B. J. Frey and D. J. C. MacKay (1998) In *Proceedings of the* 35$^{\text{th}}$ *Allerton Conference on Communication, Control and Computing 1997*, Champaign-Urbana, Illinois.

Trellis-Constrained Codes

Brendan J. Frey
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
bfrey@turbo.beckman.uiuc.edu

David J. C. MacKay
Department of Physics, Cavendish Laboratories
Cambridge University
mackay@mrao.cam.ac.uk

Abstract

We introduce a class of iteratively decodable *trellis-constrained codes* as a generalization of turbocodes, low-density parity-check codes, serially-concatenated convolutional codes, and product codes. In a trellis-constrained code, multiple trellises interact to define the allowed set of codewords. As a result of these interactions, the minimum-complexity single trellis for the code can have a state space that grows exponentially with block length. However, as with turbocodes and low-density parity-check codes, a decoder can approximate bit-wise maximum *a posteriori* decoding by using the sum-product algorithm on the factor graph that describes the code. We present two new families of codes, *homogeneous trellis-constrained codes* and *ring-connected trellis-constrained codes*, and give results that show these codes perform in the same regime as do turbo-codes and low-density parity-check codes.

1 Introduction

Recent interest in the impressive performances of turbocodes and low-density parity-check codes has led to several attempts at generalizing these codes in ways that lead to efficient iterative decoders. In one view that is now quickly propagating across the research network a code is described by graphical constraints on a system of variables, and the iterative decoder is based on a decade-old expert systems algorithm applied to the graph which describes the constraints [1–7]. This *probability propagation algorithm* [8,9] is exact only in cycle-free graphs. However, as evidenced by the excellent error-correcting capabilities of the iterative decoders for turbocodes [10] and low-density parity-check codes [4], the algorithm works impressively well in the graphs that describe these codes, even though they contain many cycles. See [3] and [5] for extensive dissertations on this subject.

