ALGORITHMIC CODING THEORY*

Atri Rudra

State University of New York at Buffalo

1 Introduction

Error-correcting codes (or just **codes**) are clever ways of representing data so that one can recover the original information even if parts of it are corrupted. The basic idea is to judiciously introduce redundancy so that the original information can be recovered when parts of the (redundant) data have been corrupted.

Perhaps the most natural and common application of error correcting codes is for communication. For example, when packets are transmitted over the Internet, some of the packets get corrupted or dropped. To deal with this, multiple layers of the TCP/IP stack use a form of error correction called CRC Checksum [Peterson and Davis, 1996]. Codes are used when transmitting data over the telephone line or via cell phones. They are also used in deep space communication and in satellite broadcast. Codes also have applications in areas not directly related to communication. For example, codes are used heavily in data storage. CDs and DVDs can work even in the presence of scratches precisely because they use codes. Codes are used in Redundant Array of Inexpensive Disks (RAID) [Chen et al., 1994] and error correcting memory [Chen and Hsiao, 1984]. Codes are also deployed in other applications such as paper bar codes, for example, the bar

*To appear as a book chapter in the CRC Handbook on Algorithms and Complexity Theory edited by M. Atallah and M. Blanton.

Please send your comments to atri@cse.buffalo.edu

code used by UPS called MaxiCode [Chandler et al., 1989].

In addition to their widespread practical use, codes have also been used in theoretical computer science; especially in the 15 years or so (though there are a few notable exceptions). They have found numerous applications in computational complexity and cryptography. Doing justice to these connections is out of the scope of this chapter: we refer the reader to some of the surveys on these connections [Trevisan, 2004; Sudan, 2000; Guruswami, 2006c, 2004a].

In this chapter, we will think of codes in the communication scenario. In this framework, there is a sender who wants to send k message symbols over a noisy channel. The sender first encodes the k message symbols into n symbols (called a **codeword**) and then sends it over the channel. Since codes introduce redundancy, $n \geqslant k$. The receiver gets a **received word** consisting of n symbols. The receiver then tries to decode and recover the original k message symbols. We assume that the sender and receiver only communicate via the channel: specifically the receiver has no side information about the contents of the message. The main challenge in algorithmic coding theory is to come up with "good" codes along with efficient **encoding** and **decoding** algorithms. Next, we elaborate on some of the core issues in meeting the above challenge. The first issue is combinatorial while the second is algorithmic.

The combinatorial issue is the inherent tension between the amount of redundancy used and the number of errors such codes can tolerate. Let us illustrate this with two examples of codes whose symbols are bits: such codes as also called **binary codes**. In our first code (also called the repetition code), every message bit is repeated a fixed number of times (for concreteness say 100 times). Intuitively, this code should be able to tolerate many errors. In fact, here is a natural decoding procedure for such a code– for every contiguous 100 bits in the received word, declare the corresponding message bit to be the majority bit. In a typical scenario such a decoding procedure will recover the original message. However, the problem with the repetition code is that it uses too much redundancy. In our example, for every 100 bits that are transmitted, there is only one bit of information. On the other extreme is the so-called parity code. In a parity code, the parity of all

the bits in the message is tagged at the end of the code and then sent over the channel. For example, for the message 0000001, 00000011 is sent over the channel. Parity codes use the minimum amount of redundancy. However, for such a code it is not possible to even detect two errors. Using our last example, suppose that when 00000011 is transmitted, the first two bits are flipped. That is, the received word is 11000011. Note that 11000011 is a valid codeword for the message 1100001. Now if the decoding procedure gets the received word 11000011 it is justified in outputting 1100001 as the transmitted message, which of course is not desirable.

The problem with the parity code is that the codewords corresponding to two different messages do not differ by much. In other words, we need the codewords corresponding to different messages to be "far apart." However, in order to make sure that any two codewords corresponding to any pair of different messages are far apart, one needs to map messages into a large space (that is, n needs to be much larger than k). This in turn implies that the code uses more redundancy. In a nutshell, the combinatorial challenge is to design codes that balance these two contradictory goals well. We will formalize this inherent tradeoff more formally in the next section.

However, meeting the combinatorial challenge is just one side of the coin. To use codes in practice, one also needs efficient encoding and decoding algorithms. For this chapter, by an efficient algorithm we mean an algorithm whose running time is polynomial in n (we will also talk about extremely efficient algorithms that run in time that is linear in n). Typically, the definition of the code gives an efficient encoding procedure for "free."[1] The decoding procedure is generally the more challenging algorithmic task. As we will see in this chapter, the combinatorial and the algorithmic challenges are intertwined. Indeed, all the algorithms that we will consider exploit the specifics of the code (which in turn will be crucial to show that the codes have good combinatorial property).

A random code generally has the required good combinatorial properties with high probability. How-

[1] This will not be the case when we discuss linear time encoding.

ever, to transmit messages over the channel, one needs an explicit code that has good properties. This is not a great concern in some cases as one can search for a good code in a brute-force fashion (given that a code picked randomly from an appropriate ensemble has the required property). The bigger problem with random codes is that they do not have any inherent structure that can be exploited to design efficient decoding algorithms. Thus, the challenge in algorithmic coding theory is to design explicit codes with good combinatorial properties along with efficient encoding and decoding algorithms. This will be the underlying theme for the rest of the chapter.

Before we move on, we would like to point out an important aspect that we have ignored till now: How do we model the noise in the channel? The origins of coding theory can be traced to the seminal works of [Shannon, 1948] and [Hamming, 1950]. These lead to different ways of modeling noise. Shannon pioneered the practice of modeling the channel as a stochastic process while Hamming modeled the channel as an adversary. The analyses of algorithms in these two models are quite different and these two different schools of modeling the noise seem to be "divergent."

Needless to say, it is impossible to do justice to the numerous facets of algorithmic coding theory in the confines of this chapter. Instead of briefly touching upon a long list of research themes, we will focus on the following three that have seen a spurt of research activity in the last decade or so:

- Low-Density Parity-Check (or LDPC) codes were defined in the remarkable thesis of [Gallager, 1963]. These codes were more or less forgotten for a few decades. Interestingly, there has been a resurgence in research activity in such codes (along with iterative, message-passing decoding algorithms which were also first designed by Gallager) which has led to codes with good combinatorial properties along with extremely efficient encoding and decoding algorithms for stochastic channels.

- Expander codes are LDPC codes with some extra properties. These codes have given rise to codes with linear time encoding and decoding algorithms for the adversarial noise model.

- List decoding is a relaxation of the usual decoding procedure in which the decoding algorithm is allowed to output a list of possible transmitted messages. This allows for error correction from more adversarial errors than the usual decoding procedure. As another benefit list decoding bridges the gap between the Shannon and Hamming schools of coding theory.

Progress in some of the themes above are related: we will see glimpses of these connections in this chapter. In the next section, we will first define the basic notions of codes. Then we will review some early results in coding theory that will also set the stage for the results in Section 3. Interestingly, some of the techniques developed earlier are still crucial to many of the current developments.

2 Basics of Codes and Classical Results

We first fix some notation that will be used frequently in this chapter.

For any integer $m \geqslant 1$, we will use $[m]$ to denote the set $\{0, 1, \ldots, m-1\}$. Given positive integers n and m, we will denote the set of all length n vectors over $[m]$ by $[m]^n$. By default, all vectors in this chapter will be row vectors. The logarithm of x in base 2 will be denoted by $\log x$. For bases other than 2, we will specify the base of the logarithm explicitly: for example logarithm of x in base q will be denoted by $\log_q x$. A finite field with q elements will be denoted by $\mathbb{F}_q$. For any real value x in the range $0 \leqslant x \leqslant 1$, we will use $H_q(x) = x \log_q(q-1) - x \log_q x - (1-x) \log_q(1-x)$ to denote the q-ary entropy function. For the special case of $q = 2$, we will simply use $H(x)$ for $H_2(x)$. For any finite set S, we will use $|S|$ to denote the size of the set.

We now move on to the definitions of the basic notions of error correcting codes.

2.1 Basic Definitions for Codes

Let $q \geqslant 2$ be an integer. An error correcting code (or simply a code) C is a subset of $[q]^n$ for positive integers q and n. The elements of C are called codewords. The parameter q is called the alphabet size of C. In this case, we will also refer to C as a q-ary code. When $q = 2$, we will refer to C as a binary code. The parameter n is called the **block length** of the code. For a q-ary code C, the quantity $k = \log_q |C|$ is called the **dimension** of the code (this terminology makes more sense for certain classes of codes called **linear codes**, which we will discuss shortly). For a q-ary code C with block length n, its **rate** is defined as the ratio $R = \frac{\log_q |C|}{n}$.

Often it will be useful to use the following alternate way of looking at a code. We will think of a q-ary code C with block length n and $|C| = M$ as a function $[M] \to [q]^n$. Every element x in $[M]$ is called a message and $C(x)$ is its associated codeword. If M is a power of q, then we will think of the message as length-k vector in $[q]^k$.

Given any two vectors $\mathbf{v} = \langle v_1, \ldots, v_n \rangle$ and $\mathbf{u} = \langle u_1, \ldots, u_n \rangle$ in $[q]^n$, their **Hamming distance** (or simply distance), denoted by $\Delta(\mathbf{v}, \mathbf{u})$, is the number of positions that they differ in. The **(minimum) distance** of a code C is the minimum Hamming distance between any two codewords in the code. More formally $\mathrm{dist}(C) = \min_{\substack{c_1, c_2 \in C, \\ c_1 \neq c_2}} \Delta(c_1, c_2)$. The relative distance of a code C of block length n is defined as $\delta = \frac{\mathrm{dist}(C)}{n}$.

