Improved Low-Density Parity-Check Codes Using Irregular Graphs

Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, and Daniel A. Spielman

***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 and a new rigorous analysis of hard-decision decoding of these codes. We also provide efficient methods for finding good irregular structures for such decoding algorithms. Our rigorous analysis based on martingales, our methodology for constructing good irregular codes, and the demonstration that irregular structure improves performance constitute key points of our contribution.**

We also consider irregular codes under belief propagation. We report the results of experiments testing the efficacy of irregular codes on both binary-symmetric 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.

***Index Terms*—Belief propagation, concentration theorem, Gallager codes, irregular codes, low-density parity-check codes.**

I. Introduction

LOW-density parity-check codes, introduced by Gallager in 1962 [7], have been the subject of much recent experimentation and analysis (e.g., [3], [4], [16], [17], [20], [21], [24]). 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], [10]–[13], [16], [20], [21], [24], [25], [27]. Moreover, there are connections between these codes and turbo codes, introduced by Berrou, Glavieux, and Thitimajshima [1], as the latter can be described in the framework of low-density parity-check codes (see, e.g., [14]). Moreover, the turbo decoding algorithm can be understood as a belief propagation based algorithm [15], [9], 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 bipartite 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 $\boldsymbol{x} = (x_1, \ldots, x_n)$ is a codeword if and only if $H\boldsymbol{x} = 0$, 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. (We note that our methodology can also be used to construct codes that can be encoded in linear time with similar rate and error-correction threshold by using a cascading series of bipartite graphs, as described in [10], [27]. For convenience, we will not address this issue here.) More specific details are given in Section II-A.

Most previously studied low-density parity-check codes have been constructed using sparse regular, or nearly regular, random bipartite graphs [3], [7], [16], [17]. 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.

That irregular structure improves performance is not surprising in light of recent work rigorously proving the power of irregular graphs in designing erasure correcting codes [10], [11], [27]. Irregular graphs appear to have been rarely studied

Manuscript received April 22, 1999; revised September 8, 2000. The work of M. G. Luby was supported in part by the National Science Foundation under Operating Grant NCR-9416101. The work of M. Mitzenmacher was supported in part by the Alfred P. Sloan Research Fellowship and by the National Science Foundation CAREER Grant CCR-9983832. The work of M. A. Shokrollahi was partially supported by Habilitationsstipendium of the Deutsche Forschungsgemeinschaft under Grant Sh 57/1–1. The work of D. A. Spielman was supported in part by the National Science Foundation under Mathematical Sciences Postdoctorla Fellowship, NSF CAREER Award CCR-9701304, and a Sloan/Cabot Award from the MIT School of Science. The material in this paper was presented in part at the 20th Annual ACM Symposium on Theory of Computing, Dallas, TX, May 23–26, 1998.

M. G. Luby and M. A. Shokrollahi were with the International Computer Science Institute, Berkley, CA and Digital Equipment Corporation, Systems Research Center, Palo Alto, CA. They are now with Digital Fountain, Inc., San Francisco, CA 94110 USA (e-mail: luby@digital.fountain.com; amin@digital.fountain.com).

M. Mitzenmacher was with the Digital Equipment Corporation, Systems Research Center, Palo Alto, CA. He is now with Harvard University, Cambridge, MA 02138 USA (e-mail: michaelm@eecs.harvard.edu).

D. A. Spielman is with the Department of Mathematics, the Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: spielman@math.mit.edu).

Communicated by F. R. Kschischang, Associate Editor for Coding Theory.

Publisher Item Identifier S 0018-9448(01)00736-2.

0018–9448/01$10.00 © 2001 IEEE

in the setting of error-correcting codes because of the difficulty in determining what irregular structures might perform well. The techniques for finding good erasure correcting codes determined in [10], [11], [27] provide the basis for some of the codes and the techniques we develop here.

In the following, we would like to 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 [16], [17]. If one allows irregular graphs, however, there is significantly more flexibility in balancing these competing requirements. There is reason to believe that a wide spread of degrees, at least for message nodes, could be useful. Message nodes with high degree tend to correct their 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 have the potential to 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. We meet this challenge in two ways. First, we design a rigorous analysis for both regular and irregular graphs for a hard-decision decoding algorithm also suggested by Gallager. Even though these decoders do 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* asymptotic statements about the performance of hard-decision decoding of irregular graphs. Using ideas from [11] for studying random processes, we show in Section II-A 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 [22], and prove in Section II-B that this new hybrid method successfully finishes the decoding with high probability. This analysis easily extends to the irregular codes that we introduce in Section III. 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 emphasize that our approach differs strongly from Gallager's original approach, since we do not assume our graphs lack small cycles. Instead, our analysis applies to randomly chosen graphs.

