On the Practicality of Low-Density Parity-Check Codes

Alex C. Snoeren
MIT Lab for Computer Science
Cambridge, MA 02138

snoeren@lcs.mit.edu

June 27, 2001

Abstract

Recent advances in coding theory have produced two classes of codes, *turbo* codes and low-density parity-check (LDPC) codes, which approach the Shannon limit of channel capacity while admitting efficient implementations of message encoding and decoding in software. Theoretic results about the latter have been shown to apply to the former, hence we examine the evolution of LDPC codes from their origin in Gallager's 1963 thesis to their current incarnation as *tornado* codes developed by Luby, Mitzenmacher, Shokrollahi, and Spielman. After considering several analytic approaches to quantifying their performance, we discuss the practicality of LDPC codes, particularly those designed for the erasure channel, when applied to a number of current problems in networking.

1 Introduction

The late Claude Shannon founded an entire field of study in 1948 with his discovery of the Noisy Channel Coding Theorem [15]. Shannon proved that every communication channel has a fixed capacity, which can be expressed in terms of bits per second, and that it is not possible to send information across a channel at a rate exceeding its capacity, or *Shannon limit*, as it has come to be known. Since that time, an entire field of study has grown out of attempts to design coding schemes that reach or approach the Shannon limit of various channels, including those that simply drop message bits, termed erasure channels [5], and those that add various types of noise to the signal.

One class of codes considers encoding k-dimensional message blocks of length n over a finite field F_q, resulting in a k-dimensional linear subspace of F_q^n. Clearly, in order to successfully transmit a message over a lossy channel, k is selected to be strictly less than n, producing an over-complete basis for the k-dimensional message space. The process of encoding maps a k-dimensional message into an n-dimensional codeword, and can be described succinctly by a $k \times n$ generator matrix. The ratio of information to data sent, k/n, serves as a efficiency metric, and is known as the rate of the code.

By carefully selecting the basis vectors, a message can be reconstructed with any appropriately-sized subset of the n vectors. If each vector is linearly independent, any k vectors will do. Since the channel may also introduce noise, codes attempt to maximize the distance between constituent codewords to reduce the likelihood of confusion. Note codewords are easily identified through the use of a parity-check matrix, which maps all valid codewords to zero (by verifying the linear constraints between basis vectors). Codes with maximum Hamming distance between constituent codewords are termed maximum-distance separable (MDS), and can be shown to achieve full capacity.

A common family of MDS codes is given by Reed-Solomon (RS) codes [17], which underlie the coding schemes of a large number of technologies, including audio Compact Discs. By utilizing cyclic groups as their finite field, algorithms exist for RS codes that enable encoding and decoding in time $O(n \log^2 n \log\log n)$ asymptotically, although quadratic, matrix-based algorithms are often faster for small values of n [7]. Due to the inherent performance limitations of RS codes, researchers have continued to search for other classes of block codes that approach the Shannon bound.

One particularly attractive class of block codes that has received a good deal of recent attention is low-density parity-check (LDPC) codes, originally proposed by Gallager in his 1963 thesis [6]. Unlike Reed-Solomon codes, which rely on dense parity-check matrices, Gallager's matrices are sparse, enabling more efficient decoding. As originally proposed, the column vectors of the parity-check matrices all had equal weight, resulting in so-called regular codes. Luby, Mitzenmacher, Shokrollahi, and Spielman further improved the efficiency of LDPC through the use of irregular codes [8]. Recent work shown encoding can also be done in near-linear time [7, 10], and that these codes are amenable to rigorous theoretical analysis [6, 7, 8, 14], admitting tight efficiency bounds provably close to the Shannon limit for a large class of communication channels [14].

The study of low-density parity-check codes has even greater value due to their relationship to a larger class of codes [9], including the *turbo* codes introduced by Berrou, Glavieux, and Thitimajshima [1]. As their name implies, turbo codes represent the other currently-known class of codes to provide efficient (essentially linear) encoding and decoding with capacities approaching the Shannon limit. Similarly, turbo code decoding has been framed as a belief propagation algorithm [11], a class of decoding algorithms first developed for LDPC codes [10]. Hence, theoretical results derived from the study of LDPC codes are likely to impact both classes of efficient, high-capacity coding schemes, although the converse is not necessarily true.[1]

Due to their remarkable performance, low-density parity-check codes have been proposed for use repeatedly in the context of computer systems [3, 4], where efficient encoding and decoding are of paramount importance. Furthermore, in the specific case of packet-based communication (such as the packet-switched Internet), codes designed for the erasure channel are particularly well suited. We consider the utility of LDPC for a number of networking applications in this paper.

The remainder of the paper is organized as follows. We begin in section 2 with a brief tutorial on low-density parity-check codes as proposed by Gallager [6], outlining his proposed decoding algorithms and their performance over the binary-symmetric channel. We present improved decoding algorithms in section 3 that use an extended message alphabet. Section 4 discusses results that show LDPC codes based on random graphs perform almost as well as Gallager's explicit constructions. Section 5 discusses the further extension to random graphs of irregular degree, as proposed by Luby, Mitzenmacher, Shokrollahi, and Spielman [8].

We then consider more practical aspects, beginning in section 6 with the development of extremely fast random irregular LDPC codes for the erasure channel, namely the tornado codes of Luby, Mitzenmacher, Shokrollahi, and Spielman [7]. Finally, we consider the applicability of LDPC codes on the erasure channel to a number of problems in computer systems in section 7 before concluding in section 8.

2 Gallager codes

In this section, we will introduce the class of regular low-density parity-check codes (also known as Gallager codes), and show their relation to bipartite graphs. We subsequently describe the class of output-symmetric channels these codes were designed to operate over. Following the notation of Richardson and Urbanke [14], we will then present Gallager's hard-decision message-passing decoding algorithms, and analyze their performance in a simplistic channel model.

[1] Turbo codes are typically constructed by concatenating specific (convolutional) constituent codes, as opposed to random members of an ensemble, as analyzed in the LDPC case.

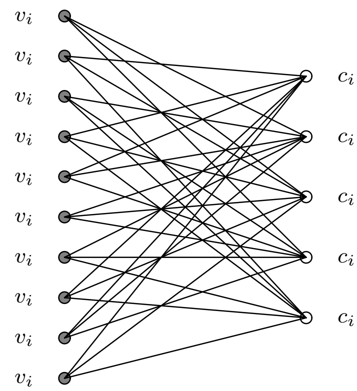

Figure 1: A bipartite graph representation of a $(3, 6)$-regular (Gallager) code with design rate $1/2$. In conventional linear code notation, this is a $[10, 5]_2$-code, since it has length 10 and dimension 5. (Adapted from [14, fig. 1].)

2.1 Construction

Any linear code can be expressed as the set of solutions to a parity-check equation, $C := \{c : Hc^T = 0\}$ [17]. The matrix H is known as the parity-check matrix for the code C, as it represents a series of parity check equations. If we consider codes over GF(2)–binary bits, or *binits*, as Gallager called them, then each parity check equation is just a series of binary XOR operations. Gallager defined a (d_v, d_c)-regular LDPC code as a linear code over GF(2) where each message node is involved in d_v parity check equations, and each parity check equation consists of d_c message nodes. Each column of the matrix H has d_v ones, and each row has d_c ones, hence the name low-density parity-check matrices.

It is often convenient to view such codes as a bipartite graph. The term regular code follows directly from the graphical interpretation of the code: each message node has degree d_v, and check nodes d_c. Figure 1 shows the graphical representation of a $(3, 6)$ Gallager code with length 10. The codewords correspond to those sequences of 10 bits the XOR of the d_c message nodes adjacent to each check node is zero.

Each linearly independent parity check equation reduces the dimension of the codeword by one. Hence the *design* rate of a Gallager code of length n with $m = nd_v/d_c$ check bits is given by (from [14])

$$R = \frac{n-m}{n} = 1 - \frac{d_v}{d_c}.$$

For any particular code, the constraint set might not be completely independent, hence the actual rate of a particular code may be higher.

