laSalle

UNIVERSITAT RAMON LLULL

Escola Tècnica Superior d'Enginyeria La Salle

Treball Final de Màster

Màster Universitari en Enginyeria de Xarxes i Telecomunicacions

FEC encoding: BCH-LDCP

Alumne
Enrique Novellón Gironès

Professor Ponent
Xavier Vilasis Cardona

ACTA DE L'EXAMEN DEL TREBALL FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Enrique Novellón Gironès

va exposar el seu Treball de Fi de Carrera, el qual va tractar sobre el tema següent:

FEC encoding: BCH-LDCP

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Università La Sapienza

Facoltà di Ingegneria

MASTER FINAL THESIS

FEC Encoding: BCH-LDPC

Enrique Novellón Gironés

Master in Network Engineering and Telecommunications

Roma - May 26, 2010

Abstract

At this fully digital age and with the massive transmission of information by electronic media, the error detection and correction is something inherent to the electronic transmission of information. This transmission is performed by several channels: copper cable, optical fiber, electromagnetic waves.... There exist interferences, noise, that affect to these channels and can introduce errors during the transmission. For this reason it is necessary to know how to detect when these errors occur and to be able to correct them whenever it is required.

In the theory of information, the Shannon-Hartley theorem is an application of the theorem of codification for noisy channels. The theorem determines the Shannon-capacity of the channel, a superior bound that establishes the maximum quantity of digital data that can be transmitted without error (i.e. information) by that channel with a specific bandwidth and which is affected by the presence of noise.

The last few years have witnessed a significant decrease in the gap between the Shannon channel capacity limit and what is practically achievable. Progress has resulted from novel extensions of previously known coding techniques involving interleaved concatenated codes. A considerable body of simulation results is now available, supported by an important but limited theoretical basis.

This thesis presents a deep analysis about the types of coding used in the ultimate versions of the DVB standards, such as terrestrial (T2), satellite (S2) and cable (C2). This codes are Bose and Chaudhuri Hocquenghem (BCH) and Low-Density Parity-Check (LDPC) and they successfully approach the Shannon limit. It is presented, first of all, an introduction to the world of error-correcting codes so as to get a global idea. There is also a final comparison between LDPC and Turbo-Codes, another important type of coding although not used in the standards mentioned before. At the end, it is presented a brief summary of the main ideas and a list of conclusions.

Contents

List of Figures

Chapter 1

Introduction to the world of error-correcting codes

Before immersing into the contents of the new and current error-correcting codes, we should have a global vision about why they exist, how they work and how they have evolved along our days, as well as about their current situation.

At this fully digital age and with the massive transmission of information by electronic media, the error detection and correction is something inherent to the electronic transmission of information. This transmission is performed by several channels: copper cable, optical fiber, electromagnetic waves.... There exist interferences, noise, that affect to these channels and can introduce errors during the transmission. For this reason it is necessary to know how to detect when these errors occur and to be able to correct them whenever it is required.

Human beings use a language to communicate among themselves, and only one. However, there exist many languages by which human beings can communicate. In a similar way, in digital communications many languages are used, error-correcting codes, for the transmission of information.

Thus, error-correcting codes are not a protocol of communications but a way of encoding the information that is going to be transmitted, so that the receiver of the information can detect whether there have been errors during the transmission and correct them in certain cases. Thanks to them the receiver knows when the received information is correct and the fact of being able to correct the errors avoids it from requesting the transceiver the retransmission of data.

The targets of the coding theory are the following ones:

- Building codes for the transmission of the information.

- These codes must detect the errors during the transmission.
- They must correct the maximum number of errors.

Thus, we must be sure that if in a transmission some errors occur, they can be detected. The method to detect and correct errors lies in including in the blocks of the transmitted data some additional bits called **redundancy-bits**.

Two basic strategies have been developed to manage the errors:

- *Error-Detection Codes:* They include only the necessary redundant information in each data-block to detect the errors. In this case the number of redundancy bits is smaller.
- *Error-Correction Codes:* They include enough redundant information in each data-block so that the erroneous bits can be detected and corrected.

If we consider a data-block formed by m data-bits and r redundancy-bits, the final length of the block will be n, where $n = m + r$.

1.1 Error-detection

Detecting errors is quite easy for human beings. We have a grammar and syntax in the language which we use to establish a communications. Following this "human method" is not functional for an electronic device, by now. Especially when the transmitted information by electronic media goes beyond the human condition.

The transmitted information by electronic media is not limited to the information that we want to transmit. Furthermore, all the data necessary for the transmission of information is included, such as the protocols of communication which are being used. In the error-correcting codes the size of the word is fixed so that all the words have the same size. In our code we use a restricted set of these words.

If we suppose that we have an alphabet composed by 26 letters and the words of our code are formed by 4 letters, how many words can we form?

$$26^4 = 456976$$

From these 456976 available words we will choose a set of them and we will communicate ourselves with this set. Thus, when we receive in the transmission a

word which is not included in this set we will assume that there has been an error during the transmission.

This presents a problem. What would happen if when an error occurs it results in a word from our set? In this case we will not be able to detect the error. This is why the words from our set must accomplish some requirements.

1.1.1 Hamming distance

The Hamming distance is defined as the number of characters which differ in two words. For instance:

$$d_{Hamming}(abcd, acbd) = 2$$

In this case we say that there exists a Hamming weight of 2 between both words. If we choose the words from the code so that the minimum of the Hamming distances among them is as large as possible, we will reduce the probability of that an error occurred during the transmission results in another word from the code. That is to say, we will reduce the probability that non-detectable errors occur. We call this distance as the minimum distance of the code.

How many errors can a code detect?

Theorem 1 *A code detects errors of a Hamming weight minor or equal to* n *if and only if the minimum distance of the code is minor to* n.

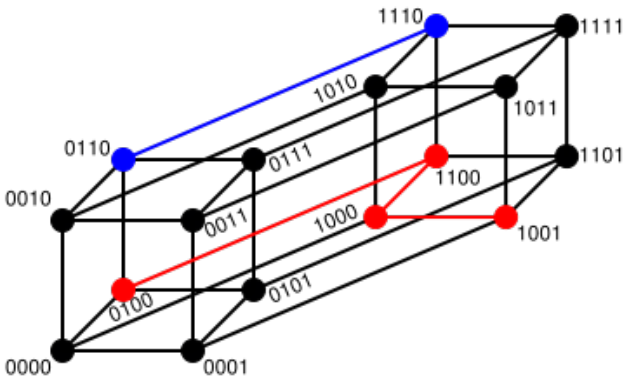


Figure 1.1: 4-bit binary hypercube for finding Hamming distance

In the figure 1.1 we can observe two examples:

1. Red path: 0100-1001 has distance 3 (0100 requires 3 bits to change in order to become 1001).

2. Blue path: 0110-1110 has distance 1 (0110 requires 1 bit to change in order to become 1110).

1.1.2 Types of error-detecting codes

Several schemes exist to achieve error detection. The general idea is to add some redundancy (i.e., some extra data) to a message, which enables detection of any errors in the delivered message.

Most such error-detection schemes are systematic: The transmitter sends the original data bits, and attaches a fixed number of check bits, which are derived from the data bits by some deterministic algorithm. The receiver applies the same algorithm to the received data bits and compares its output to the received check bits; if the values do not match, an error has occurred at some point during the transmission. In a system that uses a "non-systematic" code, such as some raptor codes, the original message is transformed into an encoded message that has at least as many bits as the original message.

1.1.2.1 Simple parity (horizontal check)

It consists of adding an extra bit to the string that we want to send, and it will indicate us if the number of ones (bits set to 1) is even or odd. If it is even we will include this bit with value = 0; otherwise, we will include it with value = 1.

** Example of generation of one bit of simple parity:*

We want to send the string "1110100":

1. We count the quantity of ones included: 4 ones.

2. The number of ones is even so we add a bit with value = 0.

3. The sent string is 11101000.

The receiver now repeats the operation of counting the quantity of ones included (except the last bit) and if it matches, no error has occur.

Problems with this method:

There is a high probability that there exist some cases in which there has been an error, and it has not been detected, such as if two numbers change in the transmission instead of one.

1.1.2.2 Diagonal parity (horizontal-vertical check)

In order to improve the previous method, we perform a parity that affects both the bits of every string or word and a set of all of them. We always use relatively short strings so as to avoid us from having lots of errors.

To see it more clearly, we usually group the bits in a matrix of N rows and K columns. After that, we perform all the horizontal parities by the previous method and at the end, we repeat the same operation of calculating the number of ones, but now from every column.

The probability of finding just one error is the same, but instead, the probability of finding an even number of errors is not zero anymore, as in the previous case. Even so, there still exist a great quantity of non-detectable errors.

** Example of diagonal parity (or geometric code):*

1. We have this code to transmit: 11001011110101100101110101l0.
2. We group the code into every word, forming a matrix of N x K:

1100101
1110101
1001011
1010110

3. We add the horizontal parity bits:

1100101 0
1110101 1
1001011 0
1010110 0

4. We add the vertical parity bits:

1100101 0
1110101 1
1001011 0
1010110 0
0001101 1

Once we create the matrix, we can send it by rows or by columns. By sending the words by columns we increase the probability of correcting a word which has suffered a burst error[1].

1.1.2.3 Cyclic redundancy checks (CRCs)

By trying to improve the codes that only control the parity-bit, there exist the cyclic codes. These codes use the modular arithmetic to detect a greater number of errors, by using operations in module 2 and the sums and subtractions are performed without carriage (becoming operations Or-Exclusive or XOR). Furthermore, in order to ease the calculations they work with polynomials, although only theoretically.

The aim of this method is to create a part of the redundancy which we add at the end of the code to transmit (just like in the methods of parity). This part must be as small as possible and must detect the greatest possible number of errors. But besides this, it must be a systematic method, that is to say, with the same code to transmit (and the same generator polynomial) it must always generate the same final code.

Generator polynomial:

It is a polynomial which has been chosen previously and minimizes the redundancy. It usually has a length of 16 bits, for messages of 128 bytes, which indicates that its efficiency is good, since it only increases the length in an approximate 1.6%:

$$(16bits/(128bytes * 8bitsperbyte)) * 100 = 1,5625\%$$

An example of generator polynomial used normally in networks WAN is: $g(x) = x^{16} + x^{12} + x^5 + 1$

The operations performed by the transceiver to calculate its CRC are:

1. It adds as many zeros at the right of the original message as the grade of the generator polynomial indicates.
2. It divides the message with the zeros included by the generator polynomial.
3. The remainder obtained from the division is added to the message with the zeros included.

[1]Errors which affect several bits in a row, generally due to electronic causes, such as sparks, and which could make the word to be lost completely.

