Analysis of Low Density Codes and Improved Designs Using Irregular Graphs

Michael G. Luby[*] **Michael Mitzenmacher**[†]

M. Amin Shokrollahi[‡] **Daniel A. Spielman**[§]

TR-97-045

Abstract

In [6], Gallager introduces a family of codes based on sparse bipartite graphs, which he calls low-density parity-check codes. He suggests a natural decoding algorithm for these codes, and proves a good bound on the fraction of errors that can be corrected. As the codes that Gallager builds are derived from regular graphs, we refer to them as *regular codes*.

Following the general approach introduced in [7] for the design and analysis of loss-resilient codes, we consider error-correcting codes based on random irregular bipartite graphs, which we call *irregular codes*. We introduce tools based on linear programming for designing linear time irregular codes with better error-correcting capabilities than possible with regular codes. For example, the decoding algorithm for the rate 1/2 regular codes of Gallager can provably correct up to 5.17% errors, whereas we have found irregular codes for which our decoding algorithm can provably correct up to 6.27%.

[*]International Computer Science Institute, Berkeley, CA. Parts of this research were done while still at the Digital Equipment Corporation Systems Research Center, Palo Alto, CA. Research partially supported by NSF operating grant NCR-9416101. E-mail: `luby@icsi.berkeley.edu`.

[†]Digital Equipment Corporation, Systems Research Center, Palo Alto, CA. E-mail: `michaelm@pa.dec.com`.

[‡]International Computer Science Institute Berkeley, and Institut für Informatik der Universität Bonn, Germany. Research supported by a Habilitationsstipendium of the Deutsche Forschungsgemeinschaft, Grant Sh 57/1–1. E-mail: `amin@icsi.berkeley.edu`.

[§]Department of Mathematics, M.I.T. E-mail: `spielman@math.mit.edu`.

1 Introduction

In [6], Gallager introduces a family of codes based on sparse bipartite graphs, which he calls low-density parity-check codes. As the codes that Gallager builds are derived from regular graphs, we refer to them as *regular codes*. He suggests a natural decoding algorithm for these codes, and proves a good lower bound on the fraction of errors that can be corrected, assuming that there are no short cycles in the underlying graph. While much of his work concerns randomly chosen graphs, his analysis does not directly apply to such graphs. Instead, he constructs explicit graphs of large girth to which his analysis does apply.

The main contribution of this paper is the design and analysis of low-density parity-check codes based on irregular graphs. This work follows the general approach introduced in [7] for the design and analysis of loss-resilient codes. There it is shown that using irregular graphs yields codes with much better performance than regular graphs. In accordance with [7], we consider error-correcting codes based on random irregular bipartite graphs, which we call *irregular codes*. We develop tools based on linear programming for designing linear time encodable and decodable irregular codes with better error-correcting capabilities. For example, the rate 1/2 regular codes of Gallager can provably correct up to 5.17% errors, whereas we have found irregular codes that can provably correct up to 6.27%.

The only method we currently have for constructing irregular codes is by randomly choosing the irregular graph. However, the analysis used by Gallager does not directly apply to such graphs. Thus, to analyze the performance of the irregular codes, we develop an analysis that applies to randomly chosen graphs. Using techniques from [8] for studying random processes, we show that Gallager's original algorithm with high probability successfully corrects all but an arbitrarily small constant fraction of the message. Once the number of erroneous bits is reduced to this level, we switch from Gallager's algorithm to one used by Spielman and Sipser in [15], and prove that this new hybrid method successfully finishes the decoding with high probability. This analysis easily extends to the irregular codes that we introduce.

Gallager's decoding algorithm is a simplification of "belief propagation" [14]. Belief propagation has been extensively tested with Gallager's low-density parity-check codes [2, 6, 11, 12, 17] and is strongly related to the highly successful turbo codes [1, 3, 10, 5]. In a separate work, we describe empirical tests on irregular codes using a full belief propagation algorithm and demonstrate irregular codes with better performance than regular codes [9]. We believe our analysis here provides an important step towards analyzing codes based on belief propagation techniques.

The paper proceeds as follows: in Section 2.1, we present a description of regular codes and analyze Gallager's decoding scheme. We show in Section 2.2 how expander-based arguments can be used in addition to the previous analysis to demonstrate a decoding algorithm that works with high probability. We introduce irregular graphs in Section 3, where we demonstrate that our previous arguments generalize to irregular graphs and describe how to find irregular graphs that lead to good codes. In Section 4, we discuss some simulation results that show the effectiveness of our analysis for designing practical codes. We conclude with a discussion of open problems.

2 Regular Codes

2.1 Analyzing Regular Codes

We first review the codes developed by Gallager and his analysis [6]. Later we explain how his analysis combined with the argument from [8] shows that his suggested decoding algorithm corrects all but an arbitrarily small constant fraction of the nodes with high probability for random regular codes. The decoding algorithm of Gallager's that we analyze is an example of *hard decision decoding*, which signifies that at each step the state is derived from local decisions of whether each bit is 0 or 1, and this is all the information the state contains (as opposed to more detailed probabilistic information). We note that Gallager also proposes a belief propagation type decoding algorithm, which uses a more complicated state; for more details, see [11, 4, 17, 9].

