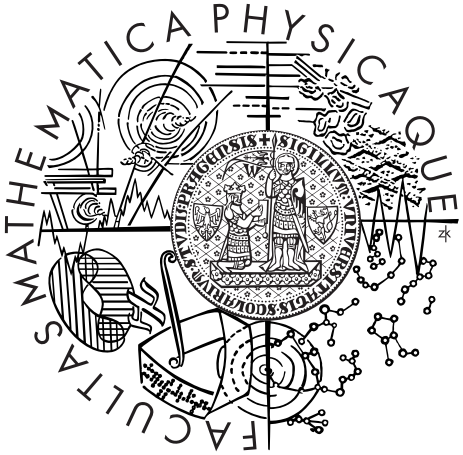
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Andrew Kozlík

Coding and effectivity of LDPC codes

Department of Algebra

Supervisor of the thesis: prof. RNDr. Aleš Drápal CSc., DSc.

Study programme: Mathematics

Specialization: Mathematical Methods of Information Security

Prague 2011

The author would like to thank Prof. Aleš Drápal for his advice in supervising this thesis.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

Prague, 4. August 2011 Andrew Kozlík

Název práce: Kódování a efektivita LDPC kódů

Autor: Andrew Kozlík

Katedra: Katedra algebry

Vedoucí diplomové práce: prof. RNDr. Aleš Drápal CSc., DSc., Katedra algebry

Abstrakt: LDPC kódy jsou lineární samoopravné kódy, které jednak umožňují přenos dat rychlostí libovolně blízkou kapacitě kanálu, a zároveň pro ně existují vysoce účinné dekódovací algoritmy. Naproti tomu hlavní nevýhodou většiny LDPC kódů je vysoká časová náročnost jejich kódovacího algoritmu. V této práci se nejdříve věnujeme podrobnému rozboru tzv. sum-product dekódovacího algoritmu. Následně zkoumáme výkonnost LDPC kódů na binárním vymazávacím kanálu za použití sum-product algoritmu, čímž získáme kritéria pro design kódů, které umožňují spolehlivý přenos dat rychlostí libovolně blízkou kapacitě kanálu. Na základě těchto kritérií ukážeme, jak probíhá design takovýchto kódů. Poté prezentujeme experimentálně získané výsledky a srovnáváme je s teoretickými odhady. Na závěr poskytneme přehled několika způsobů, kterými lze řešit problém vysoké časové náročnosti kódování.

Klíčová slova: Binární vymazávací kanál, kódování, LDPC kódy.

Title: Coding and effectivity of LDPC codes

Author: Andrew Kozlík

Department: Department of Algebra

Supervisor: prof. RNDr. Aleš Drápal CSc., DSc., Department of Algebra

Abstract: Low-density parity-check (LDPC) codes are linear error correcting codes which are capable of performing near channel capacity. Furthermore, they admit efficient decoding algorithms that provide near optimum performance. Their main disadvantage is that most LDPC codes have relatively complex encoders. In this thesis, we begin by giving a detailed discussion of the sum-product decoding algorithm, we then study the performance of LDPC codes on the binary erasure channel under sum-product decoding to obtain criteria for the design of codes that allow reliable transmission at rates arbitrarily close to channel capacity. Using these criteria we show how such codes are designed. We then present experimental results and compare them with theoretical predictions. Finally, we provide an overview of several approaches to solving the complex encoder problem.

Keywords: Binary erasure channel, coding, low-density parity-check codes.

Contents

Chapter 1

Introduction

Low-density parity-check (LDPC) codes are linear error correcting codes which are capable of performing near channel capacity. Furthermore, they admit efficient decoding algorithms that provide near optimum performance. Their main disadvantage is that most LDPC codes have relatively complex encoders.

LDPC codes were first described by Robert G. Gallager in his doctoral dissertation, completed in 1960 at MIT. Gallager came up with an iterative decoding algorithm designed for linear codes with a sparse parity-check matrix. At the time, however, computers lacked the computational power necessary for a practical implementation of the algorithm he proposed and so LDPC codes were largely forgotten. Then in 1993 iterative decoding was rediscovered in connection with turbo codes, and the independent rediscovery of LDPC codes soon followed.

Today, LDPC codes are being integrated into standards for both wired and wireless data transmissions, and a variety of other applications are being explored.

For example 10GBASE-T Ethernet which is designed to provide 10 Gb/s performance over twisted-pair wiring uses a (6,32)-regular LDPC code of length 2048 and dimension 1723. [17]

The 802.11n Wi-Fi standard allows for the optional use of LDPC codes as a high-performance alternative to the default binary convolutional code. The standard defines 12 different systematic LDPC codes of length 648, 1296 and 1944 bits and rate 1/2, rate 2/3, rate 3/4, and rate 5/6. [18]

The 2nd generation Digital Video Broadcasting standards DVB-C2 (cable), DVB-S2 (satellite) and DVB-T2 (terrestrial) use a concatenation of BCH with LDPC inner coding whereas the first generation standards used Reed-Solomon coding. [19]

A study by Yang and Ryan has shown some promising initial results regarding the application of LDPC codes in magnetic storage systems. The codes have proven robust against large noise bursts, which may be present due to media defects or thermal asperities. [15]

NASA researchers have also designed LDPC codes that fit the specific needs of spacecraft equipment. These codes are being considered for both near-earth and deep space applications. [16]

1.1 Preliminaries

We start with an example of two communication channels:

Definition 1.1. By X_t and Y_t we denote the random variables that are equal to the channel input and output at time t, respectively. The time t is discrete and the transmitter and receiver are synchronised.

By BSC(ε) we denote the *binary symmetric channel* with cross-over probability ε. The input and output alphabet is $\{0, 1\}$. Each transmitted bit is either received correctly with probability $\Pr(Y_t = X_t) = 1-\varepsilon$ or flipped with probability $\Pr(Y_t \neq X_t) = \varepsilon$.

By BEC(ε) we denote the *binary erasure channel* with erasure probability ε. This channel models a situation, where transmitted bits are either received correctly or are known to be lost. The input alphabet is $\{0, 1\}$ and the output alphabet is $\{0, 1, ?\}$, where ? indicates an erasure. Each bit is either received correctly with probability $\Pr(Y_t = X_t) = 1 - \varepsilon$ or erased with probability $\Pr(Y_t = ?) = \varepsilon$.

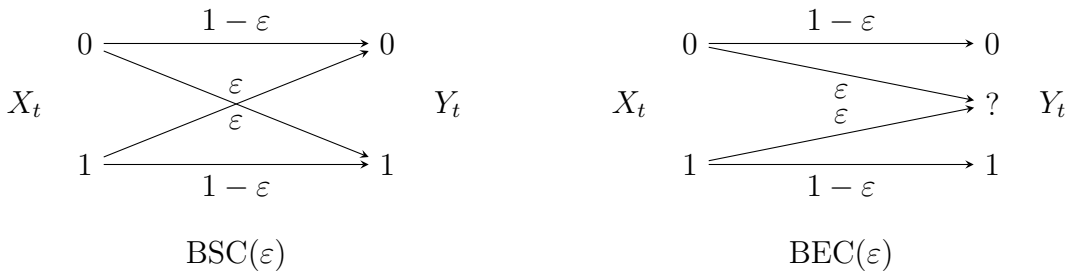


FIGURE 1.1: The binary symmetric channel and the binary erasure channel.

Although the BEC might seem like an overly simple and unrealistic model of a communication channel, there do exist real world examples. The Internet Protocol (IP) used for relaying datagrams over computer networks is one such example. IP datagrams may arrive corrupted, which is detectable by means of a checksum, or they may be lost altogether. In both of these cases we would consider the information bits carried by such datagrams erased. If on the other hand the datagrams arrive with a valid checksum, then we may consider them error free. If *sequentially numbered* datagrams of *fixed size* are transmitted over IP, then the receiver can detect not only corrupted datagrams, but also lost datagrams, and therefore knows exactly which bits were erased.

The study of LDPC codes on the BEC provides one of the few scenarios where exact analysis is possible. Quite surprisingly, however, the BEC can give us a good understanding of what happens in more general cases, because most properties and statements that we encounter in the investigation of the BEC hold in much greater generality. [13]

Definition 1.2. For a given channel, let $X = (X_1, \ldots, X_n)$ denote the random variable that is equal to the channel input and let $Y = (Y_1, \ldots, Y_n)$ denote random variable that is equal to the channel output. The channel is said to be *memoryless*

if

$$\Pr(Y = y \mid X = x) = \prod_{i=1}^{n} \Pr(Y_i = y_i | X_i = x_i),$$

where $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$.

Both the BSC and the BEC are memoryless channels since the cross-overs and erasures occur independently for each t.

Definition 1.3. A *linear error-correcting code* of *length* n and *dimension* k over a finite field $\mathbb{F}_q$ is a k-dimensional subspace $\mathcal{C}$ of $\mathbb{F}_q^n$. The code $\mathcal{C}$ is referred to as an $[n, k]_q$ code, and its elements are called *codewords*. Any $k \times n$ matrix G, whose rows form a basis of $\mathcal{C}$ is called a *generator matrix* of $\mathcal{C}$. If the generator matrix is of the form $G = (I_k|P)$, where I_k is the $k \times k$ identity matrix and P is a $k \times (n-k)$ matrix, then we say that G is in *systematic form*.

To each code $\mathcal{C}$ we associate a *dual* code $\mathcal{C}^\perp$, defined as

$$\mathcal{C}^\perp = \{\, v \in \mathbb{F}_q^n \mid xv^{\mathrm{T}} = 0, \forall x \in \mathcal{C} \,\} = \{\, v \in \mathbb{F}_q^n \mid Gv^{\mathrm{T}} = 0^{\mathrm{T}} \,\},$$

i.e. the dual code is the set of solutions to the system of equations $Gv^{\mathrm{T}} = 0^{\mathrm{T}}$. Since the k rows of G are linearly independent, $\mathcal{C}^\perp$ is an $n-k$ dimensional subspace of $\mathbb{F}_q^n$. Any $(n-k) \times n$ generator matrix of the code $\mathcal{C}^\perp$ is called a *parity-check matrix* of the original code $\mathcal{C}$, and is denoted H. By this definition, the rows of the parity-check matrix are linearly independent. In this thesis, we broaden the definition of the parity-check matrix to include all matrices whose row vectors generate the dual code, regardless of whether the rows are linearly independent or not. Therefore, if H is an $m \times n$ parity-check matrix of $\mathcal{C}$ in the sense of the broader definition, then $m \geq n - k$.

The code $\mathcal{C}$ can now be expressed as

$$\mathcal{C} = \{\, x \in \mathbb{F}_q^n \mid Hx^{\mathrm{T}} = 0^{\mathrm{T}} \,\}.$$

Consider the following problem: We wish to transmit a *message* across a noisy channel, so that the receiver can determine the message with high probability despite the imperfections of the channel. The solution is to add redundancy by mapping the messages to codewords, and transmitting these instead. The operation by which messages are mapped to codewords is termed *encoding*. Perhaps the most straightforward method of encoding a message u to a codeword x of $\mathcal{C}$ is to represent the message as a vector of length k over $\mathbb{F}_q$ and then compute $x = uG$, where G is a generator matrix of $\mathcal{C}$. If G is in systematic form, then the first k symbols of x will be the message symbols. The remaining $n-k$ symbols of x are then called *parity symbols*.

Whenever a codeword is transmitted over a noisy channel, the output of the channel shall be referred to simply as a *word*. The received word is used to estimate which codeword was transmitted over the channel. This operation is termed *decoding*. Finally, from the decoded codeword a message is retrieved, hopefully the original one. If the original message was encoded using a generator matrix in systematic form, then the retrieval is done simply by discarding the parity symbols of the codeword.

In Chapter 4 we will discuss time complexity of encoding algorithms in the length of the code. When we say that an algorithm runs in time $O(f(n))$, we mean that it can be easily implemented to run in time $O(f(n))$ on a random access machine in the uniform cost model.

The *rate* of an $[n, k]_q$ code is defined as k/n. This value gives a measure of the amount of non-redundant information contained in each codeword. Shannon's channel coding theorem asserts the existence of a maximum rate, at which information can be transmitted reliably with vanishing probability of error, provided that the encoder and decoder are allowed to operate on long enough sequences of data. This maximum rate is called the *capacity* of the channel and is denoted C.

The capacity of the BEC is $1 - \varepsilon$. It is easy to see that $C_{\text{BEC}}(\varepsilon) \leq 1 - \varepsilon$: suppose that the transmitter knew in advance which bits will be erased. This additional information can only increase the achievable rate. On average it would be possible to use $(1-\varepsilon)n$ bits of the n transmitted. The best that the transmitter could do is fill these $(1-\varepsilon)n$ bits with information. The receiver would then simply ignore the erased bits. Thus, even if the transmitter knows in advance which bits will be erased, information can be transmitted reliably at a rate of at most $1 - \varepsilon$. Reliable transmission over the BEC at a rate arbitrarily close to $1 - \varepsilon$ is possible even without this knowledge (see Example 3.6 [13]).

The capacity of the BSC is $1 + \varepsilon \log_2(\varepsilon) + (1 - \varepsilon) \log_2(1 - \varepsilon)$ (see Theorem 1.17 [13]).

The *Hamming weight* of $u \in \mathbb{F}_q^n$, denoted $w(u)$, is defined as the number of nonzero symbols in u. The *minimum weight* of a code is defined as

$$w(\mathcal{C}) = \min\{\, w(u) \mid u \in \mathcal{C}, c \neq 0 \,\}.$$

The *Hamming distance* of $u, v \in \mathbb{F}_q^n$, denoted $d(u, v)$, is the number of positions, where u differs from v. The *minimum distance* of a code $\mathcal{C}$ is defined as

$$d(\mathcal{C}) = \min\{\, d(u, v) \mid u, v \in \mathcal{C}, u \neq v \,\}.$$

For linear codes, $w(\mathcal{C}) = d(\mathcal{C})$. This can be seen as follows. Let u be a nonzero minimum weight codeword of $\mathcal{C}$, then $w(\mathcal{C}) = w(u) = d(0, u) \geq d(\mathcal{C})$. Now let $u \neq v$ be two closest codewords of $\mathcal{C}$, then $u - v$ is a codeword as well, and $d(\mathcal{C}) = d(u, v) = w(u - v) \geq w(\mathcal{C})$.

A code $\mathcal{C}$ can correct any $\lfloor (d(\mathcal{C}) - 1)/2 \rfloor$ errors introduced into a codeword. The minimum distance of a code thus gives a measure of its error-correcting ability. Codes that contain low-weight codewords have low minimum distance and hence low error-correcting ability. The presence of low-weight codewords is therefore undesirable.

1.2 Thesis outline

The thesis is outlined as follows. In Chapter 2 the concept of maximum a-posteriori decoding and maximum likelihood decoding is presented, followed by a detailed description of the sum-product decoding algorithm. Some practical

implementation issues of the algorithm are addressed. Finally, codes with cycle-free Tanner graphs, though they are in a sense optimal for use with the sum-product algorithm, are shown to have severe shortcomings when it comes to their minimum distance.

Chapter 3 provides a definition of LDPC codes and a discussion of certain structural properties that they possess. The performance of LDPC codes on the BEC is analyzed to obtain criteria for the design of good LDPC codes and to obtain theoretical estimates of their error-correcting ability. The design of codes that allow reliable transmission at rates arbitrarily close to channel capacity on the BEC is described. Experimental results are presented and compared with theoretical predictions.

In Chapter 4 various approaches to solving the complex encoder problem are presented.

Chapter 2

Iterative decoding on graphs

2.1 Tanner graph representation

Let H be an $m \times n$ parity-check matrix of a linear code $\mathcal{C}$. The rows of the parity-check matrix need not be linearly independent. We shall, however, assume throughout this thesis that every row of the parity-check matrix has at least two nonzero entries. A single nonzero entry in one of the rows would mean that all codewords would have a zero at the corresponding position, which serves no purpose.

The code $\mathcal{C}$ is the set of n-tuples that satisfy the m constraint equations

$$Hy^{\mathrm{T}} = 0^{\mathrm{T}}.$$

Using these equations, we may represent a binary linear code as a bipartite graph.

Definition 2.1. The *Tanner graph* associated with the parity-check matrix H for the binary code $\mathcal{C}$ is a bipartite graph with n *variable nodes*, corresponding to the bits of a codeword, and m *check nodes* corresponding to the constraint equations. Check node i is connected to variable node j if and only if $H_{ij} = 1$.

Note that by the assumption above, the Tanner graph has no leaf check nodes or isolated check nodes.

Example 2.2. Consider the [7,4,3] Hamming code with parity-check matrix

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

$x = (x_1, \ldots, x_7) \in \mathbb{F}_2^7$ is a codeword of $\mathcal{C}$ if and only if the following parity-check constraints are satisfied

$$\begin{aligned} x_1 + x_2 + x_4 + x_5 &= 0, \\ x_1 + x_3 + x_4 + x_6 &= 0, \\ x_2 + x_3 + x_4 + x_7 &= 0. \end{aligned} \tag{2.1}$$

Figure 2.1 shows the associated Tanner graph.

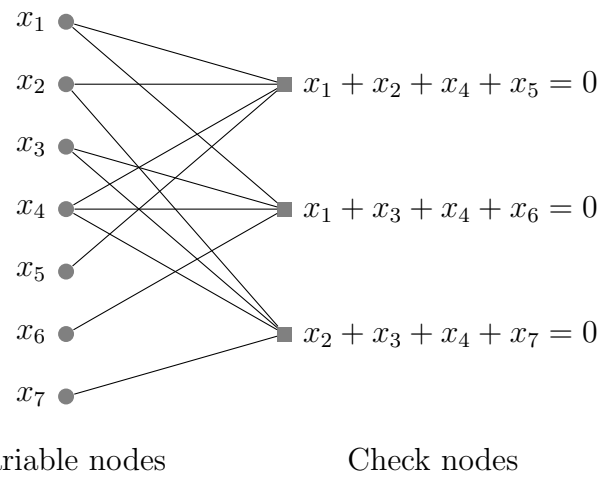


FIGURE 2.1: Tanner graph of the [7,4,3] Hamming code.

The Tanner graph is helpful in visualizing iterative decoding algorithms. Given the output of a BEC, the simplest method of correcting erasures is using a greedy algorithm:

1. Find a check node that is joined to exactly one erased variable node.
2. Calculate the sum of the unerased variable nodes that are joined to that check node and assign the result to the erased variable.
3. Repeat steps 1. and 2. until no check nodes with exactly one erased variable node remain.

