INSTITUTE OF PHYSICS PUBLISHING JOURNAL OF PHYSICS A: MATHEMATICAL AND GENERAL

J. Phys. A: Math. Gen. **36** (2003) 1267–1288 PII: S0305-4470(03)37052-0

Renormalization group approach to error-correcting codes

Jonathan S Yedidia[1] **and Jean-Philippe Bouchaud**[2]

[1] MERL, 201 Broadway, 8th Floor, Cambridge, MA 02139, USA
[2] SPEC CEA-Saclay, Orne des Merisiers, 91191 Gif Sur Yvette, France

E-mail: yedidia@merl.com and bouchaud@spec.saclay.cea.fr

Received 15 May 2002, in final form 31 October 2002
Published 22 January 2003
Online at stacks.iop.org/JPhysA/36/1267

Abstract
We explain an algorithm that approximately but efficiently assesses the performance of belief propagation decoding for particular parity check error-correcting codes of large, but finite, blocklength. This algorithm is based on the renormalization group approach from physics: the idea is to continually replace an error-correcting code with a simpler error-correcting code that has nearly identical performance, until the code is reduced to a small enough size that its performance can be computed exactly.

PACS numbers: 07.05.Mh, 02.70.−c

1. Introduction

A fundamental problem in the field of information theory is the design of optimal or nearly optimal error-correcting codes that can also be decoded practically. This problem can now be considered essentially solved in the small blocklength (e.g. $N < 100$) and very large blocklength (e.g. $N > 10^6$) regimes. However, error-correcting codes that are used in practical situations typically have blocklengths in an intermediate regime (between say $N = 200$ and $N = 10^4$). The reason that intermediate blocklength codes are used in practice is that larger blocklength codes have better performance, but also greater complexity and longer decoding times, so one will normally choose the code with the largest blocklength for which the cost and delay caused by decoding is still tolerable.

In the small blocklength regime, classical coding theory, as summarized in textbooks such as [1], provides a panoply of codes of different blocklengths and rates, many of which are known to be optimal or nearly optimal. As long as the blocklength is small enough, these codes can also be decoded practically (and optimally) using maximum-likelihood decoders.

In the last few years, the problem of finding good codes in the very large blocklength regime has been essentially solved but in a very different way; by focusing on parity check codes defined using sparse parity check matrices [2]. These kinds of codes were first introduced by Gallager in 1962 [3], but were not properly appreciated until recently. In the last decade, however, new and improved codes defined by sparse generalized parity check matrices (such as turbo codes [4, 5], irregular low-density parity check (LDPC) codes [6–10], Kanter–Saad codes [2, 11, 12], repeat-accumulate codes [13] and irregular repeat-accumulate codes [14]) have been the object of intense study. Such codes have three particularly noteworthy advantages. First, they can be efficiently decoded using belief propagation (BP) iterative decoding [15]. Secondly, their performance can often be theoretically analysed in the infinite-blocklength limit using the *density evolution* approach [16]. Finally, using the density evolution approach, or through simulations, one can demonstrate that these codes are good codes, in the sense that in the infinite-blocklength limit, BP decoding will perfectly decode all message blocks that have a noise level below some threshold level, and that threshold level is often not too far from the Shannon limit.

In recent years, a favoured way to design new codes has thus been to optimize codes for the infinite-blocklength limit using density evolution, and to hope that a scaled-down version would still be a good code [7, 9, 10, 14]. The problem with this approach is that for $N < 10^4$ at least, we are still noticeably far from the infinite-blocklength limit. In particular, simulations will find many decoding failures at noise levels far below the threshold level predicted by infinite-blocklength calculations. Furthermore, there will not necessarily even exist a way to scale down the codes derived from the density evolution approach. For example, the best known irregular LDPC codes at a given rate (in the $N \to \infty$ limit) will often have variable nodes that should participate in hundreds or even thousands of parity checks [10], which obviously makes no sense if the overall number of parity checks is much less than that. When one wants to make real codes of finite blocklength, one is therefore often forced to choose a code which is sub-optimal in the infinite-blocklength limit, with no theoretical guidance.

Our goal, which is at least partially achieved by the renormalization group approach described here, has therefore been to develop an assessment algorithm more powerful than the ordinary density evolution approach, which will predict, at least approximately, the BP decoding failure rate as a function of the noise level for a specific code of finite blocklength. It is important to realize that for finite blocklengths, one does not expect perfect decoding below any particular threshold noise level, so that to evaluate a code, one now needs a whole performance curve rather than a single number (the critical noise threshold) as might be computed in the density evolution approach.

Di *et al* [17] recently developed an alternative approach to analyse the finite-length performance of LDPC codes on the binary erasure channel (BEC). One comparative advantage of our approach is that their analysis only applies to an *ensemble* of LDPC codes defined by a certain probabilistic measure, while our analysis can be applied to a single code defined by a particular parity check matrix. On the other hand, our approach gives results that are only approximate, while their approach gives exact results. Another recent approach using 'projection algebra' [18] also gives exact or nearly exact results, and it applies to single codes, but it has the drawback of being computationally costly compared to the other techniques.

The outline of the rest of this paper is as follows. In the next section, we review the density evolution approach for the BEC, where it is particularly simple. We pay particular attention to codes defined on trees, for which the density evolution approach becomes exact. Section 3 is the heart of the paper, where we introduce and explain our renormalization group (RG) approach. We show how it recovers exact answers for codes defined on trees, and gives a procedure, which can be made increasingly accurate at the cost of more computational power to handle codes defined on graphs with cycles. We present some numerical results comparing our RG calculations with simulations of realistic finite-blocklength codes in section 4.

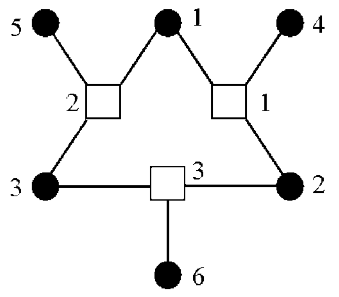

Figure 1. Tanner graph for a simple error-correcting code.

In section 5, we explain how to extend the RG approach to the additive white Gaussian noise (AWGN) channel. In section 6, we describe some remaining open problems.

2. The density evolution approach for the binary erasure channel

The density evolution approach is analytically very simple for the BEC. [6, 19] Since this approach is important background for our own RG approach, we will review it in this section.

2.1. Parity check codes

We will begin by describing linear block binary codes, which can be represented using an ordinary parity check matrix. In a parity check matrix A, the columns correspond to transmitted variable bits, while the rows define linear constraints between the variable bits. More specifically, the matrix A defines a set of valid vectors (codewords) z, such that each component of z is 0 or 1, and

$$Az = 0 \tag{1}$$

where we assume all multiplication and addition is modulo 2.

If a parity check matrix has N columns and $N - k$ rows it will represent a code of blocklength N and rate k/N (unless some of the rows are not linearly independent, in which case some of the parity checks are redundant, and the code will actually be of higher rate).

For each parity check matrix, there is a corresponding Tanner graph representation [20]. A Tanner graph is a bipartite graph with two kinds of nodes: variable nodes (which we denote by circles) and check nodes (denoted by squares). In a Tanner graph, each check node is connected to the variable nodes that represent the bits involved in that check. For example, the parity check matrix

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

corresponds to the Tanner graph shown in figure 1.

Codes represented by parity check matrices are 'linear', which means that all the codewords are linear combinations of other codewords. There will be 2^k codewords, each of length N; e.g., for the example given above, the codewords are `000000`, `001011`, `010110`, `011101`, `100110`, `101101`, `110011`, `111000`. Because of the linearity property, we may use any of the codewords as a representative; throughout this paper, we will always assume that the all-zeros codeword is transmitted.

2.2. Belief propagation decoding in the BEC

The BEC is a binary input channel with three output symbols: a `0`, a `1` and an erasure, which can be represented by a question mark `?`. The input symbol will pass through the channel as an erasure with probability x and will be received correctly with probability $1 - x$. It is important to note that the BEC never flips bits from `0` to `1` or vice versa. If we assume that the all-zeros codeword is transmitted, all received words will thus consist entirely of zeros and erasures.

