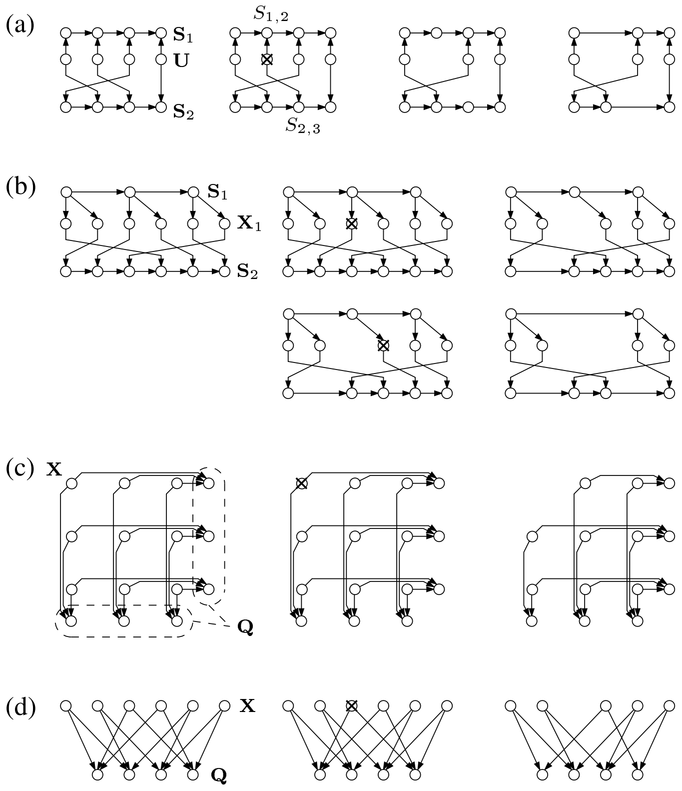


Fig. 4. Early-detection effectively modifies the Bayesian network for a code: (a) a turbocode, (b) a serially-concatenated convolutional code, (c) a product code, (d) a low-density parity-check code.

a code, some of these conditional probabilities represent deterministic relationships. For example, in the Bayesian network for a turbocode,

$$P_{S_{1,i}|S_{1,i-1},U_i}(s_{1,i}|s_{1,i-1},u_i) = \begin{cases} 1 & \text{if state } s_{1,i} \text{ follows } s_{1,i-1} \text{ for input } u_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Other probabilities do not represent deterministic relationships. such as the prior probabilities $P_{U_i}(u_i)$ for the information bits.

Including the channel likelihoods, the joint distribution given by the turbocode Bayesian network shown in the first picture of Fig. 4a is

$$\begin{aligned} P_{\mathbf{U},\mathbf{S},\mathbf{X},\mathbf{Y}}(\mathbf{u},\mathbf{s},\mathbf{x},\mathbf{y}) &= \left[\prod_{i=1}^{4} P_{U_i}(u_i)\right]\left[\prod_{i=1}^{4} p_{Y_{0,i}|U_i}(y_{0,i}|u_i)\right] \\ &\cdot \left[\prod_{i=1}^{4} p_{Y_{1,i}|S_{1,i}}(y_{1,i}|s_{1,i})\right]\left[\prod_{i=1}^{4} p_{Y_{2,i}|S_{2,i}}(y_{2,i}|s_{2,i})\right] \\ &\cdot P_{S_{1,1}|U_1}(s_{1,1}|u_1)\prod_{i=2}^{4} P_{S_{1,i}|S_{1,i-1},U_i}(s_{1,i}|s_{1,i-1},u_i) \\ &\cdot P_{S_{2,1}|U_{I(1)}}(s_{2,1}|u_{I(1)})\prod_{i=2}^{4} P_{S_{2,i}|S_{2,i-1},U_{I(i)}}(s_{2,i}|s_{2,i-1},u_{I(i)}), \end{aligned}$$

where $I(i)$ is the interleaving function (in this case, $I(1) = 3, I(2) = 1, I(3) = 2, I(4) = 4$). The channel outputs for the systematic bits are $Y_{0,i}$.