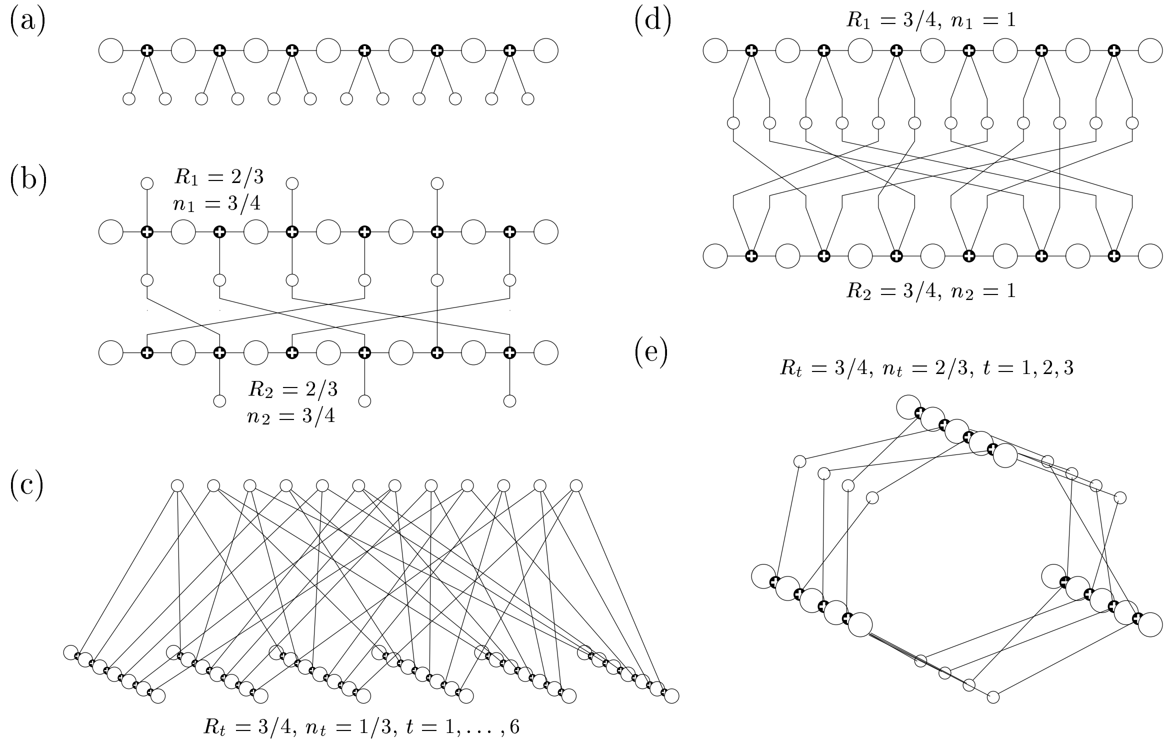

Figure 1: A codeword in a *trellis-constrained code* must simultaneously be a codeword of all the constituent trellises, with the codeword bits reordered. The factor graphs are shown for (a) a single convolutional code; (b) a turbocode, where each parity bit is a codeword bit of only one trellis; (c) a low-density parity-check code; (d) a homogeneous trellis-constrained code; and (e) a ring-connected trellis-constrained code. The small unfilled discs represent codeword bits.

Fig. 1a shows the *factor graph* (see [11] in these proceedings) for a trellis. Unlike in a trellis, in a factor graph the values that each state variable (large white discs) can take on are not explicitly shown. Any two adjacent state variables and the corresponding codeword bits (small white discs) must satisfy a linear set of equations (represented by the small black discs with a "+" inside). This representation is called a "factor graph" because it shows how the indicator function for allowed trellis behaviors factors into a product of local functions. Associated with each black disc is a local function of its neighboring variables. Each function evaluates to 1 if its neighboring variables satisfy the local set of linear equations, and to 0 otherwise. The global function is equal to the product of the local functions. A given configuration of the codeword bits is a codeword if the global function evaluates to 1 for some configuration of the state variables. In general, the local functions may be nonlinear, the factor graph variables may be real-valued, and the local functions may evaluate to elements of a semi-ring.

Although factor graphs are less explicit about local relationships than are trellises, factor graphs allow us to represent a richer set of systems. Fig. 1b shows the factor graph for a simple turbocode. A single trellis for the same turbocode would have an unwieldly large number of states. More important than representation, a factor graph provides a framework for iterative decoding via message passing on the graph. The probability propagation algorithm [8, 9], a.k.a. the *sum-product* algorithm [1, 2], can be applied to a factor graph to approximate bit-wise maximum *a posteriori* (MAP) decoding. (In the special case of a turbocode, this general algorithm reduces to turbodecoding [6,7].) Two different factor graphs for the same code may give decoders with different performances.

As another example, Fig. 1c shows the factor graph for a simple low-density parity-check code. Each of the six trellises is a simple parity-check trellis that enforces even parity on its six codeword bits [12].

In a sense, whereas the trellis assisted in the design of low-complexity codes and exact linear-time probabilistic decoders (the Viterbi algorithm and the forward-backward algorithm), the factor graph assists in the design of high-complexity codes and approximate linear-time probabilistic decoders. In this paper, we present a general class of high-complexity, linear-time decodable codes that retain the chain-type structure of trellises locally, as do turbocodes and to a lesser degree low-density parity-check codes. A codeword in a *trellis-constrained code* (TCC) must simultaneously be a codeword of multiple constituent trellises. So, if $f_t(\mathbf{x})$ is the constituent codeword indicator function for trellis $t \in \{1, \ldots, T\}$, the global codeword indicator function is

$$f(\mathbf{x}) = \prod_{i=1}^{T} f_t(\mathbf{x}). \tag{1}$$

Each constituent indicator function is given by a product of the local functions within the corresponding trellis. Usually, the codeword bits interact with the constituent trellises through permuters that rearrange the order of the codeword bits.

For the turbocode in Fig. 1b, there are two constituent functions, $f_1(\cdot)$ and $f_2(\cdot)$. $f_1(\cdot)$ indicates that the upper row of codeword bits are valid output from a convolutional encoder with the middle row of codeword bits as input. $f_1(\cdot)$ does *not* directly place any restrictions on the lower row of codeword bits, so it effectively only checks 3/4 of the codeword bits. $f_2(\cdot)$ indicates that the lower row of codeword bits are valid output from a convolutional encoder with the middle row of codeword bits as input. In contrast to $f_1(\cdot)$, $f_2(\cdot)$ does not place any restrictions on the *upper* row of codeword bits.

