SRC Technical Note
1998 - 017

December 11, 1998

A Note on Low Density Parity Check Codes for Erasures and Errors

Michael Mitzenmacher

COMPAQ

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

http://www.research.digital.com/SRC/

A Note on Low Density Parity Check Codes for Erasures and Errors

Abstract

We analyze low density parity check codes that correct both errors and erasures using a simple decoding scheme. Our framework unifies previous analyses for low density parity check codes and erasure codes. The result is a general class of codes that corrects both errors and erasures, with corresponding simple linear time encoding and decoding algorithms and provable performance guarantees. We also describe how these codes can be applied to yield a new fast algorithm for the setting of of Gaussian noise, and leave several open questions regarding the construction of these codes and the associated decoding algorithms.

1 Introduction

Simple linear time erasure codes with nearly optimal correction properties were introduced and analyzed in [2, 3]. The analysis of these codes are based on the analysis of a simple stochastic process on an irregular random bipartite graph. In [4], a similar analysis was used to develop and prove bounds on the behavior of irregular low density parity check codes, extending the previous work of Gallager [1] on regular low density parity check codes.

In this work, we consider low density parity check codes that handle both errors and erasures, building on the previous work of [2, 3, 4]. Our work generalizes and unifies the previous analyses in a natural way. As a result of this unification, we find simple codes for both erasures and errors with associated linear time encoding and decoding algorithms and provable performance bounds. For convenience, we call these codes LDEE codes (for Low Density codes for Errors and Erasures).

Before beginning, we note that one one obvious way to handle erasures using low density parity check codes is to simply replace any erased bit by a random bit. On average half of the erasures become errors. (Alternatively, one could first set for example all bits to 0, and rerun if necessary with all bits set to 1; in one of these two cases, at least half the erased bits are set correctly.) Of course, this approach ignores a great deal of useful information. By making better use of this information, our simple approach allows more erasures and errors to be simultaneously corrected.

It is also worth noting the connection between our work and the work on belief propagation (see, for example, [1, 5, 7]). Our decoding algorithms can be seen as simplified versions of belief propagation, where instead of probabilities being passed along edges of a graph, one of a small number of values is passed. Although our simplified LDEE schemes cannot successfully decode as many errors (or errors and erasures) as belief propagation, because our schemes are simpler, faster, and use less memory, they may still prove useful in practice. Another important advantage of these simplified schemes is that we can analyze them, to the extent that we can make provable statements about their asymptotic performance. We cannot yet make similar statements for codes based on belief propagation.

In Section 2, we describe our codes and general methods for analyzing them. We then explain in Section 3 how these codes can also easily be used in settings with Gaussian noise. We suggest how our codes could be improved and leave several related open questions in Section 4. In particular, there are many questions regarding the best design for both the decoding algorithms and the codes.

2 The Codes

In the following we refer to the nodes on the left and right sides of a bipartite graph as its *message* nodes and *check* nodes respectively. A bipartite graph with n nodes on the left and r nodes on the right gives rise to a linear code of dimension $k \geq n - r$, block length n, and rate $R = k/n$ in the following way: the bits of a codeword are indexed by the message nodes. A binary vector $\mathbf{x} = (x_1, \ldots, x_n)$ is a codeword if and only if $H\mathbf{x} = 0$, where H is the $r \times n$ incidence matrix of the graph whose rows are indexed by the check nodes and whose columns are indexed by the message nodes. In other words, $(x_1, \ldots, x_n)$ is a codeword if and only if for each check node the exclusive-or of its incident message nodes is zero.

An alternative approach is to allow the nodes on the right to represent bits rather than restrictions, and then use a cascading series of bipartite graphs, as described for example in [10] or [2]. In this situation, we know inductively the correct value of the check nodes in each layer when we correct the message nodes, and the check nodes are the exclusive-or of their incident message nodes.

In the sequel we focus on one bipartite graph only, and assume that only the nodes on the left are in error. The analysis that we provide in this case works for either of the two approaches given above, as we may inductively focus on just one layer in the context of cascading series of graphs [2, 10].

We now describe a decoding algorithm for codes based on irregular graphs, in the context where both erasures and errors are possible. In particular, we assume that with probability p_0 a message node receives the wrong bit, and with probability q_0 the bit corresponding to the message node is erased. We also assume that these parameters are known initially.