Example 2.3. Let $y = (1, 0, ?, ?, 0, 1, ?)$ be the output of a transmission over a BEC using the [7,4,3] Hamming code from the previous example. Figure 2.2 illustrates the state at the beginning of every cycle of the greedy algorithm. The edges that are incident to an erased variable node are dashed, those that are incident to a variable node with value 0 are solid, and those that are incident to a variable node with value 1 are snaked. The algorithm proceeds as follows:

(i) Only the top check node is joined to exactly one erased variable x_4. Calculate $x_1 + x_2 + x_5 = 1$ and assign the result to x_4.

(ii) Now only the middle check node is joined to exactly one erased variable x_3. Calculate $x_1 + x_4 + x_6 = 1$ and assign the result to x_3.

(iii) Now only the bottom check node is joined to exactly one erased variable x_7. Calculate $x_2 + x_3 + x_4 = 0$ and assign the result to x_7.

We have successfully recovered the codeword $(1, 0, 1, 1, 0, 1, 0)$.

Example 2.4. Let $x = (?, 0, ?, ?, 0, 1, 0)$ be the output of a transmission over a BEC using the [7,4,3] Hamming code. Figure 2.3 illustrates the state at the beginning of the greedy algorithm. This time, however, none of the check nodes is joined to exactly one erased variable node. We are unable to recover the codeword

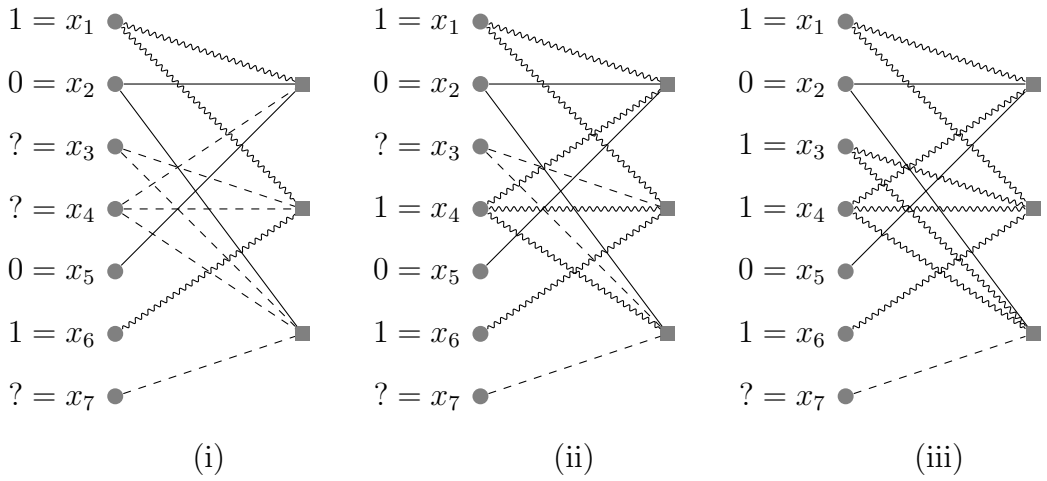


FIGURE 2.2: Cycles of the greedy algorithm for $x = (?, 0, ?, ?, 0, 1, 0)$.

using the greedy algorithm. Nevertheless it is possible to recover the codeword by solving a system of linear equations. Substituting the values of the unerased variables into (2.1) yields

$$\begin{aligned} x_1 + x_4 &= 0, \\ x_1 + x_3 + x_4 + 1 &= 0, \\ x_3 + x_4 &= 0. \end{aligned}$$

The solution is $x_1 = x_4 = x_3 = 1$, so once again we obtain the codeword $(1, 0, 1, 1, 0, 1, 0)$.

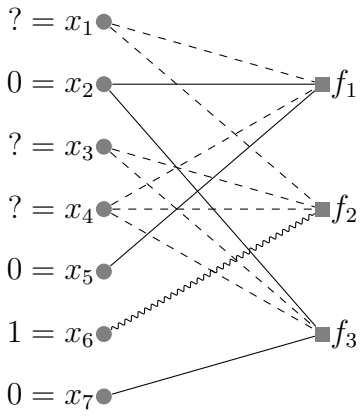


FIGURE 2.3: The greedy algorithm fails to decode $x = (?, 0, ?, ?, 0, 1, 0)$.

The reason why the greedy algorithm was not capable of recovering the erased variables is the presence of the cycle $\{x_1, f_1, x_4, f_3, x_3, f_2\}$ in the Tanner graph, and the fact that the all of its variables are erased. In general, cycles of small length are undesirable. The smaller the cycle, the greater the probability that all its variables will be erased. If all the variables in the cycle are erased, then the algorithm will fail, unless one of the variables is recovered by means of a check node that is not present in the cycle.

2.2 MAP and ML decoding

Consider the scenario, where the transmitter choses a codeword X from $\mathcal{C}$ with probability distribution $\Pr(X = x)$. The codeword is then transmitted and Y is the random variable that is equal to the channel output. If we decode the received word y to $\hat{x}(y)$, then the probability that we have correctly recovered the original codeword is $\Pr(X = \hat{x}(y) \mid Y = y)$. We should choose $\hat{x}(y)$ such that this probability is maximized. This scheme is called *maximum a-posteriori* (MAP) decoding and may be expressed as

$$\hat{x}^{\mathrm{MAP}}(y) = \operatorname*{argmax}_{x \in \mathcal{C}} \Pr(X = x \mid Y = y), \tag{2.2}$$

where argmax stands for the argument of the maximum. The *argument of the maximum* $\operatorname{argmax}_{x \in \mathcal{X}} f(x)$ is defined as the point $x \in \mathcal{X}$ for which f attains its maximum value. If there are multiple such points, then $\operatorname{argmax}_{x \in \mathcal{X}} f(x)$ is undefined.

Alternatively we could choose to decode y as the codeword that is most likely to have brought about the observed output, i.e. that which maximizes the probability $\Pr(Y = y \mid X = \hat{x}(y))$. This scheme is called *maximum likelihood* (ML) decoding and may be expressed as

$$\hat{x}^{\mathrm{ML}}(y) = \operatorname*{argmax}_{x \in \mathcal{C}} \Pr(Y = y \mid X = x).$$

Recall Bayes' Theorem, which states that for any two events A and B

$$\Pr(A \mid B) = \frac{\Pr(B \mid A) \Pr(A)}{\Pr(B)},$$

provided that the probability of B is not zero.

Applying Bayes' rule to (2.2) yields

$$\hat{x}^{\mathrm{MAP}}(y) = \operatorname*{argmax}_{x \in \mathcal{C}} \Pr(Y = y \mid X = x) \frac{\Pr(X = x)}{\Pr(Y = y)}.$$

Since y is given to the decoding algorithm as an input, the probability $\Pr(Y = y)$ plays the role of a positive multiplicative constant in the expression being maximized. It therefore does not affect the argument of the maximum and may be disregarded:

$$\hat{x}^{\mathrm{MAP}}(y) = \operatorname*{argmax}_{x \in \mathcal{C}} \Pr(Y = y \mid X = x) \Pr(X = x).$$

We see that if X is uniformly distributed, i.e. all codewords are equally likely to be transmitted, then $\hat{x}^{\mathrm{MAP}}(y) = \hat{x}^{\mathrm{ML}}(y)$.

Maximum likelihood decoding for linear codes on the BSC is NP-complete as shown by Berlekamp, McEliece, and van Tilborg [3]. However, this merely says that there is no efficient algorithm known that solves the ML decoding problem for *all* linear codes. It leaves open the possibility that some subclasses of codes have an efficient ML decoding algorithm.

In comparison, on the BEC maximum likelihood decoding can be achieved in polynomial time in the length of the code, as we will see shortly.

MAP decoding on the BEC

We shall assume that the transmitted codewords are chosen from a uniform distribution. This assumption is reasonable (provided that the mapping from source messages to codewords is onto), since source message redundancy can be removed by data compression if necessary.

Let $\mathcal{E}$ denote the *index set of erasures*, i.e. $i \in \mathcal{E}$ if and only if $y_i = ?$, and let $\bar{\mathcal{E}} = \{1, \ldots, n\} \setminus \mathcal{E}$. Let $H_\mathcal{E}$ denote the submatrix of H, restricted to the columns indexed by $\mathcal{E}$. For every $x \in \mathcal{C}$ we have $Hx^\mathrm{T} = 0^\mathrm{T}$, we can rewrite this as $H_\mathcal{E} x_\mathcal{E}^\mathrm{T} + H_{\bar{\mathcal{E}}} x_{\bar{\mathcal{E}}}^\mathrm{T} = 0^\mathrm{T}$ or

$$H_\mathcal{E} x_\mathcal{E}^\mathrm{T} = H_{\bar{\mathcal{E}}} x_{\bar{\mathcal{E}}}^\mathrm{T}.$$

The right side is equal to $H_{\bar{\mathcal{E}}} y_{\bar{\mathcal{E}}}^\mathrm{T}$ and is therefore known to the receiver. We can now find $x_\mathcal{E}$ by solving a system of linear equations. Since x is a valid codeword of $\mathcal{C}$ a solution surely exists. This solution is unique if and only if $\operatorname{rank}(H_\mathcal{E}) = |\mathcal{E}|$. We define the set of candidate codewords

$$\mathcal{X}(y) = \{\, x \in \mathcal{C} \mid H_\mathcal{E} x_\mathcal{E}^\mathrm{T} = H_{\bar{\mathcal{E}}} y_{\bar{\mathcal{E}}}^\mathrm{T}, x_{\bar{\mathcal{E}}} = y_{\bar{\mathcal{E}}} \,\}.$$

Assuming that the codewords are chosen from a uniform distribution

$$\hat{x}^\mathrm{MAP}(y) = \operatorname*{argmax}_{x \in \mathcal{C}} \Pr(X = x \mid Y = y) = \operatorname*{argmax}_{x \in \mathcal{C}} \Pr(Y = y \mid X = x).$$

For any codeword x such that $x_{\bar{\mathcal{E}}} = y_{\bar{\mathcal{E}}}$

$$\Pr(Y = y \mid X = x) = \varepsilon^{|\mathcal{E}|}(1 - \varepsilon)^{n - |\mathcal{E}|},$$

therefore all elements of $\mathcal{X}(y)$ are equally likely and so

$$\hat{x}^\mathrm{MAP}(y) = \begin{cases} x \in \mathcal{X}(y) & \text{if } |\mathcal{X}(y)| = 1, \\ \text{fail} & \text{otherwise.} \end{cases}$$

We see that MAP decoding for the BEC can be accomplished in complexity at most $O(n^3)$ by solving a system of linear equations, e.g. by Gaussian elimination.

MAP decoding on a memoryless channel

Let us now look more closely at MAP decoding on memoryless channels in general. We shall proceed one bit at a time. By X_i we denote the random variable that is equal to the ith bit of the channel input and by Y_i the random variable that is equal to the ith bit of the channel output. We again assume that the transmitted codewords are chosen from a uniform distribution. If we decode the ith bit y_i of the received word y to $\hat{x}_i(y)$, then the probability that we have correctly recovered the original bit is $\Pr(X_i = \hat{x}_i(y) \mid Y = y)$. We should choose $\hat{x}_i(y)$ such that this probability is maximized. This scheme is called *bitwise* MAP decoding and may be expressed as

$$\hat{x}_i^\mathrm{MAP}(y) = \operatorname*{argmax}_{x_i \in \{0,1\}} \Pr(X_i = x_i \mid Y = y).$$

Recall the law of total probability, which states that if $\{ B_i \mid i \in I \}$ is a set of pairwise disjoint events whose union is the entire sample space and I is a finite index set, then for any event A of the same probability space

$$\Pr(A) = \sum_{i \in I} \Pr(A \cap B_i).$$

Consider the index set $\mathbb{F}_2^n$. For each $z \in \mathbb{F}_2^n$ let B_z be the event $X = z$, i.e. the event that the transmitter chooses to transmit z. Note that if $z \notin \mathcal{C}$ then $\Pr(B_z) = 0$. Let A be the event $(X_i = x_i) \cap (Y = y)$. We now apply the law of total probability

$$\begin{aligned}
\Pr(X_i = x_i \mid Y = y) &= \frac{\Pr((X_i = x_i) \cap (Y = y))}{\Pr(Y = y)} \\
&= \frac{1}{\Pr(Y = y)} \sum_{z \in \mathbb{F}_2^n} \Pr((X_i = x_i) \cap (Y = y) \cap (X = z)) \\
&= \sum_{z \in \mathbb{F}_2^n} \Pr((X_i = x_i) \cap (X = z) \mid Y = y) \\
&= \sum_{\substack{z \in \mathbb{F}_2^n \\ z_i = x_i}} \Pr(X = z \mid Y = y).
\end{aligned}$$

Thus we have

$$\hat{x}_i^{\mathrm{MAP}}(y) = \operatorname*{argmax}_{x_i \in \{0,1\}} \sum_{\substack{z \in \mathbb{F}_2^n \\ z_i = x_i}} \Pr(X = z \mid Y = y).$$

Applying Bayes' theorem, as we did in Section 2.2, and using the fact that the channel is memoryless

$$\begin{aligned}
\hat{x}_i^{\mathrm{MAP}}(y) &= \operatorname*{argmax}_{x_i \in \{0,1\}} \sum_{\substack{z \in \mathbb{F}_2^n \\ z_i = x_i}} \Pr(Y = y \mid X = z) \frac{\Pr(X = z)}{\Pr(Y = y)} \\
&= \operatorname*{argmax}_{x_i \in \{0,1\}} \sum_{\substack{z \in \mathbb{F}_2^n \\ z_i = x_i}} \Pr(Y = y \mid X = z) \Pr(X = z) \\
&= \operatorname*{argmax}_{x_i \in \{0,1\}} \sum_{\substack{z \in \mathbb{F}_2^n \\ z_i = x_i}} \left(\prod_j \Pr(Y_j = y_j \mid X_j = z_j) \right) \Pr(X = z).
\end{aligned}$$

Assuming that the codewords are chosen from a uniform distribution, we can replace $\Pr(X = z)$ by the indicator function $\mathbf{1}[z \in \mathcal{C}]$, which is defined as 1 if the condition in the brackets is true and 0 otherwise.

$$\hat{x}_i^{\mathrm{MAP}}(y) = \operatorname*{argmax}_{x_i \in \{0,1\}} \sum_{\substack{z \in \mathbb{F}_2^n \\ z_i = x_i}} \left(\prod_j \Pr(Y_j = y_j \mid X_j = z_j) \right) \mathbf{1}[z \in \mathcal{C}] \tag{2.3}$$

For any linear code the indicator function can be factorised according to the parity check equations. Consider the linear code $\mathcal{C}$ with parity-check matrix

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}. \tag{2.4}$$

In this case $\mathbf{1}[z \in \mathcal{C}]$ factorizes to

$$\mathbf{1}[z \in \mathcal{C}] = \mathbf{1}[z_1 + z_2 + z_3 = 0]\,\mathbf{1}[z_2 + z_5 = 0]\,\mathbf{1}[z_3 + z_4 = 0].$$

We see that bitwise MAP decoding can be performed by computing sums of products of certain multivariate functions.

2.3 Marginalization by message passing

Definition 2.5. Let f be a function from a finite set $\mathcal{X}^n$ to $\mathbb{R}$ and let $i \in \mathbb{N}$, $1 \leq i \leq n$. The *marginal of* $f(x_1, \ldots, x_n)$ *with respect to* x_i is a function from $\mathcal{X}$ to $\mathbb{R}$, defined as

$$\sum_{(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)\in\mathcal{X}^{n-1}} f(x_1, \ldots, x_n). \tag{2.5}$$

In order to facilitate the expression of marginals, we introduce the following simplified notation for (2.5)

$$\sum_{\sim x_i} f(x_1, \ldots, x_n).$$

The symbol $\sum_{\sim \ldots}$ denotes summation over all variables contained in the expression except the one listed.

Under the right conditions the computational complexity of marginalization of a function can be significantly reduced by exploiting its factorisation. The idea is illustrated in this simple example.

Example 2.6. Let $f_1, \ldots, f_n$ be real-valued functions defined on the set $\mathcal{X}$, then

$$\sum_{(x_1,\ldots,x_n)\in\mathcal{X}^n} f_1(x_1)\cdots f_n(x_n) = \left(\sum_{x\in\mathcal{X}} f_1(x)\right)\cdots\left(\sum_{x\in\mathcal{X}} f_n(x)\right).$$

Evaluating the left side requires $n\,|\mathcal{X}|^n - 1$ operations of addition and multiplication, whereas evaluating the right side requires only $n\,|\mathcal{X}| - 1$ operations.

The next example shows a scenario, where the idea is applied recursively.

Example 2.7. Let f be a function defined on $\mathbb{F}_2^5$ such that

$$f(x_1, \ldots, x_5) = f_1(x_1, x_2, x_3) f_2(x_2, x_5) f_3(x_3, x_4).$$

The marginal of f with respect to x_3 is

$$\begin{aligned}
&\sum_{x_1,x_2,x_4,x_5\in\mathbb{F}_2} f_1(x_1, x_2, x_3) f_2(x_2, x_5) f_3(x_3, x_4) \\
&\quad= \Bigg(\sum_{x_1,x_2,x_5\in\mathbb{F}_2} f_1(x_1, x_2, x_3) f_2(x_2, x_5)\Bigg)\Bigg(\sum_{x_4\in\mathbb{F}_2} f_3(x_3, x_4)\Bigg) \\
&\quad= \Bigg(\sum_{x_1,x_2\in\mathbb{F}_2} f_1(x_1, x_2, x_3) \sum_{x_5\in\mathbb{F}_2} f_2(x_2, x_5)\Bigg)\Bigg(\sum_{x_4\in\mathbb{F}_2} f_3(x_3, x_4)\Bigg) \\
&\quad= \Bigg(\sum_{\sim x_3} f_1(x_1, x_2, x_3) \sum_{\sim x_2} f_2(x_2, x_5)\Bigg)\Bigg(\sum_{\sim x_3} f_3(x_3, x_4)\Bigg).
\end{aligned}$$

The last line shows the use of the simplified notation.