The rate R of a TCC is related to the rates of the constituent trellises R_t, $t = 1, \ldots, T$ in a simple way. If n_t is the fraction of codeword bits that trellis t checks, then trellis t removes at most $(1 - R_t)n_t N$ binary degrees of freedom from the code. It may remove a small number less if some of its constraint equations are linearly dependent on those given by other constituent trellises. For large, randomly generated permuters this effect is quite small, so we will ignore it when computing rates in the remainder of this paper. (As a result, the actual rates may be slightly *higher* than the given rates.) The total number of binary degrees of freedom left after all trellis constraints are satisfied is $N - \sum_{t=1}^{T}(1 - R_t)n_t N$, so the rate of the TCC is

$$R = 1 - \sum_{t=1}^{T}(1 - R_t)n_t. \tag{2}$$

From this equation, it is easy to verify that the turbocode in Fig. 1b has rate 1/2 and that the low-density parity-check code in Fig. 1c also has rate 1/2. (Note that a k-bit parity-check trellis has rate $(k-1)/k$.)

Unlike encoding turbocodes and serially-concatenated convolutional codes, encoding a general TCC takes quadratic time in N. In a general TCC, we can designate a subset of the codeword bits as the systematic bits of the entire code and then use Gaussian elimination to compute a generator matrix (once only). Using such a systematic generator matrix for encoding requires $R(1 - R)N^2$ binary operations.

Decoding a TCC involves performing the forward-backward algorithm for each trellis and exchanging information between trellises in the fashion specified by the sum-product algorithm. The constituent trellises may be processed in parallel or sequentially.

In the following two sections, we present two new families of TCC's and show that they perform in the same regime as do turbocodes and low-density parity-check codes.

2 Homogeneous Trellis-Constrained Codes

In a turbocode, the constituent trellises share only a systematic subset of their codeword bits. The other parity bits of each constituent encoder are not constrained by the other encoders. Fig. 1d shows the factor graph for a simple *homogeneous TCC* with $T = 2$, in which *all* of the bits are constrained by each constituent trellises. From the general rate formula in (2), we see that the rate for a homegenous turbocode is

$$R = 1 - T(1 - R_{\text{avg}}), \tag{3}$$

where $R_{\text{avg}} = \left(\sum_{t=1}^{T} R_t\right)/T$.

One difference between the homogeneous TCC and the turbocode is that the rate of a homogeneous TCC decreases directly with the number of trellises T. In the simulations discussed below, we used $T = 2$ and $R_1 = R_2 = R_{\text{avg}} = 3/4$ to get $R = 1/2$. To obtain the same rate with $T = 3$ would require $R_{\text{avg}} = 5/6$. In contrast, the rate for a turbocode varies roughly inversely with T. A rate $1/2$ turbocode with $T = 3$ can be obtained with $R_1 = R_2 = R_3 = 3/4$. Another difference is that the permuter length of a homogeneous TCC is N, whereas for a turbocode, the permuter length is RN.

2.1 Encoding and Decoding

The trellises in a homogeneous TCC share all their bits, so we can't simply encode by dividing the bits in each constituent trellis into a systematic set and a parity set and running a linear-time encoding method for each trellis, as is possible in a turbocode. Instead, we apply a previously computed generator matrix to a previously selected systematic subset of codeword bits, which takes $R(1 - R)N^2$ binary operations.

The iterative decoder processes each constituent trellis using the forward-backward algorithm, and passes "extrinsic information" between the trellises in the manner specified by the sum-product algorithm. For two trellises, the decoding schedule is straightforward. For $T > 2$, different decoding schedules are possible. The trellises may be processed sequentially, in which case the current trellis uses the most recently computed probabilities produced by the other trellises. Alternatively, the trellises may be processed in parallel, in which case the current trellis uses the probabilities produced by the other trellises in the previous decoding iteration.

For the sake of gaining insight into these new compound codes and the behavior of their iterative decoders, we prefer to decode until a codeword is found or we are sure the iterative decoder has failed to find a codeword. After each decoding iteration, the current bit-wise MAP estimates are used to determine whether a valid codeword has been found, in which case the iterative procedure is terminated. If 100 iterations are completed without finding a codeword, we label the block a decoding failure. Notice that given the factor graph of a code, determining that a codeword is valid is simply a matter of checking that all the local functions evaluate to 1.