From our analysis, we develop in Section III-B methods based on linear programming to find good irregular graph structures using the hard-decision decoding algorithm. The corresponding degree distributions have been tested extensively, and we report on some of these tests in Section IV.

The second way in which we meet the challenge of designing irregular graphs is to test the belief propagation algorithm on graphs that have been proven to be effective for erasure-correcting codes [10], [11], [27]. Intuitively, graphs that work well for erasure-correcting codes should also work well for error-correction codes, since the two are closely related.

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 in our experiments. On 64 000 message bits, a similar 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 report on our experiments and simulations with the belief propagation algorithm in Section V-A.

We note that since this work originally appeared in [12] and [13], a great deal of progress has been made in this area. In particular, the work of Davey and MacKay demonstrates another approach to improving low-density parity-check performance by treating small numbers of bits as elements of an appropriate finite field [4]. By using irregular graphs and this technique, they have in some cases matched turbo code performance. More recent work by Richardson and Urbanke [21] and Richardson, Shokrollahi, and Urbanke [20] extends our analysis in Section II-A to message-passing systems where a message can take on one of a finite number of values. Using their extensions, they have obtained nearly tight provable bounds on regular and irregular codes using belief propagation and have developed techniques for designing irregular graphs that perform well under belief propagation.

II. Analyzing Message Passage Decoding

In this section, we consider a message-passing algorithm where in each round one bit 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. We demonstrate that using irregular graphs can greatly improve performance of this decoding scheme. To ease the presentation, we first detail our arguments for regular graphs.

A. Regular Graphs

As described in Section I, 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 $\boldsymbol{x} = (x_1, \ldots, x_n)$ is a codeword if and only if $H\boldsymbol{x} = 0$, 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. To allow encoding in linear time, one could allow the nodes on the right to represent bits rather than restrictions, and then use a cascading series of bipartite graphs, as described for example in [10], [27]. 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. The resulting code has the same rate and error-correction threshold as the corresponding low-density parity-check code, although its likelihood of decoding error increases.

In what follows, we again focus on one bipartite graph only, and assume that only the message nodes 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 [10], [27].

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. The bit sent from a message node m to a check node c at the ith step of the decoding is denoted $g^i_{m,c}$, while the message sent from the check node c to the message node m at round i is denoted $g^i_{c,m}$. The bit $g^i_{m,c}$ is a guess of the correct bit of message bit m at round i. Similarly, $g^i_{c,m}$ is a guess, from the point of view of the check node c, of what the correct value of m should be. The messages passed contain only *extrinsic information*, that is, the value of $g^i_{m,c}$ depends only on the values $g^i_{c',m}$ for all check nodes c' incident to m *other than* c. (Similarly, for $g^i_{m,c}$.) 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.) We assume that in the zeroth round of the process messages are sent from message nodes to check nodes. Each subsequent round consists of passing messages from check nodes to message nodes and back. In full detail, each round consists of an execution of the script at the bottom of this page.

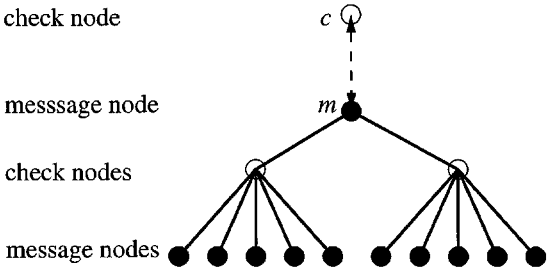

Fig. 1. Representing the code as a tree.

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.

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.

To analyze 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 Fig. 1. For now let us assume that the neighborhood of m is accurately described by a tree for some fixed number of rounds.

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

```
•For all edges $(m, c)$ do the following in parallel:
    Update for $g^i_{m,c}$
  −If this is the zeroth round, then set $g^0_{m,c} := r_m$.
  −If this is the $i$th round with $i > 0$, then $g^i_{m,c}$ is computed as follows:
  $*$ if $g^i_{c',m}$ equals $b$ for all adjacent check nodes $c'$ of $m$ other than $c$, then set $g^i_{m,c} := b$.
  $*$ else set $g^i_{m,c} := r_m$.
    Update for $g^i_{c,m}$
  −For all edges $(m, c)$ set $g^i_{c,m}$ as the exclusive-or of the values $g^{i-1}_{m',c}$
  where $m'$ ranges over all adjacent message nodes of $c$ other than $m$.
```

nodes other than m sending c' the wrong bit. As each bit was correctly sent to c' with probability $1-p_i$, 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

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

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 II-B, 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 improved 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.[1] Using the same analysis as for (2), we may find a recursive description of the p_i

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

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

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

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 [11] that the recursive description given by (3) is correct with high probability over any constant number of rounds. (We note also that a similar extension of this proof based on the original paper [13] has also appeared in the subsequent work of Richardson and Urbanke [21].)