4. The result obtained is transmitted.

These operations are generally incorporated in the hardware in order to be calculated with the highest speed, but in theory they only use the polynomials in order to ease de calculations.

** Example of how to obtain the CRC:*

Data:

Coded binary message: 1101001
Generator polynomial: $x^4 + x + 1$

Operations:

1. Obtain the equivalent polynomial of the message: $x^6 + x^5 + x^3 + 1$
2. Multiply the message by x^4 (adding 4 zeros at the right): $x^{10} + x^9 + x^7 + x^4$
3. Divide in binary the message by the generator polynomial and obtain the remainder: $x^2 + 1$
4. Transmit the message with the remainder (also in module 2): $x^{10} + x^9 + x^7 + x^4 + x^2 + 1$

The receiver must check the CRC code to detect if some errors have occurred.

** Example of the calculations in the receiver:*

1. They agree to the generator polynomial by corresponding protocol.
2. It divides the received code by the generator polynomial.
3. It checks the remainder of this operation:
   - If the remainder is zero, no errors have occurred and the message is processed.
   - If the remainder is nonzero, it means that some errors have occurred so the message must be resent or it must try to correct the errors by the error-coding correctors.

To sum up, this method requires a generator polynomial which, chosen properly, can achieve a great number of errors: simple, double, in the odd positions of the bits, bursts with a smaller length than the grade of the polynomial, etc.

1.1.2.4 Checksums

It is an easy method but efficient only with the short-length strings of words. For this reason, it is only used in headers of important frames or another important strings, and in combination with other methods.

It involves grouping the message to transmit in strings of a certain length L which is not very big, for example 16 bits, considering every string as an integer numbered according to the numbering system $2^L - 1$. After that it sums up the value of all the words in which the message is divided and it adds the result to the message to transmit, but with the sign changed. With this, the receiver only has to sum up all the strings, and if the result is 0 it means that no errors have occurred.

* *Example:*

Message 101001110101

1. Agree to the length of the string: 3
2. Agree to the numbering system: $2^3 - 1 = 7$
3. Split the message: 101 001 110 101
4. Transform every string in an integer: 5 1 6 5
5. Sum up all the values and add the number with its sign changed: -17
6. Send 5 1 6 5 -17 binary-coded
7. The receiver sums up all the values and if it is = 0 processes the message; otherwise, some errors have occurred.

This is the easiest method and it is optimum to be implemented in software, since it can reach a speed of calculation similar to the hardware implementation.

1.2 Error-correction

So far, the methods that we have studied are included in the error-detection, with capacity of detecting but not correcting. Now we are going to develop the erro-correcting codes. There are two basic ways to design the channel code and protocol for an error-correcting system:

- *Automatic Repeat Request (ARQ):* The transmitter sends the data and also an error-detection code, which the receiver uses to check for errors, and requests retransmission of the data that was deemed erroneous. In many cases the request is implicit: The receiver sends an acknowledgement (ACK) of correctly received data, and the transmitter re-sends anything not acknowledged within a reasonable period of time.

- *Forward Error Correction (FEC):* The transmitter encodes the data with an error-correcting code (ECC) and sends the coded message. The receiver never sends any messages back to the transmitter. The receiver decodes what it receives into the "most likely" data. Forward error-correction codes are designed for real-time systems, such as video transmission.

It is possible to combine the ARQ and FEC so that minor errors are corrected without retransmission, and major errors are corrected via a request for retransmission. The combination is called a hybrid automatic repeat-request.

Originally the error was only treated by probability, according to its random nature. As from the Shannon theory (see page 12), which establishes units of measurement of information, it is also possible to measure and quantify the error, even determine previously the maximum levels of error in a message.

Theoretically it is possible to correct any fragment of binary code automatically. For that, error-detecting codes are complemented with error-correcting codes, of a greater mathematical complexity and a greater number of necessary redundancy bits. The cost of decreasing the level of error of a message, is the increase of the redundancy bits and latency due to more complex algorithms, and the economic factor.

The necessity of a greater number of redundancy bits sometimes makes the correction of multiple bits become non-viable and inefficient. For this reason the error-correcting codes usually correct 1, 2 or 3 bits.

How many errors can a code correct?

Theorem 2 *A code can correct all the errors of a weight minor or equal to* t *if and only if its minimum Hamming distance is bigger or equal to* $2 \cdot t + 1$.

1.2.1 Types of error-correcting codes (FEC)

There is a great quantity of methods to generate FEC, but they are all orientated to reconstruct in the receiver the data-sequence originated in the transceiver. In the video transmission we use two methods:

- *Block codes:* they work on fixed-size blocks (packets) of bits or symbols of predetermined size. Practical block codes can generally be decoded in polynomial time to their block length. There are many types of block codes, but among the classical ones the most notable is Reed-Solomon coding because of its widespread use.

- *Convolutional codes:* they work on bit or symbol streams of arbitrary length. They are most often decoded with the Viterbi algorithm, though other algorithms are sometimes used. Viterbi decoding allows asymptotically optimal decoding efficiency with increasing constraint length of the convolutional code, but at the expense of exponentially increasing complexity. A convolutional code can be turned into a block code, if desired.

1.2.1.1 Reed-Solomon (RS)

The RS is a FEC code, whose coding process is similar to all the block codes, involving an operation that adds a group of known bits to a group of bits of the message to be transmitted. The relation between the added bits and the message bits is known by the receiver so that it can decode it.

The RS code, and in general all the block codes, is defined as systematic codes. This indicates that the message is not altered by the addition of redundancy bits, since the position of the added bits to the message is always well-defined according to the error-correcting polynomials. This way, it is not necessary to include all the parity bits calculated in the RS-encoder inside the message, being added only at the end of the stream.

The entrance to the RS encoder is a stream of k bytes of message (mixed in the randomizer) and its exit is a stream of n bytes, where n is bigger than k, so that n - k is the number of parity bytes added.

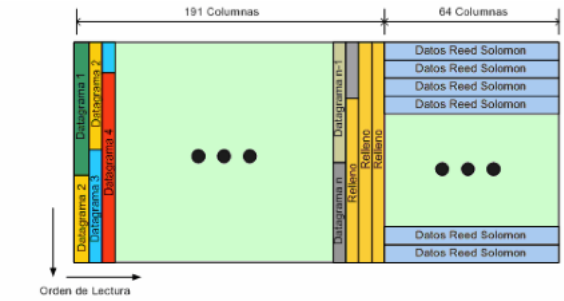


Figure 1.2: Example of RS stream at the exit of the encoder

The RS code is represented as RS(n,k,t) or just RS(n,k), where $t = (n - k)/2$ indicates the maximum number of erroneous bytes that can be corrected. A typical example of RS code in the transmission of video is RS(204,188,8) which indicates that to an input of 188 bytes we add 16 parity bytes, generating a capacity of correcting 8 erroneous bytes. The correction of 1 erroneous byte is achieved independently of the total number of erroneous bits it has, it could be one, two or all the bits from the byte.

The capacity of the RS code to correct groups of bits or bytes is adequate to correct burst errors. However, since this capacity is limited, it is not used for long burst errors. The interleaver located at the exit of the RS coder makes the burst to affect different frames.

We will see later how we can correct bits by a series of calculations with polynomials.

1.2.1.2 Viterbi

The convolutional coding is a technique to control the error which generates a coded sequence of bits from an input sequence of bits of information.

The coded sequence at the exit is generated from every bit of the input sequence and not from the blocks as in RS codes. The output of the encoder, apart from depending on the current input bit, depends on the previous input bits, in a continuous process that creates redundancy.

The encoder has inner memories in order to remember the bits from previous inputs. The greater quantity of memories it has, the higher probability of decreasing the error. The disadvantage are a greater quantity of redundancy bits, a greater latency due to the complexity of the operations in each memory, more computer resources needed and a higher economic cost.

The Viterbi algorithm is a technique to decode the possible states of a state machine, which would correspond to a transmission of information. Viterbi says that the current state of an event depends on a previous state, and it provides with a method to determine all the states with minimum calculation. This algorithm lets us detect the real sequence of transmitted information in case an error occures in the channel.

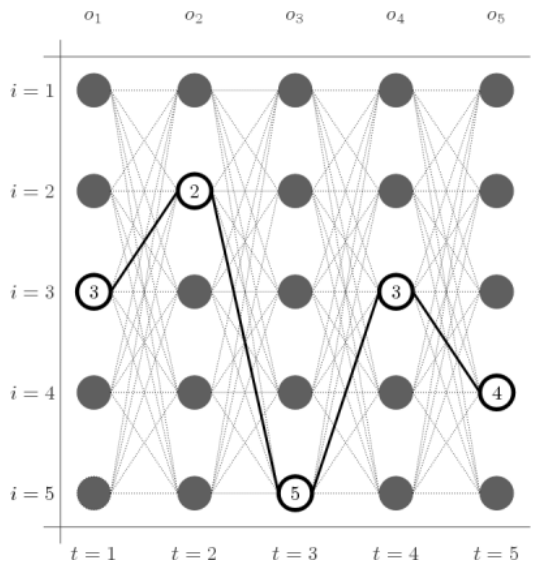


Figure 1.3: Example of application of the Viterbi algorithm

In a real convolutional encoder, the input sequence is split in segments of k bits which are introduced simultaneously at the entrance of the encoder. For each input segment of k elements, the encoder generates an output segment of n elements, where is bigger than k.

The n bits of the output segment depend on the k bits of the input segment at the time t and on the bits of the previous segments. That is to say, the segments at the instants *t-1, t-2, ..., t-K*, where$K$ is the number of previous inputs that can store the encoder.

The formal notation of the convolutional codes is indicating the parameters *n, k, K* as *C(n,k,K)*. But the common thing is to indicate the quotient between the input elements and the output ones as $R=k/n$, called code rate. For instance, if the elements of an input segment are 2 and the output segment has 3 elements, then R=2/3, indicating explicitly that for every 2 input bits another correcting bit is added. If R=7/8, then for every 7 input bits another correcting bit is added.

The code rate 2/3 adds more redundancy bits than 7/8, achieving a higher possibility of correcting errors.

1.3 Shannon theorem

In the theory of information, the Shannon-Hartley theorem is an application of the theorem of codification for noisy channels. A very frequent case is an analogic

channel of communication continuous in time that presents a gaussian noise.