To evaluate the marginal we start with the innermost sum $\sum_{x_5 \in \mathbb{F}_2} f_2(x_2, x_5)$ and evaluate it for both values of x_2, call the results $\mu_1(0)$ and $\mu_1(1)$. Now evaluate the left term for both values of x_3 using the results from the previous step $\sum_{x_1, x_2 \in \mathbb{F}_2} f_1(x_1, x_2, x_3)\mu_1(x_2)$ and call the results $\mu_2(0)$ and $\mu_2(1)$. Similarly evaluate the right term for both values of x_3 and call the results $\mu_3(0)$ and $\mu_3(1)$. Finally compute $\mu_2(x_3)\mu_3(x_3)$ for both values of x_3 to obtain the end results.

Our goal is to generalize the described procedure. For this purpose, we associate a graph with every factorisation.

Definition 2.8. The *factor graph* associated with the factorisation of the function f is a bipartite graph with *variable nodes* corresponding to the variables of f, and *factor nodes* corresponding to the factors of f. A factor node f_i is connected to a variable node x_j if and only if the factor f_i is dependent on the variable x_j.

Figure 2.4 shows the factor graph associated with the factorisation of f from Example 2.7. Note that the graph is in fact identical to the Tanner graph associated with the parity-check matrix (2.4).

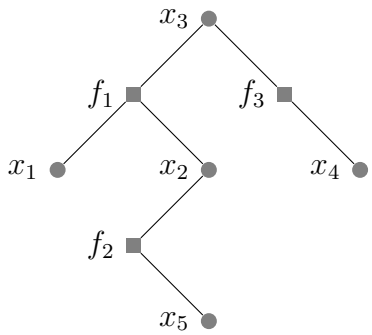


FIGURE 2.4: Factor graph associated with the factorisation of f from Example 2.7.

Assume that a function f defined on $\mathbb{F}_2^n$ has a factorisation $\prod_i f_i$ such that its factor graph is a tree. We shall compute the marginal of $f(x_1, \ldots, x_n)$ with respect to the variable x_1. Designate x_1 as the root of the factor graph. There exists a factorisation $\prod_j g_j$ of f such that the variable x_1 appears in each factor, while every other variable appears in exactly one factor. The factors g_j correspond to the subtrees emanating from vertex x_1. Each factor g_j is equal to the product of the factors f_i, whose factor nodes lie in the corresponding subtree. If any variable other than x_1 were to appear in two different factors g_{j_1} and g_{j_2}, then this would imply that the associated factor graph contains a cycle, which contradicts the assumption that the factor graph is a tree. The fact that variables other than x_1 are not shared between factors g_j allows us to exchange the order of summation and multiplication:

$$\sum_{\sim x_1} f(x) = \sum_{\sim x_1} \prod_j g_j(x) = \prod_j \sum_{\sim x_1} g_j(x) = \prod_j \mu_j(x_1),$$

where $\mu_j(x_1) = \sum_{\sim x_1} g_j(x)$ is the marginal of g_j with respect to x_1.

The next step is to simplify the computation of the marginals $\mu_j(x_1)$. For any g_j there exists a factorisation $\prod_{k=0}^{K} h_k$ of g_j such that the factor h_0 is dependent on x_1 and on some variables that we shall call $z_1, \ldots, z_K$, and such that no two factors h_{k_1}, h_{k_2}, $1 \leq k_1 < k_2 \leq K$ share a common variable. The factor h_0 is the root node of the subtree corresponding to g_j. The variables $z_1, \ldots, z_K$ are the child nodes of h_0, and each h_k, $1 \leq k \leq K$ corresponds to the subtree rooted at z_k. Thus we get

$$\begin{aligned}\mu_j(x_1) &= \sum_{\sim x_1} g_j(x) = \sum_{\sim x_1} \prod_k h_k(x) = \sum_{\sim x_1} h_0(x_1, z_1, z_2, \ldots, z_K) \prod_{k=1}^{K} h_k(z_k, \ldots) \\ &= \sum_{z_1,\ldots,z_K} h_0(x_1, z_1, z_2, \ldots, z_K) \prod_{k=1}^{K} \sum_{\sim z_k} h_k(z_k, \ldots) \\ &= \sum_{z_1,\ldots,z_K} h_0(x_1, z_1, z_2, \ldots, z_K) \prod_{k=1}^{K} \mu'_k(z_k),\end{aligned}$$

where $\mu'_k(z_k) = \sum_{\sim z_k} h_k(z_k, \ldots)$ is the marginal of h_k with respect to z_k. We are again faced with the task of computing marginals of functions that have a factorisation such that the factor graph is a tree.

The steps described above can now be applied recursively until the leaf nodes are reached. For a variable leaf node the marginal is simply equal to 1 as it has no child factor nodes. For a factor leaf node $g_j(x_i)$ the marginal is the factor itself, since $\sum_{\sim x_i} g_j(x_i) = g_j(x_i)$. This leads us to the following algorithm.

2.4 The sum-product algorithm

The sum-product algorithm proceeds by sending messages, denoted μ, along the edges of the factor graph. The messages that pass to and from variable node x_i are functions of the variable x_i. In practice messages are represented as vectors $(\mu(0), \mu(1))$. Message passing starts at the leaf nodes, proceeds up the tree, and finishes at the root node. The rules for every type of node are as follows:

(a) A variable leaf node x_i sends the message $\mu(x_i) = 1$ to its parent.

(b) A factor leaf node $f_j(x_i)$ sends the message $\mu(x_i) = f_j(x_i)$ to its parent node x_i.

(c) A variable node x_i that has received messages $\mu_1, \ldots, \mu_J$ from all its children, sends its parent the message

$$\mu(x_i) = \prod_{j=1}^{J} \mu_j(x_i).$$

(d) A factor node $f_j(x_i)$ that has received messages $\mu_1, \ldots, \mu_K$ from all its children, call them $z_1, \ldots, z_K$, respectively, sends its parent node x_i the message

$$\mu(x_i) = \sum_{z_1,\ldots,z_K} f_j(x_i, z_1, z_2, \ldots, z_K) \prod_{k=1}^{K} \mu_k(z_k).$$

The marginal of f with respect to x_1 is obtained by multiplying all the messages received at the root node.

In practice we wish to compute the marginals of f with respect to every variable. This can be done by running the algorithm n times, every time designating a different variable node as the root of the factor graph. Notice, however, that the message that is passed from some node a to its parent node b, remains the same regardless of which node in the factor graph is designated as the root, as long as b remains the parent of a with respect to the newly designated root node. If the parent-child relationship between nodes a and b is reversed as a result of designating a different root node, then no message will pass from a to b (some message will pass in the opposite direction instead). This allows us to compute the marginals all in one go by simultaneously perceiving each variable node of the factor graph as a root node.

Method 1

We start at the leaf nodes, applying rules (a) and (b). At every inner node we treat each one of the neighbouring nodes as if it were a parent node. In other words, for each inner node a let $\mathcal{N}(a)$ be the set of neighbouring nodes of node a, for each node $b \in \mathcal{N}(a)$ wait until messages from nodes $\mathcal{N}(a) \setminus \{b\}$ have arrived, then apply the appropriate rule (c) or (d) and send the resulting message to node b. Thus messages are passed along every edge in each direction, and after a finite number of steps, every message is created. The marginal of f with respect to any x_i is then obtained by multiplying all the messages received at node x_i.

Method 2

Another way to achieve the same result is to initialize the algorithm by passing the message $\mu \equiv 1$ from *each* variable node to the neighbouring factor nodes, and then to proceed in iterations. An iteration starts by processing messages at factor nodes and sending the resulting messages to variable nodes along all edges. These messages are processed at the variable nodes and the resulting messages are sent to factor nodes along all edges. This constitutes one iteration. After a number of iterations approximately equal to half the diameter of the graph, the algorithm will converge to the same set of messages as the previous algorithm. The marginal of f with respect to any x_i is then obtained by multiplying all the messages received at node x_i in the last iteration.

Compared with the first method, this one involves a lot of wasted computations, however it has the advantage that it can be applied to a graph that has cycles. On a graph with cycles, the algorithm does not necessarily have to converge and it does not even compute the correct marginals, nevertheless it produces very good results when used for decoding, since we only need to know which of the marginals is greater. The algorithm is usually terminated after a predetermined number of iterations. This number can be as high as half the diameter of the Tanner graph, but usually one would choose a smaller value, which compromises between the need to maximise effectiveness of decoding and the need to minimise the number of computations. Alternatively, one can compute all the marginals

after every iteration, and if these produce a valid codeword, i.e. all constraint equations are satisfied, then the algorithm is terminated early.

2.5 Sum-product algorithm for bitwise MAP decoding

We can now apply the sum product algorithm to the bitwise MAP decoding problem discussed in Section 2.2. Figure 2.5 shows the factor graph associated with the factorisation

$$\left(\prod_{j=1}^{5} \Pr(Y_j = y_j \mid X_j = x_j)\right) \mathbf{1}[x_1 + x_2 + x_3 = 0]\, \mathbf{1}[x_2 + x_5 = 0]\, \mathbf{1}[x_3 + x_4 = 0].$$

This factorisation corresponds to the bitwise MAP decoding of the code with parity-check matrix (2.4).

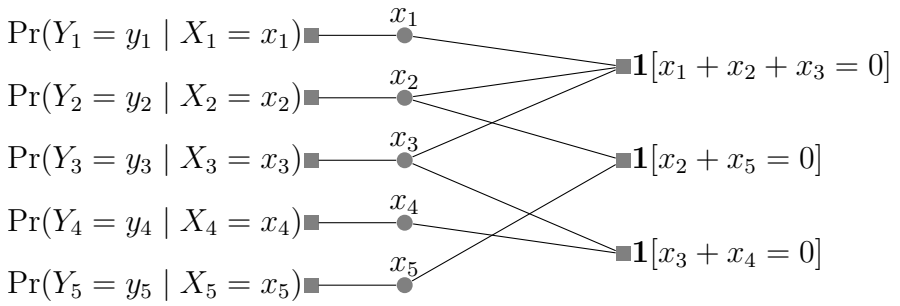


FIGURE 2.5: Factor graph for the bitwise MAP decoding of the code with parity-check matrix (2.4).

If the factor graph is a tree, like in Figure 2.5, we can use the first method of sum-product decoding, however all the rules we derive in this section apply to the second method just as well. The algorithm starts at the factor leaf nodes on the left, for each i passing the message

$$(\mu(0), \mu(1)) = (\Pr(Y_i = y_i \mid X_i = 0), \Pr(Y_i = y_i \mid X_i = 1))$$

to the variable node x_i. In practice, we work with the ratios $r = \mu(0)/\mu(1)$ instead, in order to reduce the number of computations. The rules for working with these ratios are as follows:

According to rule (c) of the sum-product algorithm, at a variable node of degree $J + 1$ we multiply J incoming messages to produce the outgoing message

$$\mu(0) = \prod_{j=1}^{J} \mu_j(0), \qquad \mu(1) = \prod_{j=1}^{J} \mu_j(1).$$

When working with ratios $r_1, \ldots, r_J$, the outgoing ratio at a variable node is simply

$$r = \frac{\prod_{j=1}^{J} \mu_j(0)}{\prod_{j=1}^{J} \mu_j(1)} = \prod_{j=1}^{J} \frac{\mu_j(0)}{\mu_j(1)} = \prod_{j=1}^{J} r_j.$$

Let us now look at the inner factor nodes, i.e. the nodes on the right in Figure 2.5. Here the incoming messages from variable nodes $z_1, \ldots, z_K$ produce the outgoing message

$$\mu(0) = \sum_{z_1,\ldots,z_K} \mathbf{1}\Big[\sum_{k=1}^{K} z_k = 0\Big] \prod_{k=1}^{K} \mu_k(z_k), \qquad \mu(1) = \sum_{z_1,\ldots,z_K} \mathbf{1}\Big[\sum_{k=1}^{K} z_k = 1\Big] \prod_{k=1}^{K} \mu_k(z_k).$$

When working with ratios, the outgoing ratio at these factor nodes is

$$\begin{aligned} r &= \frac{\sum_{z_1,\ldots,z_K} \mathbf{1}[\sum_{k=1}^{K} z_k = 0] \prod_{k=1}^{K} \mu_k(z_k)}{\sum_{z_1,\ldots,z_K} \mathbf{1}[\sum_{k=1}^{K} z_k = 1] \prod_{k=1}^{K} \mu_k(z_k)} = \frac{\sum_{z_1,\ldots,z_K} \mathbf{1}[\sum_{k=1}^{K} z_k = 0] \prod_{k=1}^{K} \frac{\mu_k(z_k)}{\mu_k(1)}}{\sum_{z_1,\ldots,z_K} \mathbf{1}[\sum_{k=1}^{K} z_k = 1] \prod_{k=1}^{K} \frac{\mu_k(z_k)}{\mu_k(1)}} \\ &= \frac{\sum_{z_1,\ldots,z_K} \mathbf{1}[\sum_{k=1}^{K} z_k = 0] \prod_{k=1}^{K} r_k^{\mathbf{1}[z_k=0]}}{\sum_{z_1,\ldots,z_K} \mathbf{1}[\sum_{k=1}^{K} z_k = 1] \prod_{k=1}^{K} r_k^{\mathbf{1}[z_k=0]}} = \frac{\prod_{k=1}^{K}(r_k+1) + \prod_{k=1}^{K}(r_k-1)}{\prod_{k=1}^{K}(r_k+1) - \prod_{k=1}^{K}(r_k-1)}. \end{aligned}$$

In order to understand the last step expand the products in the last formula to get

$$\prod_{k=1}^{K}(r_k+1) = \sum_{\mathcal{I}\subseteq\{1,\ldots,K\}} \prod_{i\in\mathcal{I}} r_i = \sum_{\mathcal{I}\subseteq\{1,\ldots,K\}} \prod_{i\in\{1,\ldots,K\}\setminus\mathcal{I}} r_i = \sum_{z_1,\ldots,z_K} \prod_{k=1}^{K} r_k^{\mathbf{1}[z_k=0]}$$

and

$$\begin{aligned} \prod_{k=1}^{K}(r_k-1) &= \sum_{\mathcal{I}\subseteq\{1,\ldots,K\}} (-1)^{K-|\mathcal{I}|} \prod_{i\in\mathcal{I}} r_i = \sum_{\mathcal{I}\subseteq\{1,\ldots,K\}} (-1)^{|\mathcal{I}|} \prod_{i\in\{1,\ldots,K\}\setminus\mathcal{I}} r_i \\ &= \sum_{z_1,\ldots,z_K} (-1)^{\mathbf{1}[\sum_{k=1}^{K} z_k=1]} \prod_{k=1}^{K} r_k^{\mathbf{1}[z_k=0]}. \end{aligned}$$

Now it is easy to see that the nominator can be expressed

$$\prod_{k=1}^{K}(r_k+1) + \prod_{k=1}^{K}(r_k-1) = 2 \sum_{z_1,\ldots,z_K} \mathbf{1}[\sum_{k=1}^{K} z_k = 0] \prod_{k=1}^{K} r_k^{\mathbf{1}[z_k=0]},$$

and analogously for the denominator.

Finally dividing the numerator and denominator by $\prod_{k=1}^{K}(r_k+1)$ yields

$$r = \frac{1 + \prod_{k=1}^{K} \frac{r_k-1}{r_k+1}}{1 - \prod_{k=1}^{K} \frac{r_k-1}{r_k+1}}. \tag{2.6}$$

Another useful computational trick is to work with logarithms of ratios $l_i = \ln r_i$. This means that at the variable nodes the outgoing message is computed as $l = \sum_{j=1}^{J} l_j$. At the factor nodes the computation of the outgoing message also attains an elegant form. Equation (2.6) implies that

$$\frac{r-1}{r+1} = \prod_{k=1}^{K} \frac{r_k-1}{r_k+1},$$

after substituting $r_k = \mathrm{e}^{l_k}$ we have

$$\tanh(l/2) = \frac{\mathrm{e}^l - 1}{\mathrm{e}^l + 1} = \prod_{k=1}^{K} \frac{\mathrm{e}^{l_k} - 1}{\mathrm{e}^{l_k} + 1} = \prod_{k=1}^{K} \tanh(l_k/2).$$

So at inner factor nodes the outgoing message will be

$$l = 2 \tanh^{-1}\left(\prod_{k=1}^{K} \tanh(l_k/2)\right).$$

The number of operations can be reduced by omitting messages with large $|l_k|$, taking into account only their sign, since $\lim_{l_k \to \pm\infty} \tanh(l_k/2) = \pm 1$. Even for relatively small values the function rapidly approaches 1, e.g. $\tanh(5) \approx 0.99991$. Thus if we have for example the incoming messages $l_1 = 3$ and $l_2 = -10$, then the outgoing message will be $l \approx -2.9991$.

The evaluation of the hyperbolic tangent and its inverse can be implemented by means of a look-up table. We can also replace the multiplication with summation by working with the logarithm of the hyperbolic tangent. However, we need to be careful with negative l_k. We therefore factorise $\tanh(l_k/2)$ into its sign and magnitude:

$$\tanh(l/2) = \left(\prod_{k=1}^{K} \operatorname{sgn} l_k\right)\left(\prod_{k=1}^{K} \tanh(|l_k|/2)\right).$$

Then we apply the logarithm

$$\begin{aligned} l &= \left(\prod_{k=1}^{K} \operatorname{sgn} l_k\right) 2 \tanh^{-1}\left(\prod_{k=1}^{K} \tanh(|l_k|/2)\right) \\ &= \left(\prod_{k=1}^{K} \operatorname{sgn} l_k\right) 2 \tanh^{-1} \ln^{-1}\left(\sum_{k=1}^{K} \ln \tanh(|l_k|/2)\right) \\ &= \left(\prod_{k=1}^{K} \operatorname{sgn} l_k\right) \phi^{-1}\left(\sum_{k=1}^{K} \phi(|l_k|)\right), \end{aligned}$$

where $\phi(x) = -\ln \tanh(x/2)$. Notice the minus sign in the definition of ϕ, its presence causes ϕ to be self inverse, while it makes no difference in the above. The evaluation of ϕ can once again be implemented by means of a lookup table. We have thus reduced the operations of the sum-product algorithm to a series of floating point additions and look-ups into a single look-up table. On the BSC, the algorithm can be simplified even further, since the factor leaf nodes pass only two different messages. The same applies on the BEC, where the factor leaf nodes pass only three different messages.

