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

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 and **P. Glenn Gulak**
Department of Electrical and Computer Engineering, University of Toronto
10 King's College Road, Toronto, Canada M5S 3G4
frank@comm.utoronto.ca and gulak@eecg.utoronto.ca

August 12, 1996

Abstract

The excellent bit error rate performance of new soft iterative decoding algorithms (cf. 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 are detected early on in the iterative decoding procedure, leading to a reduction in the computational complexity of further processing. We derive an instance of this algorithm that can be applied to turbo-codes. The resulting *trellis splicing* algorithm is easy to implement and leads to a reduction in computational complexity of over a factor of four relative to conventional turbo-decoding in a specific operating regime.

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 more general class of decoding algorithms called *soft iterative decoding*. Soft iterative decoding is an error-correction decoding method that makes use of two or more *constituent codes*. The constituent codes are obtained by projecting the overall code on different (usually overlapping) sets of codeword symbol positions. The decoder operates on the received word in *cycles*. During each cycle, the decoder processes a signal constituent code using the output of the previous cycle to bias the result. The special case of soft iterative decoding introduced by Berrou *et al.* [1] uses two or more convolutional constituent codes that differ only in that each is derived from a different permutation of the information symbols. However, as pointed out by Wiberg *et al.* [2] and MacKay and Neal [3], the key ideas of soft iterative decoding are present in Gallager's 1963 work on low-density parity-check codes [4]. In 1981, Tanner [5] developed Gallager's work further and derived some lower bounds on rate and minimum distance for "recursive" codes. A large body of work that ranges from the very practical to the very theoretical has recently emerged [2, 3, 6–14].

In this paper we make the observation that for soft iterative decoding, very often the final decoding decision affecting a particular information symbol is determined after only a relatively small number of cycles. That is, after only a few cycles, many of the information symbols can be detected with a low rate of error. We present a general method, called *early detection*, for reducing the computational complexity of later cycles by detecting some information symbols early. We examine an instance of early detection, called *thresholded early detection*, that can be used to decide which information symbols are to be detected early. Our basic method can be applied to *any* soft iterative decoding algorithm, but as an example we present computational complexity performance results for the case of convolutional constituent codes (turbo-codes).

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 $P(X_k = x_k|\mathbf{y})$. Many codes are very naturally described using graphical models [2–5, 15–17], and in fact every block code can be represented graphically [18, 19]. 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 (*eg.*, computing probabilities such as $P(X_k = x_k|\mathbf{y})$). For example, soft iterative decoding is fundamentally related to the method of *probability propagation* [20] which is an algorithm used for computing conditional probabilities using graphical models (see textbook references [21,22]).

Gallager [4], Tanner [5] and Pearl [20] have 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. 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 model which is singly connected. An algorithm for computing $P(X_k = x_k|\mathbf{y})$ on such a graphical model was developed by Baum and Petrie [23] as well as by Bahl *et al.* [24] (who cite [23]). 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 the special case of probability propagation applied to a trellis. Furthermore, it turns out that soft iterative decoding is probability propagation applied to graphical models that are *not singly-connected.* Because probability propagation is only approximate in this case, the entire propagation procedure must be iterated until (hopefully) convergence. 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.

Probability propagation between constituent codes works as follows. Given an *a priori* (usually product form) probability distribution over a sequence of information symbols, the received signals for one constituent code are used to obtain a product form approximation to the *a posteriori* probability distribution over the sequence of information symbols[1]. Each component of the product form distribution can be interpreted as the output of a soft decoder which provides an information symbol decision and a reliability value. After a slight modification, this distribution is then used as the *a priori* distribution when processing the next constituent code during the following cycle. Usually, the constituent codes are processed in order, and once all the constituent codes have been processed, the first *iteration* is complete. If the graphical model corresponding to the entire code structure were singly-connected, exact decoding would be finished after one iteration, and the final approximate *a posteriori* distribution would exactly equal the true *a posteriori* distribution. Instead, the cyclic procedure iterates until (hopefully) convergence, at which time information symbols are detected, usually in the fashion of the maximum *a posteriori* (MAP) symbol probability decoding rule.

Fig. 1 shows the flowchart for soft iterative decoding. In general, there are J constituent codes indexed by $j \in \{0, \ldots, J-1\}$. The ith cycle produces a soft decision for each information symbol using constituent code $m(i)$, 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.*,

[1] This distribution is not the true product form *a posteriori* distribution, since the latter must properly take into account *all* constituent codes simultaneously.