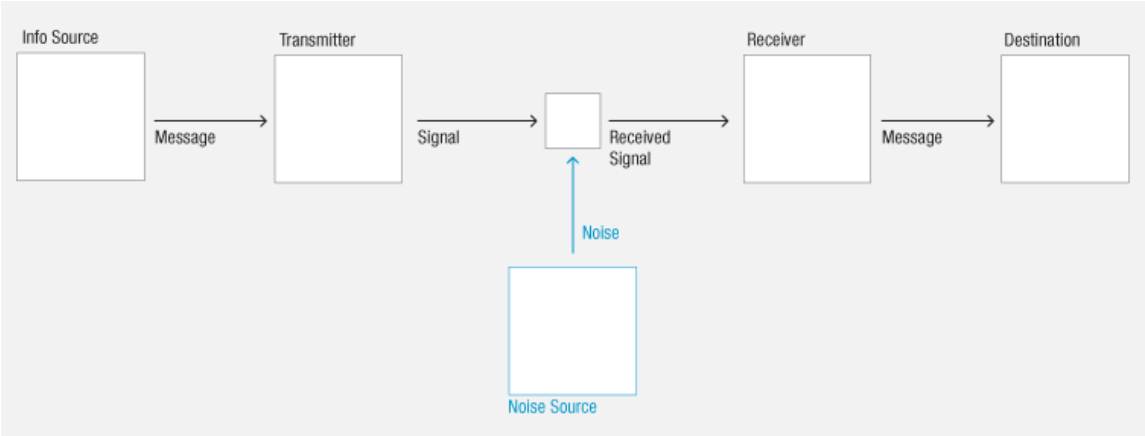


Figure 1.4: Schematic diagram of a general communication system

The theorem determines the Shannon-capacity of the channel, a superior bound that establishes the maximum quantity of digital data that can be transmitted without error (i.e. information) by that channel with a specific bandwidth and which is affected by the presence of noise.

In the main hypothesis, for the correct application of the theorem, we assume a limitation in the power of the signal and also that the process of the gaussian noise is characterized by a known power or a spectral density of power.

1.3.1 Nyquist rate

In 1927, Nyquist determined that the number of independent beats which could go through a telegraph channel, per unite of time, was limited to the double of the bandwidth of the channel.

$$f_b \leq 2 \cdot B \tag{1.1}$$

; where f_b is the frequency of the beat (in beats per second) and B is the bandwidth (in Herz). The quantity $2 \cdot B$ was called, afterwards, Nyquist rate.

1.3.2 Shannon limit

Claude Shannon, after Nyquist's investigation, studied how noise affects the transmission of data. Shannon took into account the relation signal-noise of the channel of transmission (measured in dB) and got the theorem of Shannon-capacity:

Theorem 3 *Given a noisy channel with capacity of channel C and information transmitted with a rate R, then if $R < C$ there exist codes which let the probability of error in the receiver be arbitrarily small.*

$$C = B \cdot log_2(1 + S/N)$$

This means that, theoretically, it is possible to transmit the information almost without error in any way under a limiting rate, C. This maximum capacity is not reachable, since the Shannon formula assumes some conditions which in practice do not exist. It does not take into account the impulsive noise, neither the attenuation or distortion. It simply represents the theoretic reachable limit.

In this thesis, we will analyze the new correcting codes which approximate more to this Shannon limit, such as *Low-Density Parity-Check (LDPC)*, which are being used in the new european standards of digital transmission (DVB-T2, C2 y S2).

Chapter 2

Overview of the new DVB standards

The Digital Video Broadcasting Project (DVB) is an industry-led consortium of around 250 broadcasters, manufacturers, network operators, software developers, regulatory bodies and others in over 35 countries committed to designing open technical standards for the global delivery of digital television and data services. Services using DVB standards are available on every continent with more than 500 million DVB receivers deployed.

Figure 2.1: Logo of DVB

In this chapter it is presented an overview of the new standards DVB-T2, C2 and S2, which use the error-correcting codes that we will explain later.

2.1 DVB-T2

DVB-T2 is an abbreviation for *Digital Video Broadcasting – Second Generation Terrestrial*; it is the extension of the television standard DVB-T, issued by the consortium DVB, devised for the broadcast transmission of digital terrestrial television.

This system transmits compressed digital audio, video, and other data in "physical layer pipes" (PLPs), using OFDM modulation with concatenated channel coding and interleaving. It is currently broadcasting in parts of the UK under the brand name *Freeview HD*.

The generic T2 system model is represented in figure 2.2. The system input(s) may be one or more MPEG-2 Transport Stream(s) and/or one or more Generic Stream(s). The Input Pre-Processor, which is not part of the T2 system, may include a Service splitter or de-multiplexer for Transport Streams (TS) for separating the services into the T2 system inputs, which are one or more logical data streams. These are then carried in individual Physical Layer Pipes (PLPs).

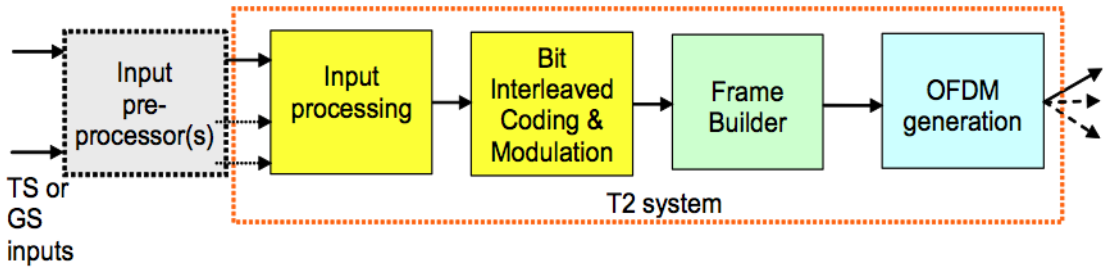


Figure 2.2: High level T2 block diagram

The system output is typically a single signal to be transmitted on a single RF channel. Optionally, the system can generate a second set of output signals, to be conveyed to a second set of antennas in what is called MISO[1] transmission mode. The standard defines a single profile which incorporates time-slicing but not time-frequency-slicing (TFS). It is not intended that a receiver with a single tuner should support TFS.

The input data streams shall be subject to the constraint that, over the duration of one physical-layer frame (T2-frame), the total input data capacity (in terms of cell throughput, following null-packet deletion, if applicable, and after coding and modulation), shall not exceed the T2 available capacity (in terms of data cells, constant in time) of the T2-frame for the current frame parameters. Typically, this will be achieved by arranging that PLPs within a group of PLPs will always use same modulation and coding (MODCOD), and interleaving depth, and that one or more groups of PLPs with the same MODCOD and interleaving depth originate from a single, constant bit-rate, statistically-multiplexed source.

Each group of PLPs may contain one common PLP, but a group of PLPs need not contain a common PLP. When the DVB-T2 signal carries a single PLP there is

[1]Multiple-input and Single-output

no common PLP. It is assumed that the receiver will always be able to receive one data PLP and its associated common PLP, if any.

More generally, the group of statistically multiplexed services can use variable coding and modulation (VCM) for different services, provided they generate a constant total output capacity (i.e. in terms of cell rate including FEC and modulation).

When multiple input MPEG-2 TSs are transmitted via a group of PLPs, splitting of input TSs into TSPS streams (carried via the data PLPs) and a TSPSC stream (carried via the associated common PLP), shall be performed immediately before the Input processing block shown in figure 2.2. This processing shall be considered an integral part of an extended DVB-T2 system.

The maximum input rate for any TS, including null packets, shall be 72 Mbit/s. The maximum achievable throughput rate, after deletion of null packets when applicable, is more than 50 Mbit/s (in an 8 MHz channel).

2.2 DVB-C2

DVB-C stands for *Digital Video Broadcasting - Cable* and it is the DVB European consortium standard for the broadcast transmission of digital television over cable. On February 18, 2008 it was announced that a new standard - DVB-C2 - would be developed during 2008, and a "Call for Technologies" was issued. Proposals including simulation programs and information on patent rights could be submitted until June 16, 2008.

The results of the DVB-C2 Study Mission already provided clear indications that technologies are available allowing the performance of the second generation DVB cable transmission system to get so close to the theoretical Shannon Limit that any further improvements in the future would most likely not be able to justify the introduction of a disruptive third generation of cable transmission system.

The generic C2 System model is represented in figure 2.3. The system input(s) may be one or more MPEG-2 Transport Stream(s) and/or one or more Generic Stream(s). The Input pre-processor, which is not part of the C2 System, may include a service splitter or a demultiplexer for Transport Streams (TS) used to separate the services into the C2 System inputs, which are one or more logical data streams. These are then carried in individual Physical Layer Pipes (PLPs). The system output is a single signal to be transmitted on a single RF channel.

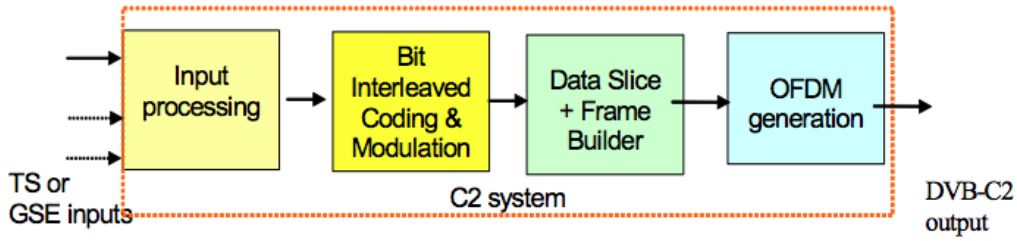


Figure 2.3: High level C2 block diagram

The input data streams shall be subject to the constraint that, over the duration of one physical-layer frame (C2 Frame), the total input data capacity (in terms of cell throughput, following Null Packet Deletion, if applicable, and after coding and modulation), shall not exceed the C2 available capacity (in terms of Data Cells, constant in time) of the C2 Frame for the current frame parameters. One or more PLPs are arranged in a group of PLPs and one or more of such groups of PLPs form a Data Slice. A C2 System may consist of one or more Data Slices. Each group of PLPs may contain one Common PLP, but a group of PLPs need not contain a Common PLP. When the DVB-C2 signal carries a single PLP there is no Common PLP. It is assumed that the receiver will always be able to receive one Data PLP and its associated Common PLP, if any.

More generally, the group of statistically multiplexed services can use variable coding and modulation (VCM) for different services, provided they generate a constant total output capacity (i.e. in terms of cell rate including FEC and modulation).

When multiple input MPEG-2 TSs are transmitted via a group of PLPs, splitting of input TSs into TSPS streams (carried via the Data PLPs) and a TSPSC stream (carried via the associated Common PLP) shall be performed immediately before the Input processing block shown in figure 2.3. This processing shall be considered an integral part of an extended DVB-C2 System.

2.3 DVB-S2

Digital Video Broadcasting - Satellite - Second Generation is an enhanced specification to replace the DVB-S standard, developed in 2003 and ratified by ETSI (EN 302307) in March 2005. The development of DVB-S2 coincided with the introduction of HDTV and H.264 (MPEG-4 AVC) video codecs.

According to figure 2.4, the DVB-S2 System shall be composed of a sequence of functional blocks as described below.