Following the notation used in [2], for an irregular bipartite graph we say that an edge has degree i on the left (right) if its left (right) hand neighbor has degree i. Let us suppose we have an irregular bipartite graph with some maximum left degree d_ℓ and some maximum right degree d_r. We specify our irregular graph by sequences $(\lambda_1, \lambda_2, \ldots, \lambda_{d_\ell})$ and $(\rho_1, \rho_2, \ldots, \rho_{d_r})$, where λ_i (ρ_i) is the fraction of edges with left (right) degree i. Further, we define $\rho(x) := \sum_i \rho_i x^{i-1}$. In practice, given a desired pair of degree sequences, an appropriate graph (or equivalently a parity check matrix H) can easily be constructed randomly, as described in [2, 4].

For most of this paper, we use example codes based on regular bipartite graphs, or graphs where all nodes on the left have the same degree and all nodes on the right have the same degree. For example, for a rate $1/2$ code where all message nodes have degree 3 and all check nodes have degree 6, we have $\lambda_3 = 1$ and $\rho_6 = 1$.

The decoding process proceeds in *rounds*, where in each round first the message nodes send each incident check node a value and then the check nodes send each incident message node a value. To picture the decoding process, consider an individual edge (m, c) between a message node m and a check node c, and an associated tree describing a neighborhood of m. This tree is rooted at m, and the tree branches out from the check nodes of m excluding c, as shown in Figure 1. For now let us assume that the neighborhood of m is accurately described by a tree for some fixed number of rounds.

We say that each message node m receives a value r_m. This value r_m is a 0 or 1 bit, or in the case where an erasure occurs, we use the notation $r_m = -1$. When we say a message node receives a bit (as opposed to a value) we exclusively mean either a 0 or a 1. A received bit r_m is purported to be the correct message bit. Thus, the value r_m is an incorrect message bit with probability p_0, and it is -1 with probability q_0. Each edge (m, c) remembers a value $g_{m,c}$ that is a guess of the correct bit of m, or possibly a -1. Initially this guess is just the value r_m. During each round a value is passed in each direction across each edge (m, c). The value $g_{m,c}$ is continually updated each round based on all information that is passed from to m from nodes other than c. Specifically, each round consists of an execution of the script in Figure 2, where the b_i used in the script are quantities to be determined later.

We describe the intuition behind the process described in Figure 2. A node m that was initially an erasure sends the 0/1 bit that corresponds to the majority vote of its other neighboring check nodes. This bit is the current best guess that node m has for its value. If the other check nodes are evenly divided, a -1 is sent. A node m that was not initially erased sends on to c the best guess for its value, based on the original bit received and the messages most recently sent by its other neighbors. (The best guess is associated with the value b_i, as we shall explain.) A check node c sends on the value to m that it believes m should have based

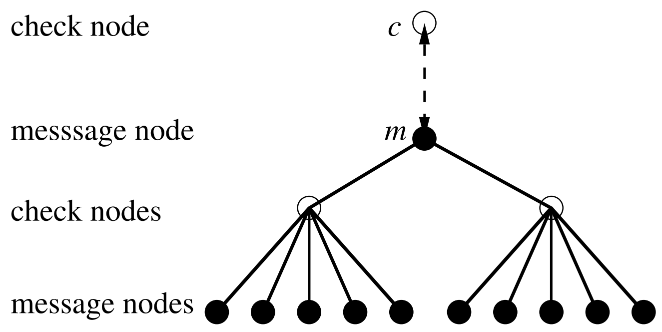

Figure 1: Representing the code as a tree.