The min-sum algorithm

As noted above, messages with large $|l_k|$ can be ignored in the sum-product algorithm without impacting its decoding ability. We need only take into account

the sign of l_k. If we go so far as to ignore all l_k except the smallest one, then the computation of the outgoing message at the factor nodes simplifies to

$$l = \left(\prod_{k=1}^{K} \operatorname{sgn} l_k\right) \phi^{-1}\left(\min_{k\in\{1,\dots,K\}} \phi(|l_k|)\right) = \left(\prod_{k=1}^{K} \operatorname{sgn} l_k\right) \min_{k\in\{1,\dots,K\}} |l_k| \,.$$

At the variable nodes, we leave the rule unchanged: the outgoing message is computed as $l = \sum_{j=1}^{J} l_j$. The algorithm employing these rules is called the min-sum algorithm.

2.6 Sum-product algorithm on the BEC

The sum-product decoding algorithm can be greatly simplified on the BEC. The algorithm is initialized by sending the message $(1, 1)$ from each variable node to the neighbouring factor nodes. During every iteration of the algorithm the leaf factor nodes pass the message

$$(\mu(0), \mu(1)) = (\Pr(Y_i = y_i \mid X_i = 0), \Pr(Y_i = y_i \mid X_i = 1)),$$

which is $(1-\varepsilon, 0)$, $(0, 1-\varepsilon)$ or $(\varepsilon, \varepsilon)$ depending on whether the value of y_i is 0, 1 or ?, respectively. Recall that the algorithm works just as well with ratios $\mu(0)/\mu(1)$, so there is no harm done in normalizing the messages and calling them $(1, 0)$, $(0, 1)$ and $(1, 1)$, respectively. These three messages are in fact the only ones that get passed around (up to normalization). We will also represent them by the symbols 0, 1 and ?.

At the inner factor nodes (check nodes) the incoming messages $\mu_1, \dots, \mu_K$ from variable nodes $z_1, \dots, z_K$ produce the message

$$\left(\sum_{z_1,\dots,z_K} \mathbf{1}\left[\sum_{k=1}^{K} z_k = 0\right] \prod_{k=1}^{K} \mu_k(z_k), \sum_{z_1,\dots,z_K} \mathbf{1}\left[\sum_{k=1}^{K} z_k = 1\right] \prod_{k=1}^{K} \mu_k(z_k)\right). \tag{2.7}$$

Notice that if any one of the incoming messages, say μ_1, is $(1, 1)$, then we have

$$\left(\sum_{z_1} \sum_{z_2,\dots,z_K} \mathbf{1}\left[\sum_{k=2}^{K} z_k = z_1\right] \prod_{k=2}^{K} \mu_k(z_k), \sum_{z_1} \sum_{z_2,\dots,z_K} \mathbf{1}\left[\sum_{k=2}^{K} z_k = 1 + z_1\right] \prod_{k=2}^{K} \mu_k(z_k)\right),$$

i.e. $\mu(0) = \mu(1)$. It is nonzero, since for at least one $(K-1)$-tuple $(z_2, \dots, z_K)$ from $\mathbb{F}_2^{K-1}$ the product $\prod_{k=2}^{K} \mu_k(z_k)$ is nonzero, and then the indicator function $\mathbf{1}[\sum_{k=2}^{K} z_k = z_1]$ is nonzero for either $z_1 = 0$ or $z_1 = 1$. We can therefore normalize the outgoing message to $(1, 1)$.

Looking back at (2.7), if none of the incoming messages is $(1, 1)$, then for exactly one K-tuple $(z_1, \dots, z_K) \in \mathbb{F}_2^K$ the product $\prod_{k=1}^{K} \mu_k(z_k)$ will come out nonzero. This K-tuple corresponds to the values of the incoming messages, each z_k being 0 if the kth incoming message is $(1, 0)$ and 1 if the message is $(0, 1)$. The outgoing message is then

$$\left(\mathbf{1}\left[\sum_{k=1}^{K} z_k = 0\right], \mathbf{1}\left[\sum_{k=1}^{K} z_k = 1\right]\right),$$

which is either $(1,0)$ or $(0,1)$. If we represent the messages as symbols 0, 1 and ?, then the rule simplifies to:

- If none of the incoming messages at a check node is ?, then the outgoing message is the mod-2 sum of the incoming messages. Otherwise the outgoing message is ?.

At the variable nodes we obtain the outgoing message by point-wise multiplication of the incoming messages. If all incoming messages are $(1,1)$, then the outgoing message is $(1,1)$. Assuming that a valid codeword was sent over the channel, and that the channel erased some bits, but did not alter any of them, then at any point in the algorithm, the messages $(1,0)$ and $(0,1)$ can never both arrive at the same variable node. If this did happen, it would be in violation of the constraint equations, since, as we just saw, the sum-product algorithm calculates the mod-2 sum at the check nodes. This means that all the incoming messages at a variable node either lie in the set $\{(1,0),(1,1)\}$ or in the set $\{(0,1),(1,1)\}$. If at least one of the incoming messages is not $(1,1)$, then by point-wise multiplication the outgoing message will be equal to that message. If we represent the messages as symbols, then the rule reads

- If at least one of the incoming messages μ_k at a variable node is not ?, then the outgoing message is equal to μ_k. Otherwise the outgoing message is ?.

The algorithm proceeds in iterations, until its state ceases to progress from one iteration to the next. The incoming messages at every variable node are then multiplied to obtain the (normalized) marginal with respect to that variable. Once again, if at least one of these messages μ_k is not $(1,1)$, then the marginal μ is equal to μ_k. The ith bit of the received word then decodes to $\operatorname{argmax}_{x_i \in \mathbb{F}_2} \mu(x_i)$, which is 0 if the marginal is $(1,0)$ and 1 if the marginal is $(0,1)$. If the marginal is $(1,1)$, then we have a decoding failure for x_i.

Example 2.9. Let $y = (1,0,?,?,0,1,?)$ be the output of a transmission over a BEC using the [7,4,3] Hamming code from Example 2.2. The sum-product algorithm is initialized by sending the message $(1,1)$ from each variable node to the neighbouring factor nodes, and then it proceeds in iterations. Figure 2.6 shows the first four iterations of the algorithm. The message $(1,0)$, which corresponds to 0, is represented by a solid line, the message $(0,1)$, which corresponds to 1, is represented by a snaked line, and the message $(1,1)$, which corresponds to ?, is represented by a dashed line. Notice that for every node the outgoing message along an edge depends on the incoming messages along all incident edges other then the edge itself. After four iterations the algorithm does not progress any further (the right diagram in the fourth iteration is the same as the right diagram in the third iteration). Looking at the left diagram in the fourth iteration, we see that the word decodes to $(1,0,1,1,0,1,0)$.

The sum-product algorithm on the BEC produces the same results as the greedy algorithm described in Section 2.1. In practice one would use the greedy algorithm for decoding on the BEC, however the sum-product algorithm has the advantage that it can be analyzed quite precisely on the BEC as we will see in Section 3.2.

2.7 Cycle-free graphs

In Section 2.4 we described an adaptation of the sum-product algorithm to graphs with cycles. However, the adapted algorithm requires more computations and it does not actually perform MAP decoding on graphs with cycles. The reason why we need to concern ourselves with decoding on such graphs is that the class of codes with cycle-free Tanner graphs is not powerful enough to perform well. The following theorem shows that cycle-free codes of rate above $\frac{1}{2}$ contain words of weight 2, and that the lower bound on the number of words of weight 2 exhibits hyperbolic growth as the rate of the code approaches 1. Codes of rate below $\frac{1}{2}$ also contain low-weight codewords. In [5] Etzion, Trachtenberg and Vardy have shown that for any $[n, k, d]_q$ cycle-free linear code

$$d \leq \left\lfloor \frac{n}{k+1} \right\rfloor + \left\lfloor \frac{n+1}{k+1} \right\rfloor < \frac{2}{R}.$$

Proposition 2.10. *Let $\mathcal{C}$ be a binary linear code of rate $r \geq 1/2$ that admits a Tanner graph that is a forest. Then $\mathcal{C}$ contains at least $\frac{rn(2r-1)}{2(1-r)}$ codewords of weight 2.*

Proof. The Tanner graph has n variable nodes and $(1-r)n$ check nodes, since for a forest the rows of the parity-check matrix that correspond to the check nodes are linearly independent. To see this, assume to the contrary that the rows $h_1, \ldots, h_m$ of the parity-check matrix are linearly dependent, then there exists a nonempty subset $\mathcal{I} \subseteq \{1, \ldots, m\}$ such that $\sum_{i \in \mathcal{I}} h_i = 0$. Restrict the parity-check matrix to the rows indexed by $\mathcal{I}$. This restricted matrix corresponds to a subgraph of the forest and as such, the subgraph is a (nonempty) forest itself. But every column in the restricted matrix has an even number of ones, which means that the subgraph has no variable leaf nodes. By assumption there are no leaf check nodes either. This is in contradiction to the subgraph being a nonempty forest.

The total number of nodes is $(2-r)n$ and the number of edges in the forest is therefore $(2-r)n - t$, where t is the number of trees in the forest. This means that the average degree of the variable nodes is less than $2-r$. It follows that the fraction of variable nodes that have degree at least 2 is less than $1-r$. The number of variable leaf nodes is therefore greater than rn.

For the next step, recall Jensen's inequality, which states that if f is a convex function on the interval $[a, b]$ and $x_1, \ldots, x_n \in [a, b]$, then

$$\frac{1}{n} \sum_{i=1}^{n} f(x_i) \geq f\left(\frac{1}{n} \sum_{i=1}^{n} x_i \right).$$

Any two variable leaf nodes that are connected to the same check node give rise to a codeword of weight 2. Let l_i denote the number of variable leaf nodes connected to the ith check node, these l_i leaf nodes give rise to $l_i(l_i-1)/2$ codewords of weight 2. By applying Jensen's inequality to the real convex function $x \mapsto x(x-1)/2$, we get

$$\frac{\sum_{i=1}^{(1-r)n} l_i(l_i-1)}{(1-r)n} \geq \frac{\sum_{i=1}^{(1-r)n} l_i}{(1-r)n} \left(\frac{\sum_{i=1}^{(1-r)n} l_i}{(1-r)n} - 1 \right) \geq \frac{rn}{(1-r)n} \left(\frac{rn}{(1-r)n} - 1 \right).$$

The total number of codewords of weight 2 is therefore at least

$$\sum_{i=1}^{(1-r)n} \frac{l_i(l_i-1)}{2} \geq \frac{rn}{2}\left(\frac{rn}{(1-r)n} - 1\right) = \frac{rn(2r-1)}{2(1-r)}.$$

□

2.8 Notes

In addition to the cited sources, the books *Modern Coding Theory* [13] by Richardson and Urbanke, *Information Theory, Inference, and Learning Algorithms* [10] by David MacKay, and *An Introduction to LDPC Codes* [14] by William E. Ryan were used throughout this chapter. Proposition 2.10 was adapted from Lemma 2.24 [13] with an improvement on the lower bound for the number of codewords of weight 2.

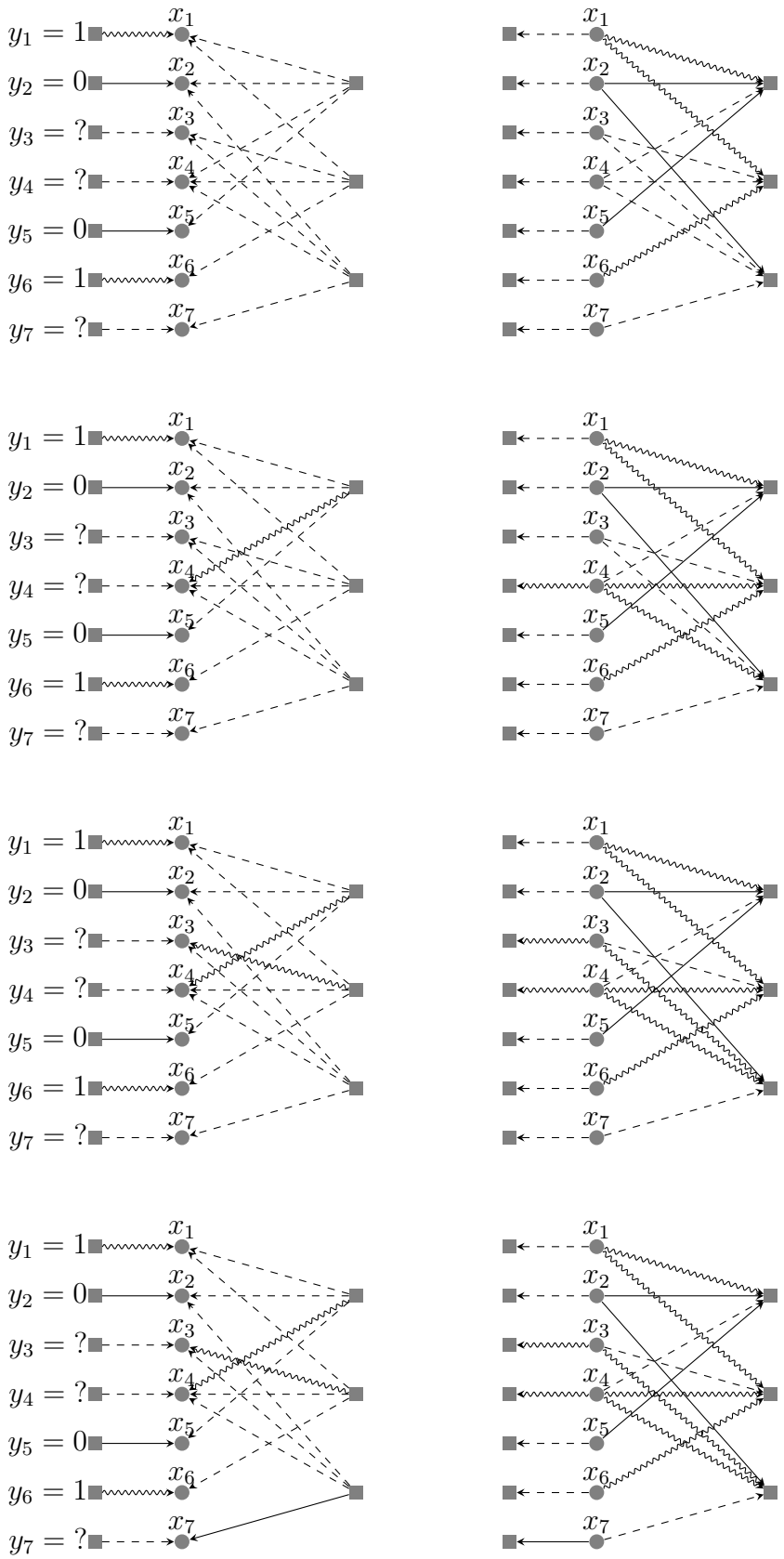


FIGURE 2.6: Iterations of the sum-product algorithm for the [7,4,3] Hamming code from Example 2.2 with $y = (?, 0, ?, ?, 0, 1, 0)$.

Chapter 3

Low-density parity-check codes

Just as there are many parity-check matrices for one code, there are many Tanner graphs for one code. The codes that have at least one *sparse* Tanner graph are called *low-density parity-check* (LDPC) codes. By sparse, we mean that the number of edges in the graph is of the order of a small multiple of n, where n is the length of the code. The sparsity of the graph is what brings about good performance under message-passing decoding. One of the reasons for this is that the number of operations of the sum-product algorithm is linear in the number of edges of the Tanner graph.

Definition 3.1. We call an LDPC code (l, r)-*regular* if it has a Tanner graph such that every variable node has degree l and every check node has degree r.

The *design rate* of an LDPC code with an $m \times n$ parity-check matrix is defined as $R = (n - m)/n$.

Note that the design rate might be less than the actual rate of the code since the rows of the parity-check matrix need not be linearly independent. But in fact the expected rank of a randomly chosen parity-check matrix is very close to m.

The number of edges in the Tanner graph of an (l, r)-regular LDPC code is $nl = mr$. The design rate R therefore satisfies

$$1 - R = \frac{m}{n} = \frac{l}{r}.$$

Notice that for constant l, the number of edges in the Tanner graph (i.e., the number of ones in the $n(1 - R) \times n$ parity-check matrix) grows linearly in n. In comparison, for a randomly generated $n(1 - R) \times n$ parity-check matrix the number of ones grows quadratically in n.

Definition 3.2. Let $\mathcal{C}$ be an LDPC code with a Tanner graph such that the number of variable nodes of degree i is Λ_i and the number of check nodes of degree i is P_i. We call the polynomial $\Lambda(x) = \sum_{i=1}^{l_{\max}} \Lambda_i x^i$ the *variable degree distribution*, $\mathrm{P}(x) = \sum_{i=1}^{r_{\max}} \mathrm{P}_i x^i$ the *check degree distribution*, and we call $\mathcal{C}$ a $(\Lambda(x), \mathrm{P}(x))$-*irregular* LDPC code.

We shall only consider codes without degree 1 nodes. As mentioned in Section 2.1, check nodes of degree 1 serve no purpose. As for variable nodes of degree 1, if two of them are connected to the same check node, then the code

will contain words of weight 2. In general, it is possible to allow variable nodes of degree 1, as long as they are placed with care.

Example 3.3. The variable degree distribution of the [7,4,3] Hamming code from Example 2.2 is $\Lambda(x) = x^3 + 3x^2 + 3x$ and the check degree distribution is $\mathrm{P}(x) = 3x^4$.

The length of a $(\Lambda(x), \mathrm{P}(x))$-irregular LDPC code is $\sum_i \Lambda_i = \Lambda(1)$ and the number of check nodes is $\sum_i \mathrm{P}_i = \mathrm{P}(1)$. The design rate therefore satisfies

$$1 - R = \frac{\mathrm{P}(1)}{\Lambda(1)}.$$

The number of edges in the Tanner graph can be expressed as $\sum_i i\Lambda_i = \Lambda'(1)$ or equivalently $\sum_i i\mathrm{P}_i = \mathrm{P}'(1)$.

