Improved Low Density Parity Check Codes Using Irregular Graphs[*]

Michael G. Luby[†] Michael Mitzenmacher[‡] M. Amin Shokrollahi[§]

Daniel A. Spielman[¶]

February 5, 1999

Abstract

We construct new families of error-correcting codes based on Gallager's low density parity check codes. We improve on Gallager's results by introducing irregular parity check matrices. Demonstrating that irregular structure can improve performance is a key point of our work.

We consider irregular codes under both belief propagation and hard decision decoding, obtaining improvements over previous results for both types of decoding. For belief propagation, we report experimental results of irregular codes on both binary symmetric channels and Gaussian channels. For example, using belief propagation, for rate 1/4 codes on 16,000 bits over a binary symmetric channel, previous low density parity check codes can correct up to approximately 16% errors, while our codes correct over 17%. In some cases our results come very close to reported results for turbo codes, suggesting that variations of irregular low density parity check codes may be able to match or beat turbo code performance.

We also study hard decision decoding, even though it does not perform as well as belief propagation, because it gives insight into the design of parity check codes. In particular, for hard decision decoding, we demonstrate how to prove performance guarantees for irregular parity check codes, extending the original work by Gallager. We also provide efficient methods for finding good irregular structures for hard decision decoding algorithms. Our rigorous analysis and our methodology for constructing good irregular codes constitute our other key contributions.

1 Introduction

In this paper, we construct new families of low density parity check codes, which we call *irregular codes*, that have significantly improved performance over previously known codes of this type. Our

[*]An extended abstract of this paper has appeared in [11].

[†]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`.

[‡]Harvard University. Most of this work was done while employed at Digital Equipment Corporation, Systems Research Center, Palo Alto, CA. E-mail: `michaelm@eecs.harvard.edu`.

[§]Bell Labs, 2C–353, 700 Mountain Ave, Murray Hill, NJ, 07974, USA. Research done while at the International Computer Science Institute Berkeley. Research financed in part by a Habilitationsstipendium of the Deutsche Forschungsgemeinschaft, Grant Sh 57/1–1. E-mail: `amin@research.bell-labs.com`.

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

codes can correct a significantly higher number of errors than previous low density parity check codes, albeit at the expense of a slightly slower running time. Our work significantly lowers the gap between the performance of low density parity check codes and the best known turbo codes, opening the question of whether further improvements may yield low density parity check codes with equivalent or better performance.

Low density parity check codes, introduced by Gallager in 1962 [7], and their performance under "belief propagation" decoding, has been the subject of much recent experimentation and analysis (e.g., [2, 3, 13, 14, 17, 20]). The interest in these codes stems from their near Shannon limit performance, their simple descriptions and implementations, and their amenability to rigorous theoretical analysis [7, 8, 9, 10, 11, 13, 20, 21]. Moreover, there appears to be a connection between these codes and turbo codes, introduced by Berrou, Glavieux, and Thitimajshima [1]. In particular, the turbo code decoding algorithm can be understood as a belief propagation based algorithm [12, 6], and hence any understanding of belief propagation on low density parity check codes may be applicable to turbo codes as well.

We find it helpful to describe low density parity check codes in terms of graphs. In the following we refer to the nodes on the left and the right 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 for each check node the exclusive-or of its incident message nodes is zero. (We note that our code construction can also be used to construct codes that can be encoded in linear time based on cascading series of bipartite graphs, as described for example in [19] or [8], but for convenience we will not address this issue here.) More specific details are given in Section 2.

Most previously studied low density parity check codes have been constructed using sparse regular, or nearly regular, random bipartite graphs [2, 7, 13, 14]. That is, the degrees of all message nodes are equal, and the degrees of all check nodes are equal. This means that the parity check matrix of the code described above contains the same number of ones in each row and the same number of ones in each column. We call these codes *regular codes*. Our improved performance comes from using codes based on *irregular* graphs. That is, the degrees of the nodes on each side of the graph can vary widely. In terms of the parity check matrix H, the weight per row and column is not uniform, but instead governed by an appropriately chosen distribution of weights. By carefully choosing the distributions, we achieve improved performance. In fact, the codes we describe use a number of largely disparate weights, suggesting that often the best distributions are far from those that produce regular codes.