- For all edges (m, c) do the following in parallel:
  - If this is the zero'th round, then set $g_{m,c}$ to r_m.
  - If this is a subsequent round $i + 1$, then $g_{m,c}$ is computed as follows:
    - if r_m is -1,
      - if the check nodes neighboring m excluding c sent more 0 bits than 1 bits to m, set $g_{m,c}$ to 0,
      - if the check nodes neighboring m excluding c sent more 1 bits than 0 bits to m, set $g_{m,c}$ to 1,
      - otherwise, set $g_{m,c}$ to -1,
    - if r_m is not -1,
      - if the difference between the number of 0 bits and 1 bits check nodes the neighboring m excluding c sent to m is at least b_i, set $g_{m,c}$ to 0,
      - if the difference between the number of 1 bits and 0 bits check nodes the neighboring m excluding c sent to m is at least b_i, set $g_{m,c}$ to 1,
      - otherwise, set $g_{m,c}$ to r_m,
  - In either case, m sends $g_{m,c}$ to c.
- For all edges (m, c) do the following in parallel:
  - if the check node c receives no -1 from nodes other than m, it sends to m the exclusive-or of the values received in this round from its adjacent nodes excluding m.
  - otherwise c sends m a -1

Figure 2: Pseudo-code for the main decoding loop.

on the messages of c's other neighbors. If one of those neighbors sends a -1, representing that it has no reason to favor sending a 0 or 1, the check node has no reason to favor a value for m as well, and hence sends on a -1. This process generalizes the algorithm for error-correction presented in [1, 4]; in the case of no erasures, a 0/1 bit is sent every round along every edge. This process also generalizes the sequential algorithm for erasure correction of [2]; in the case of no errors, note that the only values passed along edges are -1 or the true bit value. Hence a node can be considered corrected as soon as it receives a bit value. (And in fact the algorithm of [2] takes advantage of this, sequentially filling in values for nodes as they become available.)

Of course the parallel work can easily be simulated sequentially. Moreover, the work per round can easily be coded so that it is linear in the number of edges.

After each round the message nodes can choose a guess for their correct value based on the edge information; if all guesses are consistent with the check nodes, then the process may stop having found a codeword. Presumably the codeword is the correct one. To show the process stops with the correct code word, we consider the edges, since it is clear that if each edge passes on the correct value, the process terminates successfully.

Let p_i be the probability that m sends c an incorrect bit $g_{m,c}$ in round i. Similarly, let q_i be the probability that m sends c the value $g_{m,c} = -1$ in round i. Following the approach used in [1, 2, 3, 4], we determine a recursive equation describing the evolution of both p_i and q_i over a constant number of rounds.

To analyze the algorithm, consider the process from the point of view of an edge (m, c). First we consider what some other check node c' sends to m at the end of the ith round. The node c' sends m a -1 if any of its other neighboring message nodes sends it a -1. As the probability each other message node sent a -1 is q_i, the probability that c' sends m a -1 is $1 - \rho(1 - q_i)$. (Note here that we have not specified the degree of c'; instead we have used that the degree of the edge (m, c') on the right is j with probability ρ_j, and that leads to the above expression.) Otherwise, c' sends m its correct bit as long as there are an even number (including possibly 0) of other message nodes sending c' the wrong bit. As the probability each bit was sent incorrectly to c' is p_i, and the probability a -1 was sent is q_i, it is simple to check that the probability that c' receives an even number of errors and no -1 is $\frac{\rho(1-q_i)+\rho(1-q_i-2p_i)}{2}$. Similarly, the probability that c' receives an odd number of errors and no -1 is $\frac{\rho(1-q_i)-\rho(1-q_i-2p_i)}{2}$.

For notational convenience, we define

$$\rho_{+,i} = \frac{\rho(1 - q_i) + \rho(1 - q_i - 2p_i)}{2}; \;\; \rho_{-,i} = \frac{\rho(1 - q_i) - \rho(1 - q_i - 2p_i)}{2}; \;\; \rho_{?,i} = 1 - \rho(1 - q_i).$$

That is, $\rho_{+,i}$, $\rho_{-,i}$, and $\rho_{?,i}$ are the probabilities that a node c' sends respectively the correct bit, the wrong bit, and a -1 value to node m in round i.