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 η (depending on the maximum degree $\max\{d_\ell, d_r\}$) such that for any $\epsilon > 0$ and sufficiently large n we have

$$\Pr(|Z_i - p_i| > \epsilon) < \exp(-c\epsilon^2 n).$$

Proof: Let M be the number of edges in the graph. We show the equivalent assertion

$$\Pr(|MZ_i - Mp_i| > M\epsilon) < \exp(-c\epsilon^2 M).$$

[1] The first algorithm, where all of the other neighbors of a message node must disagree with the received bit for it to change its message, is nowadays referred to as Gallager's Algorithm A. The improvement is often referred to as Gallager's Algorithm B. Our experiments and analysis apply to Gallager's Algorithm B in the most general sense; that is, for any predetermined values b_i. Of course Gallager's Algorithm A is then just a special case.

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 nontrees is small, we still need to prove tight concentration of p_k 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.

First, we consider the number of edges (m, c) such that if we expand the neighborhood below m for $2i$ levels, we do not obtain a tree. For such edges we cannot say anything about their behavior, so we must show that there are few of them. Note that as the number of nodes in a tree of $2i$ levels is exponential in i, this necessarily implies that the number of nodes n in the graph must be exponential in i. Recall that in the statement of the theorem, however, i is a fixed constant and n is taken to be sufficiently large.

It is easily seen that there is a constant 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. To see this, consider the neighborhood stemming from an edge by expanding outward level by level, one edge at a time. As there are fewer than $(d_l\, d_r)^i$ total nodes in the tree, the probability at any step that an edge in the neighborhood hits a vertex already in the neighborhood is bounded above by $d_l\, d_r(d_l\, d_r)^i/(n-(d_l\, d_r)^i)$. From a union bound, the total probability that the neighborhood fails to be a tree is therefore bounded above by

$$(d_l\, d_r)^{2i+1}/(n-(d_l\, d_r)^i) < \gamma/n$$

for a suitable constant γ. Hence the expected number of edges that might fail because their neighborhood structure is not a tree is only a constant. More concretely, for sufficiently large n the value γ/n is less than $\epsilon/4$. Hence, if we let M^* be the number of edges (m, c) for which the neighborhood of up to $2i$ levels is a proper tree, we obtain

$$\boldsymbol{E}[M^*] \geq M(1-\epsilon/4).$$

We now obtain a concentration result for M^*, by exposing the edges of the graph one by one using an edge exposure martingale and applying Azuma's inequality [18, Sec. 4.4]. In particular, we think in terms of exposing the permutation π that defines our bipartite graph one entry at a time, in order. We may then define Z_i to be the expected value for M^*, given the results of the first i exposures. In particular, $Y_0 = \boldsymbol{E}[M^*]$, $Y_M = M^*$, and the sequence Z_k forms a standard Doob's martingale, with $\boldsymbol{E}[Y_{k+1}|Y_k] = \boldsymbol{E}[Y_k]$. Moreover, consecutive values of Y_k differ only by a constant, as we show in the following lemma. Hence, using Azuma's inequality

$$\Pr(|M^* - M| > M\epsilon/2) < \exp(-\eta_1 \epsilon^2 M). \qquad (5)$$

Lemma 1: $|Y_{k+1} - Y_k|$ is bounded by a constant.

Proof: Consider all possible results from exposing the $(k+1)$st edge. The value of $|Y_{k+1} - Y_k|$ is bounded by the maximum difference in the expectation of M^* from any two such results. This, in turn, is bounded by the maximum difference in the value of M^* between any two permutations that differ in the placement of two edges. That is, consider two possible results: $\pi_1(k+1) = x$ and $\pi_2(k+1) = y$. There is a one-to-one correspondence between the remaining possibilities for π_1 and π_2, such that for some $j > k+1$, the correspondence has $\pi_1(j) = y$, $\pi_2(j) = x$, and π_1 and π_2 agree at all other places. Hence, the expectation over the two possibilities given by π_1 and π_2 differs at most by the maximum difference in the value of M^* between any two permutations that differ in the placement of two edges.