As an example of our improved performance, we have found a rate 1/4 irregular code that, on 16,000 message bits, corrects over 17% random errors with high probability on our experiments. On 64,000 message bits, this code corrects up to 18% random errors on our experiments. In contrast, the best regular code corrects up to approximately 16.0% random errors with 16,000 message bits and approximately 16.2% on 64,000 message bits. (The Shannon bound for rate 1/4 codes is 21.45%.)

We provide some useful intuition for why irregular codes should provide better performance than regular codes in Section 3. However, that irregular structure improves performance is not surprising in light of recent work rigorously proving the power of irregular graphs in designing erasure codes [8, 9]. Irregular graphs appear to have been rarely studied in the setting of error-correcting codes because of the difficulty in determining what irregular structures might perform well. The erasure codes and the techniques for finding good erasure codes determined in [8, 9] provide the basis for some of the codes and the techniques we develop here.

Besides studying the performance of irregular codes under belief propagation, we also analyze irregular codes that use a simpler hard decision decoding scheme. The simplification arises by reducing the complexity of the messages being passed between nodes of the graph. The case of belief propagation corresponds to a real number being passed along each edge in each direction in each round. For our simpler hard decision decoding process, just one bit is passed in each direction in each round.

Hard decision decoding does not perform as well as belief propagation, as one might expect. Such schemes may still be useful in practice, since they are simpler and require less memory. Our main motivation for studying this model, however, is that we can make provable statements about the performance of hard decision decoding of irregular graphs. In doing so, we extend the previous analysis of Gallager, who provided an analysis for the case of regular graphs with no small cycle. In Section 5.1, we provide a new argument showing how to extend Gallager's analysis to random regular graphs. We then show that our analysis easily extends to randomly chosen irregular graphs, and describe how to find irregular graphs that lead to good codes.

We emphasize that our approach differs strongly form Gallager's original approach. Currently the only method we know for constructing irregular codes requires randomly choosing the corresponding irregular graph. However, the analysis used by Gallager does not directly apply to randomly chosen graphs; instead, Gallager relied on an explicit construction to obtain regular graphs with no small cycles. It is therefore important that our analysis applies to randomly chosen graphs. Our technique can be summarized as follows. Using ideas from [9] for studying random processes, we show that with high probability hard decision decoding successfully corrects all but an arbitrarily small constant fraction of the message bits. 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 [18], 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. Additionally, the bound on the probability of error we derive using this methodology improves upon the bound derived by Gallager for the regular graphs he explicitly constructed.

We note that since this work originally appeared in the papers [10] and [11], a great deal of progress has been made in this area. In particular, the work of Davey and MacKay demosntrates another approach to improving low density parity check performance by treating small numbers of bits as elements of an appropriate finite field [3]. By using irregular graphs and this technique, they have in some cases matched turbo code performance. More recent work by Richardson and Urbanke [17] extends our approach to message passing systems where a message can take on one of a finite number of values. Using this technique, they have come close to obtaining tight provable bounds on regular codes using belief propagation. Extending their and our ideas in order to find irregular graphs specifically designed to perform well using belief propagation appears just around the corner.

2 Low Density Parity Check Codes and Belief Propagation

We first review the low density parity check codes developed by Gallager and his suggested decoding algorithm, using the framework of MacKay and Neal [7, 13]. The parity check matrix H of the code is obtained by creating a matrix chosen uniformly (or near uniformly) at random such that the weight per column is a fixed constant and the weight per row is as uniform as possible. The decoding algorithm attempts to find the most probable vector $\mathbf{x}$ such that $H\mathbf{x} = 0 \bmod 2$.

In practice, the parity check matrix H can easily be constructed by randomly generating an appropriate bipartite graph. We describe the construction method. Message nodes are located on the left and the check nodes on the right. Each message node has a certain number of edges which connect to check nodes; similarly each check node has a certain number of edges connecting to message nodes. The total number of edges in the graph is e. A random permutation π of $\{1, \ldots, e\}$ is chosen, and then, for all $i \in \{1, \ldots, e\}$, the edge with index i out of the left side is identified with the edge with index π_i out of the right side. Note that this may potentially lead to multi-edges; often in practice multi-edges and small cycles can be removed to improve performance [13].