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 $k \geq n - r$ and block length 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 [16] or [7]. 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 [16, 7] in the context of cascading series of graphs. We call the linear codes that are obtained by either of the above constructions *regular codes*.

Consider a regular random graph with the message nodes having degree d_ℓ and the check nodes having degree d_r. With probability p a message node receives the wrong bit. The decoding process proceeds in *rounds*, where in each round first the message nodes send each incident check node a single bit and then the check nodes send each incident message node a single bit. 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.

Each message node m remembers the received bit r_m that is purported to be the correct message bit. (Thus, r_m is not the correct message bit with probability p.) Each edge (m, c) remembers a bit $g_{m,c}$ that is a guess of the correct bit of m. This bit is continually updated each round based on all information that is passed from c to m. During each round a bit is passed in each direction across edge (m, c). Each round consists of an execution of the following script:

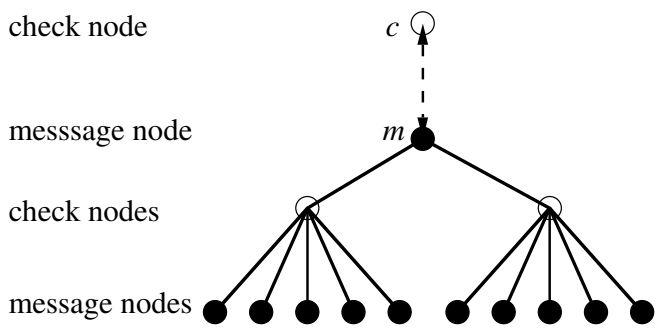

Figure 1: Representing the code as a tree.

- For all edges (m, c) do the following in parallel:
  - If this is the zeroth round, then set $g_{m,c}$ to r_m.
  - If this is a subsequent round, then $g_{m,c}$ is computed as follows:
    - if all the check nodes of m excluding c sent the same value to m in the previous round, then set $g_{m,c}$ to this value,
    - else 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:
  - the check node c sends to m the exclusive-or of the values it received in this round from its adjacent message nodes excluding m.

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.

Let p_i be the probability that m sends c an incorrect value $g_{m,c}$ in round i. Initially $p_0 = p$. Following the work of Gallager, we determine a recursive equation describing the evolution of p_i over a constant number of rounds.

Consider the end of the ith round, and consider a check node c' of m other than c. The node c' sends m its correct value as long as there are an even number (including possibly 0) message nodes other than m sending c' the wrong bit. As each bit was correctly sent to c' with probability p_i, it is easy to check that the probability that c' receives an even number of errors is

$$\frac{1 + (1 - 2p_i)^{d_\ell - 1}}{2}. \tag{1}$$

Hence, the probability that m was received in error and sent correctly in round $i+1$ is

$$p_0 \left[\frac{1 + (1-2p_i)^{d_r - 1}}{2} \right]^{d_\ell - 1},$$

and similarly the probability that m was received correctly but sent incorrectly in round $i+1$ is given by

$$(1-p_0) \left[\frac{1 - (1-2p_i)^{d_r - 1}}{2} \right]^{d_\ell - 1}.$$

This yields an equation for p_{i+1} in terms of p_i:

$$p_{i+1} = p_0 - p_0 \left[\frac{1 + (1-2p_i)^{d_r - 1}}{2} \right]^{d_\ell - 1} + (1-p_0) \left[\frac{1 - (1-2p_i)^{d_r - 1}}{2} \right]^{d_\ell - 1}. \tag{2}$$

Gallager's idea is then to find the supremum p^* of all values of p_0 for which the sequence p_i is monotonically decreasing and hence converges to 0. Note, however, that even if p_i converges to 0, this does not directly imply that the process necessarily corrects all message nodes, even 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 any constant number of rounds it is true only with high probability.

Gallager proves that, as the block length of the code and girth of the graph grow large, this decoding algorithm works for all $p_0 < p^*$. Since random graphs do not have large girth, Gallager introduced explicit constructions of regular sparse graphs that do have sufficiently large girth for his analysis to hold. We will shortly provide an analysis that shows that Gallager's decoding algorithm successfully corrects a large fraction of errors for a randomly chosen regular graph with high probability. Then in Section 2.2 we show how to ensure the decoding terminates successfully with high probability using a slightly different decoding rule.

Gallager notes that the decoding rule can be relaxed in the following manner: at each round, there is a universal threshold value b (to be determined below) that depends on the round number. For each message node m and neighboring check node c, if at least b neighbors of m excluding c sent the same bit to m in the previous round, then m sends this bit to c in this round; otherwise m sends to c its initial bit r_m. The rest of the decoding algorithm is the same. Using the same analysis as for equation (2), we may find a recursive description of the p_i.

$$\begin{aligned} p_{i+1} \quad = \quad & p_0 - p_0 \sum_{t=b}^{d_\ell - 1} \binom{d_\ell - 1}{t} \left[\frac{1 + (1-2p_i)^{d_r - 1}}{2} \right]^t \left[\frac{1 - (1-2p_i)^{d_r - 1}}{2} \right]^{d_\ell - 1 - t} + \\ & (1-p_0) \sum_{t=b}^{d_\ell - 1} \binom{d_\ell - 1}{t} \left[\frac{1 - (1-2p_i)^{d_r - 1}}{2} \right]^t \left[\frac{1 + (1-2p_i)^{d_r - 1}}{2} \right]^{d_\ell - 1 - t}. \end{aligned} \tag{3}$$