We will assume that the receiver decodes using a belief propagation decoder with discrete messages. A message m_{ia} will be sent from each variable node i to each check a that it participates in, with the message representing information about the state of the variable node. In general, the message can be in one of the three states: `1`, `0` or `?`, but since we assume that the all-zeros state is always transmitted, we can ignore the possibility that m_{ia} has value `1`.

Similarly, there will be a message m_{ai} sent from each check node a to all the variable nodes i that participate in that check. These messages should be interpreted as directives from the check to the variable node about what state it should be in, based on the states of the other variable nodes participating in the check. The check-to-bit messages can again in principle take on the values `0`, `1` or `?`, but again only the two messages `0` and `?` will be relevant when the all-zeros codeword is transmitted.

In the BP decoding algorithm for the BEC, a message m_{ia} from a variable node to a check node will be equal to a non-erasure received message (because such messages are always correct in the BEC), or to an erasure if all incoming messages are erasures. A message m_{ai} from a check node a to a variable node i will be an erasure if any incoming message from another node participating in the check is an erasure, otherwise it will take on the value of the binary sum of all incoming messages from other nodes participating in the check.

The BP decoding algorithm is an iterative algorithm. One should initialize the messages so that all variable nodes that are not erased send out messages equal to the corresponding received bit, and all other messages are initially erasures. Iterating the BP message equations, one will eventually always converge to stationary messages (convergence of BP decoding algorithms is guaranteed for the particularly simple BEC, but not for other channels). The final decoded value of any erased variable node is just the value of any non-erasure message coming into that node, unless there is no incoming non-erasure message, in which case the BP decoding algorithm gives up and fails to decode that particular variable node.

2.3. Density evolution

We now consider the average of BP decoding over many blocks. Associated with each message m_{ia}, we introduce a real number p_{ia} which represents the probability that the message m_{ia} is an erasure. Similarly, we associate with each message m_{ai} a real number q_{ai} which represents the probability that the message m_{ai} is an erasure.

In the density evolution approach, we compute the probabilities p_{ia} and q_{ai} in a way that is exact as long as the Tanner graph representing the code has no cycles. We take

$$p_{ia} = x \prod_{b \in N(i) \backslash a} q_{bi} \tag{3}$$

where $b \in N(i)\backslash a$ represents all check nodes that neighbour variable node i except for check node a. This equation can be derived from the fact that for a message m_{ia} to be an erasure, the variable node i must be erased in transmission, and all incoming messages from other checks must be erasures as well. Of course, if the incoming messages were correlated, this equation would not be correct, but on a Tanner graph with no cycles, each incoming message is independent of the others.

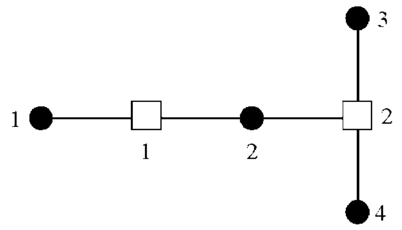

Figure 2. Tanner graph corresponding to the parity check matrix given in equation (6).

Similarly, we find that

$$q_{ai} = 1 - \prod_{j \in N(a) \setminus i} (1 - p_{ja}) \tag{4}$$

which can be derived (again assuming incoming messages are uncorrelated) from the fact that a message m_{ai} will only be in a `0` or `1` state if all incoming messages are in a `0` or `1` state.

The density evolution equations (3) and (4) can be solved by iteration. A good initialization is $p_{ia} = x$ for all messages from variable nodes to check nodes and $q_{ai} = 0$ for all messages from check nodes to variable nodes, as long as one begins the iteration with the q_{ai} messages. The BEC density evolution equations should ultimately converge (this can be guaranteed for codes defined on graphs without cycles). One can finally compute b_i, which is the probability of a failure to decode at variable node i, from the formula

$$b_i = x \prod_{a \in N(i)} q_{ai}. \tag{5}$$

2.4. Exact solution of a small code

As mentioned, the density evolution equations (3), (4) and (5) should be exact when the code has a Tanner graph representation without cycles. Let us work through a small example, to see how this works. We consider the code with parity check matrix

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \tag{6}$$

and a corresponding Tanner graph shown in figure 2.

This code has four codewords: `0000`, `0011`, `1101` and `1110`. If the `0000` message is transmitted, there will be 16 possible received messages: `0000`,`000?`,`00?0`,`00??`,`0?00` and so on. The probability of receiving a particular message with n_e erasures is $x^{n_e}(1-x)^{(4-n_e)}$. Messages might be partially or completely decoded by a BP decoder; for example the received message `?00?` will be fully decoded to `0000`, but the message `0???` will only be partially decoded to `00??`, because there is not enough information to determine whether the transmitted codeword was actually `0000` or `0011`.

We can easily compute the probability that a given bit will remain an erasure after BP decoding by summing over the 16 possible received messages weighted by their probabilities. For example, the first bit will only be decoded as an erasure if one of the following messages are received: `???0`,`??0?` or `????`, so the total probability that the first bit will not be decoded is $2x^3(1-x)+x^4 = 2x^3 - x^4$. If we focus on the last bit instead, we find that it will be decoded unless one of the following messages is sent: `00??`,`0???`,`?0??`,`??0?` or `????`, so the overall probability that the fourth bit is not decoded will be $x^2(1-x)^2+3x^3(1-x)+x^4 = x^2+x^3-x^4$.

In the density evolution approach, we need to solve equations for the following variables: $p_{11}, p_{21}, p_{22}, p_{32}, p_{42}, q_{11}, q_{12}, q_{22}, q_{23}, q_{24}, b_1, b_2, b_3$ and b_4. The equations are

$$p_{11} = x \tag{7a}$$

$$p_{21} = xq_{22} \tag{7b}$$

$$p_{22} = xq_{12} \tag{7c}$$

$$p_{32} = x \tag{7d}$$

$$p_{42} = x \tag{7e}$$

$$q_{11} = p_{21} \tag{7f}$$

$$q_{12} = p_{11} \tag{7g}$$

$$q_{22} = 1 - (1 - p_{32})(1 - p_{42}) \tag{7h}$$

$$q_{23} = 1 - (1 - p_{22})(1 - p_{42}) \tag{7i}$$

$$q_{24} = 1 - (1 - p_{22})(1 - p_{32}) \tag{7j}$$

and

$$b_1 = xq_{11} \tag{8a}$$

$$b_2 = xq_{12}q_{22} \tag{8b}$$

$$b_3 = xq_{23} \tag{8c}$$

$$b_4 = xq_{24}. \tag{8d}$$

Solving these equations, we find

$$p_{11} = x \tag{9a}$$

$$p_{21} = 2x^2 - x^3 \tag{9b}$$

$$p_{22} = x^2 \tag{9c}$$

$$p_{32} = x \tag{9d}$$

$$p_{42} = x \tag{9e}$$

$$q_{11} = 2x^2 - x^3 \tag{9f}$$

$$q_{12} = x \tag{9g}$$

$$q_{22} = 2x - x^2 \tag{9h}$$

$$q_{23} = x + x^2 - x^3 \tag{9i}$$

$$q_{24} = x + x^2 - x^3 \tag{9j}$$

and

$$b_1 = 2x^3 - x^4 \tag{10a}$$

$$b_2 = 2x^3 - x^4 \tag{10b}$$

$$b_3 = x^2 + x^3 - x^4 \tag{10c}$$

$$b_4 = x^2 + x^3 - x^4. \tag{10d}$$

Examining the results for b_1 and b_4, we see that the density evolution solution agrees exactly with the direct approach for this code.

2.5. The large blocklength limit

If we assume that all local neighbourhoods look identical, we can simplify the density evolution equations. For example, if each variable node belongs to d_v parity checks, and each check node is attached to d_c variable nodes, then we can take all the p_{ia} equal to the same value p, all the q_{ai} equal to the same value q, and all b_i equal to the same value b. We then find

$$p = xq^{d_v-1} \tag{11}$$

$$q = 1 - (1-p)^{d_c-1} \tag{12}$$

and

$$b = xq^{d_v} \tag{13}$$

which are the density evolution equations for (d_v, d_c) *regular Gallager codes*, valid in the $N \to \infty$ limit. A regular Gallager code [3] is a code defined by a sparse random parity check matrix characterized by the restriction that each row has exactly d_c 1's in it, and each column contains exactly d_v 1's. The intuitive reason that these equations are valid in the infinite-blocklength limit is that as $N \to \infty$, the size of typical cycles in the Tanner graph of a regular Gallager code will also go to infinity, so all incoming messages to a node will be independent, and a regular Gallager code will behave like a code defined on a graph without cycles.