Gallager's decoding algorithm uses the idea of belief propagation. As explained in [13], the algorithm runs two alternating phases, in which for each non-zero entry in H with row i and column j (or, in other terms, for each edge of the associated bipartite graph) two values q_{ij} and r_{ij} are iteratively updated. The quantity q_{ij}^z approximates the probability that the jth bit of x is z, given the information obtained from all checks of j other than i. Similarly, the quantity r_{ij}^z approximates the probability that the ith check node is satisfied when the jth bit of x is z and all other message bits j' associated with check i have a separable distribution given by the appropriate $q_{ij'}$. That is, we assume that the other message bits j' are independently 1 with probability $q_{ij'}^1$, and use this to calculate r_{ij}^z. Over a binary symmetric channel with cross-over probability p, all q_{ij} have initially the value $1-p$. In the first phase of a round, all the r_{ij} values are updated in parallel; then in the second phase all the q_{ij} values are updated in parallel. (These parallel updates can also be simulated sequentially in a straightforward manner.) The total amount of work performed each round is linear in the number of edges of the graph. If the bipartite graph defined by H contains no cycles of length up to $2r$, then after r rounds of updates the algorithm produces the exact posterior probabilities that each message bit is in error based on the neighborhood within a diameter of $2r$ of the message node. The presence of cycles in the graph skews the probabilities, but in practice the effect on the algorithm appears to be small. More details can be found in [5, 7, 13, 20].

3 Irregular Graphs: Intuition

Before we demonstrate irregular random graphs that improve the performance of low-density parity check codes, we offer some intuition as to why irregular graphs should improve performance. Consider trying to build a regular low density parity check code that transmits at a fixed rate. 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 lower the degree of a check node, the more valuable the information it can transmit back to its neighbors.

These two competing requirements must be appropriately balanced. Previous work has shown that for regular graphs, low degree graphs yield the best performance [13, 14]. If one allows irregular graphs, however, there is significantly 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.

This intuition (which we observe in our experiments) unfortunately does not provide clues as to how to construct appropriate irregular graphs. Moreover, because belief propagation is not yet well understood mathematically, creating the proper irregular graphs appears a daunting challenge. We meet this challenge by using irregular graphs that have been proven to be effective for erasure codes that function in a similar manner. In the area of erasure codes, the mathematical framework has been established to both design irregular graphs and prove their effectiveness. Intuitively, graphs that work well for erasure codes should also work well for error-correction codes, since the two are closely related.

4 Irregular Graph Performance: Simulations

We describe a few important details of our experiments and implementations. We performed simulations using two types of channels for several rates and block lengths. The first channel we model is a binary symmetric channel. To more accurately compare code quality, instead of introducing errors with probability p, we introduced the same number of errors at each trial (corresponding to a fraction p of the block length). This procedure allows for easier comparison with other codes and minimizes the variance in the experiments that might arise from the variance in the number of errors. The second channel type we model is a white Gaussian channel with binary input ± 1 and an additive noise of variance σ^2. We report results for the Gaussian channel of rate R and additive noise σ^2 in terms of the signal to noise ratio $E_b/N_0 = 1/2R\sigma^2$ expressed as decibels ($10 \log_{10} E_b/N_0$). Here E_b represents the average energy per bit ($1/R$) and N_0 represents the noise spectral density ($2\sigma^2$).

Rather than encoding a message for each trial, we use an initial message consisting entirely of zeroes. Since the code is linear and the decoding algorithm respects its linearity, no generality is lost. In our experiments, we allowed the belief propagation algorithm to run for up to 200 rounds. If the algorithm failed to converge on a codeword within 200 rounds, a failure was reported. This was in fact the only failure we saw in our experiments; that is, the algorithm never returned a codeword that differed from the initial message.

A different random graph was conducted for each trial. No effort was made to test graphs and 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. Also, following the ideas of [13, 18], when necessary we remove double edges from our graphs.