We choose b so as to minimize p_{i+1}. To do this we compare the odds of being right initially to the odds of being right using the check nodes and the threshold b. As determined

by Gallager, the correct choice of b is the smallest integer that satisfies

$$\frac{1-p_0}{p_0} \leq \left[\frac{1+(1-2p_i)^{d_r-1}}{1-(1-2p_i)^{d_r-1}}\right]^{2b-d_\ell+1}. \tag{4}$$

Note that b is an increasing function of p_i; this is intuitive, since as p_i decreases, smaller majorities are needed to get an accurate assessment of m's correct value. Also, note that while the algorithm functions by passing values along the edges, it can also keep a running guess for the value of each message node based on the passed values. The algorithm continues until the proposed values for the message nodes satisfy all the check nodes, at which point the algorithm terminates with the belief that it has successfully decoded the message, or it can fail after a preset number of rounds.

It follows simply from a similar argument in [8] that the recursive description given by equation (3) is correct with high probability over any constant number of rounds.

Theorem 1 *Let $i > 0$ be an integer constant and let Z_i be the random variable describing the fraction of edges set to pass incorrect messages after i rounds of the above algorithm. Further, let p_i be as given in the recursion* (3)*. Then there is a constant c such that for any $\epsilon > 0$ and sufficiently large n we have*

$$\Pr(|Z_i - p_i| > \epsilon) < \exp(-c\epsilon n).$$

Proof: See the Appendix.

Corollary 1 *Given a random regular code with p_i as defined by equation (3), if the sequence p_i converges to 0, then for any $\eta > 0$ there is a sufficiently large message size n such that Gallager's hard decision decoding correctly decodes all but at most ηn bits in some constant number r_η of rounds with high probability.*

2.2 Completing the Work: Expander-based Arguments

In the previous section we have shown that the hard decision decoding corrects all but an arbitrarily small constant fraction of the message nodes for regular codes with sufficiently large block lengths. The analysis, however, is not sufficient to show that the decoding process completes successfully. In this section, we show how to finish the decoding process with high probability once the number of errors is sufficiently small using slightly different algorithms. Our work utilizes the expander-based arguments in [15, 16].

We first define what we require in terms of the bipartite graph represented by the code being a good expander.

Definition 1 *A bipartite graph has expansion (α, β) if for all subsets S of size at most αn of the vertices on the left, the size of the neighborhood $N(S)$ of S on the right satisfies $N(S) \geq \beta|\delta(S)|$, where $\delta(S)$ is the set of edges attached to vertices in S.*

Following the notation of [15], we call a message node corrupt if it differs from its correct value, and we call a check node satisfied (respectively unsatisfied) if its value is (is

not) the sum of the values of its adjacent message nodes. The work of [15] shows that if the underlying bipartite graph of a code has sufficient expansion for sets of size up to αn, then both of the following algorithms can correct any set of $\alpha n/2$ errors:

Sequential decoding: if there is a message node that has more satisfied than unsatisfied neighbors, flip the value of that message node. Repeat until no such message node remains.

Parallel decoding: for each message node, count the number of unsatisfied check nodes among its neighbors. Flip in parallel each message node with a majority of unsatisfied neighbors.

Note that the above algorithms are very similar to Gallager's hard decision decoding algorithm, except that here we need not hold values for each (message node,check node) pair. We call upon the results of [15] to show that once we use hard decision decoding to correct all but some arbitrarily small fraction of the message nodes, we can finish the process. The next lemma follows from Theorems 10 and 11 of [15].

Lemma 1 *Let $\alpha > 0$ and $\beta > 3/4 + \epsilon$ for some fixed $\epsilon > 0$. Let B be an (α, β) expander. Then the sequential and parallel decoding algorithms correct up to $\alpha n/2$ errors. The sequential decoding algorithm does so in linear time and the parallel decoding algorithm does so in $O(\log n)$ rounds, with each round requiring linear time.*

We use the following standard lemma to claim that the graph we choose is an appropriate expander, and hence we can finish off the analysis of the decoding process using the previous lemma.

Lemma 2 *Let B be a bipartite graph formed as follows with n nodes on the left and αn nodes on the right, where $\alpha > 0$ is a fixed constant. Suppose that a degree is assigned to each node so that all left nodes have degree at least five, and all right nodes have degree at most C for some constant C. Suppose that a random permutation is chosen and used to match each edge out of a left node with each edge into a right node. Then, with $1 - O(1/n)$, for some fixed $\alpha > 0$, $\epsilon > 0$, and $\beta = 3/4 + \epsilon$, B is an (α, β) expander.*

We note that the restriction in Lemma 2 that the left degrees are at least five appears necessary. For example, it is entirely possible for random graphs with degree three on the left to fail to complete using the proposed sequential and parallel algorithms even after almost all nodes have been corrected. A problem occurs when the graph has a small even cycle. In this case, if all the nodes in the cycle are received incorrectly, the algorithm may fail to terminate correctly. (See Figure 2.) Even cycles of any constant length occur with constant probability, so errors remain with constant probability.