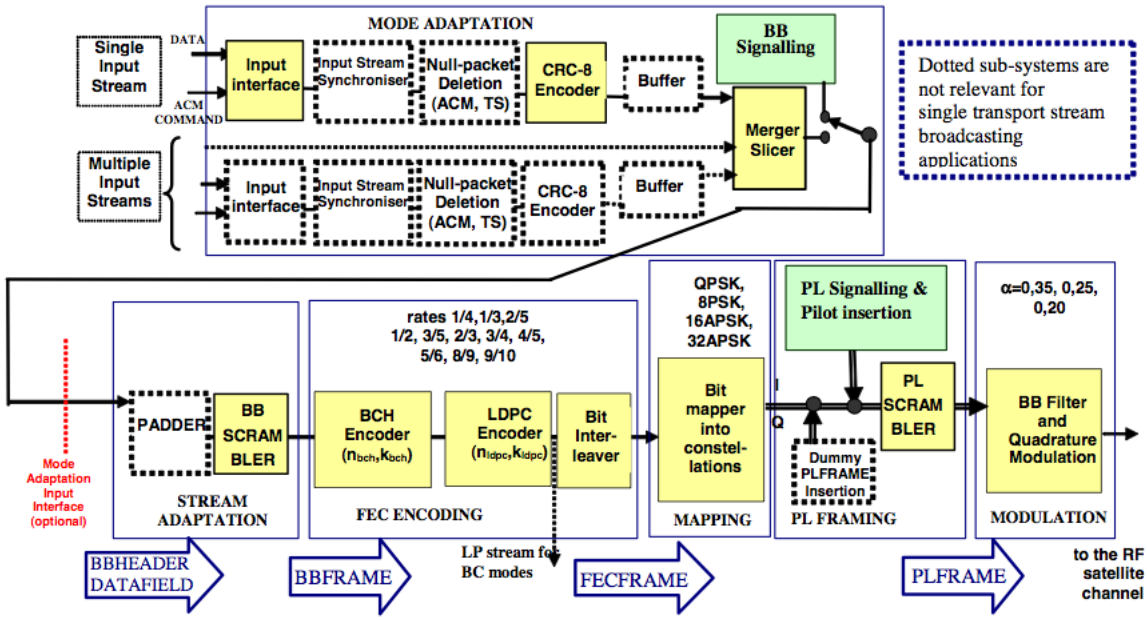


Figure 2.4: Functional block diagram of the DVB-S2 System

Mode adaptation shall be application dependent. It shall provide input stream interfacing, Input Stream Synchronization (optional), null-packet deletion (for ACM and Transport Stream input format only), CRC-8 coding for error detection at packet level in the receiver (for packetized input streams only), merging of input streams (for Multiple Input Stream modes only) and slicing into DATA FIELDs. For Constant Coding and Modulation (CCM) and single input Transport Stream, Mode Adaptation shall consist of a "transparent" DVB-ASI (or DVB-parallel) to logical-bit conversion and CRC-8 coding.

A Base-Band Header shall be appended in front of the Data Field, to notify the receiver of the input stream format and Mode Adaptation type. To be noted that the MPEG multiplex transport packets may be asynchronously mapped to the Base-Band Frames.

For applications requiring sophisticated merging policies, in accordance with specific service requirements (e.g. Quality of Service), Mode Adaptation may optionally be performed by a separate device, respecting all the rules of the DVB-S2 specification. To allow standard interfacing between Mode and Stream Adaptation functions, an optional modulator interface (Mode adaptation input interface) is defined. Stream adaptation shall be applied, to provide padding to complete a Base-Band Frame and Base-Band Scrambling.

Forward Error Correction (FEC) Encoding shall be carried out by the concatenation of BCH outer codes and LDPC (Low Density Parity Check) inner codes (rates 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9, 9/10). Depending on the application area, the FEC coded block shall have length nldpc= 64 800 bits or 16 200 bits. When VCM and ACM is used, FEC and modulation mode may be changed in different frames, but remains constant within a frame. For Backwards Compatible modes, the bit-stream at the output of the FEC encoder shall be processed. Bit interleaving shall be applied to FEC coded bits for 8PSK, 16APSK and 32APSK. Mapping into QPSK, 8PSK, 16APSK and 32APSK constellations shall be applied, depending on the application area. Gray mapping of constellations shall be used for QPSK and 8PSK.

Physical layer framing shall be applied, synchronous with the FEC frames, to provide Dummy PLFRAME insertion, Physical Layer (PL) Signalling, pilot symbols insertion (optional) and Physical Layer Scrambling for energy dispersal. Dummy PLFRAMEs are transmitted when no useful data is ready to be sent on the channel. The System provides a regular physical layer framing structure, based on SLOTs of M = 90 modulated symbols, allowing reliable receiver synchronization on the FEC block structure. A slot is devoted to physical layer signalling, including Start-of-Frame delimitation and transmission mode definition. This mechanism is suitable also for VCM and ACM demodulator setting. Carrier recovery in the receivers may be facilitated by the introduction of a regular raster of pilot symbols (P = 36 pilot symbols every 16 SLOTs of 90 symbols), while a pilot-less transmission mode is also available, offering an additional 2,4% useful capacity.

Base-Band Filtering and Quadrature Modulation shall be applied, to shape the signal spectrum (squared-root raised cosine, roll-off factors 0,35 or 0,25 or 0,20) and to generate the RF signal.

From now on, we will dedicate exclusively to analyze the FEC coding mentioned above, which is applied for all the standards commented before.

Chapter 3

Bose and Chaudhuri Hocquenghem (BCH)

3.1 Introduction

In coding theory the BCH codes form a class of parameterised error-correcting codes which have been the subject of much academic attention in the last fifty years. BCH codes were invented in 1959 by Hocquenghem, and independently in 1960 by Bose and Ray-Chaudhuri. The acronym BCH comprises the initials of these inventors' names.

The principal advantage of BCH codes is the ease with which they can be decoded, via an elegant algebraic method known as syndrome decoding. This allows very simple electronic hardware to perform the task, obviating the need for a computer, and meaning that a decoding device may be made small and low-powered. As a class of codes, they are also highly flexible, allowing control over block length and acceptable error thresholds, meaning that a custom code can be designed to a given specification (subject to mathematical constraints).

In technical terms a BCH code is a multilevel cyclic variable-length digital error-correcting code used to correct multiple random error patterns. BCH codes may also be used with multilevel phase-shift keying whenever the number of levels is a prime number or a power of a prime number. A BCH code in 11 levels has been used to represent the 10 decimal digits plus a sign digit.

BCH codes are also useful in theoretical computer science, for instance in the MAXEkSAT problem.

3.2 To the math

Regarding a coding system to reduce errors, Shannon says that it can be done, but not how. That is the curse of existence theorems. However, many have gone before us and laid the foundations of error control coding. Let's explore those foundations.

Transmission on computer and radio networks uses binary symbols. For example, the American Standard Code for Information Interchange defines the letter "A" as corresponding to a binary value of 1000001. The letter "B" is 100010, and so on. Each letter, number, and symbol on the typical personal computer keyboard has its own numeric value. A sentence is then represented by a string of these values: "ABC" = 1000001, 1000010, 1000011. If we desire to detect errors in the transmission of such a string of ones and zeros, then it is a good design decision to make a coding system that works with binary symbols.

However, what type of system shall we choose? We anticipate that encoding and decoding data will not be trivial, so we will need the most capable system available. From our study of abstract algebra (see any basic text on the subject), we know that the most flexible, yet still basic system of rules is that of the field. In a field, we can add, subtract, multiply and divide.

What field do we choose? Our binary symbols are strings of ones and zeros taken from $\mathbf{Z_2}$ (the field of binary numbers). Perhaps we could use it. However, it only has two digits, so if each character is to be represented in the field, we will need a bigger field. Well, the order of (number of elements in) every finite field is p^m for some prime p and integer $m > 0$, so we are stuck with the power of a prime number of elements in our field. That is coincidentally good, however, because digital data is transmitted typically in multiples of eight bits (2^3), so each message can be considered to have values from a field of 2^m bits, a power of a prime. Therefore, we conclude that $\mathbf{Z_{2^m}}$ is a good candidate for our field, with m to be determined. Unfortunately, we soon learn that ordinary arithmetic in $\mathbf{Z_{2^m}}$ does not meet the criterion for inverses. For example, in $\mathbf{Z_{2^4}}$ (containing the integers 0...15), 2 has no inverse. That is, 2b mod 16 = 1 has no solution b. (Remember that two field elements are multiplicative inverses if we get 1 when they are multiplied together.) Therefore, $\mathbf{Z_{2^4}}$ is not a field with standard multiplication as $\mathbf{Z_p}$ is. Ordinary arithmetic fails because $\mathbf{Z_{16}} \approx \mathbf{Z}/ < \mathbf{16} >$ ($\approx$ denotes isomorphism), and the generator of $< 16 >$ is a composite number. $\mathbf{Z_p} \approx \mathbf{Z}/ < \mathbf{p} >$ is a field for p a prime. We need a different operation, at least for multiplication.

That different arithmetic is polynomial arithmetic. We know that $\mathbf{Z}_\alpha \approx \mathbf{Z_2}[\mathbf{x}]/ < \mathbf{p}(\mathbf{x}) >$ where p(x) is a minimal polynomial with root α. Thus, we have replaced the non-prime 2^m with a "prime polynomial," irreducible p(x).

The isomorphism above tells us again that the required field exists. Now we have to build an example and see how it works. The following is a standard example of the construction of $\mathbf{Z_{2^4}}$ from the references. I will not go through all the details, but this is the basic outline.

We are constructing the extension field of $\mathbf{Z_2}[\mathbf{x}], \mathbf{Z_2}(\alpha)$. The isomorphism suggests that we need a minimal polynomial with a root in $\mathbf{Z_{2^4}}$. Since we want 16 elements in $\mathbf{Z_2}(\alpha)$, the degree of the polynomial should be 4, so that $2^4 = 16$.

However, there is an additional condition. The polynomial must be primitive in GF(2^4)[1]. That means that every element in $\mathbf{Z_{2^4}}$ must be expressible as some power of p(x). With this limitation, checking the minimal polynomials of all the nonzero elements in GF(2^4), we find two polynomials, $x^4 + x^3 + 1$ and $x^4 + x + 1$, that are primitive. Either may be used.

To generate the elements, we start with three initial elements of the extension field and perform all arithmetic modulo $p(x) = x^4 + x^3 + 1$. The initial elements are 0, 1, and α. Raising α to powers successively identifies α^2 and α^3 as members of the extension field. When we get to α^4, we realize that it does not represent an element of $\mathbf{Z_{2^4}}$, but we know that $p(\alpha) = 0 = x^4 + x^3 + 1$, so $\alpha^4 = \alpha^3 + 1$ (in a binary field addition is equivalent to subtraction). Thus, we reduce each power greater than three using the identity $\alpha^4 = \alpha^3 + 1$.