If we solve equations (11) and (12) for specific values of d_v and d_c, we find that below a critical erasure threshold x_c, the solution is $p = q = b = 0$, which means that decoding is perfect. Above x_c, b will have a non-zero solution, which corresponds to decoding failures. x_c is easy to determine numerically. For example, if $d_v = 3$ and $d_c = 5$, then $x_c \approx 0.517\,57$.

These density evolution calculations can be generalized to irregular Gallager codes [6], or other codes such as irregular repeat-accumulate codes [14] which have a finite number of different classes of nodes with different neighbourhoods. In this generalization, one derives a system of equations, typically with one equation for the messages leaving each class of node. By solving the system of equations, one can again find a critical threshold x_c, below which decoding is perfect. Such codes can thus be optimized in the $N \to \infty$ limit by finding the code that has maximal noise threshold x_c. Simulations of such codes with very large blocklengths agree quite well with the density evolution predictions.

Unfortunately, the density evolution approach is useless, or at least misleading, for codes with finite blocklength. One might think that one could solve equations (3) and (4) for any finite code, and hope that ignoring the presence of cycles is not too important a mistake. This does not work out, as one can simply see by considering regular Gallager codes. Equations (3), (4) and (5) for a finite-blocklength regular Gallager code will have exactly the same solutions as one would find in the infinite-blocklength limit, so one would not predict *any* finite-size effects. Simulations, on the other hand, show that the real performance of finite-blocklength regular Gallager codes is considerably different (and worse) than that predicted by such a naive approach.

3. The renormalization group approach

3.1. Intuition

The basic idea behind the 'real-space' renormalization group approach from physics [21] is very similar to the idea behind recursion from computer science. To evaluate the performance of a large but finite code, we try to replace the code with a slightly smaller code with the same performance. In particular, at each step in the process, we keep a Tanner graph and a set of p_{ia} and q_{ai} variables just as in the density evolution approach. We will call the combination of a Tanner graph and the p and q variables a 'decorated Tanner graph'. The heart of the RG approach is the RG transformation, by which we eliminate ('renormalize away') one node in the decorated Tanner graph, and adjust the remaining values of the p and q messages so that the new code has a decoding failure rate as close as possible to the old code. With each renormalization step, the decorated Tanner graph representing our code will thus shrink by one node, until it is finally small enough that the performance of the code can be computed exactly in an efficient way. The RG approach is sometimes referred to as 'decimation', and has also been used in the graphical models literature, notably to inference and learning in Boltzmann machines [22, 23].

We will explain all the details in the following subsections, but in general, the RG algorithm will work as follows:

(i) Choose a 'target' variable node i for which we want to compute the decoding failure rate b_i.

(ii) While the number of nodes remaining in the graph is greater than the number that one can comfortably handle exactly, repeatedly renormalize away nodes from the graph according to the following procedure:

  (a) Mark the 'distance' of every node from the 'target' node. The distance between two nodes is the minimal number of nodes that one needs to pass through on the graph to travel from one node to the other.

  (b) As long as there are any 'leaf' (a *leaf* is a node which is only connected to one other node in the graph) check or variable nodes, renormalize them away. The order in which they are renormalized away will not matter, but for concreteness, we will renormalize those furthest from the 'target' node first, breaking ties randomly.

  (c) Otherwise, choose a single variable node from among those furthest from the target node, that has the fewest neighbouring check nodes, and renormalize it away.

(iii) Compute b_i for the remaining graph exactly.

It should be clearly understood that the RG approach is approximate, and that there exists considerable freedom in the implementation. Different choices made in the implementation will lead to slightly different results. One can hope to deal with this problem by constructing a series of better RG approximations that should eventually converge to the exact answer. We shall see how this works out in section 3.4.

3.2. The RG transformation for Tanner graphs with no cycles

First we consider loop-free Tanner graphs, and write down the RG transformations that are sufficient to give exact results for such codes. In later subsections, we will extend the RG transformations in order to obtain good approximate results for Tanner graphs with cycles.

We will always initialize our decorated Tanner graph such that all $b_i = x$, $p_{ia} = x$ and all $q_{ai} = 0$. Imagine that we are interested in the decoding failure rate b_i at a specific node i. Our procedure will be to obtain b_i by repeatedly renormalizing away nodes, other than the variable node i itself, that are 'leaves' of the decorated Tanner graph.

The first possibility that we need to concern ourselves with is when we renormalize away a 'leaf' variable node i that is connected to a single check node a. Clearly, when the node i vanishes, p_{ia} and q_{ai} will also be discarded. We need to renormalize all the q_{aj} variables leading out of the check a to other nodes j. Our formula will be

$$q_{aj} \leftarrow 1 - (1 - q_{aj})(1 - p_{ia}) \tag{14}$$

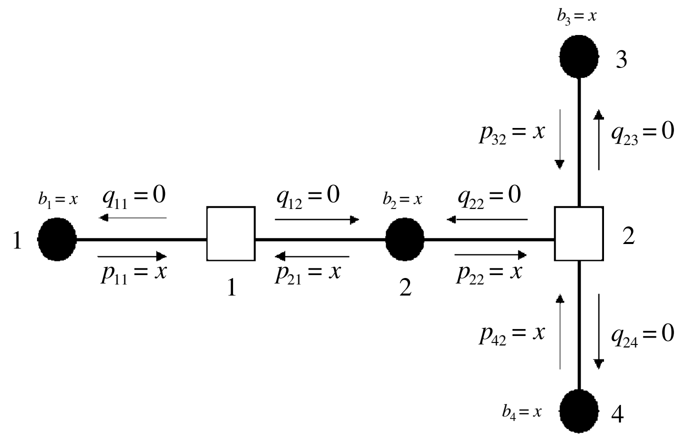

Figure 3. Decorated Tanner graph.

where the left arrow indicates that we replace the old value of q_{aj} with this new value. Note that each renormalization of q_{aj} will increase its value.

When we renormalize away a 'leaf' check node a that is only connected to a single variable node i, we need to adjust the values of all the p_{ib} variables leading to other checks b that node i is attached to. The renormalization group transformation will be

$$p_{ib} \leftarrow p_{ib} q_{ai}. \tag{15}$$

Note that each renormalization of p_{ib} will decrease its value. At the same time, we should also renormalize the b_i as follows:

$$b_i \leftarrow b_i q_{ai}. \tag{16}$$

When only the 'target' node i remains, we can just read off the current value of b_i and that will serve as the RG prediction.

One way to understand these RG transformations is to see that they naturally arise when we try to rewrite the corresponding density evolution equations as a series of transformations. For example, the transformation (15) reproduces the density evolution equation $p_{ib} = x \prod_{a \in N(i) \backslash b} q_{ai}$ as a series of transformations that handle one check node at a time, after initializing $p_{ib} = x$.

3.3. A small example

The RG procedure might be easier to understand if we work through a small example. Recall the code defined by the parity check matrix

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}. \tag{17}$$

Let us imagine that we would like to compute the decoding failure rate at the second variable node b_2. We initialize $p_{11} = p_{21} = p_{22} = p_{32} = p_{42} = x, q_{11} = q_{12} = q_{22} = q_{23} = q_{24} = 0$ and $b_2 = 0$. In figure 3, we show the decorated Tanner graph for this code. All of the variable nodes other than variable node 2 are leaf nodes, so we can renormalize any of them away. According to our general algorithm, we should renormalize away the one furthest from node 2, breaking ties randomly. Imagine we choose variable node 4. Then we discard p_{42} and q_{24} and obtain new values $q_{22} = x$ and $q_{23} = x$ using equation (14). The

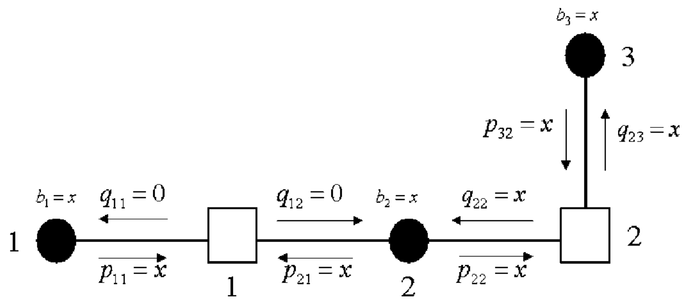

Figure 4. Decorated Tanner graph after renormalizing variable node 4.