To circumvent this problem Gallager designs regular graphs with no small cycles [6]. To circumvent this problem in random graphs, we make a small change in the structure of the graph, similar to that in [7]. Suppose that we use the previous analysis to correct all but at most ηn message bits with high probability. We add an additional $\eta' n$ check nodes, where η' is a constant that depends on η, and construct a regular random graph with degree 5 on the left between all the n message nodes and the $\eta' n$ check nodes. The

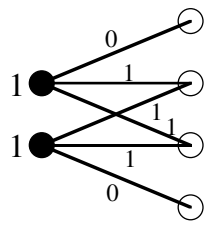

Figure 2: If the two left nodes are supposed to be 0, and all other nodes are correct, then the majority tells the left nodes not to change.

decoding proceeds as before on the original random graph, correcting all but at most ηn message bits. We then use the $\eta' n$ check nodes previously held in reserve to correct the remaining message bits using the Sipser-Spielman algorithm. That this procedure works follows directly from Lemmas 1 and 2. Moreover, as both η and η' can be made arbitrarily small by Corollary 1, the change in the rate of the code due to this additional structure is negligible, and is ignored in the sequel.

2.3 Theoretically Achievable Error Correction

For every rate, and for every possible left degree and corresponding right degree, the value of p^* can be computed by the above analysis. A natural question to ask is which regular code can achieve the largest value of p^*. Among rate 1/2 regular codes, it turns out that the largest p^* is achieved when all left nodes have degree 4 and all right nodes have degree 8, in which case $p^* \approx 0.0517$. Thus, combining Corollary 1, Lemma 1, and Lemma 2, we have shown that when the corresponding bipartite graph is chosen randomly this code can correct all errors with high probability when the initial fraction of errors approaches 0.0517. All of these regular codes run in linear time if we use the sequential decoding algorithm in the final stage. This follows from the fact that we need to run the hard decision decoding only for a constant number of rounds (at linear time per round), and then the sequential decoding algorithm can fix the remaining errors in linear time.

3 Irregular Codes

3.1 Intuition

Before we show how to derive irregular random graphs that improve upon the performance of Gallager's low-density parity-check codes, we offer some intuition as to why irregular graphs prove useful. It is convenient to think of the process as a game, with the message nodes and the check nodes as the players, and each player trying to choose the right number of edges. A constraint on the game is that the message nodes and the check nodes must agree on the total number of edges. From the point of view of a message node, it is best to have high degree, since the more information it gets from its check nodes the more accurately it can judge what its correct value should be. In contrast, from the point of view

of a check node, it is best to have low degree, since the higher the degree of a check node, the more likely it is to transmit incorrect guesses to the message node.

These two competing requirements must be appropriately balanced. If one allows irregular graphs, there is more flexibility in balancing these competing requirements. Message nodes with high degree tend to their correct value quickly. These nodes then provide good information to the check nodes, which subsequently provide better information to lower degree message nodes. Irregular graph constructions thus lead to a wave effect, where high degree message nodes tend to get corrected first, and then message nodes with slightly smaller degree, and so on down the line.

3.2 Analyzing Irregular Codes

We now describe a decoding algorithm for codes based on irregular graphs, or what we call *irregular codes*. Following the notation used in [7], 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 $\rho(x) := \sum_i \rho_i x^{i-1}$.

Our decoding algorithm in the case of irregular graphs is similar to Gallager's hard decision decoding as described in Section 2.1, but generalized to take into account the varying degrees of the nodes. Again we look at the process from the point of view of an edge (m, c). Consider the end of the ith round, and consider a check node c' of m other than c. The node c' sends m its correct value as long as there are an even number (including possibly 0) of other message nodes sending c' the wrong bit. As each bit was correctly sent to c' with probability p_i, it is simple to check that the probability that c' receives an even number of errors is

$$\frac{1 + \rho(1 - 2p_i)}{2}. \tag{5}$$

Equation 5 is the generalization of equation 1, taking into account the probability distribution on the degree of c'.

Also similarly to Section 2.1, a message node m of degree j passes its initial value along (m, c) to check node c unless at least b_j of the check nodes c' adjacent to m other than c send m the same value. Note that now the threshold value for a node depends on its degree. Also, the value of b_j may still change according to the round.

To analyze the decoding process, consider a random edge (m, c). The degree of m is j with probability λ_j. It thus follows from the same argument as in Section 2.1 that the recursive description for p_i is

$$\begin{aligned} p_{i+1} \quad = \quad & p_0 - \sum_{j=1}^{d_\ell} \lambda_j \left[p_0 \sum_{t=b_j}^{j} \binom{j-1}{l} \left[\frac{1+\rho(1-2p_i)}{2}\right]^t \left[\frac{1-\rho(1-2p_i)}{2}\right]^{j-1-t} + \right. \\ & \left. (1-p_0) \sum_{t=b_j}^{j-1} \binom{j-1}{t} \left[\frac{1-\rho(1-2p_i)}{2}\right]^t \left[\frac{1+\rho(1-2p_i)}{2}\right]^{j-1-t} \right]. \end{aligned} \tag{6}$$

We need to determine b_j so as to minimize the value of p_{i+1}. As in equation (4), the best value of b_j is given by the smallest integer that satisfies:

$$\frac{1-p_0}{p_0} \leq \left[\frac{1+\rho(1-2p_i)}{1-\rho(1-2p_i)}\right]^{2b_j-j+1}. \tag{7}$$

This equation has an interesting interpretation. Note that $2b_j - j + 1$ is a constant fixed by the above equation. The value $2b_j - j + 1 = b_j - (j - 1 - b_j)$ 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. We call this difference the *discrepancy* of a node. Equation (7) tells us that we need only check that the discrepancy is above a certain threshold to decide which value to send, regardless of the degree of the node.