For example, when multiplying 1011 by 1101, the polynomial representation is:

$$(x^3 + x + 1)(x^3 + x^2 + 1) = x^6 + x^5 + x^4 + 3x^3 + x^2 + x + 1$$

Reducing this using the identity, finally we see that 1011 x 1101 = 0010, or:

$$(x^3 + x + 1)(x^3 + x^2 + 1) = x$$

By computing successive powers of α and reducing using the identity, we can build the entire field $\mathbf{Z_{2^4}}$. There is a program to do the busy work, which syntax appears in appendix A. Now that we have a field with all the proper operations, we may call it GF(2^4) or GF(16), the *Galois* field with 16 elements.

Below in the figure 3.1 appear the elements generated by both primitive polynomials. There are three forms of each element. The power form expresses the reality that each field element except zero is the power of the generating element, α, and is useful when multiplying field elements. The polynomial form is useful for adding field elements, and the binary values in the centers of the tables are merely

[1] Galois Field

the coefficient of these polynomials, with the highest power of α on the left. Notice that the two fields contain the same elements in different order.

Field of 16 elements generated by $x^4 + x + 1$		
Power Form	n-Tuple Form	Polynomial Form
0	0000	0
1	0001	1
α	0010	α
α^2	0100	α^2
α^3	1000	α^3
α^4	0011	$\alpha + 1$
α^5	0110	$\alpha^2 + \alpha$
α^6	1100	$\alpha^3 + \alpha^2$
α^7	1011	$\alpha^3 + \alpha + 1$
α^8	0101	$\alpha^2 + 1$
α^9	1010	$\alpha^3 + \alpha$
α^{10}	0111	$\alpha^2 + \alpha + 1$
α^{11}	1110	$\alpha^3 + \alpha^2 + \alpha$
α^{12}	1111	$\alpha^3 + \alpha^2 + \alpha + 1$
α^{13}	1101	$\alpha^3 + \alpha^2 + 1$
α^{14}	1001	$\alpha^3 + 1$

Field of 16 elements generated by $x^4 + x^3 + 1$		
Power Form	n-Tuple Form	Polynomial Form
0	0000	0
1	0001	1
α	0010	α
α^2	0100	α^2
α^3	1000	α^3
α^4	1001	$\alpha^3 + 1$
α^5	1011	$\alpha^3 + \alpha + 1$
α^6	1111	$\alpha^3 + \alpha^2 + \alpha + 1$
α^7	0111	$\alpha^2 + \alpha + 1$
α^8	1110	$\alpha^3 + \alpha^2 + 1$
α^9	0101	$\alpha^2 + 1$
α^{10}	1010	$\alpha^3 + \alpha$
α^{11}	1101	$\alpha^3 + \alpha^2 + 1$
α^{12}	0011	$\alpha + 1$
α^{13}	0110	$\alpha^2 + \alpha$
α^{14}	1100	$\alpha^3 + \alpha^2$

Figure 3.1: Fields of 16 elements generated by two different generators.

To examine why a primitive polynomial is needed to generate all the field elements, let's see what happens by using a non-primitive polynomial. Notice in figure 3.2 repetition in the elements.

Field of 16 elements generated by $x^4 + x^3 + x^2 + x + 1$		
Power Form	n-Tuple Form	Polynomial Form
0	0000	0
1	0001	1
α	0010	α
α^2	0100	α^2
α^3	1000	α^3
α^4	1111	$\alpha^3 + \alpha^2 + \alpha + 1$
α^5	0001	1
α^6	0010	α
α^7	0100	α^2
α^8	1000	α^3
α^9	1111	$\alpha^3 + \alpha^2 + \alpha + 1$
α^{10}	0001	1
α^{11}	0010	α
α^{12}	0100	α^2
α^{13}	1000	α^3
α^{14}	1111	$\alpha^3 + \alpha^2 + \alpha + 1$

Figure 3.2: Field of 16 elements generated by a non-primitive generator.

This is undesirable because not all the values in GF(16) are represented by the computations, and our ability to encode and decode messages using arithmetic in this set would be impaired or destroyed. (However, some families of codes do use non-primitive polynomials because it allows a greater range of selection of code lengths; the code can be tailored to the application.)

Mechanically, these field elements are easy to generate. Multiplication by element α results in a left shift of the previous binary value. If a one is shifted out of the fourth position (in GF(16)), that constitutes a polynomial of fourth degree and the result is reduced by subtracting the generator polynomial from the result. However, subtraction in binary arithmetic is merely the exclusive OR function, so an encoder for field elements is easy to build in logic circuits. Next time, we will look at various types of codes that arose over the years as a result of the investigations reproduced above.

3.3 Types of codes

We are looking at the BCH error detection and correction code, going first through some preliminary background on the mathematical basis of the code. Now that we have a field in which to do computations, what we need next is a philosophy for encoding and decoding data in order to detect and possibly correct errors. Here

we are taking advantage of results discovered by the hard work of perhaps a hundred individuals just in the last 50 years.

There are several types of codes. The first major classification is linear vs. non-linear. Linear codes in which we are interested may be encoded using the methods of linear algebra and polynomial arithmetic. Then we have block codes vs. convolutional codes. Convolutional codes operate on streams of data bits continuously, inserting redundant bits used to detect and correct errors.

Our area of investigation here will be linear block codes. Block codes differ from convolutional codes in that the data is encoded in discrete blocks, not continuously. The basic idea is to break our information into chunks, appending redundant check bits to each block, these bits being used to detect and correct errors. Each data + check bits block is called a codeword. A code is linear when each codeword is a linear combination of one or more other codewords. This is a concept from linear algebra and often the codewords are referred to as vectors for that reason.

Another characteristic of some block codes is a cyclic nature. That means any cyclic shift of a codeword is also a codeword. So linear, cyclic, block code codewords can be added to each other and shifted circularly in any way, and the result is still a codeword. You might expect that it takes some finesse to design a set of binary words to have these properties.

Since the sets of codewords may be considered a vector space, and also may be generated through polynomial division (the shifting algorithm, above), there are two methods of performing computations: linear algebra and polynomial arithmetic. We will dwell on polynomial arithmetic methods later in this paper.

Assume that we have a code that can detect t errors in a codeword. That means up to t errors can occur and the receiver will say with 100% certainty that the codeword contains errors. How does this work? What is the intuitive structure of these codewords in the field?

Let us say that we transmitted one of the codewords generated previously in GF(16) by the polynomial $\alpha^4 + \alpha + 1$. If an error occurs, the result will be another codeword in this field. We have no way of knowing exactly which bits were changed. That is because every possible bit pattern is a codeword.

However, if we used a larger field, say, GF(32), and then transmitted our four bit information words plus one check bit, half of the 32 codewords would be valid and the other half would not. If we received one of the invalid codewords, we could request a retransmission. That is exactly how parity check bits work, the simplest error detection system.

With the example above of the character "A", binary 1000001, only seven bits are needed to represent the entire alphabet and several punctuation marks, 128 values

in total. Adding an eighth bit, which is a parity sum of the other seven, enlarges the required field to 256 elements, with 128 of them representing valid information.

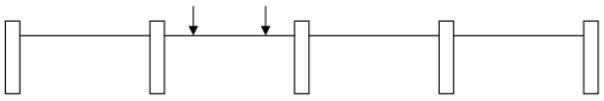
Then our letter "A" would be represented as 01000001, with the leftmost bit being an even parity bit. This bit is set or reset to give the entire binary word an even number of one bits. For the letter "C", the value is 11000011, and the parity bit is 1 because there are three ones in the rest of the word, four total. If a character is received with a parity error, it is discarded. The odd parity scheme is equivalent in performance and effectively identical. Now the parity bit is used in communications links today, and many chips have the capability of encoding and decoding it in hardware, flagging errant data words. However, it can only detect errors, and only those which change an odd number of bits. One may suspect that adding more check bits increases the error detection capability of the code, and that is right. Much effort has been expended to find powerful codes that can detect and correct a significant number of errors while still being easy to encode and decode.

Now, we will look intuitively at how codewords are selected to allow error correction.

3.4 Codeword Selection Criteria

We suspect, however, that the selection of codewords in a more error tolerant coding system would have to be done by a deeper method of which the parity example above is only a special case. What criterion do we use to select codewords?

The criterion is related to the relative distances between objects in space, but a space of perhaps many dimensions. For example, when placing fenceposts on a one-dimensional line, the farmer spaces them evenly to maximize and equalize the support given to each part of the fence. Consider our codewords spaced along the fence.



The subset of our field elements which are valid codewords are the fenceposts. Two other invalid codewords are noted by arrows. Errors occur during communication change the codewords, moving them along the fence line. To maximize our chances of guessing the correct codeword in spite of the errors, we need to likewise space our codewords evenly, as in the diagram. For that we use a concept called Hamming distance.

Hamming distance (or simply distance) is a very simple tool. To find the distance between two binary words, we just count the number of bits differing. For example, the distance between binary 01010011 and 01011100 is four, because the four rightmost bits differ. The distance between 10100 and 11001 is three bits.

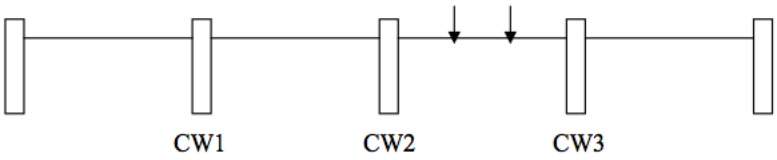
This computation can be performed easily by using the logical exclusive OR function:

```
    10100
xor 11001
    -----
    01101
```

The result has three ones, or we say a weight of three.

Given a requirement for 256 codewords in a larger set of 1024, for example, our task is to find a method of selecting the 256 valid codewords that maximizes the distance between them. Then, when errors occur, we can make a good estimate as to what correct codeword was transmitted. Each code has a minimum distance, called d that represents this value.

Now just from the minimum distance between the codewords we can draw a powerful conclusion. Consider the codewords arrayed on the fence again. If the distance between the codewords is three bits, then how many errors can we correct and be sure that we have not over corrected?



Looking at the diagram above, assume that codeword two has been transmitted. If one error occurs, it takes us one bit closer to codeword three, but we can see from the diagram that the obvious selection of the corrected codeword is still codeword two. If two errors occur, then the received codeword is closer now to codeword three than codeword two, and the decoder will select codeword three as the right one, which is a mistake.