2.1.1 Code Families

The focus of this chapter will be on the asymptotic performance of algorithms. For such analysis to make sense, we need to work with an infinite family of codes instead of a single code. In particular, an infinite family of q-ary codes $\mathcal{C}$ is a collection $\{C_i | i \in \mathbb{Z}\}$, where for every i, C_i is a q-ary code of length n_i and

$n_i > n_{i-1}$. The rate of the family $\mathcal{C}$ is defined as

$$R(\mathcal{C}) = \lim\inf_i \left\{ \frac{\log_q |C_i|}{n_i} \right\}.$$

The relative distance of such a family is defined as

$$\delta(\mathcal{C}) = \lim\inf_i \left\{ \frac{\mathrm{dist}(C_i)}{n_i} \right\}.$$

From this point on, we will overload notation by referring to an infinite family of codes simply as a code. In particular, from now on, whenever we talk a code C of length n, rate R and relative distance δ, we will implicitly assume the following. We will think of n as large enough so that its rate R and relative distance δ are (essentially) same as the rate and the relative distance of the corresponding infinite family of codes.

Given this implicit understanding, we can talk about the asymptotics of different algorithms. In particular, we will say that an algorithm that works with a code of block length n is efficient if its running time is $O(n^c)$ for some fixed constant c.

2.2 Linear Codes

We will now consider an important sub-class of codes called linear codes. Let q be a prime power. A q-ary code C of block length n is said to be linear if it is a linear subspace (over some field $\mathbb{F}_q$) of the vector space $\mathbb{F}_q^n$.

The size of a q-ary linear code is obviously q^k for some integer k. In fact, k is the dimension of the corresponding subspace in $\mathbb{F}_q^n$. Thus, the dimension of the subspace is same as the dimension of the code. (This is the reason behind the terminology of dimension of a code.) We will denote a q-ary linear code of dimension k, length n and distance d as an $[n, k, d]_q$ code. Most of the time, we will drop the distance part and just refer to the code as an $[n, k]_q$ code.

Any $[n, k]_q$ code C can be defined in the following two ways.

- C can be defined as a set $\{\mathbf{xG}|\mathbf{x} \in \mathbb{F}_q^k\}$, where $\mathbf{G}$ is an $k \times n$ matrix over $\mathbb{F}_q$. $\mathbf{G}$ is called a generator matrix of C. Given the generator matrix $\mathbf{G}$ and a message $\mathbf{x} \in \mathbb{F}_q^n$, one can compute $C(x)$ using $O(nk)$ field operations (by multiplying $\mathbf{x}^T$ with $\mathbf{G}$).

- C can also be characterized by the following subspace $\{\mathbf{c}|\mathbf{c} \in \mathbb{F}_q^n \text{ and } H\mathbf{c}^T = \mathbf{0}\}$, where H is an $(n-k) \times n$ matrix over $\mathbb{F}_q$. H is called the parity check matrix of C.

2.3 Reed-Solomon Codes

In this subsection, we review a family of codes that we will encounter in multiple places in this chapter. A Reed-Solomon code (named after its inventors [Reed and Solomon, 1960]) is a linear code that is based on univariate polynomials over finite fields. More formally, an $[n, k+1]_q$ Reed-Solomon code with $k < n$ and $q \geqslant n$ is defined as follows. Let $\alpha_1, \ldots, \alpha_n$ be distinct elements from $\mathbb{F}_q$ (which is why we needed $q \geqslant n$). Every message $\mathbf{m} = \langle m_0, \ldots, m_k \rangle \in \mathbb{F}_q^{k+1}$ is thought of as a degree k polynomial over $\mathbb{F}_q$ by assigning the $k+1$ symbols to the $k+1$ coefficients of a degree k polynomial. In other words, $P_{\mathbf{m}}(X) = m_0 + m_1 X + \cdots + m_k X^k$. The codeword corresponding to $\mathbf{m}$ is defined as $RS(\mathbf{m}) = \langle P_{\mathbf{m}}(\alpha_1), \ldots, P_{\mathbf{m}}(\alpha_n) \rangle$. Now a degree k polynomial can have at most k roots in any field. This implies that any two distinct degree k polynomials can agree in at most k places. In other words, an $[n, k+1]_q$ Reed-Solomon code is an $[n, k+1, d = n-k]_q$ code.

By the Singleton bound (cf. Theorem 5), the distance of any code of dimension $k+1$ and length n is at most $n-k$. Thus, Reed-Solomon codes have optimal distance. The optimal distance property along with its nice algebraic structure has made Reed-Solomon codes the center of much research in coding theory. In addition to their nice theoretical applications, Reed-Solomon codes have found widespread use in practical applications. We refer the reader to [Wicker and Bhargava, 1999] for more details on some of the applications of Reed-Solomon codes.

Next, we turn to the question of how we will model the noise in the channel.

2.4 Modeling the Channel Noise

As was mentioned earlier, Shannon modeled the channel noise probabilistically. The channel is assumed to be memory-less, that is, the noise acts independently on each symbol. The channel has an input alphabet $\mathcal{X}$ and an output alphabet $\mathcal{Y}$. The behavior of the channel is modeled by a probability transition matrix M, where for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, $M(x, y)$ denotes the probability that when x is transmitted through the channel, the receiver gets y. We now look at some specific examples of such stochastic channels which we will encounter later in this chapter.

Definition 1 (Binary Symmetric Channel). *Given a real* $0 \leqslant p \leqslant 1/2$*, the* Binary Symmetric Channel *with* cross-over probability *p, denoted by* BSC_p *is defined as follows. The input and output alphabets are the same:* $\mathcal{X} = \mathcal{Y} = \{0, 1\}$*. For any pair* $(x, y) \in \{0, 1\} \times \{0, 1\}$*,* $M(x, y) = p$ *if* $x \neq y$ *and* $M(x, y) = 1 - p$ *otherwise.*

In other words, BSC_p channel flips each bit with probability p. There is a generalization of this channel to a q-ary alphabet.

Definition 2 (q-ary Symmetric Channel). *Let* $q \geqslant 2$ *be an integer and let* $0 \leqslant p \leqslant 1 - 1/q$ *be a real. The* q-ary Symmetric Channel *with* cross-over probability *p, denoted by* $\mathrm{QSC}_{q,p}$ *is defined as follows. The input and output alphabets are the same:* $\mathcal{X} = \mathcal{Y} = [q]$*. For any pair* $(x, y) \in [q] \times [q]$*,* $M(x, y) = \frac{p}{q-1}$ *if* $x \neq y$ *and* $M(x, y) = 1 - p$ *otherwise.*

In other words, in $\mathrm{QSC}_{q,p}$ every symbol from $[q]$ remains untouched with probability $1 - p$ while it is changed to each of the other symbols in $[q]$ with probability $\frac{p}{q-1}$. We will also look at the following noise model.

Definition 3 (Binary Erasure Channel). *Given a real* $0 \leqslant \alpha \leqslant 1$*, the* Binary Erasure Channel *with* erasure probability α*, denoted by* BEC_α *is defined as follows. The input alphabet is* $\mathcal{X} = \{0, 1\}$ *while the output*

channel is given by $\mathcal{Y} = \{0, 1, ?\}$ *where* ? *denotes an* erasure. *For any* $x \in \{0, 1\}$, $M(x, x) = 1 - \alpha$ *and* $M(x, ?) = \alpha$ *(the rest of the entries are* 0*).*

BEC_p is a strictly weaker noise model than BSC_p. The intuitive reason is that in BEC_p, the receiver knows which symbols have errors while this is not the case in BSC_p. In Section 2.5.1, we will formally see this difference.

Finally, we turn to the adversarial noise model pioneered by Hamming.

Definition 4 (Worst-Case Noise Model). *Let* $q \geqslant 2$ *be an integer and let* $0 \leqslant p \leqslant 1 - 1/q$ *be a real. Then the* q-ary Hamming Channel *with* maximum fraction of errors p*, denoted by* $\mathrm{HAM}_{q,p}$ *is defined as follows. The input and output alphabets of the channel are both* $[q]$*. When a codeword of length* n *is transmitted over* $\mathrm{HAM}_{q,p}$ *any arbitrary set of* pn *positions is in error. Also for each such position, the corresponding symbol* $x \in [q]$ *is mapped to a completely arbitrary symbol in* $[q] \setminus \{x\}$.

Two remarks are in order. First, we note that while the notations in Definitions 1, 2 and 3 are standard, the notation $\mathrm{HAM}_{q,p}$ in Definition 4 is not standard. We use this notation in this chapter for uniformity with the standard definitions of the stochastic noise models. Second, the worst case noise model is stronger than the stochastic noise model in the following sense (for concreteness we use $q = 2$ below). Given a decoding algorithm D for a code C of block length n that can tolerate p fraction of errors (that is, it recovers the original message for any error pattern with at most pn errors), one can use D for reliable communication over $\mathrm{BSC}_{p-\varepsilon}$ for any $\varepsilon > 0$ (that is, D fails to recover the transmitted message with a probability exponentially small in n). This claim follows from the simple argument. Note that the expected number of errors in $\mathrm{BSC}_{p-\varepsilon}$ is $(p - \varepsilon)n$. Now as the errors for each position are independent in $\mathrm{BSC}_{p-\varepsilon}$, by the Chernoff bound, with all but an exponentially small probability the actual number of errors will be at most pn in which case D can recover the transmitted message.

2.5 The Classical Results

2.5.1 Shannon's Result

The main contribution of Shannon's work was a precise characterization of when error correction can and cannot be achieved on stochastic channels. Let us make this statement more precise for the BSC_p channel. First we note that one cannot hope for perfect decoding. For example, there is some chance (albeit very tiny) that all the bits of the transmitted codeword may be flipped during transmission. In such a case there is no hope for any decoder to recover the transmitted message. Given a decoding function (or a decoding algorithm), we define the decoding error probability to be the maximum, over all transmitted messages, of the probability that the decoding function outputs a message different from the transmitted message. We would like the decoding error probability of our decoding algorithm to vanish with the block length. Ideally, we would like this error probability to be exponentially small in n.

Shannon proved a general theorem that pinpoints the tradeoff between the rate of the code and reliable communication for stochastic channels. In this section, we will instantiate Shannon's theorem for the stochastic channels we saw in Section 2.4.