2.2 Channel assumptions

A noisy channel may cause codewords to be received incorrectly, hence the process of decoding attempts to map a received message to the codeword sent with highest probability. Clearly codewords can be identified through matrix multiplication (recall codewords must satisfy the equation $Hc^T = 0$), but this provides little guidance for messages that are not valid codewords. For a given channel model, the corresponding highest-likelihood codeword for each message could be pre-computed and stored in an exponentially-large lookup table, however, as the block length grows large this approach rapidly becomes infeasible. We will see later that long block lengths provide better performance, hence we seek a general algorithm to map a received message to its highest-likelihood codeword.

Gallager initially considered decoding in the presence of a binary symmetric channel [6], the simplest channel error model. A binary symmetric channel (BSC) with parameter p has a binary alphabet with equal cross-over probability p. That is to say, for an input symbol x_t at time t from the alphabet $x_t \in \mathcal{I} := \{-1, +1\}$, the channel output symbol at time t is given by $y_t \in \mathcal{O} := \{-1, +1\}$, where

$$\Pr[y_t = -1 | x_t = 1] = \Pr[y_t = 1 | x_t = -1] = p.$$

Richardson and Urbanke considered decoding over a larger variety of memory-less (where each message bit is transmitted independently) channel models, which we consider in section 3.3, but required the channels retain the output-symmetric property, namely for any symbols in the input alphabet $i \in \mathcal{I}$ and output alphabet $o \in \mathcal{O}$,

$$\Pr[y_t = -o | x_t = i] = \Pr[y_t = o | x_t = -i].$$

2.3 Message-passing decoding

We now describe two decoding algorithms proposed by Gallager [6] and then consider their performance. A message passing algorithm proceeds by exchanging messages between adjacent nodes in the graph. The decoding process is considered to operate in rounds, where in each round a value from some message alphabet $\mathcal{M}$ is sent from each message node to its adjacent check nodes, and the check nodes respond with a value per adjacent message node.

In the first round, each message node simply sends the value initially received on the channel, requiring $\mathcal{O} \subset \mathcal{M}$. The check nodes process these messages, and respond in kind with a message based upon the messages received from adjacent message nodes. At the start of the following round, the message nodes process the messages received from adjacent check nodes and, in combination with the value received on the channel, compute a new message value to send to the check nodes. This process continues indefinitely, hopefully converging on the maximum-likelihood codeword for the received message.

The decoding algorithms presented here, and, indeed, all the algorithms considered in this paper, generate messages based on only *extrinsic* information. That is, the messages sent to a particular node are in no way dependent on any messages received from that node. This property turns out to be essential in proving performance bounds for the decoding algorithms, and will be considered further in section 4.2.

For continuity, we will express all decoding algorithms in the style of Richardson and Urbanke [14], regardless of their origin. For each algorithm, we will define two message maps, one for the message nodes $\Psi_v : \mathcal{O} \times \mathcal{M}^{d_v - 1} \to \mathcal{M}$, and one for the check nodes $\Psi_c : \mathcal{M}^{d_c - 1} \to \mathcal{M}$. We start by examining Gallager's original algorithms over a discrete message alphabet, so-called *hard-decision* algorithms.

2.3.1 Gallager's algorithm A

Gallager first proposed an algorithm in which, for each message node, adjacent check nodes simply sent the XOR of the messages received from all other incident message nodes: $m_i = m_1 \oplus \ldots \oplus m_{d_c - 1}$. If one considers the message alphabet to be $\mathcal{M} := \{-1, +1\}$, this can be written

$$\Psi_c(m_1, \ldots, m_{d_c - 1}) = \prod_{i=1}^{d_c - 1} m_i.$$

The message nodes continue to send the received bit to a check node unless the messages received from all other incident check nodes are identical, and disagree with the received bit: $\Psi_v(m_0, m_1, \ldots, m_{d_v - 1}) = -m_0$ if and only if $m_1 = \ldots = m_{d_v - 1} = -m_0$, m_0 otherwise.

2.3.2 Gallager's algorithm B

Gallager observed that the above algorithm could be improved by allowing message nodes to switch their value sooner. In his revised algorithm, for each round, j, there is a universal threshold value, b_j, at which point message nodes should switch their value. The message node message map now varies with respect to the round. We denote the map at round j as Ψ_v^j; hence, $\Psi_v^j(m_0, m_1, \ldots, m_{d_v - 1}) = -m_0$ if and only if $|\{i : m_i = -m_0\}| \geq b_j$, m_0 otherwise. The map for check nodes remains unchanged.

2.4 Performance

The performance of a decoding algorithm can be described as a function of a channel parameter. We would like to find a threshold value below which the decoding algorithm is guaranteed to succeed, or at least do so with high probability. The BSC is well known to have a Shannon capacity of

$$C_{BSC}(p) = 1 + p \log p + (1 - p) \log(1 - p),$$

hence we can directly compare the performance of the above decoders against the theoretical capacity limit for the BSC.

As stated above, we wish to find a threshold value, p^*, below which, if we run a message-passing decoder long enough, it will converge to the correct message.

We focus our attention on determining the probability that a particular message node remains in error after some number of rounds, j. We begin by outlining Gallager's intuitive analysis, which defines a recursive expression for the probability of sending an incorrect message.

2.4.1 Recursive enumeration

Let p_j be the probability that a message node, m, has an incorrect value in round j. Clearly $p_0 = p$. We are interested in the cases where $\lim_{j\to\infty} p_j = 0$. For Gallager's algorithm A, there are precisely two ways the message node could be in error in round $j+1$. Either it initially received the wrong message, or it initially received the correct message, but was convinced to change by its check nodes. It is also possible, however, that the message was originally received in error, but corrected in round $j+1$. Recall a message node only changes its message if all of its other check nodes are in agreement. A check node sends the correct value to a message node precisely when an even number (including zero) of its incident message nodes (other than m) are correct. With appropriate independence assumptions about message bits (which we shall return to examine in section 4.2), this is given by

$$\frac{1+(1-2p_j)^{d_c-1}}{2}. \tag{1}$$

Hence the probability the message was received in error but corrected in round $j+1$ is precisely

$$p_0\left[\frac{1+(1-2p_j)^{d_c-1}}{2}\right]^{d_v-1}.$$

By symmetry, the probability the message was received correctly by coerced into an incorrect value is given by

$$(1-p_0)\left[\frac{1-(1-2p_j)^{d_c-1}}{2}\right]^{d_v-1}.$$

By combining these three cases, we have

$$\begin{aligned} p_{j+1} = p_0 - p_0\left[\frac{1+(1-2p_j)^{d_c-1}}{2}\right]^{d_v-1} \\ + (1-p_0)\left[\frac{1-(1-2p_j)^{d_c-1}}{2}\right]^{d_v-1}. \end{aligned} \tag{2}$$

As noted by Richardson and Urbanke [14], for a fixed p_j, p_{j+1} is an increasing function of p_0. Similarly, in the base case, for a fixed p_0, p_{j+1} is an increasing function of p_j. Therefore, by induction, p_j is an increasing function of p_0. Let p^* be the supremum of all values $p_0 \in [0,1]$ such that $\lim_{j\to\infty} p_j = 0$. Gallager showed for a certain set of explicitly-constructed graphs, which satisfied the independence criteria mentioned above, that $\lim_{j\to\infty} p_j = 0$ for all $p < p^*$.

This formula can clearly be generalized to Gallager's Algorithm B by summing over the number of check nodes in excess of b_j (equation 2 is simply equation 3 with $b_j = d_v - 1$):

$$\begin{aligned} p_{j+1} = p_0 - p_0 \sum_{t=b_j}^{d_v-1} \binom{d_v-1}{t} \left[\frac{1+(1-2p_j)^{d_c-1}}{2}\right]^{t} \\ \cdot \left[\frac{1-(1-2p_j)^{d_c-1}}{2}\right]^{d_v-1-t} \\ + (1-p_0) \sum_{t=b_j}^{d_v-1} \binom{d_v-1}{t} \left[\frac{1-(1-2p_j)^{d_c-1}}{2}\right]^{t} \\ \cdot \left[\frac{1+(1-2p_j)^{d_c-1}}{2}\right]^{d_v-1-t}. \end{aligned} \tag{3}$$

We are looking to minimize p_{j+1}. Gallager showed the equation above is minimized when the probability of correcting the message using the check nodes and the threshold b_j just exceeds the probability of receiving the correct message initially [6]. This corresponds to the smallest integer b_j that satisfies:

$$\frac{1-p_0}{p_0} \le \left[\frac{1+(1-2p_j)^{d_c-1}}{1-(1-2p_j)^{d_c-1}}\right]^{2b_j-d_v+1}. \tag{4}$$

Once again, the supremum p^*, of all values of p_0 for which the algorithm converges ($\lim_{j\to\infty} p_j = 0$), signifies the capacity threshold. Given appropriately-constructed codes that satisfy the necessary independence conditions, the same argument holds as before that for all values $p_0 < p^*$, $\lim_{j\to\infty} p_j = 0$.