Now consider any pair of graphs given by permutations π and σ, where π and σ differ only on the placement of two edges. In this case, the difference in M^* for σ and π is bounded by a constant, since the placement of these edges can only affect a constant number of trees (this constant depending on i and the maximum degree). Hence the lemma is proved. □

Now let M' be the number of edges from the M^* edges with valid tree neighborhoods for $2i$ levels below set to pass incorrect messages after i rounds. Clearly, $\boldsymbol{E}[M'] = M^* p_i$. We again obtain a high probability result using a martingale argument. We may reveal the initial value received at each node, one at a time. Again we may define Y_k to be the expected value for M', given the results of the first k exposures, in which case the Y_k form a standard Doob's martingale. Here, it is easy to see that consecutive values of Y_k differ only by a constant, as each revealed node can only affect the edges where in lies in the corresponding tree. Hence

$$\Pr(|M' - M^* p_i| > M\epsilon/2) < \exp(-\eta_2 \epsilon^2 M). \qquad (6)$$

The assertion follows from the two inequalities (5) and (6), as

$$M' \leq M Z_i \leq M' + |M^* - M|$$

and hence

$$\Pr(|M Z_i - M p_i| > M\epsilon) < \exp(-\eta \epsilon^2 M)$$

for some constant η. □

Corollary 1: Given a random regular code with p_i as defined by (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.

B. 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 [22] and [23]. Alternatively, one should be able to construct a similar argument using the approach of [26] and [8]. We note that the recent work of Burshtein and Miller [2] shows that the hard-decision decoding algorithm is guaranteed to correct all message nodes once it has corrected a sufficiently large fraction of the message nodes, provided that the underlying graph is a sufficiently good expander. Thus, the change of decoding algorithm suggested in this section is technically unnecessary; we include it for completeness.

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 [22], 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 [22] 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 [22] 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 [22, Theorems 10 and 11].

Lemma 2: 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 3: Let B be a bipartite graph, with nodes divided into left and right sides. 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 an edge into a right node. Then, with probability $1 - O(1/n)$, for some fixed $\alpha > 0$, $\epsilon > 0$, and $\beta = 3/4 + \epsilon$, B is an (α, β) expander.

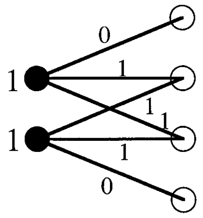

Fig. 2. If the two left nodes are supposed to be 0, and all other nodes are correct, then the majority tells the left nodes not to change.

We note that the restriction in Lemma 3 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 Fig. 2). Even cycles of any constant length occur with constant probability, so errors remain with constant probability.

To circumvent this problem Gallager designs specific 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 [10], [27]. 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 2 and 3. 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.

C. 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 2, and Lemma 3, 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.

III. Irregular Codes

A. Analyzing Irregular Codes

We now describe a decoding algorithm for codes based on irregular graphs, which we call *irregular codes*. We first describe the construction of such codes. 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 $\pi(i)$ out of the right side. Note that this may potentially lead to nodes with several edges between them, or *multiedges*; often in practice multiedges and small cycles can be removed to improve performance [16].

Following the notation used in [10] and [27], 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 II-A, 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 $1 - p_i$, it is simple to check that the probability that c' receives an even number of errors is

$$\frac{1 + \rho(1 - 2p_i)}{2}. \tag{7}$$

Expression (7) is the generalization of (1), taking into account the probability distribution on the degree of c'.

Also similarly to Section II-A, 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 II-A that the recursive description for p_i is

$$\begin{aligned} p_{i+1} = {} & p_0 - \sum_{j=1}^{d_\ell} \lambda_j \\ & \cdot \left[p_0 \sum_{t=b_{i,j}}^{j} \binom{j-1}{l} \left[\frac{1 + \rho(1 - 2p_i)}{2}\right]^t \right. \\ & \cdot \left[\frac{1 - \rho(1 - 2p_i)}{2}\right]^{j-1-t} \\ & + (1 - p_0) \sum_{t=b_{i,j}}^{j-1} \binom{j-1}{t} \left[\frac{1 - \rho(1 - 2p_i)}{2}\right]^t \\ & \left. \cdot \left[\frac{1 + \rho(1 - 2p_i)}{2}\right]^{j-1-t} \right]. \end{aligned} \tag{8}$$

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

$$\frac{1 - p_0}{p_0} \le \left[\frac{1 + \rho(1 - 2p_i)}{1 - \rho(1 - 2p_i)}\right]^{2b_{i,j} - j + 1}. \tag{9}$$

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 (9) 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.

B. 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

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

where now

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

and the λ_j are variables to be determined. Observe that condition (8) 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 of the form $f(x) < x$ for values of x that are 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

$$\sum_{\ell \in L} \lambda_\ell/\ell = (1-R) \sum_i \rho_i/i \tag{10}$$

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 also 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 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 (10)) when all check nodes have equal degree, or as nearly equal as possible.