Sometimes it is convenient to work with the *normalized* degree distributions

$$L(x) = \frac{\Lambda(x)}{\Lambda(1)} \quad \text{and} \quad R(x) = \frac{\mathrm{P}(x)}{\mathrm{P}(1)}.$$

Further define

$$\lambda(x) = \sum_i \lambda_i x^{i-1} = \frac{\Lambda'(x)}{\Lambda'(1)} = \frac{L'(x)}{L'(1)}, \tag{3.1}$$

$$\varrho(x) = \sum_i \varrho_i x^{i-1} = \frac{\mathrm{P}'(x)}{\mathrm{P}'(1)} = \frac{R'(x)}{R'(1)}. \tag{3.2}$$

If E is the total number of edges in the Tanner graph, then $\lambda_i = i\Lambda_i/E$. In other words λ_i is the fraction of edges that are incident to a variable node of degree i. The same applies to ϱ_i in relation to the check nodes. Note that $\lambda(0) = 0 = \varrho(0)$, since we assume that there are no nodes of degree 1. Integrating (3.1) and (3.2) gives

$$\frac{\Lambda(x)}{\Lambda'(1)} = \int_0^x \lambda(z)\,\mathrm{d}z, \qquad \frac{\mathrm{P}(x)}{\mathrm{P}'(1)} = \int_0^x \varrho(z)\,\mathrm{d}z.$$

We can now express the average variable degree l_{avg} and the average check degree r_{avg} as

$$l_{\text{avg}} = \frac{\Lambda'(1)}{\Lambda(1)} = L'(1) = \frac{1}{\int_0^1 \lambda(z)\,\mathrm{d}z}, \tag{3.3}$$

$$r_{\text{avg}} = \frac{\mathrm{P}'(1)}{\mathrm{P}(1)} = R'(1) = \frac{1}{\int_0^1 \varrho(z)\,\mathrm{d}z}. \tag{3.4}$$

The design rate can also be expressed in terms of the average variable and check degrees as

$$1 - R = \frac{l_{\text{avg}}}{r_{\text{avg}}}.$$

The average degree d of all the nodes in the Tanner graph is

$$d = \frac{\Lambda'(1) + \mathrm{P}'(1)}{\Lambda(1) + \mathrm{P}(1)} = 2\frac{l_{\text{avg}} r_{\text{avg}}}{l_{\text{avg}} + r_{\text{avg}}} = \frac{2}{\int_0^1 \lambda(z) + \varrho(z)\,\mathrm{d}z}.$$

3.1 The girth of Tanner graphs of LDPC codes

As we saw in Example 2.4, the presence of small cycles in the Tanner graph is undesirable. This applies not only on the BEC, but on other channels as well. Whenever there is a cycle, any errors in the corresponding bits will effectively propagate through the cycle back to themselves, thereby reinforcing themselves. However, as we saw in Section 2.7, the complete absence of cycles in the Tanner graph is also undesirable, as the corresponding codes contain low-weight codewords. As a compromise, we seek codes that have a Tanner graph with large girth. The *girth* of a graph is defined as the length of the shortest cycle in the graph.

Consider the Tanner graph of an (l, r)-regular LDPC code. Denote the girth of the Tanner graph by g. Since the graph is bipartite, the girth is even. For any variable node x, the ball of radius $g/2 - 1$ around x is a tree. Designate x as the root of this tree. The root has degree l, the nodes of odd depth in the tree have degree $r-1$, and the nodes of even depth have degree $l-1$. The number of nodes at every depth, starting with depth 0 is as follows: 1, l, $l(r-1)$, $l(r-1)(l-1)$, $l(r-1)(l-1)(r-1)$, ... The total number of variable nodes in the tree is

$$1 + l(r-1) \sum_{i=0}^{\lfloor g/4 \rfloor - 1} ((l-1)(r-1))^i = 1 + l(r-1) \frac{((l-1)(r-1))^{\lfloor g/4 \rfloor} - 1}{(l-1)(r-1) - 1}. \quad (3.5)$$

This gives us a lower bound on the number of variable nodes in the Tanner graph. We can use it to obtain an upper bound on the girth. For example, for a $(3, 6)$-regular LDPC code of length 2048 we find that $g \leq 14$. We see that for fixed l and r, the upper bound on g is of the order of $\ln n$.

For irregular LDPC codes, we can utilize a lower bound on the total number of nodes in a graph of girth g and average degree $d > 2$ given by Alon, Hoory and Linial in [1]. For a Tanner graph, their result translates to

$$n(2 - R) \geq 2 \frac{(d-1)^{g/2} - 1}{d - 2}.$$

This gives the following upper bound on the girth of the graph

$$g \leq \frac{2 \ln(\frac{1}{2} n(2 - R)(d - 2) + 1)}{\ln(d - 1)}. \quad (3.6)$$

For a $(3, 6)$-regular LDPC code the average degree is 4, so the bound would be $g \leq 2 \log_3(1.5n + 1)$. When $n = 2048$ we obtain the same upper bound as that given by the first estimate $g \leq 14$.

In [11] it was shown that for any given degree pair (l, r) and any $\ell \in \mathbb{N}$, the probability that the depth-ℓ neighborhood of a randomly chosen node x in a randomly chosen (l, r)-regular LDPC code is a tree, goes to 1 as the length of the code goes to infinity. This result was then used to show that as the length of an LDPC code increases, the behavior of the sum-product algorithm converges to the behavior it exhibits on cycle-free graphs.

3.2 Performance of LDPC codes on the BEC

We analyze the performance of the sum-product algorithm for large random LDPC codes on the BEC(ε). This is one of the few scenarios in which exact analysis is possible. However, the results obtained here are qualitatively indicative of what happens in more general scenarios. Consider a $(\Lambda(x), \mathrm{P}(x))$-irregular LDPC code. We shall track the progress of the algorithm by the expected fraction of ? messages passed between check nodes and variable nodes during every iteration. When we speak of the degree of a variable node, we mean its degree in the Tanner graph. Denote by p the probability that a message passed from a variable node to a check node is ?, and by q the probability that a message passed from a check node to a variable node is ?. We will assume that at each node, the incoming messages are independent of each other. This assumption is true only during the first $g/2$ iterations, where g is the girth of the Tanner graph. We justify this assumption by the sparseness and randomness of the parity-check matrix, and by the large length of the code. These factors should insure reasonably large girth.

Since the sum-product algorithm is initialized by sending the message ? from each variable node to the neighbouring check nodes, the initial value of p is 1.

The probability that the outgoing message from a check node of degree d along one of the edges is ? is equal to the probability that at least one of the incoming messages along the other $d-1$ edges is ?. Assuming that the values of the incoming messages are independent events, the probability that none of the $d-1$ incoming messages is ? is equal to $(1-p)^{d-1}$. Therefore the probability that at least one them is ? is equal to $1-(1-p)^{d-1}$. The expected number of outgoing ? messages from check nodes of degree d is $d\mathrm{P}_d(1-(1-p)^{d-1})$, so the expected fraction of ? messages passed from all check nodes to variable nodes is

$$\begin{aligned} q &= \frac{1}{E}\sum_d d\mathrm{P}_d(1-(1-p)^{d-1}) = \frac{\sum_d d\mathrm{P}_d}{E} - \sum_d \frac{d\mathrm{P}_d}{E}(1-p)^{d-1} \\ &= 1 - \sum_d \varrho_d (1-p)^{d-1} = 1 - \varrho(1-p), \end{aligned}$$

where E is the total number of edges in the Tanner graph.

The probability that the outgoing message from a variable node of degree d along one of the edges is ? is equal to the probability that all the incoming messages along the other $d-1$ edges as well as along the edge from the leaf factor node are ?. Assuming that the values of the incoming messages are independent events, the probability that they are all ? is εq^{d-1}. The expected number of outgoing ? messages from all variable nodes of degree d is $d\Lambda_d \varepsilon q^{d-1}$, so the expected fraction of ? messages passed from all variable nodes to check nodes is

$$p = \frac{1}{E}\sum_d d\Lambda_d \varepsilon q^{d-1} = \varepsilon \sum_d \lambda_d q^{d-1} = \varepsilon\lambda(q).$$

For an (l, r)-regular LDPC code the expressions simplify to

$$q = 1-(1-p)^{r-1} \quad \text{and} \quad p = \varepsilon q^{l-1}.$$

Figure 3.1 shows a plot of these two functions for a $(3, 4)$-regular LDPC code on a BEC with erasure probability 0.6. The dashed line indicates the progress of the decoding algorithm. The algorithm is initialized with $p = 1$. In the first iteration we get $q = 1$ and then $p = 0.6$. In the second iteration we get $q = 0.936$ and then $p \approx 0.526$, and so on. Both values eventually converge to 0, i.e. the decoding is successful. We call this diagram an extrinsic information transfer (EXIT) chart.

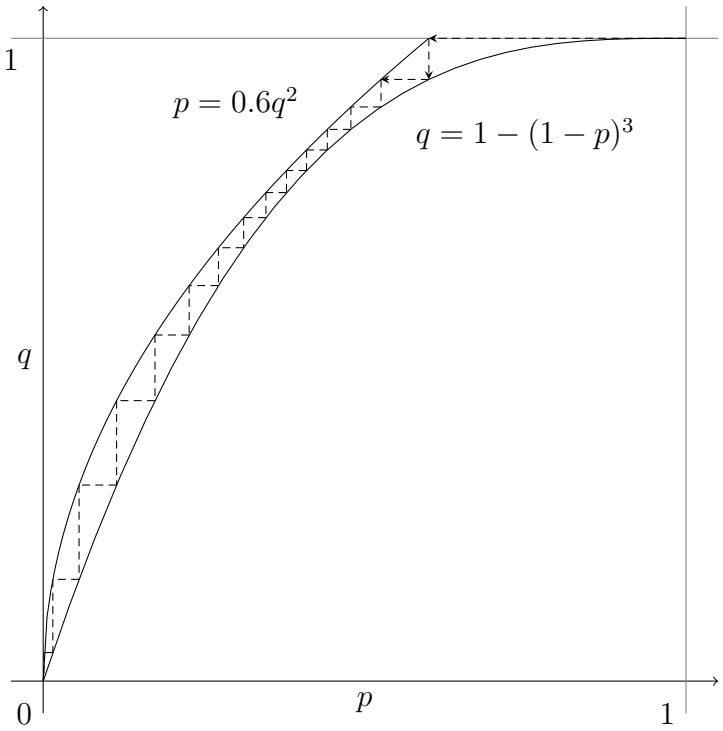


FIGURE 3.1: EXIT chart for the decoding of a (3,4)-regular LDPC code on a BEC(0.6).

For a (3,4)-regular LDPC code of length 2048 on a BEC(0.6) the expected number of ? messages evolves as follows (values rounded to the nearest integer):

[6144p]: 6144 3684 3171 2853 2591 2356 2109 1862 1608 1347 1064 755 424 156 22 2 0

[6144q]: 6144 5724 5445 5209 4984 4714 4363 4000 3584 3152 2614 1945 1162 454 62 6

And below we have an example of how the number of ? messages actually evolves for a randomly generated (3,4)-regular LDPC code of length 2048 on a BEC(0.6):

6144p: 6144 3686 3230 2942 2716 2517 2326 2129 1916 1676 1396 1069 702 343 92 7 0

6144q: 6144 5751 5488 5274 5077 4880 4670 4430 4142 3781 3308 2681 1874 973 273 22

One complete iteration of the algorithm can be expressed in terms of p as

$$p_{i+1} = \varepsilon\lambda(1 - \varrho(1 - p_i)). \tag{3.7}$$

For a regular code this becomes

$$p_{i+1} = \varepsilon(1 - (1 - p_i)^{r-1})^{l-1}.$$

Lemma 3.4. *For a given degree distribution pair* (λ, ϱ) *define*

$$f(\varepsilon, p) = \varepsilon\lambda(1 - \varrho(1 - p)).$$

Then for $p, \varepsilon \in [0, 1]$ *the function* $f(\varepsilon, p)$ *is increasing in both arguments and* $0 \leq f(\varepsilon, p) \leq \varepsilon$.

Proof. The coeficients of ϱ are nonnegative, therefore $\varrho(1 - p)$ is decreasing and $0 \leq \varrho(1 - p) \leq \varrho(1) = 1$ for $p \in [0, 1]$. Since the coeficients of λ are also nonnegative, $\lambda(1 - \varrho(1 - p))$ is increasing and nonnegative for $p \in [0, 1]$. Finally $\lambda(1 - \varrho(1 - p)) \leq \lambda(1) = 1$ for $p \in [0, 1]$, hence the upper bound on $f(\varepsilon, p)$. □

Lemma 3.5. *Let* $p_0, \varepsilon \in [0, 1]$. *For* $i \in \mathbb{N}$, *define* $p_i = f(\varepsilon, p_{i-1})$. *Then* $\{p_i\}_{i\in\mathbb{N}_0}$ *is a monotone sequence converging to some* $p_\infty \in [0, \varepsilon]$. *This* p_∞ *is a solution of the equation* $p = f(\varepsilon, p)$, *and it is the nearest solution to* p_0 *(in the direction of monotonicity).*

Proof. If $\varepsilon = 0$ or $p_0 = 0$, then $p_i = 0$ for all $i \in \mathbb{N}$ and the solution is 0. If $p_i \leq p_{i-1}$, then $p_{i+1} = f(\varepsilon, p_i) \leq f(\varepsilon, p_{i-1}) = p_i$, and the same is true if the inequalities are reversed. Hence the monotone nature of the sequence $\{p_i\}_{i\in\mathbb{N}_0}$.

The monotonicity of $\{p_i\}_{i\in\mathbb{N}_0}$ and the fact that it is bouded by 0 and ε implies that it converges to some $p_\infty \in [0, \varepsilon]$. Since f is continuous, p_∞ is a solution of the equation $p = f(\varepsilon, p)$:

$$p_\infty = \lim_{i\to\infty} p_i = \lim_{i\to\infty} f(\varepsilon, p_{i-1}) = f(\varepsilon, \lim_{i\to\infty} p_{i-1}) = f(\varepsilon, p_\infty).$$

It remains to be shown that this is the nearest solution. Assuming the sequence is decreasing, let p' be the nearest solution such that $p' \leq p_0$. For every $i \in \mathbb{N}$, $p' \leq p_{i-1}$ implies $p' = f(\varepsilon, p') \leq f(\varepsilon, p_{i-1}) = p_i$, thus $p' \leq p_\infty \leq p_0$ and since p' is the nearest solution, equality follows. If the sequence is increasing, then let p' be the nearest solution such that $p' \geq p_0$ and reverse the inequalities to get the corresponding result. □

The decoding algorithm is initialized with $p_0 = 1$. What this lemma tells us is that if $p' = f(\varepsilon, p')$ for some $p' \in (0, 1]$, then the decoding algorithm will converge to $p = p' > 0$. The decoding is therefore expected to be unsuccessful. Such a point p' manifests itself in an EXIT chart as an intersection of the two curves. The EXIT chart in Figure 3.2 shows this for a $(3, 4)$-regular LDPC code on a BEC with erasure probability 0.65. The dashed line indicates the progress of the decoding algorithm as it converges to $p \approx 0.481$ and $q \approx 0.860$.

Even if p and q do not converge to 0, the algorithm is likely to have recovered a portion of the erased bits. The probability that a bit corresponding to a variable node of degree d fails to decode is equal to the probability that all the incoming messages along the d edges as well as along the edge from the leaf factor node are ?. Assuming that the values of the incoming messages are independent events, the probability that they are all ? is εq^d. The expected fraction of bits that fail to decode is therefore

$$\frac{1}{n}\sum_d \Lambda_d \varepsilon q^d = \varepsilon L(q).$$

Generally we wish to know the highest erasure probability, below which decoding is expected to succeed. This value is called the threshold.

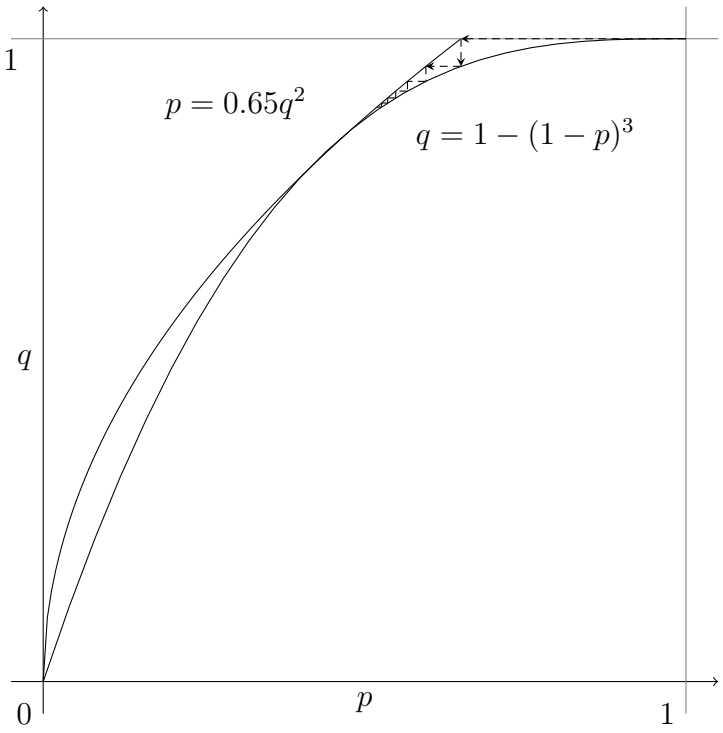


FIGURE 3.2: EXIT chart for the decoding of a (3,4)-regular LDPC code on a BEC(0.65).

Definition 3.6. The *threshold* associated with the degree distribution pair (λ, ϱ), call it $\varepsilon(\lambda, \varrho)$, is defined as

$$\varepsilon(\lambda, \varrho) = \inf\{\, \varepsilon \in [0,1] \mid p = f(\varepsilon, p) \text{ has a solution } p \in (0,1] \,\},$$

where $f(\varepsilon, p) = \varepsilon\lambda(1 - \varrho(1-p))$.

It is also possible to define the threshold as the lowest erasure probability, above which decoding is expected to fail.

Theorem 3.7. *For a given degree distribution pair* (λ, ϱ)

$$\varepsilon(\lambda, \varrho) = \sup\{\, \varepsilon \in [0,1] \mid p = f(\varepsilon, p) \textit{ has no solution } p \in (0,1] \,\}.$$

Proof. It suffices to prove that for any $\varepsilon, \varepsilon' \in [0,1]$ such that $\varepsilon' \leq \varepsilon$, if $p = f(\varepsilon, p)$ has no solution $p \in (0,1]$, then $p' = f(\varepsilon', p')$ has no solution $p' \in (0,1]$.