Shannon's theorem implies the following for BSC_p:

Theorem 1. *For every real $0 \leqslant p < 1/2$ and $0 < \varepsilon < 1/2 - p$, there exists δ in $\Theta(\varepsilon^2)$ such that the following holds for large enough n. There exists an encoding function $E : \{0,1\}^k \to \{0,1\}^n$ and a decoding function $D : \{0,1\}^n \to \{0,1\}^k$ for $k = \lfloor (1 - H(p+\varepsilon))n \rfloor$ such that the decoding error probability is at most $2^{-\delta n}$.*

The proof of the above theorem is one of the early uses of the probabilistic method (cf. [Alon and Spencer, 1992]). The proof of Theorem 1 proceeds as follows. First the encoding function E is picked at random. That is, for every message $m \in \{0,1\}^k$, its corresponding codeword $E(m)$ is picked uniformly at randomly from $\{0,1\}^n$. Further, this choice is independent for every distinct message. The decoding function D performs maximum likelihood decoding (or MLD), that is, for every received word $y \in \{0,1\}^n$,

$D(y) = \arg\min_{m \in \{0,1\}^k} \Delta(y, E(m))$. In fact, the analysis shows that with high probability over the random choice of E, the functions E and (the corresponding MLD function) D satisfy Theorem 1. Shannon also showed the following converse result.

Theorem 2. *For every real $0 \leqslant p < 1/2$ and $0 < \varepsilon < 1/2 - p$ the following holds. For large enough n and $k = \lfloor (1 - H(p) + \varepsilon)n \rfloor$ there does not any exist any pair of encoding and decoding functions $E : \{0,1\}^k \to \{0,1\}^n$ and $D : \{0,1\}^n \to \{0,1\}^k$ such that D has small decoding error probability.*

We note that unlike Theorem 1, in Theorem 2, the decoding error probability can be as large as a constant (recall that in Theorem 1 the decoding error probability was exponentially small). Further, ε need not be a constant and can be as small as an inverse polynomial in n. The reader might be puzzled by the appearance of the entropy function in these two theorems. Without going into details of the analysis, we point out the following fact that is used crucially in the analysis: $2^{H(p)n}$ is a very good estimate of the number of vectors in $\{0,1\}^n$ that are within a Hamming distance of $\lfloor pn \rfloor$ from any fixed vector in $\{0,1\}^n$ (cf. [MacWilliams and Sloane, 1981]).

Theorems 1 and 2 pin down the best possible rate with which reliable communication over BSC_p can be achieved to $1 - H(p)$. This quantity $1 - H(p)$ is called the **capacity** of BSC_p.

Shannon's general result also implies the following special cases.

Theorem 3. *Let $0 \leqslant \alpha \leqslant 1$. The capacity of BEC_α is $1 - \alpha$.*

Theorem 4. *Let $q \geqslant 2$ be an integer and let $0 \leqslant p \leqslant 1 - 1/q$ be a real. The capacity of $\mathrm{QSC}_{q,p}$ is $1 - H_q(p)$.*

Problems Left Open by Shannon's Result Even though Shannon's theorem pinpoints the best possible rate for reliable communication over various stochastic channels, there are two unsatisfactory aspects of Shannon's proofs.

First, the encoding functions are chosen completely at random. Note that a general encoding function has no succinct representation. So even if one found a good encoding function as guaranteed by Shannon's

result, the lack of a succinct representation would seem to preclude any efficient encoding algorithm. However, as we saw in Section 2.2, linear codes do have a succinct representation and can be encoded in time quadratic in the block length. A natural question to ask is whether we can prove Theorem 1 and (the positive parts of) Theorems 3 and 4 for random linear codes[2]? The answer is yes (and in fact, the analysis of Theorem 1 becomes somewhat easier for random linear codes). However, these codes are still not explicit. This is the right time to clarify what we mean by an explicit code. We say that a code is explicit if a succinct representation of such a code can be computed by an algorithm that runs in time polynomial in the block length of the code.

As was mentioned in the introduction, even though having an explicit encoding function that satisfies (for example) the conditions of Theorem 1 would be nice, in some applications one could use a brute force algorithm that runs in exponential time to find a good encoding function. The real problem with a random (linear) codes is that they do not seem to have any structural property that can be exploited to design efficient decoding algorithms. Recall that Shannon's proof used the maximum likelihood decoding function. This is a notoriously hard decoding function to implement in polynomial time. In fact, there exist linear codes for which MLD is intractable [Berlekamp et al., 1978]. Further, for any non-trivial code, the only known implementation of the MLD function is the brute force exponential time algorithm. Thus, the grand challenge in algorithmic coding theory after Shanon's work was the following:

Question 1. *Can one design an explicit code with rate that achieves the capacity of* BSC_p *(and other stochastic channels)? Further, can one come up with efficient decoders with negligible decoding error probability for such codes?*

We will see a positive resolution of the question above in Section 3.1 for BSC_p. In fact to get within ε of capacity for codes of block length n, results in Section 3.1 achieve a decoding time complexity of $n2^{1/\varepsilon^c}$,

[2] A random linear code with encoding function that maps k bits to n bits can be chosen by picking a random $k \times n$ matrix over the appropriate alphabet as the generator matrix.

where $c \geqslant 2$ is a constant (the encoding complexity is $n/\varepsilon^{O(1)}$). Now if we think of ε as constant and n as growing (as we will do in most of this chapter), then this implies linear-time encoding and decoding. However, codes in practice typically have moderate block lengths and the $2^{1/\varepsilon^c}$ factor becomes prohibitive. For example even to get within 10% of capacity, this factor is at least 2^{100}! This leads to the following question.

Question 2. *Can one design an explicit code with rate that is within ε of the capacity of BSC_p along with encoders and decoders (with negligible decoding error probability for such codes) that have a time complexity of $n/\varepsilon^{O(1)}$?*

In Section 3.2, we will review some work from the turn of the century that answers the above question in the affirmative for the weaker BEC_α model.

2.6 Hamming's Work

As was mentioned earlier, Hamming studied worst case noise. In such a scenario, the distance of the code becomes an important parameter: the larger the distance of the code, the larger the number of errors that can be corrected. Before we make this notion more precise, let us briefly look at what it means to do error correction in this noise model. In Hamming's model, we will insist on perfect decoding. Note that if all the symbols can be in error then we cannot hope for such a decoding. Hence, we will also need an upper bound on the number of errors that can be tolerated. More precisely, we will examine unique decoding. Under unique decoding, given an upper bound on the total number of errors $p_U n$ (so p_U is the fraction of errors), the decoding algorithm has to output the transmitted message for any error pattern with at most $p_U n$ many errors. Given this setup, the natural question to ask is how large can p_U be? The next proposition relates this question to the distance of the code.

Proposition 1. *For any q-ary code C of block length n and minimum distance d, there does not exist any*

unique decoding algorithm for $\mathrm{HAM}_{q,p}$ *for any* $p \geqslant \frac{1}{n}\lceil\frac{d}{2}\rceil$. *Further for any* $p \leqslant \frac{1}{n}\lfloor\frac{d-1}{2}\rfloor$, *unique decoding can be done on* $\mathrm{HAM}_{q,p}$.

The negative result follows from the following argument. Consider two codewords $c_1, c_2 \in C$ such that $\Delta(c_1, c_2) = d$, where for simplicity assume that d is even. Now consider a received word $\mathbf{y}$ such that $\Delta(\mathbf{y}, c_1) = \Delta(\mathbf{y}, c_2) = d/2$. Note that this is a possible received word under $\mathrm{HAM}_{q,d/(2n)}$. Now the decoder has no way of knowing whether c_1 or c_2 is the transmitted codeword[3]. Thus, unique decoding is not possible. For the positive side of the argument, using triangle inequality, one can show that every received word in the channel $\mathrm{HAM}_{q,(d-1)/(2n)}$ has a unique closest by codeword and thus, for example, an MLD algorithm can recover the transmitted codeword.

The above result pinpoints the maximum number of errors that can be corrected using unique decoding to be $d/2$. Thus, in order to study the tradeoff between the rate of the code and the fraction of errors that can be tolerated (as was done in the Shannon's work) it is enough to study the tradeoff between the rate and the (relative) distances of codes. Further, we mention that for specific families of codes (such as Reed-Solomon codes from Section 2.3 among others) polynomial time unique decoding algorithms are known. There is a huge body of work that deals with the tradeoff between rate and distance (and polynomial time algorithms to correct up to half the distance). This body of work is discussed in detail in any standard coding theory text book such as [MacWilliams and Sloane, 1981; van Lint, 1999]. For this chapter, we will need the following tradeoff.

Theorem 5 (Singleton Bound). *Any code of dimension* k *and block length* n *has distance* $d \leqslant n - k + 1$.

Let us now return to the tradeoff between the rate of a code and the fraction of errors that can be corrected. By Proposition 1 and Theorem 5, to correct p fraction of errors via unique decoding, the code can have a rate of at most $1 - 2p$. This tradeoff can also be achieved by explicit code and efficient encoding

[3]Recall we are assuming that the sender and the receiver only communicate via the channel.

and decoding algorithms. In particular, a Reed-Solomon code with relative distance of $\delta = 2p$ has a rate (slightly more than) $1 - 2p$. Further, as was mentioned earlier, there are polynomial time unique decoding algorithms that can correct up to $\delta/2 = p$ fraction of errors. Recall that as a Reed-Solomon code is a linear code, a quadratic time decoding algorithm is immediate. However, if we are interested in extremely efficient algorithms, i.e. linear time encoding and decoding algorithms, then algorithms for Reed-Solomon codes do not suffice. This leads to the following question:

Question 3. *Let $0 < p < 1$ and $\varepsilon > 0$ be arbitrary reals. Do explicit codes of rate $1 - 2p - \varepsilon$ exist such that they can be encoded as well as uniquely decoded from a p fraction of errors in time linear in the block length of the code?*

In Section 3.3, we will see a positive resolution of the above question.

Let us pause for a bit and see how the bound of $1 - 2p$ compares with corresponding capacity result for $\mathrm{QSC}_{q,p}$. Theorem 4 states that it is possible to have reliable communication for rates at most $H_q(1 - p)$. Now for large enough q, $H_q(1 - p)$ is almost $1 - p$ (see for example [Rudra, 2007, Chap. 2]). Thus, there is a gap in how much redundancy one needs to use to achieve reliable communication in Shannon's stochastic noise model and Hamming's worst case noise model. Another way to look at this gap is the following question. Given a code of rate R (say over large alphabets), how much error can we hope to tolerate? In the q-ary Symmetric Channel, we can tolerate close to $1 - R$ fraction of errors, while in the worst case noise model we can only tolerate half as much, that is, $(1 - R)/2$ fraction of errors. In the next subsection, we will look at a relaxation of unique decoding called list decoding that can bridge this gap.