We describe the irregular graphs used in Table 1, using the notation of [8]. We say that an edge has degree i on the left if its adjacent node on the left has degree i, and we similarly define an edge with degree i on the right. Our graphs are described in the following terms: for each graph, there is a corresponding left degree vector λ and a right degree vector ρ. The value λ_i represents the fraction of *edges* with degree i on the left, and similarly the value ρ_j represents the fraction of *edges* with degree j on the right. Note that given a vector λ and ρ one can construct a graph with (approximately) the correct edge fractions for any number of nodes, using the construction method described in Section 2. (Some care must be taken because of rounding and the necessity to have the number of edges on the left equal the number of edges on the right; however, this is easily handled.)

4.1 Binary Symmetric Channel

Table 2 compares the performance of regular and irregular codes of rates 1/2 and 1/4. Our results for regular codes (based on graphs in which all nodes on the left have degree 3) are slightly better than (but consistent with) previous results reported in [13]. In the table, n represents the block length, R represents the rate, f represents the fraction of errors introduced, and C represents the capacity of the binary symmetric channel with cross-over probability f. The results are reported

Code Rate 1/2	
Left Degrees	$\lambda_3 = 0.166600$, $\lambda_5 = 0.166600$, $\lambda_9 = 0.166600$, $\lambda_{17} = 0.166600$, $\lambda_{33} = 0.166600$, $\lambda_{65} = 0.166700$
Right Degrees	$\rho_7 = 0.154091$, $\rho_8 = 0.147486$, $\rho_{19} = 0.121212$, $\rho_{20} = 0.228619$, $\rho_{84} = 0.219030$, $\rho_{85} = 0.129561$
Code Rate 1/4	
Left Degrees	$\lambda_3 = 0.166600$, $\lambda_5 = 0.166600$, $\lambda_9 = 0.166600$, $\lambda_{17} = 0.166600$, $\lambda_{33} = 0.166600$, $\lambda_{65} = 0.166700$
Right Degrees	$\rho_4 = 0.160416$, $\rho_{10} = 0.404478$, $\rho_{33} = 0.303338$, $\rho_{34} = 0.131768$

Table 1: Parameters of our codes.

in terms of the number of trials, or blocks decoded, and the number of errors, or the number of blocks for which the decoding algorithm failed to find a solution within 200 rounds.

At rate 1/2, our irregular codes perform only slightly better than the regular codes at block lengths of 16,000 bits. They do appear notably more robust at higher error rates, however. At 64,000 bits, our code can handle over a half a percent more errors. While it has been previously noted that low density parity check codes perform better as the block length increases [13], we believe that this effect is magnified for our irregular codes, because the degrees of the nodes can be quite high. For example, our irregular rate 1/2 codes have nodes on the left of degree 65 and nodes on the right of degree 85.

At rate 1/4, we have different irregular codes with lower degrees. This code greatly outperforms the regular codes, even at block lengths of 16,000, where they correct approximately 1% more errors. At block lengths of 64,000 bits, the effect is even more dramatic, and the irregular codes appear to correct more than 1% more errors. We note that initial experiments at other rates further validate our contention that irregular codes can outperform regular codes in terms of the number of errors that can be corrected.

For both regular and irregular codes, the number of operations required is proportional to the product of the number of edges in the corresponding graph and the number of rounds until the process terminates. The irregular graphs have approximately 2.5 times as many edges as the regular graphs, and at higher error rates they can take approximately 1.5 times as many iterations to complete. Hence it takes approximately 4 times as many operations to decode at higher rates. In practice, performance can actually be worse than this, however, since the larger graph size for the irregular codes requires more accesses to slower levels of the memory hierarchy. However, we believe the slower running time is not dramatic in light of the improved performance.

4.2 Gaussian Channel

Figures 1 and 2 compare the performance (in terms of the bit error rate (BER)) of irregular codes of rate 1/2 and 1/4 with reported results for turbo codes [4] and regular codes [14] at these rates. Again, our experiments were with block lengths of 16,000 bits, and for this block length each data point is the result of 10,000 trials. (We compare with results using comparable block lengths. The results from [4] are available at `http://www331.jpl.nasa.gov/public/TurboPerf.html`. The results we report from [14] are only approximate, as [14] does not provide actual numbers but only a graph.) For our irregular codes, the belief propagation algorithm terminated after 200 rounds if the solution was not found.