2.2 Performance on an AWGN Channel

Using Monte Carlo, we estimated the performance of an $N = 131,072$, $T = 2$ homogeneous TCC with $R_1 = R_2 = 3/4$, giving $R = 1/2$. (See App. A for a description of how the BER confidence intervals were computed.) Each rate 3/4 trellis was obtained by shortening every fifth bit of a rate 4/5 nonsystematic convolutional code with maximum $d_{\min}$. (The generator polynomials for this code are given in [13] and are $(32, 4, 22, 15, 17)_{\text{octal}}$.) Fig. 2 shows the performance of this homogeneous TCC, relative to the turbocode introduced by Berrou *et. al.* [14] and the best rate 1/2, $N = 65,389$ low-density parity-check code published to date [4]. Although it does not perform as well as the turbocode, it performs significantly better than the low-density parity-check code. We believe there is room for improvement here, since we chose the set of generator polynomials that gave maximum $d_{\min}$ and this is quite likely not the best choice. (Keep in mind, however, that the performance of a homogeneous TCC is not necessarily governed by the same constituent trellis properties that govern the performance of a turbocode.) Of significant importance compared to turbocodes, we have observed that this homogeneous TCC does not have low-weight codewords.

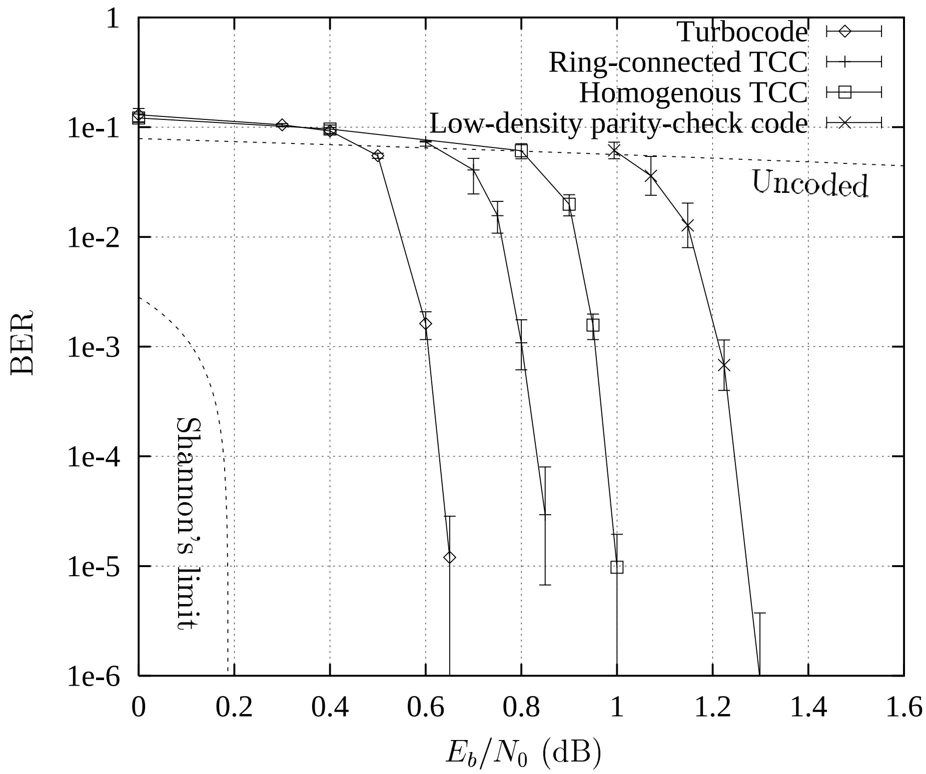

Figure 2: The performances of a homogeneous TCC and a ring-connected TCC compared to the best rate 1/2 turbocode and low-density parity-check code performances published to date [4, 10].

3 Ring-Connected Trellis-Constrained Codes

Fig. 1e shows the factor graph for a simple *ring-connected TCC* with $T = 3$. This code can be viewed as a serially-concatenated convolutional code [15, 16] in which some of the output bits are constrained to be equal to some of the input bits. The factor graph thus forms a ring of connected trellises. In the ring-connected TCC shown, each constituent trellis checks exactly $2/T$ of the codeword bits. From the general rate formula in (2), the rate for such a ring-connected turbocode is

$$R = 2(R_{\text{avg}} - 1/2). \tag{4}$$

Unlike turbocodes and homogeneous TCC's, for such a ring-connected TCC the rate does not depend on the number of constituent trellises T. However, the permuter lengths are $1/T$ relative to the block length.