Before we wrap up this subsection, we mention another well-known tradeoff between rate and distance (because any survey on algorithmic coding theory needs to do so).

Theorem 6 (Gilbert-Varshamov Bound). *For any $q \geqslant 2$, there exist q-ary linear codes with rate R and relative distance δ such that $\delta \geqslant H_q^{-1}(1 - R)$,*

Theorem 6 is proved by picking a random linear code of large enough block length n and dimension Rn and then showing that with high probability, the relative distance of such codes is at least $H_q^{-1}(1-R)$.

2.7 List Decoding

Recall that the upper bound on the fraction of errors that can be corrected via unique decoding followed by exhibiting a received word that has two codewords in a code of relative distance δ within a fractional Hamming distance of at most $\delta/2$. However, this is an overly pessimistic estimate of the maximum fraction of errors that can be corrected, since the way Hamming spheres pack in space, for most choices of the received word there will be at most one codeword within distance p from it even for p much greater than $\delta/2$. Therefore, always insisting on a unique answer will preclude decoding most such received words owing to a few pathological received words that have more than one codeword within distance roughly $\delta/2$ from them.

A notion called list decoding provides a clean way to get around this predicament and yet deal with worst-case error patterns. Under list decoding, the decoder is required to output a list of all codewords within distance p from the received word. The notion of list decoding itself is quite old and dates back to work in 1950's by Elias and Wozencraft [Elias, 1957; Wozencraft, 1958]. However, the algorithmic aspects of list decoding were not revived until the more recent works [Goldreich and Levin, 1989; Sudan, 1997] which studied the problem for complexity-theoretic motivations.

We now state the definition of list decoding that we will use in this chapter.

Definition 5 (List-Decodable Codes). *Let C be a q-ary code of block length n. Let $L \geqslant 1$ be an integer and $0 < \rho < 1$ be a real. Then C is called (ρ, L)-list decodable if every Hamming ball of radius ρn has at most L codewords in it. That is, for every $\mathbf{y} \in \mathbb{F}_q^n$, $|\{c \in C | \Delta(c, \mathbf{y}) \leqslant \rho n\}| \leqslant L$.*

2.7.1 List-Decoding Capacity

In Section 2.7, we informally argued that list decoding has the potential to correct more errors than unique decoding. We will now make this statement more precise. The following results were implicit in [Zyablov and Pinsker, 1982] but were formally stated and proved in [Elias, 1991].

Theorem 7. *Let $q \geqslant 2$ be an integer and $0 < \delta \leqslant 1$ be a real. For any integer $L \geqslant 1$ and any real $0 < \rho < 1 - 1/q$, there exists a (ρ, L)-list decodable q-ary code with rate at least $1 - H_q(\rho) - \frac{1}{L+1} - \frac{1}{n^\delta}$.*

Theorem 8. *Let $q \geqslant 2$ be an integer and $0 < \rho \leqslant 1 - 1/q$. For every $\varepsilon > 0$, there do not exist any q-ary code with rate $1 - H_q(\rho) + \varepsilon$ that is $(\rho, L(n))$-list decodable for any function $L(n)$ that is polynomially bounded in n.*

Theorems 7 and 8 show that to be able to list decode with small lists on $\mathrm{HAM}_{q,p}$ the best rate possible is $1 - H_q(p)$. Another way to interpret Theorem 7 is the following. One can have codes of rate $1 - H_q(p) - \varepsilon$ that can tolerate p fraction of adversarial errors under list decoding with a worst-case list size of $O(1/\varepsilon)$. In other words, one can have the same tradeoff between rate and fraction of errors as in $\mathrm{QSC}_{q,p}$ if one is willing to deal with a small list of possible answers. Due to this similarity, we will call the quantity $1 - H_q(p)$ the list-decoding capacity of the $\mathrm{HAM}_{q,p}$ channel.

The proof of Theorem 7 is similar to the proof in Shannon's work: it can be shown that with high probability a random code has the desired property. Theorem 8 is proved by showing that for any code of rate $1 - H_q(p) + \varepsilon$, there exists a received word with superpolynomially many codewords with a relative Hamming distance of p. Note that as a list-decoding algorithm must output all the near by codewords, this precludes the existence of a polynomial time list-decoding algorithm. Now, the only list-decoding algorithm known for random codes used in the proofs of Theorem 7 and 8 is the brute-force list-decoding algorithm that runs in exponential time. Thus, the grand challenge for list decoding is the following.

Question 4. *Can one design an explicit code with rate that achieves the list-decoding capacity of* $\mathrm{HAM}_{q,p}$*? Further, can one come up with efficient list-decoding algorithms for such codes?*

In Section 3.4, we will look at some recent work that answer the question above in the positive for large enough q.

3 Explicit Constructions and Efficient Algorithms

3.1 Code Concatenation

In this subsection, we return to Question 1. Forney answered the question in the affirmative by using a code composition method called code concatenation. Before we show how Forney used concatenated codes to design binary codes that achieve the capacity of BSC_p, we digress a bit to talk about code concatenation.

3.1.1 Background and Definition

We start by formally defining concatenated codes. Say we are interested in a code over $[q]$. Then the outer code C_{out} is defined over $[Q]$, where $Q = q^k$ for some positive integer k. The second code, called the inner code is defined over $[q]$ and is of dimension k (Note that the message space of C_{in} and the alphabet of C_{out} have the same size). The concatenated code, denoted by $C = C_{out} \circ C_{in}$, is defined as follows. Let the rate of C_{out} be R and let the block lengths of C_{out} and C_{in} be N and n respectively. Define $K = RN$ and $r = k/n$. The input to C is a vector $\mathbf{m} = \langle m_1, \ldots, m_K \rangle \in ([q]^k)^K$. Let $C_{out}(\mathbf{m}) = \langle x_1, \ldots, x_N \rangle$. The codeword in C corresponding to $\mathbf{m}$ is defined as follows

$$C(\mathbf{m}) = \langle C_{in}(x_1), C_{in}(x_2), \ldots, C_{in}(x_N) \rangle.$$

It is easy to check that C has rate rR, dimension kK and block length nN. While the concatenated construction still requires an inner q-ary code, this is a small/short code with block length n, which is typically

logarithmic or smaller in the length of the outer code. A good choice for the inner code can therefore be found efficiently by a brute-force search, leading to a polynomial time construction of the final concatenated code.

Ever since its discovery and initial use in [Forney, 1966], code concatenation has been a powerful tool for constructing error-correcting codes. The popularity of code concatenation arises due to the fact that is often difficult to give a direct construction of codes over small alphabets. On the other hand, over large alphabets, an array of powerful algebraic constructions (such as Reed-Solomon codes) with excellent parameters are available. This paradigm draws its power from the fact that a concatenated code, roughly speaking, inherits the good features of both the outer and inner codes. For example, the rate of the concatenated code is the product of the rates of the outer and inner codes, and the minimum distance is at least the product of the distances of the outer and inner codes. The alphabet size of the concatenated code equals that of the inner code.

3.1.2 Achieving Capacity on BSC_p

We now return to the question of achieving the capacity of BSC_p for some $0 \leqslant p < 1/2$. More precisely, say we want to construct codes of rate $1 - H(p) - \varepsilon$ that allow for reliable communication over BSC_p. As mentioned earlier, we will use a concatenated code to achieve this. We now spell out the details.

Let b be an integer parameter we will fix later and say we want to construct a binary code C. We will pick $C = C_{out} \circ C_{in}$, where the outer and inner codes have the following properties:

- C_{out} is a code over $\mathbb{F}_{2^b}$ of length n with rate $1 - \varepsilon/2$. Further, let D_{out} be unique decoding algorithm that can correct a small fraction $\gamma = \gamma(\varepsilon)$ fraction of worst-case errors.

- C_{in} is a binary code of dimension b with rate $1 - H(p) - \varepsilon/2$. Further, let D_{in} be a decoding algorithm that can recover the transmitted message over BSC_p with probability at least $1 - \gamma/2$.

Let us defer for a bit how we obtain the codes C_{out} and C_{in} (and their decoding algorithms). Assuming we have the requisite outer and inner codes, we analyze the parameters of $C = C_{out} \circ C_{in}$ and present a natural decoding algorithm for C. Note that C has rate $(1 - \varepsilon/2) \cdot (1 - H(p) - \varepsilon/2) \geqslant 1 - H(p) - \varepsilon$ as required (its block length is $N = nb/(1 - H(p) - \varepsilon/2)$).

The decoding algorithm for C is fairly natural. For notational convenience define $b' = b/(1 - H(p) - \varepsilon/2)$. The decoding algorithm has the following steps:

1. Given a received word $\mathbf{y} \in \mathbb{F}_2^N$, divide it up into n contiguous blocks, each consisting of b' bits– denote the ith block by y_i. Note that b' is the block length of C_{in}.

2. For every i, decode y_i, using D_{in} to get y'_i. Let $\mathbf{y}' = (y'_1, \ldots, y'_n) \in \mathbb{F}_{2^b}^n$ be the intermediate result. Note that $\mathbf{y}'$ is a valid received word for D_{out}.

3. Decode $\mathbf{y}'$ using D_{out} to get a message m and output that as the transmitted message.

We now briefly argue that the decoder above recovers the transmitted message with high probability. Note that as the noise in BSC_p acts on each bit independently, in step 2 for any block i, y_i is a valid received word given a codeword from C_{in} was transmitted over BSC_p. Thus, by the stated property of D_{in}, for any block D_{in} makes an error with probability at most $\gamma/2$. Further, as the noise on each block is independent, by the Chernoff bound, except with exponentially small probability, at most γ fraction of the blocks are decoded incorrectly. In other words, at the beginning of Step 3, $\mathbf{y}'$ is a received word with at most γ fraction of errors. Thus, by the stated property of D_{out}, step 3 will output the transmitted message for C with high probability as desired.