The *probability propagation* algorithm for estimating $P_{Z_i|\mathbf{O}}(z_i|\mathbf{o})$ for an arbitrary observed subset $\mathbf{O}$ of $\mathbf{Z}$ in a Bayesian network was introduced in [25] and [26]. It turns out that the soft-decision iterative decoders for turbocodes, serially-concatenated convolutional codes, product codes, and low-density parity-check codes can be viewed as the application of this algorithm to Bayesian networks like the ones shown in the first column of pictures in Fig. 4. Let $|P_{Z_i|\mathbf{A}_i}|$ be the number of configurations of a discrete variable z_i and its discrete parents $\mathbf{a}_i$ for which $P_{Z_i|\mathbf{A}_i}(z_i|\mathbf{a}_i) \neq 0$, and let $|\mathbf{A}_i|$ be the number of parents for Z_i. (If Z_i has no parents, let $|\mathbf{A}_i| = 1$.) Then, the computational complexity of each iteration of iterative decoding usually scales as [15]

$$\chi = \sum_{i=1}^{N} |P_{Z_i|\mathbf{A}_i}||\mathbf{A}_i|^2. \tag{4}$$

For example, if the constituent convolutional code for the turbocode described above has memory ν, then $|P_{S_{1,i}|S_{1,i-1},U_i}| = 2^{\nu+1}$ and so the state variable $s_{1,i}$ contributes a complexity of $|P_{S_{1,i}|S_{1,i-1},U_i}| \cdot 2^2 = 4 \cdot 2^{\nu+1}$.

An early-detection can reduce the computational complexity given in (4) both directly and indirectly. The first three pictures in Fig. 4a show how the early-detection of information bit U_2 directly simplifies the Bayesian network, thereby decreasing χ. The modified sum in (4) no longer includes the term $|P_{U_2}| \cdot 1$ $(= 2)$ for U_2, and in each of the terms for the children $S_{1,2}$ and $S_{2,3}$ of U_2, the number of configurations is reduced by a *factor* of 2 and the number of parents is decreased by 1. In the case of the turbocode, the former reduction decreases the complexity contributed by $S_{1,i}$ from $4 \cdot 2^{\nu+1}$ to 2^{ν}.

The indirect effect of detecting U_2 early is shown by the fourth picture in Fig. 4a. Since the objective of the decoder is to make decisions for the information bits, the two states $S_{1,2}$ and $S_{2,3}$ can actually be removed from the network. Suppose $\hat{u}_2$ is the early-detected value of U_2. Then, the new conditional distributions for $S_{1,3}$, after $S_{1,2}$ has been removed, is

$$\begin{aligned} &{P'}_{S_{1,3}|S_{1,1},U_3}(s_{1,3}|s_{1,1},u_3) \\ &\quad = P_{S_{1,3}|S_{1,1},U_3,U_2,Y_{1,2}}(s_{1,3}|s_{1,1},u_3,\hat{u}_2,y_{1,2}) \\ &\quad = N\sum_{s_{1,2}} P_{S_{1,3}|S_{1,2},U_3}(s_{1,3}|s_{1,2},u_3) \\ &\quad \cdot P_{S_{1,2}|S_{1,1},U_2}(s_{1,2}|s_{1,1},\hat{u}_2) \\ &\quad \cdot p_{Y_{1,2}|S_{1,2}}(y_{1,2}|s_{1,2}), \end{aligned} \tag{5}$$

where N is a normalization operator, which ensures that $\sum_{s_{1,3}} P'_{S_{1,3}|S_{1,1},U_3}(s_{1,3}|s_{1,1},u_3) = 1$. Notice that each of the terms in this sum includes a channel likelihood. The computation of these new conditional probabilities is actually performed as a normal part of iterative decoding. so, in practice all that is needed is a small integer lookup table to relate the configurations of $S_{1,1}$ and U_3 to the proper values of $S_{1,3}$.

Figs. 4b to 4d show how the networks for other codes are simplified by detecting variables. In the serially-concatenated convolutional code, detecting information bits (not shown) early leads to relatively little reduction in χ. Instead, the intermediate codeword bits can be early-detected to obtain a significant reduction in the complexity of decoding. Notice that only one trellis is simplified by a single early-detection. Each section of the upper trellis requires that both its outputs be early-detected, as shown by the lower two pictures in Fig. 4b. For the product code and the low-density parity-check code, detecting codeword bits early simplifies the relevant constituent parity check equations.