Let us first consider the recursive formula for q_{i+1}. A random edge (m, c) has left degree j with probability λ_j. It passes on a -1 only if m received a -1 and the number of neighboring check nodes c' other than c sending the bit 0 equals the number of such neighboring check nodes sending the bit 1. Letting h represent the number of correct bits sent, we find

$$q_{i+1} \;\; = \;\; q_0 \sum_{j=1}^{d_\ell} \lambda_j \sum_{h=0}^{\lfloor (j-1)/2 \rfloor} \binom{j-1}{h;\, h;\, j-1-2h} \rho_{?,i}^{j-1-2h} \;\rho_{+,i}^{h} \,\rho_{-,i}^{h} \qquad (1)$$

Now let us consider the recursive formula for p_{i+1}. For convenience, let us consider the case where a node m that received a bit r_m initially and obtains k bits (that is, not -1's!) from neighbors other than c in round i. In this case, we suppose that node m passes on r_m unless at least $b_{i,k}$ of the k bits received are not r_m. Note that $b_{i,k}$ depends not on the degree of the node, but on the number of actual bits the node m receives from neighbors other than c. To analyze p_{i+1} again consider a random edge (m, c). The probability p_{i+1}

can be expressed as the sum of disjoint cases: if m initially received the wrong bit, it might not be corrected; if m initially received the correct bit, it might passed the wrong value; and if m was initially erased, it might pass on the wrong value. The recursive description for p_i is thus

$$p_{i+1} = p_0 + \sum_{j=1}^{d_\ell} \lambda_j \sum_{k=0}^{j-1} \binom{j-1}{k} \rho_{?,i}^{j-1-k} \left[\sum_{h=b_{i,k}}^{k} \binom{k}{h} \left((1-p_0-q_0)\rho_{+,i}^{k-h}\, \rho_{-,i}^{h} - p_0 \rho_{+,i}^{h}\, \rho_{-,i}^{k-h} \right) + \sum_{h=\lceil (k+1)/2 \rceil}^{k} \binom{k}{h} q_0 \rho_{+,i}^{k-h}\, \rho_{-,i}^{h} \right]. \quad (2)$$

The value of $b_{i,k}$ that minimizes the value of p_{i+1} is given by the smallest integer that satisfies:

$$\frac{1-p_0-q_0}{p_0} \le \left(\frac{\rho_{+,i}}{\rho_{-,i}} \right)^{2b_{i,k}-k}. \quad (3)$$

This equation is derived by setting $b_{i,k}$ to the smallest value that makes the probability of changing a correct message to an incorrect message smaller than the probability of changing an incorrect message to a correct message. Equation (3) has an interesting interpretation. (Here we follow [4].) Note $2b_{i,k} - k$ is a constant fixed by the above equation. The number $b_i = 2b_{i,k} - k = b_{i,k} - (k - b_{i,k})$ can be interpreted as the difference between the number of check nodes that agree in the majority and the number that agree in the minority when k of the $j-1$ check nodes send actual bit values. We call this difference the *discrepancy* of a node. Equation (3) tells us that we need only check that the discrepancy is above a certain threshold (independent of k) to decide which value to send for a message node that received a bit. Hence we described the algorithm originally in this manner.

With the above equations, we can check for a given degree sequence (λ and ρ) and given error probabilities (p_0 and q_0) whether both p_i and q_i converge to 0. Note, however, that even if p_i and q_i converge to 0, this does not directly imply that the process correctly finds a codeword, even just with high probability. This is because our assumption that the neighborhood of (m, c) is accurately represented by a tree for arbitrarily many rounds is not true. In fact, even for a randomly chosen graph with the right degree sequence, for any constant number of rounds it is true only with high probability.

This problem can be overcome in standard ways. In the case of regular graphs (where all edges have the same left degree and the same right degree), Gallager developed an explicit construction of graphs with no small cycles that leads to high probability bounds. Alternatively, random constructions yield few small cycles. Using this fact, one can show that (up to lower order terms) the equations above approximately hold for any constant number of rounds, for a large enough message size n. (The analysis is entirely the same as in [3, 4].) A constant number of rounds suffices to correct almost all errors and erasures. Fixing the remaining errors and erasures can be easily handled using a small additional graph structure and an additional code (such as the regular, constructive versions of these codes, or the codes of Spielman[10]). These concerns are primarily theoretical; in practice, random constructions generally work well without any additional graph structure, and the process terminates having successfully found a codeword. Using the theory, however, allows us to state the following theorem:

Theorem 1 *Consider parameters p_0,q_0, $(\lambda_1, \lambda_2, \ldots, \lambda_{d_\ell})$ and $(\rho_1, \rho_2, \ldots, \rho_{d_r})$, with all $\lambda_i, \rho_i \ge 2$. If there exist thresholds $b_{i,k}$ such that the recurrences (1) and (2) define a sequence (p_i, q_i) with $\lim_{i\to\infty}(p_i, q_i) \to (0, 0)$, then for any $\epsilon > 0$ and sufficiently large block size n there is a code of rate $R - \epsilon$ that simultaneously corrects errors made independently with probability p_0 and erasures made independently with probability q_0 with high probability (polynomially small in n).*