We now return to the task of getting our hands on appropriate outer and inner codes. We start with C_{in}. Note that by Theorem 1, there exists a C_{in} with required properties if we pick b in $\Theta(\log(1/\gamma)/\varepsilon^2)$. Further, by the discussion in Section 2.5.1, C_{in} is also a linear code. This implies that such a code can be found by a brute-force search (which will imply constant time complexity that depends only on ε). Further, C_{in} can

be encoded in time $O(b^2)$ (since any linear code has quadratic encoding time complexity). D_{in} is the MLD algorithm which runs in time $2^{O(b)}$.

We now turn to the outer codes. Interestingly, C_{out} is also a binary code. We think of C_{out} as a code over $\mathbb{F}_{2^b}$ by simply grouping together blocks of b bits. Note that if a decoding algorithm can decode from γ fraction of worst case errors over $\mathbb{F}_2$, then it can also decode from γ fraction of worst case errors over $\mathbb{F}_{2^b}$. In [Forney, 1966], the code C_{out} was in turn another concatenated code (where the outer code is the Reed-Solomon code and the inner code was chosen from the so called Wozencraft ensemble [Justesen, 1972]). However, for our purposes we will use binary codes from Theorem 13 (with γ in $O(\varepsilon^2)$). Finally, we estimate the decoding time complexity for C. The decoding time is dominated by Step 2 of the algorithm, which takes time $n2^{O(b)}$. Recalling that $n = N(1 - H(p) - \varepsilon/2)/b$ and the instantiations of γ and b, we have the following:

Theorem 9. *There exist explicit codes that get within ε of capacity of BSC_p that can be decoded and encoded in time $N2^{O(\log(1/\varepsilon)/\varepsilon^2)}$ and $N/\varepsilon^{O(1)}$ respectively (where N is the block length of the code).*

Thus, we have answered Question 1. In fact, encoding and decoding can be carried out in linear time (assuming that ε is fixed while N is increasing).

3.2 LDPC Codes

In this subsection, we return to Question 2. Unfortunately, a positive answer to this question is not known to date. In this subsection we will look at a family of codes called Low Density Parity Check (or LDPC) codes along with iterative message passing decoding algorithms that experimentally seem to answer Question 2 in the affirmative, though no theoretical guarantees are known. However, for the weaker model of BEC_α, the corresponding question can be answered in the affirmative. We will focus mostly on BEC in this subsection, though we will also discuss results in BSC. We start with the definitions related to LDPC codes and a high level overview of iterative message passing decoding algorithms. Both LDPC codes and message passing

algorithms were introduced and studied in the remarkable thesis [Gallager, 1963] which was way ahead of its time.

3.2.1 Definitions

LDPC codes are linear binary codes with sparse parity check matrices. In particular, each row and column of the parity check matrix has at most a fixed number of 1s. A useful way to think about an LDPC code is by its corresponding factor graph. Given a binary code of dimension k and block length n, its factor graph is a bipartite graph where the left side has n vertices called variable nodes (each of which corresponds to a position in a codeword). The right side has $n - k$ vertices called check nodes (corresponding to a parity check or a row of the parity check matrix). Every check node is connected to variable nodes whose corresponding codeword symbol appears in the associated parity check. In other words, the incidence matrix of the factor graph is exactly the parity check matrix.

Gallager considered regular LDPC codes for which the corresponding factor graph is (d_v, d_c)-regular (that is, every left vertex has degree d_v and every right vertex has degree d_c). Later on in the subsection, we will briefly look at irregular LDPC codes for which the corresponding factor graph is not regular. For the rest of the subsection we will exclusively think of the LDPC codes in terms of their factor graphs. Finally, for the remainder of the subsection, for notational convenience we will think of the bits to take values form $\{-1, 1\}$. -1 and 1 correspond to the "conventional" 1 and 0 respectively.

3.2.2 Iterative Message Passing Decoding Algorithms for LDPC Codes

As the name suggests, iterative message passing decoding algorithms occur in rounds. In particular, in alternating rounds, check nodes pass messages to their neighboring variable nodes and vice-versa. Initially, every variable node v_j ($1 \leqslant j \leqslant n$) has its corresponding symbol in the received word y_j (note that for BSC and BEC channels this is a random variable). In the first round, every variable node sends a message to its

neighboring check nodes (which typically is just y_j for v_j). A check node after receiving messages from its neighboring variable nodes, computes a message using a pre-determined function on the received messages and sends it back to its neighboring variable nodes. The variable node v_j upon receiving messages from its neighboring check nodes computes another message using another pre-determined function on the received messages and y_j and sends it to its neighboring check nodes. Messages are passed back and forth in this manner till a predetermined number of rounds is completed.

Three remarks are in order. First, the functions used by variable and check nodes to compute messages can depend on the iteration number. However, typically these functions have the same structure over different rounds. Second, the message sent to a neighboring variable (resp. check) node v (resp. c) by a check (resp. variable) node is independent of the message sent to it by v (resp. c) in the previous round. In other words only extrinsic information is used to compute new messages. This is a very important restriction that is useful in the analysis of the algorithm. Third, there is an intuitive interpretation of the messages in the algorithm. In particular, they are supposed to be votes on the value of codeword bits. If the messages take values in $\{-1, 1\}$ then they correspond to the actual bit value. One can add 0 to denote an erasure or an "absentee" vote. If a message takes a real value then the sign will denote the vote, while the absolute value denotes the confidence in the vote.

We now state some of the results in [Gallager, 1963], which among other things will give concrete examples of the general paradigm discussed above.

3.2.3 Gallager's Work

To present the main ideas in Gallager's work we apply his methods to BEC even though [Gallager, 1963] did not explicitly consider this channel. The first step is to design a (d_v, d_c)-regular factor graph with n variable nodes such that it has no cycle of sub-logarithmic size (i.e., the girth of the graph is $\Omega(\log n)$).

In the second step we need to specify the functions that variable and check nodes use to compute the

messages. For the BEC the following is the natural choice. When a variable node needs to send a message to its neighboring check node it sends the corresponding codeword bit if it is known (either from the received word or as a message from a check node in an earlier round), otherwise it sends an erasure. On the other hand when a check node c needs to send a message to a variable node v, it sends an erasure if at least one neighbor other than v sent it an erasure in the previous round; Otherwise it sends v the parity of the messages it received in the previous round. Note that this parity is the correct codeword bit for v. This algorithm can be implemented as a peeling decoder, where each edge of the factor graph is used to pass a message only once. This implies that the decoding algorithm is linear in the number of edges (which in turn is linear in n).

The analysis of the decoding algorithm above proceeds as follows. The first step of the analysis is to obtain a recursive relation on the fraction of messages that are erasures in a particular round (in terms of the fraction in the previous round). This part crucially uses the facts that only extrinsic information is used to compute new messages and that the factor graph has logarithmic girth. The second step of the analysis is to come up with a threshold α^* on the erasure probability such that for any $\alpha < \alpha^*$, under BEC_α, the decoder in its last iteration would have a negligible fraction of messages as erasures. Using the fact that the number of iterations is logarithmic in n (since the girth is logarithmic), it can be shown that the expected fraction of messages that are erasures vanishes as n increases. This implies that except with a negligible probability, the decoder outputs the transmitted codeword. Making the above discussion formal results in the following (though this does not achieve the capacity of the BEC).

Theorem 10. *For integers $3 \leqslant d_v < d_c$, there exists an explicit family of codes of rate $1 - \frac{d_v}{d_c}$ that can be reliably decoded in linear time on BEC_α, provided $\alpha < \alpha^*$. The threshold α^* is given by the expression $\frac{1-\gamma}{(1-\gamma^{d_c-1})^{d_v-1}}$, where γ is the unique positive root of the polynomial $((d_v-1)(d_c-1)-1)x^{d_c-2} - \sum_{j=0}^{d_c-3} x^j$.*

We briefly mention how Gallager's result for BSC "extends" the techniques above. The first main

difference is in the maps used to compute new messages. The check nodes use the parity of the all incoming messages as their function. For the variable nodes, Gallager proposed two functions. In the first one, which leads to the so called Gallager's Algorithm A, the variable node sends its received bit to a check node unless messages coming from all other check nodes in the previous node indicate otherwise. In that case, it sends the complement of the received value. In the second function which leads to Gallager's algorithm B, the variable node sends the complement of its received value if more than a pre-fixed number of messages from check nodes in the previous round say otherwise.

3.2.4 Irregular LDPC Codes

We now briefly look at the some more recent work that builds on Gallager's work and achieves the capacity of BEC with extremely efficient encoding and decoding algorithms. For rest of the subsection, we will concentrate mostly on the BEC.

The work of [Luby et al., 2001a] introduced the study of LDPC codes based on irregular factor graphs. We start with some intuition as to why having an irregular factor graph might help while running Gallager's decoding algorithms. From the perspective of the variable node, it is beneficial to have more adjacent check nodes as the variable node would obtain more information from the check nodes which should intuitively help the variable node in computing its bit value. On the other side, a check node would prefer to have fewer adjacent variable nodes because the parity function becomes more unpredictable with more inputs. However, for the rate to be positive, the number of check nodes has to be fewer than the number of variable nodes. Meeting these contradictory goals is difficult. Due to their less stringent conditions on vertex degrees irregular graphs provide more flexibility in meeting the competing degree requirements discussed above. The motivation of having a spread of degrees is that variable nodes with high degree could be expected to converge to their correct value faster than their regular factor graph counterpart. This would in turn lead to the neighboring check nodes getting better information, which would then be relayed to variable nodes with

smaller degrees. Thus, the hope is that this cascading effect would lead to better algorithms. Of course, making this intuition formal requires some effort, which leads to the following result.