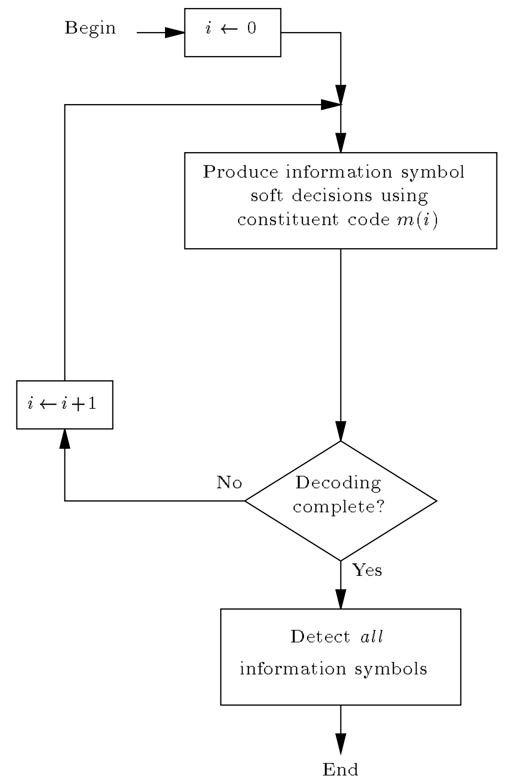

Figure 1: The flowchart for soft iterative decoding using multiple constituent codes. For example, in [1] each constituent code consists of a convolutional code produced using a different permutation of the information symbols.

$m(i) = i \bmod J$.

A soft decision at cycle i for information bit k takes the form of a log-odds value, $\tilde{L}_k^i$, which is an approximation to the posterior log-odds. (For the sake of notational simplicity, we will consider binary information symbols — this is not an essential restriction.) If $\mathbf{X}$ is the random variable representing the information bit sequence and $\mathbf{y} = (\mathbf{y}_0 \ldots \mathbf{y}_{J-1})$ is the total received signal sequence (including each received signal sequence $\mathbf{y}_j$ for constituent code j), then

$$\tilde{L}_k^i = \log \frac{\tilde{P}^i(X_k = 1|\mathbf{y})}{\tilde{P}^i(X_k = 0|\mathbf{y})}, \tag{1}$$

where $\tilde{P}^i(X_k|\mathbf{y})$ is an approximation to the posterior distribution over information bit X_k, produced at cycle i. Equivalently, the log-odds can

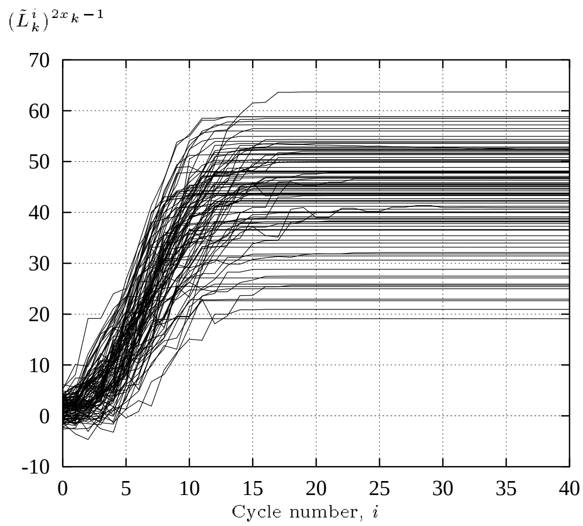

Figure 2: A plot of *corrected* log-odds versus decoding cycle number, for each information bit in an unbiased sample of 100 cases (the code is described in Section 3).

be viewed as an information bit decision accompanied by a reliability value:

$$\tilde{x}_k^i = \begin{cases} 0 & \text{if } \tilde{L}_k^i < 0 \\ 1 & \text{if } \tilde{L}_k^i \geq 0 \end{cases} \qquad \tilde{R}_k^i = |\tilde{L}_k^i|.$$