Proof: One constructs a random graph with the given edge degree sequences to yield the code. The proof follows from the same arguments as [3, 4]. A martingale argument (on the edges) shows that for sufficiently large n, most of the graph behaves like a tree for sufficiently many rounds. Furthermore, for each tree shape, the number of edges that lie atop a tree of that shape is close to its expected value with high probability. A second martingale argument on the received values r_m then shows that the number of edges that pass on the correct value is close to its expectation with high probability. Putting this together shows that our recursion provides nearly correct results for sufficiently large n for a suitably large constant number of rounds. A simple clean-up stage to correct remaining errors reduces the rate from R to $R - \epsilon$. ■

When p_i and q_i do not both converge to 0, we have found their values converge to a *fixed point*. That is, we reach a point (p_i, q_i) such that $(p_{i+1}, q_{i+1}) = (p_i, q_i)$. This fixed point corresponds to the asymptotic fraction of edges passing errors and -1 values even if one runs the process for arbitrarily many rounds.

In practice, the equations can be used to estimate actual code performance. Although the equations describe the asymptotic performance as the number of nodes n grows to infinity, generally the model provides reasonably accurate estimates of performance even for relatively small n (in the thousands).

3 On Gaussian Noise

Low density parity check codes for correcting errors can easily be applied to the model where noise is Gaussian by converting the received signal to a bit value; errors occur for bits where the noise is large. Of course this approach removes significant information from the received transmission, so one can not hope to handle as many errors as, for example, belief propagation. In return one gains simplicity and speed.

Using LDEE codes keeps the advantages of simplicity and speed while substantially improving performance by using the -1 value. With LDEE codes, the Gaussian signal is initially converted to a 0,1, or -1 value. The choice of how to map signals to values affects the probability that that each bit is an error or is treated as an erasure. A suitable mapping choice can be determined given the strength of the Gaussian noise and the code.

For example, consider the case of the regular code with message nodes having degree 3 and check nodes having degree 6 (a rate 1/2 code), using the decoding algorithm of Section 2. We graph the asymptotic maximum fraction of erasures possible for a given fraction of errors in Figure 3, which we call the *tolerance curve*. (The curve is approximate; it was obtained by using the Equations (1) and (2) to find the maximum q_0 for $p_0 = 0, 0.002, 0.004, \ldots$.) Figure 3 also shows a curve demonstrating the tradeoff between error and erasure generation under Gaussian noise, which we call the *tradeoff curve*. This curve is for the case where the received signal should have either mean 1 or -1 depending on the bit value, and the standard deviation of the signal is $\sigma = 0.70$. The curve is generated as follows: for a given noise, consider the effect when all received values in the range $[-z, z]$ are treated as erasures. Then the probability of an erasure is $\Phi\left(\frac{-1+z}{\sigma}\right) - \Phi\left(\frac{-1-z}{\sigma}\right)$, and the probability of an error is $\Phi\left(\frac{-1-z}{\sigma}\right)$. Considering how these points vary as a function of z yields the appropriate curve.

If the tolerance curve lies below the tradeoff curve everywhere, then regardless of how one maps the received signal to errors and erasures, the decoding algorithm fails (with high probability). Where the tradeoff curve falls below the tolerance curve yields points where the mapping allows decoding. Hence from Figure 3 we can tell that at the given noise level simply translating the signal directly to either a 0 or 1 bit would be ineffective; however, by treating all signals in the range $[-0.5, 0.5]$ as erasures, decoding is possible (asymptotically, for large enough messages).

