LDPC Codes: An Introduction

Amin Shokrollahi

Digital Fountain, Inc.
39141 Civic Center Drive, Fremont, CA 94538
amin@digitalfountain.com

April 2, 2003

Abstract

LDPC codes are one of the hottest topics in coding theory today. Originally invented in the early 1960's, they have experienced an amazing comeback in the last few years. Unlike many other classes of codes LDPC codes are already equipped with very fast (probabilistic) encoding and decoding algorithms. The question is that of the design of the codes such that these algorithms can recover the original codeword in the face of large amounts of noise. New analytic and combinatorial tools make it possible to solve the design problem. This makes LDPC codes not only attractive from a theoretical point of view, but also perfect for practical applications. In this note I will give a brief overview of the origins of LDPC codes and the methods used for their analysis and design.

1 Introduction

This note constitutes an attempt to highlight some of the main aspects of the theory of low-density parity-check (LDPC) codes. It is intended for a mathematically mature audience with some background in coding theory, but without much knowledge about LDPC codes.

The idea of writing a note like this came up during conversations that I had with Dr. Khosrovshahi, head of the Mathematics Section of the Institute for Studies in Theoretical Physics and Mathematics in December 2002. The main motivation behind writing this note was to have a written document for Master's and PhD students who want to gain an understanding of LDPC codes at a level where they feel comfortable deciding whether or not they want to pursue it as their research topic. The style is often informal, though I have tried not to compromise exactness.

The note is by no means a survey. I have left out a number of interesting aspects of the theory, such as connections to statistical mechanics, or artificial intelligence. The important topics of general Tanner graphs, and factor graphs have also been completely omitted. Connections to Turbo codes have also been left untouched.

My emphasis in writing the notes has been on algorithmic and theoretical aspects of LDPC codes, and within these areas on statements that can be proved. I have not discussed any of the existing and very clever methods for the construction of LDPC codes, or issues regarding their implementation.

Nevertheless, I hope that this document proves useful to at least some students or researchers interested in pursuing research in LDPC codes, or more generally codes obtained from graphs.

2 Shannon's Theorem

The year 1948 marks the birth of information theory. In that year, Claude E. Shannon published his epoch making paper [1] on the limits of reliable transmission of data over unreliable channels and methods on how to achieve these limits. Among other things, this paper formalized the concept of information, and established bounds for the maximum amount of information that can be transmitted over unreliable channels.

A communication channel is usually defined as a triple consisting of an input alphabet, an output alphabet, and for each pair (i, o) of input and output elements a transition probability $p(i, o)$. Semantically, the transition probability is the probability that the symbol o is received given that i was transmitted over the channel.

Given a communication channel, Shannon proved that there exists a number, called the capacity of the channel, such that reliable transmission is possible for rates arbitrarily close to the capacity, and reliable transmission is not possible for rates above capacity.

The notion of capacity is defined purely in terms of information theory. As such it does not guarantee the existence of transmission schemes that achieve the capacity. In the same paper Shannon introduced the concept of *codes* as ensembles of vectors that are to be transmitted. In the following I will try to motivate the concept. It is clear that if the channel is such that even one input element can be received in at least two possible ways (albeit with different probabilities), then reliable communication over that channel is not possible if only single elements are sent over the channel. This is the case even if multiple elements are sent that are not correlated (in a manner to be made precise). To achieve reliable communication, it is thus imperative to send input elements that are correlated. This leads to the concept of a code, defined as a (finite) set of vectors over the input alphabet. We assume that all the vectors have the same length, and call this length the *block length* of the code. If the number of vectors is $K = 2^k$, then every vector can be described with k bits. If the length of the vectors is n, then in n times use of the channel k bits have been transmitted. We say then that the code has a rate of k/n bits per channel use, or k/n bpc.

Suppose now that we send a codeword, and receive a vector over the output alphabet. How do we infer the vector that we sent? If the channel allows for errors, then there is no general way of telling which codeword was sent with absolute certainty. However, we can find the most likely codeword that was sent, in the sense that the probability that this codeword was sent given the observed vector is maximized. To see that we really can find such a codeword, simply list all the K codewords, and calculate the conditional probability for the individual codewords. Then find the vector or vectors that yield the maximum probability and return one of them. This decoder is called the maximum likelihood decoder. It is not perfect: it takes a lot of time (especially when the code is large) and it may err; but it is the best we can do.

Shannon proved the existence of codes of rates arbitrarily close to capacity for which the probability of error of the maximum likelihood decoder goes to zero as the block length of the code goes to infinity. (In fact, Shannon proved that the decoding error of the maximum likelihood decoder goes to zero exponentially fast with the block length, but we will not discuss it here.)

Codes that approach capacity are very good from a communication point of view, but Shannon's theorems are non-constructive and don't give a clue on how to find such codes. More importantly, even if an oracle gave us sequences of codes that achieve capacity for a certain rate, it is not clear how to encode and decode them efficiently. Design of codes with efficient encoding and decoding algorithms which approach the capacity of the channel is the main topic of this note.

Before I close this section, let me give an example of two communication channels: the binary erasure channel (BEC), and the binary symmetric channel (BSC). These channels are described in Figure 1. In both cases the input alphabet is binary, and the elements of the input alphabet are called *bits*. In the case of the binary erasure channel the output alphabet consists of 0, 1, and

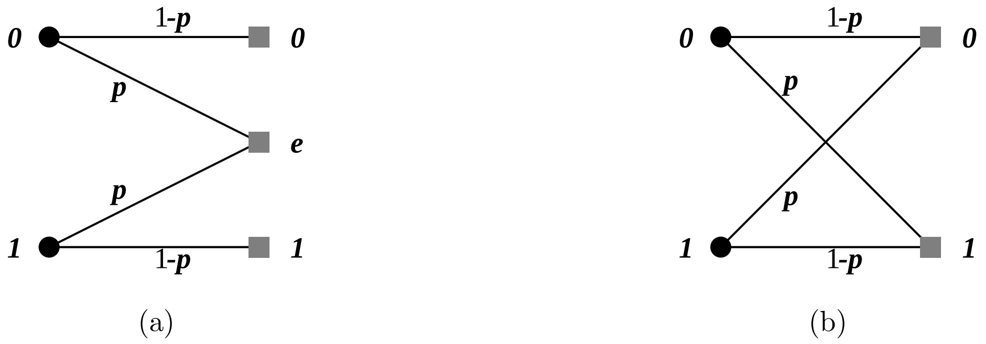

Figure 1: Two examples of channels: (a) The Binary Erasure Channel (BEC) with erasure probability p, and (b) The Binary Symmetric Channel (BSC) with error probability p

an additional element denoted e and called *erasure*. Each bit is either transmitted correctly (with probability $1 - p$), or it is erased (with probability p). The capacity of this channel is $1 - p$.

In the case of the BSC both the input and the output alphabet is $\mathbb{F}_2$. Each bit is either transmitted correctly with probability $1 - p$, or it is *flipped* with probability p. This channel may seem simpler than the BEC at first sight, but in fact it is *much* more complicated. The complication arises since it is *not* clear which bits are flipped. (In the case of the BEC it is clear which bits are erased.) The capacity of this channel is $1 + p \log_2(p) + (1 - p) \log_2(1 - p)$. Maximum likelihood decoding for this channel is equivalent to finding, for a given vector of length n over $\mathbb{F}_2$, a codeword that has the smallest Hamming distance from the received word. It can be shown that maximum likelihood decoding for the BSC is NP-complete [2]. In contrast, for linear codes maximum likelihood decoding on the BEC is polynomial time, since it can be reduced to solving a system of equations (cf. [3]).

3 Algorithmic Issues

Soon after Shannon's discoveries researchers found that random codes are capacity achieving. In fact, this is implicit in Shannon's treatise itself.

But achieving capacity is only part of the story. If these codes are to be used for communication, one needs fast algorithms for encoding and decoding. Note that random codes of rate R bpc are just 2^{Rn} random vectors of length n over the input alphabet. We need some description of these vectors to be able to embed information into them, or we need to write all of them down into a so-called codebook describing which sequence of Rn bits gets mapped to which codeword. This requires a codebook of size 2^{Rn}, which is too big for any reasonably sized code (say of length 1000 and rate 0.5, which yields 2^{500} vectors—too large to handle).

If the input alphabet has the structure of a field (for example the binary alphabet which yields the field $\mathbb{F}_2$), then one can do better, at least as far as encoding goes. Elias and Golay independently introduced the concept of linear codes of block length n and dimension k defined as subspaces of the vector space $\mathbb{F}_2^n$. Such codes have rate k/n (we will omit the unit bpc for binary codes from now on), and since they are linear, they can be described in terms of a basis consisting of k vectors of length n. A codebook can now be implicitly described in a natural manner by mapping a bit vector $(x_1, \ldots, x_k)$ to the vector obtained by taking linear combinations of the basis vectors given by the coefficients $x_1, \ldots, x_k$.

The class of linear codes is very rich. Shannon's arguments can be used (almost verbatim) to show that there are sequences of *linear codes* with rates arbitrarily close to capacity and for which the error probability of the maximum likelihood decoder approaches zero (exponentially fast) as the block length goes to infinity. Moreover, it can also be shown that random linear codes achieve capacity. Unlike their non-linear brethren, linear codes can be encoded in polynomial time, rather than exponential time. This is good news.

How about decoding? The decoding problem seems much tougher. As was mentioned above, the maximum likelihood problem on the BSC has been shown to be NP-hard for many classes of linear codes (e.g., general linear codes over $\mathbb{F}_q$ for any q). It is therefore unlikely to find polynomial time algorithms for maximum likelihood decoding of general linear codes. One way to get around this negative result is to try to repeat the success story for the encoding problem and to specialize to subclasses of general linear codes. However, we have not been able to find subclasses of linear codes for which maximum likelihood decoding is polynomial time *and* which achieve capacity.