For rate 1/2 codes, our irregular codes perform notably better than regular codes, greatly

	n	R	f	C	errs	trials
Reg	16000	0.5	0.078	0.605	0	10000
		0.5	0.080	0.598	35	10000
		0.5	0.082	0.591	1033	10000
Irreg	16000	0.5	0.078	0.605	1	10000
		0.5	0.080	0.598	14	10000
		0.5	0.082	0.591	40	10000
		0.5	0.084	0.584	116	10000
Reg	64000	0.5	0.082	0.590	1	1000
		0.5	0.084	0.583	249	1000
Irreg	64000	0.5	0.086	0.577	0	1000
		0.5	0.088	0.570	0	1000
		0.5	0.090	0.563	25	1000
Reg	16000	0.25	0.158	0.370	0	10000
		0.25	0.160	0.366	0	10000
		0.25	0.162	0.361	45	10000
		0.25	0.164	0.356	697	10000
		0.25	0.166	0.352	3767	10000
Irreg	16000	0.25	0.166	0.352	0	10000
		0.25	0.168	0.347	0	10000
		0.25	0.170	0.342	4	10000
		0.25	0.172	0.338	15	10000
		0.25	0.174	0.333	53	10000
Reg	64000	0.25	0.164	0.356	0	1000
		0.25	0.166	0.352	176	1000
Irreg	64000	0.25	0.178	0.324	0	1000
		0.25	0.180	0.320	2	1000
		0.25	0.182	0.316	63	1000

Table 2: Comparing regular and irregular graphs.

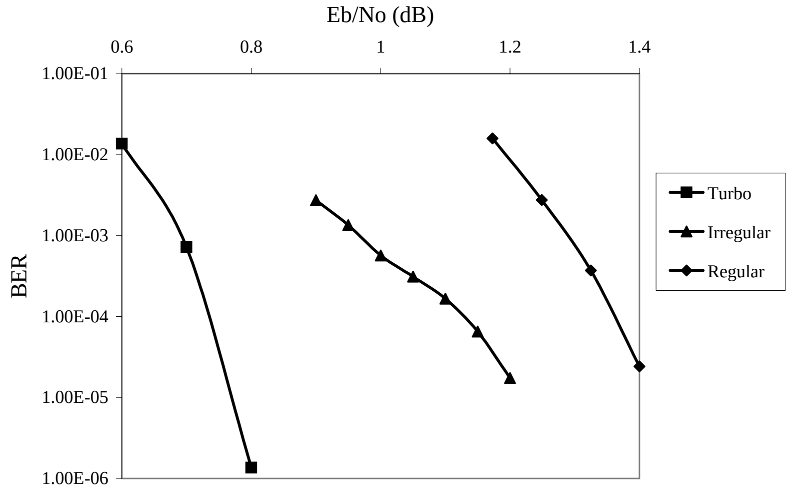

Figure 1: Irregular codes vs. regular codes and turbo codes: rate 1/2.

reducing the gap between the performance of low density parity check codes and turbo codes. This gap is further reduced when we move to larger block sizes, as our codes prove to perform better for larger block lengths in this setting as well. At block lengths of 64,000 bits, our code never failed in 1,000 trials at both 0.95 dB and 1.00 dB. Our estimates for the bit error rate from 1,000 trials at 0.9 dB is $3.97 \cdot 10^{-5}$ and at 0.85 dB is $6.03 \cdot 10^{-5}$. Again, this is much better than the performance of regular codes at a comparable block length presented in [14].

Our results for irregular codes at rate 1/4 (Figure 2) similarly show significant improvement over regular codes. At this lower rate and block length, however, turbo codes appear to have a significant edge. We believe this edge can be reduced with further exploration and experimentation.

Our irregular codes at this rate again perform significantly better with larger block lengths. At block lengths of 64,000 bits, our code never failed in 1,000 trials at both 0.70 dB and 0.60 dB. Our estimates for the bit error rate from 1,000 trials at 0.50 dB is $6.18 \cdot 10^{-4}$ and at 0.40 dB is $8.39 \cdot 10^{-3}$.