3.3 Designing Irregular Graphs

We now describe techniques for designing codes based on irregular graphs that can handle larger probabilities of error at potentially some expense in encoding and decoding time. Given our analysis of irregular codes, our goal is to find sequences $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_{d_\ell})$ and $\rho = (\rho_1, \rho_2, \ldots, \rho_{d_r})$ that yield the largest possible value of p_0 such that the sequence of p_i decreases to 0 for a given rate. We frame this problem in terms of linear programs. Our approach cannot actually determine the best sequences λ and ρ. Instead, our technique allows us to determine a good vector λ given a vector ρ and the desired rate of the code. This proves sufficient for finding codes that perform significantly better than regular codes.

Let p_0 be fixed. For a given degree sequence $\rho = (\rho_1, \rho_2, \ldots, \rho_{d_r})$ let the real valued function $f(x)$ be defined by

$$\begin{aligned} f(x) \;=\; p_0 - \sum_{j=1}^{d_\ell} \lambda_j \Bigg[& p_0 \sum_{t=b_j}^{j-1} \binom{j-1}{t} \left[\frac{1+\rho(1-2x)}{2}\right]^t \left[\frac{1-\rho(1-2x)}{2}\right]^{j-1-t} + \\ & (1-p_0) \sum_{t=b_j}^{j-1} \binom{j-1}{t} \left[\frac{1-\rho(1-2x)}{2}\right]^t \left[\frac{1+\rho(1-2x)}{2}\right]^{j-1-t} \Bigg], \end{aligned}$$

where now

$$b_j = \left\lceil \left(j - 1 + \frac{\log((1-p_0)/p_0)}{\log((1+x)/(1-x))} \right) /2 \right\rceil$$

and the λ_j are variables to be determined. Observe that condition (6) now reads as $p_{i+1} = f(p_i)$. For a given p_0 and right hand degree sequence ρ, we are interested in finding a degree sequence $(\lambda_1, \ldots, \lambda_{d_\ell})$ such that the corresponding function $f(x)$ satisfies $f(x) < x$ on the open interval $(0, p_0)$. We begin by choosing a set L of positive integers which constitute the range of possible degrees on the left hand side. To find appropriate λ_ℓ, $\ell \in L$, we use the condition $f(x) < x$ above to generate linear constraints that the λ_ℓ must satisfy by considering different values of x. For example, by examining the condition at $x = 0.01$, we obtain the constraint $f(0.01) < 0.01$, which is linear in the λ_ℓ.

We generate constraints by choosing for x multiples of p_0/N for some integer N. We also include the constraints $\lambda_\ell \geq 0$ for all $\ell \in L$, as well as the constraint

$$\sum_{\ell \in L} \lambda_\ell / \ell = R \sum_i \rho_i / i,$$

where R is the rate of the code. This condition expresses the fact that the number of edges incident to the left nodes equals the number of edges incident to the right nodes. We then use linear programming to determine if suitable λ_ℓ exist that satisfy our derived constraints. The choice for the objective function is arbitrary as we are only interested in the existence of feasible solutions.

Given the solution from the linear programming problem, we can check whether the λ_ℓ computed satisfy the condition $f(x) < x$ on $(0, p_0)$. The best value for p_0 is found by binary search. Due to our discretization, there are usually some *conflict intervals* in which the solution does not satisfy this inequality. Choosing large values for the tradeoff parameter N results in smaller conflict intervals, although it requires more time to solve the linear program. For this reason we use small values of N during the binary search phase. Once a value for p_0 is found, we use larger values of N for that specific p_0 to obtain small conflict intervals. In the last step we get rid of the conflict intervals by slightly decreasing the value of p_0.

This linear programming tool allows for efficient search for good codes. That is, given a vector ρ we can find a good partner vector λ. Unfortunately, we do not yet have a means for determining which vectors ρ yield the best codes. Using the linear programming technique, however, we have considered graphs where the nodes on the left side may have varying degrees and the nodes on the right side all have the same degree. In other words, we have found good codes by considering ρ vectors with just one non-zero entry. As we shall see in Section 4, this suffices to find codes with significantly better performance than that given by codes determined by regular graphs.

It remains to show that the codes we derive in this manner in fact function as we expect. That is, given a vector $(\lambda_1, \ldots, \lambda_d)$, the right degree d_r, and the initial error probability p_0, if the sequence p_i given by equation (6) is monotonically decreasing and hence converges to 0, then the code obtained from the corresponding irregular random graph corrects a p_0-fraction of errors, with high probability. We first note that the equivalent of Theorem 1 holds in this case as well, by a similar proof (modified to take into account the different degrees). That is, we can use the hard decision decoding algorithm to decrease the number of erroneous bits down to any constant fraction.