3 Expanded alphabets

Subsequent to Gallager's initial work, researchers discovered that decoding performance can be improved by extending the message alphabet used by the decoder. Proposals have included both larger discrete alphabets and continuous ones. Unfortunately, the additional decoder complexity makes the algorithms more difficult to analyze using the asymptotic analysis above, as the number of coupled equations grows linearly in the size of the message alphabet [14]. Instead, Richardson and Urbanke developed a numerical procedure to approximate the threshold value by modeling the evolution of message probability densities [14].

3.1 Decoding

We first introduce two additional decoding algorithms that utilize expanded message alphabets, one with a ternary alphabet, and one continuous, and then consider computing their threshold values through density evolution. We will see shortly that both algorithms provide substantial improvements upon their predecessors.

3.1.1 *Mitzenmacher's algorithm E*

Mitzenmacher extended Gallager's second algorithm in a straightforward fashion to allow nodes to be indecisive [12]. The first step is simply to expand the decoder's message alphabet to include erasures, $\mathcal{M} := \{-1, 0, 1\}$. The message node map is then redefined to calculate a rough majority of the check nodes, giving the original received message certain weight, w_j (somewhat analogous to Gallager's b_j), which varies as rounds progress. This can be expressed mathematically through the sgn function:

$$\Psi_v(m_0, m_1, \ldots, m_{d_v-1}) = \operatorname{sgn}(w_j m_0 + \sum_{i=1}^{d_v-1} m_i).$$

Once again, the map for the check nodes remains the same (note the typo in [14]):

$$\Psi_c(m_1, \ldots, m_{d_c-1}) = \prod_{i=1}^{d_c-1} m_i.$$

3.1.2 *Belief propagation*

The increased performance of a ternary message alphabet clearly suggests considering even larger alphabets. Indeed, Richardson and Urbanke construct a decoder for Gaussian channels using an alphabet of eight symbols [14, ex. 7], which we will not consider here. The limit, obviously, is a completely continuous message alphabet. The class of message-passing algorithms based on continuous alphabets is known as sum-product analysis, or belief propagation [10], and provides more robust decoding at the expense of increased decoder complexity.

A message (m, c) sent by a message node in hard-decision algorithms represents m's "best guess" of its correct value, based on received information from all adjacent check nodes other than c. Using a continuous alphabet, belief propagation is instead able to communicate an approximate probability, expressed as posterior densities, that the variable associated with m takes on the value in question.

Similarly, messages $(c, m)_z$ sent from a check node to a message node (there is actually only one message per round as before, but it is convenient for the time being to consider sending multiple messages for each possible value of m) represent the probability, conditioned on information received from all other adjacent message nodes, that the check node will be satisfied if node m takes value z. Following the framework of [14] for a BSC, it is convenient to express the two relevant probabilities, p_1 and p_{-1} as a log-likelihood ratio, $\log \frac{p_1}{p_{-1}}$. Since $p_1 + p_{-1} = 1$ on a BSC, one message is sufficient to precisely communicate two conditional probabilities.

We make a similar independence assumption as before, i.e. the random variables on which messages are based are independent, in which case messages represent distributions conditioned on the respective value of the variable, but conditionally independent of everything else. We now formalize the behavior of both message and check nodes in belief propagation using the message map abstraction of Richardson and Urbanke [14].

As with all message passing algorithms, message nodes initially send the received value of their associated variable. In successive rounds, each message node computes an updated conditional distribution based upon the messages received from check nodes, yet continuing to respect the extrinsic principle introduced above. Technically, the message sent is the *a posteriori* probability that the value of the associated variable based on the values of all nodes observed up to and including the last round. Since each received distribution is independent, the distribution conditioned on all of the variables is simply their product, which, in log-likelihood form, can be expressed as

$$\Psi_v(m_0, m_1, \ldots, m_{d_v-1}) = \sum_{i=0}^{d_v-1} m_i.$$

The computations performed by check nodes are slightly less intuitive. As stated above, a message (c, m) is a log-likelihood ratio $\log \frac{\tilde{p}_1}{\tilde{p}_{-1}}$, where $\tilde{p}_{\pm 1}$ is the probability the check node is satisfied of m takes on the value ± 1. Recall the incoming messages from each of the $d_c - 1$ other adjacent message nodes $m' \neq m$ are of the form $\log p_1^{m'}/p_{-1}^{m'}$. Hence $\tilde{p}_{\pm 1}$ is simply the probability that

$$\prod_{m'=1}^{d_c-1} p_{\pm 1}^{m'} = \pm 1.$$

By using a clever change of variables and appealing to the Fourier transform over GF(2), Richardson and Urbanke [14] show that this calculation can be represented by the following message map

$$\Psi_c(m_1, \ldots, m_{d_c-1}) = \log\left(\frac{1 + \prod_{i=1}^{d_c-1} \tanh \frac{1}{2} m_i}{1 - \prod_{i=1}^{d_c-1} \tanh \frac{1}{2} m_i}\right).$$

3.2 Performance

As before, we can construct a coupled set of recursive equations for Mitzenmacher's algorithm to compute the evolution of p_{-1}, p_0, and p_1 through successive rounds. In the interest of brevity, we do not show them here, but suffice it to say they are somewhat unwieldy [14]. Note that as with Gallager's b_j's, we need to determine good values of w_j for each iteration, and there is no clear ordering amongst the three values, hence we cannot derive an equation for optimum values of w_j analogous to equation 4 for b_j.

One alternative approach, suggested by Richardson and Urbanke, is to set a desired weight for erasures when compared to incorrect values, say $1/2$, and then use dynamic programming to compute the optimum weight for each round. They find $w_1 = 2; w_j = 1, j \geq 2$ is optimum for a $(3, 6)$-regular code [14]. While broadly applicable, this is computationally intensive, and becomes infeasible for larger alphabets. Hence, they suggest using a sensible heuristic based on the channel characteristics, but make no further claims about optimality. Thresholds computed using this approach are shown in table 1, along with thresholds for the other decoding algorithms.

3.2.1 *Density evolution*

Unfortunately the asymptotic analysis fails us when considering belief propagation algorithms. Richardson and Urbanke instead developed a numerical procedure to approximate the threshold, p^*, below which the algorithms are asymptotically successful [14]. Rather than explicitly tracking the values of each message during each round, it models the evolution of the probability density functions over all possible values of the messages. Hence, they termed their procedure "density evolution."

Returning to the notation introduced previously, let $\Pi_{\mathcal{M}}$ denote the space of probability distribution defined over the alphabet $\mathcal{M}$. We say a message $m \in \mathcal{M}$ is a random variable distributed according to some $P \in \Pi_{\mathcal{M}}$, and let $P^{(j)}$ denote the common density associated with messages from message nodes to check nodes in round j. Similarly, $Q^{(j)}$ represents the density of messages from check nodes to message nodes in the same round. Clearly $P^{(0)}$ is simply the density of the received values.