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 nonzero entry. As we shall see in Section IV, 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 (8) 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 Theorem 1 holds for irregular graphs as well as for regular graphs, by an entirely 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 II-B. The overall decoding time is linear. (We again note that Burshtein and Miller [2] have recently shown that the hard-decision decoding algorithm could be used to finish the decoding, although this would result in an $O(n \log n)$ time algorithm.)

Lemma 4: 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 [22, Theorem 10]. 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 $|\delta(V)| = \overline{d}v$. 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

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

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

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

It follows that

$$u > \overline{d}v/2$$

and hence there is some message node with more than $1/2$ of its incident check nodes unsatisfied. Hence at each step the se-

TABLE I
PARAMETERS OF OUR CODES

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

quential 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 $u > \overline{d}\alpha n/2$. However, initially u is at most $vd < \alpha n/2$, and u decreases throughout the course of the algorithm, so this cannot happen. □

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.

C. Theoretically Achievable Error Correction

We have designed some irregular degree sequences using the linear programming methodology described in Section III-B. 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 than codes using our techniques. It is worth noting that the 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 I, 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.[2] 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 3, and thus there is no need for the additional structure discussed in Section II-B.

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 3 to hold, we can use the additional structure discussed in Section II-B 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.

IV. EXPERIMENTAL RESULTS FOR GALLAGER'S ALGORITHM

We include preliminary experimental results for new codes we have found using the linear programming approach. Our experimental design is similar to that of [22], whose results can be compared with ours. We describe a few important details of our experiments and implementations. We model 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.

Rather than encoding a message for each trial, we use an initial message consisting entirely of zeros. Since the code is linear and the decoding algorithm respects its linearity, no generality is lost.

A different random graph was constructed 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 [16] and [22], when necessary we remove multiedges from our graphs.

In our implementation, we simply run Gallager's improved decoding algorithm (with thresholds b_i) until it finishes, or until a prespecified number of rounds pass without success. Our implementation therefore 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 (9). In practice, however, the schedule determined by (9) must be slightly modified. 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 (9) 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. 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 it turns out that it is unnecessary to switch to the modified decoding algorithm of Section II-B or use the additional structure described in Section II-B, as in our experience the hard-decision decoding algorithm of Gallager finishes successfully once the number of errors becomes small.

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 as far as we know such an event is possible.

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

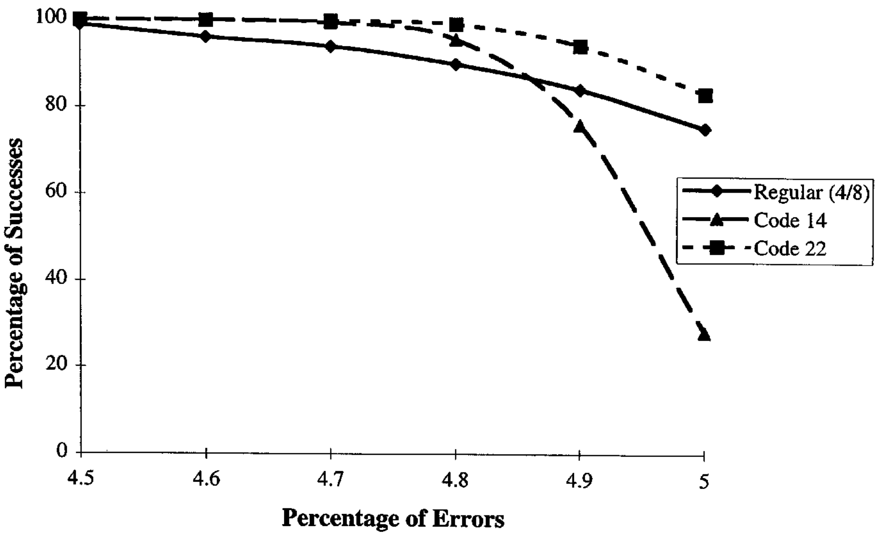

Fig. 3. Percentage of successes based on 2000 trials.

A. Experiments with Hard-Decision Decoding

We first describe experiments on codes of rate $1/2$ with 16 000 message bits and 8000 check bits. In Fig. 3, we describe the performance of Code 14 and Code 22 that we introduced in Section III-C. Each data point represents the results from 2000 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 2000 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 similar regular codes, while still having linear running time. 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 Fig. 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 2000 trials with a fraction of 0.045 errors.

We now consider Code $10'$ and Code $14'$ introduced in Section III-C (see Fig. 4). The experiments were run on 16 000 message bits and 8000 check bits for 2000 trials. In our experiments, we remove both multiedges and some small cycles, as suggested in [16]. 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.