5 Analyzing Message Passage Decoding

The experiments and experience of the previous section demonstrate the effectiveness of using irregular graphs in conjunction with belief propagation. Belief propagation, however, proves difficult to analyze. As a first step in the direction of analyzing belief propagation, we consider a simplified message-passing algorithm where in each round one bit, instead of a real number, is passed in each direction along each edge. This message-passing scheme was analyzed for specific regular codes by Gallager. Our new analysis extends to random regular and irregular graphs. As with general belief propagation, we find that using irregular graphs can improve performance of this decoding scheme.

5.1 Regular Graphs

As described in the Introduction, a bipartite graph with n message nodes on the left and r check 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

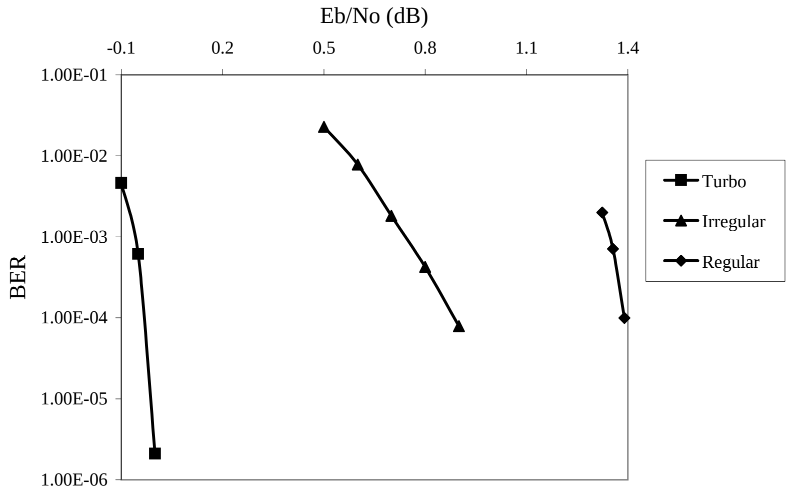

Figure 2: Irregular codes vs. regular codes and turbo codes: rate 1/4.

$\mathbf{x} = (x_1, \ldots, x_n)$ 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 [19] or [8]. 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 again 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 [19, 8].

We now review the hard decision decoding approach taken by Gallager in his original analysis [7].

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 3. 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:

