Turbo Codes are Low Density Parity Check Codes

David J. C. MacKay

July 15, 2002— Draft 0.2, not for distribution! (First draft written July 5, 1998)

Abstract

Turbo codes and Gallager codes (also known as low density parity check codes) are at present neck and neck in the race towards capacity.

In this paper we note that the parity check matrix of a Turbo code can be written as low density parity check matrix.

Turbo codes and Gallager codes are both greatly superior to error–correcting codes found in textbooks. Some similarities between these codes have been noted. For example, both families of codes can be defined in terms of sparse graphs which define the constraints satisfied by codewords (cite Tanner, Wiberg, Loeliger, Frey, MacKay). Both families of codes are decoded using a local probability propagation algorithm which is known as the sum–product algorithm or belief propagation (cite Wiberg, McEliece and MacKay, Frey).

There are also differences between the two code families. In the original form studied by Gallager and MacKay and Neal, Gallager codes are quick to decode but have an encoding time that scales as N^2, whereas Turbo codes are usually defined in terms of linear time encoders. (Fast-encodeable Gallager codes have recently been investigated, MacKay.) In Turbo codes there is a sharp distinction between the bits viewed as source bits and the bits viewed as parity check bits; they play different roles in the decoding algorithm, and posterior probabilities over the states of parity check bits are usually not computed. In Gallager codes, there is a symmetry between all bits. Turbo codes as originally defined tend to suffer from low weight codewords which cause the asymptotic performance for large E_b/N_0 to have an 'error floor'. Gallager codes, in contrast, show no such error floor, and it has been proved that they have asymptotically good distance properties. Gallager codes are simple to modify in order to create codes with higher or lower rates. In contrast, increasing the rate of a Turbo code can be tricky because simple puncturing of the parity bits might weaken the code by introducing low weight codewords.

Since these two families of codes are both so good in performance, it seems a good idea to try to relate them so as to enhance technology transfer and hybridisation between the two methodologies. However, to our knowledge, only a few researchers have tried to connect these fields together and design new codes (cite Frey and MacKay).

This paper makes a simple observation about Turbo codes: treating a Turbo code as a block code, the parity check matrix of that code is actually a low density parity check matrix. This observation is probably extremely obvious to anyone who is familiar with convolutional codes, but for the benefit of readers like myself who are not, I will spell this out in a little more detail.

1 Definitions

A low density parity check code is a block code which has a parity check matrix, $\mathbf{H}$, every row and column of which is 'sparse'.

A regular Gallager code is a low density parity check code in which every column of $\mathbf{H}$ has the same weight t and every row has the same weight t_r; regular Gallager codes are constructed at random subject to these constraints.

An (N, K) Turbo code is defined by a number of constituent encoders (often, two) and an equal number of 'interleavers' which are $K \times K$ permutation matrices. Without loss of generality, we take the first interleaver to be the identity matrix. The constituent encoders are often convolutional codes. A string of K source bits is encoded by feeding them into each constituent encoder in the order defined by the associated interleaver, and transmitting the bits that come out of each constituent encoder. For simplicity, let us concentrate on Turbo codes with two constituent codes that are both convolutional codes. Often the first constituent encoder is chosen to be a systematic encoder, and the second is a non–systematic one which emits parity bits only. The transmitted codeword then consists of K source bits followed by M_1 parity bits generated by the first convolutional code and M_2 parity bits from the second.

For the purposes of this paper we will not need to discuss the decoder for either of these codes.

One unifying viewpoint for these two code families is in terms of trellis constrained codes. A trellis constrained code is a code whose codewords satisfy a set of constraints, each constraint being compactly described by a trellis in which two or more of the codeword bits participate. Viewing these codes as trellis constrained codes, they appear rather different. The $M \times N$ parity check matrix of a regular Gallager code defines M trellises. Each trellis constrains the parity of t_r of the bits to be even, and each of the N bits participates in t trellises. We can think of a Turbo code as a trellis constrained code in which there are two trellises (figure **??**); the K source bits and the first M_1 parity bits participate in the first trellis and the K source bits and the last M_2 parity bits participate in the second trellis. Each codeword bit participates in either one or two trellises, depending on whether it is a parity bit or a source bit. See figure 2.

However, we will now see that from the point of view of their parity check matrices, Turbo codes are actually very similar to Gallager codes.

1.1 The parity check matrix of a single convolutional code

Note different meaning for parity check matrix from convolutional code literature. Here we are talking about the literal parity check matrix of the code viewed as a linear block code.

