B. J. Frey and D. J. C. MacKay (1999) In *Proceedings of the* 37th *Allerton Conference on Communication, Control and Computing 1999*, Allerton House, Illinois.

Irregular Turbocodes

Brendan J. Frey
Computer Science, University of Waterloo
Electrical and Computer Engineering, University of Illinois at Urbana
http://www.cs.uwaterloo.ca/~frey

David J. C. MacKay
Department of Physics, Cavendish Laboratories
Cambridge University
http://wol.ra.phy.cam.ac.uk/mackay

Abstract

Recently, several groups have increased the coding gain of iteratively decoded Gallager codes (low density parity check codes) by varying the number of parity check equations in which each codeword bit participates. In regular turbocodes, each "systematic bit" participates in exactly 2 trellis sections. We construct irregular turbocodes with systematic bits that participate in varying numbers of trellis sections. These codes can be decoded by the iterative application of the sum-product algorithm (a low-complexity, more general form of the turbodecoding algorithm). By making the original rate 1/2 turbocode of Berrou *et al.* slightly irregular, we obtain a coding gain of 0.15 dB at a block length of $N = 131,072$, bringing the irregular turbocode within 0.3 dB of capacity. Just like regular turbocodes, irregular turbocodes are linear-time encodable.

1 Introduction

Recent work on irregular Gallager codes (low density parity check codes) has shown that by making the codeword bits participate in varying numbers of parity check equations, significant coding gains can be achieved [1–3]. Although Gallager codes have been shown to perform better than turbocodes at BERs below 10^{-5} [4][1], until recently Gallager codes performed over 0.5 dB worse than turbocodes for BERs greater than 10^{-5}. However, in [3], Richardson *et al.* found irregular Gallager codes that perform 0.16 dB *better* than the original turbocode at BERs greater than 10^{-5} [5] for a block length of $N \approx 131,072$.

[1]Gallager codes to not exhibit decoding errors, only decoding failures, at long block lengths with $N > 5,000$.

In this paper, we show that by tweaking a turbocode so that it is irregular, we obtain a coding gain of 0.15 dB for a block length of $N = 131,072$. For example, an $N = 131,072$ irregular turbocode achieves $E_b/N_0 = 0.48$ dB at BER $= 10^{-4}$, a performance similar to the best irregular Gallager code published to date [3]. By further optimizing the degree profile, the permuter and the trellis polynomials, we expect to find even better irregular turbocodes. Like their regular cousins, irregular turbocodes exhibit a BER flattening due to low-weight codewords.

2 Irregular turbocodes

In Fig. 1, we show how to view a turbocode so that it can be made irregular. The first picture shows the set of systematic bits (middle row of discs) being fed directly into one convolutional code (the chain at the top) and being permuted before being fed into another convolutional code (the chain at the bottom). For a rate 1/2 turbocode, each constituent convolutional code should be rate 2/3 (which may, for example, be obtained by puncturing a lower-rate convolutional code).

Since the order of the systematic bits is irrelevant, we may also introduce a permuter before the upper convolutional code, as shown in the second picture. In the third picture, we have simply drawn the two permuters and convolutional codes side by side.

For long turbocodes, the values of the initial state and the final state of the convolutional chains do not significantly influence performance (*e.g.*, see [6]). So, as shown in the fourth picture, we can view a turbocode as a code that copies the systematic bits, permutes both sets of these bits, and then feeds them into a convolutional code. We refer to this turbocode as "regular", since each systematic bit is copied exactly once.

The final picture illustrates one way the above turbocode can be made irregular. Some of the systematic bits are "tied" together, in effect causing some systematic bits to be replicated more than once. Notice that too keep the rate of the overall code fixed at 1/2, some extra parity bits must be punctured.

More generally, an *irregular turbocode* has the form shown in Fig. 2, which is a type of "trellis-constrained code" as described in [7]. We specify a *degree profile*, $f_d \in [0,1], d \in \{1,2,\ldots,D\}$. f_d is the fraction of codeword bits that have degree d and D is the maximum degree. Each codeword bit with degree d is repeated d times before being fed into the permuter. Several classes of permuter lead to linear-time encodable codes. In particular, if the bits in the convolutional code are partitioned into "systematic bits" and "parity bits", then by connecting each parity bit to a degree 1 codeword bit, we can encode in linear time.