IV. Early-Detection for Turbocodes: Trellis Splicing

In this section, we illustrate how early-detection applied to turbocodes can be used to reduce the overall decoding complexity. For turbocodes, the Bayesian network consists of two or more chains that are processed using a special case of the probability propagation algorithm, called the forward-backward (a.k.a. "BCJR") algorithm [27, 28]. This algorithm computes the *a posteriori* information bit probabilities using the channel output and *a priori* information bit probabilities. The forward-backward algorithm can be viewed simply as a combination of probabilistic "flows" [29] computed in the forward direction and in the backward direction. Alternatively, a soft-output Viterbi algorithm (SOVA) [9] can be used. Here, we consider early-detection for information symbols only. As discussed earlier, early-detection of a single information symbol reduces the complexity of both constituent codes.

Consider the simple two-state trellis shown in Fig. 5a. Let U_k be the random variable for the information bit in the kth section of the trellis, and let 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 $u_k \in \{0, 1\}$ has an associated branch metric, $\gamma_k^{u_k}(s_k)$. 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 $p_{Y_k|U_k,S_k}(y_k|u_k, s_k)$ is the likelihood function for the kth received signal and $P_{U_k}(u_k)$ is the *a priori* probability for information bit U_k, then $\gamma_k^{u_k}(s_k) = P_{U_k}(u_k)p_{Y_k|U_k,S_k}(y_k|u_k, s_k)$. The forward pass consists of computing the flows from these metrics in the forward direction. This results in a flow value $\alpha_k(s_k)$ for each state s_k at each section k, $k = 0 \ldots K-1$, computed as $\alpha_{k+1}(0) = \gamma_k^0(0)\alpha_k(0) + \gamma_k^1(1)\alpha_k(1)$, and $\alpha_{k+1}(1) = \gamma_k^1(0)\alpha_k(0) + \gamma_k^0(1)\alpha_k(1)$. 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: $\beta_k(0) = \gamma_k^0(0)\beta_{k+1}(0) + \gamma_k^1(0)\beta_{k+1}(1)$, and $\beta_k(1) = \gamma_k^1(1)\beta_{k+1}(0) + \gamma_k^0(1)\beta_{k+1}(1)$. These two types of flow are combined to obtain the *a posteriori* log-probability ratio that each information bit is 1 versus 0, given the received signal sequence $\mathbf{y}$:

$$\log\frac{P_{U_k|\mathbf{Y}}(1|\mathbf{y})}{P_{U_k|\mathbf{Y}}(0|\mathbf{y})} = \log\frac{\alpha_k(0)\gamma_k^1(0)\beta_{k+1}(1) + \alpha_k(1)\gamma_k^1(1)\beta_{k+1}(0)}{\alpha_k(0)\gamma_k^0(0)\beta_{k+1}(0) + \alpha_k(1)\gamma_k^0(1)\beta_{k+1}(1)}. \quad (6)$$

The computational cost of each section in the forward-backward algorithm thus consists of the time spent computing the α's and β's for each state, as well as the time spent computing the *a posteriori* log-probability ratios. Although there are various useful techniques and approximations for decreasing this cost [9, 13], such as the SOVA [9], 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 U_{k+1} is 1. (Here, we will consider early-detection for information bits only.) As a consequence, the trellis simplifies to the one shown in Fig. 5b. The trellis can be simplified further by multiplying out the path metrics, giving the trellis shown in Fig. 5c. 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, $(s_k, u_k) \longrightarrow s_{k+1}$. Whereas in the original trellis this mapping is very regular, after trellis splicing it is usually not. (*Eg.*, the information bits associated with the outgoing edges of the kth state in Fig. 5c have *opposite* values compared to those in Fig. 5a.) 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 V, 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.