A systematic recursive convolutional code, as used in Turbo codes, is equivalent (in the sense that its codewords are the same) to a nonsystematic nonrecursive convolutional code (figure **??**(b)). Now, what parity constraints are satisfied by the latter code? Well, if we pass stream b through the convolutional filter that generated stream a and vice versa, then the two resulting streams are identical. So the parity check matrix of a single convolutional code may be written as a low density parity check matrix as shown in figure **??**(b).

Issue neglected here: termination. Termination simply adds an extra k constraints, where k is the constraint length. Not a big deal.

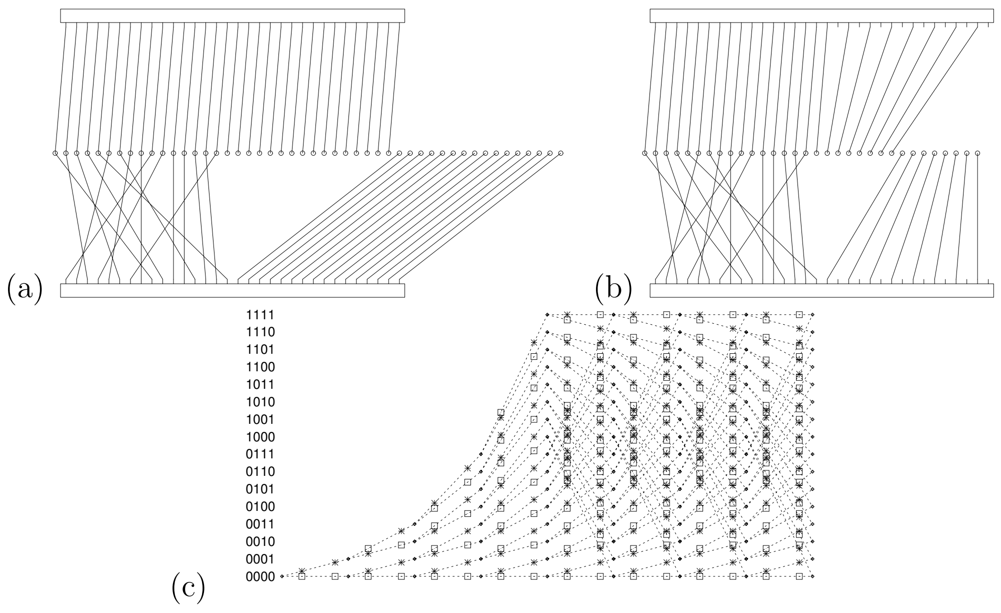

Figure 1: (a,b) A rate 1/3 (a) and rate 1/2 (b) Turbo code represented as trellis constrained codes. The circles represent the codeword bits. The two rectangles represent trellises of rate 1/2 convolutional codes, with the systematic bits occupying the left half of the rectangle and the parity bits occupying the right half. The puncturing of these constituent codes in the rate 1/2 Turbo code is represented by the lack of connections to half of the parity bits in each trellis. (c) The contents of the rectangles. Each trellis is generated by the recursive filter shown in figure 3 which has a state space of 4 bits. In each pair of adjacent bits, one is a systematic bit and one is a parity bit.

