B. J. Frey and D. J. C. MacKay (1998) In *Proceedings of the* 35th *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, *homogenous 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.

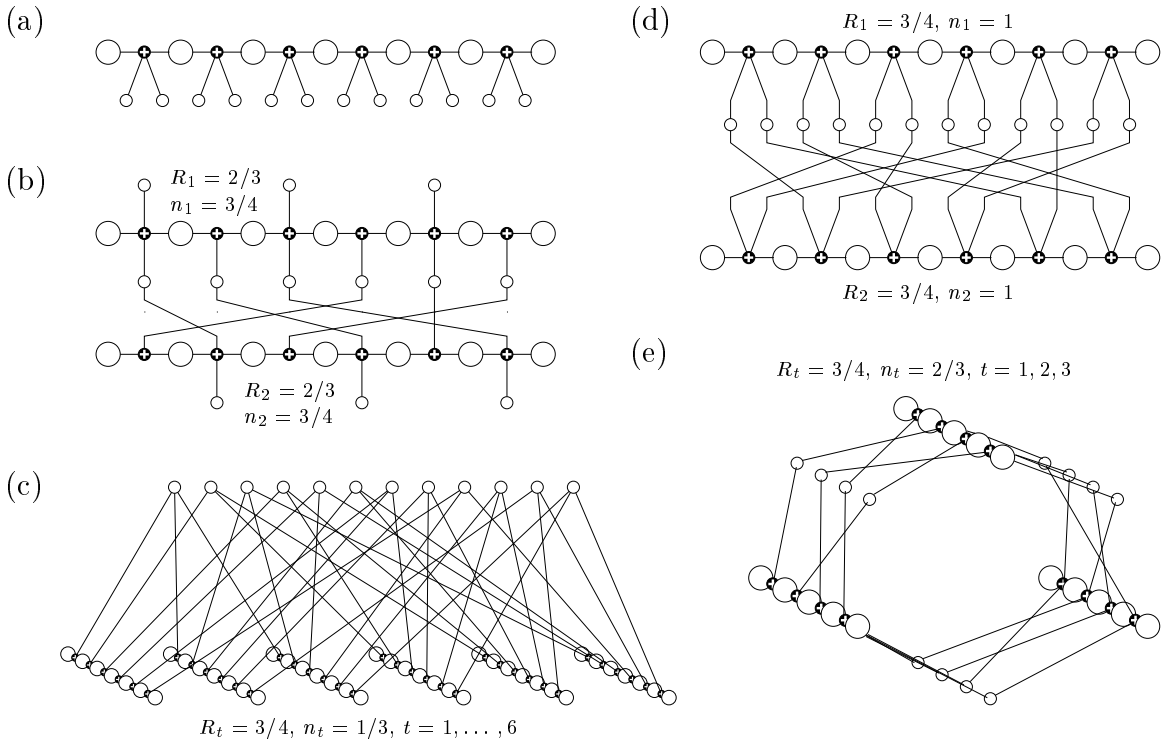

Figure 1: A codeword in a *trellis-constrained code* must simultaneously be a codeword of all the constituent trellises, with the codeword bits reordered. The factor graphs are shown for (a) a single convolutional code; (b) a turbocode, where each parity bit is a codeword bit of only one trellis; (c) a low-density parity-check code; (d) a homogenous 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 Homogenous 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 *homogenous 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 homogenous TCC and the turbocode is that the rate of a homogenous 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 homogenous TCC is N, whereas for a turbocode, the permuter length is RN.

2.1 Encoding and Decoding

The trellises in a homogenous 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 $N = 131,072$, $T = 2$ homogenous 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 homogenous 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 homogenous 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 homogenous TCC does not have low-weight codewords.

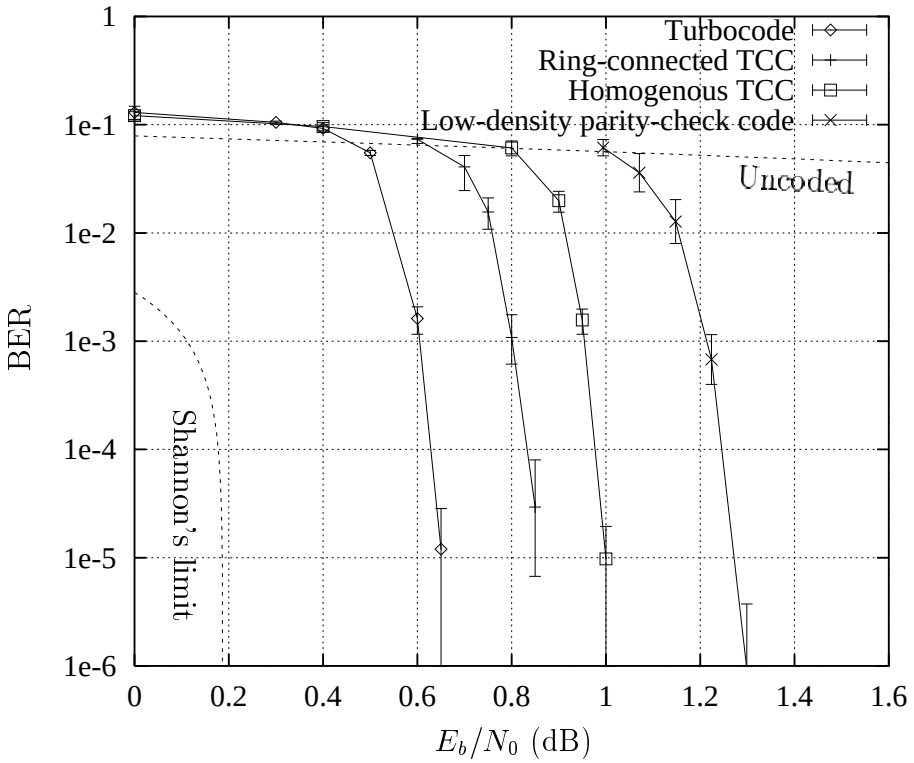

Figure 2: The performances of a homogenous TCC and a ring-connected TCC compared to the best rate 1/2 turbocode and low-density parity-check code performances published to date [4, 10].

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

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.