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

3 Early Detection

Often, after only a relatively small number of cycles, many of the information 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. 2 shows how the approximate *a posteriori* log-odds (1) changes during decoding, for each information bit in an unbiased sample of 100 cases. The log-odds values are *corrected* so that a positive value indicates the bit is being correctly decoded, whereas a negative value indicates the bit is being erroneously decoded. The corrected value of $\tilde{L}_k^i$ is $(\tilde{L}_k^i)^{2x_k-1}$. 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 reduces the computational requirements for making soft decisions for the remaining symbols. In Section 4 we show this for the case of turbo-codes. It is also true for the graphical constituent codes discussed in [2–5]. From a graphical viewpoint, when the value of an information symbol is known, the associated node can be removed from the code graph, simplifying further probability propagation. So, overall computational complexity can be reduced by correctly detecting information symbols early on in the iterative process. Of course, it is not possible to identify with certainty a large fraction of information symbols that can be correctly detected. The hope is that if the number of erroneous detections can be kept extremely low, an acceptable final symbol error rate can be obtained with reduced computational complexity. We call any such scheme *early detection*. Fig. 3 shows a flowchart for this method. After each constituent code is processed, some information symbols are detected early according to an early detection criterion. In general, the criterion for early detection of information bit k at cycle i depends on all previously computed log-odds values, $\tilde{L}_{k'}^{i'}, \forall k', \forall i' < i$. Note that although the flowchart in Fig. 3 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 $i \equiv 0 \bmod J$).

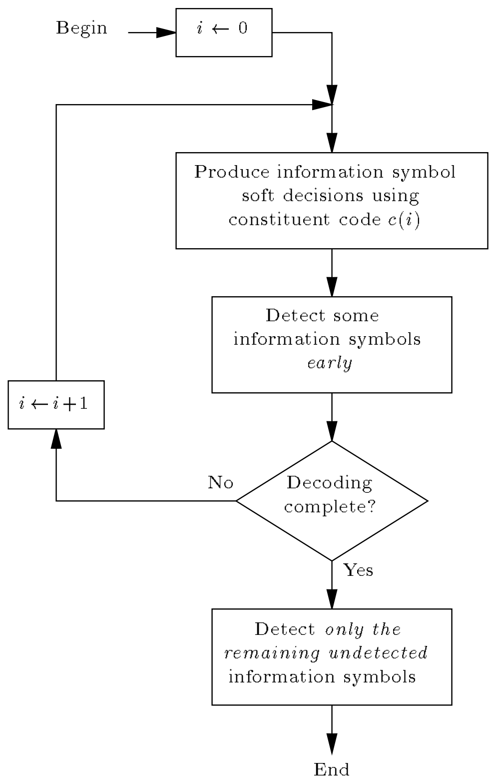

Figure 3: The flowchart for soft iterative decoding with early detection. At the end of each soft decoding cycle, some of the information symbols are detected so that the computational demands of later cycles are reduced.

Since early detection will inevitably lead to some errors in the early-detected symbols, the early detection criterion must be designed with a desired symbol error rate in mind. Obviously, the criterion should not give rise to an early-detected symbol error rate that is *higher* than the desired symbol error rate. More subtle, however, is the issue of what consequences an erroneously early-detected symbol 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 information symbol is detected and its corresponding node removed from the code graph, 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 bit at each cycle with a static threshold, θ. (Alternatively, the threshold can be applied after each complete iteration.) If at cycle i, $\tilde{R}_k^i \geq \theta$, then we detect information bit k early and set its value

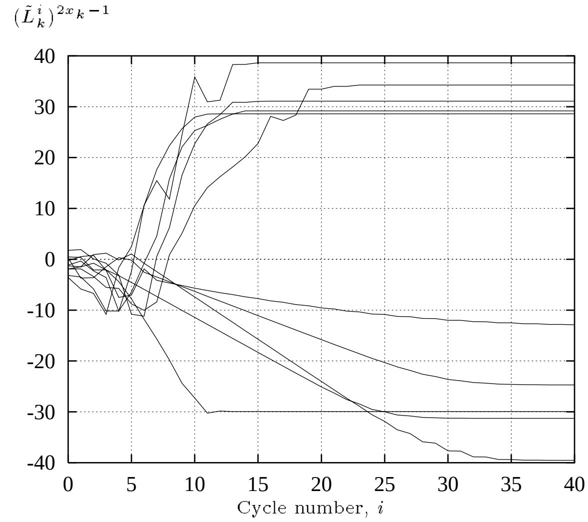

Figure 4: A plot of *corrected* log-odds versus decoding cycle number, for an unbiased sample of 10 cases where the corrected log-odds drop below -10.0 during decoding (the code is described in Section 3).