Another possiblity is to look at sub-optimal algorithms that are polynomial time by construction. This is the path we will follow in the next section.

4 LDPC Codes

LDPC codes were invented by Robert Gallager [4] in his PhD thesis. Soon after their invention, they were largely forgotten, and reinvented several times for the next 30 years. Their comeback is one of the most intriguing aspects of their history, since two different communities reinvented codes similar

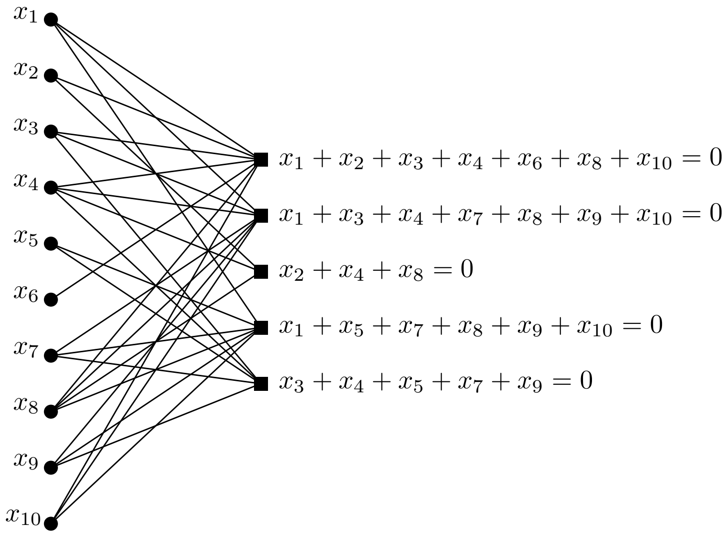
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
$x_1 + x_2 + x_3 + x_4 + x_6 + x_8 + x_{10} = 0$
$x_1 + x_3 + x_4 + x_7 + x_8 + x_9 + x_{10} = 0$
$x_2 + x_4 + x_8 = 0$
$x_1 + x_5 + x_7 + x_8 + x_9 + x_{10} = 0$
$x_3 + x_4 + x_5 + x_7 + x_9 = 0$

Figure 2: An LDPC code

to Gallager's LDPC codes at roughly the same time, but for entirely different reasons.

LDPC codes are linear codes obtained from sparse bipartite graphs. Suppose that $\mathcal{G}$ is a graph with n left nodes (called message nodes) and r right nodes (called check nodes). The graph gives rise to a linear code of block length n and dimension at least $n - r$ in the following way: The n coordinates of the codewords are associated with the n message nodes. The codewords are those vectors $(c_1, \ldots, c_n)$ such that for all check nodes the sum of the neighboring positions among the message nodes is zero. Figure 2 gives an example.

The graph representation is analogous to a matrix representation by looking at the adjacency matrix of the graph: let H be a binary $r \times n$-matrix in which the entry (i, j) is 1 if and only if the ith check node is connected to the jth message node in the graph. Then the LDPC code defined by the graph is the set of vectors $c = (c_1, \ldots, c_n)$ such that $H \cdot c^\top = 0$. The matrix H is called a *parity check matrix* for the code. Conversely, any binary $r \times n$-matrix gives rise to a bipartite graph between n message and r check nodes, and the code defined as the null space of H is precisely the code associated to this graph. Therefore, any linear code has a representation as a code associated to a bipartite graph (note that this graph is not uniquely defined by the code). However, not every binary linear code has a representation by a *sparse* bipartite graph.[1] If it does, then the code is

[1] To be more precise, sparsity only applies to *sequences* of matrices. A sequence of $m \times n$-matrices is called c-sparse if mn tends to infinity and the number of nonzero elements in these matrices is always less than $c \max(m, n)$.

called a low-density parity-check (LDPC) code.

The sparsity of the graph structure is key property that allows for the algorithmic efficiency of LDPC codes. The rest of this note is devoted to elaborating on this relationship.

5 Decoding Algorithms: Belief Propagation

Let me first start by describing a general class of decoding algorithms for LDPC codes. These algorithms are called *message passing algorithms*, and are iterative algorithms. The reason for their name is that at each round of the algorithms messages are passed from message nodes to check nodes, and from check nodes back to message nodes. The messages from message nodes to check nodes are computed based on the observed value of the message node and some of the messages passed from the neighboring check nodes to that message node. An important aspect is that the message that is sent from a message node $\mathtt{v}$ to a check node $\mathtt{c}$ must not take into account the message sent in the previous round from $\mathtt{c}$ to $\mathtt{v}$. The same is true for messages passed from check nodes to message nodes.

One important subclass of message passing algorithms is the *belief propagation* algorithm. This algorithm is present in Gallager's work [4], and it is also used in the Artificial Intelligence community [5]. The messages passed along the edges in this algorithm are probabilities, or beliefs. More precisely, the message passed from a message node $\mathtt{v}$ to a check node $\mathtt{c}$ is the probability that $\mathtt{v}$ has a certain value given the observed value of that message node, and all the values communicated to $\mathtt{v}$ in the prior round from check nodes incident to $\mathtt{v}$ other than $\mathtt{c}$. On the other hand, the message passed from $\mathtt{c}$ to $\mathtt{v}$ is the probability that $\mathtt{v}$ has a certain value given all the messages passed to $\mathtt{c}$ in the previous round from message nodes other than $\mathtt{v}$.

It is easy to derive formulas for these probabilities under a certain assumption called *independence assumption*, which I will discuss later. It is sometimes advantageous to work with likelihoods, or sometimes even log-likelihoods instead of probabilities. For a binary random variable x let $L(x) = \Pr[x = 0]/\Pr[x = 1]$ be the *likelihood* of x. Given another random variable y, the *conditional likelihood* of x denoted $L(x \mid y)$ is defined as $\Pr[x = 0 \mid y]/\Pr[x = 1 \mid y]$. Similarly, the *log-likelihood* of x is $\ln L(x)$, and the *conditional log-likelihood* of x given y is $\ln L(x \mid y)$.

If x is an equiprobable random variable, then $L(x \mid y) = L(y \mid x)$ by Bayes' rule. Therefore, if

$y_1, \ldots, y_d$ are independent random variables, then we have

$$\ln L(x \mid y_1, \ldots, y_d) = \sum_{i=1}^{d} \ln L(x \mid y_i). \tag{1}$$

Now suppose that $x_1, \ldots, x_\ell$ are binary random variables and $y_1, \ldots, y_\ell$ are random variables. Denote addition over $\mathbb{F}_2$ by $\oplus$. We would like to calculate $\ln L(x_1 \oplus \cdots \oplus x_\ell \mid y_1, \ldots, y_\ell)$. Note that if $p = 2\Pr[x_1 = 0 \mid y_1] - 1$ and $q = 2\Pr[x_2 = 0 \mid y_2] - 1$, then $2\Pr[x_1 \oplus x_2 = 0 \mid y_1, y_2] - 1 = pq$. (Why?) Therefore, $2\Pr[x_1 \oplus \cdots \oplus x_\ell = 0 \mid y_1, \ldots, y_\ell] - 1 = \prod_{i=1}^{\ell}(2\Pr[x_i = 0 \mid y_i] - 1)$. Since $\Pr[x_i = 0 \mid y_i] = L(x_i \mid y_i)/(1 + L(x_i \mid y_i))$, we have that $2\Pr[x_i = 0 \mid y_i] - 1 = (L-1)/(L+1) = \tanh(\ell/2)$, where $L = L(x_i \mid y_i)$ and $\ell = \ln L$. Therefore, we obtain

$$\ln L(x_1 \oplus \cdots \oplus x_\ell \mid y_1, \ldots, y_\ell) = \ln \frac{1 + \left(\prod_{i=1}^{\ell} \tanh(\ell_i/2)\right)}{1 - \left(\prod_{i=1}^{\ell} \tanh(\ell_i/2)\right)}, \tag{2}$$

where $\ell_i = \ln L(x_i \mid y_i)$. The belief propagation algorithm for LDPC codes can be derived from these two observations. In round 0, the check nodes send along all the outgoing edges their log-likelihoods conditioned on their observed value. For example, if the channel used is the BSC with error probability p, then the first message sent to all the check nodes adjacent to a message node is $\ln(1-p) - \ln p$ if the node's value is zero, and it is the negative of this value if the node's value is one. In all the subsequent rounds of the algorithm a check node $\mathtt{c}$ sends to an adjacent message node $\mathtt{v}$ a likelihood according to (2). A message node $\mathtt{v}$ sends to the check node $\mathtt{c}$ its log-likelihood conditioned on its observed value and on the incoming log-likelihoods from adjacent check nodes other than $\mathtt{c}$ using the relation (1).

Let $\mathtt{m}_{\mathtt{vc}}^{(\ell)}$ be the message passed from message node $\mathtt{v}$ to check node $\mathtt{c}$ at the ℓth round of the algorithm. Similarly, define $\mathtt{m}_{\mathtt{cv}}^{(\ell)}$. At round 0, $\mathtt{m}_{\mathtt{vc}}^{(0)}$ is the log-likelihood of the message node $\mathtt{v}$ conditioned on its observed value, which is independent of $\mathtt{c}$. We denote this value by $\mathtt{m}_{\mathtt{v}}$. Then the update equations for the messages under belief-propagation can be described as

$$\mathtt{m}_{\mathtt{vc}}^{(\ell)} = \begin{cases} \mathtt{m}_{\mathtt{v}}, & \text{if } \ell = 0, \\ \mathtt{m}_{\mathtt{v}} + \sum_{\mathtt{c}' \in C_{\mathtt{v}} \setminus \{\mathtt{c}\}} \mathtt{m}_{\mathtt{c}'\mathtt{v}}^{(\ell-1)}, & \text{if } \ell \geq 1, \end{cases} \tag{3}$$

$$\mathtt{m}_{\mathtt{cv}}^{(\ell)} = \ln \frac{1 + \prod_{\mathtt{v}' \in V_{\mathtt{c}} \setminus \{\mathtt{v}\}} \tanh\left(\mathtt{m}_{\mathtt{v}'\mathtt{c}}^{(\ell)}/2\right)}{1 - \prod_{\mathtt{v}' \in V_{\mathtt{c}} \setminus \{\mathtt{v}\}} \tanh\left(\mathtt{m}_{\mathtt{v}'\mathtt{c}}^{(\ell)}/2\right)}, \tag{4}$$

where $C_{\mathtt{v}}$ is the set of check nodes incident to message node $\mathtt{v}$, and $V_{\mathtt{c}}$ is the set of message nodes incident to check node $\mathtt{c}$.