Theorem 11. *There exist codes that get within ε of capacity of BEC_α. Further, these codes can be decoded and encoded in time $n\log(1/\varepsilon)$.*

We wrap up this section with two remarks regarding the above result. First, unlike Gallager's result designing an explicit irregular factor graph with a large girth seems like a difficult task. The results of [Luby et al., 2001a] instead work on ensembles of irregular factor graphs. By an ensemble of codes, we mean a family of codes that are parametrized by two distributions $\{\lambda_i\}$ and $\{\rho_i\}$. Here λ_i (resp. ρ_i) is the fraction of edges that are incident on variable (resp. check) nodes of degree i. It is shown that there exist appropriate choices of these distributions for which if one samples from the corresponding sample then with high probability the resulting factor graph will have the required properties. Thus, we no longer have explicit construction of codes. Second, additional work needs to be done to get linear time encoding schemes. [Luby et al., 2001a] used a cascade of so called low-density generator matrix code to obtain linear time encoding. An alternate approach from [Richardson and Urbanke, 2001b] is to find an "approximate" lower triangulation of the parity check matrix that is still sparse, which suffices for a linear time encoding.

Thus, Theorem 11 answers Question 2, though for the weaker BEC. Techniques mentioned above have been extended to the other stochastic channels like the BSC. These perform very well experimentally, though rigorous theoretical guarantees have so far been elusive.

3.3 Expander Codes

In this subsection, we will look at codes that are constructed from certain combinatorial objects called expanders. Expanders are sparse graphs that are still well-connected. There are two main ways in which expanders are used to define codes: (i) Using the graph as a factor graph for LDPC codes, which we discussed in Section 3.2 and (ii) Using the edges of the graph to "move" symbols around during encoding. Both

techniques will be required to obtain a positive answer to Question 3. Before we delve into the techniques mentioned above, we first define the version of expanders graphs that we will need in this subsection.

Definition 6 (Expander Graphs). *A bipartite graph $G = (L, R, E)$ is said to be an $(\ell, n, d_L, \alpha, \gamma)$-*expander *if the following holds. $|L| = \ell$, $|R| = n$ and the degree of each node in L is d_L. More importantly, for every subset of nodes $A \subseteq L$ with $|A| \leqslant \alpha\ell$, its set of neighbors in Y is at least $\gamma|A|$.*

We will think degree parameter d_L as a constant. Ideally, we would like γ to be as large as possible. Note that $\gamma \leqslant d_L$. It can be shown that random bipartite graph with the left-degrees being d_L with high probability have $\gamma = d(1 - \varepsilon)$ for any $\varepsilon > 0$. Such expanders are called loss-less expanders.

3.3.1 The Basic Construction

As was mentioned earlier, one way to use an $(n, m, d_L, \alpha, \gamma)$-expander is as a factor graph for an LDPC code. Note that such a code will have rate at least $1 - m/n$. The decoding algorithm for such a code follows by several rounds of "bit-flipping." More precisely, in every round, every variable node flips its value in parallel if the number of neighboring checks nodes with unsatisfied parity checks is at least $2d/3$. Otherwise it does not change its value. We make some remarks on this decoding algorithm. First, the proof of correctness of this algorithm proceeds by showing that if the fraction of errors to begin with is bounded then in each round the number of bits that are in error decrease by a factor of $2/3$. This claim crucially uses the connectivity and sparsity properties of the underlying expander graph to argue that most of the unsatisfied check nodes are adjacent to a single variable node that has an erroneous bit (and thus, flipping the value corresponding to that variable node would satisfy the parity check). There is a danger that too many variable nodes with correct values can also flip their bits. However, again using the connectivity property of the underlying expander one can show that this is not the case. Second, a careful accounting for the nodes that need to be flipped in each round leads to an linear time implementation. Picking parameters correctly, one can make the argument above rigorous and obtain the following result.

Theorem 12. *Let* $0 < \varepsilon < \frac{1}{12}$ *and* C *be the LDPC code corresponding to an* $(n, m, d, \alpha, d(1-\varepsilon))$*-expander.* C *has rate at least* $1 - m/n$ *and can be decoded from a* $\alpha(1 - 3\varepsilon)$ *fraction of errors in* $O(n)$ *time.*

Note that for the code above to be explicit, one needs explicit constructions of loss-less expanders. Such a construction was recently obtained [Capalbo et al., 2002].

3.3.2 Linear Time Encoding

We now briefly discuss the code construction in [Spielman, 1996] that in addition to the linear decoding complexity as guaranteed by Theorem 12 also has linear encoding time complexity. These codes are the only binary codes known that have provable linear time encoding and decoding for worst-case errors. Unlike the construction of Theorem 12 where expanders are used to define the parity check matrix, in Spielman's construction, the expander graph is used to define the generator matrix. In particular, Spielman's construction first defines what he called "error reduction codes." These are systematic codes. that is, the codeword consists of the message bits followed by some parity check bits. The structure of the expander graphs is used to compute the parity checks from the message bits. In this construction, the variable nodes only correspond to the message bits. Further, the expander graphs are (d_v, d_c)-regular graphs (as was the case with Gallager's LDPC codes from Section 3.2.3). Thus, such error reduction codes have a trivial linear-time encoding algorithm. Unfortunately, these codes by themselves cannot be decoded from a large number of errors. However, an algorithm similar to the one used in Theorem 12 can be used to obtain an intermediate received word that has at most half the number of errors in the original received word. Such error reduction codes can be used recursively to obtain the final linear-time encodable and decodable codes.

Here we just state a special case of the general result in [Spielman, 1996].

Theorem 13. *For every small enough* $\gamma > 0$*, there exists explicit binary codes of rate* $1/(1 + \gamma)$ *that can be encoded in linear time and decoded in linear time from up to* $\Omega(\gamma^2/\log^2(1/\gamma))$ *fraction of errors.*

3.3.3 Approaching Half the Singleton Bound

Unfortunately the result of [Spielman, 1996] only works for about 10^{-6} fraction of errors. We will now see another general technique introduced in [Alon et al., 1995] that uses expanders to improve the fraction of errors that can be corrected. We start with an informal description of the construction. The code uses two codes C_{out} and C_{in} and an expander graph G. The final code C^* is constructed as follows. Let $C' = C_{out} \circ C_{in}$ be the code concatenation of C_{out} and C_{in} (see Section 3.1 for more details on code concatenation). For our purposes C' will be a binary code. Now symbols in codewords from C' are redistributed in the following manner to obtain codewords in C^*. Let G be a (d_L, d_R)-regular bipartite expander. The symbols from a codeword in C' are blocked into d_L bits (each corresponding to a codeword in C_{in}) and then "placed" on the left vertices of G. These symbols are then "pushed" along the edges in some pre-determined order. For example, the ℓ'th bit in the r'th C_{in} encoding can be sent to the ℓ'th right vertex neighbor of the r'th left vertex. A right vertex then "collects" the bits on its incident edges and then juxtaposes them to form symbols in $\{0,1\}^{d_R}$. These juxtaposed symbols on the right vertices form the codeword in C^*. Note that as G is used to redistribute the symbols, the rates of C^* and C' are the same.

We now briefly discuss how the technique above was used in [Guruswami and Indyk, 2005]. The code C_{in} will be a Reed-Solomon code over constant-sized alphabet. C_{out} will the binary code from Theorem 13, which we will think of as a code over a suitable larger alphabet by simply grouping together bits of appropriate length (recall we did a similar thing with the outer code in Section 3.1.2). Since C_{out} can be encoded in linear-time, C^* can also be encoded in linear time. The decoding algorithm is very natural. First, given the received word $\mathbf{y}$, we "invert" the symbol re-distribution using G in the encoding procedure to get an intermediate received word $\mathbf{y}'$. In the next stage, $\mathbf{y}'$ is then decoded using the natural decoder for the concatenated code C' (as discussed in Section 3.1.2): C_{in} can be decoded using the polynomial time unique decoding algorithm for Reed-Solomon codes while C_{out} can be decoded using the linear-time

decoding algorithm from Theorem 13. Note that the resulting decoding algorithm also runs in linear time. The proof of correctness of this algorithm proceeds by showing that a certain "pseudorandomness" property of G "smoothens" out the errors when passing from $\mathbf{y}$ to $\mathbf{y}'$. More precisely, for most of the inner blocks corresponding to received words for C_{in}, the fraction of errors is roughly the fraction of errors in $\mathbf{y}'$. For each such block, the decoding algorithm for C_{in} corrects all the errors. The few remaining errors are then corrected by the decoding algorithm for C_{out}. Selecting parameters carefully and formalizing the argument above leads to the following result, which answers Question 3 in the affirmative.

Theorem 14. *For every $0 < R < 1$, and all $\varepsilon > 0$, there is an explicit family of codes of rate at least $1 - R - \varepsilon$ over an alphabet of size $2^{O\left(\log(1/\varepsilon)/(\varepsilon^4 R)\right)}$, that can encoded in linear time and decoded from a $(1 - R - \varepsilon)/2$ fraction of errors in linear time.*

3.4 List Decoding

In this section we return to Question 4. The answer to this question is known in the affirmative only for codes over large alphabets. In this subsection we review the sequence of work that has led to this partial positive answer. Reed-Solomon codes will play a crucial role in our discussions.

3.4.1 List Decoding of Reed-Solomon Codes

Consider the $[n, k+1]_q$ Reed-Solomon codes with the set of evaluation points as the non-zero elements of $\mathbb{F}$, which is denoted by $\mathbb{F}_q^* = \{1, \gamma, \gamma^2, \ldots, \gamma^{n-1}\}$ where $n = q - 1$ and γ is the generator of the cyclic group $\mathbb{F}_q^*$. Under list decoding of such a Reed-Solomon code, given the received word $\mathbf{y} = \langle y_0, \ldots, y_{n-1} \rangle$, we are interested in all degree k polynomials $f(X)$ such that for at least $(1 + \delta)\sqrt{R}$ fraction of positions $0 \leqslant i \leqslant n - 1$, $f(\gamma^i) = y_i$. We now sketch the main ideas of the algorithms in [Sudan, 1997; Guruswami and Sudan, 1999]. The algorithms have two main steps: the first is an interpolation step and the second one is a root finding step. In the interpolation step, the list-decoding algorithm finds a bivariate polynomial

$Q(X,Y)$ that fits the input. That is,

> for every position i, $Q(\gamma^i, y_i) = 0$.

Such a polynomial $Q(\cdot,\cdot)$ can be found in polynomial time if we search for one with large enough total degree. This amounts to solving a system of linear equations. After the interpolation step, the root finding step finds all factors of $Q(X,Y)$ of the form $Y - f(X)$. The crux of the analysis is to show that

> for every degree k polynomial $f(X)$ that satisfies $f(\gamma^i) = y_i$ for at least $(1+\delta)\sqrt{R}$ fraction of positions i, $Y - f(X)$ is indeed a factor of $Q(X,Y)$.