Let $p_0(\varepsilon) = 1$ and for $i \in \mathbb{N}$ define $p_i(\varepsilon) = f(\varepsilon, p_{i-1}(\varepsilon))$. Then by Lemma 3.5 $\lim_{i\to\infty} p_i(\varepsilon) = 0$, since the only solution of $p = f(\varepsilon, p)$ on $[0,1]$ is $p = 0$.

For every $i \in \mathbb{N}$, $p_i(\varepsilon') \leq p_i(\varepsilon)$ implies

$$p_{i+1}(\varepsilon') = f(\varepsilon', p_i(\varepsilon')) \leq f(\varepsilon, p_i(\varepsilon)) = p_{i+1}(\varepsilon).$$

Hence $\lim_{i\to\infty} p_i(\varepsilon') \leq 0$ and by Lemma 3.5 $p' = f(\varepsilon', p')$ has no solution on $(0,1]$. □

This theorem allows us to approximate $\varepsilon(\lambda, \varrho)$ numerically by doing a binary search on the interval $[0,1]$. This method was used to compute the thresholds

associated with some regular LDPC codes. Table 3.1 shows the results and compares them with the corresponding Shannon thresholds (the thresholds achievable with optimal codes under optimal decoding at the given rates). The exact values of some of the thresholds $\varepsilon(\lambda, \varrho)$ can be found in [2].

l	r	Design rate	$\varepsilon^{\mathrm{Sha}}(\lambda, \varrho)$	$\varepsilon(\lambda, \varrho)$
2	8	0.75	0.25	0.1429
3	12	0.75	0.25	0.2105
4	16	0.75	0.25	0.1931
2	6	0.6667	0.3333	0.2
3	9	0.6667	0.3333	0.2828
4	12	0.6667	0.3333	0.2571
2	4	0.5	0.5	0.3333
3	6	0.5	0.5	0.4294
4	8	0.5	0.5	0.3834
6	12	0.5	0.5	0.3075
2	3	0.3333	0.6667	0.5
4	6	0.3333	0.6667	0.5061
6	9	0.3333	0.6667	0.4035
3	4	0.25	0.75	0.6474
6	8	0.25	0.75	0.4499
9	12	0.25	0.75	0.3483

TABLE 3.1: The $\varepsilon(\lambda, \varrho)$ thresholds and the Shannon thresholds $\varepsilon^{\mathrm{Sha}}(\lambda, \varrho)$ associated with some (l, r)-regular LDPC codes. All values rounded to four decimal digits.

The threshold value for a $(3, 4)$-regular LDPC code is approximately 0.6474. In order to see how this theoretical estimate compares with empirical data, the decoding algorithm was simulated on some randomly generated codes. Table 3.2 shows how the probability of erasure ε affects the number of iterations of the message passing algorithm required to successfully decode a received word and the percentage of cases where the algorithm is successful. For every erasure probability 10000 measurements were taken on LDPC codes of length 2048 and 100 measurements were taken on LDPC codes of length 2^{21}.

For every measurement a random (3,4)-regular LDPC code of length 2048 was generated, together with a word that had random erasures with probability ε at every bit. The LDPC code was constructed by first generating a random permutation π of length $nl = 6144$ and then for every i, $1 \leq i \leq 1536$ prescribing the constraint equation

$$x_{\lfloor \pi(i*4)/3 \rfloor + 1} + x_{\lfloor \pi(i*4-1)/3 \rfloor + 1} + x_{\lfloor \pi(i*4-2)/3 \rfloor + 1} + x_{\lfloor \pi(i*4-3)/3 \rfloor + 1} = 0.$$

This essentially means that the variable nodes in the Tanner graph were randomly connected with the check nodes. However, this method does not rule out the possibility that a variable node is connected to a check node multiple times, thereby reducing the degree of the check node from 4 to 2 and the degree of the variable

node from 3 to 1. Unfortunately this happens in about 95 % of the randomly generated codes. On average 3 out of the 1536 check nodes are affected. When this happened, the code was discarded and a new one was generated instead.

The same method was used to generate the random (3,4)-regular LDPC codes of length $2^{21} = 2048 * 1024$.

For codes of length 2048 we see that the success rate of the decoding algorithm experiences a rapid decline on the interval $[0.61, 0.68]$, which is centered near the predicted threshold. For codes of length 2^{21} the success rate experiences a swift decline on a much narrower interval $[0.6465, 0.6480]$, which is again centered near the predicted threshold. The presence of these intervals of decline as opposed to a strict single-point threshold is most likely due to two reasons. Firstly, the independence assumption stated at the beginning of this section is not entirely valid, especially not for codes of small length. The second reason is statistical dispersion. For example, in order to simulate the behaviour of the BEC, erasures occur randomly with probability ε at every bit. The fraction of erasures in the words that are being decoded is therefore generally not equal to ε. As the length of the code increases, the effects of these phenomena diminish. The experimental results are in accordance with the predicted threshold.

The results also show that as the erasure probability nears the threshold the number of iterations of the decoding algorithm required to successfully decode a word grows rapidly. This is to be expected, since increasing the erasure probability brings the curves in the EXIT chart closer together. Looking at the chart in Figure 3.1 we see that the number of iterations is significantly greater in the narrow region between the two curves than in the regions where the curves are further apart.

For any given erasure probability, the mean number of iterations required to successfully decode a word does not vary significantly between the codes of length 2048 and 2^{21}, with the exception of the area near the threshold. When the erasure probability is well below the threshold, the codewords are recovered early on in the algorithm, therefore the length of the code has little effect on the number of iterations. But as we approach the threshold, the number of iterations approaches maximum, which is approximately half the diameter of the Tanner graph. The longer codes have a Tanner graph of greater diameter than the shorter codes, which explains the difference between the two near the threshold.

3.3 Stability condition

In order for the two curves in an EXIT chart not to cross on the interval $(0, 1]$, the curve $p = \varepsilon\lambda(q)$ must lie above the curve $q = 1 - \varrho(1 - p)$. Consider the two curves in an EXIT chart near the point $(0, 0)$. If the derivative of the first one with respect to p at $(0, 0)$ is less than the derivative of the second one with respect to p at $(0, 0)$, then the curves must cross on $(0, 1]$. Thus we have the following necessary condition for the decoding to be successful:

$$\frac{1}{\varepsilon\lambda'(0)} \geq \varrho'(1).$$

This is expressed in the following theorem.

Theorem 3.8. *Let $\varepsilon \in [0,1]$ and let (λ, ϱ) be a degree distribution pair. If $\varepsilon\lambda'(0)\varrho'(1) > 1$, then the equation $p = \varepsilon\lambda(1-\varrho(1-p))$ has a solution $p \in (0,1]$.*

Proof. For $i \in \mathbb{N}$ define

$$p_i = \varepsilon\lambda(1-\varrho(1-p_{i-1})).$$

Expanding the right side into a Taylor series around 0 we get

$$p_i = \varepsilon\lambda'(0)\varrho'(1)p_{i-1} + O(p_{i-1}^2).$$

If $\varepsilon\lambda'(0)\varrho'(1) > 1$, then for some sufficiently small $p_0 \in (0,1]$ we have $p_1 > p_0$. By Lemma 3.5 the sequence $\{p_i\}_{i\in\mathbb{N}_0}$ converges to some $p_\infty \in (0,1]$, which is a solution of the equation. □

Corollary 3.9 (Stability condition)**.** *For any degree distribution pair (λ, ϱ)*

$$\varepsilon(\lambda,\varrho) \leq \frac{1}{\lambda'(0)\varrho'(1)} = \frac{1}{\lambda_2\varrho'(1)}.$$

3.4 Approaching channel capacity on the BEC

Consider the areas under the curves $p = \varepsilon\lambda(q)$ and $q = 1-\varrho(1-p)$ in an EXIT chart. If the sum of these two areas is greater than or equal to the area of the EXIT chart, which is 1, then the curves must cross somewhere on the interval $(0,1]$. We find the area under the first curve by applying (3.3)

$$\int_0^1 \varepsilon\lambda(x)\,\mathrm{d}x = \frac{\varepsilon}{l_{\text{avg}}}.$$

The area under the second curve can be obtained by substituting y for $1-x$ in the following integral and by applying (3.4)

$$\int_0^1 1-\varrho(1-x)\,\mathrm{d}x = -\int_1^0 1-\varrho(y)\,\mathrm{d}y = \int_0^1 1-\varrho(y)\,\mathrm{d}y = 1-\frac{1}{r_{\text{avg}}}.$$

Thus we have another necessary condition for the decoding to be successful

$$\frac{\varepsilon}{l_{\text{avg}}} + 1 - \frac{1}{r_{\text{avg}}} < 1.$$

Expressed in terms of design rate R we have $\varepsilon < l_{\text{avg}}/r_{\text{avg}} = 1-R$ or equivalently $R < 1-\varepsilon = C_{\text{BEC}}(\varepsilon)$. This implies that as R nears the capacity of the channel, the sum of the two areas nears 1 and the two curves come closer together, but must not touch. Approaching channel capacity on the BEC is therefore a matter of choosing degree distributions that fulfill this criterion.

In [9] Luby, Shokrollahi *et al.* designed a family of irregular LDPC codes that allow transmission at rates arbitrarily close to channel capacity on the BEC. For any $\delta > 0$, they describe degree distributions that yield codes which have rate $1-\varepsilon(1+\delta)$ and threshold at least ε. The degree distribution pair $(\lambda(x), \varrho(x))$ is created by taking high order Taylor polynomials of $-\ln(1-x)$ and $\mathrm{e}^{\alpha(x-1)}$, and

scaling them suitably. The resulting Tanner graph has some variable nodes of degree $1/\delta$, but the average variable node degree is only $\ln(1/\delta)$.

The authors also describe a method based on linear programming that has proven effective in finding a good ϱ for a given λ and vice versa. They start out with a pair of equivalent conditions

$$1 - \varrho(1 - \varepsilon\lambda(q)) < q \quad \text{for all } q \in (0, 1], \tag{3.8}$$
$$\varepsilon\lambda(1 - \varrho(1 - p)) < p \quad \text{for all } p \in (0, 1]. \tag{3.9}$$

Each of these conditions implies that the equation $\varepsilon\lambda(1 - \varrho(1 - p)) = p$ has no solution on $(0, 1]$ and the decoding is therefore expected to be successful. Given λ, one chooses which of the coeficients of ϱ are to be nonzero. Then based on the condition (3.8) several linear constraints on the coefficients of ϱ are produced by choosing q to be multiples of $1/N$, for some suitably large N. The function to be minimized is $q - (1 - \varrho(1 - \varepsilon\lambda(q)))$, which effectively means that the distance between the curves in the EXIT chart is being minimized. Certain numerical problems can be avoided by including the stability condition $\lambda_2 \leq 1/(\varepsilon\varrho'(1))$.

After obtaining the solution ϱ of the linear programming problem, the process is repeated the other way around, finding good λ for the obtained ϱ. This time the linear constraints being based on (3.9). After obtaining the new λ, the process is repeated again to obtain good ϱ and so on.

The authors give the following example of a degree distribution they found using this method

$$\begin{aligned}\lambda(x) &= 0.430034x^2 + 0.237331x^{12} + 0.007979x^{13} + 0.119493x^{47} + 0.052153x^{48} + \\ &\quad + 0.079630x^{161} + 0.073380x^{162}, \\ \varrho(x) &= 0.713788x^9 + 0.122494x^{10} + 0.163718x^{199}.\end{aligned} \tag{3.10}$$

This degree distribution yields a code of rate 0.5 with average variable node degree $l_{\text{avg}} = 6$ and average check node degree $r_{\text{avg}} = 12$. A binary search for the threshold on the interval $[0, 1]$ gives $\varepsilon(\lambda, \varrho) \approx 0.49563$. In comparison, the thresholds of the regular LDPC codes of rate 0.5 shown in Table 3.1, do not exceed 0.4294, which is the threshold for $(3, 6)$-regular LDPC codes.

Figure 3.3 shows the EXIT chart of this degree distribution for $\varepsilon = 0.4$. If we plot the two curves for the threshold value $\varepsilon = 0.49563$, they become visually indistinguishable, and the sum of the areas under the curves comes out to 0.99927.

In order to see how the theoretical threshold compares with empirical data, the decoding was again simulated on some randomly generated codes. The graph in Figure 3.4 shows the success rate of this degree distribution for codes of length 2048 and codes of length 2^{21}, and for comparison the success rate of $(3, 6)$-regular and $(6, 12)$-regular LDPC codes of length 2048. For each erasure probability and for each type of code 10000 measurements were taken on randomly generated codes of length 2048 and 100 measurements were taken on the irregular codes of length 2^{21}. These measurements, including the number of iterations required to successfully decode a word, can be found in Tables A.1, A.2 and A.3 in the appendix. The number of iterations does not vary significantly between the regular and irregular codes of length 2048.

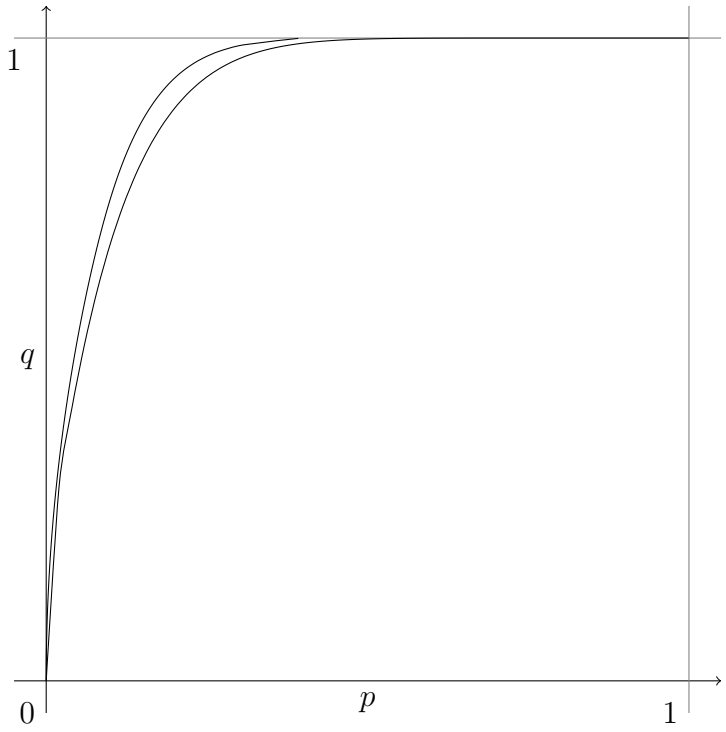


FIGURE 3.3: EXIT chart for the near capacity irregular LDPC code with degree distribution (3.10) on a BEC(0.4).

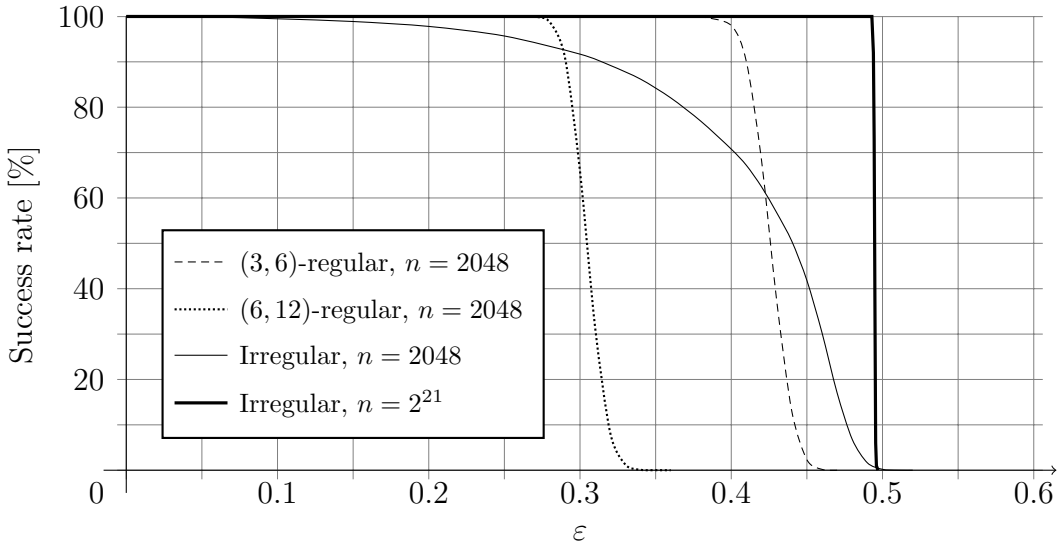


FIGURE 3.4: Success rate of the near capacity irregular LDPC code with degree distribution (3.10) on the BEC(ε) in comparison with $(3, 6)$-regular and $(6, 12)$-regular LDPC codes.

For length 2048, we notice that the irregular codes perform better than the $(3, 6)$-regular codes only on the interval $[0.42, 0.50]$. Their performance on the whole is quite disappointing. The success rate of the irregular codes begins to drop as early as $\varepsilon = 0.15$, whereas the success rate of the $(3, 6)$-regular codes begins to drop around $\varepsilon = 0.4$. For low erasure probabilities, the irregular codes are even outperformed by $(6, 12)$-regular codes. The poor performance of the irregular codes of length 2048 is probably caused by the presence of cycles of small length in the Tanner graph. The existence of such cycles is more likely in the irregular codes than in the regular codes, because the irregular codes contain nodes of very high degree: Approximately 6 variable nodes of degree 163, 6 variable nodes of degree 162 and 10 check nodes of degree 200. In light of what is discussed in Section 3.1, increasing the length of the code should make small cycles less likely. We see that increasing the length of the irregular codes to 2^{21} does indeed remedy their performance. The success rate of the longer codes begins to drop at $\varepsilon = 0.494$, which is on the verge of the predicted threshold.

3.5 Notes

In addition to the cited sources, the book *Modern Coding Theory* [13] by Richardson and Urbanke, and Chapter 13 of the lecture notes for Principles of Digital Communication II [6] by David Forney were used throughout this chapter. All empirical results were obtained by the author.