3.1 Encoding and Decoding

For ring-connected TCC's, we cannot use the same type of encoding procedure that is used for serially-concatenated convolutional codes, since some of the "output" bits must match some of the "input" bits. As with the homogeneous TCC, we can encode with approximately $R(1-R)N^2$ binary operations by applying a previously computed generator matrix to a previously selected systematic subset of codeword bits. However, if $R_t = 3/4$, $t = 1, \dots, T$ (giving $R = 1/2$), encoding computations can be saved in the following way. First, we pick $3N/2T$ systematic bits for trellis 1 and generate the corresponding $N/2T$ parity bits, using a number of computations that is linear in N. Then, we work around the ring and for each trellis pick $N/2T$ systematic bits and generate the corresponding $N/2T$ parity bits. When all but two trellises have been encoded, the last two trellises are used to form a binary vector that when multiplied by a previously computed $N/T \times N/T$ binary matrix yields the final set of N/T parity bits. The computations for the last step dominate and require approximately N^2/T^2 binary operations. For $R_{\text{avg}} = 3/4$, $R = 1/2$, $T = 3$, the first method described above takes $N^2/4$ operations, whereas the second method takes $N^2/9$ operations.

As with homogeneous TCC's, when $T > 2$ ring-connected TCC's can be iteratively decoded using a variety of schedules. In our simulations, we process the trellises sequentially while passing probabilities around the ring in one direction. Iterative decoding continues until a valid codeword is found or until 100 iterations are completed.

3.2 Performance on an AWGN Channel

We estimated the performance of an $N = 131,070$, $T = 3$ ring-connected TCC with $n_1 = n_2 = n_3 = 2/3$ and $R_1 = R_2 = R_3 = 3/4$, giving $R = 1/2$. Each rate 3/4 trellis used the generator polynomials $(12, 4, 17, 11)_{\text{octal}}$, which we found by trial and error. The constituent codeword bits were shared alternately with the two neighboring trellises. Fig. 2 shows the performance of this ring-connected TCC. It performs signficantly better than the homogeneous TCC and the low-density parity-check code and only 0.2 dB worse than the turbocde.

4 Discussion

There is some literature on the properties of the constituent trellises that make good turbocodes [10, 17, 18] and serially-concatenated convolutional codes [19]. We are currently exploring similar arguments for choosing the properties of the constituent trellises that will make good homogeneous TCC's and ring-connected TCC's. The behavior of these two new families of TCC's is quite different from that of turbocodes, so we expect different properties to be relevant. For example, in order to avoid low-weight codewords in a turbocode, we try to avoid permuters that keep a pair of bits the same distance apart in the two constituent trellises [10]. This degenerate effect is "broken" by the ring of a ring-connected TCC, which requires not only that two neighboring trellises have an equal set of shared "input" bits, but also that their "output" bits must satisfy the constraints given by the ramainder of the ring. However, in a ring-connected TCC low-weight codewords are introduced by permuters that preserve a degenerate pattern around the entire ring. Another important question is what properties of a TCC lead to the initial drop (say down to a BER of 0.01)? These properties may very well be independent (or even contrary to) those properties that give high minimum distance.

We believe that the general class of "trellis-constrained codes" presented in this paper is a fruitful generalization of several other iteratively decodable codes. If we think of entire trellises as nodes in a graph whose edges represent bundles of codeword bits, it is evident that there are a variety of other interesting new TCC's aside from those shown in Fig. 1 that have yet to be explored.

Acknowledgements

We appreciate discussions we had with members of the "Ulm group", Dave Forney, Ralph Koetter, Frank Kschischang, Andi Loeliger, Bob McEliece, Michael Tanner, and Niclas Wiberg, as well as discussions we had with Rob Calderbank and Alexander Vardy.