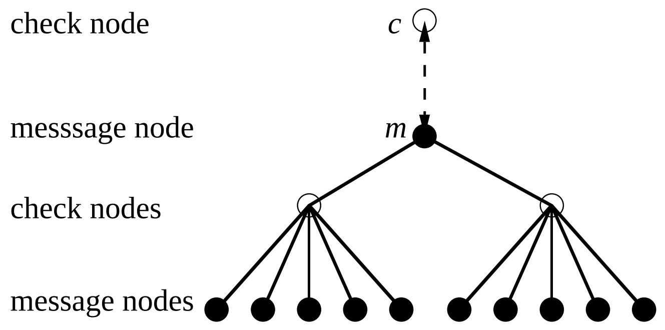

Figure 3: Representing the code as a tree.

```
• 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. Indeed, this algorithm is entirely similar to a belief propagation based algorithm, except that bits are passes instead of probabilities, and hence no calculations of a posteriori probabilities is necessary.

The process can run for a preset number of rounds, after which each message node can determine its most likely value based on its neighbors. If the check nodes are satisfied, then a codeword has been found; otherwise the decoding has failed. Alternatively, after each round, each message node can determine its most likely value and a check can be performed to see if a codeword has been found. If not, the process continues until the decoder decides to stop with a failure.

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

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

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 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 5.2 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_i (to be determined below) that depends on the round number. For each message node m and neighboring check node c, if at least b_i 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.

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

We choose b_i 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_i. As determined by Gallager, the correct choice of b_i is the smallest integer that satisfies

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

Note that b_i 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 [9] 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: We sketch the proof. (A full proof, containing all of the details fully specified, has appeared in the subsequent work of Richardson and Urbanke [17]; we refer the interested reader there.) 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 [15, 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.

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.*

5.2 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 [18, 19].

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 [18], 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 [18] 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 [18] 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 [18].

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 a linear amount of work.*

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.*

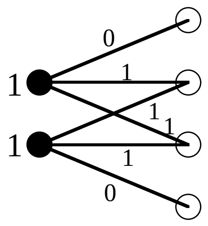

Figure 4: If the two left nodes are supposed to be 0, and all other nodes are correct, then the majority tells the left nodes not to change.

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 4.) 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 [7]. To circumvent this problem in random graphs, we make a small change in the structure of the graph, similar to that in [8]. 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 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.

It is worth noting that since explicit constructions are known for regular expanders, using the previous analysis (Theorem 1 and Lemma 1) we may construct regular codes with the same asymptotic performance as Gallager's regular codes that are guaranteed to work with probability exponential in n. Gallager proved that his codes and decoding algorithm worked correctly with probability exponential in a root of n. Hence our proof yields slightly better bounds on the error probability in this case.

5.3 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.

6 Irregular Codes

6.1 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 [8], 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 5.1, 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 5.1, after round i a message node m of degree j passes its initial value along (m, c) to check node c unless at least $b_{i,j}$ 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 $b_{i,j}$ changes according to the round.

To analyze the decoding process, consider a random edge (m, c). The left degree of (m, c) is j with probability λ_j. It thus follows from the same argument as in Section 5.1 that the recursive description for p_i is

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

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

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

This equation has an interesting interpretation. Note that $2b_{i,j} - j + 1$ is a constant fixed by the above equation. The value $2b_{i,j} - j + 1 = b_{i,j} - (j - 1 - b_{i,j})$ 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.

6.2 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. (Similarly, we may also apply this technique to determine a good vector ρ given a vector λ and the desired rate; as we explain below, however, this does not prove useful in this setting.)

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

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

where now

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

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

$$\sum_{\ell \in L} \lambda_\ell / \ell = R \sum_i \rho_i / i, \tag{8}$$

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. In a similar fashion, we can similarly find a good partner vector ρ from a given λ. However, our experiments reveal that the best ρ vector for this decoding algorithm is always the one where are the nodes on the right have the same degree (or all nodes have as close to the same degree as possible).