	Length 2048		Length 2^{21}	
Probability of erasure ε	Number of iterations*	Success rate	Number of iterations*	Success rate
0.20	4.2 (0.4)	100.00 %	5.0 (0.1)	100 %
0.25	4.8 (0.4)	100.00 %	5.9 (0.3)	100 %
0.30	5.2 (0.4)	99.99 %	6.0 (0.0)	100 %
0.35	5.9 (0.4)	99.99 %	7.0 (0.0)	100 %
0.40	6.6 (0.5)	99.99 %	7.6 (0.5)	100 %
0.45	7.6 (0.5)	99.96 %	8.5 (0.5)	100 %
0.50	8.9 (0.6)	99.96 %	10.0 (0.0)	100 %
0.55	11.2 (0.9)	99.94 %	12.0 (0.0)	100 %
0.56	11.9 (1.0)	99.93 %	13.0 (0.1)	100 %
0.57	12.6 (1.2)	99.93 %	13.3 (0.5)	100 %
0.58	13.6 (1.4)	99.93 %	14.2 (0.4)	100 %
0.59	14.9 (1.8)	99.93 %	15.6 (0.5)	100 %
0.60	16.5 (2.6)	99.89 %	17.0 (0.1)	100 %
0.61	19.0 (4.2)	99.61 %	19.0 (0.1)	100 %
0.62	22.8 (6.8)	97.55 %	22.0 (0.2)	100 %
0.63	28.3 (10.0)	87.48 %	27.4 (0.5)	100 %
0.64	34.5 (12.4)	62.84 %	41.5 (1.1)	100 %
0.6460	38.1 (13.6)	42.43 %	97.5 (16.7)	100 %
0.6465	38.4 (13.7)	40.75 %	130.8 (48.5)	99 %
0.6470	38.5 (13.5)	38.92 %	183.5 (95.6)	80 %
0.6475	38.7 (13.5)	37.36 %	262.7 (125.4)	33 %
0.6480	39.0 (13.6)	35.83 %	308.0 (142.7)	3 %
0.6485	39.3 (13.8)	34.15 %	n/a (n/a)	0 %
0.65	40.2 (13.9)	29.59 %	n/a (n/a)	0 %
0.66	45.2 (15.0)	8.49 %	n/a (n/a)	0 %
0.67	48.5 (13.6)	1.51 %	n/a (n/a)	0 %
0.68	62.7 (18.1)	0.07 %	n/a (n/a)	0 %
0.69	n/a (n/a)	0.00 %	n/a (n/a)	0 %

*Mean value with sample standard deviation in parentheses.

TABLE 3.2: Decoding (3,4)-regular LDPC codes on a BEC(ε).

Chapter 4

Encoding LDPC codes

The most straightforward method of encoding an LDPC code, given its $m \times n$ parity-check matrix H, is to precompute the generator matrix G using Gaussian elimination, and then multiply each message block by G. Generally, G will not be a sparse matrix, and the multiplication will therefore require time $O(n(n-m))$, which is $O(n^2)$, assuming constant rate.

Ideally, we would like to perform the task in time $O(n)$. There are two possible approaches that can be taken. One is to design a general algorithm that works efficiently on any LDPC code. The other is to design LDPC codes that admit an efficient encoding algorithm. We start with a method introduced by Richardson and Urbanke, which uses the first approach.

4.1 The general case

Given a sparse parity-check matrix over $\mathbb{F}_2$ in upper triangular form with ones on the diagonal, the encoding of a message into a codeword x can be performed by filling the last $n-m$ entries of x with the message bits and solving for the remaining entries by back substitution. We start by solving for x_m, which can be done in time $O(1)$, as the matrix is sparse. We continue with x_{m-1} and so on, taking time $O(m) = O(n)$ to perform the whole task.

Given a sparse parity check matrix that is not in upper triangular form, we could convert it by Gaussian elimination and column permutation, but the resulting matrix would generally not be sparse. We therefore need to choose a more cautious approach. Instead of using algebraic operations to perform the conversion to upper triangular form, we limit ourselves to row and column permutations. These operations are generally not powerful enough to yield a matrix in the desired form, so the best we can do is obtain a matrix in *approximate upper triangular form* as shown in Figure 4.1, where the size of the gap g is as small as possible. We shall assume throughout this section that we are given a parity-check matrix that has full rank.

Split the vector x into vectors p_1, p_2 and s of length $m-g$, g and $n-m$, respectively. Vectors p_1 and p_2 contain the parity bits of x, while s contains the message bits. Assuming we have the parity-check matrix in approximate upper triangular form, we could finish the conversion to full upper triangular

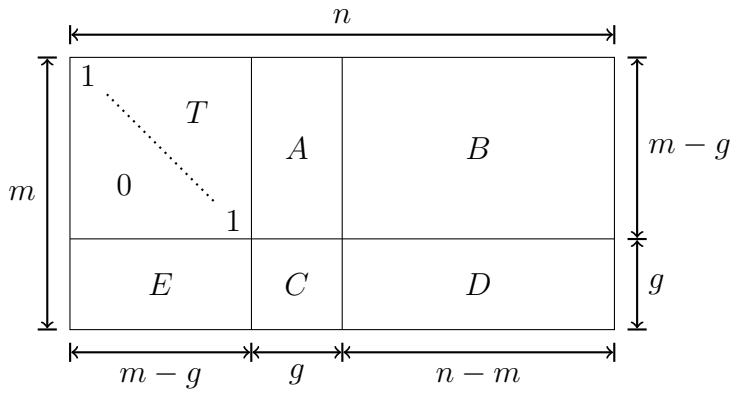


FIGURE 4.1: Parity-check matrix H in approximate upper triangular form.

form by Gaussian elimination and column permutation, obtaining a matrix with $E = 0$ and C in upper triangular form, with ones on the diagonal. However, the submatrices C and D would not be sparse after this operation, and the computation of p_2 by back substitution would therefore require time $O(ng)$. The remaining parity bits in p_1 would then be computed as before, in time $O(n)$. We will show that it is possible to improve upon this result by computing p_2 in time $O(n + g^2)$.

Assume that the first m columns of the parity-check matrix are linearly independent. If not, then we can permute the columns of H to make the first m linearly independent. We eliminate the submatrix E by multiplying H from the left by a regular matrix:

$$\begin{pmatrix} I & 0 \\ -ET^{-1} & I \end{pmatrix} \begin{pmatrix} T & A & B \\ E & C & D \end{pmatrix} = \begin{pmatrix} T & A & B \\ 0 & C - ET^{-1}A & D - ET^{-1}B \end{pmatrix}.$$

Define $C' = C - ET^{-1}A$, this matrix is regular. This can be seen as follows. The submatrix $\left(\begin{smallmatrix} T & A \\ E & C \end{smallmatrix}\right)$ of H is regular by assumption, and we multiplied from the left by a regular matrix, thus the submatrix $\left(\begin{smallmatrix} T & A \\ 0 & C' \end{smallmatrix}\right)$ of the result is regular, its rows are linearly independent and therefore the rows of C' are linearly independent. In order to find p_2, we need to solve the equation $C'p_2^{\mathrm{T}} + (D - ET^{-1}B)s^{\mathrm{T}} = 0$, which translates to $p_2^{\mathrm{T}} = -C'^{-1}(D - ET^{-1}B)s^{\mathrm{T}}$. Since B is sparse, Bs^{T} can be computed in time $O(n)$. The multiplication by T^{-1} can also be performed in linear time, since its equivalent to solving $Ty^{\mathrm{T}} = Bs^{\mathrm{T}}$, which can be done by back substitution (T is upper triangular and sparse). Multiplication by the sparse matrices D and E, and the subtraction are all $O(n)$. Only multiplication by the $g \times g$ matrix C'^{-1} remains, and this requires time $O(g^2)$.

Assuming we have converted the parity-check matrix into approximate upper triangular form, such that the first m columns are linearly independent, and assuming we have precomputed C'^{-1}, the overall complexity of this encoding algorithm is $O(n + g^2)$.

Conversion into approximate upper triangular form

We now discuss a simple algorithm that brings the parity-check matrix H into approximate upper triangular form by performing row and column permutations.

We start with $H_0 = H$ and proceed in steps. The parity-check matrix after step t is denoted H_t. The size of the upper triangular submatrix T starts at 0×0 and increases by one row and one column in each step of the algorithm. The size of the gap g starts at 0 and increases as needed. The algorithm stops when $g + t = m$.

The submatrix of H_t consisting of rows $t+1, \ldots, m-g$ and columns $t+1, \ldots, n$ will be referred to as the *residual* parity-check matrix. At the beginning of the algorithm, the residual parity-check matrix is the whole matrix H. In each step, we find a column of the residual parity-check matrix that has minimum positive weight. Denote the position of this column with respect to H by j. Denote the positions of the non-zero entries in this column with respect to H by $i_1, \ldots, i_d$. Choose one of these non-zero entries, say i_1, and swap columns j and $t+1$ of H and rows i_1 and $t+1$ of H, so as to bring the non-zero entry onto the diagonal. Move the rows $i_2, \ldots, i_d$ that contain the remaining nonzero entries, to the bottom of the matrix. Increase t by 1 and increase g by $d-1$. Repeat, until $g + t = m$.

In [12, 13] Richardson and Urbanke show how to estimate the size of the gap left by this algorithm by solving a system of differential equations. For a $(3, 6)$-regular code their estimate is approximately $0.017n$. When the algorithm was applied to a randomly generated $(3, 6)$-regular LDPC code of length 2048, the gap came out to 39, which is indeed close to the estimate, $39/2048 \approx 0.019$. The number of operations required to encode one block is therefore $0.017^2 n^2 + O(n)$. Although the encoding complexity is quadratic in n, the small constant makes it well manageable even for n as large as 100000.

The authors also showed that codes that allow transmission close to capacity have gaps of order less than $\sqrt{n}$, with high probability. This makes the encoding complexity linear in n. In practice, the resulting gap for such codes is typically in the range of one to three, even for very large lengths like one million.

4.2 Codes with cascaded sparse Tanner graphs

In [9] Luby, Shokrollahi *et al.* introduce a class of codes with a simple linear time encoding algorithm. Their codes have the advantage that they can be designed to allow communication at rates arbitrarily close to channel capacity on a BEC, provided the length of the code is large enough.

Consider a sparse Tanner graph with k variable nodes and $n-k$ check nodes. We shall refer to its variable nodes as *message nodes*. To each of the $n-k$ check nodes, attach a new variable node of degree 1. We shall refer to these new variable nodes as *parity nodes*. Figure 4.2 shows an example of such a graph with the parity nodes placed on the right.

The encoding algorithm on such graphs is quite simple. Fill the first k entries of x with the message bits. Then for each check node, compute the mod-2 sum of the message bits, which are connected to that check node, and assign the result to the corresponding parity bit. Assuming the Tanner graph has $O(n)$ edges, this encoding algorithm requires $O(n)$ operations.

Assume that we transmit a codeword of this code over a BEC(ε), and that some message bit, say x_i, is erased in the process. Recall that under sum-product decoding, a check node can recover an erased variable node only if all the other

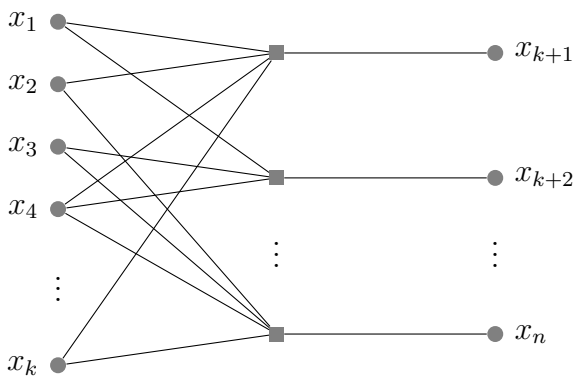


FIGURE 4.2: Tanner graph with added parity nodes.

variable nodes connected to that check node are unerased. The probability that a check node which is connected to x_i cannot recover the erased bit is at least ε (the probability that the corresponding parity bit is erased). The probability that none of the check nodes connected to x_i can recover the erased bit is at least ε^d, where d is the degree of the message node x_i. If we denote the fraction of message nodes that have degree d by L_d, then the probability that a randomly chosen message bit is erased and is unrecoverable under iterative decoding is at least $\varepsilon \sum_d L_d \varepsilon^d$. This result is independent of the length of the code. Therefore, if we maintain the message node degree distribution, this lower bound on the expected fraction of unrecovered bits will remain constant even for large n.

The problem lies in the fact that the parity nodes have degree 1. The erasure of a parity bit permanently impairs the ability of the corresponding check node to recover erased message bits. In order to overcome this problem, the parity bits also need to have the ability to be recovered in case of erasure. We therefore need to increase the degree of the parity nodes, while maintaining the efficiency of the encoding algorithm. The solution is to protect the parity bits in the same fashion as we protect the message bits, i.e. with another level of parity bits, and these are in turn protected by yet another level of parity bits and so on, forming a cascade of bipartite graphs. Figure 4.3 illustrates the scheme. At each level, the number of parity nodes decreases by a fraction β, i.e. the first level contains βk parity bits, the second βk^2, and so on. Finally, the last level is protected by a conventional code (e.g. a Reed-Solomon code or an LDPC code) of rate $1 - \beta$. The total number of parity bits then adds up to

$$\frac{\beta^N k}{1-\beta} + \sum_{i=1}^{N-1} \beta^i k = \frac{\beta^N k}{1-\beta} + k\left(\frac{\beta^N - 1}{\beta - 1} - 1\right) = \frac{\beta k}{1-\beta},$$

where N is the number of levels of parity bits. The number of levels N is chosen so that $\beta^N k \approx \sqrt{k}$. The resulting code will have rate

$$\frac{k}{n} = \frac{k}{k + \frac{\beta k}{1-\beta}} = 1 - \beta.$$

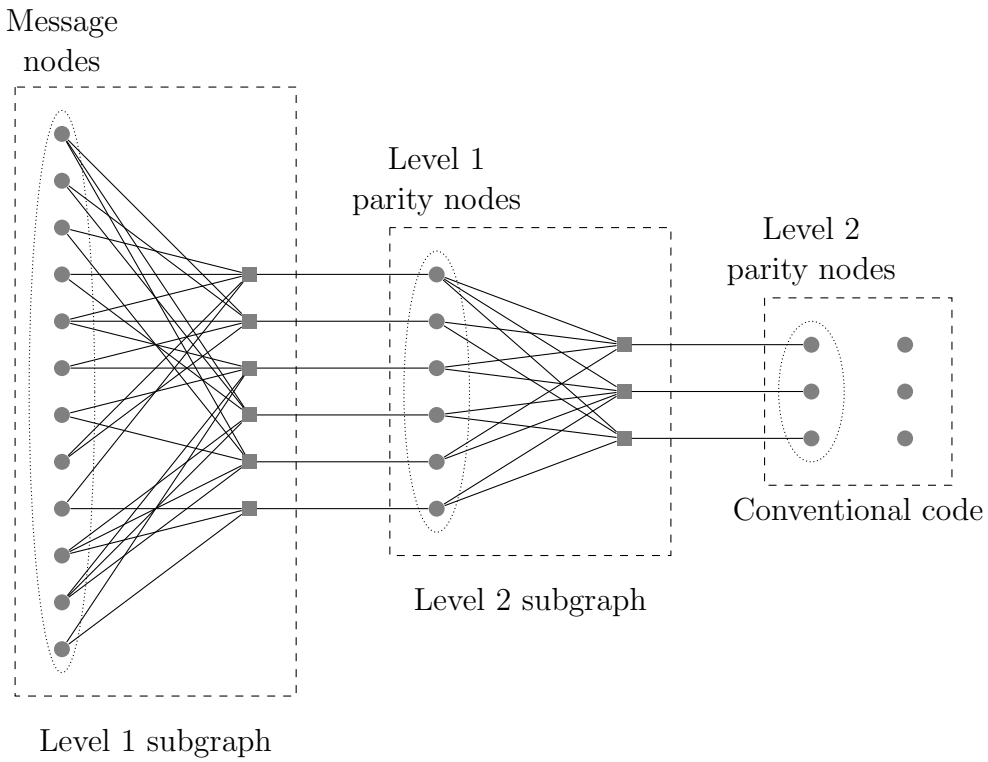


FIGURE 4.3: Cascaded Tanner graph with $k = 12$, $\beta = 0.5$ and $N = 2$.

The encoding now takes place separately at every level, starting from the left. At each level, parity bits are computed the same way as before. The results enter the next level, and are used to compute the next set of parity bits, and so on. The number of operations required to compute all parity bits up to level N is proportional to the number of edges in Tanner graph. The level N parity bits, of which there are $\beta^N k \approx \sqrt{k}$, are then encoded by the conventional code. Assuming that the conventional code can be encoded in quadratic time and that the total number of edges in the Tanner graph is $O(n)$, the encoding takes time $O((\sqrt{k})^2) + O(n)$, which is $O(n)$, assuming constant rate.

The question now is how to design the subgraphs that form the levels of the cascaded Tanner graph, so that the resulting code provides good performance on the BEC(ε). Recall that in Section 3.4 we discussed the design of degree distributions, which yield codes that allow communication at rates arbitrarily close to channel capacity on the BEC. This allows us to find a degree distribution, such that the corresponding LDPC code is capable of recovering any ε-fraction of erasures with high probability, and its rate R comes arbitrarily close to the channel capacity $1 - \varepsilon$. Let us construct the subgraphs of our cascade, so that each of them has such a degree distribution. The number of check nodes in any of the subgraphs is then $(1 - R)$ times the number of variable nodes in that subgraph, in other words $\beta = 1 - R$. For the conventional code, we use a code that can recover any ε-fraction of erasures with high probability. (A code with the required properties does exist, since its rate is $1 - \beta = R < 1 - \varepsilon = C_{\text{BEC}}(\varepsilon)$.)

Consider the following decoding algorithm for transmissions over BEC(ε). We proceed on a level by level basis, starting from the right. First, the conventional

code is used to recover the highest level parity bits, these are in turn used in recovering erased parity bits at the next lower level, and so on. The recovery operation taking place at each of the subgraphs is simply the greedy decoding algorithm discussed in Section 2.1. Assume that the conventional code recovered all the level N parity bits. Use the sum-product algorithm on the level N subgraph to recover any erased level $N-1$ parity bits. Since the level N parity bits are assumed to be known, they will have no impact on the success rate of the sum-product algorithm, and the algorithm will therefore recover any ε-fraction of erasures among the level $N-1$ parity bits with high probability. We can now assume that all the level $N-1$ parity bits have been recovered, and apply the argument inductively to decoding at the next lower level. Eventually, we reach the level 1 subgraph and we are able to recover the message bits with high probability.