V. Low-Density Parity-Check Codes and Belief Propagation

In this section, we review belief propagation for the low-density parity-check codes developed by Gallager using the framework of MacKay and Neal [7], [16].

Similar to the hard-decision decoding algorithm of Section II-A, only extrinsic information is passed from the message bits to check bits and back. As explained in [16], the algorithm runs two alternating phases, in which for each nonzero 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 the codeword $\boldsymbol{x}$ 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 the codeword $\boldsymbol{x}$ 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 crossover 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 in 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

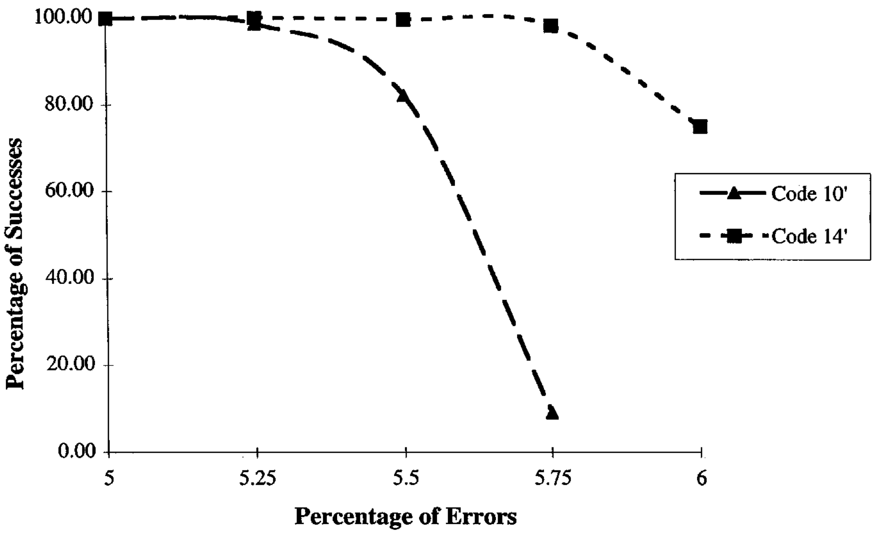

Fig. 4. Percentage of successes based on 2000 trials.

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 [6], [7], [16], and [24].

A. Simulation of Irregular Graph Performance Under Belief Propagation

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. We use a message consisting entirely of zeros, introduce a fixed number of errors corresponding to a fraction p of the block length, create a new random graph for each trial, and remove multiedges as necessary. 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 $(2\sigma^2)$.

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. Again, 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.

We describe the irregular graphs used in Table II, which are derived from irregular graphs for erasure codes [27]. 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 III. (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.)

TABLE II
PARAMETERS OF OUR CODES

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	$\rho_4 = 0.160416, \rho_{10} = 0.404478,$
	$\rho_{33} = 0.303338, \rho_{34} = 0.131768$

B. Binary-Symmetric Channel

Table III 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 [16]. (The differences may be due to our fixing the number of errors, while the results of [16] use a genuine binary-symmetric channel. Of course, there may also be other minor differences in the graph construction.) 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 crossover probability f. The results are reported 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 fail notably less often 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 [16], we believe that

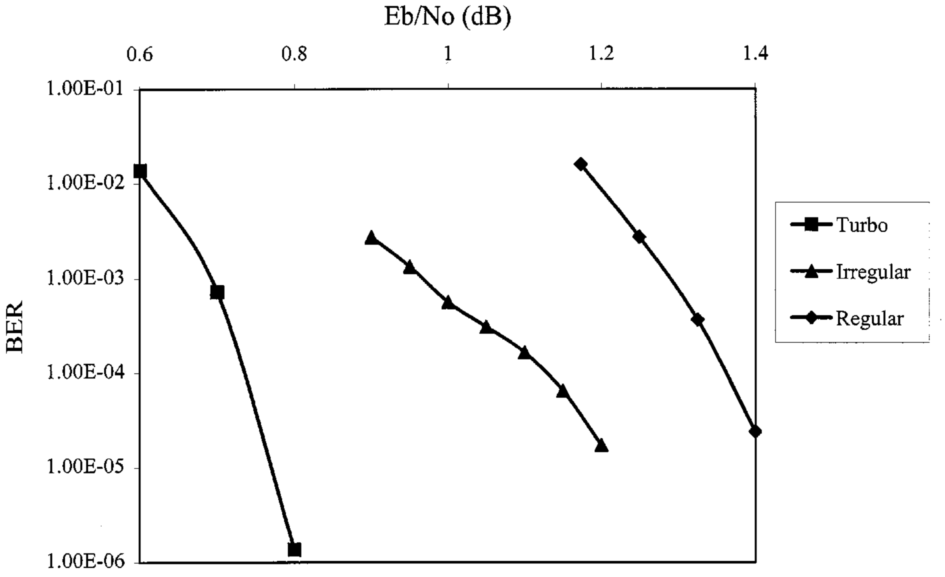

Fig. 5. Irregular codes versus regular codes and turbo codes: rate $1/2$.