Density evolution iterates over $P^{(j)}$, which requires the ability to calculate $P^{(j+1)}$ from $P^{(j)}$. Richardson and Urbanke [14] describe a convenient change of measure from a density of log-likelihoods P to an equivalent density $\tilde{P}$ over GF(2) $\times [0, \infty)$,

$$\begin{aligned} \tilde{P}^0(y) &= \frac{1}{\sinh(y)} P\left(-\log\tanh\frac{y}{2}\right), \\ \tilde{P}^1(y) &= \frac{1}{\sinh(y)} P\left(\log\tanh\frac{y}{2}\right), \end{aligned}$$

which allows us to determine the density $\tilde{Q}^{(j)}$ as

$$\begin{aligned} \hat{\tilde{Q}}^{(j),0} - \hat{\tilde{Q}}^{(j),1} &= \left(\hat{\tilde{P}}^{(j-1),0} - \hat{\tilde{P}}^{(j-1),1}\right)^{d_c-1} \\ \hat{\tilde{Q}}^{(j),0} + \hat{\tilde{Q}}^{(j),1} &= \left(\hat{\tilde{P}}^{(j-1),0} + \hat{\tilde{P}}^{(j-1),1}\right)^{d_c-1}, \end{aligned} \tag{5}$$

d_v	d_c	Rate	$p^*(A)$	$p^*(B)$	$p^*(E)$	$p^*(BP)$	p^*_{opt}
3	6	0.5	0.04	0.04	0.07	0.084	0.11
4	8	0.5	0.047	0.051	0.059	0.076	0.11
5	10	0.5	0.027	0.041	0.055	0.068	0.11
3	5	0.4	0.061	0.061	0.096	0.113	0.146
4	6	0.333	0.066	0.074	0.09	0.116	0.174
3	4	0.25	0.106	0.106	0.143	0.167	0.215

Table 1: The capacity threshold for each of the decoding algorithms. The optimal threshold is given by the Shannon limit for a BSC multiplied by the rate of the code. (From [14, Tbl. 1])

where $\hat{\tilde{P}}$ denotes the Laplace transform of $\tilde{P}$. Hence, we can compute $P^{(j+1)}$ from $P^{(0)}$ and $P^{(j)}$ using the FFT:

$$\mathcal{F}\left(P^{(j+1)}\right) = \mathcal{F}\left(P^{(0)}\right)\left(\mathcal{F}\left(Q^{(j)}\right)\right)^{d_v-1}.$$

Threshold values for belief propagation over the BSC calculated by Richardson and Urbanke using the algorithm above are shown in in table 1. In this case the initial density function is given by

$$\begin{aligned} P^{(0)}(x) = p\delta & \left(x + \log\left(\frac{1-p}{p}\right)\right) \\ & + (1-p)\delta\left(x - \log\left(\frac{1-p}{p}\right)\right), \end{aligned}$$

where δ is the Dirac delta function.

3.3 General channel models

The beauty of Richardson and Urbanke's density evolution is that it does not depend on the underling channel model. Provided the channel meets the symmetry requirements mentioned previously, it suffices to express the initial variable settings as a probability density, and run the density evaluation procedure for the desired decoder.

3.3.1 *Continuous additive channels*

An important class of channels with practical implications is memory-less with binary input and continuous output and additive noise. Perhaps the best known example of such a channel is the binary additive white Gaussian noise (BI-AWGN) channel, for which Richardson and Urbanke provide the following equation for initial density

$$P_0\left(\frac{2}{\sigma^2}x\right) = \frac{\sigma}{2\sqrt{2\pi}} e^{\frac{-(x-1)^2}{2\sigma^2}}.$$

They derive a similar expression for the binary Laplace (BIL) channel. Using these equations, they are able to generate columns of table 1 for belief propagation over these two channel models as well.

3.3.2 *Physical degradation*

For each of the three channels considered so far, namely BSC, BIAWGN, and BIL, there is a real-valued channel parameter that reflects a natural ordering with respect to channel capacity. That is, as the channel parameter increases, the capacity decreases. In some cases, however, a class of channels may not have such a natural ordering, yet it would be useful if one could still define a partial ordering with respect to a particular choice of code and decoder.

Richardson and Urbanke prove just such a result for a well-known class of channels that can be regarded as physically degraded [14, Thm. 1]. Let channel W have transition probability $p_W(y|x)$. We say W' is physically degraded with respect to W if $p_{W'}(y'|x) = P_Q(y'|y)p_W(y|x)$ for some auxiliary channel Q. They prove that for two memory-less channels W and W' satisfying the symmetry conditions expressed earlier, where W' is physically degraded with respect to W, a belief-propagation decoder will perform at least as well on W as it does on W'. Formally, let p be the expected fraction of incorrect messages passed (again, with appropriate independence assumptions) in the jth round of decoding a message passed over channel W, and let p' correspond to the value over W'. Then $p \leq p'$.

This monotonicity guarantee has important consequences for practical implementations of decoders, as channels observed in practice can often be considered to be composed of multiple primitive channels. For instance, concatenations of BSC, BIAWGN channels, Cauchy channels, and even the erasure channels we shall introduce shortly are all monotone with respect to a belief propagation decoder.

4 Random graphs

The previous analysis made significant assumptions about the independence of messages used during the decoding process. Gallager deliberately constructed codes satisfying the required constraints. Luby, Mitzenmacher, Shokrollahi, and Spielman further showed that such graphs can be obtained by selecting a random member of an ensemble [8]. Their "concentration theorem" shows that a random graph will do; the behavior of any member of the ensemble converges to the expectation exponentially fast in the block length. They initially proved the theorem for Gallager's hard-decision decoders in the BSC, but Richardson and Urbanke extended it for message-passing decoders in an arbitrary channel model [14], which we will show here. Before doing so, however, we first discuss how to construct ensembles of random graphs and observe a fact about the tree-like structure of random graphs.

4.1 Ensembles

For a (d_v, d_c)-regular code of length n, we define the ensemble $C^n(d_v, d_c)$ in the following fashion. Consider labeling the message and check nodes, ordering the nd_v edges in the graph, and connecting them in order to the message nodes. Hence, edges $e_{d_v(i-1)+1} \ldots e_{d_v i}$ are adjacent to message node i. To connect the edges to check nodes, we define a permutation π on the set $\{1 \ldots nd_v\}$. For each edge e_i, let $e_i = (i, \pi(i))$. This induces a uniform distribution on the ensemble $C^n(d_v, d_c)$. Both Luby, Mitzenmacher, Shokrollahi, and Spielman [8] and Richardson and Urbanke [14] note that while multiedges are strictly allowed, codes perform better in practice if they are removed.

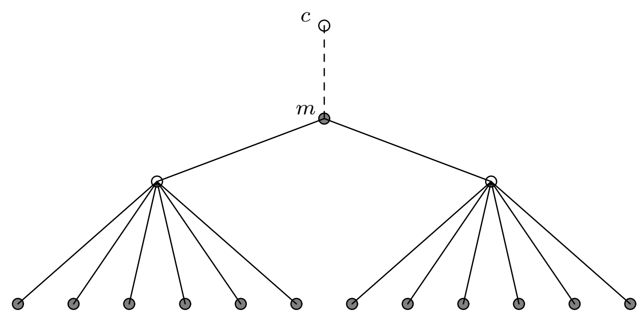

Figure 2: A tree-like decoding neighborhood, $\mathcal{N}_e^2$, of depth 2 about $e = (m, c)$. The message $(m, c)_1$ depends on all of the message nodes at the base of the tree. (Adapted from [8, fig. 1] and [14, fig. 2].)

4.2 Tree-like neighborhoods

Let $\mathcal{N}_e^{2j}$ denote the directed neighborhood of an edge $e = (m, c)$ of depth $2j$. Figure 2 depicts $\mathcal{N}_e^2$. It turns out that the independence assumption made previously, namely that an extrinsic message set on an edge $e = (m, c)$ in round j is independent of all previous messages sent on (c, m), is equivalent to requiring there are no cycles in the directed neighborhood of depth $2j$. We call such a neighborhood tree-like.

For a random (d_v, d_c)-regular graph, there exists a constant γ depending on j and the maximum degree such that the probability that $\mathcal{N}_e^{2j}$ is not tree-like is $\leq \gamma/n$. This is proven by Richardson and Urbanke [14, App. A]; we give an intuition due to Luby, Mitzenmacher, Shokrollahi, and Spielman.