We tested our argument above with simulation. We treated all signals in the range $[-0.5, 0.5]$ as erasures, but used a standard deviation of $\sigma = 0.65$, allowing ourselves some room. (Recall that our theoretical results imply that that when the tradeoff curve falls below the tolerance curve, decoding occurs with high

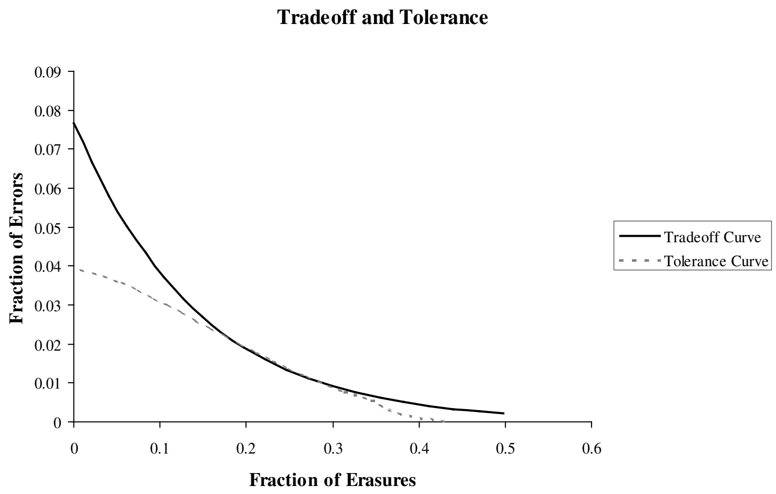

Figure 3: Tradeoff curve for $\sigma = 0.70$ and tolerance curve for a 3-6 regular bipartite graph.

probability for sufficiently large graphs; for moderate sized graphs, the noise should be slightly lower than the asymptotic theory suggests.) On a single randomly chosen bipartite graph with 16,000 nodes of degree 3 on the left and 8,000 nodes of degree 6 on the right (modified to remove small cycles of length 2 and 4), the decoding algorithm worked successfully on all 1,000 trial runs. More experimentation would be worthwhile; however, this result demonstrates that our analysis is accurate enough to be useful on reasonable sized systems.

More generally, this approach allows an approximate determination the maximum acceptable signal noise for a code specified by its degree sequences using a simple binary search approach. We can test a given noise value (given say by σ) by plotting the appropriate tradeoff curve and comparing it to the tolerance curve of the given code. If the tradeoff curve lies somewhere below the tolerance curve, a higher noise level can be tested; otherwise, a lower noise level can be tested. Noise distributions other than the Gaussian can be handled in an entirely similar fashion.

4 Improvements

4.1 Improving the Decoding

In the decoding algorithm of Section 2, we allow three possible values to be passed along the edges. The value -1 specifically denotes a complete lack of preference among the two possible choices for a bit. It seems highly unlikely that this is the best use for this symbol.

That passing more detailed information each round can lead to better decoding is intuitively obvious. The limiting case, where nodes pass estimates of their probability of being a 0 (or 1) according to a Bayesian calculation along the edges, is just belief propagation. Belief propagation has proven highly effective, although it is generally slower than the simpler approach taken here, and there are no provable performance guarantees.

It is not clear, however, how to best make use of the ability to pass three possible values (or, for that matter, how to best make use of the ability to pass a small finite number of values). One possibility we explore here is to use each value to denote a range of probabilities. More explicitly, during each round the value 0, 1, or -1 will be sent by a message node. Sending a -1 will denote that the conditional probability of being either a 0 or a 1 is between α_i and $1 - \alpha_i$, for some $\alpha_i < 0.5$ chosen explicitly in advance. Sending a 0 along an edge denotes that the conditional probability of being a 0 is at least $1-\alpha_i$, and similarly for a 1. Check nodes will behave as in the algorithm of Section 2, providing the exclusive-or of other neighboring values when none of these values are -1. (Note that, when $\alpha_i = 0.5$, we obtain the original algorithm presented in Section 2.)

We offer some intuition as to why varying the parameter α might be helpful. Suppose we run the original algorithm ($\alpha = 0.5$) until it reaches a fixed point. By then decreasing the parameter α, we reduce the number of errors, making each sent bit more valuable, at the expense of introducing more -1 values. It is possible this tradeoff might be beneficial; if so, then by continuing with the original algorithm, we may find that we have passed the fixed point, allowing the decoding to complete.