TABLE III
COMPARING REGULAR AND IRREGULAR GRAPHS

	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

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.

When decoding 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 four times as many operations to decode at higher rates. In software implementations, performance can actually be worse than this, however, since the larger graph size for the irregular codes may require 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.

C. Gaussian Channel

Figs. 5 and 6 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 [5] and regular codes [17] 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 [5] are available at http://www331.jpl.nasa.gov/public/TurboPerf.html.) For our

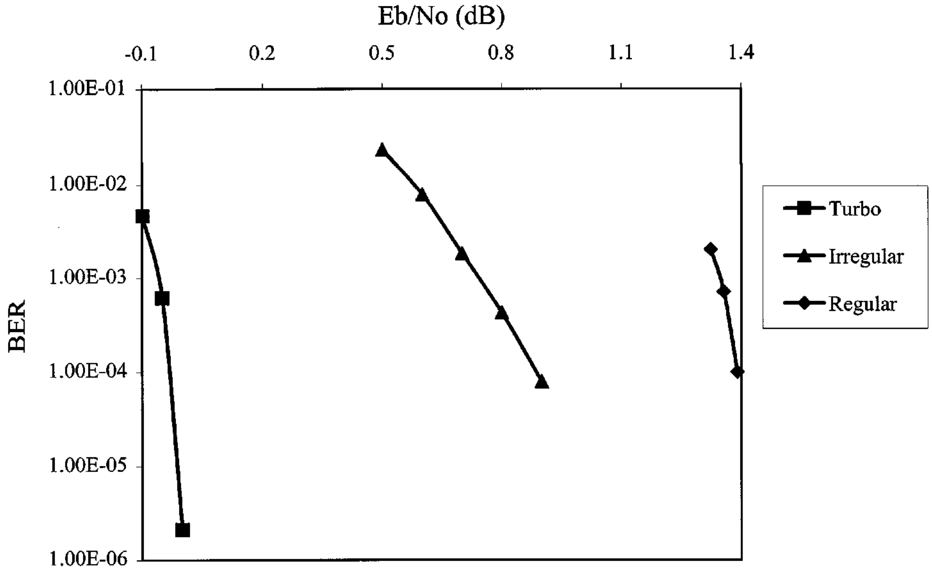

Fig. 6. Irregular codes versus regular codes and turbo codes: rate $1/4$.

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 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 1000 trials at both 0.95 and 1.00 dB. Our estimates for the BER from 1000 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 [17].

Our results for irregular codes at rate $1/4$ (Fig. 6) similarly show significant improvement over regular codes. At this lower rate and block length, however, turbo codes appear to have a significant edge. The edge has subsequently been reduced with further exploration and experimentation [20], [21].

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 1000 trials at both 0.70 and 0.60 dB. Our estimates for the BER from 1000 trials at 0.50 dB is $6.18 \cdot 10^{-4}$ and at 0.40 dB is $8.39 \cdot 10^{-3}$.

VI. Conclusion and Further Developement

This paper offers two important contributions. First, we demonstrate that low-density parity-check codes based on irregular graphs can substantially outperform similar codes based on regular graphs. We show this both through asymptotic analysis for a hard-decision decoding algorithm and experimentally for belief propagation. Second, we introduce new methods for analyzing low-density parity-check codes using concentration bounds based on martingales.

These contributions have been built upon in subsequent work. Our main "concentration theorem" based on martingales has since been extended to a large class of channel models by Richardson and Urbanke [21]. Based on this approach, they also developed the "density evolution" algorithm, a numerical procedure to approximate the threshold of noise below which the belief propagation algorithm is asymptotically successful. Indeed, sequences of codes have since been constructed for which the belief propagation algorithm had a performance extremely close to the Shannon capacity, beating the best performing turbo codes known at the time [20].

There remains, however, much more to be done. We suggest one problem in particular. The concentration bounds we use apply to the asymptotic behavior of low-density parity-check codes, but they do not adequately explain the behavior of small codes, say with only thousands of bits. For such small codes, the corresponding bipartite graphs necessarily have small cycles, which is a complication our asymptotic analysis cannot adequately handle. More understanding of small codes could be extremely useful for designing low-density parity-check codes and making them the code of choice in practice.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE Int. Communications Conf.*, 1993.