The computations at the check nodes can be simplified further by performing them in the log-domain. Since the value of $\tanh(x)$ can be negative, we need to keep track of its sign separately. Let γ be a map from the real numbers $[-\infty, \infty]$ to $\mathbb{F}_2 \times [0, \infty]$ defined by $\gamma(x) := (\mathrm{sgn}(x), -\ln \tanh(|x|/2))$ (we set $\mathrm{sgn}(x) = 1$ if $x \geq 1$ and $\mathrm{sgn}(x) = 0$ otherwise.) It is clear that γ is bijective, so there exists an inverse function γ^{-1}. Moreover, $\gamma(xy) = \gamma(x) + \gamma(y)$, where addition is component-wise in $\mathbb{F}_2$ and in $[0, \infty]$. Then it is very easy to show that (4) is equivalent to

$$\mathtt{m}_{\mathtt{cv}}^{(\ell)} = \gamma^{-1}\left(\sum_{\mathtt{v}' \in V_{\mathtt{c}} \setminus \{\mathtt{v}\}} \gamma\left(\mathtt{m}_{\mathtt{v}'\mathtt{c}}^{(\ell-1)}\right)\right) \tag{5}$$

We will use this representation when discussing density evolution later.

In practice, belief propagation may be executed for a maximum number of rounds or until the passed likelihoods are close to certainty, whichever is first. A *certain* likelihood is a likelihood in which $\ln L(x \mid y)$ is either ∞ or $-\infty$. If it is ∞, then $\Pr[x = 0 \mid y] = 1$, and if it is $-\infty$, then $\Pr[x = 1 \mid y] = 1$.

One very important aspect of belief propagation is its running time. Since the algorithm traverses the edges in the graph, and the graph is sparse, the number of edges traversed is small. Moreover, if the algorithm runs for a constant number of times, then each edge is traversed a constant number of times, and the algorithm uses a number of operations that is linear in the number of message nodes!

Another important note about belief propagation is that the algorithm itself is entirely independent of the channel used, though the messages passed during the algorithm are completely dependent on the channel.

One question that might rise is about the relationship of belief propagation and maximum likelihood decoding. The answer is that belief propagation is in general less powerful than maximum likelihood decoding. In fact, it is easy to construct classes of LDPC codes for which maximum likelihood decoding can decode many more errors than belief propagation (one example is given by biregular bipartite graphs in which the common degree of the message nodes is very large but the reader is not required to see this right away).

6 Asymptotic Analysis of Belief Propagation and Density Evolution

The messages passed at each round of the belief propagation algorithm are random variables. If at every round in the algorithm the incoming messages are statistically independent, then the update equation correctly calculates the corresponding log-likelihood based on the observations. (This is what I meant by the independence assumption above.) This assumption is rather questionable, though, especially when the number of iterations is large. In fact, the independence assumption is correct for the ℓ first rounds of the algorithm only if the neighborhood of a message node up to depth ℓ is a tree.

Nevertheless, belief propagation can be analysed using a combination of tools from combinatorics and probability theory. The first analysis for a special type of belief propagation appeared in [6], and was applied to hard decision decoding of LDPC codes in [7]. The analysis was vastly generalized in [8] to belief propagation over a large class of channels.

The analysis starts by proving that if ℓ is fixed and n and r are large enough, then for random bipartite graphs the neighborhood of depth ℓ of most of the message nodes is a tree. Therefore, for ℓ rounds the belief propagation algorithm on these nodes correctly computes the likelihood of the node. Let us call these nodes the *good* nodes. We will worry about the other nodes later.

Next the expected behavior of belief propagation is calculated by analysing the algorithm on the tree, and a martingale is used to show that the actual behavior of the algorithm is sharply concentrated around its expectation. This step of the analysis is rather standard, at least in Theoretical Computer Science.

Altogether, the martingale arguments and the tree assumption (which holds for large graphs and a fixed iteration number ℓ) prove that a heuristic analysis of belief propagation on trees correctly mirrors the actual behavior on the full graph for a fixed number of iterations. The probability of error among the good message nodes in the graph can be calculated according to the behavior of belief propagation. For appropriate degree distributions this shows that the error probability of the good message nodes in the graph can be made arbitrarily small. What about the other (non-good) message nodes? Since their fraction is smaller than a constant, they will contribute only a sub-constant term to the error probability and their effect will disappear asymptotically, which means that they are not relevant for an asymptotic analysis. Details can be found in the above mentioned literature.

The analysis of the expected behavior of belief propagation on trees leads to a recursion for the density function of the messages passed along the edges. The general machinery shows that, asymptotically, the actual density of the messages passed is very close to the expected density. Tracking the expected density during the iterations thus gives a very good picture of the actual behavior of the algorithm. This method, called *density evolution* [8, 9, 7], is one of the crown jewels of the asymptotic theory of LDPC codes. In the following, I will briefly discuss this.

As a first remark note that if $X_1, \ldots, X_d$ are i.i.d. random variables over some (additive) group G, and if f is the common density of the X_i, then the density F of $X_1 + \cdots + X_d$ equals the d-fold convolutional power of f. (For any two integrable functions f and g defined over G the convolution of f and g, denoted $f \otimes g$, is defined as $(f \otimes g)(\tau) = \int_G f(\sigma)g(\tau - \sigma)\,\mathrm{d}G$, where $\mathrm{d}G$ is the Haar measure on G.) If G is the group of real numbers with respect to multiplication, then $f \otimes g$ is the well-known convolution of real functions.

Let now g_i denote the common density function of the messages $\mathtt{m}_{\mathtt{cv}}^{(i)}$ sent from check nodes to message nodes at round i of the algorithm, and let f denote the density of the messages $\mathtt{m}_v$, i.e., the likelihood of the messages sent at round 0 of the algorithm. Then the update rule for the densities in (3) implies that the common density f_{i+1} of the messages sent from message nodes to check nodes at round $i + 1$ conditioned on the event that the degree of the node is d equals $f \otimes g_i^{\otimes(d-1)}$.

Next we assume that the graph is random such that each edge is connected to a message node of degree d with probability λ_d, and each edge is connected to a check node of degree d with probability ρ_d. Then the expected density of the messages sent from message nodes to check nodes at round $i+1$ is $f \otimes \lambda(g_i)$, where $\lambda(g_i) = \sum_d \lambda_d g_i^{\otimes(d-1)}$. (All this of course assumes the independence assumption.)

To assess the evolution of the densities at the check nodes, we need to use the operator γ introduced above. For a random variable X on $[-\infty, \infty]$ with density F let $\Gamma(F)$ denote the density of the random variable $\gamma(X)$. $\gamma(X)$ is defined on the group $G := \mathbb{F}_2 \times [0, \infty]$. Therefore, the density of $\gamma(X) + \gamma(Y)$ is the convolution (over G) of $\Gamma(F)$ and $\Gamma(H)$, where H denotes the density of Y. Following (5) and assuming independence via the independence assumption, we see that the common density g_i of the messages passed from check to message nodes at round i is $\Gamma^{-1}(\rho(\Gamma(f_i)))$, where $\rho(h) = \sum_d \rho_d h^{\otimes(d-1)}$. All in all, we obtain the following recursion for the densities f_i:

$$f_{i+1} = f \otimes \lambda(\Gamma^{-1}(\rho(\Gamma(f_i)))). \tag{6}$$

This recursion is called density evolution. The reason for the naming should be obvious.

I have not made the recursion very explicit. In fact, the operator Γ has not been derived at all. For that I refer the reader to [8] and [9].

Density evolution can be used in conjunction with Fourier Transform techniques to obtain asymptotic thresholds below which belief propagation decodes the code successfully, and above which belief propagation does not decode successfully ([8, 9]).

Density evolution is exact only as long the incoming messages are independent random variables. For a finite graph this can be the case only for a small number of rounds.

7 Decoding on the BEC

Perhaps the most illustrative example of belief propagation is when it is applied to LDPC codes over the BEC with erasure probability p. In fact, almost all the important and interesting features of the belief propagation algorithm are already present on the BEC. A thorough analysis of this special case seems thus to be a prerequisite for the general case.

It is sufficient to assume that the all-zero codeword was sent. The log-likelihood of the messages at round 0, $\mathtt{m}_{\mathtt{v}}$, is $+\infty$ if the corresponding message bit is not erased, and it is 0 if the message bit is erased. Moreover, consulting the update equations for the messages, we see that if $\mathtt{v}$ is not erased, then the message passed from $\mathtt{v}$ to any of its incident check nodes is always $+\infty$.

The update equations also imply that $\mathtt{m}_{\mathtt{cv}}$ is $+\infty$ if and only if all the message nodes incident to $\mathtt{c}$ except $\mathtt{v}$ are not erased. In all other cases $\mathtt{m}_{\mathtt{cv}}$ is zero.

If $\mathtt{v}$ is an erased message node, then $\mathtt{m}_v = 0$. The message $\mathtt{m}_{\mathtt{vc}}$ is $+\infty$ if and only if there is some check node incident to $\mathtt{v}$ other than $\mathtt{c}$ which was sending a message $+\infty$ to $\mathtt{v}$ in the previous round.