There is a natural intuition explaining this phenomenon. ¿From the point of view of a message node m, it appears best if the expected number of other neighbors a neighboring check node c has is as small as possible. This can be seen as follows. At the end of the ith round, the probability that c sends the correct vote to m is $\frac{1+\rho(1-2p_i)}{2}$. For small p_i values, this is approximately $1 - p_i \sum_{i=1}^{d_r}(i-1)\rho_i$. To maximize this probability, we seek to minimize $\sum_{i=1}^{d_r}(i-1)\rho_i$, which is exactly the expected number of other neighbors c has. This quantity is minimized (subject to the constraints $\sum_{i=1}^{d_r}\rho_i = 1$ and equation (8)) when all check nodes have equal degree, or as nearly equal as possible. In constrast, we note that using varying degrees for the check nodes is advantageous when using a more complicated decoding algorithm based on belief propagation [11].

Using the linear programming technique, 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 7, 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 5.2. 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: We follow Theorem 10 of [18]. 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

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

Code Name	**Right Degree**	**Left Degree Parameters**	**Value of p^***
Code 14	14	$\lambda_5 = 0.496041$, $\lambda_6 = 0.173862$, $\lambda_{21} = 0.077225$, $\lambda_{23} = 0.252871$	0.0505
Code 22	22	$\lambda_5 = 0.284961$, $\lambda_6 = 0.124061$, $\lambda_{27} = 0.068844$, $\lambda_{29} = 0.109202$, $\lambda_{30} = 0.119796$, $\lambda_{100} = 0.293135$	0.0533
Code 10'	10	$\lambda_3 = 0.123397$, $\lambda_4 = 0.555093$, $\lambda_{16} = 0.321510$	0.0578
Code 14'	14	$\lambda_3 = 0.093368$, $\lambda_4 = 0.346966$, $\lambda_{21} = 0.159355$, $\lambda_{23} = 0.400312$	0.0627

Table 3: Parameters of our codes.

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.

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.

6.3 Theoretically Achievable Error Correction

We have designed some irregular degree sequences using the linear programming methodology described in subsection 6.2. 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.

Code 14 and Code 22, described fully in Table 3 are two irregular codes that we designed. For Code 14 all nodes on the right have degree 14, and for Code 22 all nodes on the right have degree 22.[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 5.2.

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 5.2 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. Recall that 0.0517 is the best value of p^* that is possible using regular graphs for rate 1/2 codes.

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

7 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 [18], 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 pre-specified 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 5.2 or use the additional structure described in Section 5.2, 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 $2b_{i,j} - j + 1$ 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 of 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 [18] and [13], when necessary we remove double edges from our graphs.

7.1 Some Experiments

We first describe experiments on codes of rate 1/2 with 16,000 message bits and 8,000 check bits. In Figure 5, we describe the performance of Code 14 and Code 22 that we introduced in subsection 6.3. 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 [18], albeit at

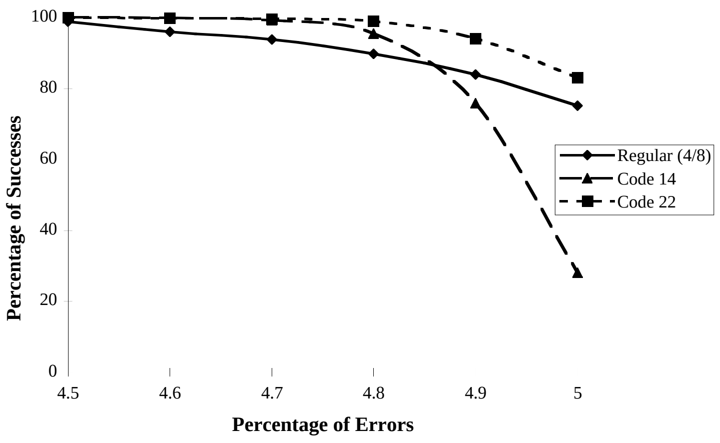

Figure 5: Percentage of successes based on 2000 trials.

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 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 5. 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 6.3. 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 [13]. 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.

8 Conclusion

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, 1996.*

[3] M.C. Davey and D. J. C. MacKay, "Low Density Parity Check Codes over GF(q)", available from **http://wol.ra.phy.cam.ac.uk/mackay**.

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

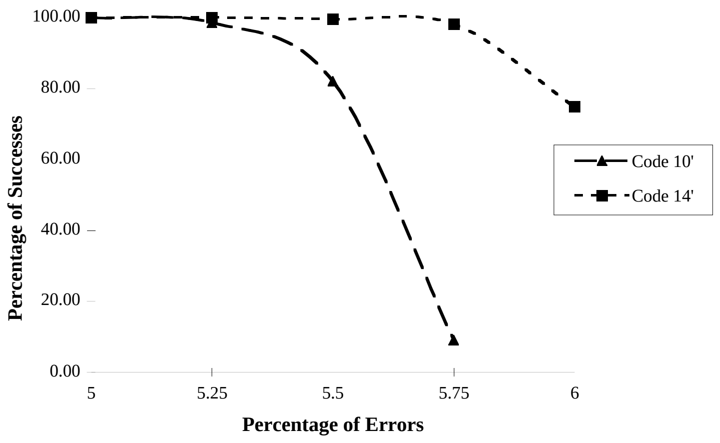

Figure 6: Percentage of successes based on 2000 trials.

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

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

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

[8] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical Loss-Resilient Codes", in *Proceedings of the* 29^{th} *Annual Symposium on Theory of Computing*, pp. 150–159, 1997.

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

[10] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman, "Improved Low-Density Parity-Check Codes Using Irregular Graphs and Belief Propagation", in Proceedings of the 1998 International Symposium on Information Theory.

[11] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman, "Analysis of Low Density Codes and Improved Designs Using Irregular Graphs", in *Proceedings of the* 30^{th} *Annual Symposium on Theory of Computing*, pp. 249–258, 1998.

[12] 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.*

[13] 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**.

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

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

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

[17] T. Richardson and R. Urbanke, "The Capacity of Low-Density Parity Check Codes under Message-Passing Decoding", preprint, available at `http://cm.bell-labs.com/cm/ms/who/ruediger/pub.html`.

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

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

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

[21] N. Wiberg, H.-A. Loeliger and R. Kötter, "Codes and iterative decoding on general graphs", *European Transactions on Telecommunications*, vol. 6, September 1995, pp. 513-526.