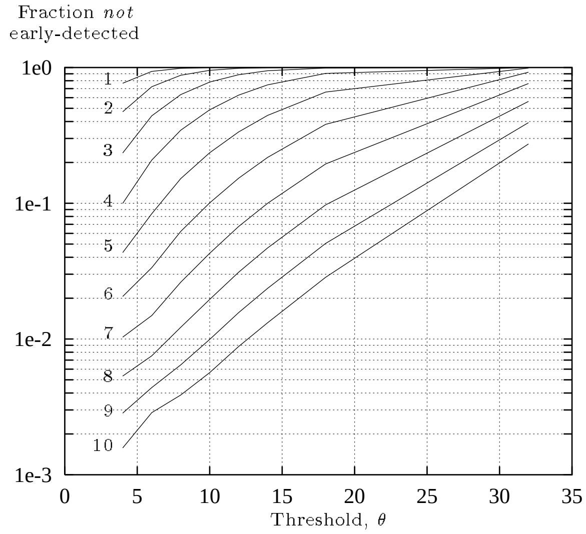

Figure 5: The fraction of information bits *not* early-detected in each of the first 10 decoding cycles as a function of threshold (the code is described in Section 3).

to $\hat{x}_k = \tilde{x}_k^i$. We refer to this method of early detection as *thresholded early detection*. Setting the threshold to a very large positive value effectively causes the algorithm to behave like standard soft iterative decoding. On the other hand, setting the threshold 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. 4 shows an unbiased sample of 10 cases for which the corrected 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 BER of early-detected bits is to use a very large threshold. However, if the threshold is set too high, the fraction of bits that are early-detected will be negligible. In this case, the computational complexity of further iterations will not be significantly reduced. Fig. 5 shows the fraction of bits *not* early-detected as a function of threshold, in each of the first 10 decoding cycles. For a given threshold, 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 section operations have taken place.

We are currently exploring a more sophisticated termination criterion that depends on the log-likelihood of the overall codeword sequence, given the current estimate of the information symbol sequence. Because of the asymptotic equipartition property [25], 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 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 code is a convolutional code that is processed using the forward-backward algorithm [23,24] (see Section 2) in order to obtain *a posteriori* information bit probabilities given *a priori* information bit probabilities. The forward-backward algorithm can be viewed simply as a combination of probabilistic "flows" [19] computed in the forward direction and in the backward direction.

Consider the simple two-state trellis shown in Fig. 6a, that corresponds, say, to the jth constituent code. 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 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 constituent code j alone*:

$$\frac{P(X_k = 1|\mathbf{y}_j)}{P(X_k = 0|\mathbf{y}_j)} = \frac{\alpha_k(0)\gamma_k^1(0)\beta_{k+1}(1) + \alpha_k(1)\gamma_k^1(1)\beta_{k+1}(0)}{\alpha_k(0)\gamma_k^0(0)\beta_{k+1}(0) + \alpha_k(1)\gamma_k^0(1)\beta_{k+1}(1)},$$

where $\mathbf{y}_j$ is the received signal sequence for constituent code 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 are various useful techniques and approximations for decreasing this cost [12, 14], 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. As a consequence, the trellis simplifies to the one shown in Fig. 6b. The trellis can be simplified further by multiplying out the path metrics, giving the trellis shown in Fig. 6c. 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