Because of the binary feature of the messages, belief propagation on the erasure channel can be described much easier in the following:

1. [Initialization] Initialize the values of all the check nodes to zero.

2. [Direct recovery] For all message nodes $\mathtt{v}$, if the node is received, then add its value to the values of all adjacent check nodes and remove $\mathtt{v}$ together with all edges emanating from it from the graph.

3. [Substitution recovery] If there is a check node $\mathtt{c}$ of degree one, substitute its value into the

value of its unique neighbor among the message nodes, add that value into the values of all adjacent check nodes and remove the message nodes and all edges emanating from it from the graph.

This algorithm was first proposed in [10] though connections to belief propagation were not realized then. It is clear that the number of operations that this algorithm performs is proportional to the number of edges in the graph. Hence, for sparse graphs the algorithm runs in time linear in the block length of the code. However, there is *no* guarantee that the algorithm can decode all message nodes. Whether or not this is the case depends on the graph structure.

The decoding algorithm can be analysed along the same lines as the full belief propagation. First, we need to find the expected density of the messages passed at each round of the algorithm under the independence assumption. In this case, the messages are binary (either $+\infty$ or 0), hence we only need to keep track of one parameter, namely the probability p_i that the messages passed from message nodes to check nodes at round i of the algorithm is 0. Let q_i denote the probability that the message passed from check nodes to message nodes at round i of the algorithm is 0. Then, conditioned on the event that the message node is of degree d, we have $p_{i+1} = p \cdot q_i^{d-1}$. Indeed, a message from a message node $\mathtt{v}$ to a check node $\mathtt{c}$ is 0 iff $\mathtt{v}$ was erased and all the messages coming from the neighboring check nodes other than $\mathtt{c}$ are 0, which is q_i^{d-1} under the independence assumption. Conditioned on the event that the check node has degree d we have $q_i = 1 - (1 - p_i)^{d-1}$: the check node $\mathtt{c}$ sends a message $+\infty$ to the message node $\mathtt{v}$ iff all the neighboring message nodes except for $\mathtt{v}$ send a message $+\infty$ to $\mathtt{c}$ in the previous round. Under the independence assumption that probability is $(1 - p_i)^{d-1}$, which shows the identity.

These recursions are not in a usable form yet since they are conditioned on the degrees of the message and the check nodes. To obtain a closed form we use again the numbers λ_d and ρ_d defined above. Recall that λ_d is the probability that an edge is connected to a message node of degree d, and ρ_d denotes the probability that an edge is conncted to a check node of degree d. Defining the generating functions $\lambda(x) = \sum_d \lambda_d x^{d-1}$ and $\rho(x) = \sum_d \rho_d x^{d-1}$ we obtain the following recursion using the formula for the total probability:

$$p_{i+1} = p \cdot \lambda(1 - \rho(1 - p_i)).$$

Under the independence assumption, and assuming that the underlying graph is random with edge degree distributions given by $\lambda(x)$ and $\rho(x)$, decoding is successful if $p_{i+1} < (1 - \varepsilon)p_i$ for all i and some $0 < \varepsilon \leq 1$. This yields the condition

$$p \cdot \lambda(1 - \rho(1 - x)) < x \quad \text{for} \quad x \in (0, p) \tag{7}$$

for successful decoding which was first proved in [11] and later reproduced in [10]. It is a useful and interesting exercise for the reader to show that (7) is identical to (6) in the case of the BEC.

Condition (7) was proved in [11] in a completely different way than explained here. A system of differential equations was derived whose solutions tracked the expected fraction of nodes of various degrees as the decoding process evolved. One of the solutions corresponds to the fraction of check nodes of reduced degree one during the algorithm. By keeping this fraction above zero at all times, it is guaranteed that in expectation there are always check nodes of degree one left to continue the decoding process. To show that the actual values of the random variables are sharply concentrated around their computed expectations, a large deviation result was derived which is not unsimilar to Azuma's inequality for martingales.

Condition (7) can be used to calculate the maximal fraction of erasures a random LDPC code with given edge degree distributions can correct using the simple decoding algorithm. For example, consider a random biregular graph in which each message node has degree 3 and each check node has degree 6. (Such a graph is called a $(3, 6)$-biregular graph.) In this case $\lambda(x) = x^2$ and $\rho(x) = x^5$. What is the maximum fraction of erasures p? (In fact, this value is a supremum.) You can simulate the decoder on many such random graphs with a large number of message nodes. The simulations will show that on average around 42.9% erasures can be recovered. What is this value? According to (7) it is the supremum of all p such that $p(1-(1-x)^5)^2 < x$ on $(0, p)$. The minimum of the function $x/(1-(1-x)^5)^2$ on $(0, 1)$ is attained at the unique root of the polynomial $9x^4 - 35x^3 + 50x^2 - 30x + 5$ in the interval $(0, 1)$, and this is the supremum value for p. This value can be computed exactly, using formulas for the solution of the quartic [12].

As a side remark, I would like to mention an interesting result. First, it is not hard to see that the ratio r/n between the message and the check nodes equals $\int_0^1 \rho(x)\,\mathrm{d}x / \int_0^1 \lambda(x)\,\mathrm{d}x$. The rate of the code is at least $1 - r/n$, and since the capacity of the erasure channel with erasure probability p is $1 - p$, (7) should imply that $p \leq \int_0^1 \rho(x)\,\mathrm{d}x / \int_0^1 \lambda(x)\,\mathrm{d}x$ in a purely mechanical way (without using the interpretations above). Can you see how to derive this? (See also [13].)

8 Hard Decision Decoding on the BSC

The belief propagation algorithm is the best algorithm among message passing decoders, and the accompanying density evolution provides a tool for analysing the algorithm. However, for practical applications on channels other than the BEC the belief propagation algorithm is rather complicated, and often leads to a decrease in the speed of the decoder. Therefore, often times a discretized version of the belief propagation algorithm is used. The lowest level of discretization is achieved when the messages passed are binary. In this case one often speaks of a hard decision decoder, as opposed to a soft decision decoder which uses a larger range of values. In this section I will describe two hard decision decoding algorithms on the BSC, both due to Gallager [4].

In both cases the messages passed between the message nodes and the check nodes consist of 0 and 1. Let me first describe the Gallager A algorithm: in round 0, the message nodes send their received values to all their neighboring check nodes. From that point on at each round a check node $\mathtt{c}$ sends to the neighboring message node $\mathtt{v}$ the addition (mod 2) of all the incoming messages from incident message nodes other than $\mathtt{v}$. A message node $\mathtt{v}$ sends the following message to the check node $\mathtt{c}$: if all the incoming messages from check nodes other than $\mathtt{c}$ are the same value b, then $\mathtt{v}$ sends the value b to $\mathtt{c}$; otherwise it sends its received value to $\mathtt{c}$.

An exact analysis of this algorithm was first given in [7]. The analysis is similar to the case of the BEC. We first find the expected density of the messages passed at each round. Again, we can assume that the all-zero word was transmitted over the BSC with error probability p. Since the messages are 0 and 1, we only need to track p_i, the probability that the message sent from a message node to a check node at round i is 1. Let q_i denote the probability that the message sent from a check node to a message node at round i is 1. Conditioned on the event that the message node is of degree d, and under the independence assumption, we obtain $p_{i+1} = (1-p)q_i^{d-1} + p \cdot (1-(1-q_i)^{d-1})$. To see this, observe that the message 1 is passed from message node $\mathtt{v}$ to check node $\mathtt{c}$ iff one of these two cases occurs: (a) the message node was received in error (with probability p) and at least one of the incoming messages is a 1 (with probability $1-(1-q_i)^{d-1}$), or (b) the message was received correctly (probability $1-p$) and all incoming messages are 1 (probability q_i^{d-1}). To assess the evolution of q_i in terms of p_i, note that a check node $\mathtt{c}$ sends a message 1 to message node $\mathtt{v}$ at round i iff the addition mod 2 of the incoming messages from message nodes other than $\mathtt{v}$ in the previous round is 0. Each such message is 1 with probability p_i, and the messages are independent. Conditioned on the event that the check node is of degree ℓ, there are $\ell - 1$ such messages. The probability that their addition mod 2 is 1 is $q_i = (1 - (1 - 2p_i)^{\ell-1})/2$ (why?).

These recursions are for the conditional probabilities, conditioned on the degrees of the nodes. Introducing the generating functions $\lambda(x)$ and $\rho(x)$ as above, we obtain the following recursion for the probabilities themselves:

$$p_{i+1} = (1-p) \cdot \lambda\left(\frac{1 - \rho(1-2p_i)}{2}\right) + p \cdot \left(1 - \lambda\left(\frac{1 + \rho(1-2p_i)}{2}\right)\right). \tag{8}$$

If $\lambda(x)$, $\rho(x)$, and p are such that p_i is monotonically decreasing, then decoding will be successful asymptotically with high probability, as long as the independence assumption is valid.

For example, consider a $(3, 6)$-biregular graph. In this case $\lambda(x) = x^2$ and $\rho(x) = x^5$, and the condition becomes

$$(1-p) \cdot \left(\frac{1 - (1-2x)^5}{2}\right)^2 + p \cdot \left(1 - \left(\frac{1 + (1-2x)^5}{2}\right)^2\right) < x$$

for $x \in (0, p)$. A numerical calculation shows that the best value for p is around 0.039.

By Shannon's theorem the maximum error probability that a code of rate 1/2 can correct is the maximum p such that $1 + p\log_2(p) + (1-p)\log_2(1-p) = 0.5$. A numerical approximation shows that p is around 11%, which means that the Gallager A algorithm on the biregular $(3, 6)$-graph is very far from achieving capacity. Bazzi et al. [12] show that for rate 1/2 the best graph for the Gallager A algorithm is the biregular $(4, 8)$-graph for which the maximum tolerable error probability is roughly 0.0475—still very far from capacity. This shows that this algorithm, though simple, is very far from using all the information that can be used.