There are fewer than $(d_v d_c)^j$ nodes in $\mathcal{N}_e^{2j}$. Consider exposing the neighborhood edge by edge. The probability that the exposed edge is incident on a node already in the neighborhood is clearly bounded above by $d_v d_c (d_v d_c)^j / (n - (d_v d_c)^j)$. Thus, by the union bound, the probability of any exposed edge being a member of the neighborhood previously is at most $(d_v d_c)^{2j+1}/(n - (d_v d_c)^j)$, which can be made less than γ/n for an appropriate choice of γ dependent only on j and the maximum degree.

4.3 Concentration theorem

We now turn to the concentration theorem, first proved by Luby, Mitzenmacher, Shokrollahi, and Spielman [8, Thm.

1], and trivially generalized by Richardson and Urbanke [14, Thm. 2]. We will use the notation from Richardson and Urbanke. Formally, for some integer $j > 0$, let Z_j be a random variable representing the number of edges that pass incorrect messages from message nodes to check nodes in round j of a message passing algorithm, and let p be the expected fraction of incorrect messages passed along edges with a tree-like neighborhood of depth at least $2j$ at the jth iteration. There exists a positive constant β such that for any $\epsilon > 0$,

$$\Pr[|Z - nd_v p| > nd_v \epsilon] \leq 2e^{-\beta\epsilon^2 n}. \tag{6}$$

This implies that if a message-passing decoder converges for some error probability p that converges to 0, there exists a sufficiently large n such that it correctly decodes all but an arbitrarily small fraction of message nodes with high probability.

We first observe that

$$|E[Z] - nd_v p| < nd_v \epsilon/2. \tag{7}$$

This follows from linearity of expectation. Let $E[Z_i]$ be the expected number of incorrect messages passed along edge e_i, averaged over all possible graphs and decoder inputs. By symmetry,

$$E[Z] = \sum_{i \in [nd_v]} E[Z_i] = nd_v E[Z_1].$$

Note $E[Z_1]$ can be written conditionally as

$$\begin{aligned} E[Z_1] = {} & E[Z_1 | \mathcal{N}_{e_1}^{2j} \text{ tree} - \text{like}] \Pr[\mathcal{N}_{e_1}^{2j} \text{ tree} - \text{like}] \\ & + E[Z_1 | \mathcal{N}_{e_1}^{2j} \text{ not tree} - \text{like}] \Pr[\mathcal{N}_{e_1}^{2j} \text{ not tree} - \text{like}], \end{aligned}$$

so it follows that

$$nd_v p \left(1 - \frac{\gamma}{n}\right) \leq E[Z] \leq nd_v \left(p + \frac{\gamma}{n}\right).$$

The desired bound follows, provided $n > 2\gamma/\epsilon$.

We are now prepared to give an edge-exposure Martingale argument to prove the theorem, due to Richardson and Urbanke. Consider $(G, R) \in \Omega$, where G is a graph in the ensemble, R is setting of the variables, and Ω is the respective probability space. We now define a refinement $=_i$ for $0 \leq i \leq (d_v + 1)n$, ordered by partial equality. That is to say $(G, R) =_i (G', R')$ implies $(G, R) =_{i-1} (G', R')$. Now consider proceeding down the equivalence classes of this refinement. In particular, for the first $d_v n$ steps, we expose the edges of the graph G in some order. The last d_v steps simply expose the settings of the variables. $(G, R) =_i (G', R')$ implies that the information revealed up to the ith step is the same for both pairs.

We now define Z_i as the expectation of Z (the number of edges set to pass incorrect messages from message to check nodes in round j) in (G, R) given some refinement i:

$$Z_i(G, R) = E[Z(G', R') | (G', R') =_i (G, R)].$$

Clearly $Z_0 = E[Z]$ and $Z_{(d_v+1)n} = Z$; hence, by construction, $Z_0, Z_1, \ldots, Z_{(d_v+1)n}$ forms a Doob's Martingale [13, ex. 4.2].

We claim that consecutive values differ only by a constant, that is

$$|Z_{j+1}(G, R) - Z_j(G, R)| \leq \alpha_j,$$

for some α_i that depends only on j and the maximum degree. Richardson and Urbanke provide a formal proof; we provide the intuition of Luby, Mitzenmacher, Shokrollahi, and Spielman, which is based on another edge-exposure argument. The value of $|Z_{j+1}(G, R) - Z_j(G, R)|$ is clearly bounded by the maximum difference between conditional expectations. Intuitively, for $j \leq nd_v$, this is bounded by the maximum difference between any two graphs that differ in the placement of two edges (since the graphs are defined by permutations, if they differ in one edge, they must differ in at least two). The placement of two graph edges can only affect a constant (in terms of j and the maximum degree) number of trees. Differences in the last n exposures, that is a setting of the variables, can affect only the neighborhood (with depth $2j$) of that variable, the size of which is again clearly constant in j and the maximum degree.

We can now apply Azuma's inequality [13, Thm. 4.16] to the Martingale, which says

$$\Pr[|Z_m - Z_0| > nd_v \epsilon/2] \leq 2e^{-\beta\epsilon^2 n},$$

for some constant β dependent on d_v and α_i (Richardson and Urbanke show $1/(544 d_v^{2j} d_c^{2j})$ will suffice). Recall $Z_0 = E[Z]$, and $Z_m = Z$; hence, we can substitute this into equation 7, which proves the theorem.

4.4 Finishing up

The theorem itself is somewhat unsatisfying, however, as it only proves that the decoder correctly decodes all but a small fraction of the bits with high probability. We would like to be assured the remaining bits can be corrected as well. It turns out the success of the remaining nodes depends on the non-existence of small cycles in the graph.

As discussed before, Gallager deliberately constructed his codes to avoid cases with small cycles. Luby, Mitzenmacher, Shokrollahi, and Spielman [7, 8] suggest a small change that can be made to any random graph to ensure it meets the necessary requirements as well. Basically, they add a small number of additional check nodes, and construct a regular graph of degree 5 between the message nodes and the new nodes.

Luby, Mitzenmacher, Shokrollahi, and Spielman then appeal to a result on expander graphs by Sipser and Spielman [16] which shows a decoder that succeeds with high probability on such graphs. It turns out this change in decoders is unnecessary, however, as Burshtein and Miller [2] later showed that a hard-decision decoding algorithm is guaranteed to succeed once it has corrected a sufficient number of message nodes, and the theorem above shows that we can correct down to an arbitrary fraction with high probability.

5 Irregular Codes

Up to this point we've considered only regular codes, as proposed by Gallager [6]. It turns out better performance can be obtained through be use of irregular graphs, as proposed by Luby, Mitzenmacher, Shokrollahi, and Spielman [8]. An irregular code allows the degree of nodes on either side of the graph to vary. They show several codes with significantly disparate node degrees that far out-perform the best regular codes under both hard-decision decoding and belief propagation.

5.1 Code construction and decoding

Ensembles of irregular graphs are constructed in the same fashion as regular graphs, except that the neither the number of edges adjacent to each message node nor each check node is constant. Luby, Mitzenmacher, Shokrollahi, and Spielman [7] introduced the following notation to describe irregular graphs. Define an edge to have left (right) degree i if the left (right) end point has degree i. Assume a graph has some maximum message node degree d_v^* and check node degree d_c^*. Then an irregular graph is specified by sequences $(\lambda_1, \lambda_2, \ldots, \lambda_{d_v^*})$ and $(\rho_1, \rho_2, \ldots, \rho_{d_c^*})$, where λ_i (ρ_i) is the fraction of edges with left (right) degree i.

Intuitively, irregular codes should out-perform regular codes for the following reason. Each message node in a regular code is equally "protected." That is to say, the number of check nodes is constant for each message node. Clearly, the larger the number, the greater the protection. Unfortunately, as check nodes increase in degree, they become less reliable, since they are dependent on more message nodes to be received correctly. Regular codes are forced to balance these requirements uniformly.

Irregular codes, on the other hand, are free to construct a certain fraction of extremely well protected message bits without diluting the value of all check bits. This leads to a type of "wave" effect (which Luby, Mitzenmacher, Shokrollahi, and Spielman report observing in practice [8]) in which the well-protected nodes are corrected first, and then propagate their results through check nodes to those less well protected.