Appendix A: Computing BER Confidence Intervals

When analytic methods are not available for computing bit error rates in error-correcting coding systems, we must resort to simulation. Estimated BER's can vary significantly from experiment to experiment, and so it is often desirable to include confidence intervals. This is especially important for the long block length codes discussed in this paper, since significant variability can be introduced by our inability to simulate enough blocks to pin down the word error rate. Also, for low bit error rates (*e.g.*, 10^{-6}) we may not be able to measure the distribution of bit errors within erroneously decoded words. In this section, we present a Monte Carlo approach for estimating the median and a 2.5% / 97.5% confidence interval for the BER, based on errors measured by Monte Carlo simulation.

The error model contains two parameters: the probability p_w of word error, and the probability p_b of bit error *within erroneous words*. This is a rather crude approximation, since in practice we expect there to be more than one failure mode, *i.e.*, there ought to be several p_b's corresponding to different failure modes.

Let M be the number of words transmitted and let n_w be the number of measured word errors. Let K be the number of information bits per word, and let n_b be the *total* number of bit errors measured while transmitting all M blocks. We will refer to the measured values as the data, $\mathcal{D} = \{n_w, n_b\}$. From the Bayesian perspective, before observing $\mathcal{D}$, we place a prior distribution $p(p_w, p_b)$ on the error model parameters. After observing $\mathcal{D}$, we draw conclusions (*e.g.*, compute a confidence interval) from the posterior distribution $p(p_w, p_b|\mathcal{D})$, where

$$p(p_w, p_b|\mathcal{D}) \propto p(p_w, p_b)P(\mathcal{D}|p_w, p_b). \tag{5}$$

In this equation, the constant of proportionality does not depend on p_w or p_b. The last factor $P(\mathcal{D}|p_w, p_b)$ is called the likelihood.

We let p_w and p_b be independent beta-distributed random variables under the prior: $p(p_w, p_b) = p(p_w)p(p_b)$, where

$$p(p_w) \propto p_w^{\alpha_w - 1}(1 - p_w)^{\beta_w - 1}, \quad \text{and} \quad p(p_b) \propto p_b^{\alpha_b - 1}(1 - p_b)^{\beta_b - 1}. \tag{6}$$

In frequentist terms, α_w and β_w have the effect of shrinking our measurements toward a word error rate of $\alpha_w/(\alpha_w + \beta_w)$, where the influence of this shrinkage grows with $\alpha_w + \beta_w$. Typically, we choose $\alpha_w = \beta_w = 1$, which gives a uniform prior over p_w as shown in Fig. 3a.

As for the prior over p_b, it should be chosen while keeping in mind the behavior of the decoder. If the main source of bit errors is a failure to decode, and if we believe that for failures the decoder will produce a probability of bit error that is roughly equal to the probability p_u of bit error for uncoded transmission, then the prior should place weight on $p_b = p_u$. In this case, we choose $\alpha_b = 2$ and $\beta_b = 1/p_u$, which ensures that the mode of the prior occurs at p_u and that the prior is relatively broad. For example, for $E_b/N_0 = 1$ dB we have $p_u = 0.0563$, and so We choose $\alpha_b = 2$ and $\beta_b = 1/0.0563 = 17.76$, giving the prior distribution for p_b shown in Fig. 3b.