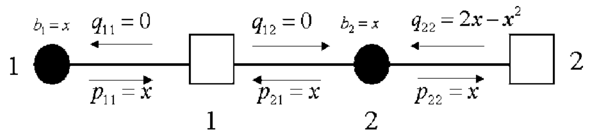

Figure 5. Decorated Tanner graph after renormalizing variable nodes 3 and 4.

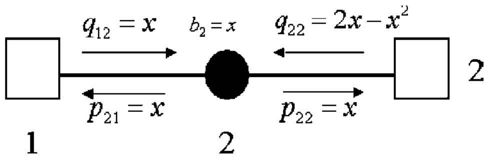

Figure 6. Decorated Tanner graph after renormalizing variable nodes 1, 3 and 4.

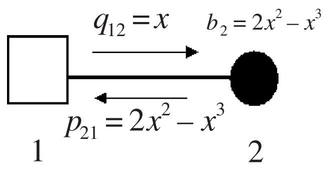

Figure 7. Decorated Tanner graph after renormalizing variable nodes 1, 3 and 4, and check node 2.

new decorated Tanner graph is shown in figure 4. We might next renormalize away variable node 3. We discard p_{32} and q_{23} and renormalize q_{22} to the value $1 - (1 - x)^2 = 2x - x^2$. The new decorated Tanner graph is shown in figure 5. Next we renormalize away variable node 1. We discard p_{11} and q_{11} and obtain the new renormalized value $q_{12} = x$; the Tanner graph is now shown in figure 6. Next we renormalize away check node 2. We can discard p_{22} and q_{22} and obtain $p_{21} = b_2 = 2x^2 - x^3$ (shown in figure 7). Finally we renormalize away check node 1. We are left with only a single node (our original node 2) and b_2 gets renormalized to its correct value $b_2 = 2x^3 - x^4$.

This example makes it clear why the RG approach is exact for a code defined on a graph without cycles: the RG transformations essentially reconstruct the density evolution equations, and we know that density evolution is exact for such codes. As we shall see, the advantage of the RG approach is that it still gives a reasonable approximation for codes defined on graphs with cycles.

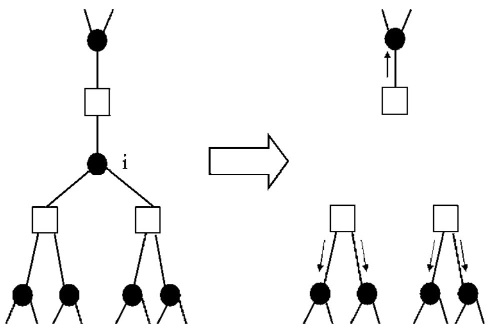

Figure 8. Removing the non-leaf node i. The arrows indicate the q variables that will be renormalized as a result.

3.4. The RG approach for a graph with cycles

For a code defined on a graph that has cycles, we will eventually have to renormalize away a variable node i that is not a 'leaf' node. (Note that we could also renormalize away non-leaf check nodes by defining the appropriate RG transformations, but we will choose instead to always renormalize away non-leaf variable nodes.) To do this, we first collect all the check nodes a, b, etc, that node i is attached to. Obviously, we will discard $q_{ai}, q_{bi}, p_{ia}, p_{ib}$, etc. For any given check node attached to i (say check node a), we must also collect all the other variable nodes j attached to a, and renormalize the values of q_{aj}. In figure 8, we illustrate the process of removing a non-leaf node.

The renormalization of the q_{aj} variable can be done to varying degrees of accuracy. The simplest approach would be to use equation (14) directly. The problem with this approach is that the value of p_{ia} which is used will always be an overestimate. Recall that p_{ia} decreases with every renormalization. Since we are renormalizing away the ith node before it has become a leaf node, p_{ia} has not yet been fully renormalized, and is thus overestimated.

Instead of using p_{ia} directly, we could use the value that it would have after we renormalized away all the checks connected to it; that is we could replace p_{ia} in equation (14) with an effective p_{ia}^{eff} given by

$$p_{ia}^{\mathrm{eff}} = p_{ia} \prod_{b \in N(i) \backslash a} q_{bi}. \tag{18}$$

On the other hand, we know that the values of the q_{bi} are *underestimates* since they have not yet been fully renormalized either, so p_{ia}^{eff} as written above would also be an underestimate. We could attempt to correct this mistake by going further another level: before we estimate a p_{ia}^{eff}, we first re-estimate the q_{bi} which feed into it. Thus, we replace the p_{ia} in equation (14) with an effective p_{ia}^{eff} given by

$$p_{ia}^{\mathrm{eff}} = p_{ia} \prod_{b \in N(i) \backslash a} q_{bi}^{\mathrm{eff}} \tag{19}$$

where q_{bi}^{eff} is in turn given by

$$q_{bi}^{\mathrm{eff}} = 1 - (1 - q_{bi}) \prod_{k \in N(b) \backslash i} (1 - p_{kb}). \tag{20}$$

Putting all these together, we finally get the RG transformation

$$q_{aj} \leftarrow 1 - (1 - q_{aj}) \left(1 - p_{ia} \prod_{b \in N(i) \backslash a} \left[1 - (1 - q_{bi}) \prod_{k \in N(b) \backslash i} (1 - p_{kb}) \right] \right). \tag{21}$$

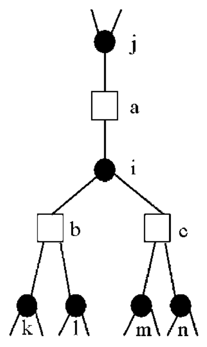

Figure 9. Node *i* sees a local tree-like structure.

The RG transformation (21) is worth explaining in more detail. In figure 9, we illustrate the equation for a case where variable node i is attached to three check nodes a, b and c, and check node a is in turn attached to a variable node j. Check nodes b and c in turn are connected to their own variable nodes labelled k, l, m and n. We would like to know the new probability q_{aj} that check node a will send variable node j an erasure message, taking into account the information that flows through node i. We already have some previous accumulated probability q_{aj} that check node a sends variable node j an erasure message (because of other nodes previously attached to a that have already been renormalized). The new probability of an erasure message can be figured out from a logical argument: 'm_{aj} will be an erasure it was already, *or* if m_{ia} is an erasure *and* (m_{bi} *or* m_{kb} *or* m_{lb} are erasures) *and* (m_{ci} *or* m_{mc} *or* m_{nc} are erasures)'. Converting such a logical argument into an equation for probabilities is straightforward: when we see 'm_1 *and* m_2' for two statistically independent messages in a logical argument, it translates to $(p_1 p_2)$ for the corresponding probabilities, while 'm_1 *or* m_2' translates to $(1-(1-p_1)(1-p_2))$. Converting our full logical argument for figure 9 into an equation for probabilities, we thus recover an example of the RG transformation (21).