Decoding is performed using a generalization of Gallager's Algorithm B. The algorithm is essentially identical, except that the threshold value $b_{j,i}$ is now also dependent on the degree of the message node, i.

The difficulty in constructing irregular codes is figuring out which irregular distributions perform well. It is not known how to analytically determine the best codes (values λ and ρ). Instead, Luby, Mitzenmacher, Shokrollahi, and Spielman compute a linear programming approximation [8]. Given a desired rate and fixed ρ, they determine a good λ by selecting a set L of candidate message node degrees, and attempting to satisfy the constraints given by the recursive probability enumeration shown in the following section. Additional constraints are inserted to ensure the resulting edge degree distribution is possible in a connected, bipartite graph.

Luby, Mitzenmacher, Shokrollahi, and Spielman noted that for their hard-decision decoder, experimental evidence shows that the best codes use a fixed ρ. They provide the following intuitive reasoning (which doesn't hold up under belief propagation—this is also seen experimentally). The probability a check node sends the correct message to node m in round j (receives an even number of errors) is

$$\frac{1 + \sum_i \rho_i (1 - 2p_j)^{i-1}}{2}, \tag{8}$$

which is simply a generalization of equation 1 over the probability distribution of check node degrees ρ. For brevity, we write $\rho(x) = \sum_i \rho_i x^{i-1}$, hence equation 8 is just

$$\frac{1 + \rho(1 - 2p_j)}{2}.$$

They claim, for small values of p_i, that this is approximately

$$1 - p_i \sum_{i=1}^{d_c} (i-1)\rho_i,$$

which is minimized (subject to the constraints that the edges form a connected, bipartite graph) when all check nodes have as equal a degree as possible [8].

5.2 Performance

Luby, Mitzenmacher, Shokrollahi, and Spielman provide an analytic evaluation for their hard-decision decoding algorithm, deriving a recursive expression analogous to equation 3 [8, eqn. 8]):

$$\begin{aligned} p_{j+1} = p_0 - \sum_{i=1}^{d_\ell} \lambda_i \\ \cdot \Bigg[p_0 \sum_{t=b_{j,i}}^{i-1} \binom{i-1}{t} \left[\frac{1+\rho(1-2p_j)}{2}\right]^t \\ \cdot \left[\frac{1-\rho(1-2p_j)}{2}\right]^{i-1-t} \\ + (1-p_0) \sum_{t=b_{j,i}}^{i-1} \binom{i-1}{t} \left[\frac{1-\rho(1-2p_j)}{2}\right]^t \\ \cdot \left[\frac{1+\rho(1-2p_j)}{2}\right]^{i-1-t} \Bigg]. \end{aligned} \tag{9}$$

As before, the optimal value of $b_{j,i}$ is the smallest integer solution to

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

They use these equations as constraints in the linear programming procedure described above to generate several rate $1/2$ irregular codes, which have threshold values p^* up to 0.0627 with Gallager's hard-decision decoder. Compare this to the best performing regular code from table 1, which has a p^* threshold of only 0.051.

As with regular codes, irregular codes should similarly benefit from more sophisticated decoders. Lacking the analytical tools, Luby, Mitzenmacher, Shokrollahi, and Spielman were only able to simulate the performance of a belief propagation algorithm over their codes. Using their density evaluation model, however, Richardson and Urbanke we able to numerically compute performance bounds for belief propagation over irregular codes [14]. They constructed two polynomials defined by λ and ρ

$$\lambda(x) = \sum_{i\geq 2}^{d_v^*} \lambda_i x^{i-1},\ \rho(x) = \sum_{i\geq 2}^{d_c^*} \rho_i x^{i-1}.$$

Using these polynomials, it turns out that the modifications to account for irregular graphs are minor. Equation 5 becomes

$$\begin{aligned} \hat{\hat{Q}}^{(j),0} - \hat{\hat{Q}}^{(j),1} &= \rho\left(\hat{\hat{P}}^{(j-1),0} - \hat{\hat{P}}^{(j-1),1}\right) \\ \hat{\hat{Q}}^{(j),0} + \hat{\hat{Q}}^{(j),1} &= \rho\left(\hat{\hat{P}}^{(j-1),0} + \hat{\hat{P}}^{(j-1),1}\right), \end{aligned}$$

and the resulting FFT simply

$$\mathcal{F}\left(P^{(j+1)}\right) = \mathcal{F}\left(P^{(0)}\right)\lambda\left(\mathcal{F}\left(Q^{(j)}\right)\right).$$

Using an irregular rate $1/2$ code constructed by Luby, Mitzenmacher, Shokrollahi, and Spielman, Richardson and Urbanke calculate a threshold p^* value of 0.094, which substantially out-performs the best instance of belief-propagation over a regular rate $1/2$ code, shown in table 1 to have a threshold of 0.084.

6 Tornado codes

In an erasure channel, first introduced by Elias [5], each codeword symbol is lost independently with a fixed constant probability, p. This loosely models the behavior seen in most packet-based networks, such as the Internet, as well as fail-stop distributed computing environments. Hence codes well-suited to this channel have many immediate applications.

The (Shannon) capacity of an erasure channel is $1-p$, and Elias [5] further showed any rate $R < 1-p$ can be achieved with a random linear code, including traditional LDPC codes. It is easy to show that MDS codes of rate R can recover from the loss of a fraction $(1-R)$ of their codeword symbols. The main obstacle to the use of LDPC codes in this channel is the complexity of encoding and decoding. As described previously, LDPC codewords consist of n message nodes satisfying a set of constraints imposed by the m check nodes. Encoding a message of dimension $n-m$ requires computing a codeword that satisfies the m constraints. This can clearly be done through matrix multiplication in quadratic time. Unfortunately, this does not approach the speed of efficient implementations of standard MDS codes such as Reed-Solomon. To address this, Luby, Mitzenmacher, Shokrollahi, and Spielman developed a class of rate R irregular low-density parity-check codes for the erasure channel which can recover from the loss of a $(1-\epsilon)(1-R)$ fraction of its message bits and can be both encoded and decoded in near-linear time [7].

6.1 Code construction

Luby, Mitzenmacher, Shokrollahi, and Spielman suggest avoiding satisfying constraints in the encoding process altogether. Instead, they consider computing values for the check nodes (using XOR as before), and sending their values along with the message nodes. Clearly this takes linear time, and turns out to be the same rate—while the codewords are longer ($n+m$ bits) they now have a full n degrees of freedom. While we (and they) describe the code over GF(2), it is important to note that it can be extended to any arbitrary alphabet size, which will be useful in the applications discussed in the section 7.

Decoding over the erasure channel is straightforward. Provided the value of a check node and all but one of its adjacent message nodes, the missing message node must be the XOR of the check node and the other message nodes. This does, however, require knowledge of the value of the check node, which, since it is now being sent on the channel, could be unknown. Luby, Mitzenmacher, Shokrollahi, and Spielman avoid this by cascading a series of irregular LDPC graphs, and decoding can be viewed as proceeding in stages. Hence, at each stage, the correct values of the check nodes are known. Figure 3 shows an example of such a cascade.

Formally, let $\beta = 1-R$. If a code $\mathcal{C}(B)$ with n message bits and βn check bits recovers from the loss of $(1-\epsilon)\beta n$ of its message bits, then they construct a family of codes $\mathcal{C}(B_0), \ldots, \mathcal{C}(B_m)$ where B_i has $\beta^i n$ left nodes and $\beta^{i+1} n$ right nodes. By selecting m so that $\beta^{m+1} n$ is about $\sqrt{n}$, the check nodes of the last code can be encoded using a standard quadratic time code C of rate R that can recover from the loss of a β fraction of its message bits (e.g, a Reed-Solomon code). We denote the cascaded code $\mathcal{C}(B_0, B_1, \ldots, B_m, C)$.