Gallager's algorithm B is slightly more powerful than algorithm A. In this algorithm, for each degree j and each round i there is a threshold value $b_{i,j}$ (to be determined) such that at round i for each message node $\mathtt{v}$ and each adjacent check node $\mathtt{c}$, if at least $b_{i,j}$ neighbors of $\mathtt{v}$ excluding $\mathtt{c}$ sent the same information in the previous round, then $\mathtt{v}$ sends that information to $\mathtt{c}$; otherwise $\mathtt{v}$ sends its received value to $\mathtt{c}$. The rest of the algorithm is the same as in algorithm A.

It is clear that algorithm A is a special case of algorithm B, in which $b_{i,j} = j - 1$ independent of the round.

This algorithm can be analysed in the same manner as algorithm A, and a recursion can be obtained for the probability p_i that a message node is sending the incorrect information to a check node at round i:

$$\begin{aligned} p_{i+1} \;=\; p - \sum_{j \geq 1} \lambda_j \Bigg[p \sum_{t=b_{i,j}}^{j} \binom{j-1}{l} \left[\frac{1+\rho(1-2p_i)}{2}\right]^{t} \left[\frac{1-\rho(1-2p_i)}{2}\right]^{j-1-t} + \\ (1-p) \sum_{t=b_{i,j}}^{j-1} \binom{j-1}{t} \left[\frac{1-\rho(1-2p_i)}{2}\right]^{t} \left[\frac{1+\rho(1-2p_i)}{2}\right]^{j-1-t} \Bigg], \end{aligned} \tag{9}$$

where the value of $b_{i,j}$ is the smallest integer that satisfies

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

(See [7] for details of the analysis.)

For another hard decision decoder on the BSC (called “erasure decoder”), see [8].

The above one parameter recursions can be used to design codes that asymptotically perform very well for a given amount of noise. The method of choice in these cases is linear programming. For details I refer the reader to [10, 7].

9 Completing the Analysis: Expander Based Arguments

Density evolution and its instantiations are valid only as long as the incoming messages are independent. The messages are independent for ℓ rounds only if the neighborhoods of depth ℓ around the message nodes are trees. But this immediately puts an upper bound on ℓ (of the order $\log(n)$, where n is the number of message nodes, see Section 11). But this number of rounds is usually not sufficient to prove that the decoding process corrects all errors. A different analysis is needed to complete the decoding.

One property of the graphs that guarantees successful decoding is *expansion*. A bipartite graph with n message nodes is called an (α, β)-expander if for any subset S of the message nodes of size at most αn the number of neighbors of S is at least $\beta \cdot a_S \cdot |S|$, where a_S is the average degree of the nodes in S. In other words, if there are many edges going out of a subset of message nodes, then there should be many neighbors.

Expansion arguments have been used by many researchers in the study of decoding codes obtained from graphs [14, 15, 16, 17]. Later, [10, 7] used expander based arguments to show that the erasure correction algorithm on the BEC and Gallager’s hard decision decoding algorithm will decode all the erasures/errors if the fraction of errors is small and the graph has sufficient expansion. Burshtein and Miller [18] generalized these results to general message passing algorithms.

To give the reader an idea of how these methods are used, I will exemplify them in the case of the BEC. Choose a graph with edge degree distributions given by $\lambda(x)$ and $\rho(x)$ at random. The analysis of the belief propagation decoder for the BEC implies that if condition (7) is true, then for any $\varepsilon > 0$ there is an n_0 such that for all $n \geq n_0$ the erasure decoder reduces the number of erased message nodes below εn. The algorithm may well decode all the erasures, but the point is that the analysis does not guarantee that.

To complete the analysis of the decoder, we first note the following fact: if the random graph is an $(\varepsilon, 1/2)$-expander, then the erasure decoding algorithm recovers any set of εn or fewer erasures. Suppose that this were not the case and consider a minimal counterexample consisting of a nonempty set S of erasures. Consider the subgraph induced by S, and denote by $\Gamma(S)$ the set of neighbors of S. No node in $\Gamma(S)$ has degree 1, since this neighbor would recover one element in S and would contradict the minimality of S. Hence, the total number of edges emanating from these nodes is at least $2|\Gamma(S)|$. On the other hand, the total number of edges emanating from S is $a_S \cdot |S|$, so $a_S \cdot |S| \geq 2|\Gamma(S)|$ which implies $|\Gamma(S)| \leq a_S \cdot |S|/2$ and contradicts the expansion property of the graph.

In [10] it is shown that for a random bipartite graph without message nodes of degree one or two there is a constant ε depending on the rate of the induced code and on the degrees of the message nodes such that the graph is an $(\varepsilon, 1/2)$-expander with high probability. On random graphs without message nodes of degrees one or two we see that the erasure decoding algorithm succeeds with high probability provided condition (7) is satisfied.

10 Achieving Capacity

Recall Shannon's theorem from Section 2 which states the existence of codes that come arbitrarily close to the capacity of the channel when decoded with maximum likelihood decoding. LDPC codes were designed to have decoding algorithms of low complexity, such as belief propagation and its variants. But how close can we get to capacity using these algorithms?

There is no satisfactory answer to this question for arbitrary channels. What I mean by a satisfactory answer is an answer to the question whether subclasses of LDPC codes, for example

LDPC codes with an appropriate degree distribution, will provably come *arbitrarily* close to the capacity of the channel. Optimization results for various channels, such as the Additive White Gaussian Noise (AWGN) channel and the BSC have produced specific degree distributions such that the corresponding codes come very close to capacity, see [9, 19].

We call an LDPC code ε-close for a channel $\mathcal{C}$ with respect to some message passing algorithm if the rate of the code is at least $\mathrm{Cap}(\mathcal{C}) - \varepsilon$ and if the message passing algorithm can correct errors over that channel with high probability. We call a sequence of degree distributions $(\lambda^{(n)}(x), \rho^{(n)}(x))$ *capacity-achieving* over that channel with respect to the given algorithm if for any ε there is some n_0 such that for all $n \geq n_0$ the LDPC code corresponding to the degree distribution $(\lambda^{(n)}(x), \rho^{(n)}(x))$ is ε-close to capacity. Using this notation, the following question is open:

Is there a nontrivial channel other than the BEC and a message passing algorithm for which there exists a capacity-achieving sequence $(\lambda^{(n)}(x), \rho^{(n)}(x))$?

I believe that this question is one of the fundamental open questions in the asymptotic theory of LDPC codes.

In [10] the authors describe capacity-achieving sequences for the BEC for any erasure probability p. Let $\varepsilon > 0$ be given, let $D := \lceil 1/\varepsilon \rceil$, and set

$$\lambda(x) = \frac{1}{H(D)} \sum_{i=1}^{D} \frac{x^i}{i}, \quad \rho(x) = \mathrm{e}^{\alpha(x-1)},$$

where $\alpha = H(D)/p$. (Technically, $\rho(x)$ cannot define a degree distribution since it is a power series and not a polynomial. But the series can be truncated to obtain a function that is arbitrarily close to the exponential.) We now apply (7):

$$\begin{aligned} p\lambda(1-\rho(1-x)) &< -\frac{p}{H(D)} \ln(\rho(1-x)) \\ &= \frac{\alpha p}{H(D)} x \\ &= x. \end{aligned}$$

This shows that a corresponding code can decode a p-fraction of erasures with high probability. [2]

[2] Actually, as was discussed before, (7) only shows that the fraction of erasures can be reduced to any constant fraction of the number of message nodes. To show that the decoding is successful all the way to the end, we need a different type of argument. Expansion arguments do not work for the corresponding graphs, since there are many message nodes of degree 2. For a way to resolve these issues, see [10].

What about the rate of these codes? Above, we mentioned that the rate of the code given by the degree distributions $\lambda(x)$ and $\rho(x)$ is at least $1 - \int_0^1 \rho(x)\,\mathrm{d}x / \int_0^1 \lambda(x)\,\mathrm{d}x$. In our case, this lower bound equals $1 - p(1 + 1/D)(1 - \mathrm{e}^{-\alpha})$ which is larger than $1 - p(1 + \varepsilon)$.

The degree distribution above is called the *Tornado* degree distribution and the corresponding codes are called *Tornado codes.* These codes have many applications in computer networking which I will not mention here (see, e.g., [20]).

Tornado codes were the first class of codes that could provably achieve the capacity of the BEC using belief propagation. Since then many other distributions have been discovered [13, 21]. The latter paper also discusses general methodologies for constructing such degree distributions, and also discusses optimal convergence speeds to capacity.

11 Graphs of Large Girth

As is clear from the previous discussions, if the smallest cycle in the bipartite graph underlying the LDPC code is of length 2ℓ, then independence assumption is valid for ℓ rounds of belief propagation. In particular, density evolution describes the expected behavior of the density functions of these messages exactly for this number of rounds.

The *girth* of a graph is defined as the length of the smallest cycle in the graph. For bipartite graphs the girth is necessarily even, so the smallest possible girth is 4. It is easy to obtain an upper bound for the girth of a biregular bipartite graph with n message nodes of degree d and r check nodes of degree k: if the girth is 2ℓ, then the neighborhood of depth $\ell - 1$ of any message node is a tree with a root of degree d, and in which all nodes of odd depth have degree $k - 1$, while all nodes of even depth have degree $d - 1$ (we assume that the root of the tree has depth 0). The number of nodes at even depths in the tree should be at most equal to the message nodes, while the number of nodes at odd depths in the tree should be at least equal to the check nodes. The number of nodes at even depths in the tree equals 1 for depth 0, $d(k-1)$ for depth 2, $d(k-1)D$ for depth 4, $d(k-1)D^2$ for depth 6, etc., where $D = (d-1)(k-1)$. The total number of nodes at even depths is equal to

$$1 + d(k-1)\frac{D^{\lfloor \frac{\ell}{2} \rfloor} - 1}{D - 1}.$$

This number has to be less than or equal to n, the number of message nodes. This yields an upper bound on 2ℓ, the girth of the graph. The bound has order $\log_D(n)$. Similar bounds can be obtained by considering nodes of odd depths in the tree.