In real life, this situation is multidimensional, with each codeword having many close neighbours in its field, distance-wise. But even this simple example suggests that the number of errors that we can correct is t = (d-1)/2, or half the distance, not including the middle bit for odd values of d. For d = 3, t = 1. For d = 4, t = 1 still.

Note that if we are not correcting errors, we can detect more than t. In the example, we could detect as many as two bit errors in either direction from codeword two. The number of detectable errors is in general d-1, because d errors would transform one codeword into another.

That brings up an important notion. A large number of errors in a codeword (d or more) can possibly transform a codeword into another valid, but unintended codeword. This situation is called an undetectable error. For example, if two bit errors occurred in the transmission of the letter "A" with parity bit (01000001), it could be mistaken for a "C" (11000011). To guard against this, communications engineers sometimes use an additional overall check code that tests the entire message for validity.

3.5 Two Linear Block Codes

The first code developed was the Hamming code, in 1950. It actually consists of a whole class of codes with the following characteristics:

- Block Length: n = 2^m - 1
- Information Bits: k = 2^m - m - 1
- Parity Check Bits: n - k = m
- Correctable Errors: t = 1

These conditions are true for $m > 2$. For example, with m = 4, there are n = 15 total bits per block or codeword, k = 11 information bits, n - k = 4 parity check bits, and the code can correct t = 1 error. A representative codeword would be:

10010100101 0010

; where the four bits on the right (0010) are the parity checkbits. By choosing the value of m, we can create a single error correcting code that fits our block length and correction requirements. This one is customarily denoted a (15, 4) code, telling us the total number of bits in a codeword (15) and the number of information bits (4).

We omit the details of encoding and decoding the Hamming code here because such will be covered in detail for the BCH code, later. The *Golay* code is another code, more powerful than the Hamming code, and geometrically interesting. This

(23, 12) code was discovered by Marcel J. E. Golay in 1949. It may also be extended using an overall parity bit to make a (24, 12) code. The minimum distance is seven, so it can detect up to six errors, or correct up to t = (7 - 1)/2 = 3 errors.

The aspect of the *Golay* and Hamming codes that makes them interesting is the fact that they are perfect. With any code, the codewords can be considered to reside within spheres packed into a region of space. The entire space is GF(2^m). Each sphere contains a valid codeword at its center and also all the invalid codewords that correct to the valid codeword, those being a distance of three or fewer bits from the center in the case of the *Golay* code (t = 3). If there are orphan binary words outside the spheres, then the code is termed imperfect.

Just how many codewords are in a sphere? With the *Golay* code, we first have the valid codeword. Then add the invalid codewords produced by introducing a single error in each of the 23 bits of the valid codeword, C(n, 1) = 23. Add to that the invalid codewords produced by introducing two and three errors in the valid codeword, C(n, 2) = 253, and C(n, 3) = 1771. Adding these up, we see the sphere contains 2048 = 2^{11} words.

There are also 4096 = 2^{12} total valid codewords (and spheres) in the *Golay* code, so the sum of all the codewords in all the spheres is $2^{11} \cdot 2^{12} = 2^{23}$, and that is the entire set of 23-bit binary words in GF(2^{23}). So there is no binary word in GF(2^{23}) that is not correctable to one of the 4096 valid codewords. That is a perfect code. An imperfect code has some elements in GF(2^m) outside of any such sphere, so a correction algorithm may not produce a useful result with such elements.

The abstract beauty of this structure is remarkable, but even more remarkable is the fact that perfect codes are rare. Pless and others have proven this fact, that the only nontrivial multiple error correcting perfect binary codes are equivalent to the binary Golay (23, 12) code. This sweeping conclusion comes about from a result that states that a perfect binary (n, k) code that corrects t errors must have n, k, and t satisfy the following relationship:

$$2^k \sum_{i=0}^{t} \binom{n}{i} = 2^n$$

The proof concludes that there are only a few values of n, k, and t that provide equality, indicating a perfect code. For the binary Golay code, the expression works out to:

$$2^{12}\left(\binom{23}{0}+\binom{23}{1}+\binom{23}{2}+\binom{23}{3}\right)=2^{12}\,(1+23+253+1771)=2^{23}$$

The binary Hamming codes are perfect as well, and there is a ternary Golay (11, 6) code with minimum distance 5 that is perfect. Aside from some other trivial codes that are of no practical interest (repetition codes decoded with majority logic gating), that is the extent of the perfect codes. One might suppose that there is some n-dimensional space where another perfect code exists, but that is not the case.

3.6 Main Event: The BCH Code

The BCH abbreviation stands for the discoverers, Bose and Chaudhuri (1960), and independently Hocquenghem (1959). These codes are multiple error correcting codes and a generalization of the Hamming codes. These are the possible BCH codes for $m > 3$ and $t < 2^m - 1$:

- Block Length: n = 2^m - 1
- Parity Check Bits: $n - k \leq mt$
- Minimum distance: $d \geq 2t + 1$

The codewords are formed by taking the remainder after dividing a polynomial representing our information bits by a generator polynomial. The generator polynomial is selected to give the code its characteristics. All codewords are multiples of the generator polynomial.

Let us turn to the construction of a generator polynomial. It is not simply a minimal, primitive polynomial as in our example where we built GF(16). It is actually a combination of several polynomials corresponding to several powers of a primitive element in GF(2^m).

The discoverers of the BCH codes determined that if α is a primitive element of GF(2^m), the generator polynomial is the polynomial of lowest degree over GF(2) with $\alpha, \alpha^2, \alpha^3, ..., \alpha^{2t}$ as roots. The length of a codeword is $2^m - 1$ and t is the number of correctable errors. Lin concludes that the generator is the least common multiple of the minimal polynomials of each α^i term. A simplification is possible because every even power of a primitive element has the same minimal polynomial

as some odd power of the element, halving the number of factors in the polynomial. Then $g(x) = lcm(m_1(x), m_3(x), ..., m_{2t-1}(x))$.

These BCH codes are called primitive because they are built using a primitive element of GF(2^m). BCH codes can be built using non-primitive elements, too, but the block length is typically less than $2^m - 1$.

As an example, let us construct a generator polynomial for BCH(31,16). Such a codeword structure would be useful in simple remote control applications where the information transmitted consists of a device identification number and a few control bits, such as "open door" or "start ignition."

This code has 31 codeword bits, 15 check bits, corrects three errors (t = 3), and has a minimum distance between codewords of 7 bits or more. Therefore, at first glance we need 2t - 1 = 5 minimal polynomials of the first five powers of a primitive element in GF(32). But the even powers' minimal polynomials are duplicates of odd powers' minimal polynomials, so we only use the first three minimal polynomials corresponding to odd powers of the primitive element.

The field we are working in is GF(32), shown below in figure 3.3. This was generated using primitive polynomial $x^5 + x^2 + 1$ over GF(32).

Field of 32 elements generated by $x^5 + x^2 + 1$		
Power Form	n-Tuple Form	Polynomial Form
0	00000	0
1	00001	1
α	00010	α
α^2	00100	α^2
α^3	01000	α^3
α^4	10000	α^4
α^5	00101	$\alpha^2 + 1$
α^6	01010	$\alpha^3 + \alpha$
α^7	10100	$\alpha^4 + \alpha^2$
α^8	01101	$\alpha^3 + \alpha^2 + 1$
α^9	11010	$\alpha^4 + \alpha^3 + \alpha$
α^{10}	10001	$\alpha^4 + 1$
α^{11}	00111	$\alpha^2 + \alpha + 1$
α^{12}	01110	$\alpha^3 + \alpha^2 + \alpha$
α^{13}	11100	$\alpha^4 + \alpha^3 + \alpha^2$
α^{14}	11101	$\alpha^4 + \alpha^3 + \alpha^2 + 1$
α^{15}	11111	$\alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1$
α^{16}	11011	$\alpha^4 + \alpha^3 + \alpha + 1$
α^{17}	10011	$\alpha^4 + \alpha + 1$
α^{18}	00011	$\alpha + 1$
α^{19}	00110	$\alpha^2 + \alpha$
α^{20}	01100	$\alpha^3 + \alpha^2$
α^{21}	11000	$\alpha^4 + \alpha^3$
α^{22}	10101	$\alpha^4 + \alpha^2 + 1$
α^{23}	01111	$\alpha^3 + \alpha^2 + \alpha + 1$
α^{24}	11110	$\alpha^4 + \alpha^3 + \alpha^2 + \alpha$
α^{25}	11001	$\alpha^4 + \alpha^3 + 1$
α^{26}	10111	$\alpha^4 + \alpha^2 + \alpha + 1$
α^{27}	01011	$\alpha^3 + \alpha + 1$
α^{28}	10110	$\alpha^4 + \alpha^2 + \alpha$
α^{29}	01001	$\alpha^3 + 1$
α^{30}	10010	$\alpha^4 + \alpha$

Figure 3.3: Field of 32 elements generated by a generator.

We need first a primitive element. Well, α is a primitive element in GF(32). Next we need the minimal polynomials of the first three odd powers of α. Tables of minimal polynomials appear in most texts on error control coding. Lin and Costello, Pless, and Rorabaugh exhibit algorithms for finding them using cyclotomic cosets. From Lin and Costello, the first three odd power of α minimal polynomials are:

- $\alpha : m_1(x) = x^5 + x^2 + 1$
- $\alpha^3 : m_3(x) = x^5 + x^4 + x^3 + x^2 + 1$
- $\alpha^5 : m_5(x) = x^5 + x^4 + x^2 + x + 1$

Therefore, $g(x) = lcm(m_1(x), m_3(x), m_5(x)) = m_1(x)m_3(x)m_5(x)$ (since these are irreducible).

So $g(x) = (x^5 + x^2 + 1)(x^5 + x^4 + x^3 + x^2 + 1)(x^5 + x^4 + x^2 + x + 1) = x^{15} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1$.