However, the above is not true for every bivariate polynomial $Q(X,Y)$ that satisfies $Q(\gamma^i, y_i) = 0$ for all positions i. The main ideas in [Sudan, 1997; Guruswami and Sudan, 1999] were to introduce more constraints on $Q(X,Y)$. In particular, [Sudan, 1997] added the constraint that a certain weighted degree of $Q(X,Y)$ is below a fixed upper bound. Specifically, $Q(X,Y)$ was restricted to have a non-trivially bounded $(1,k)$-weighted degree. The $(1,k)$-weighted degree of a monomial X^iY^j is $i + jk$ and the $(1,k)$-weighted degree of a bivariate polynomial $Q(X,Y)$ is the maximum $(1,k)$-weighted degree among its monomials. The intuition behind defining such a weighted degree is that given $Q(X,Y)$ with weighted $(1,k)$ of D, the univariate polynomial $Q(X, f(X))$, where $f(X)$ is some degree k polynomial, has total degree at most D. The upper bound D is chosen carefully such that if $f(X)$ is a codeword that needs to be output, then $Q(X, f(X))$ has more than D zeroes and thus $Q(X, f(X)) \equiv 0$, which in turn implies that $Y - f(X)$ divides $Q(X,Y)$. To get to the bound of $1 - (1+\delta)\sqrt{R}$, [Guruswami and Sudan, 1999] added a further constraint on $Q(X,Y)$ that required it to have r roots at (γ^i, y_i), where r is some parameter (in [Sudan, 1997] $r = 1$ while in [Guruswami and Sudan, 1999], r is roughly $1/\delta$). Choosing parameters carefully leads to the following result.

Theorem 15. *Let $0 < R < 1$. Then any Reed-Solomon code of rate at least R can be list-decoded from a $1 - \sqrt{R}$ fraction of errors.*

We note that the above result holds for any Reed-Solomon code and not just the code where the set of evaluation points is $\mathbb{F}_q^*$.

3.4.2 List Decoding of Reed-Solomon Like Codes

We now discuss the recent developments in [Parvaresh and Vardy, 2005; Guruswami and Rudra, 2006]. These consider variants of Reed-Solomon codes that are no longer linear. The codes considered in [Guruswami and Rudra, 2006] are a strict subset of those considered in [Parvaresh and Vardy, 2005]. For the ease of presentation, we will present the ideas of [Parvaresh and Vardy, 2005] using these smaller subset of codes.

Folded Reed-Solomon code with "folding parameter" $m \geqslant 1$, is exactly the Reed-Solomon code considered in Section 3.4.1, but viewed as a code over $(\mathbb{F}_q)^m$ by bundling together m consecutive symbols of codewords in the Reed-Solomon code. For example with $m = 2$ (and n even), the Reed-Solomon codeword $\langle f(1), f(\gamma), f(\gamma^2), f(\gamma^3), \ldots, f(\gamma^{n-2}), f(\gamma^{n-1})\rangle$ will correspond to the following codeword in the folded Reed-Solomon code: $\langle (f(1), f(\gamma)), \left(f(\gamma^2), f(\gamma^3)\right), \ldots, \left(f(\gamma^{n-2}), f(\gamma^{n-1})\right)\rangle$. We will now briefly present the ideas which can be used to show that folded Reed-Solomon codes with folding parameter m can be list decoded up to a $1 - (1+\delta)\left(\frac{m}{m-s+1}\right) R^{s/(s+1)}$ fraction of errors for any $1 \leqslant s \leqslant m$. Note that Theorem 15 handles the $m = s = 1$ case.

We now consider the next non-trivial case of $m = s = 2$. The ideas for this case can be easily extended to the general $m = s$ case. Note that now given the received word $\langle (y_0, y_1), (y_2, y_3), \ldots, (y_{n-2}, y_{n-1})\rangle$ we want to find all degree k polynomials $f(X)$ such that for at least $2(1+\delta)\sqrt[3]{R^2}$ fraction of positions $0 \leqslant i \leqslant n/2 - 1$, $f(\gamma^{2i}) = y_{2i}$ and $f(\gamma^{2i+1}) = y_{2i+1}$. As in the Reed-Solomon case, we will have an interpolation and a root finding step. The interpolation step is a straightforward generalization of the Reed-Solomon case: we find a trivariate polynomial $Q(X, Y, Z)$ that fits the received word, that is, for every $0 \leqslant i \leqslant n/2 - 1$, $Q(\gamma^{2i}, y_{2i}, y_{2i+1}) = 0$. Further, $Q(X, Y, Z)$ has an upper bound on its $(1, k, k)$-weighted

degree (which is a straightforward generalization of the $(1, k)$-weighted degree for the bivariate case) and has a multiplicity of r at every point. For the root finding step, it suffices to show that for every degree k polynomial $f(X)$ that needs to be output, $Q(X, f(X), f(\gamma X)) \equiv 0$. This, however does not follow from weighted degree and multiple root properties of $Q(X, Y, Z)$. Here we will need two new ideas, the first of which (due to [Guruswami and Rudra, 2006]) is to show that for some irreducible polynomial $E(X)$ of degree $q - 1$, $f(X)^q \equiv f(\gamma X) \mod (E(X))$.[4] The second idea, due to [Parvaresh and Vardy, 2005], is the following. We first obtain the bivariate polynomial (over an appropriate extension field) $T(Y, Z) \equiv Q(X, Y, Z) \mod (E(X))$. Note that by the first idea, we are looking for solutions on the curve $Z = Y^q$ (Y corresponds to $f(X)$ and Z corresponds to $f(\gamma X)$ in the extension field). The crux of the argument is to show that all the polynomials $f(X)$ that need to be output correspond to (in the extension field) some root of the equation $T(Y, Y^q) = 0$.

To go from $s = m$ to any $s \leqslant m$ requires the following idea due to [Guruswami and Rudra, 2006]: We will reduce the problem of list decoding folded Reed-Solomon code with folding parameter m to the problem of list decoding folded Reed-Solomon code with folding parameter s. We then use the algorithm outlined in the previous paragraph for the folded Reed-Solomon code with folding parameter s. A careful tracking of the agreement parameter in the reduction brings down the final agreement fraction that is required for the original folded Reed-Solomon code with folding parameter m from $m(1 + \delta) \sqrt[m+1]{R^m}$ (which can be obtained without the reduction) to $(1 + \delta) \left(\frac{m}{m-s+1}\right) \sqrt[s+1]{R^s}$. Choosing parameters carefully leads to the following result:

Theorem 16. *For every $0 < R < 1$ and $0 < \varepsilon \leqslant R$, there is a family of folded Reed-Solomon codes that have rate at least R and which can be list decoded up to a $1 - R - \varepsilon$ fraction of errors in time (and outputs a list of size at most) $(N/\varepsilon^2)^{O(\varepsilon^{-1} \log(1/R))}$ where N is the block length of the code. The alphabet size of the code as a function of the block length N is $(N/\varepsilon^2)^{O(1/\varepsilon^2)}$.*

[4] This idea shows that folded Reed-Solomon codes are special cases of the codes considered in [Parvaresh and Vardy, 2005].

One drawback of the above result is that the alphabet size of the code increases with the block length. However, it is shown in [Guruswami and Rudra, 2006] that the code concatenation and expander-based techniques from Section 3.3.3 can be used to obtain the following result (which resolves Question 4 for large enough alphabets).

Theorem 17. *For every R, $0 < R < 1$, every $\varepsilon > 0$, there is a polynomial time constructible family of codes over an alphabet of size $2^{O(\varepsilon^{-4}\log(1/\varepsilon))}$ that have rate at least R and which can be list decoded up to a fraction $(1 - R - \varepsilon)$ of errors in polynomial time.*

We remark that the best known explicit construction of codes with list decoding algorithms for binary codes use code concatenation (Section 3.1). Expander codes (Section 3.3) have been used to obtain explicit codes with linear time list decoding algorithms.

4 Summary and Research Issues

The goal of algorithmic coding theory is to design (explicit) codes along with efficient encoding and decoding algorithms such that the best possible combinatorial tradeoff between the rate of the code and the fraction of errors that can be corrected is achieved. (This tradeoff is generally captured by the notion of the capacity of a channel.) This is generally achieved via two steps. First, the combinatorial tradeoff is established. Generally random codes achieve this tradeoff. The next and more challenging step is to achieve the capacity of the channel (possibly with explicit codes and) with efficient encoding and decoding algorithms. In the first part of this chapter, we presented results on capacity of some classical channels. In the second part, we presented mostly recent work on the progress towards capacity-achieving explicit codes along with efficient encoding and decoding algorithms.

Almost sixty years since the birth of coding theory, codes over large alphabets are better understood. However, challenging open questions related to such codes still remain. For example, as we saw in Sec-

tion 3.4, Question 4 has been answered for codes over large alphabets. Although these codes do achieve the best possible tradeoff between rate and fraction of errors, the guarantee on the worst-case list size if far from satisfactory. In particular, to get within ε of capacity the worst case size guaranteed Theorem 17 grows as $n^{1/\varepsilon}$. This should be compared to the worst case list size of $O(1/\varepsilon)$ achieved by random codes (Theorem 7).

As was alluded to earlier, most of the outstanding open questions in algorithmic coding theory relate to codes over small or fixed alphabets, in particular binary codes. For the stochastic noise models, getting to within ε of capacity (for example the BSC) with encoding and decoding time complexities that have polynomial dependence on $1/\varepsilon$ and linear dependence on the block length is an important challenge. One promising avenue is to prove that irregular LDPC codes from Section 3.2.4 meet this challenge. Experimental results suggest that this is true.