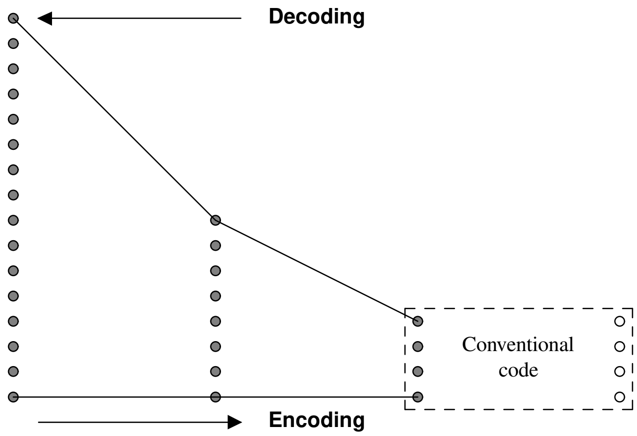

Figure 3: A cascade of three codes, depicting the direction of encoding and decoding (adapted from [7, Fig. 3]).

It is easy to verify the code has n message bits and

$$\sum_{i=1}^{m+1} \beta^i n + \beta^{m+2} n/(1-\beta) = n\beta/(1-\beta)$$

check bits, resulting in a rate of R.

6.2 Performance bounds

A trade-off exists in the performance of a particular instance of this code ensemble. Luby, Mitzenmacher, Shokrollahi, and Spielman show that the number of erasures tolerated can be made increasingly close to optimal in return for longer decoder running time [7]. They label this tuning parameter D, and prove that the code resulting from cascading versions of the construction below with $D = \lceil 1/\epsilon \rceil$ for a sufficiently large n is a linear code, that, for any $0 < R < 1$ and $0 < \epsilon < 1$, can be successfully decoded in time $O(n \ln 1/\epsilon)$ even in the face of a $(1-R)(1-\epsilon)$ fraction of erasures with probability $1 - O(n^{-3/4})$ [7, Thm. 3].

We now give a brief overview of their suggested construction. The code used in each level of the cascade is based upon a member of an irregular LDPC ensemble as described in the preceding section, hence we denote the message and check node degrees as λ and ρ as before. Let $\mathcal{C} = \mathcal{C}(B_0, \ldots, B_m, C)$, where B_0 has n left nodes, and suppose each B_i is chosen at random from the ensemble specified by λ and ρ with $\lambda_1 = \lambda_2 = 0$, and δ is such that

$$\rho(1 - \delta\lambda(x)) > 1 - x$$

(where $\lambda(x)$ and $\rho(x)$ are as before) for all $0 < x \leq 1$. Luby, Mitzenmacher, Shokrollahi, and Spielman show that if at most a δ-fraction of the codeword symbols are erased independently at random, their decoding algorithm succeeds with probability $1 - O(n^{-3/4})$ [7, Thm. 2].[2]

A particular class of codes *almost* satisfying this requirement is given by

$$\lambda_i = 1/(H(D)(i-1)), \quad \rho_i = \frac{e^{-\alpha}\alpha^{i-1}}{(i-1)!},$$

where $H(x)$ is the harmonic sum, and α_i is chosen to ensure the appropriate average check node degree, $\bar{d}_c$. Namely, $\alpha\epsilon^{\alpha}/(e^{\alpha} - 1) = \bar{d}_c = \bar{d}_v\beta$. We say almost because $\lambda_2 \neq 0$. As was done to avoid the small cycle problem previously, Luby, Mitzenmacher, Shokrollahi, and Spielman describe a small modification to B to obtain a satisfactory graph with roughly the same properties [7], and term this class of distributions "heavy tail."

7 Implementation issues

As described above, tornado codes seem like an attractive tool for communicating over erasure channels commonly found in computer systems. It is not surprising, then, the they have been proposed for use in a number of networking applications, including bulk transfer [4], parallel downloading [3], and layered multicast. Tornado codes are not a panacea, however, and in this section we detail several pragmatic considerations that apply to real-world implementations.

7.1 Decoding complexity

There is no debating tornado codes are fast. Theoretical analysis of complexity in terms of big-Oh notation is useful to an extent, but the utility of actual implementations depends a great deal on the constant factors hidden by this analysis. The performance of tornado Codes [7] has been shown to be far superior to Reed-Solomon codes [4]. This stems to a certain degree from the differences in complexity. If an encoded message has k message bits and l check bits, tornado code decoding is linear in $(k+l)$, while Reed-Solomon is linear in kl. Equally as important in many applications, however, is that tornado decoding is fundamentally an XOR operation, while Reed-Solomon requires field operations. In fact, as the size of the field grows larger, tornado operations become even faster, as CPUs typically perform operations many (32 or 64) bits at a time, hence 32 XORs are often as fast as 1.

7.2 Decoding inefficiency

It is often useful to think of the bandwidth overhead of a coding scheme in terms of its decoding inefficiency. A message of dimension k encoded at rate R will give rise to a codeword of dimension k/R. If we think of each codeword symbol as having dimension one, then receiving k symbols is the minimum required to express the message. In an MDS erasure code, receiving exactly k symbols is sufficient. Tornado codes are only approximately MDS, however.

[2]This seemingly magical constraint is the result of a sophisticated differential-equation-based analysis [7, Sec. III] that we will not discuss here.

In fully-cascaded form, a tornado code with message dimension k must be cascaded until the last layer has at most $\sqrt{k}$ message nodes, as a standard-erasure code requires quadratic time to encode. For any rate, R, such a code has $O(\log_{1/R} k)$ levels. In the analysis of the preceding section, we assumed erasures were evenly distributed across levels. Of course, there is some variance in this distribution. For a rate $1/2$ code in the erasure channel model described above, the expected variance in the last level is $1/\sqrt[4]{k} = 0.063$, hence we expect to require 1.063 times the message length of the codeword before decoding is successful [7].

Luby, Mitzenmacher, Shokrollahi, and Spielman suggest ameliorating this issue by using many fewer layers, and continuing to use a random graph for the last level. In particular, they report a code based on a three-layer cascade that requires only 1.033 times the codeword length to decode, and another, two-layer code, named "Tornado Z," that requires 1.054 times optimal [4].

When using erasure codes over a packet-switched channel, there is even more overhead. Throughout this paper, we have assumed that we knew the associated node for each variable received on the channel. That is, the output of the channel was associated with the appropriate node in the LDPC graph. This can often be done through temporal ordering. For an erasure channel, however, assignment based on the order received is doomed, as any erasure will cause the assignments to become mis-aligned. Further, many network channels of interest (e.g. the Internet) reorder packets as well, so even without erasures, assignment is not straightforward.

Hence, each symbol of a codeword (packet) must be appropriately annotated with the node whose value it represents. While the size of this annotation only grows as $O(\log n)$, the necessary data framing and marshaling suggest that each codeword symbol (packet) must be fairly long in comparison. Hence a practical implementation over a packet network would likely work over an alphabet several orders of magnitude larger than GF(2).

7.3 Block size

The obvious application for tornado codes is in block data transfer. This was proposed by Byers, Luby, Mitzenmacher, and Rege [4], and now forms the basis of the core technology for an Internet content distribution company called Digital Fountain. By considering entire files (or blocks of files) as messages, tornado codes can be used to break the file up into many symbols (packets) which can be transmitted. For instance, using a rate 1/2 encoding, the file is expanded into a set of packets twice its size, the receipt of slightly over any half of them is sufficient to decode the file.

As with any block code, however, decoding requires operating over the entire message length at once. This has several implications.

7.3.1 Memory usage

As symbol sizes become larger, the memory requirements of decoding an extremely long message grow rapidly. This is especially important as efficiency goes up with both the size of the codeword symbols and the length of the message increase. Additionally, the proofs of decoding success provided earlier all depended on a "sufficiently large" message length, where n is non-trivial. Tornado Z, for example, has 32,000 nodes. If each node is a $\approx$ 1500-byte packet (maximum efficiency on an Ethernet network), decoding requires accessing 46 Megabytes of memory. Admittedly, this can be paged to disk during decoding, but for high degree graphs (Luby, Mitzenmacher, Shokrollahi, and Spielman constructed graphs with degree 85) the working set will be quite large, hence thrashing is likely.