To encode a block of bits, let us first select as our information the binary word 1000001 for the letter "A" and call it f(x), placing it in the 16-bit information field . Next, we append a number of zeros equal to the degree of the generator polynomial (fifteen in this case). This is the same as multiplying f(x) by x^{15}. Then we divide by the generator polynomial using binary arithmetic (information bits are **bold**):

```
1000111110101111)0000000010000010000000000000000
                 1000111110101111
                 ----------------
                     1101101011110000
                     1000111110101111
                     ----------------
                      1010101010111110
                      1000111110101111
                      ----------------
                        100101000100010
```

The quotient is not used and so we do not even write it down. The remainder is 100101000100010, or $x^{14} + x^{11} + x^9 + x^5 + x$ in polynomial form, and of course it has degree less than our generator polynomial, g(x). Thus the completed codeword is:

Information	Checkbits
0000000001000001	100101000100010

This method is called *systematic encoding* because the information and check bits are arranged together so that they can be recognized in the resulting codeword. Nonsystematic encoding scrambles the positions of the information and check bits. The effectiveness of each type of code is the same; the relative positions of the bits

are of no matter as long as the encoder and decoder agree on those positions. To test the codeword for errors, we divide it by the generator polynomial:

```
1000111110101111)0000000001000011001010001000 10
                 1000111110101111
                 ----------------
                     1100100001111000
                     1000111110101111
                     ----------------
                      1000111110101111
                      1000111110101111
                      ----------------
                                     0
```

The remainder is zero if there are no errors. This makes sense because we computed the checkbits (r(x)) from the information bits (f(x)) in the following way:

$$f(x)x^n = q(x)g(x) + r(x)$$

The operation $f(x)x^n$ merely shifts f(x) left n places. Concatenating the information bits f(x) with the checkbits r(x) and dividing by g(x) again results in a remainder, r'(x), of zero as expected because:

$$f(x)x^n + r(x) = q(x)g(x) + r'(x)$$

If there are errors in the received codeword, the remainder, r'(x), is nonzero, assuming that the errors have not transformed the received codeword into another valid codeword. The remainder is called the syndrome and is used in further algorithms to actually locate the errant bits and correct them, but that is not a trivial matter.

The BCH codes are also cyclic, and that means that any cyclic shift of our example codeword is also a valid codeword. For example, we could interchange the information and checkbits fields in the last division above (a cyclic shift of 15 bits) and the remainder would still be zero.

3.7 Decoding the BCH(31,16) Code

Determining where the errors are in a received codeword is a rather complicated process. (The concepts here are from the explanation of Lin and Costello.) Decoding involves three steps:

1. Compute the syndrome from the received codeword.
2. Find the error location polynomial from a set of equations derived from the syndrome.
3. Use the error location polynomial to identify errant bits and correct them.

We have seen that computing the syndrome is not difficult. However, with the BCH codes, to implement error correction we must compute several components which together comprise a syndrome vector. For a t error correcting code, there are 2t components in the vector, or six for our triple error correcting code. These are each formed easily using polynomial division, as above, however the divisor is the minimal polynomial of each successive power of the generating element, α.

Let v(x) be our received codeword. Then S_i = v(x) mod $m_i(x)$, where $m_i(x)$ is the minimal polynomial of α^i. In our example,

- $S_1(x)$ = v(x) mod $m_1(x)$
- $S_2(x)$ = v(x) mod $m_2(x)$
- $S_3(x)$ = v(x) mod $m_3(x)$
- $S_4(x)$ = v(x) mod $m_4(x)$
- $S_5(x)$ = v(x) mod $m_5(x)$
- $S_6(x)$ = v(x) mod $m_6(x)$

Now in selecting the minimal polynomials, we take advantage of that property of field elements whereby several powers of the generating element have the same minimal polynomial. If f(x) is a polynomial over GF(2) and α is an element of GF(2^m), then if $b = 2^i, \alpha^b$ is also a root of f(x) for $i \leq 0$[13]. These are called conjugate elements. From this we see that all powers of α such as $\alpha^2, \alpha^4, \alpha^8, \alpha^{16}, ...$ are roots of the minimal polynomial of α. In GF(32) which applies to our example, we must find the minimal polynomials for α through α^6. The six minimal polynomials are:

- $m_1(x) = m_2(x) = m_4(x) = x^5 + x^2 + 1$
- $m_3(x) = m_6(x) = x^5 + x^4 + x^3 + x^2 + 1$
- $m_5(x) = x^5 + x^4 + x^2 + x + 1$

Next, we form a system of equations in α:

- $S_1(\alpha) = \alpha + \alpha^2 + ... + \alpha^n$
- $S_2(\alpha^2) = (\alpha)^2 + (\alpha^2)^2 + ... + (\alpha^n)^2$
- $S_3(\alpha^3) = (\alpha)^3 + (\alpha^2)^3 + ... + (\alpha^n)^3$
- $S_4(\alpha^4) = (\alpha)^4 + (\alpha^2)^4 + ... + (\alpha^n)^4$
- $S_5(\alpha^5) = (\alpha)^5 + (\alpha^2)^5 + ... + (\alpha^n)^5$
- $S_6(\alpha^6) = (\alpha)^6 + (\alpha^2)^6 + ... + (\alpha^n)^6$

It turns out that each syndrome equation is a function only of the errors in the received codeword. The α^i are the unknowns, and a solution to these equations yields information we use to construct an error locator polynomial. One can see that this system is underconstrained, there being multiple solutions. The one we are looking for is the one that indicates the minimum number of errors in the received codeword (we are being optimistic).

The method of solution of this system involves elementary symmetric functions and Newton's identities and is beyond the scope of this paper. However, this method has been reduced to an algorithm by Berlekamp that builds the error locator polynomial iteratively. Using the notation of Lin and Costello, a t + 2 line table may be use to handle the bookkeeping details of the error correction procedure for binary BCH decoding. It is described next.

First, make a table (using BCH(31,16) as our example):

μ	$\sigma^{(\mu)}(x)$	d_μ	l_μ	2μ - l_μ
-½	1	1	0	-1
0	1	S_1	0	0
1				
2				
$t = 3$				

The BCH decoding algorithm follows:

1. Initialize the table as above. Set $\mu = 0$
2. If $d_\mu = 0$,then $\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x)$.Let $L = l_{\mu+1}$.
3. If $d_\mu \neq 0$, then find a preceding row (row ρ) with the most positive $2\mu - l_\mu$ and $d_\rho \neq 0$. Then $\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x) + d_\mu d_\rho^{-1} x^{2(\mu-\rho)} \sigma^{(\rho)}(x)$. If $\mu = t - 1$,terminate the algorithm.

4. $l_{\mu+1} = deg(\sigma^{(\mu+1)}(x))$.

5. $d_{\mu+1} = S_{2\mu+3}+\sigma_1^{(\mu+1)}S_{2\mu+2}+\sigma_2^{(\mu+1)}S_{2\mu+1}+...+\sigma_L^{(\mu+1)}S_{2\mu+3-L}\cdot\sigma_i$ is the coefficient of the i-th term in $\sigma(x)$.

6. Increment μ and repeat from step 2.

At each step we are computing the next approximation to the error locator polynomial $\sigma^{(\mu)}(x)$. Depending upon the result of the previous step, we may be required to add a correction term, d_μ to $\sigma^{(\mu)}(x)$. When we have completed step t - 1, $\sigma^{(\mu)}(x)$ is the final error locator polynomial if it has degree less than or equal to t. If the degree is greater than t, then the codeword cannot be corrected (there are more than t errors).

Let us work out an example. Given our sample codeword (0000000001000001100 101000100010) we introduce three errors as if it were a corrupt received codeword, v(x) = 000**1**0000**1**1000001100100**0**000100010. Now we set to work computing syndrome components. Remember that the check polynomials are, with their binary equivalents,

- $m_1(x) = m_2(x) = m_4(x) = x^5 + x^2 + 1$ (100101),
- $m_3(x) = m_6(x) = x^5 + x^4 + x^3 + x^2 + 1$(111101),
- $m_5(x) = x^5 + x^4 + x^2 + x + 1$(110111),

; so we have three divisions to do to find six syndrome components. These are done by simple binary division, as above, details omitted.

- $S_1(x) = \text{v(x) mod } m_1(x) = x^2$
- $S_2(x) = \text{v(x) mod } m_2(x) = x^2$
- $S_3(x) = \text{v(x) mod } m_3(x) = x^4 + x^3 + x + 1$
- $S_4(x) = \text{v(x) mod } m_4(x) = x^2$
- $S_5(x) = \text{v(x) mod } m_5(x) = x^4 + x$
- $S_6(x) = \text{v(x) mod } m_6(x) = x^4 + x^3 + x + 1$

We find $S_i(\alpha^i)$ by substituting α^i into the equations above, reducing using the table we derived for GF(32) when necessary. Remember that $\alpha^5 = \alpha^2 + 1$.

- $S_1(\alpha) = \alpha^2$
- $S_2(\alpha^2) = \alpha^4$
- $S_3(\alpha^3) = (\alpha^3)^4 + (\alpha^3)^3 + (\alpha^3) + 1 = \alpha^{14}$
- $S_4(\alpha^4) = \alpha^8$
- $S_5(\alpha^5) = (\alpha^5)^4 + (\alpha^5) = \alpha^{29}$
- $S_6(\alpha^6) = (\alpha^6)^4 + (\alpha^6)^3 + (\alpha^6) + 1 = \alpha^{28}$

Using the algorithm, we fill in the table.

μ	$\sigma^{(\mu)}(x)$	d_μ	l_μ	$2\mu - l_\mu$
-½	1	1	0	-1
0	1	$S_1 = \alpha^2$	0	0
1	$\alpha^2 x + 1$	α^{26}	1	1
2	$\alpha^{24}x^2 + \alpha^2 x + 1$	α^{20}	2	2
$t = 3$	$\alpha^{27}x^3 + \alpha^{11}x^2 + \alpha^2 x + 1$	—	—	—

Set $\mu = 0$. We see that $d_\mu \neq 0$, so we choose $\rho = -1/2$, and:

$\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x) + d_\mu d_\rho^{-1} x^{2(\mu-\rho)} \sigma^{(\rho)}(x) = \sigma^{(0)}(x) + d_0 d_{-1/2}^{-1} x^{2(0+1/2)} \sigma^{(-1/2)}(x) = 1 + (\alpha^2)(1)(x)(1) = \alpha^2 x + 1$.
Then, $l_{\mu+1} = deg(\sigma^{(\mu+1)}(x)) = deg(\alpha^2 x + 1) = 1$.
Finally, $d_{\mu+1} = S_{2\mu+3} + \sigma_1^{(\mu+1)} S_{2\mu+2} + \sigma_2^{(\mu+1)} S_{2\mu+1} + ... + \sigma_L^{(\mu+1)} S_{2\mu+3-L} = S_3 + \sigma_1^{(1)} S_2 = (\alpha^{14}) + \alpha^2(\alpha^4) = \alpha^{26}$.

Set $\mu = 1$. We see that $d_\mu \neq 0$, so we choose $\rho = 0$, and:

$\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x) + d_\mu d_\rho^{-1} x^{2(\mu-\rho)} \sigma^{(\rho)}(x) = \sigma^{(1)}(x) + d_1 d_0^{-1} x^{2(1-0)} \sigma^{(0)}(x) = (\alpha^2 x + 1) + (\alpha^{26})(\alpha^2)^{-1} x^2(1) = (\alpha^2 x + 1) + (\alpha^{24})x^2 = \alpha^{24}x^2 + \alpha^2 x + 1$.
Then, $l_{\mu+1} = deg(\sigma^{(\mu+1)}(x)) = deg(\alpha^{24}x^2 + \alpha^2 x + 1) = 2$.
Finally, $d_{\mu+1} = S_{2\mu+3} + \sigma_1^{(\mu+1)} S_{2\mu+2} + \sigma_2^{(\mu+1)} S_{2\mu+1} + ... + \sigma_L^{(\mu+1)} S_{2\mu+3-L} = S_5 + \sigma_1^{(2)} S_4 + \sigma_2^{(2)} S_3 = (\alpha^{29}) + \alpha^2(\alpha^8) + \alpha^{24}(\alpha^{14}) = \alpha^{20}$.