To finish the decoding, we use the sequential algorithm from Section 2.2. The overall decoding time is linear.

Lemma 3 *Let $\alpha > 0$ and $\beta > 3/4 + \epsilon$ for some fixed $\epsilon > 0$. Suppose that B is an irregular bipartite (α, β) expander, and that d is the maximum degree on a left node of B. Then the sequential decoding algorithm corrects up to $\alpha n/2d$ errors in linear time.*

Proof: See the Appendix.

It follows that the irregular codes we derive function as we expect as long as our random graphs have sufficient expansion. This expansion property holds with high probability if we choose the minimum degree to be at least five. However, as stated previously, graphs with message nodes of smaller degree may be handled with a small additional structure in the graph.

Code Name	Left Degree Parameters
Code 14	$\lambda_5 = 0.496041$, $\lambda_6 = 0.173862$, $\lambda_{21} = 0.077225$, $\lambda_{23} = 0.252871$
Code 22	$\lambda_5 = 0.284961$, $\lambda_6 = 0.124061$, $\lambda_{27} = 0.068844$, $\lambda_{29} = 0.109202$, $\lambda_{30} = 0.119796$, $\lambda_{100} = 0.293135$
Code 10'	$\lambda_3 = 0.123397$, $\lambda_4 = 0.555093$, $\lambda_{16} = 0.321510$
Code 14'	$\lambda_3 = 0.093368$, $\lambda_4 = 0.346966$, $\lambda_{21} = 0.159355$, $\lambda_{23} = 0.400312$

Table 1: Parameters of our codes.

3.4 Theoretically Achievable Error Correction

We have designed some irregular degree sequences using the linear programming methodology described in subsection 3.3. The codes we describe all have rate 1/2. These codes perform well in practice as well as according to our theoretical model. However, it is likely that one could find codes that perform slightly better codes using our techniques. It is worth noting that Shannon upper bound (or entropy bound) for p^* for codes of rate 1/2 is 11.1%. Although the irregular codes we have designed to date are far from this limit, they are still much better than regular codes.

Two examples of codes we designed are Code 14 and Code 22. For Code 14 all nodes on the right have degree 14, and for Code 22 all nodes on the right have degree 22.* The remainder of the descriptions of Code 14 and Code 22 can be found in Table 1. In both these codes, the minimum degree on the left hand side is five. This ensures that the graphs have good expansion as needed in Lemma 2, and thus there is no need for the additional structure discussed in Section 2.2. Using the analysis of Section 3.2, we determine the appropriate value of p^* is approximately 0.0505 for Code 14 and 0.0533 for Code 22.

We can achieve even better performance by considering graphs with smaller degrees on the left. While such graphs do not have sufficient expansion for Lemma 2 to hold, we can use the additional structure discussed in Section 2.2 to finish the decoding. For Code 10' all nodes on the right have degree 10, and for Code 14' all nodes on the right have degree 14. The remainder of the descriptions of Code 10' and Code 14' can be found in Table 1. Using the analysis of Section 3.2, we determine the appropriate value of p^* is approximately 0.0578 for Code 10' and 0.0627 for Code 14'. Recall that 0.0517 is the best value of p^* that is possible using regular graphs for rate 1/2 codes.

4 Experimental Results

We include preliminary experimental results for new codes we have found using the linear programming approach. Our experimental design is similar to that of [15], whose results can be compared with ours.

We describe a few important details of our experiments and implementations. In our implementation, we simply run Gallager's decoding technique until it finishes, or until a prespecified number of rounds pass without success. In our experiments it turns out that it is unnecessary to switch to the modified decoding algorithm of Section 2.2 or use the additional

*Actually, to balance the number of edges, we do allow one node on the right to have a different degree.

structure described in Section 2.2, as in our experience the hard decision decoding algorithm of Gallager finishes successfully once the number of errors becomes small.

We do not perform an actual encoding, but instead for each trial use an initial message consisting entirely of zeroes. To more accurately compare code quality, instead of introducing errors with probability p, we set the same number of errors (corresponding to a fraction p of the block length) each trial. It is worthwhile to note that even when the decoding algorithm fails to decode successfully because too many rounds have passed, it can report that failure back. We have yet to see the decoding algorithm produce a codeword that satisfied all constraints but was not the original message, although theoretically it is a possible event.

Our implementation takes as input a *schedule* that determines the discrepancy value $2b_j - j + 1$ at each round. This schedule can be calculated according to equation (7). In practice, however, the schedule determined by equation (7) must be modified somewhat. If the discrepancy threshold is changed prematurely, before enough edges transfer the correct value, the decoding algorithm is significantly more likely to fail. Hence changing the threshold according to the round as given by equation (7) often fails to work well when the block size is small, since the variance in the number of edges sending the correct value can be significant. In practice we find that stretching out the schedule somewhat, so that the discrepancy threshold is changed after a few more rounds than the equations suggest, prevents this problem, at the expense of increasing the running time to the decoding algorithm.

In our experiments, a random graph was constructed separately for each trial at a certain error rate. No effort was made to test graphs or weed out potentially bad ones, and hence we expect that our results would be slightly better if several random graphs were tested and the best ones chosen. Following the ideas of [15] and [11], when necessary we remove double edges from our graphs.