V. Results

We have simulated trellis splicing results at $E_b/N_0 = 0.1, 0.2$ and 0.3 dB, for the turbodecoding system described in Sec. II. At the end of each half-iteration of turbodecoding, the log-probability ratio of each information bit was compared with a

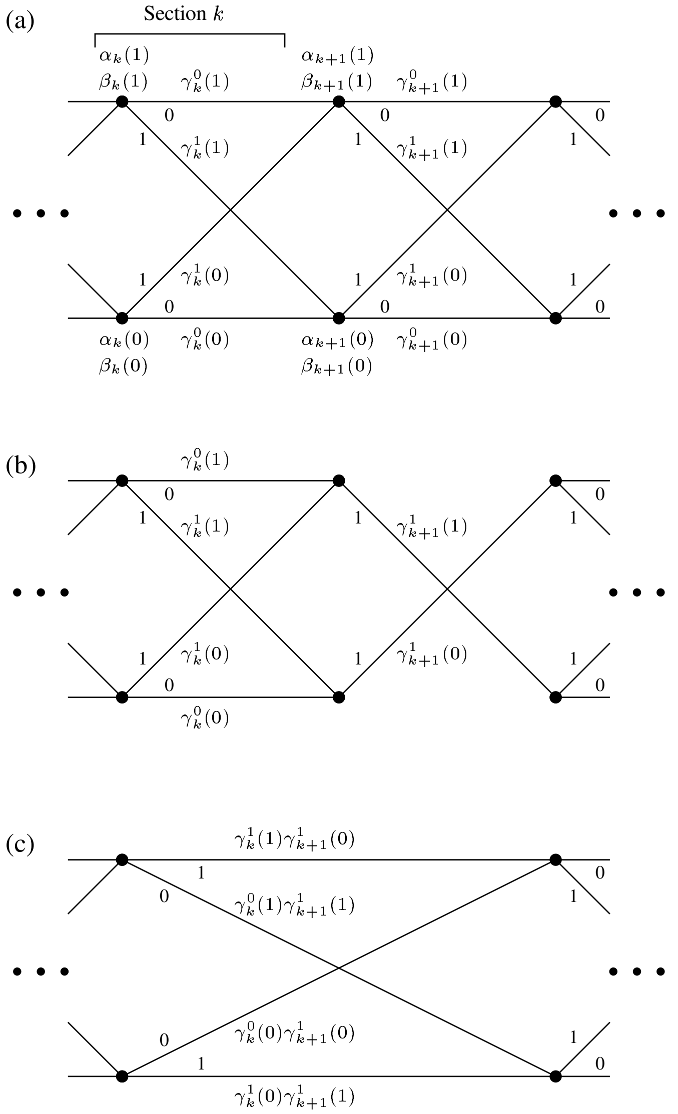


Fig. 5. Trellis splicing. (a) shows a two-state trellis with edges accompanied by information bit labels and metrics and with nodes accompanied by flows. (b) and (c): If we know that information bit $k+1$ has a value of 1, we can cut the corresponding section out of the trellis and splice the trellis back together, introducing new information bit labels and new metrics for the connecting edges.

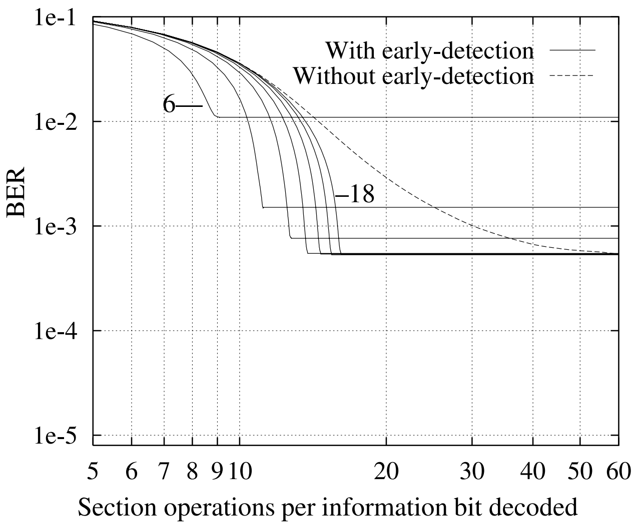


Fig. 6. BER performance of turbodecoding with and without early-detection, for $E_b/N_0 = 0.1$ dB, for thresholds of 6, 8, 10, 12, 14, 16, 18.