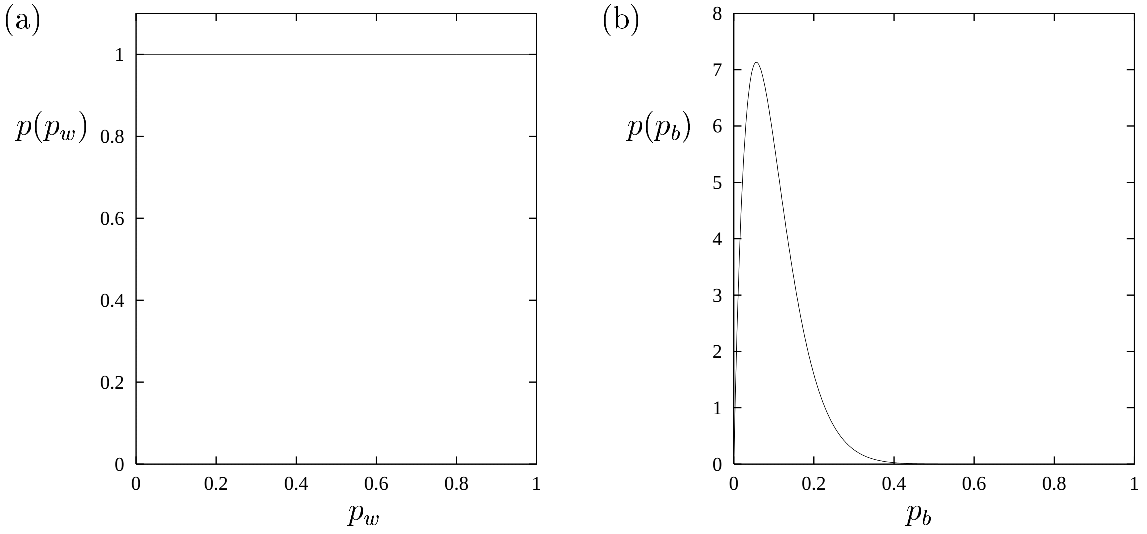

Figure 3: (a) The prior distribution over the probability of word error p_w. (b) The prior distribution over the probability of bit error p_b within erroneous words. This distribution is designed so that its median is equal to the probability of bit error for uncoded transmission.

It is straightforward to show that the likelihood is

$$P(\mathcal{D}|p_w, p_b) = P(n_w, n_b|p_w, p_b) \propto p_w^{n_w}(1 - p_w)^{M - n_w} p_b^{n_b}(1 - p_b)^{n_w K - n_b}. \tag{7}$$

This distribution is the product of a binomial distribution for the number of word errors and a binomial distribution for the number of bit errors. Combining this likelihood with the prior, we obtain the posterior,

$$p(p_w, p_b|\mathcal{D}) \propto p_w^{\alpha_w - 1 + n_w}(1 - p_w)^{\beta_w - 1 + M - n_w} p_b^{\alpha_b - 1 + n_b}(1 - p_b)^{\beta_b - 1 + n_w K - n_b}, \tag{8}$$

which is just the product of a beta distribution over p_w and a separate beta distribution over p_b. Of course, we are actually interested in the posterior distribution $p(p_w p_b|\mathcal{D})$ over the total probability of a bit error $p_w p_b$. A sample is obtained from $p(p_w p_b|\mathcal{D})$ by drawing p_w — p_b pairs from the posterior in (8) and taking the product of p_w and p_b in each pair. This sample is sorted in ascending order, and the value of $p_w p_b$ occuring half-way through the sorted list is taken as an estimate of the median of $p(p_w p_b|\mathcal{D})$. Similarly, the values of $p_w p_b$ occuring 2.5% and 97.5% through the sorted list are taken as the 95% confidence interval.

For the homogeneous TCC, we simulated the transmission of $M = 332$ blocks at $E_b/N_0 = 0.95$ dB using a block length of $N = 131,072$. We measured $n_w = 14$ and $n_b = 34,225$. Using the prior presented above for the slightly higher value of $E_b/N_0 = 1$ dB, a sample of 1000 points from the posterior over p_w and p_b was obtained and is shown in Fig. 4a. As described above, for $\gamma = 0.025$, 0.5 and 0.975, we found the values for p_γ