One of the biggest open questions in coding theory is to design explicit binary codes that have the same rate vs. distance tradeoff as random binary codes, i.e., meet the Gilbert-Varshamov (GV) bound (Theorem 6). In fact achieving this bound of any constant rate is wide open. Another classical open question in this vein is to determine the optimal tradeoff between rate and distance for binary codes: the best lower bound on rate for given distance bound is the GV bound while the best upper bound is achieved by the so called MRRW bound [McEliece et al., 1977]. There is a gap between the two bounds. Closing the gap between these two bounds remains an important open problem. Another challenge is to achieve a positive resolution of Question 4 for codes over fixed alphabets in general and binary codes in particular. One of the obstacles in meeting challenges related to binary codes is that not many explicit constructions of such codes with some constant rate and constant relative distance are known. In fact, all known constructions of such codes either use code concatenation or expander graphs.

5 Defining Terms

Code: A collection of vectors of the same length defined over a fixed set.

Codeword: A vector in a code.

Alphabet: The set of symbols over which codewords are defined.

Received word: The corrupted codeword obtained after transmission.

Encoding: Function that converts the original message to a codeword.

Decoding: Function that when given the received word outputs what it thinks was the transmitted message.

It can also report a failure.

Binary codes: Codes defined over an alphabet of size two.

Block length: Length of the codewords in a code.

Dimension: The number $\log_q(|C|)$, where the code C is defined over an alphabet of size q.

Rate: Ratio of the dimension and the block length of a code.

Hamming distance: Number of positions in which two vectors of the same length differ.

Minimum distance: The minimum Hamming distance between any two distinct codewords in a code.

Linear code: A code over $\mathbb{F}_q^n$ and blocklength n that is a linear subspace of $\mathbb{F}_q^n$.

Capacity: Threshold on the rate of a code for which reliable communication is possible on a channel.

References

Alon, N., Edmonds, J., and Luby, M. (1995). Linear time erasure codes with nearly optimal recovery (extended abstract). In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 512–519.

Alon, N. and Spencer, J. (1992). *The Probabilistic Method*. John Wiley and Sons, Inc.

Berlekamp, E. R., McEliece, R. J., and van Tilborg, H. C. A. (1978). On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24:384–386.

Capalbo, M. R., Reingold, O., Vadhan, S. P., and Wigderson, A. (2002). Randomness conductors and constant-degree lossless expanders. In *Proceedings of the 34th annual ACM symposium on Theory of computing (STOC)*, pages 659–668.

Chandler, D. G., Batterman, E. P., and Shah, G. (1989). Hexagonal, information encoding article, process and system. *US Patent Number 4,874,936*.

Chen, C. L. and Hsiao, M. Y. (1984). Error-correcting codes for semiconductor memory applications: A state-of-the-art review. *IBM Journal of Research and Development*, 28(2):124–134.

Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., and Patterson, D. A. (1994). RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185.

Dumer, I. I. (1998). Concatenated codes and their multilevel generalizations. In Pless, V. S. and Huffman, W. C., editors, *Handbook of Coding Theory*, volume 2, pages 1911–1988. North Holland.

Elias, P. (1957). List decoding for noisy channels. *Technical Report 335, Research Laboratory of Electronics, MIT*.

Elias, P. (1991). Error-correcting codes for list decoding. *IEEE Transactions on Information Theory*, 37:5–12.

Forney, G. D. (1966). *Concatenated Codes*. MIT Press, Cambridge, MA.

Gallager, R. G. (1963). *Low-Density Parity-Check Codes*. MIT Press, Cambridge.

Goldreich, O. and Levin, L. (1989). A hard-core predicate for all one-way functions. *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 25–32.

Guruswami, V. (2004a). Error-correcting codes and expander graphs. *SIGACT News*, pages 25–41.

Guruswami, V. (2004b). *List decoding of error-correcting codes*. Number 3282 in Lecture Notes in Computer Science. Springer. (Winning Thesis of the 2002 ACM Doctoral Dissertation Competition).

Guruswami, V. (2006a). Algorithmic results in list decoding. In *Foundations and Trends in Theoretical Computer Science (FnT-TCS)*, volume 2. NOW publishers.

Guruswami, V. (2006b). Iterative decoding of low-density parity check codes. *Bulletin of the EATCS*, 90:53–88.

Guruswami, V. (2006c). List decoding in pseudorandomness and average-case complexity. In *IEEE Information Theory Workshop*.

Guruswami, V. and Indyk, P. (2005). Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400.

Guruswami, V. and Rudra, A. (2006). Explicit capacity-achieving list-decodable codes. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10.

Guruswami, V. and Sudan, M. (1999). Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767.

Hamming, R. W. (1950). Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29:147–160.

Høholdt, T., van Lint, J. H., and Pellikaan, R. (1998). Algebraic geometry codes. In V. S. Pless, W. C. H. and A.Brualdi, R., editors, *Handbook of Coding Theory*. North Holland.

Justesen, J. (1972). A class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18:652–656.

Luby, M., Mitzenmacher, M., Shokrollahi, M. A., and Spielman, D. A. (2001a). Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584.

Luby, M., Mitzenmacher, M., Shokrollahi, M. A., and Spielman, D. A. (2001b). Improved low-density parity-check codes using irregular graphs. *IEEE Transactions on Information Theory*, 47(2):585–598.

MacWilliams, F. J. and Sloane, N. J. A. (1981). *The Theory of Error-Correcting Codes*. Elsevier/North-Holland, Amsterdam.

McEliece, R. J., Rodemich, E. R., Rumsey Jr., H., and Welch, L. R. (1977). New upper bounds on the rate of a code via the Delsarte-Macwilliams inequalities. *IEEE Transactions on Information Theory*, 23:157–166.

Parvaresh, F. and Vardy, A. (2005). Correcting errors beyond the Guruswami-Sudan radius in polynomial time. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 285–294.

Peterson, L. L. and Davis, B. S. (1996). *Computer Networks: A Systems Approach*. Morgan Kaufmann Publishers, San Francisco.

Pless, V. S. and Huffman, W. C., editors (1998). *Handbook of Coding Theory*. North Holland.

Reed, I. S. and Solomon, G. (1960). Polynomial codes over certain finite fields. *SIAM Journal on Applied Mathematics*, 8:300–304.

Richardson, T. and Urbanke, R. (2007). *Modern Coding Theory*. `http://lthcwww.epfl.ch/mct/index.php`.

Richardson, T. J., Shokrollahi, M. A., and Urbanke, R. L. (2001). Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):619–637.

Richardson, T. J. and Urbanke, R. L. (2001a). The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2):599–618.

Richardson, T. J. and Urbanke, R. L. (2001b). Efficient encoding of low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):638–656.

Rudra, A. (2007). *List Decoding and Property Testing of Error Correcting Codes*. PhD thesis, University of Washington.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656.

Sipser, M. and Spielman, D. (1996). Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722.

Spielman, D. (1996). Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1732.

Sudan, M. (1997). Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193.

Sudan, M. (2000). List decoding: Algorithms and applications. *SIGACT News*, 31:16–27.

Sudan, M. (2001). Lecture notes on algorithmic introduction to coding theory.

Trevisan, L. (2004). Some applications of coding theory in computational complexity. *Quaderni di Matematica*, 13:347–424.

van Lint, J. H. (1999). *Introduction to Coding Theory*. Graduate Texts in Mathematics **86**, (Third Edition) Springer-Verlag, Berlin.

Wicker, S. B. and Bhargava, V. K., editors (1999). *Reed-Solomon Codes and Their Applications*. John Wiley and Sons, Inc.

Wozencraft, J. M. (1958). List Decoding. *Quarterly Progress Report, Research Laboratory of Electronics, MIT*, 48:90–95.

Zyablov, V. V. and Pinsker, M. S. (1981 (in Russian); pp. 236-240 (in English), 1982). List cascade decoding. *Problems of Information Transmission*, 17(4):29–34.

6 Further Information

Given the space limitations of this chapter, many important research themes in algorithmic coding theory have regretfully been omitted. Results from classical work in coding theory can be found in the standard coding textbooks such as [MacWilliams and Sloane, 1981; van Lint, 1999]. Another great resource for such results are the two volumes of the Handbook of Coding theory [Pless and Huffman, 1998]. An excellent source for most of the material covered in this chapter is Sudan's notes from his course on Coding Theory [Sudan, 2001]. A conspicuous absence from this chapter is any discussion of algebraic geometric codes (cf. [Høholdt et al., 1998]). These are codes based on some deep mathematics involving function fields that can be thought of as a generalization of Reed-Solomon codes. These techniques give codes with excellent rate vs. distance properties over small alphabets. For alphabets of size greater than 49 these codes beat the Gilbert-Varshamov bound.

For more details on code concatenation, the reader can consult any of the references above (there is a dedicated chapter in the handbook of coding theory [Dumer, 1998]). Guruswami's introductory survey on LDPC codes [Guruswami, 2006b] is a good place to get more details regarding the material presented in Section 3.2. The upcoming book [Richardson and Urbanke, 2007] has a more comprehensive treatment. Another valuable resource is the February 2001 issue of Volume 47 of the journal IEEE Transactions on

Information Theory: this was a special issue dedicated to iterative decoding and specifically contains the sequence of papers [Luby et al., 2001a,b; Richardson and Urbanke, 2001a; Richardson et al., 2001; Richardson and Urbanke, 2001b]. This series of papers perhaps constituted the most important post-Gallager work on LDPC codes and laid the foundation for the recent spurt in research activity in LDPC codes. Guruswami's survey [Guruswami, 2004a] is a good starting point for more details on expander codes. The material presented in Section 3.3 appeared in [Sipser and Spielman, 1996; Spielman, 1996; Alon et al., 1995; Guruswami and Indyk, 2005]. [Sudan, 2000] is a nice introductory survey on list decoding and its applications in complexity theory. The authoritative text on list decoding is [Guruswami, 2004b]. For more details on the recent developments see the survey [Guruswami, 2006a] or the author's thesis [Rudra, 2007].

As was mentioned in the introduction, codes have found numerous applications in theoretical science. The reader is referred to these surveys (and the references within) for more details: [Trevisan, 2004; Sudan, 2000; Guruswami, 2006c, 2004a].

Many, if not most, papers on algorithmic coding theory are published in the IEEE Transactions on Information Theory. A number of research papers are presented at the annual International Symposium on Information Theory (ISIT). Papers are also presented at the IEEE Information Theory workshop and the Allerton Conference on Communication, Control and Computing.