For this variation, there are three threshold values to compute. The first, $b^1_{i,k}$, represents the number of neighbors that must agree on a bit for a message node m with $r_m = -1$ to send that bit in round $i+1$. The correct choice for the threshold $b^1_{i,k}$ is the smallest integer that satisfies

$$\frac{1-\alpha_i}{\alpha_i} \leq \left(\frac{\rho_{+,i}}{\rho_{-,i}}\right)^{2b^1_{i,k}-k+1}. \tag{4}$$

This choice guarantees that a bit is sent only if the conditional probability that it is correct is at least $1-\alpha$.

The threshold $b^2_{i,k}$ represents the number of neighbors that must agree with a message node m with value $r_m \neq -1$ to send on r_m; similarly, $b^3_{i,k}$ represents the number of neighbors that must disagree with a message node m with value $r_m \neq -1$ to send on the other possible bit value. The thresholds are the smallest integers satisfying:

$$\frac{(1-\alpha_i)p_0}{\alpha_i(1-p_0-q_0)} \leq \left(\frac{\rho_{+,i}}{\rho_{-,i}}\right)^{2b^2_{i,k}-k+1}; \tag{5}$$

$$\frac{(1-\alpha_i)(1-p_0-q_0)}{\alpha_i p_0} \leq \left(\frac{\rho_{+,i}}{\rho_{-,i}}\right)^{2b^3_{i,k}-k+1} \tag{6}$$

Note that if neither threshold is met, the message node m passes on the value -1.

In the following recursive equations for p_{i+1} and q_{i+1}, we use k to denote a number of message nodes not passing -1, and h to denote a number of message nodes passing the correct bit.

$$\begin{aligned} q_{i+1} = & \sum_{j=1}^{d_\ell} \lambda_j \sum_{k=0}^{j-1} \binom{j-1}{k} \rho_{?,i}^{j-1-k} \left[\left(q_0 \sum_{h=k-b^1_{i,k}+1}^{b^1_{i,k}-1} \binom{k}{h} \rho_{+,i}^h \rho_{-,i}^{k-h} \right) + \right. \\ & \left(p_0 \sum_{h=k-b^2_{i,k}+1}^{b^3_{i,k}-1} \binom{k}{h} \rho_{+,i}^h \rho_{-,i}^{k-h} \right) + \\ & \left. \left((1-p_0-q_0) \sum_{h=k-b^3_{i,k}+1}^{b^2_{i,k}-1} \binom{k}{h} \rho_{+,i}^h \rho_{-,i}^{k-h} \right) \right] \end{aligned} \tag{7}$$

$$\begin{aligned} p_{i+1} = & \sum_{j=1}^{d_\ell} \lambda_j \sum_{k=0}^{j-1} \binom{j-1}{k} \rho_{?,i}^{j-1-k} \left[\left(q_0 \sum_{h=0}^{k-b^1_{i,k}} \binom{k}{h} \rho_{+,i}^h \rho_{-,i}^{k-h} \right) + \right. \\ & \left(p_0 \sum_{h=0}^{k-b^2_{i,k}} \binom{k}{h} \rho_{+,i}^h \rho_{-,i}^{k-h} \right) + \end{aligned} \tag{8}$$

$$\left((1-p_0-q_0)\sum_{h=0}^{k-b^3_{i,k}}\binom{k}{h}\rho_{+,i}^{h}\rho_{-,i}^{k-h}\right)\Bigg]$$

We have experimented with this framework on regular codes, where the left degrees are all the same and the right degrees are all the same. We began by using $\alpha = 0.5$; that is, we ran the original algorithm. After reaching a fixed point, we tried different values of α_i. In some cases, we successfully moved off the fixed point and found a trajectory that led the p_i and q_i values to both converge to 0. However, the improvement in the number of errors and erasures that can be corrected over the original algorithm of using this ad hoc technique as yet appear inconsequential. Determining whether this approach can lead to significant improvements and determining a methodology for appropriately setting the α_i values remain challenging open questions. Similarly, there may be better ways of using the three values 0, 1, and -1 being sent along the edges, and better schemes that use a larger number of possible values. Answering these questions might well provide us with a greater understanding of the limiting case of belief propagation.