[2] D. Burshtein and G. Miller, "Expander graph arguments for message passing algorithms," *IEEE Trans. Inform. Theory*, vol. 47, pp. 782–790, Feb. 2001.

[3] J.-F. Cheng and R. J. McEliece, "Some high-rate near capacity codecs for the Gaussian channel," in *34th Allerton Conf. Communications, Control and Computing*, 1996.

[4] M. C. Davey and D. J. C. MacKay, "Low-density parity-check codes over GF (q)," *IEEE Commun. Lett.*, vol. 2, pp. 165–167, June 1998.

[5] D. Divsalar and F. Pollara, "On the design of turbo codes," JPL TDA, Progr. Rep. 42-123.

[6] G. D. Forney Jr., "The forward–backward algorithm," in *Proc. 34th Allerton Conf. Communications, Control and Computing*, 1996, pp. 432–446.

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

[8] S. I. Kovalev, "Decoding of low-density codes," *Probl. Inform. Transm.*, vol. 27, no. 4, pp. 317–321, 1992.

[9] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 219–230, Feb. 1998.

[10] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, “Practical loss-resilient codes,” in *Proc. 29th Annu. Symp. Theory of Computing*, 1997, pp. 150–159.

[11] M. Luby, M. Mitzenmacher, and M. A. Shokrollahi, “Analysis of random processes via and-or trees,” in *Proc. 9th Annu. ACM-SIAM Symp. Discrete Algorithms*, 1998, pp. 364–373.

[12] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman, “Improved low-density parity-check codes using irregular graphs and belief propagation,” in *Proc. 1998 Int. Symp. Information Theory*, p. 117.

[13] ——, “Analysis of low-density codes and improved designs using irregular graphs,” in *Proc. 30th Annu. Symp. Theory of Computing*, 1998, pp. 249–258.

[14] D. J. C. MacKay. (1998) Turbo codes are low-density parity-check codes. [Online]. Available: http://www.cs.toronto.edu/~mackay/abstracts/-turbo-ldpc.html

[15] D. J. C. MacKay, R. J. McEliece, and J.-F. Cheng, “Turbo coding as an instance of Pearl’s ‘belief propagation’ algorithm,” *IEEE J. Select. Areas Commun.*, vol. 17, pp. 1632–1650, Sept. 1999.

[16] D. J. C. MacKay, “Good error correcting codes based on very sparse matrices,” *IEEE Trans. Inform. Theory*, vol. 45, pp. 399–431, Mar. 1999.

[17] D. J. C. MacKay and R. M. Neal, “Near Shannon limit performance of low-density parity-check codes,” *Electron. Lett.*, vol. 32, pp. 1645–1646, 1996.

[18] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 1995.

[19] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA: Morgan Kaufmann, 1988.

[20] T. Richardson, A. Shokrollahi, and R. Urbanke, “Design of capacity-approaching low-density parity-check codes,” *IEEE Trans. Inform. Theory*, vol. 47, pp. 619–637, Feb. 2001.

[21] T. Richardson and R. Urbanke, “The capacity of low-density parity-check codes under message-passing decoding,” *IEEE Trans. Inform. Theory*, vol. 47, pp. 599–618, Feb. 2001.

[22] M. Sipser and D. A. Spielman, “Expander codes,” *IEEE Trans. Inform. Theory*, vol. 42, pp. 1710–1722, Nov. 1996.

[23] D. A. Spielman, “Linear time encodable and decodable error-correcting codes,” *IEEE Trans. Inform. Theory*, vol. 42, pp. 1723–1731, Nov. 1996.

[24] N. Wiberg, “Codes and decoding on general graphs,” Ph.D. dissertation, Dept. Elec. Eng., Univ. Linköping, Sweden, Apr. 1996.

[25] N. Wiberg, H.-A. Loeliger, and R. Kötter, “Codes and iterative decoding on general graphs,” *Euro. Trans. Telecommun.*, vol. 6, pp. 513–526, Sept. 1995.

[26] V. V. Zyablov and M. S. Pinsker, “Estimation of the error-correction complexity of Gallager low-density codes” (in Russia), *Probl. Pered. Inform.*, vol. 11, pp. 23–26, Jan. 1975. English translation in *Probl. Inform. Transm.*, vol. 11, no. 1, pp. 18–28. 1976.

[27] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, “Efficient erasure correcting codes,” *IEEE Trans. Inform. Theory*, vol. 47, pp. 569–584, Feb.. 2001.