A similar bound as above can also be deduced for irregular graphs [22], but I will not discuss it here.

As I said before, graphs of large girth are interesting because of the accuracy of belief propagation. However, this is not interesting for practical purposes, since for obtaining accuracy for many rounds the girth of the graph has to be large which means that the number of nodes in the graph has to be very large.

There are other reasons to study graphs of large girth, however. From the point of view of combinatorics graphs of large girth which satisfy (or come close to) the upper bound on the girth are extremal objects. Therefore, to construct them, methods from extremal graph theory need to be applied. From the point of view of coding theory eliminating small cycles is very similar to eliminating words of small weight in the code. This is because a word of weight d leads to a cycle of length $2d$ or less. (Why?)

How does one construct bipartite graphs of large girth? There are a number of known techniques with origins in algebra and combinatorics. For example, it is very easy to construct optimal bipartite graphs of girth 6. Below I will give such a construction. Let C be a Reed-Solomon code of dimension 2 and length n over the field $\mathbb{F}_q$. By definition, this code has q^2 codewords and the Hamming distance between any two distinct codewords is at least $n-1$. From C we construct a bipartite graph with q^2 message nodes and nq check nodes in the following way: The message nodes correspond to the codewords in C. The check nodes are divided in groups of q nodes each; the nodes in each such group corresponds to the elements of $\mathbb{F}_q$. The connections in the graph are obtained as follows: A message node corresponding to the codewords $(x_1, \ldots, x_n)$ is connected to the check nodes corresponding to x_1 in the first group, to x_2 in the second group, $\ldots$, to x_n in the last group. Hence, all message nodes have degree n, and all check nodes have degree q, and the graph has in total nq^2 edges. Suppose that this graph has a cycle of length 4. This means that there are two codewords (corresponding to the two message nodes in the cycle) which coincide at two positions (corresponding to the two check nodes in the cycle). This is impossible by the choice of the code C, which shows that the girth of the graph is at least 6. To show the optimality of these graphs, we compare the number of check nodes to the above bound. Let 2ℓ denote the girth of the graph. If $\ell = 3$, then $1 + n(q-1) \leq q^2$, which shows that $n \leq q+1$. By choosing $n = q+1$ we obtain optimal graphs of girth 6.

If $n < q+1$, the graphs obtained may not be optimal, and their girth may be larger than 6. For $n \neq 2$ it is easy to see that the girth of the graph is indeed 6. For $n = 2$ the girth is 8. (A cycle of length 6 in the graph corresponds to three codewords such that every two coincide in exactly one position. This is possible for $n > 2$, and impossible for $n = 2$.)

There are many constructions of graphs without small cycles using finite geometries, but these constructions are for the most part not optimal (except for cases where the girth is small, e.g., 4, or cases where the message nodes are of degree 2).

The sub-discipline of combinatorics dealing with such questions is called *extremal combinatorics.* One of the questions studied here is that of existence of graphs that do not contain a subgraph of a special type (e.g., a cycle). I will not go deeper into these problems here and refer the reader to appropriate literature (e.g., [23]).

A discussion of graphs of large girth is not complete without at least mentioning Ramanujan graphs which have very large girth in an asymptotic sense. I will not discuss these graphs at all in this note and refer the reader to [24, 25]. For interesting applications of these graphs in coding theory I refer the reader to [26].

12 Encoding Algorithms

An encoding algorithm for a binary linear code of dimension k and block length n is an algorithm that computes a codeword from k original bits $x_1, \ldots, x_k$. To compare algorithms against each other, it is important to introduce the concept of cost, or operations. For the purposes of this note the cost of an algorithm is the number of arithmetic operations over $\mathbb{F}_2$ that the algorithm uses.

If a basis $g_1, \ldots, g_k$ for the linear code is known, then encoding can be done by computing $x_1 g_1 + \cdots + x_k g_k$. If the straightforward algorithm is used to perform the computation (and it is a-priori not clear what other types of algorithms one may use), then the number of operations sufficient for performing the computation depends on the Hamming weights of the basis vectors. If the vectors are dense, then the cost of the encoding is proportional to nk. For codes of constant rate, this is proportional to n^2, which may be too slow for some applications.

Unfortunately LDPC codes are given as the null space of a sparse matrix, rather than as the space generated by the rows of that matrix. For a given LDPC code it is highly unlikely that there exists a basis consisting of sparse vectors, so that the straightforward encoding algorithm uses a number of operations that is proportional to n^2. However, we would like to design algorithms for which the encoding cost is proportional to n.

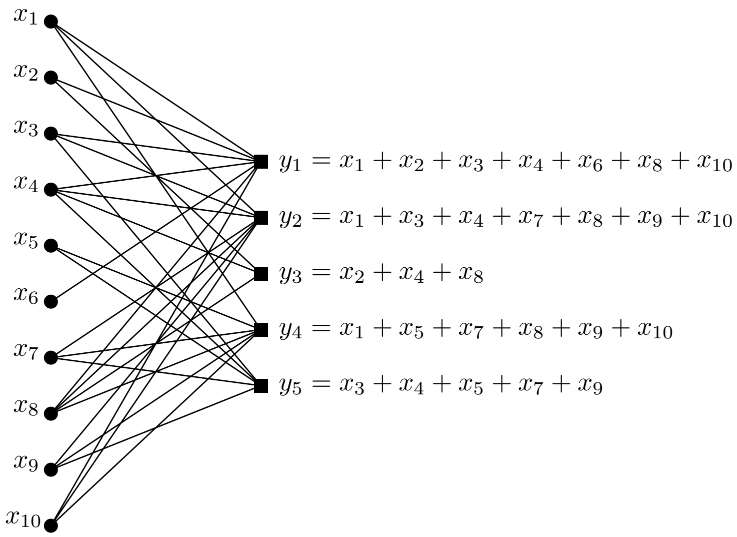
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
$y_1 = x_1 + x_2 + x_3 + x_4 + x_6 + x_8 + x_{10}$
$y_2 = x_1 + x_3 + x_4 + x_7 + x_8 + x_9 + x_{10}$
$y_3 = x_2 + x_4 + x_8$
$y_4 = x_1 + x_5 + x_7 + x_8 + x_9 + x_{10}$
$y_5 = x_3 + x_4 + x_5 + x_7 + x_9$

Figure 3: Construction with fast encoder

At this point there are at least two possible ways to go. One is to consider modifications of LDPC codes which are automatically equipped with fast encoding algorithms. The other is to try to find faster encoding algorithms for LDPC codes. I will discuss both these approaches here, and outline some of the pro's and con's for each approach.

One simple way to obtain codes from sparse graphs with fast encoding is to modify the construction of LDPC codes in such a way that the check nodes have values, and the value of each check node is the addition of the values of its adjacent message nodes. (In such a case, it would be more appropriate to talk about redundant nodes, rather than check nodes, and of information nodes rather than message nodes. But to avoid confusion, I will continue calling the right nodes check nodes and the left nodes message nodes.) Figure 3 gives an example. The number of additions needed in this construction is upper bounded by the number of edges. So, efficient encoding is possible if the graph is sparse. The codwords in this code consist of the values of the message nodes, appended by the values of the check nodes.

This construction leads to a linear time encoder, but it has a major problem with decoding. I will exemplify the problem for the case of the BEC. First, it is not clear that the belief propagation decoder on the BEC can decode all the erasures. This is because the check nodes can also be erased (in contrast to the case of LDPC codes where check nodes do not have a value per-se, but only keep track of the dependencies among the values of the message nodes). This problem is not an artifact of the non-optimal belief propagation decoder. Even the error probability of the maximum likelihood decoder is lower bounded by a constant in this case. Let me elaborate. Suppose that a codeword is transmitted over a BEC with erasure probability p. Then an expected p-fraction of the message nodes and an expected p-fraction of the check nodes will be erased. Let Λ_d be the fraction of message nodes of degree d. Because the graph is random, a message node of degree d will have all its neighbors in the set of erased check nodes with probability p^d. This probability is conditioned on the event that the degree of the message node is d. So, the probability that a message node has all its neighbors within the set of erased check nodes is $\sum_d \Lambda_d p^d$, which is a constant independent of the length of the code. Therefore, no algorithm can recover the value of that message node.

In [17] and [7] the following idea is used the to overcome this difficulty: the redundant nodes will be protected themselves with another graph layer to obtain a second set of redundant nodes; the second set will be protected by a third set, etc. This way a cascade of graphs is obtained rather than a single graph. At each stage the number of message and check nodes of the graphs decreases by a constant fraction. After a logarithmic number of layers the number of check nodes is small enough so the check nodes can be protected using a sophisticated binary code for which we are allowed to use a high-complexity decoder. Details can be found in [10]. If any single graph in the cascade is such that belief propagation can decode a p-fraction of errors, then the entire code will have the same property, with high probability (provided the final code in the cascade has that property, but this can be adjusted). All in all, this construction provides linear time encodable and decodable codes.

The idea of using a cascade, though appealing in theory, is rather cumbersome in practice. For example, in the case of the BEC, the variance of the fraction of erasures per graph-layer will often be too large to allow for decoding. Moreover, maintaining all the graphs is rather complicated and may lead to deficiencies in the decoder. (For some ideas on how to decrease these deficiencies, see [10].)

Another class of codes obtained from sparse graphs and equipped with fast encoders are the *Repeat-Accumulate* (RA) codes of Divsalar et al. [27]. The construction of these codes is somewhat similar to the construction discussed above. However, instead of protecting the check nodes with another layer of a sparse graph, the protection is done via a dense graph, and the check nodes of the first graph are never transmitted. Dense graphs are in general not amenable to fast encoding. However, the dense graph chosen in an RA code is of a special structure which makes its computation easy.