4.1 Some Experiments

We first describe experiments on codes of rate 1/2 with 16,000 message bits and 8,000 check bits. In Figure 3, we describe the performance of Code 14 and Code 22 that we introduced in subsection 3.4. Each data point represents the results from 2,000 trials. Recall that the appropriate value of p^* is approximately 0.0505 for Code 14 and 0.0533 for Code 22. Recall that p^* represents the error rate we would expect to be able to handle for arbitrarily long block lengths, and that we only expect to approach p^* asymptotically in practice as the number of nodes grows.

Our results show that for block lengths of length 16,000 the codes appear to perform extremely well when a random fraction 0.045 (or 720) of the original message bits are in error. For the 2,000 trials, Code 14 never failed, and Code 22 failed just once. (In fact in 10,000 trials with this number of errors, Code 14 proved successful every time.) The probability that the code succeeds falls slowly as the error probability approaches p^*. Further experiments with larger block lengths demonstrate that performance improves with the number of bits in the message, as one would expect. These codes therefore perform better than the codes based on regular graphs presented in [15], albeit at the expense of a greater (but still linear) running time. They also perform much better than regular codes. For instance, as mentioned before, the best regular code of rate 1/2 is obtained from

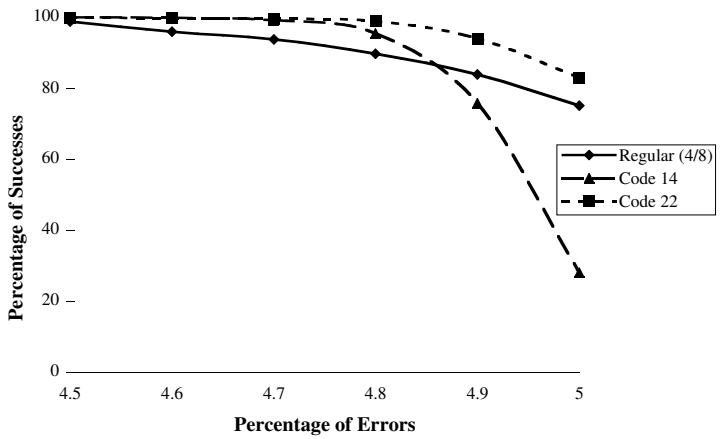

Figure 3: Percentage of successes based on 2000 trials.

random regular bipartite graphs with degree 4 on the left and degree 8 on the right. The performance of this code is also shown in Figure 3. Although the p^* value for this regular code is approximately 0.0517, in practice, with 16,000 message bits this regular code failed 23 times in 2,000 trials with a fraction of 0.045 errors.

We now consider Code 10' and Code 14' introduced in subsection 3.4. The experiments were run on 16,000 message bits and 8,000 check bits for 2,000 trials. In our experiments, we remove both double edges and some small cycles, as suggested in [11]. Recall that the appropriate value of p^* is approximately 0.0578 for Code 10' and 0.0627 for Code 14'. These codes again perform near what our analysis suggests, and they significantly outperform previous similar codes with similar decoding schemes, including regular codes.

In summary irregular codes Code 14 and Code 22 appear superior to any regular code in practice, and irregular codes Code 10' and Code 14' are far superior to any regular code. We have similarly found irregular codes that perform well at other rates.

5 Conclusion

We have proven that several classes of linear time error-correcting codes correct a large fraction of errors with high probability. We have also determined new codes based on irregular graphs that perform better than codes based on regular graphs on systems of practical size, as well as described a general technique for producing such codes.

Our work leaves several interesting open questions. One compelling open problem is to use our analysis to find good codes based on more general irregular random graphs, i.e., random graphs having more than one degree on the right. It seems quite possible that codes with significantly better performance than those we have tested here can be found. A more ambitious project is to fully analyze the behavior of these codes when using a decoding algorithm based on belief propagation. Such decoding algorithms are similar to the decoding algorithm of Gallager described in Section 2.1, except that more extensive information is passed through messages along the edges each round. Analyzing these algorithms would be

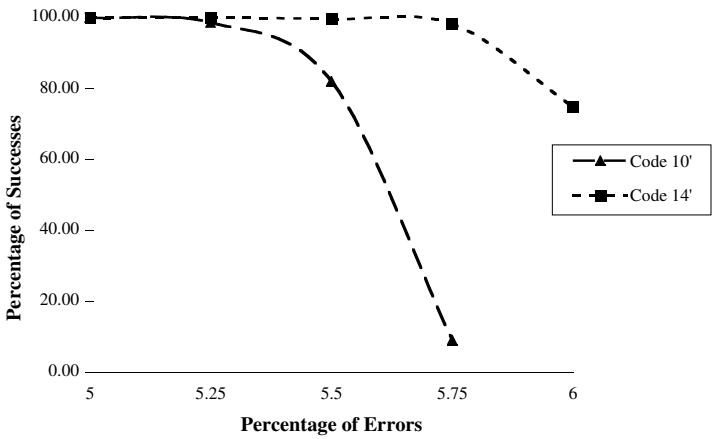

Figure 4: Percentage of successes based on 2000 trials.

a significant breakthrough in the theory of codes based on low-density parity-check matrices. Finally, an interesting question is to tie together more strongly the theory and practice of these codes. Our equations that describe the asymptotic behavior of the codes do not tell us which codes perform best for reasonably sized systems (say, with thousands or tens of thousands of bits). A more systematic approach rather than trial and error would be useful.