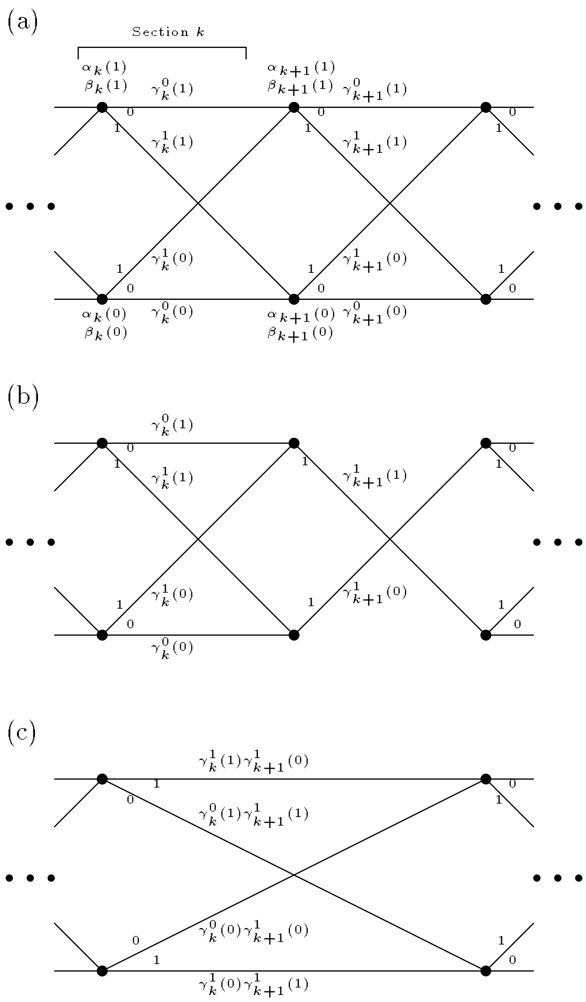

Figure 6: Trellis splicing. (a) shows a two-state trellis with edges accompanied by information bit labels and metrics and with nodes accompanied by flows. (b) and (c): If we know that information bit $k+1$ has a value of 1, we can cut the corresponding section out of the trellis and splice the trellis back together, introducing new information bit labels and new metrics for the connecting edges.

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. 6c have *opposite* values compared to those in Fig. 6a. 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 5, 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.

5 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. 7 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 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 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. 7 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. 8 which shows the computational reduction

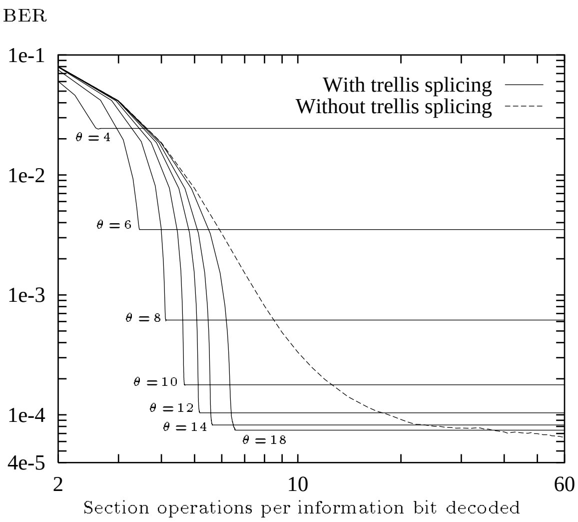

Figure 7: BER performance turbo-decoding with and without thresholded early detection (the code is described in Section 3). In order to average out the effects of block failures, we simulated the transmission of 200 000 information blocks.

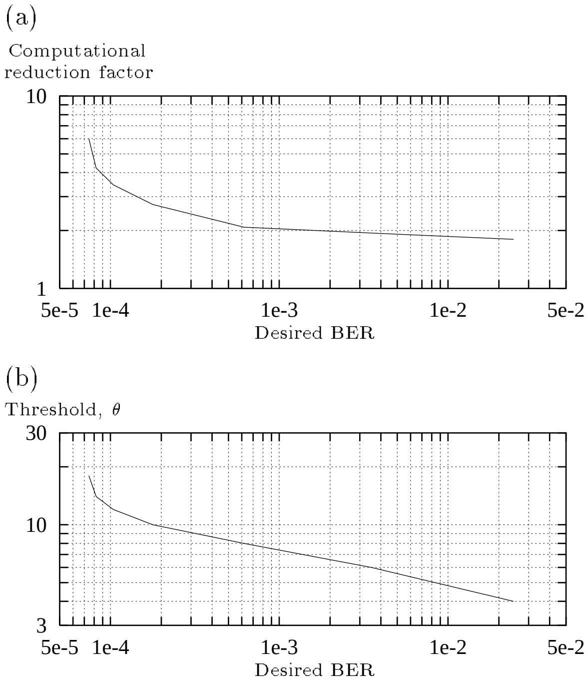

Figure 8: (a) Achievable computational reduction versus BER for the code described in Section 3). (b) Threshold values that give the reductions in (a).

factor and the threshold versus the designed BER. For a specified BER, the achievable computational reduction can be determined from Fig. 8a, and this reduction can be obtained using the threshold determined from Fig. 8b. 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.

6 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 5. 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 5 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.

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] 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.

[3] 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.

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

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

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

[7] 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.

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

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

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

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

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

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

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

[23] 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.

[24] 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.

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