More formally, the encoding process for RA codes is as follows. Consider an LDPC code whose

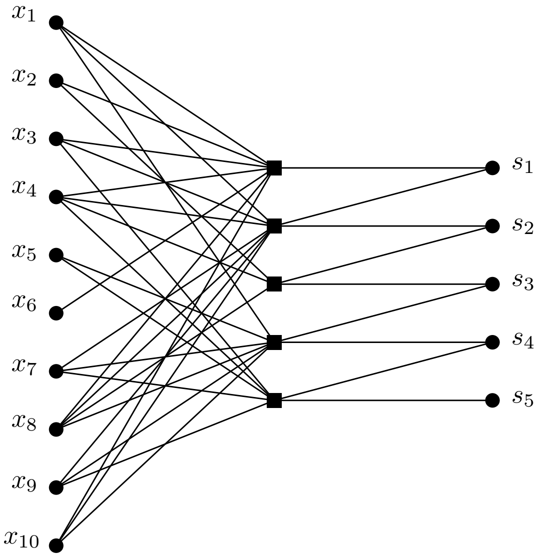

Figure 4: An irregular RA code. The left nodes are the information symbols, and the rightmost nodes are the redundant nodes. The squares in between are check nodes. Their values are computed as the addition of the values of their neighbors among the information nodes. The values of the redundant nodes are calculated so as to satisfy the relation that the values of the check nodes is equal to the addition of the values of the neighboring redundant nodes.

graph has n message nodes and r check nodes. The value of the r check nodes is computed using the procedure introduced above, i.e., the value of each check node is the addition of the values of its adjacent message nodes. Let $(y_1, \ldots, y_r)$ denote the values of these check nodes. The redundant values $(s_1, \ldots, s_r)$ are now calculated as follows: $s_1 = y_1, s_2 = s_1 + y_2, \ldots, s_r = s_{r-1} + y_r$. (This explains the phrase "accumulate.") An example of an RA code is given in Figure 4.

The original RA codes used a $(1, k)$-biregular graph for some k as the graph defining the LDPC code. (This explains the phrase "repeat.") RA codes were generalized to encompass *irregular* RA codes for which the underlying graph can be any bipartite graph [28]. The same paper introduces degree distributions for which the corresponding RA codes achieve capacity of the BEC.

We conclude this section by mentioning the work of Richardson and Urbanke [29] which provides an algorithm for encoding LDPC codes. They show that if the degree distribution $(\lambda(x), \rho(x))$ is such that $\rho(1-\lambda(x)) < x$ for $x \in (0, 1)$, and such that $\lambda_2\rho'(1) > 1$, then the LDPC code can be encoded in linear time. The condition $\lambda_2\rho'(1) > 1$ has the following interpretation: consider the graph generated by the message nodes of degree 2. This graph induces a graph on the check nodes, by interpreting the message nodes of degree 2 as edges in that graph (see Section 14). Then $\lambda_2\rho'(1) > 1$ implies that this induced graph has a connected component whose number of vertices is a constant fraction of the number of check nodes. This will be explained further in Section 14, where this condition is actually used to devise a linear time encoding algorithm for a certain type of graphs. I will not discuss the result of Richardson and Urbanke further, and will refer the reader to [29].

13 Finite-Length Analysis

Density evolution gives a somewhat satisfactory answer to the asymptotic performance of random LDPC codes with a given degree distribution. It is possible to refine the analysis of density evolution to obtain upper bounds on the error probability of the decoder in terms of the degree distributions, and in terms of the number of message and check nodes. However, these bounds are very poor even when the number of message nodes is several tens of thousands large. This is primarily due to two reasons: density evolution is only valid as long as the neighborhood around message nodes is a tree. For small graphs this corresponds to a very small number of iterations, which is usually too small to reduce the fraction of errors in the graph to a reasonable amount. The second source of inaccuracy for the error probability is the set of tools used, since the bounds obtained from the probabilistic

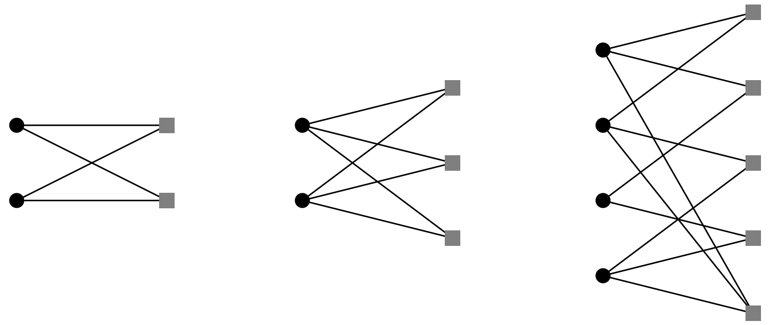

Figure 5: Examples of graphs that are stopping sets

analysis are too weak for small lengths.

For these reasons it is important to develop other methods for analysing the performance of message passing algorithms on small graphs. So far this has only started for the case of the BEC [30]. In this case the analysis is of a combinatorial flavor. Given a bipartite graph, its associated code, and a set of erasures among the check nodes, consider the graph induced by the erased message nodes. A *stopping set* in this graph is a set of message nodes such that the graph induced by these message nodes has the property that no check node has degree one. The number of message nodes in the stopping set is called its *size*. It should be clear that belief propagation for the BEC stops prematurely (i.e., without recovering all the message nodes) if and only if this subgraph has a stopping set. Figure 5 gives some examples of graphs that are themselves stopping sets. Since unions of stopping sets are stopping sets, any finite graph contains a unique maximal stopping set (which may be the empty set). For a random bipartite graph the probability that belief propagation on the BEC has not recovered ℓ message nodes at the point of failure (ℓ can be zero) is the probability that the graph induced by the erased message nodes has a maximal stopping set of size ℓ.

Besides [30] several papers discuss finite-length analysis of LDPC codes on the BEC [31, 32, 33].

I am not aware of similar analysis tools for channels other than the BEC. Generalizing the concept of stopping sets to other channels would certainly be a worthwhile effort.

14 An Example

In this section I will exemplify most of the above concepts for a special type of LDPC codes. The codes I will describe in this section certainly do not stand out because of their performance. However, it is rather easy to derive the main concepts for them and this warrants their discussion in the framework of this note. Moreover, it seems that a thorough understanding of their behavior is very important for understanding belief propagation for general LDPC codes. I will try to clarify this more at the end of the section.

For given n and r Let $\mathbf{P}(n,r)$ denote the ensemble of bipartite graphs with n message nodes and r check nodes for which each message node has degree 2 and its two neighbors among the check nodes are chosen independently at random. The check node degrees in such a graph are binomially distributed, and if n and r are large, then the distribution is very close to a Poisson distribution with mean $2n/r$. (This is a well-known fact, but the reader may try to prove it for herself.) It turns out that the edge degree distribution of the graph is very close to $\mathrm{e}^{\alpha(x-1)}$ where $\alpha = 2n/r$ is the average degree of the check nodes.

First, let us see how many erasures this code can correct. The maximum fraction of erasures is $1 - R$, where R is the rate, which is at least $1 - r/n$. We should therefore not expect to be able to correct more than an r/n-fraction of erasures, i.e., more than a $2/\alpha$-fraction. We now apply Condition (7): p is the maximum fraction of correctable erasures iff $p \cdot (1 - \mathrm{e}^{-\alpha x}) < x$ for $x \in (0,p)$. Replacing x by px, this condition becomes

$$1 - \mathrm{e}^{-\beta x} < x, \quad \beta = p\alpha. \tag{10}$$

This latter condition has an interesting interpretation: the graph induced by the p-fraction of erasures is a random graph in the ensemble $\mathbf{P}(e,r)$, where e is the number of erasures, the expected value of which is en. For this graph the edge distribution from the point of view of the check nodes is $\mathrm{e}^{-p\alpha(x-1)}$, and thus (7) implies (10).

Next, I will show that the maximum value of β for which Condition (10) holds is $\beta = 1$. For the function $1 - \mathrm{e}^{-\beta x} - x$ to be less than 0 in $(0,1)$, it is necessary that the derivative of this function be non-positive at 0. The derivative is $\beta\mathrm{e}^{-\beta x} - 1$, and its value at 0 is $\beta - 1$. Hence, $\beta \leq 1$ is a necessary condition for (10) to hold. On the other hand, if $\beta = 1$, then (10) is satisfied. Therefore, the maximum fraction of correctable erasures for a code in the ensemble $\mathbf{P}(n,r)$ is $r/(2n)$, i.e., the performance of these codes is at half the capacity. So, the ensemble $\mathbf{P}(n,r)$ is not a very good ensemble in terms of the performance of belief propagation on the BEC.

Before I go further in the discussion of codes in the ensemble $\mathbf{P}(n,r)$, let me give a different view of these codes. A bipartite graph with n message nodes and r check nodes in which each message

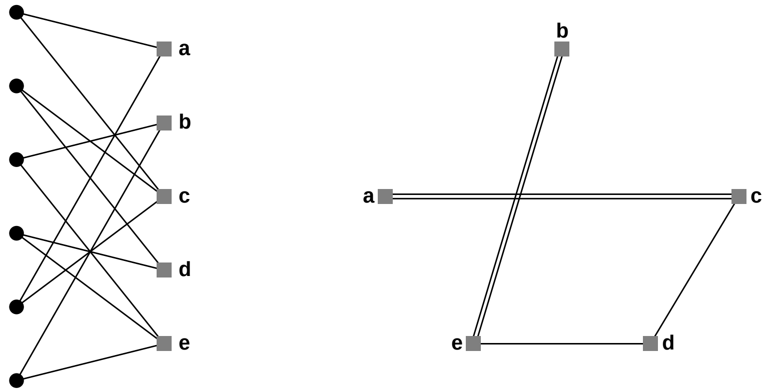

Figure 6: A graph with left degree 2 and its induced graph on the check nodes

node has degree 2 defines a (multi-)graph on the set of check nodes by regarding each message node as an edge in the graph in the obvious way. Multi-graphs and bipartite graphs with message degree 2 are in one-to-one correspondence to each other. In the following we will call the graph formed on the check nodes of a bipartite graph G with message degree 2 *induced by G*. Figure 6 gives an example.