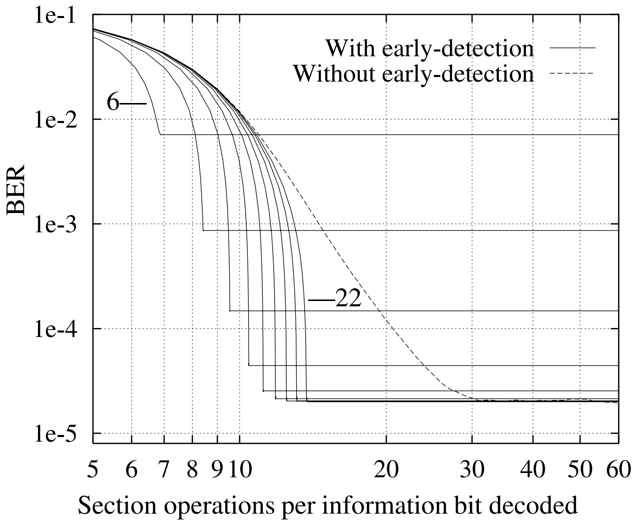


Fig. 7. BER performance of turbodecoding with and without early-detection, for $E_b/N_0 = 0.2$ dB, for thresholds of 6, 8, 10, 12, 14, 16, 18, 20, 22.

threshold in order to decide whether or not the bit should be early-detected. In order to average out the effects of block failure modes (*i.e.*, failure modes where a large fraction of the information block is incorrectly decoded), we simulated the transmission of 20,000 information blocks for each value of the threshold. The resulting number of errors and number of section operations were then averaged over block transmissions. Figs. 6, 7 and 8 show plots of BER versus average number of section operations per information bit decoded, for a variety of thresholds. The curves for turbodecoding *without* early-detection are also shown.

For a given BER, the computational complexity of decoding can be most reduced (compared to standard turbodecoding) by using the threshold that corresponds to the curve in each figure that bottoms out at the prespecified BER. Thus, the locus of points corresponding to the knees of the curves gives the optimal achievable BER-complexity performance. Using these curves, we can answer the question, "At a specified E_b/N_0 and for different BER, what is the computational gain obtained by using early detection compared to using fewer decoding iterations without early detection?" The locus of points described above is interpolated in Fig. 9 which shows the computational gain as a function of BER for the different values of E_b/N_0. For all three values of E_b/N_0, the greatest computational gain is obtained near the minimal BER, and ranges from 2 to 4.

VI. Conclusions and Further Research

In this paper, we have introduced a method called *early-detection* that can be used in conjunction with iterative decoders that use soft decisions. Information symbols, state variables, and codeword symbols are detected early on in decoding in order to

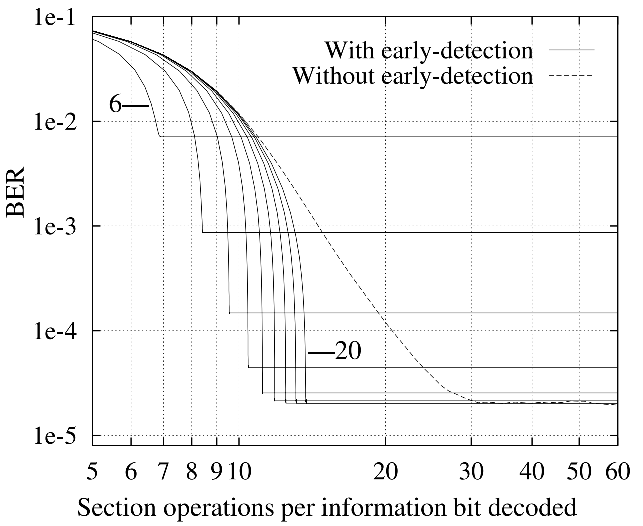


Fig. 8. BER performance of turbodecoding with and without early-detection, for $E_b/N_0 = 0.3$ dB, for thresholds of 6, 8, 10, 12, 14, 16, 18, 20.

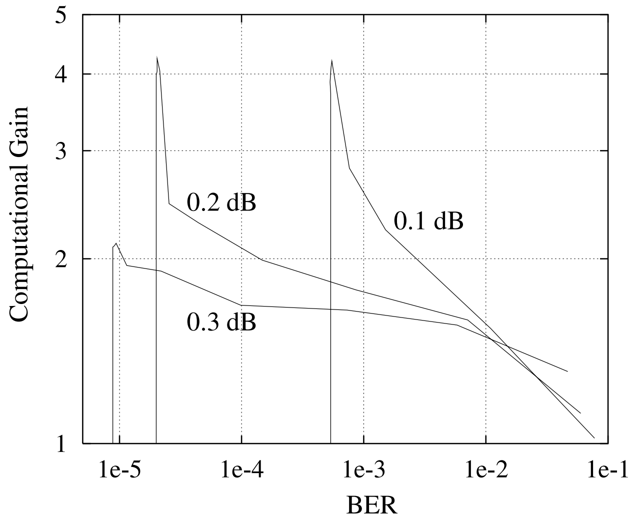


Fig. 9. The computational gain for turbodecoding with early-detection compared to turbodecoding without early-detection as a function of BER, for $E_b/N_0 = 0.1$, 0.2, and 0.3 dB.