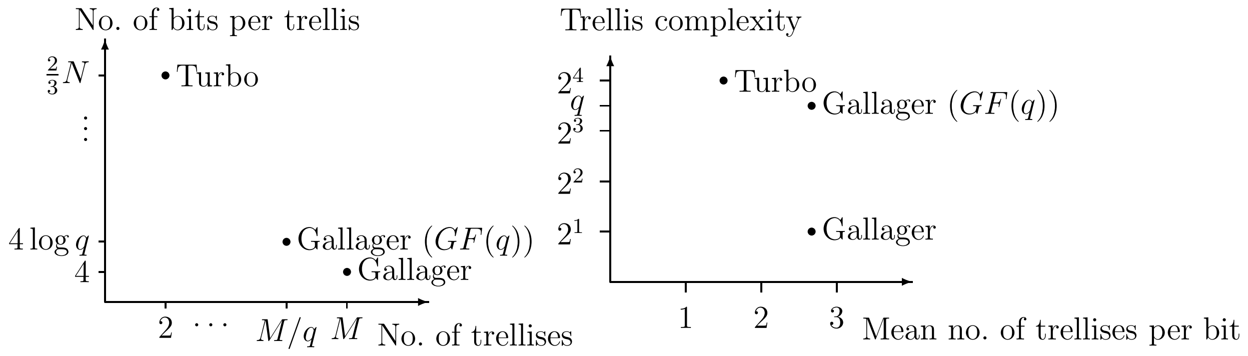

Figure 2: One view of Gallager codes and Turbo codes in 'code space', using rate 1/3 codes as an example. N denotes the block length and M is the number of parity constraints, $M = N - K$, satisfied by a codeword. Points are shown for ordinary binary Gallager codes and for Gallager codes over $GF(q)$ also ($q = 8$ and $q = 16$ have been found useful, cite Davey and MacKay). From this perspective, Gallager codes and Turbo codes seem quite different.

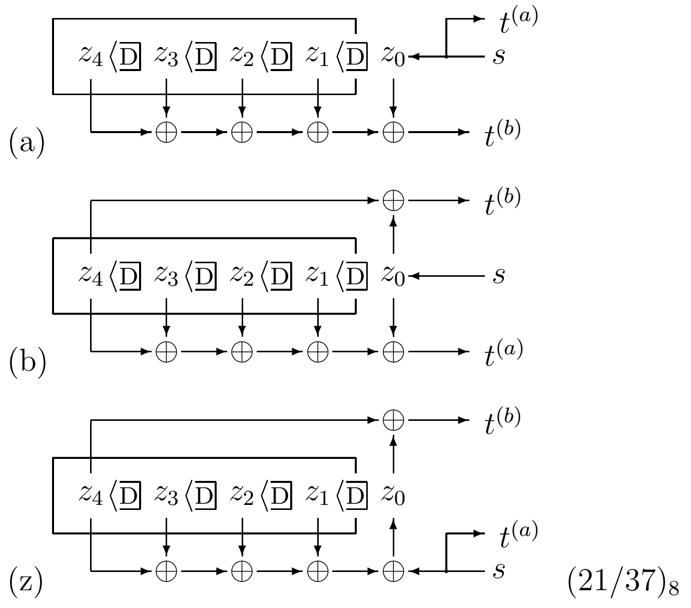

Figure 3: Linear feedback shift registers for generating convolutional codes. $K = 4$.

1.2 The parity check matrix of a Turbo code

Note that for the standard constraint length 4 convolutional codes, the profile of the turbo code's parity check matrix is roughly K columns of weight about 4, and the remaining columns of weight about 5; the row weight is about 7 for all rows.

Here puncturing ignored. How to handle it: if the bit only participates in one check, remove that check. If it participates in more than one check, use row manipulations to create new higher weight checks that don't involve that bit.

Note that classic Turbo codes are punctured down to rate 1/2.

2 Consequences

Some of the theoretical results proved for Gallager codes carry over to Turbo codes. The positive ones (for example, that Gallager codes are 'very good') don't carry over since they rely on creating the whole matrix at random. But the negative ones do. One negative result is that Turbo codes definitely can't get to the Shannon limit unless the constraint length of the constituent codes grows — just as Gallager codes are only very good as the weight per column t increases.

3 Discussion

For those of us who thought that there was a considerable distance in 'code space' between Turbo codes and Gallager codes, this observation forces a shift in viewpoint. It also allows us to roll out a few simple theorems about the maximum likelihood performance of Turbo codes.

So, given that they are such similar codes, what are the differences? Why are regular Gallager codes worse than Turbo codes? We have caught up with Turbo codes only by (1)

(a)

(a')

(b)

(c)

(d)