4.2 Finding Good Irregular Codes

The analysis of Section 2 allows one to estimate code performance in advance, given a pair of degree sequences. For the special cases of no errors or no erasures, the pair of equations (1) and (2) reduce to a single equation. As shown in [2, 4], these equations can be used to develop a tool for determining the good degree sequences for irregular codes. This tool is based on the following idea: suppose we consider the case where there are only errors. We consider the problem of finding a feasible left degree sequence $(\lambda_1, \lambda_2, \ldots, \lambda_{d_\ell})$, given a right degree sequence $(\rho_1, \rho_2, \ldots, \rho_{d_r})$ and a target error probability p_0. (A similar approach can be sued to find a feasible right degree sequence from a left degree sequence.) A sufficient condition for success (at least asymptotically) is to find a sequence λ_i for which $p_{i+1} < p_i$. Given the ρ_i, Equation (2) reduces to the statement $p_{i+1} = f(p_i)$, where f is a function linear in the λ_i. Hence we can write a linear program with the λ_i as variables, attempting to find a feasible solution for $f(x) < x$ at many sample values x spread on the interval $[0, p_0]$. The solution to this linear program should give us a suitable left hand sequence that comes very close to achieving the target error probability p_0.

It would of course be useful to have a similar tool to find good irregular degree sequences for LDEE codes. As of yet, we have not been able to design such a tool. We offer a somewhat technical explanation regarding why the previous technique does not appear to generalize. The equations (7) and (8) do similarly reduce to equations of the form $p_{i+1} = f(p_i, q_i)$ and $q_{i+1} = f(p_i, q_i)$ that are linear in the λ_i. It is not clear, however, what is the appropriate condition to place on these equations. For example, clearly we do not want to set $p_{i+1} < f(p_i, q_i)$, as in cases where one begins with a small number of errors and large number of erasures, the number of errors will initially increase substantially, as the erased nodes attempt to make their best decision. Even for more apparently reasonable conditions, such as $p_{i+1}+q_{i+1} < f(p_i, q_i)+g(p_i, q_i)$, it is not clear for what range the condition has to hold. The interval $[0, p_0] \times [0, q_0]$ appears necessary, but not sufficient, since again it is possible for $p_1 > p_0$, in which case we are at a point outside this region.

Although these difficulties do not appear to be substantial, we have not yet found a means of overcoming them. Thus developing an approach finding good irregular LDEE codes remains an open question.

5 Summary

We have demonstrated that previous work on low density parity check codes for error correction and similar codes for erasure correction can be unified under a common pair of equations that describe codes for both errors and erasures. Our approach leads to codes with provable performance results, and the decoding algorithm is simpler than belief propagation; however, as a result, the performance of these codes is not as good as that of belief propagation. Several open questions remain, including how to best design such codes and how to best design simple, effective message-passing decoding algorithms.

References

[1] R. G. Gallager, **Low-Density Parity-Check Codes**, MIT Press, 1963.

[2] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical Loss-Resilient Codes", *Proc.* 29^{th} *Symp. on Theory of Computing*, 1997, pp. 150–159.

[3] M. Luby, M. Mitzenmacher, and M. A. Shokrollahi, "Analysis of Random Processes via And-Or Trees", *Proc.* 9^{th} *Symp. on Discrete Algorithms*, pp. 364-373, 1998.

[4] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman, "Analysis of Low Density Codes and Improved Designs Using Irregular Graphs", *Proc.* 30^{th} *ACM Symp. on Theory of Computing*, pp. 249-258, 1998.

[5] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman, "Improved Low-Density Parity-Check Codes Using Irregular Graphs and Belief Propagation," appeared in ISIT 1998.

[6] D. J. C. MacKay, R, J. McEliece, and J.-F. Cheng, "Turbo Coding as an Instance of Pearl's 'Belief Propagation' Algorithm", to appear in *IEEE Journal on Selected Areas in Communication.*

[7] D. J. C. MacKay and R. M. Neal, "Good Error Correcting Codes Based on Very Sparse Matrices", available from **http://wol.ra.phy.cam.ac.uk/mackay**.

[8] D. J. C. MacKay and R. M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes", to appear in *Electronic Letters*.

[9] M. Sipser, D. A. Spielman, "Expander Codes", *IEEE Transactions on Information Theory*, 42(6), November 1996, pp. 1710-1722.

[10] D. A. Spielman, "Linear Time Encodable and Decodable Error-Correcting Codes", *IEEE Transactions on Information Theory*, 42(6), November 1996, pp. 1723-1731.