In principle, we should always take our RG transformation for q_{aj} to correspond to the logic of the local neighbourhood around the node i that we are removing. In fact, the RG transformation given in equation (21) is only appropriate if the local neighbourhood of node i is tree-like, and must be corrected if there are short cycles in the local neighbourhood. For example, in figure 10, we illustrate a case where a variable node k is attached to two check nodes b and c which are each attached to the node i that we plan to remove. First consider the renormalization of q_{aj}. Note that before check nodes b or c are renormalized, the probabilities p_{kb} and p_{kc} that variable node k sends out an erasure must be identical, because all renormalizations of p_{kb} and p_{kc} happen in tandem. Our logic argument for whether check node a will send variable node j an erasure message would thus be: 'm_{aj} will be an erasure if it was already, *or* if (m_{ia} is an erasure) *and* ((m_{kb} is an erasure) *or* (m_{bi} *and* m_{ci} are erasures))'. (We have used the fact that at this stage in the renormalization process, if m_{kb} is an erasure, m_{kc} must be as well.) Converting our logic argument into an RG transformation, we get

$$q_{aj} \leftarrow 1-(1-q_{aj})(1-p_{ia}(1-(1-p_{kb})(1-q_{bi}q_{ci}))). \qquad (22)$$

The appropriate renormalizations of q_{bk} and q_{ck} are more complicated: the messages m_{bk} and m_{ck} are correlated because of node i, and we must keep track of that correlation after node i is removed. We have tried several relatively ad hoc rules for assigning renormalized values to q that all arrive at the same node (such as renormalizing the product $q_{bk}q_{ck}$ as a whole), but found the results to be unsatisfactory because they depended sensitively on the details of the rules. In general, to correctly account for the correlations caused by such short cycles, we

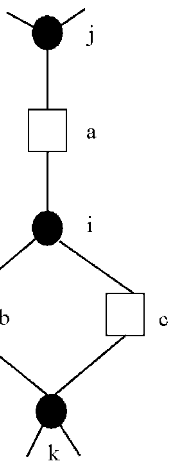

Figure 10. Node i sees a local neighbourhood with cycles.

need to introduce additional variables beyond the q and p variables that we use here. Pursuing such ideas eventually led us to the 'projection algebra' (PA) approach described elsewhere [18], which is more accurate than the RG approach described here. Unfortunately, the PA approach is also much more computationally expensive. In our view, an important goal for future research is to find a well-justified approach to deal with short cycles within the RG framework. In this paper, however, we will restrict our examples to codes where the local structure is always tree-like and such short cycles do not exist.

It is natural to hope that the procedure we are describing for renormalizing a non-leaf variable node i will become increasingly accurate as one increases the size of the neighbourhood around the node i that is treated correctly. Naturally, as we increase the size of the neighbourhood, we must pay for the increased accuracy with greater computation. We will use the following terminology: if, when renormalizing the node i, we use the values of p_{ia} directly, we will say that the resulting RG transformations have a 'depth' of 1. If we first adjust the values of p_{ia} by considering all the check nodes a attached to i and all the variable nodes k attached to those check nodes, we will say the resulting RG transformations (e.g. those described above) have a depth of 2. If we go one step further and also consider the check nodes attached to the variable nodes k and the variable nodes attached to those check nodes, we say the RG transformations have a depth of 3, and so on.

3.5. Finishing the RG computation exactly

In the RG approach, we can always renormalize nodes away until we are left with just our 'target' node i, and then read off the decoding failure rate for that node b_i. On the other hand, after we have renormalized away enough nodes, we could just as well finish the computation exactly. We note, to avoid any confusion, that when we refer to an 'exact' computation, we are referring to an exact computation of the performance of a BP decoding algorithm, rather than that of a maximum-likelihood decoding algorithm.

For the purposes of describing the exact computation, we assume that we are given a Tanner graph of N nodes, and associated with each node i is an erasure probability x_i. (This is a little different from the decorated Tanner graph we are used to dealing with, but we shall show how to convert a Tanner graph into such a form.) To exactly compute the decoding

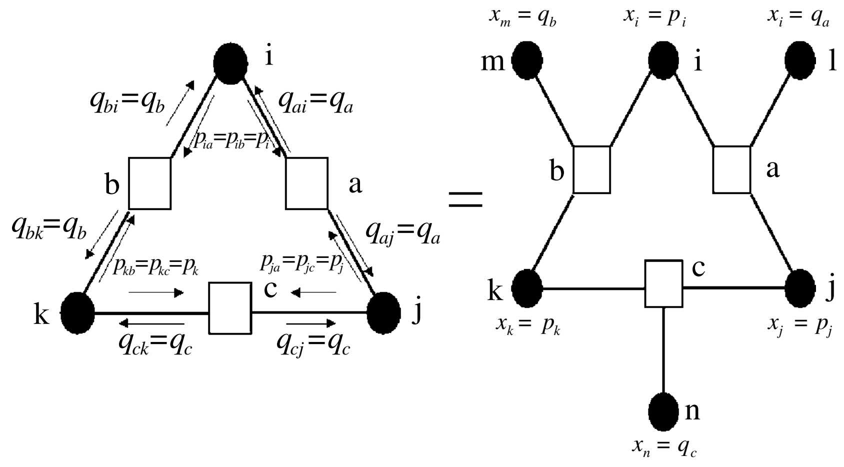

Figure 11. Expanding a decorated Tanner graph into an equivalent Tanner graph with erasure probabilities.

failure rate of a given node i, we generate all 2^N possible received message blocks (ranging from the correct all-zeros message all the way to the all-erasures message), and decode each of them using a BP decoder. Each message block has a probability

$$p = \prod x_i \prod (1 - x_j) \tag{23}$$

where the first product is over all nodes that are erased and the second product is over all nodes that are not erased. We simply compute b_i by taking the weighted average over all received messages of the probability that node i decodes, using the BP decoder, to an erasure. Of course, the complexity of the exact calculation is $O(2^N)$, so we are restricted to small N, but nevertheless one can gain some accuracy by switching to an exact calculation after one has renormalized away enough nodes.

The one subtlety in the exact final calculation is that one needs a Tanner graph and the associated erasure probabilities at each node, but in the RG approach, we manipulate decorated Tanner graphs. Fortunately, it is easy to convert a decorated Tanner graph into the appropriate form. Note that at each step of the RG approach, all the probabilities q_{ai} leading out of the check node a must be equal (we say $q_{ai} = q_a$) and all the probabilities p_{ia} leading out of the variable node i will be equal (we say $p_{ia} = p_i$). We can set all the q_a probabilities equal to zero if we expand the graph by adding a new variable node k to node a with $p_{ka} = q_a$. When we are left with a decorated Tanner graph such that all q probabilities are zero, and all p_{ia} probabilities coming out of each variable node are equal to p_i, we may interpret the p_i as the erasure probabilities of the variable nodes. In figure 11, we give an example of expanding a decorated Tanner graph into an equivalent Tanner graph with erasure probabilities.

3.6. Extension to generalized parity check matrices

Many of the best modern codes, such as turbo codes, Kanter–Saad codes and repeat-accumulate codes, are easily represented in terms of *generalized* parity check matrices [2]. In a generalized parity check matrix, additional columns are added to a parity check matrix which represent 'hidden nodes'—state variables which are not transmitted. A good notation for the state variables is a horizontal line above the corresponding columns. For example, we would write

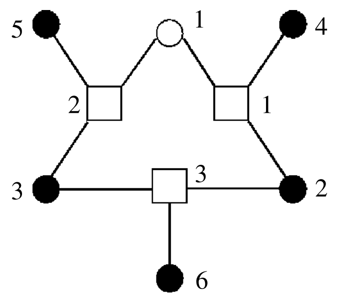

Figure 12. A Wiberg graph.

$$A = \begin{pmatrix} \bar{1} & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{24}$$

to indicate a code where the first variable node was a hidden node. To indicate that a variable node is a hidden node in our graphical model, we use an open circle rather than a filled-in circle. Such a graph, which generalizes Tanner graphs, is called a 'Wiberg graph' [24, 25]. In figure 12, we give the Wiberg graph corresponding to the code defined by the generalized parity check matrix (24).

The generalization of our RG procedure to handle Wiberg graphs is very straightforward. We initialize the probabilities p_{ia} coming out of a hidden node at 1, instead of at the erasure rate x as we do for ordinary transmitted variable nodes. This reflects the fact that hidden nodes are automatically erased, while ordinary variable nodes are only erased with probability x.

4. Comparison with numerical simulations for the BEC

We now present a comparison of the predictions of our RG approach with numerical simulations. We first used a parity check matrix corresponding to a (3, 5) regular Gallager code with $N = 60$ and $k = 24$. That is, each of the 36 rows in the parity check matrix had five entries that were ones (the rest were zeros), and each of the 60 columns had three entries which were ones. There were no hidden nodes.

We also took care to ensure that no two parity checks shared more than one variable node. That meant that there were no cycles of length 4, so we could use the RG transformation (21) (an RG transformation of 'depth' 2) whenever we renormalized away a non-leaf variable node. We renormalized nodes away until we were left with seven nodes, and then finished the computation exactly.