The average codeword bit degree is

$$\bar{d} = \sum_{d=1}^{D} d \cdot f_d \tag{1}$$

The overall rate R of an irregular turbocode is related to the rate R' of the convolutional code and the average degree $\bar{d}$ by

$$\bar{d}(1 - R') = 1 - R. \tag{2}$$

So, if the average degree is increased, the rate of the convolutional code must also be increased to keep the overall rate constant. This can be done by puncturing the convolutional code or by designing a new, higher rate convolutional code.

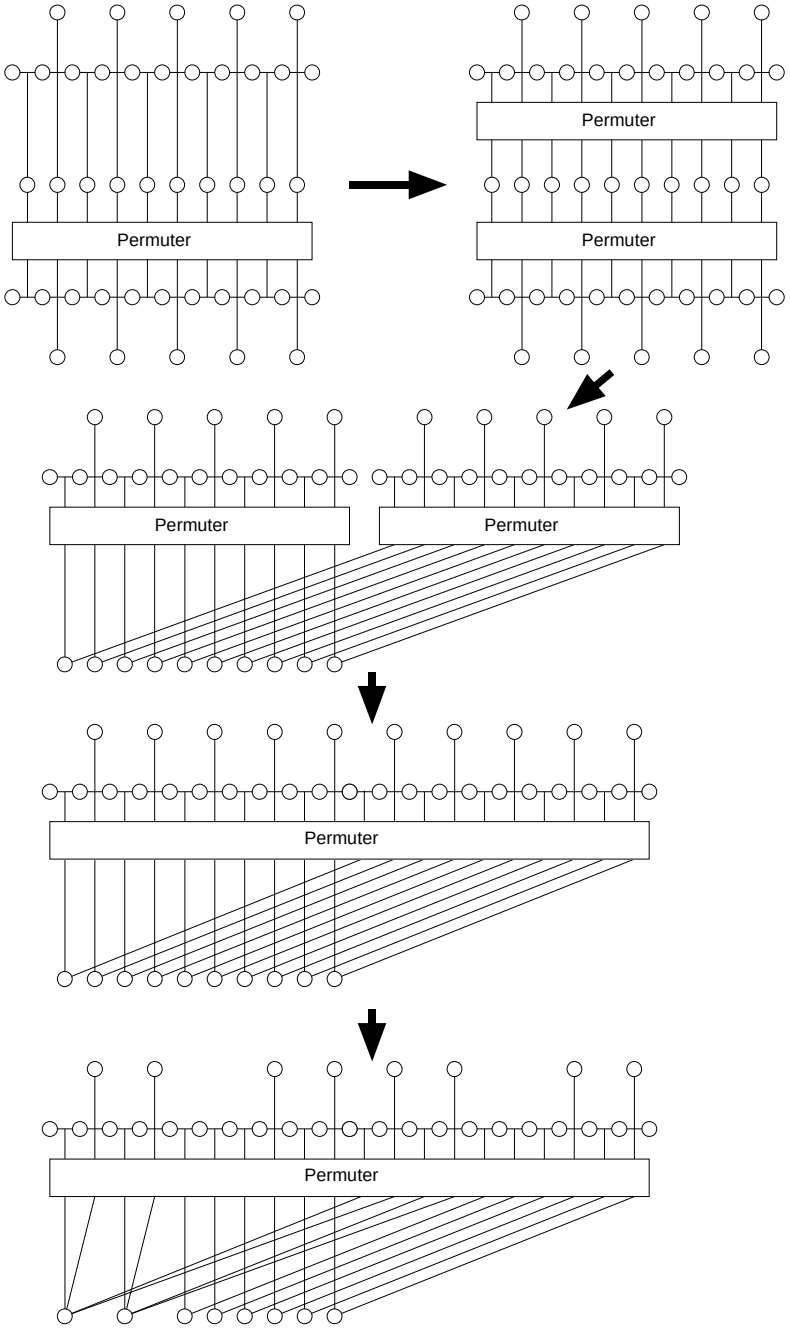

Figure 1: The first 4 pictures show that a turbocode can be viewed as a code that copies the systematic bits, permutes both sets of these bits and then feeds them into a convolutional code. The 5th picture shows how a turbocode can be made irregular by "tying" some of the systematic bits together, *i.e.*, by having some systematic bits replicated more than once. Too keep the rate fixed, some extra parity bits must be punctured. Too keep the block length fixed, we must start with a longer turbocode.