7.3.2 Streaming

A hot topic in networking research and product development today is streaming. Streaming is the process of delivering data to a client at or above the rate at which it can be consumed by the client, supporting simultaneous playback or viewing of the received data. Audio and video content are typical candidates for streaming delivery. One of the biggest problems in streaming data is determining the rate at which a consumer can receive the data, and adjusting accordingly. Clearly if the client cannot consume data at a rate fast enough to support playback this is infeasible, but Internet hosts often have unstable bandwidth capacities, hence even while the long-term average is sufficient, short term variations may cause data packets to be lost.

Erasure coding obviates the need to retransmit dropped packets to clients. In principle, if the message length is long enough to see long-term average receive rates, and they are large enough to support the information rate, it suffices simply to send the message symbols at a speed inversely proportional to the information rate (rate $1/2$ codes must be sent at twice as fast as rate 1 codes to provide the same amount of content), and ignore lost packets.

The problem, of course, is that tornado codes (indeed, any block codes) cannot be decoded until the entire message is received. Hence true streaming is impossible using such coding schemes. Instead, it is possible to simulate streaming by breaking the stream up into blocks and sending a block at a time, playing back one block while receiving the next. This is the approach taken by Digital Fountain. Unfortunately, this has an obvious drawback: latency. Both encoding and decoding must be delayed by a full block size. Further, encoding cannot commence until the entire block is available. In the case of recording a live data stream, the encoder must parallelize encoding a block with beginning to record the following block.

7.4 Redundant delivery

An attractive consequence of the independence of symbols within a codeword is that, assuming the rate is low enough, a client can arrange to receive symbols from a number of sources. For instance, consider encoding a file at rate 1/2 into blocks $b_1, \ldots, b_n$, and serving the encoded data stream from two servers. By staggering the delivery such that at any point in time server 1 is sending b_t and server 2 is sending $b_{(t+n/2 \bmod n)}$, a client can receive the file in slightly over half the time (assuming it is able to consume the packets at the rate they are being sent from each server).

Clearly the code rate can be set sufficiently low that the chance of receiving redundant packets, even from uncoordinated servers, is arbitrarily low. This represents an attractive opportunity for network clients with disjoint bottlenecks along paths to multiple servers. That is to say clients which, while receiving data at full capacity from one server, can additionally receive data from a different server simultaneously without decreasing the rate of the first flow. Such a scheme was proposed recently by Byers, Luby, and Mitzenmacher [3]. They note, however, that the number of clients meeting the disjoint bottleneck requirement is small.

7.5 Channel assumptions

Unfortunately, while the erasure channel provides a good basic model of the Internet "channel," it is far from exact. In particular, the Internet is a shared channel, hence optimality cannot be considered from the point of view of one flow or client alone. Secondly, routers in the Internet maintain queues, which function to defeat the memory-less assumption. These two facts cause the actual utility of tornado codes to be somewhat lower than one might imagine.

7.5.1 Congestion

The major drawback of both applications described above is that they waste bandwidth. From the point of view of the client, these schemes are great—it receives data at close to the optimum rate of the channel. From the point of view of the network, however, these schemes are extremely wasteful. The capacity of an Internet path is limited by the bottleneck link, that is the edge along the path with the smallest capacity. Packets arriving at that edge at a rate exceeding the link capacity are dropped (not exactly, we'll return to this in a moment). Hence while they do not affect the capacity of the bottleneck link, they are competing with other traffic for scarce capacity at every point prior. If the source does not adjust its rate of sending, these extra packets will continue to steal resources from other flows while providing no marginal utility to the client.

7.5.2 Queues

Further, the Internet is far from memory-less. The Internet is a store-and-forward network, meaning each packet is stored at each router before being forwarded on the next link on the path to the destination. If the rate of incoming packets exceeds the capacity of the outgoing links, a queue builds up at the router. Most routers in the Internet today perform drop-tail queuing, which means packets are simply dropped as the queue overflows. This leads to a high correlation between packet drops, and destroys the independence assumptions upon which tornado codes are based. This bursty loss model may negatively affect the decoding success probability. One might hypothesize that RED (Random, Early Detection) routers, which attempt to drop packets at something approaching a rate proportional to the size of the flow, might provide a better substrate for tornado codes.

8 Conclusion

In this paper, we have summarized the state of the art in low-density parity-check codes, with particular emphasis on the work of Luby, Mitzenmacher, Shokrollahi, Spielman [7, 8], and Richardson and Urbanke [14]. We introduced Gallager's initial, regular LDPC codes, and gave a brief performance analysis of his proposed decoding algorithms. We then demonstrated the additional performance gains achieved by considering decoders with expanded alphabets, and extending the family of codes to include those based on random and irregular bipartite graphs.

We then examined the application of LDPC codes to the erasure channel, as pioneered by Luby, Mitzenmacher, Shokrollahi, and Spielman. After explaining the construction of their turbo codes, we provided highlights of their performance analysis. Finally, we considered the application of LDPC-based erasure codes to several problems in networking, and indicated several issues of concern for practical deployment of such schemes. In particular, the requirements for large block and symbol sizes, combined with the mischaracterization of the Internet channel, give pause, especially when deploying protocols based on tornado codes without appropriate throttling on bandwidth utilization.

References

[1] BERROU, C., GLAVIEUX, A., AND THITIMAJSHIMA, P. Near shannon limit error-correcting coding and decoding: Turbo-codes. In *Proc. IEEE International Communications Conference* (1993).

[2] BURSHTEIN, D., AND MILLER, G. Expander graph arguments for message passing algorithms. *IEEE Trans. on Information Theory 47* (Feb. 2001), 782–790.

[3] BYERS, J. W., LUBY, M., AND MITZENMACHER, M. Accessing multiple mirror sites in parallel: Using tor-

nado codes to speed up downloads. In *Proc. IEEE Infocom* (Mar. 1999), pp. 275–283.

[4] BYERS, J. W., LUBY, M., MITZENMACHER, M., AND REGE, A. A digital fountain approach to reliable distribution of bulk data. In *Proc. ACM SIGCOMM* (Aug. 1998), pp. 56–67.

[5] ELIAS, P. Coding for two noisy channels. In *Proc. Third London Symposium on Information Theory* (1955), pp. 61–76.

[6] GALLAGER, R. G. *Low-Density Parity-Check Codes*. PhD thesis, Massachusetts Institute of Technology, 1963.

[7] LUBY, M. G., MITZENMACHER, M., SHOKROLLAHI, M. A., AND SPIELMAN, D. A. Efficient erasure correcting codes. *IEEE Trans. on Information Theory 47*, 2 (Feb. 2001), 509–584.

[8] LUBY, M. G., MITZENMACHER, M., SHOKROLLAHI, M. A., AND SPIELMAN, D. A. Improved low-density parity-check codes using irregular graphs. *IEEE Trans. on Information Theory 47*, 2 (Feb. 2001), 585–598.

[9] MACKAY, D. J. C. Turbo codes are low density parity check codes. `http://www.cs.toronto.edu/~mackay/abstracts/turbo-ldpc.html`.

[10] MACKAY, D. J. C. Good error correcting codes based on very sparse matrices. *IEEE Trans. on Information Theory 45* (Mar. 1999), 399–431.

[11] MACKAY, D. J. C., MCELIECE, R. J., AND CHENG, J.-F. Turbo coding as an instance of pearl's 'belief propagation' algorithm. *IEEE J. on Selected Areas in Communications 17* (1999), 1632–1650.

[12] MITZENMACHER, M. A note on low density parity check codes for erasures and errors. SRC technical note 1998-017, Dec. 1998.

[13] MOTWANI, R., AND RAGHAVAN, P. *Randomized Algorithms*. Cambridge University Press, 1995.

[14] RICHARDSON, T. J., AND URBANKE, R. L. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Trans. on Information Theory 47*, 2 (Feb. 2001), 599–628.

[15] SHANNON, C. E. A mathematical theory of communication. *Bell System Technical Journal 27* (July 1948), 379–423.

[16] SIPSER, M., AND SPIELMAN, D. Expander codes. *IEEE Trans. on Information Theory 42* (Nov. 1996), 1710–1722.

[17] VANSTONE, S. A., AND VAN OORSCHOT, P. C. *An Introduction to Error Correcting Codes with Applications*. Klewer Academic Publishers, 1989.