We considered erasure rates x at intervals of 0.05 between $x = 0$ and $x = 1$. When we used the RG approximation, we averaged our decoding failure rates b_i over all 60 nodes i to get an overall bit error rate. Our simulations used the standard discrete BP decoding algorithm, and we ran a sufficient number of trials so as to obtain at least 200 block decoding failures at every erasure rate considered.

Our results are presented in figure 13, where we compare the simulation results with the predictions of our RG approach and the density evolution approach. As one can see, the agreement between the RG approach and simulations is relatively good, although at low erasure rates, the RG predictions for the BP decoding performance are too optimistic.

The density evolution prediction is precisely the same as it would be in the infinite-blocklength limit. Of course, nobody claims that the density evolution approach should

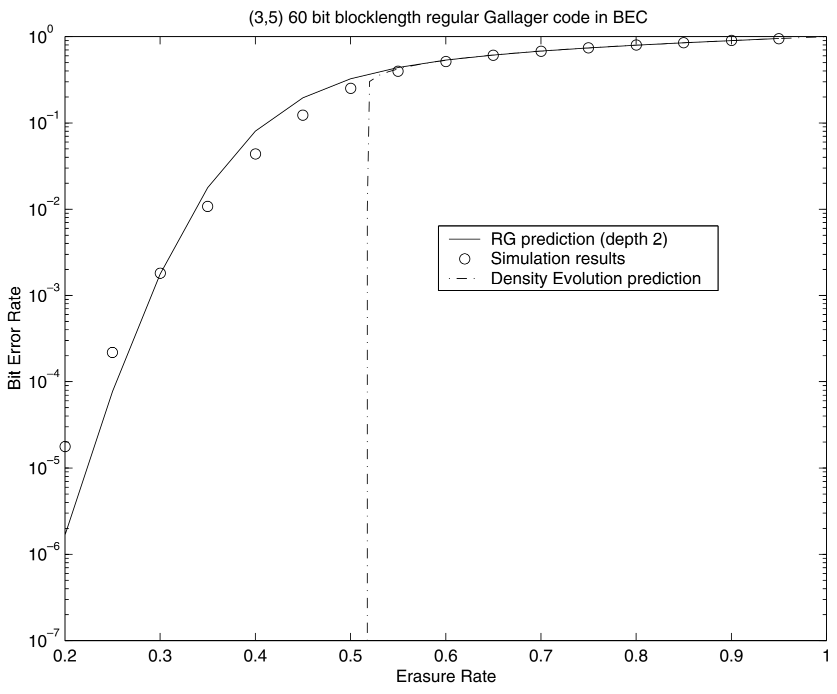

Figure 13. Simulation results compared with RG and density evolution predictions for a small rate 2/5 60 bit blocklength regular Gallager code. Error bars for the simulation results would be smaller than the size of the symbols used.

be taken seriously for blocklengths as low as 60, and figure 13 shows why: the density evolution prediction of a threshold-like behaviour is completely incorrect for small or medium blocklength regular Gallager codes.

We then constructed, by a somewhat random procedure, a particular irregular Gallager code of rate $2/5$ and blocklength $N = 100$. Each variable node belonged to between one and four parity checks, and each parity check involved between three and five variable nodes. No special effort was made to construct a particularly good error-correcting code, but we did ensure that the Tanner graph had no short cycles of length 4 or 6 (counting both variable and check nodes). This meant that all local neighbourhoods could be considered tree-like up to RG transformations of depth 3.

Our procedures were the same as described for the regular Gallager code except that we implemented RG transformations of depths 1, 2 and 3. Our numerical simulations again consisted of between 10^5 and 10^7 trials at each erasure rate.

In figure 14, we compare the simulation results with the prediction of our RG approach for the bit error rate averaged over all nodes. As one can see, the agreement is remarkably good, especially for the RG transformations of depth 3. The density evolution prediction spuriously shows a quasi-threshold behaviour around $x \approx 0.555$.

The irregular Gallager code has interesting variation in its bit error rates across the different bits of the code. In figure 15 we plot the predicted (using depth 3 RG transformations) and simulated bit error rates for every bit in the code at an erasure rate of $x = 0.55$. This plot demonstrates that the RG approach can in fact predict the bit-by-bit variation in the bit error rate at high erasure rates. Although the RG prediction is systematically slightly too high at this erasure rate, it captures the ordering of how easily the bits are decoded quite well. On the

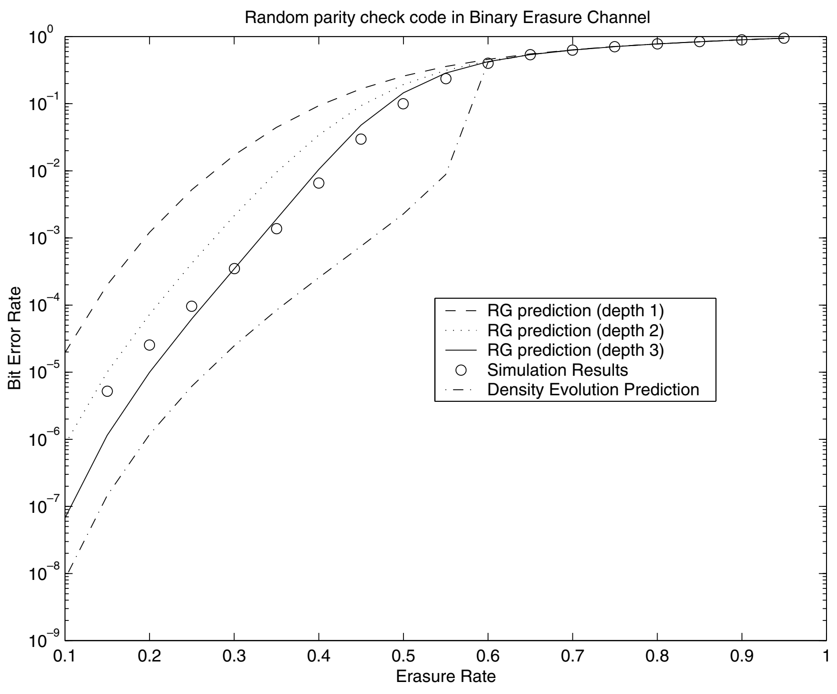

Figure 14. Simulation results compared with RG predictions using depths from 1 to 3 and density evolution predictions for a small rate 2/5 100 bit blocklength irregular Gallager code. Error bars for the simulation results would be smaller than the size of the symbols used.

other hand, we found that the bit-by-bit RG predictions at low erasure rates are not accurate. At low erasure rates, the bits that have the highest failure rates will be those that belong to the largest 'stopping sets' (see [17] and [18] for a detailed explanation) and the RG approach has no way of picking out these stopping sets.

5. Extension to the Gaussian noise channel

5.1. Background

In this section, we consider the extension of the RG approach to the additive white Gaussian noise (AWGN) channel. We will build on the Gaussian approximation to density evolution for the AWGN channel described by Chung *et al* [26], so we first describe that approximation.

In the AWGN channel, there are only two possible inputs, 0 and 1, but the output alphabet is the set of real numbers: if x is the input, then the output would be $y = (-1)^x + z$, where z is a Gaussian random variable with zero mean and variance σ^2. For each received bit i in the code, we can compute the log-likelihood ratio $m_i^0 = \ln(p(y_i|x_i = 0)/p(y_i|x_i = 1))$ which tells us the relative log-likelihood ratio that the transmitted bit i was a zero given the received real number is y_i.