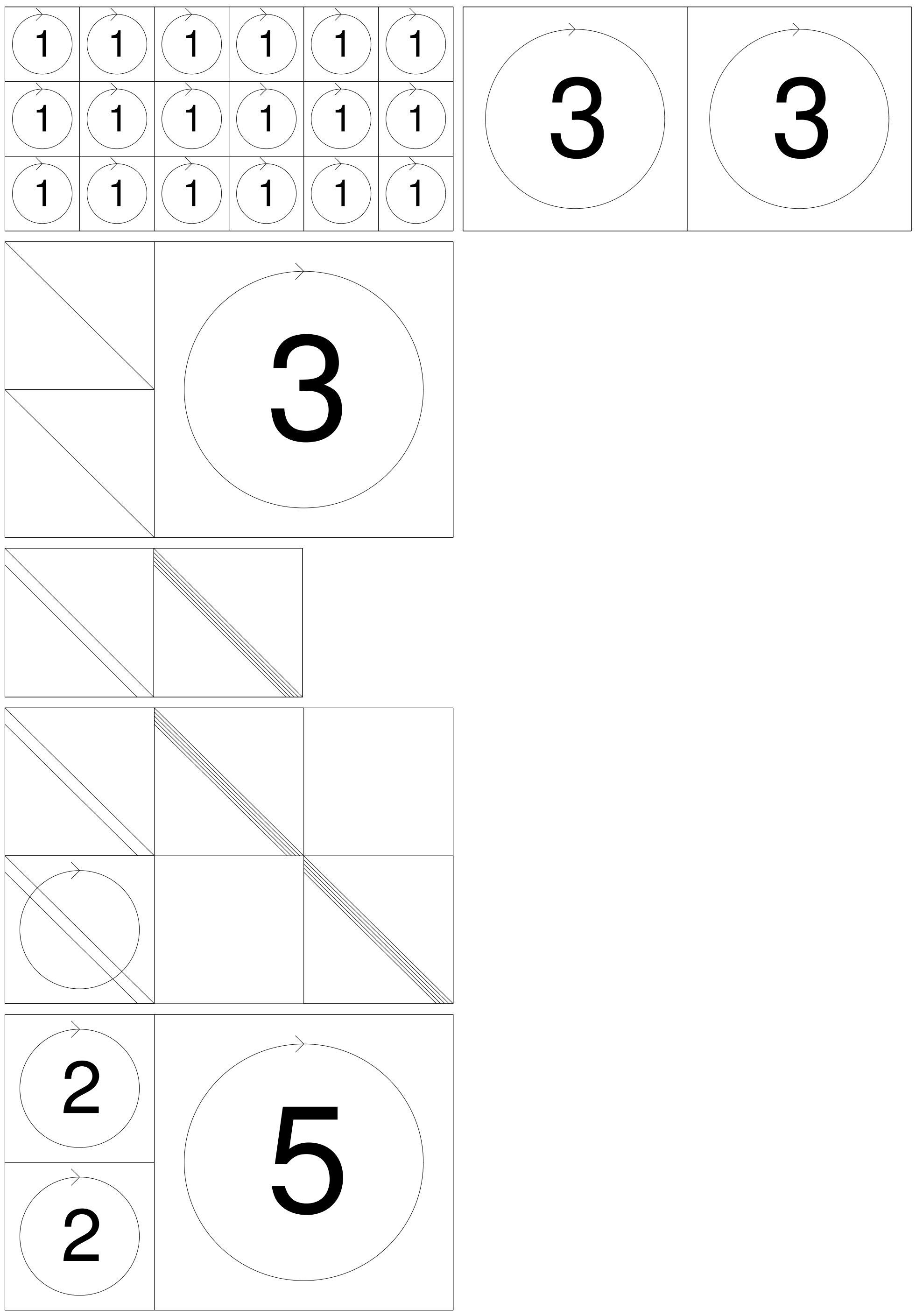

Figure 4: Schematic pictures of the parity check matrices of (a) a regular Gallager code, rate 1/2, (a') an almost regular Gallager code, rate 1/3, (b) a convolutional code, rate 1/2, and (c) a Turbo code, rate 1/3. Notation: A diagonal line represents an identity matrix. A band of diagonal lines represent a band of diagonal 1s. A circle inside a square represents the random permutation of all the columns in that square. A number inside a square represents the number of random permutation matrices superposed in that square. Horizontal and vertical lines indicate the boundaries of the blocks within the matrix. (d) shows another code with roughly the same profile as a Turbo code.

making them over GF(16); (2) making them irregular. But notice these Turbo codes are binary and their parity check matrices are pretty near regular!

I have some ideas about why Turbo are better.

Notice that the standard way of decoding a Gallager code is not how Turbo codes are decoded. Message passing different and would be more inaccurate in the Gallager style. Turbo sends messages all the way along trellises, so the within–trellis messages are correct.

3.1 Wasted checks

Consider a Gallager code with $t_r = 4$. Imagine that of the four bits that participate in one check, three of them happen to be well–determined given the output of the channel. This check allows us instantly to correct the fourth bit, but then it plays no useful role. This is not the way a good code works!

In contrast, in a Turbo code, if some bits are well determined, the sole effect is to prune the trellis. Whatever bits are well–determined, the remaining bits participate in a trellis that looks (locally) just the same.