3 Decoding irregular turbocodes

Fig. 2 can be interpreted as the graphical model [6, 8–10] for the irregular turbocode. Decoding consists of the iterative application of the sum-product algorithm (a low-complexity, more general form of turbodecoding) in this graph.

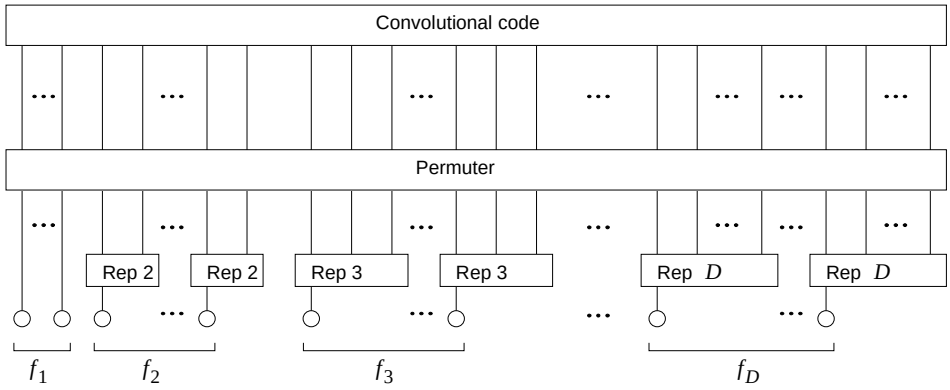

Figure 2: A general *irregular turbocode*. For $d = 1, \ldots, D$, fraction f_d of the codeword bits are repeated d times, permuted and connected to a convolutional code.

After receiving the channel output, the decoder computes the channel output log-likelihood ratios for the N codeword bits,

$$L_1^0, L_2^0, \ldots, L_N^0, \tag{3}$$

and then repeats each log-likelihood ratio appropriately. If codeword bit i has degree d, we set

$$L_{i,1} \leftarrow L_i^0,\ L_{i,2} \leftarrow L_i^0,\ \ldots,\ L_{i,d} \leftarrow L_i^0. \tag{4}$$

Next, the log-likelihood ratios are permuted and fed into the BCJR algorithm [11] for the convolutional code. The BCJR algorithm assumes the inputs are *a priori* log-probability ratios and uses the forward-backward algorithm [12] to compute a set of *a posteriori* log-probability ratios. If codeword bit i has degree d, the algorithm produces d *a posteriori* log-probability ratios,

$$L'_{i,1}, L'_{i,2}, \ldots, L'_{i,d}. \tag{5}$$

For a regular turbocode, there are just two *a posteriori* log-probability ratios, $L'_{i,1}$ and $L'_{i,2}$, for each degree 2 bit and they correspond to the "extrinsic information" produced by each constituent convolutional code.

The current estimate of the log-probability ratio for bit i given the channel output is

$$\hat{L}_i \leftarrow L_i^0 + \sum_{k=1}^{d} (L'_{i,k} - L_{i,k}). \tag{6}$$

To compute the inputs to the BCJR algorithm needed for the next iteration, we subtract off the corresponding outputs from the BCJR algorithm produced by the previous iteration:

$$L_{i,k} \leftarrow \hat{L}_i - L'_{i,k}. \tag{7}$$

So, each iteration consists of computing the inputs to the BCJR algorithm, permuting the inputs, applying the BCJR algorithm, permuting the outputs of the BCJR algorithm, and taking the repetitions into account to combine the outputs to form estimates of the log-probability ratios of the codeword bits given the channel output.

In our simulations, after each iteration, we check to see if the current decision gives a codeword. If it does, the iterations terminate and otherwise, the decoder iterates further until some maximum number of iterations is reached and a decoding failure is declared.