For a graph in the ensemble $\mathbf{P}(n, r)$ the corresponding induced graph is a random graph of type $G_{r,n}$, where $G_{m,E}$ denotes the random graph on m vertices in which E edges are chosen randomly and with replacement among all the possible $\binom{m}{2}$ edges in the graph.

For a bipartite graph G with message degree 2 the stopping sets are precisely the edges of a 2-core. Let me define this notion: For any graph and any integer k the k-core of the graph is the unique maximal subgraph of G in which each node has degree k. The k-core may of course be empty.

It is a well-known fact [34] that a giant 2-core exists with high probability in a random graph with E edges in m nodes iff the average degree of a node is larger than 1, i.e., iff $E \geq m$. (A *giant* 2-core in the graph is a 2-core whose size is linear in the number of vertices of the graph.) Condition (10) is a new proof for this fact, as it shows that if the average degree of the induced graph is smaller than 1, then with high probability the graph does not contain a 2-core of linear size. It is also a well-known fact that this is precisely the condition for the random graph to contain a *giant component*, i.e., a component with linearly many nodes. Therefore, condition (10) can also be viewed as a condition on the graph not having a large component. (This condition is even more precise, as it gives the expected fraction of unrecovered message nodes at the time of failure of the decoder: it is p times the unique root of the equation $1 - x - \mathrm{e}^{-\beta x}$ in the interval $(0, 1)$; incidentally, this is exactly the expected size of the giant component in the graph, as is well-known in random graph theory [34].)

More generally, one can study graphs from the ensemble $\mathcal{L}(n, r, \rho(x))$ denoting random graphs with n message and r check nodes with edge degree distribution on the check side given by $\rho(x) = \sum_d \rho_d x^{d-1}$ (i.e., probability that an edge is connected to check node of degree d is ρ_d). The maximum fraction of tolerable erasures in this case is the supremum of all p such that $1 - \rho(1 - px) - x < 0$ for $x \in (0, 1)$. This yields the stability condition $p\rho'(1) < 1$. This condition is also sufficient, since it implies that $p\rho'(1 - px) < 1$ on $(0, 1)$, hence $1 - \rho(1 - px) - x$ is monotonically decreasing, and since this function is 0 at $x = 0$, it is negative for $x \in (0, 1)$.

The condition $p\rho'(1) < 1$ is equivalent to the statement that the graph induced on the check nodes by the bipartite graph has a giant component. This follows from results in [35]. According to that paper, if a graph is chosen randomly on n nodes subject to the condition that for each d the fraction of nodes of degree d is essentially R_d (see the paper for a precise definition), then the graph has almost surely a giant component iff $\sum_d d(d-2)R_d > 0$. Consider the graph obtained from the restriction of the message nodes to a p-fraction, and consider the graph induced by this smaller graph on the check nodes. Then, it is not hard to see that the degree distribution for this graph is $R(px + 1 - p)$, where $R(x) = c \int \rho(x)\, \mathrm{d}$ and c is the average degree of the check nodes in the smaller bipartite graph. Therefore, the condition in [35] for the induced graph to have a giant component equals $pR''(1) < c$, where $R''(x)$ is the second derivative of $R(x)$. This is precisely equal to $p\rho'(1) < 1$, i.e., the stability condition is equivalent to the statement that the induced graph has a giant component. Incidentally, this is also equivalent to the condition that the graph does not have a giant 2-core, since stopping sets are equivalent to 2-cores in this setting. The fraction of nodes in the giant 2-core (if it exists) is equal to the unique solution of the equation $1 - \rho(1 - px) - x = 0$ in $(0, 1)$. (Compare this also to [36], which obtains formulas for the size of the giant component in a random irregular graph.)

LDPC codes from graphs with left degree 2 play an important role. For example, consider the stability condition proved in [9]. It states that small amounts of noise are correctable by belief propagation for an LDPC code with degree distribution given by $\lambda(x)$ and $\rho(x)$ if and only if $\lambda_2\rho'(1) < \left(\int_{-\infty}^{\infty} f(x)\mathrm{e}^{-x/2}\, \mathrm{d}x\right)^{-1}$, where $f(x)$ is the density of the log-likelihood of the channel. For example, for the BEC with erasure probability p we obtain $\lambda_2\rho'(1) < 1/p$, and for the BSC with error probability p we obtain $\lambda_2\rho'(1) < 1/\sqrt{p(1-p)}$. The stability condition is actually the condition that belief propagation is successful on the subgraph induced by message nodes of degree 2. This is not surprising, since these message nodes are those that are corrected last in the algorithm. (I do not give a proof of this, but this should sound reasonable, since message nodes of degree 2 receive very few messages in each round of iteration, and hence get corrected only when all the incoming messages are reasonably correct.)

15 Acknowledgements

Many thanks to Frank Kurth, Mehdi Molkaraie, and Mahmoud Rashidpour for pointing out typos and proof-reading an earlier version of this paper.

References

[1] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 1948.

[2] E. Berlekamp, R. McEliece, and H. van Tilborg, "On the inherent intractability of certain coding problems," *IEEE Trans. Inform. Theory*, vol. 24, pp. 384–386, 1978.

[3] P. Elias, "Coding for two noisy channels," in *Information Theory, Third London Symposium*, pp. 61–76, 1955.

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

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

[6] M. Luby, M. Mitzenmacher, and A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 364–373, 1998.

[7] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," *IEEE Trans. Inform. Theory*, vol. 47, pp. 585–598, 2001.

[8] 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, 2001.

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

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

[11] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, “Practical loss-resilient codes,” in *Proceedings of the 29th annual ACM Symposium on Theory of Computing*, pp. 150–159, 1997.

[12] L. Bazzi, T. Richardson, and R. Urbanke, “Exact thresholds and optimal codes for the binary symmetric channel and Gallager's decoding algorithm A,” *IEEE Trans. Inform. Theory*, vol. 47, 2001.

[13] A. Shokrollahi, “New sequences of linear time erasure codes approaching the channel capacity,” in *Proceedings of the 13th International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes* (M. Fossorier, H. Imai, S. Lin, and A. Poli, eds.), no. 1719 in Lecture Notes in Computer Science, pp. 65–76, 1999.

[14] V. V. Zyablov and M. S. Pinsker, “Estimation of error-correction complexity of Gallager low-density codes,” *Probl. Inform. Transm.*, vol. 11, pp. 18–28, 1976.

[15] M. R. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. Inform. Theory*, vol. 27, pp. 533–547, 1981.

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

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

[18] D. Burshtein and G. Miller, “Expander graph arguments for message-passing algorithms,” *IEEE Trans. Inform. Theory*, vol. 47, 2001.

[19] S.-Y. Chung, D. Forney, T. Richardson, and R. Urbanke, “On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit,” *IEEE Communication Letters*, vol. 5, pp. 58–60, 2001.

[20] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *proceedings of ACM SIGCOMM '98*, 1998.

[21] P. Oswald and A. Shokrollahi, "Capacity-achieving sequences for the erasure channel," *IEEE Trans. Inform. Theory*, vol. 48, pp. 3017–3028, 2002.

[22] N. Alon, S. Hoory, and N. Linial, "The Moore bound for irregular graphs." To appear, 2002.

[23] B. Bollobas, *Extremal Graph Theory.* Academic Press, 1978.

[24] G. A. Margulis, "Explicit group-theoretic constructions of combinatorial schemes and their applications in the construction of expanders and concentrators," *Problems of Information Transmission*, vol. 24, no. 1, pp. 39–46, 1988.

[25] A. Lubotzky, R. Phillips, and P. Sarnak, "Ramanujan graphs," *Combinatorica*, vol. 8, no. 3, pp. 261–277, 1988.

[26] J. Rosenthal and P. Vontobel, "Construction of LDPC codes using Ramanujan graphs and ideas from Margulis," in *Proceedings of the 38th Allerton Conference on Communication, Control, and Computing*, pp. 248–257, 2000.

[27] D. Divsalar, H. Jin, and R. McEliece, "Coding theorems for 'Turbo-like' codes," in *Proceedings of the 1998 Allerton Conference*, pp. 201–210, 1998.

[28] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," in *Proc. 2nd International Symposium on Turbo Codes*, pp. 1–8, 2000.

[29] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 638–656, 2001.

[30] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inform. Theory*, vol. 48, pp. 1570–1579, 2002.

[31] A. Orlitsky, R. Urbanke, K. Viswanathan, and J. Zhang, "Stopping sets and the girth of tanner graphs," in *Proceedings of the International Symposium on Information Theory*, 2002.

[32] A. Orlitsky and J. Zhang, "Finite-length analysis of LDPC codes with large left degrees," in *Proceedings of the International Symposium on Information Theory*, 2002.

[33] T. Richardson, A. Shokrollahi, and R. Urbanke, "Finite-length analysis of various low-density parity-check ensembles for the binary erasure channel," in *Proceedings of the International Symposium on Information Theory*, 2002.

[34] B. Bollobas, *Random Graphs.* Academic Press, 1985.

[35] M. Molloy and B. Reed, "A critical point for random graphs with a given degree sequence," *Random Structures and Algorithms*, vol. 6, pp. 161–179, 1995. can be downloaded from `http://citeseer.nj.nec.com/molloy95critical.html`.

[36] M. Molloy and B. Reed, "The size of the giant component of a random graph with a given degree sequence," *Combin. Probab. Comput.*, vol. 7, pp. 295–305, 1998. can be downloaded from `http://citeseer.nj.nec.com/molloy98size.html`.