A more extensive probabilistic analysis of the decoding process is conducted in [9]. The authors show that for any $\delta > 0$ this method can be used to produce codes of rate $1 - \varepsilon(1+\delta)$ that recover from any ε-fraction of erasures with high probability for sufficiently large n. The average variable node degree in each of the subgraphs is then $\ln(1/\delta)$ and so both the encoding and the decoding algorithm run in time $O(n \ln(1/\delta))$.

4.3 Repeat-accumulate codes

In [4] Divsalar, Jin and McEliece introduced a class of simple turbo-like codes which they call *repeat and accumulate* (RA) codes. For the purposes of encoding, these codes can be described as interleaved serially concatenated convolutional codes, and for the purposes of decoding, they can be viewed as LDPC codes, because they have a sparse parity check matrix. Figure 4.4 shows the encoding scheme. A message of length k is repeated q times, forming a vector of length qk. The entries of this vector are permuted and it is then encoded by a rate 1 accumulator. The output of the accumulator is then transmitted. The accumulator is a linear transformation, which maps $(x_1, x_2, \ldots, x_{qk})$ to $(y_1, y_2, \ldots, y_{qk})$, where

$$\begin{aligned} y_1 &= x_1, \\ y_2 &= x_1 + x_2, \\ y_3 &= x_1 + x_2 + x_3, \\ &\vdots \\ y_{qk} &= x_1 + x_2 + x_3 + \cdots + x_{qk}. \end{aligned}$$

Since repetition, permutation and accumulation are all linear transformations, the encoding operation is a linear mapping. The image of $\mathbb{F}_2^k$ under RA encoding is therefore a subspace of $\mathbb{F}_2^{qk}$. Thus RA codes are linear $[qk, k]$ codes of rate $1/q$. Their encoding time is linear in the length of the code, since the permutation requires no more than $O(qk)$ operations and the accumulator performs $qk - 1$ operations of mod-2 addition.

Figure 4.5 shows the Tanner graph of a very simple example of an RA code with $q = 3$ and $k = 2$. The variable nodes corresponding to the message bits have

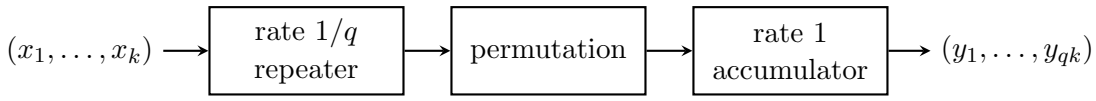


FIGURE 4.4: Encoder of a $[qk, k]$ repeat and accumulate code.

been placed on the left side of the Tanner graph, while the remaining variable nodes are placed on the right, we shall call these nodes *message nodes* and *parity nodes*, respectively. The arrangement of the parity nodes and check nodes acts as the accumulator. The input message (x_1, x_2) gets repeated 3 times yielding $(x_1, x_2, x_1, x_2, x_1, x_2)$. This is permuted to obtain the vector $(x_2, x_1, x_1, x_2, x_1, x_2)$, which is then accumulated to obtain the parity bits $(y_1, \ldots, y_6)$.

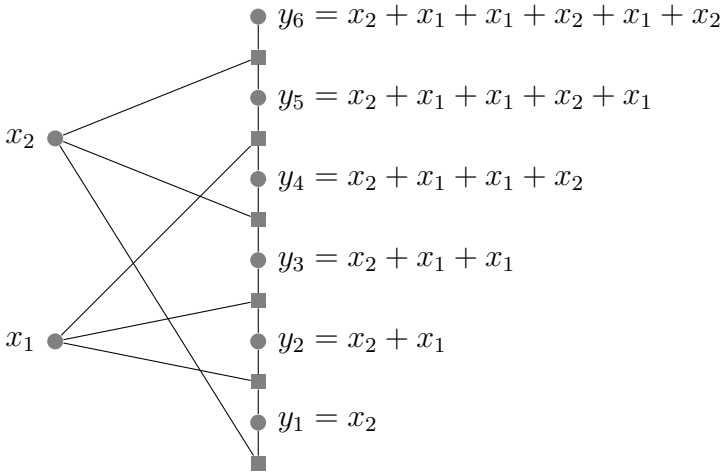


FIGURE 4.5: Tanner graph of a repeat and accumulate code with $q = 3$ and $k = 2$.

The Tanner graph of an RA code is sparse, since the number of edges is $qk + 2qk - 1 < 3qk = 3n$. Notice that by the above definition, only the parity bits $y_1, \ldots, y_{qk}$ get transmitted. We can think of the message bits $x_1, \ldots, x_k$ as being guaranteed to be erased. The sum-product decoding algorithm is then applied to recover the message bits.

RA codes which transmit both the message bits and the parity bits are called *systematic* RA codes. Systematic RA codes are $[(q+1)k, k]$ codes of rate $1/(q+1)$.

One of the drawbacks of these codes is that they are inherently low-rate, and that the range of possible rates is very limited $(1/2, 1/3, 1/4, \ldots)$. This problem can be overcome by deleting some of the parity bits in the encoder output. This procedure is known as *puncturing*. A common way to do this is for the encoder to output only every ath parity bit: $(y_a, y_{2a}, y_{3a}, \ldots)$. Figure 4.6 shows how the Tanner graph of the code from the previous example is changed, when we apply this approach with $a = 2$. What effectively happens is that every a check nodes are merged into a single check node of degree $a + 2$. We see that this method cannot be applied to non-systematic RA codes, because if all the message bits are erased, then each check node is connected to at least two erased nodes, and therefore the decoding algorithm cannot recover any information. The systematic RA codes that employ this puncturing method are $[k + qk/a, k]$ codes of rate

$a/(a+q)$. It was shown in [8] that on the BEC these RA codes can outperform randomly generated regular LDPC codes of equivalent length, rate and parity-check matrix density.

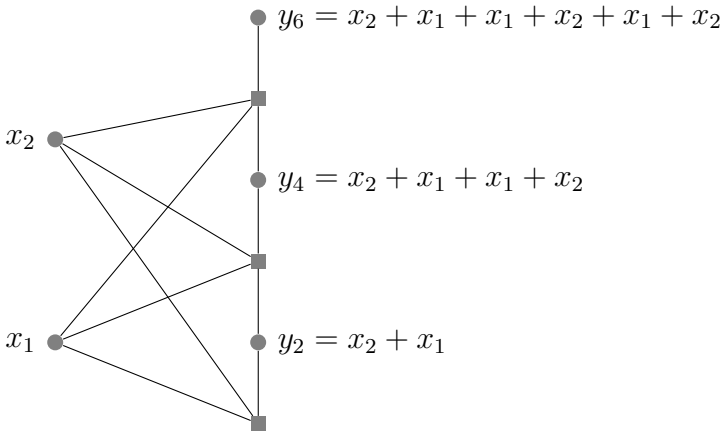


FIGURE 4.6: Tanner graph of a repeat and accumulate code with $q = 3$, $k = 2$ and $a = 2$.

The performance of these codes can be further improved by repeating some bits more often than others. That way, the message nodes in the Tanner graph will have an irregular degree distribution. Such codes are called *irregular repeat-accumulate* (IRA) codes. These codes were introduced in [7], where their performance on the BEC was studied. The authors showed how to choose for any $\varepsilon \in (0, 1)$ and any $a \geq 1$ the message node degree distribution, so that the code is able to correct any ε-fraction of erasures with high probability. The rate of the codes with such message node degree distribution can be brought arbitrarily close to the channel capacity $1 - \varepsilon$ by choosing a large enough value of a. Note that for $a > 1$, the codes have to be systematic due to the same reason as regular RA codes. The codes then have length $k + \bar{q}k/a$, where $\bar{q}$ is the average message node degree, and rate $1/(1 + \bar{q}/a)$. Since the rate is upper bounded by the capacity $1 - \varepsilon$ of the channel, we have $\bar{q} > a\varepsilon/(1 - \varepsilon)$. Increasing the value of a therefore increases the average message degree $\bar{q}$. But the encoding requires time $O(\bar{q}k)$, so increasing the value of a also increases the encoding time. The number of edges in the Tanner graph is $\bar{q}k + 2\bar{q}k/a - 1$, which is $O(\bar{q}k)$, so increasing the value of a increases the decoding time as well. Fortunately, experiments show that the value of a need not be very large. For example, the authors of [7] constructed a rate 0.4978 IRA code with $a = 5$ that has threshold 0.4929 on the BEC.

Conclusion

We have given a detailed discussion of the sum-product algorithm and proven its optimality on cycle-free Tanner graphs. We then showed that codes of rate greater than 0.5 that admit a cycle-free Tanner graph contain codewords of weight 2, which severely diminishes their error correcting ability and establishes the need for the study of codes that have cycles in their Tanner graph. The lower bound for the number of codewords of weight 2 was improved over previous results.

We have given a probabilistic analysis of the sum-product algorithm on the BEC, and used it to show how to determine the thresholds of LDPC codes on the BEC. The simulations presented in Chapter 3 confirmed that randomly generated LDPC codes do indeed allow reliable transmission over any BEC that has erasure probability below the predicted threshold, provided that the length of the code is sufficiently large. We discussed the design of irregular LDPC codes that allow transmission at rates arbitrarily close to channel capacity on the BEC. Based on simulations, we saw that these codes perform as expected, provided their length is sufficiently large, but give very poor results for low lengths, at which they may even be outperformed by comparable regular codes with low thresholds.

Finally, we have seen that the complex encoder problem of LDPC codes has been largely solved using two main approaches. Firstly, by an algorithm that exploits the sparseness of the parity-check matrix to obtain an efficient encoder. Secondly, by specifically designing codes that admit an efficient encoding algorithm.

Bibliography

[1] Alon, N., Hoory, S., and Linial, N. The Moore bound for irregular graphs. *Graphs Combin.* 2002, 18, pp. 53–57.

[2] Bazzi, L., Richardson, T. J., and Urbanke, R. L. Exact thresholds and optimal codes for the binary-symmetric channel and Gallager's decoding algorithm A. *IEEE Trans. Inform. Theory.* 2004, 50, pp. 2010–2021.

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

[4] Divsalar, D., Jin, H., and McEliece R. Coding Theorems for "Turbo-Like" Codes. *Proc. 36th Annual Allerton Conf. on Comm., Control, and Computing.* 1998, pp. 201–210.

[5] Etzion, T., Trachtenberg, A., and Vardy, A. Which Codes Have Cycle-Free Tanner Graphs? *IEEE Trans. Inform. Theory.* 1999, 45, pp. 2173–2181.

[6] Forney, D. Principles of Digital Communication II : Lecture Notes [online]. 2005 [cited 2009-03-03]. Capacity-approaching codes. Available from World Wide Web: `http://ocw.mit.edu/courses/`.

[7] Jin, H., Khandekar, A., and McEliece R. Irregular Repeat-Accumulate Codes. *Proc. 2nd. Int. Symp. on Turbo Codes and Related Topics.* 2000, pp. 1–8.

[8] Johnson, S. J. Finite-Length Repeat-Accumulate Codes on the Binary Erasure Channel. *2005 Asia-Pacific Conference on Communications.* 2005, pp. 217–221.

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

[10] MacKay, D. J. C. *Information Theory, Inference, and Learning Algorithms.* [s.l.] : Cambridge University Press, 2003. xii, 628 p. Available from World Wide Web: `http://www.inference.phy.cam.ac.uk/mackay/itila/`. ISBN 9780521642989.

[11] Richardson, T., and Urbanke, R. The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding. *IEEE Trans. Inform. Theory.* 2001, 47, pp. 599–618.

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

[13] Richardson, T., and Urbanke, R. *Modern Coding Theory.* New York : Cambridge University Press, 2008. xvi, 572 p. ISBN 978-0-521-85229-6.

[14] Ryan, W. E. An Introduction to LDPC Codes, in *CRC Handbook for Coding and Signal Processing for Recording Systems.* CRC Press, 2004.

[15] Yang, M., and Ryan, W. E. Performance of Efficiently Encodable Low-Density Parity-Check Codes in Noise Bursts on the EPR4 Channel. *IEEE Trans. Magn.* 2004, 40, pp. 507–512.

[16] CCSDS 131.1-O-2. Low Density Parity Check Codes for Use in Near-Earth and Deep Space Applications : Experimental Specification. Washington, D.C. : CCSDS Secretariat, September 2007. vii, 36 p. Available from World Wide Web: `http://public.ccsds.org/publications/archive/131x1o2e2.pdf`.

[17] IEEE Std 802.3-2008. *Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications : Section four.* New York, NY : IEEE, 26 December 2008. 586 p. Available from World Wide Web: `http://standards.ieee.org/about/get/802/802.3.html`. ISBN 973-07381-5796-2.

[18] IEEE Std 802.11n-2009. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications : Enhancements for Higher Throughput.* New York, NY : IEEE, 30 October 2009. xxxii, 502 p. Available from World Wide Web: `http://standards.ieee.org/about/get/802/802.11.html`. ISBN 978-0-7381-6046-7.

[19] *DVB : Digital Video Broadcasting* [online]. c2003 [cited 2011-06-01]. DVB Fact Sheets. Available from World Wide Web: `http://dvb.org/technology/fact_sheets/index.xml`.

List of figures

List of tables

List of abbreviations

BCH	Bose-Chaudhuri-Hocquenghem
BEC	Binary Erasure Channel
BSC	Binary Symmetric Channel
DVB	Digital Video Broadcasting
EXIT	Extrinsic Information Transfer
IP	Internet Protocol
IRA	Irregular Repeat-Accumulate
LDPC	Low-Density Parity-Check
MAP	Maximum A-Posteriori
ML	Maximum Likelihood
RA	Repeat and Accumulate

Appendix A

Appendix

	$(3,6)$-regular			Irregular		
Probability of erasure ε	Number of iterations*		Success rate	Number of iterations*		Success rate
0.05	3.1	(0.3)	100.00 %	4.0	(0.5)	99.93 %
0.10	3.9	(0.3)	99.99 %	5.2	(0.7)	99.49 %
0.15	4.6	(0.5)	99.99 %	6.5	(0.9)	98.89 %
0.20	5.4	(0.5)	99.98 %	8.0	(1.2)	97.81 %
0.25	6.5	(0.6)	99.98 %	9.9	(1.6)	95.70 %
0.30	8.0	(0.7)	99.92 %	12.4	(2.1)	91.70 %
0.32	8.8	(0.8)	99.91 %	13.7	(2.3)	89.22 %
0.34	9.9	(0.9)	99.90 %	15.3	(2.6)	86.20 %
0.36	11.5	(1.3)	99.90 %	17.2	(2.9)	82.06 %
0.38	14.0	(2.3)	99.89 %	19.6	(3.3)	76.89 %
0.40	19.1	(5.6)	98.01 %	22.9	(4.0)	70.78 %
0.41	23.5	(8.1)	90.05 %	25.0	(4.5)	67.20 %
0.42	28.6	(10.4)	68.40 %	27.5	(5.1)	62.48 %
0.43	33.8	(11.8)	37.56 %	30.6	(5.9)	56.80 %
0.44	38.8	(13.1)	13.08 %	34.7	(7.2)	50.51 %
0.45	42.1	(12.5)	2.34 %	40.0	(9.2)	41.86 %
0.46	45.3	(12.8)	0.19 %	46.5	(11.3)	30.33 %
0.47	n/a	(n/a)	0 %	53.2	(13.3)	17.06 %
0.48	n/a	(n/a)	0 %	60.4	(14.2)	6.81 %
0.49	n/a	(n/a)	0 %	68.6	(16.5)	1.63 %
0.50	n/a	(n/a)	0 %	78.9	(14.5)	0.20 %
0.51	n/a	(n/a)	0 %	111.0	(n/a)	0.01 %
0.52	n/a	(n/a)	0 %	n/a	(n/a)	0 %

*Mean value with sample standard deviation in parentheses.

TABLE A.1: Decoding the near capacity irregular LDPC codes with degree distribution (3.10) and length 2048 on the BEC(ε) in comparison with decoding $(3,6)$-regular LDPC codes of length 2048.

Probability of erasure ε	Number of iterations*		Success rate
0.05	3.0	(0.1)	100.00 %
0.10	3.1	(0.4)	100.00 %
0.15	4.0	(0.1)	100.00 %
0.20	5.1	(0.3)	100.00 %
0.25	7.2	(0.8)	100.00 %
0.26	8.0	(1.1)	100.00 %
0.27	9.2	(1.7)	99.94 %
0.28	11.0	(3.0)	98.77 %
0.29	13.9	(4.9)	91.08 %
0.30	17.3	(6.5)	65.92 %
0.31	20.7	(7.2)	31.48 %
0.32	23.5	(7.7)	8.21 %
0.33	25.6	(7.6)	1.24 %
0.34	27.6	(8.3)	0.09 %
0.35	38.0	(n/a)	0.01 %
0.36	n/a	(n/a)	0 %

*Mean value with sample standard deviation in parentheses.

TABLE A.2: Decoding $(6, 12)$-regular LDPC codes of length 2048 on the BEC(ε).

Probability of erasure ε	Number of iterations*		Success rate
0.20	9.3	(0.5)	100 %
0.25	11.3	(0.5)	100 %
0.30	14.1	(0.2)	100 %
0.35	18.3	(0.5)	100 %
0.40	26.1	(0.3)	100 %
0.41	28.8	(0.4)	100 %
0.42	31.9	(0.4)	100 %
0.43	35.7	(0.5)	100 %
0.44	40.7	(0.5)	100 %
0.45	47.7	(0.7)	100 %
0.46	57.8	(0.8)	100 %
0.47	74.7	(1.3)	100 %
0.48	108.5	(2.6)	100 %
0.49	229.2	(11.7)	100 %
0.491	264.3	(15.7)	100 %
0.492	316.9	(24.2)	100 %
0.493	404.8	(46.3)	100 %
0.4940	588.5	(137.0)	92 %
0.4945	746.5	(157.8)	73 %
0.4950	970.3	(249.4)	38 %
0.4955	1166.8	(239.8)	6 %
0.4960	1597.0	(n/a)	1 %
0.497	n/a	(n/a)	0 %

*Mean value with sample standard deviation in parentheses.

TABLE A.3: Decoding the near capacity irregular LDPC codes with degree distribution (3.10) and length 2^{21} on the BEC(ε).