reduce the computational complexity of later processing. For the case of turbocodes, early-detection of information symbols allows sections to be removed from all constituent trellises. The resulting *spliced* trellises are shorter and so can be processed more quickly.

We presented results for turbodecoding with early-detection and found that a computational gain of up to a factor of four relative to conventional turbodecoding can be achieved. Early-detection may be applied using a wide variety of constituent codes. The enhanced algorithm is not much more difficult to implement in software 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 Sec. V. Specifically, the early-detection criterion stands to be improved. For example, it may be possible to assign a *risk factor* to each information symbol (*e.g.*, based on the length of the minimum code graph cycle containing the information symbol). 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.

In general, it seems possible to make a wide variety of adjustments to the general graph-based iterative decoding algorithms. For example, see [14] for a "concurrent turbodecoding" method that speeds up decoding by a factor of 80. We feel that investigating the complexity/performance tradeoffs for these algorithms will continue to be a rich and potentially rewarding research area.

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

[3] 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*, 1993, pp. 1740–1745.

[4] J. D. Anderson, "The TURBO-coding scheme," in *Proceedings of IEEE International Symposium on Information Theory*, 1994.

[5] J. E. M. Nillson and R. Kötter, "Iterative decoding of product code constructions," in *Proceedings of the International Symposium on Information Theory and its Applications*, 1994.

[6] 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.

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

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

[9] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, March 1996.

[10] 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, March 1996.

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

[12] B. J. Frey and F. R. Kschischang, "Probability propagation and iterative decoding," in *Proceedings of the 34^{th} Allerton Conference*, 1996, Available at `http://www.cs.utoronto.ca/~frey`.

[13] 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.

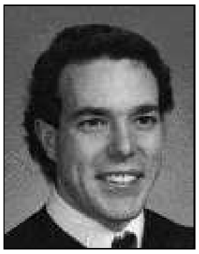
[14] B. J. Frey, *Graphical Models for Machine Learning and Digital Communication*, Cambridge, MA: MIT Press, 1998. See `http://www.cs.utoronto.ca/~frey`.

[15] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, February 1998, available at `http://www.cs.utoronto.ca/~frey`.

[16] R. J. McEliece, D. J. C. MacKay, and J. F. Cheng, "Turbo-decoding as an instance of Pearl's 'belief propagation' algorithm," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, February 1998.

[17] L. Lee, "Concatenated coding systems employing a unit-memory convolutional code and a byte-oriented decoding algorithm," *IEEE Transactions on Communications*, vol. 25, no. 10, pp. 1064–1074, October 1977.

[18] O. M. Collins, "Determinate state convolutional codes," *IEEE Transactions on Communications*, vol. 41, no. 12, pp. 1785–1794, December 1993.
[19] J. Hagenauer, E. Offer, and L. Papke, "Improving the standard coding system for deep space missions," in *Proceedings of IEEE International Conference on Communications*, 1993, pp. 1092–1097.
[20] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, San Mateo CA., 1988.
[21] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Transactions on Communications*, vol. 44, pp. 1261–1271, October 1996.
[22] N. Wiberg, *Codes and Decoding on General Graphs*, Department of Electrical Engineering, Linköping University, Linköping Sweden, 1996, Doctoral dissertation.
[23] R. G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge MA., 1963.
[24] J. F. Cheng, *Iterative Decoding*, Department of Electrical Engineering, California Institute of Technology, Pasadena California, 1997, Doctoral dissertation.
[25] J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artificial Intelligence*, vol. 29, pp. 241–288, 1986.
[26] 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.
[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, March 1974.
[29] R. J. McEliece, "On the BJCR trellis for linear block codes," *IEEE Transactions on Information Theory*, vol. 42, 1996.



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

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