References

[1] C. Berrou, A Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes", *Proceedings of IEEE International Communications Conference, 1993.*

[2] J.-F. Cheng and R. J. McEliece, "Some High-Rate Near Capacity Codecs for the Gaussian Channel", *34th Allerton Conference on Communications, Control and Computing.*

[3] D. Divsalar and F. Pollara, "On the Design of Turbo Codes", *JPL TDA Progress Report 42-123.*

[4] G. D. Forney, Jr. "The Forward-Backward Algorithm", *Proceedings of the 34th Allerton Conference on Communications, Control and Computing, 1996*, pp. 432-446.

[5] B. J. Frey and F. R. Kschischang, "Probability Propagation and Iterative Decoding", *Proceedings of the 34th Allerton Conference on Communications, Control and Computing, 1996.*

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

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

[8] M. Luby, M. Mitzenmacher, and M. A. Shokrollahi, "Analysis of Random Processes via And-Or Trees", *Proc.* 9th *Symp. on Discrete Algorithms*, 1998.

[9] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved Low Density Parity Check Codes Using Irregular Graphs and Belief Propagation", submitted to the *1998 International Symposium on Information Theory.*

[10] 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.*

[11] 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**.

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

[13] R. Motwani and P. Raghavan, **Randomized Algorithms**, Cambridge University Press, 1995.

[14] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, 1988.

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

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

[17] N. Wiberg, "Codes and decoding on general graphs" Ph.D. dissertation, Dept. Elec. Eng, U. Linköping, Sweeden, April 1996.

A Proofs

Theorem 1 Let $i > 0$ be an integer constant and let Z_i be the random variable describing the fraction of edges set to pass incorrect messages after i rounds of the above algorithm. Further, let p_i be as given in the recursion (3). Then there is a constant c such that for any $\epsilon > 0$ and sufficiently large n we have

$$\Pr(|Z_i - p_i| > \epsilon) < \exp(-c\epsilon n).$$

Proof: We sketch the proof. There are two considerations requiring care. First, the neighborhood around a message bit m may not take the form of a tree. We show that this does not happen too often with an edge exposure martingale argument. Second, even assuming the number of non-trees is small, we still need to prove tight concentration of p_i around the expectation given that message bits may be wrong initially with probability p_0. This follows from a separate martingale argument, exposing the initial values at each node one by one.

For the first consideration, it is easily seen that there is a number γ depending on i and the maximum degree of the graph such that the probability that the neighborhood of depth $2i$ stemming from an edge is not a tree is γ/n. For sufficiently large n the value γ/n is less than $\epsilon/4$. Now by exposing the edges one by one using an edge exposure martingale and applying Azuma's inequality [13, Section 4.4] we see that the fraction of edges with non-tree neighborhoods is greater than $\epsilon/2$ with probability at most $\exp(-c\epsilon n)$.

Now let $\mathcal{Z}_i$ be the expected number of edges set to pass incorrect messages after i rounds. Then $|\mathcal{Z}_i - p_i| < \epsilon/2$ with high probability by the above. We can show that $\mathcal{Z}_i$ and Z_i are close using a martingale argument, exposing the initial values at the vertices one by one. Again using Azuma's inequality we obtain $\Pr(|\mathcal{Z}_i - Z_i| > \epsilon/2) \leq \exp(-c\epsilon n)$ for some constant c (depending on i). This now gives the assertion. Q.E.D.

Lemma 3 Let $\alpha > 0$ and $\beta > 3/4 + \epsilon$ for some fixed $\epsilon > 0$. Suppose that B is an irregular bipartite (α, β) expander, and that d is the maximum degree on a left node of B. Then the sequential decoding algorithm corrects up to $\alpha n/2d$ errors in linear time.

Proof: We follow Theorem 10 of [15]. We show that the number of unsatisfied check nodes decreases after each step in the sequential algorithm. Let V be the set of corrupt message nodes, with $|V| = v$ and $|\delta(V)| = \bar{d}v$. Suppose there are u unsatisfied check nodes and let s be the number of satisfied neighbors of the corrupt variables. By the expansion of B, we have

$$u + s > (3/4)\bar{d}v.$$

As each satisfied neighbor of V shares at least two edges with V, and each unsatisfied neighbor shares at least one, we have

$$\bar{d}v \geq u + 2s.$$

It follows that

$$u > \bar{d}v/2,$$

and hence there is some message node with more than 1/2 of its incident check nodes unsatisfied. Hence at each step the sequential algorithm may flip a message node and decrease the number of unsatisfied check nodes.

Therefore the only way the algorithm can fail is if the number of corrupt message nodes increases so that $v \geq \alpha n$ during the algorithm. But if $v \geq \alpha n$, then $u > \bar{d}\alpha n/2$. However, initially u is at most $vd < \alpha n/2$, and u decreases throughout the course of the algorithm, so this cannot happen. Q.E.D.