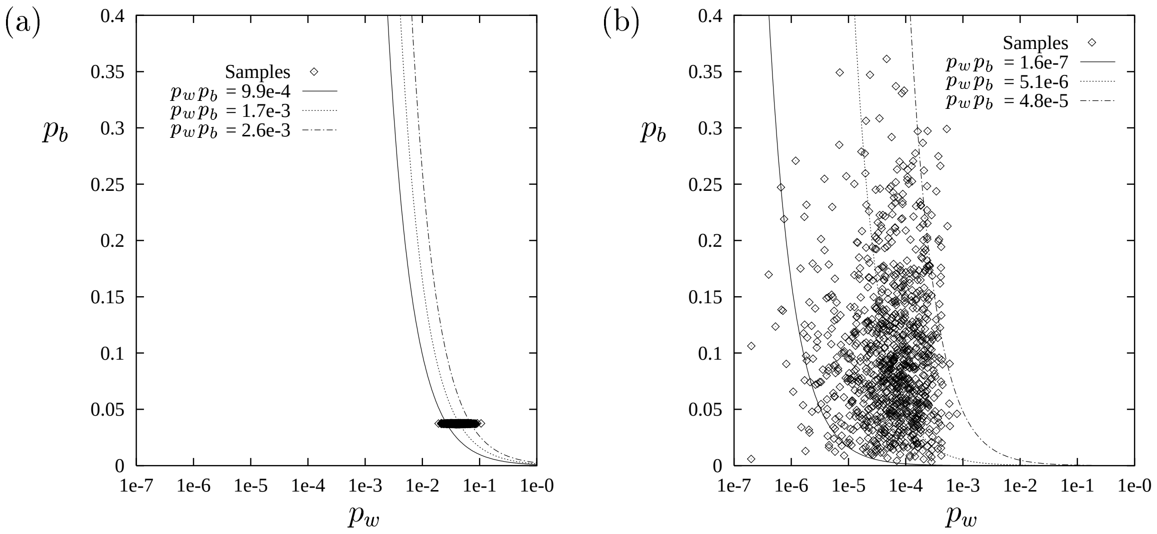

Figure 4: (a) A 1000-point sample from $p(p_w, p_b|\mathcal{D})$ for $M = 332$, $n_w = 14$, $K = 65,536$ and $n_b = 34,225$, for the prior described in the main text. (b) A 1000-point sample from $p(p_w, p_b|\mathcal{D})$ for $M = 10,216$, $n_w = 0$, $K = 65,536$ and $n_b = 0$, for the same prior.

such that $\hat{p}(p_w p_b < p_\gamma|\mathcal{D}) = \gamma$, where $\hat{p}$ is the sample distribution. The corresponding three curves of the form $p_w p_b = p_\gamma$ are shown in Fig. 4a, and the corresponding values of p_γ give a median of 1.7×10^{-3} and a 95% confidence interval of $(9.9 \times 10^{-4}, 2.6 \times 10^{-3})$. Clearly, in this case it is the values for p_w that determine the p_γ's for these curves, whereas the values for p_b are well-determined by the measurements. We could have assumed that p_b took on its measured value instead of sampling from the posterior.

For the homogeneous TCC described above, we also simulated the transmission of $M = 10,216$ blocks at $E_b/N_0 = 1.0$ dB. In this case, we measured $n_w = 0$ and $n_b = 0$. Using naive methods, we might conclude that the bit error rate is 0 and that there isn't any variation in this value. However, the Bayesian technique gives the sample from the posterior shown in Fig. 4b. In this case, the values of both p_w and p_b play a role in determining the p_γ's for the three curves. The median is 5.1×10^{-6} and the confidence interval is $(1.6 \times 10^{-7}, 4.8 \times 10^{-5})$.

References

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

[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, September/October 1995.

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

[4] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, pp. 1645–1646, August 1996. Due to editing errors, reprinted in *Electronics Letters*, vol. 33, March 1997, 457–458.

[5] B. J. Frey, *Bayesian Networks for Pattern Classification, Data Compression and Channel Coding.* Toronto Canada: Department of Electrical and Computer Engineering, University of Toronto, 1997. Doctoral dissertation available at `http://www.cs.utoronto.ca/~frey`.

[6] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models." To appear in *IEEE Journal on Selected Areas in Communications*, available at `http://www.cs.utoronto.ca/~frey`, 1998.

[7] R. J. McEliece, D. J. C. MacKay, and J. F. Cheng, "Turbo-decoding as an instance of Pearl's 'belief propagation' algorithm." To appear in *IEEE Journal on Selected Areas in Communications*, 1998.

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

[9] 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.

[10] 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.

[11] B. J. Frey, F. R. Kschischang, H. A. Loeliger, and N. Wiberg, "Factor graphs and algorithms," in *Proceedings of the* 35^{th} *Allerton Conference*, 1998.

[12] D. J. C. MacKay, "Good codes based on very sparse matrices." Submitted to *IEEE Transactions on Information Theory*, 1997.

[13] D. G. Daut, J. W. Modestino, and L. D. Wismer, "New short constraint length convolutional code constructions for selected rational rates," *IEEE Transactions on Information Theory*, vol. 28, pp. 794–800, September 1982.

[14] 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.

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

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

[17] 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.

[18] D. Divsalar and R. J. McEliece, "Effective free distance of turbocodes," *Electronics Letters*, vol. 32, pp. 445–446, February 1996.

[19] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding." To appear in *IEEE Transactions on Information Theory*, 1997.