We assume that we are again dealing with codes defined by generalized parity check matrices, that we always transmit the all-zeros codeword, and that the decoding algorithm is the sum–product belief propagation algorithm. In this decoding algorithm, we iteratively solve for real-valued messages: m_{ia} from variable nodes i to check nodes a; and m_{ai} from check nodes a to variable nodes i. The messages m_{ia} are log-likelihood ratios by which the

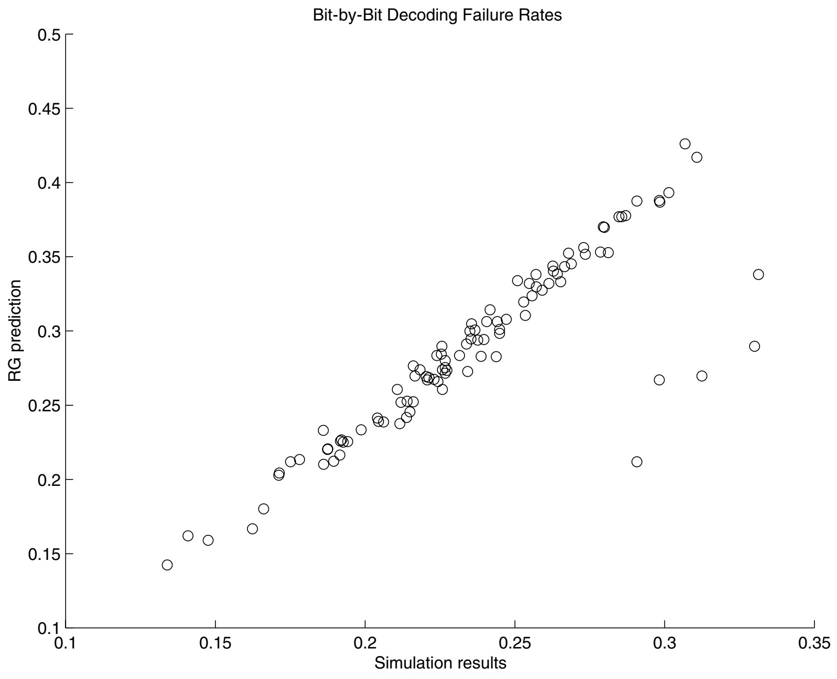

Figure 15. Bit-by-bit comparison of simulation results and RG predictions for a small rate 2/5 100 bit blocklength irregular Gallager code at an erasure rate of $x = 0.55$ in the BEC.

node i informs the node a of its probability of being a 0 or 1. For example, $m_{ia} \to \infty$ means that node i is certain it should be a 0, while $m_{ia} = 1$ means that variable node i is telling check node a that $\ln(p(x_i = 0)/p(x_i = 1)) = 1$. The messages m_{ai} are log-likelihood ratios which should be interpreted as information from the check node a to the variable node i about what state node i should be in.

In the sum–product algorithm, the messages are iteratively solved according to the update rules:

$$m_{ia} = \sum_{b \in N(i)\backslash a} m_{bi} + m_i^0 \tag{25}$$

(if i is a hidden node, the m_i^0 term is omitted) and

$$\tanh(m_{ai}/2) = \prod_{j \in N(a)\backslash i} \tanh(m_{ja}/2). \tag{26}$$

In the density evolution approach for the AWGN channel, one considers the probability distributions $p(m_{ia})$ and $p(m_{ai})$ for the messages where the probability distribution is an average over all possible received blocks. A distribution $f(x)$ is called *consistent* if $f(x) = f(-x)\,\mathrm{e}^x$ for all x [9]. Richardson and Urbanke [16] proved that the consistency condition will be preserved for the message probability distributions for all messages under sum–product decoding. If we approximate the probability distributions $p(m_{ia})$ and $p(m_{ai})$ as Gaussian distributions, the consistency condition means the means μ of these distributions will be related to the variances σ^2 by $\sigma^2 = 2\mu$. This means that we can characterize the message probability distributions by a single parameter: their mean.

Thus, by making the approximation that the message probability distributions are Gaussians, one can reduce the density evolution equations for the AWGN channel to self-consistent equations for the means v_{ia} of the probability distributions of messages from variable nodes i to check nodes a, and the means u_{ai} of the probability distributions of messages from check nodes a to variable nodes i. These equations are

$$v_{ia} = u_i^0 + \sum_{b \in N(i) \setminus a} u_{bi} \tag{27}$$

where u_i^0 is the mean value of m_i^0 (this term is omitted for hidden nodes), and

$$\phi(u_{ai}) = 1 - \prod_{j \in N(a) \setminus i} (1 - \phi(v_{ja})) \tag{28}$$

where $\phi(x)$ is a function defined by

$$\phi(x) \equiv 1 - \frac{1}{\sqrt{4\pi x}} \int_{-\infty}^{\infty} \tanh \frac{u}{2} \, \mathrm{e}^{-\frac{(u-x)^2}{4x}} \, \mathrm{d}u. \tag{29}$$

$\phi(x)$ can be approximated in a form that reproduces the correct limits as $x \to 0$ and $x \to \infty$ and is more convenient for numerical purposes. We choose

$$\phi(x) \approx \frac{\mathrm{e}^{-x/4}}{\sqrt{1 + \beta x}} \left[1 + (\sqrt{\beta \pi} - 1) \frac{\alpha x}{1 + \alpha x} \right]. \tag{30}$$

This form automatically has the correct leading behaviour as $x \to 0$ and $x \to \infty$ for any α and β. We fix α and β by matching the leading corrections in the two limits. We find $\alpha \approx 0.163\,489$ and $\beta \approx 0.634\,765$. This approximation to $\phi(x)$ is quite good for all values of x.

5.2. RG transformations for the AWGN channel

The density evolution equations (27) and (28) for the AWGN channel under the Gaussian approximation are analogues of the density evolution equations (4) and (3) for the BEC channel. Our RG procedure for the AWGN channel will be almost exactly the same as for the BEC channel; the main difference is that we need to change the RG transformations.

Just as before, we can construct a set of RG transformations which exactly reproduce the density evolution equations for a tree-like graph. We create a decorated Tanner/Wiberg graph for the code by keeping u_{ai} and v_{ia} variables between each pair of connected nodes. The u_{ai} variables are initialized to ∞, while the v_{ia} variables are initialized to u^0, unless the ith node is a hidden node, in which case the v_{ia} are initialized to zero. We also introduce the variables h_i (analogous to b_i in the BEC) which are initialized like the v_{ia} variables.

If we renormalize away a leaf check node a attached to a variable node i, we find the other check nodes b attached to i and apply the RG transformations

$$v_{ib} \leftarrow v_{ib} + u_{ai} \tag{31}$$

and

$$h_i \leftarrow h_i + u_{ai} \tag{32}$$

while if we renormalize away a leaf variable node i attached to a check node a, we find the other variable nodes j attached to a and apply the RG transformation

$$u_{aj} \leftarrow \phi^{-1}(1 - (1 - \phi(u_{aj}))(1 - \phi(v_{ia}))). \tag{33}$$

Note that with each renormalization of v_{ib}, the magnitude of v_{ib} will increase, while with each renormalization of u_{aj}, the magnitude of u_{aj} will decrease.

When we renormalize away a non-leaf variable node i which is attached to check nodes a, b, etc, we need to renormalize the variables like u_{aj}, where j is another variable node

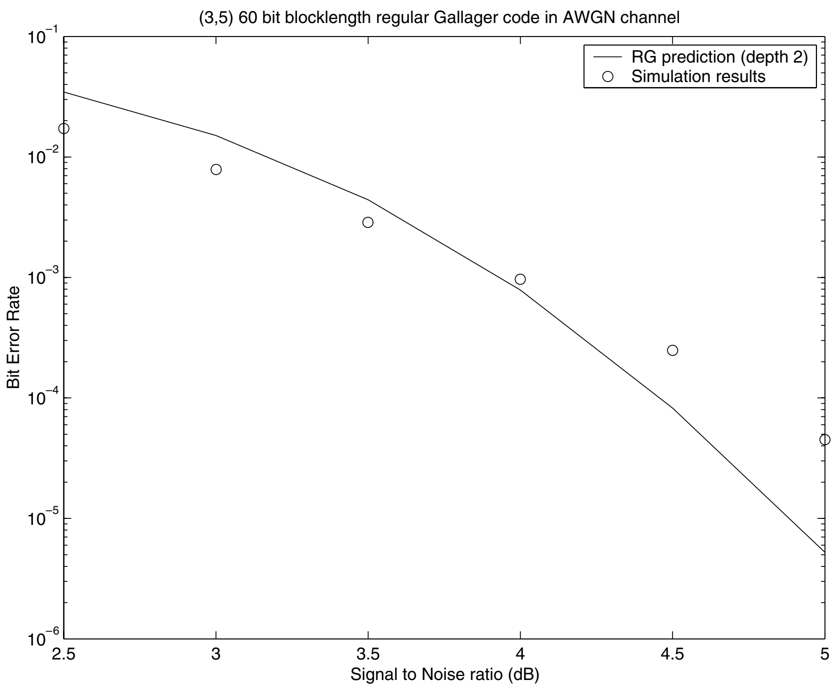

Figure 16. Simulation results compared with RG predictions for a rate 2/5 60 bit blocklength regular Gallager code on the AWGN channel.