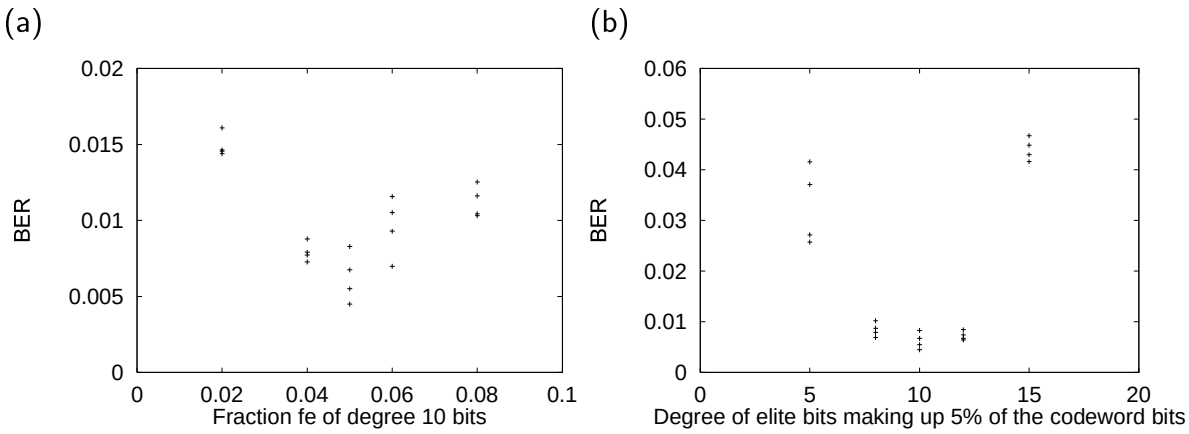

Figure 3: (a) shows the effect of changing the fraction of elite bits on the BER, while keeping the degree of the elite bits fixed at 10. (b) shows the effect of changing the degree of the elite bits on the BER, while keeping the fraction of elite bits fixed at 0.05. For each fraction and degree, the performance of 4 randomly drawn permuters is shown.

4 Selecting the profile

Finding a good profile is not trivial, since the best profile will depend on the parameters of the convolutional code, the permuter and the distortion measure (bit error rate, block error rate, decoding failure rate, high-weight decoding failure rate, *etc.*)

The results we report in this paper were obtained by making small changes to a block length $N = 10,000$ version of the original rate $R = 1/2$ turbocode proposed by Berrou *et al.*. In this turbocode, $f_1 = f_2 = 1/2$ (see Fig. 2) and the convolutional code polynomials are 37 and 21 (octal). The taps associated with polynomial 37 are connected to the degree 2 codeword bits, 1/2 of the taps associated with polynomial 21 are connected to the degree 1 bits, and the remaining 1/2 of the taps associated with polynomial 21 are punctured, giving the required convolutional code rate of $R' = 2/3$.

To simplify our search, we considered profiles where besides degrees 1 and 2, only one other degree, e for "elite", had a nonzero fraction. So, for a code with overall rate R and fraction f_e of degree e elite bits, we have

$$\begin{aligned} f_1 &= 1 - R = 1/2, \\ f_2 &= 1 - f_1 - f_e = 1/2 - f_e. \end{aligned} \tag{8}$$

In this restricted class of codes, irregularity is governed by two parameters, e and f_e.

From (2) it is clear that when the average degree is increased, the rate of the convolutional code must also be increased to keep the overall rate at 1/2. We increased the rate of the punctured convolutional code by further puncturing the taps associated with polynomial 21 to obtain a convolutional code with rate

$$R' = 1 - \frac{1-R}{\bar{d}} = 1 - \frac{1/2}{1/2 + 2(1/2 - f_e) + e f_e}. \tag{9}$$

So, in the codes we explored, the level of puncturing was quite high and some extra low-weight codewords were introduced.

To begin with, we made an irregular turbocode with $e = 10$ (chosen using intuition) and varied f_e from 0.02 to 0.08 while measuring the BER at $E_b/N_0 = 0.6$ dB. In each

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.