attached to check node a. Just as for the BEC, we should consider a local neighbourhood of nodes around the node i. For example, if no variable nodes j share two check nodes with i (there are no local cycles of length 4) then we can use the depth 2 RG transformation

$$u_{aj} \leftarrow \phi^{-1}\left(1-(1-\phi(u_{aj}))\left(1-\phi\left(v_{ia}^{\text{eff}}\right)\right)\right) \tag{34}$$

where

$$v_{ia}^{\text{eff}} = v_{ia} + \sum_{b\in N(i)\backslash a} \phi^{-1}\left(1-(1-\phi(u_{bi}))\prod_{k\in N(b)\backslash i}(1-\phi(v_{kb}))\right). \tag{35}$$

The RG procedure proceeds as in the BEC case until the final computation of the bit error rate. For the AWGN channel, it will not normally be convenient to stop the RG procedure before renormalizing all the way down to the 'target' node, because it is not simple to do an exact computation even with just a few nodes in the code. When we have renormalized all but our target node i, we will be left with a final renormalized value of h_i. Our Gaussian approximation tells us that the probability distribution for the node i being decoded as a zero will be a Gaussian with mean h_i and variance $2h_i$. Decoding failures correspond to those parts of the probability distribution which are below zero. Thus, our theoretical prediction for the bit error rate at node i will be

$$b_i = \frac{1}{2\sqrt{\pi h_i}}\int_{-\infty}^{0} \mathrm{e}^{-\frac{(x-h_i)^2}{4h_i}}\,dx = \frac{1}{2}\mathrm{erfc}\left(\frac{\sqrt{h_i}}{2}\right). \tag{36}$$

5.3. Numerical results for the AWGN channel

We implemented the RG procedure under the Gaussian approximation for the AWGN channel and compared its predictions to numerical simulations using the same rate $2/5\, N = 60$ blocklength regular Gallager code that we previously considered in section 4. The results are shown in figure 16.

In this figure, we measured the signal-to-noise ratio in decibels, as is typical in the engineering literature, rather than in terms of u^0. To make the conversion, we first note that a very straightforward calculation shows that $u^0 = 2/\sigma^2$, where we recall that σ is the variance of the AWGN channel. We then note that when one communicates over the AWGN channel using a code of rate R, the conventional practice [5] is to describe the signal-to-noise ratio using $1/(2R\sigma^2)$ and to report this number in decibels as $-10\log_{10}(2R\sigma^2)$.

We note that the RG prediction obtains the correct general trend, in contrast to the sharp threshold that would be predicted using density evolution. On the other hand, it is clear that the RG prediction is too optimistic in the regime where the bit error rate becomes small, an effect that is similar to what happened for the BEC, though somewhat more prominent in this case.

6. Summary and outlook

We have shown that one can approximately analyse the average performance of a parity check code under BP decoding using an RG approach. The technique gives reasonably accurate results for both the BEC and the AWGN channel, but its detailed predictions could still be improved. To conclude, we just mention a few open problems for future work.

One obvious problem that we have already mentioned is to refine the methods used here so as to obtain more accurate predictions, particularly for codes with short cycles.

Another open problem is to try to develop a similar method that would give upper or lower bounds on the performance of the BP decoding algorithm. Upper bounds on the error rate would be particularly valuable, especially in the regime of very small error rates when it becomes difficult or impossible to run simulations.

Given that the density evolution method has been used as a guide to developing some of the best codes decoded using BP at long blocklengths, another obvious problem is to find a practical way to use the additional information provided by the RG technique to help design codes in intermediate blocklength regimes.

Acknowledgments

We thank Bill Freeman, Dave Forney, David MacKay, Michael Mitzenmacher and Erik Sudderth for helpful discussions. Jean-Philippe Bouchaud thanks Harvard University and Daniel Fisher for their hospitality during the period when this work was done.

References

[1] MacWilliams F J and Sloane N J A 1977 *The Theory of Error-Correcting Codes* (Amsterdam: North-Holland)
[2] MacKay D J C 2000 Relationships between sparse graph codes *Proc. IBIS 2000 (Japan)* unpublished
[3] Gallager R G 1963 *Low Density Parity Check Codes* (Cambridge, MA: MIT Press)
[4] Berrou C, Glavieux A and Thitimajshima P 1993 Near Shannon-limit error-correcting coding and decoding: turbo-codes *Proc. 1993 IEEE Int. Conf. on Communications* p 1064
[5] MacKay D J C 1999 *IEEE Trans. Inf. Theory* **45** 399–431

[6] Luby M G, Mitzenmacher M, Shokrollahi M A, Spielman D and Stemann V 1997 Practical loss-resilient codes *Proc. 29th Annual ACM Symp. on Theory of Computing (STOC)* p 150

[7] Luby M G, Mitzenmacher M, Shokrollahi M A and Spielman D 2001 *IEEE Trans. Inf. Theory* **47** 585–98

[8] Davey M C 1999 Error-correcting using low-density parity check codes *PhD Thesis* Cambridge University

[9] Richardson T J, Shokrollahi M A and Urbanke R L 2001 *IEEE Trans. Inf. Theory* **47** 619–37

[10] Chung S-Y, Forney G D, Richardson T J and Urbanke R L 2001 *IEEE Commun. Lett.* **5** 58–60

[11] Kanter I and Saad D 1999 *Phys. Rev. Lett.* **83** 2660–3

[12] Kanter I and Saad D 2000 *J. Phys. A: Math. Gen.* **33** 1675–81

[13] Divsalar D, Jin H and McEliece R J 1998 Coding theorems for 'turbo-like' codes *Proc. 36th Allerton Conf. on Communications, Control, and Computing* p 201

[14] Jin H, Khandekar A and McEliece R J 2000 Irregular repeat-accumulate codes *Proc. 2nd Int. Symp. on Turbo Codes and Related Topics* p 1

[15] McEliece R J, MacKay D J C and Cheng J F 1998 *IEEE J. Sel. Areas Commun.* **16** 140–52

[16] Richardson T J and Urbanke R L 2001 *IEEE Trans. Inf. Theory* **47** 599–618

[17] Di C, Proietti D, Telatar E, Richardson T J and Urbanke R L 2002 Finite length analysis of low-density parity check codes on the binary erasure channel *IEEE Trans. Inf. Theory* **48** 1570–9

[18] Yedidia J S, Sudderth E B and Bouchaud J-P 2001 Projection algebra analysis of error-correcting codes *Proc. 39th Allerton Conf. on Communications, Control, and Computing* p 662

[19] Bazzi L, Richardson T J and Urbanke R L 2000 Exact thresholds and optimal codes for the binary symmetric channel and Gallager's decoding algorithm A *Proc. IEEE Symp. on Information Theory* p 203

[20] Tanner R M 1981 *IEEE Trans. Inf. Theory* **27** 533–47

[21] Burkhardt T W and Van Leeuwen J M J 1982 *Real Space Renormalization* (Berlin: Springer)

[22] Saul L K and Jordan M I 1994 *Neural Comput.* **6** 1174–84

[23] Ruger S M 1997 Decimatable Boltzmann machines for diagnosis: efficient learning and inference *World Congress on Scientific Computation, Modelling and Applied Mathematics, vol 4: Artificial Intelligence and Computer Science, IMACS '97 (Berlin)* p 319

[24] Wiberg N 1996 Codes and decoding on general graphs *PhD Thesis* University of Linkoping

[25] Wiberg N, Loeliger H-A and Kotter R 1995 *Eur. Trans. Telecommun.* **6** 513–25

[26] Chung S-Y, Richardson T J and Urbanke R L 2001 *IEEE Trans. Inf. Theory* **47** 657–70