Set $\mu = 2$. We see that $d_\mu \neq 0$, so we choose $\rho = 1$, and:

$\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x) + d_\mu d_\rho^{-1} x^{2(\mu-\rho)} \sigma^{(\rho)}(x) = \sigma^{(2)}(x) + d_2 d_1^{-1} x^{2(2-1)} \sigma^{(1)}(x) = (\alpha^{24}x^2 + \alpha^2 x + 1) + (\alpha^{20})(\alpha^{26})^{-1} x^2(\alpha^2 x + 1) = (\alpha^{24}x^2 + \alpha^2 x + 1) + (\alpha^{25})x^2(\alpha^2 x + 1) = \alpha^{24}x^2 + \alpha^2 x + 1 + \alpha^{27}x^3 + \alpha^{25}x^2 = \alpha^{27}x^3 + \alpha^{11}x^2 + \alpha^2 x + 1$.

The final error locator polyonomial is $\sigma^{(\mu)}(x) = \alpha^{27}x^3 + \alpha^{11}x^2 + \alpha^2 x + 1$.

We next find the roots of $\sigma^{(\mu)}(x)$ in GF(32) by trial and error substitution. (There is a search algorithm due to Chen that is more efficient.) The roots are α^4, α^9, and α^{22}. The bit positions of the error locations correspond to the inverses of these roots, or α^{27}, α^{22}, and α^9, respectively. A polynomial corresponding to the error pattern would then be $e(x) = x^{27} + x^{22} + x^9$. Adding e(x) to the received codeword corrects the errors. Examining the original corrupt codeword we created,

$$\begin{array}{rl} & v(x) = 0001000011000001100100000100010 \\ \oplus & e(x) = 0001000010000000000001000000000 \\ \hline & c(x) = 0000000001000001100101000100010 \end{array}$$

and it is clear that the calculated error pattern matches the actual error pattern and c(x) matches our original codeword.

If there are no errors, then the syndromes all work out to zero. One to three errors produce the corresponding number of bits in e(x). More than three errors typically results in an error locator polynomial of degree greater than t = 3. However, it is again possible that seven bit errors could occur, resulting in a zero syndrome and a false conclusion that the message is correct. That is why most error correction systems take other steps to ensure data integrity, such as using an overall check code on the entire sequence of codewords comprising a message.

In practice, error correction is done in either software or hardware. The Berlekamp algorithm is complex and not too attractive when considered for high-speed communications systems, or operation on power limited microprocessors. One version of the BCH code is used in pagers and another in cell phones, so optimization of the algorithm for the application is important. A cursory scan of the literature shows efforts are being made to discover alternative methods of decoding BCH codes.

Chapter 4

Low-Density Parity-Check (LDPC)

4.1 Introduction

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

At the beginning, they started using Turbo-Codes. These were discovered in 1992 being finally a reliable codification scheme at the end of the 90s, when they were used in spacial applications. But finally, some years later, the great advances achieved in LDPC codes have made them become way better than Turbo-Codes. As a matter of fact, in 2003, DVB made some trials and LDPC managed to beat up to six different Turbo-Codes, thus being LDPC adopted for the second generation of DVB standards. At the end of this chapter we will see some comparisons between both types of coding.

This note constitutes an attempt to highlight some of the main aspects of the theory of low-density parity-check (LDPC) codes.

4.2 Shannon's Theorem

The year 1948 marks the birth of information theory. In that year, Claude E. Shannon, as i commented in chapter 1 (see page 12), published his epoch making paper on the limits of reliable transmission of data over unreliable channels and methods on how to achieve these limits. Among other things, this paper formalized the concept of information, and established bounds for the maximum amount of information that can be transmitted over unreliable channels. A communication channel is usually defined as a triple consisting of an input alphabet, an output alphabet, and for each pair (i, o) of input and output elements a transition probability $p(i, o)$. Semantically, the transition probability is the probability that the symbol o is received given that i was transmitted over the channel.

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

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

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

it takes a lot of time (especially when the code is large) and it may err; but it is the best we can do.

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

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

Before I close this section, let me give an example of two communication channels: the binary erasure channel (BEC), and the binary symmetric channel (BSC). These channels are described in figure 4.1. In both cases the input alphabet is binary, and the elements of the input alphabet are called bits. In the case of the binary erasure channel the output alphabet consists of 0, 1, and an additional element denoted e and called erasure. Each bit is either transmitted correctly (with probability $1-p$), or it is erased (with probability p). The capacity of this channel is $1-p$.

Figure 4.1: Two examples of channels: (a) The Binary Erasure Channel (BEC) with erasure probability p, and (b) The Binary Symmetric Channel (BSC) with error probability p.

In the case of the BSC both the input and the output alphabet is F_2. Each bit is either transmitted correctly with probability $1-p$, or it is flipped with probability p. This channel may seem simpler than the BEC at first sight, but in fact it is much more complicated. The complication arises since it is not clear which bits are flipped. (In the case of the BEC it is clear which bits are erased.) The capacity of

this channel is $1 + p \cdot log_2(p) + (1-p) \cdot log_2(1-p)$. Maximum likelihood decoding for this channel is equivalent to finding, for a given vector of length n over F_2, a codeword that has the smallest Hamming distance from the received word. It can be shown that maximum likelihood decoding for the BSC is NP-complete. In contrast, for linear codes maximum likelihood decoding on the BEC is polynomial time, since it can be reduced to solving a system of equations.

4.3 Algorithmic Issues

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

If the input alphabet has the structure of a field (for example the binary alphabet which yields the field F_2), then one can do better, at least as far as encoding goes. Elias and Golay independently introduced the concept of linear codes of block length n and dimension k defined as subspaces of the vector space F_n^2. Such codes have rate k/n (we will omit the unit bpc for binary codes from now on), and since they are linear, they can be described in terms of a basis consisting of k vectors of length n. A codebook can now be implicitly described in a natural manner by mapping a bit vector $(x_1, ..., x_k)$ to the vector obtained by taking linear combinations of the basis vectors given by the coefficients $x_1, ..., x_k$.

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

How about decoding? The decoding problem seems much tougher. As was mentioned above, the maximum likelihood problem on the BSC has been shown to be NP-hard for many classes of linear codes (e.g., general linear codes over F_q for any q). It is therefore unlikely to find polynomial time algorithms for maximum likelihood

decoding of general linear codes. One way to get around this negative result is to try to repeat the success story for the encoding problem and to specialize to subclasses of general linear codes. However, we have not been able to find subclasses of linear codes for which maximum likelihood decoding is polynomial time and which achieve capacity.

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

4.4 LPCD Codes

LDPC codes were invented by Robert Gallager in his PhD thesis. Soon after their invention, they were largely forgotten, and reinvented several times for the next 30 years. Their comeback is one of the most intriguing aspects of their history, since two different communities reinvented codes similar to Gallager's LDPC codes at roughly the same time, but for entirely different reasons.

LDPC codes are linear codes obtained from sparse bipartite graphs. Suppose that G is a graph with n left nodes (called message nodes) and r right nodes (called check nodes). The graph gives rise to a linear code of block length n and dimension at least $n-r$ in the following way: The n coordinates of the codewords are associated with the n message nodes. The codewords are those vectors $(c_1, ..., c_n)$ such that for all check nodes the sum of the neighboring positions among the message nodes is zero. Figure 4.2 gives an example.

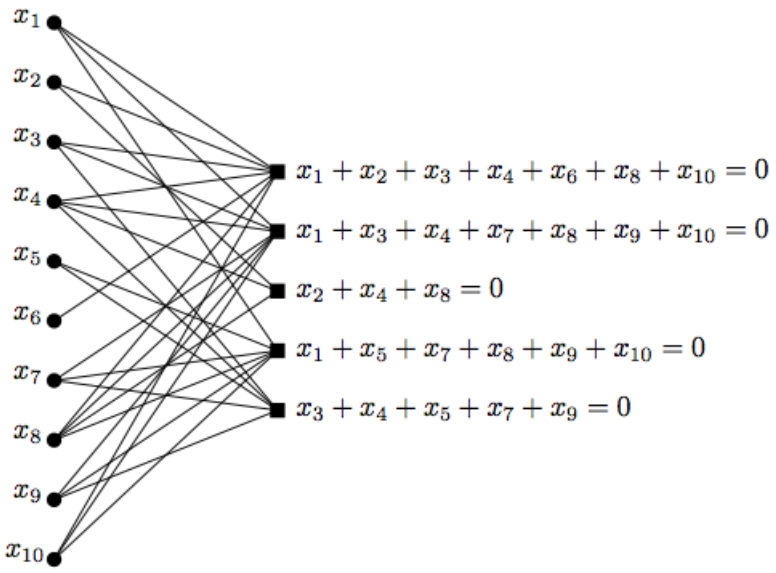


Figure 4.2: An LDPC code.

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

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

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

4.5 Decoding Algorithms: Belief Propagation

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

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

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

If x is an equiprobable random variable, then $L(x|y) = L(y|x)$ by Bayes' rule. Therefore, if $y_1, ..., y_d$ are independent random variables, then we have:

$$lnL(x|y_1, ..., y_d) = \sum_{i=1}^{d} lnL(x|y_i) \tag{4.1}$$

Now suppose that $x_1, ..., x_l$ are binary random variables and $y_1, ..., y_l$ are random variables. Denote addition over F_2 by $\oplus$. We would like to calculate $lnL(x_1 \oplus ... \oplus x_l|y_1, ..., y_l)$. Note that if $p = 2Pr[x_1 = 0|y_1] - 1$ and $q = 2Pr[x_2 = 0|y_2] - 1$, then $2Pr[x_1 \oplus x_2 = 0|y_1, y_2] - 1 = pq$. (Why?) Therefore, $2Pr[x_1 \oplus ... \oplus x_l = 0|y_1, ..., y_l] - 1 = \prod_{i=1}^{l} (2Pr[x_i = 0|y_i] - 1)$. Since $Pr[x_i = 0|y_i] = L(x_i|y_i)/(1 + L(x_i|y_i))$, we have that $2Pr[x_i = 0|y_i] - 1 = (L - 1)/(L + 1) = tanh(l/2)$, where $L = L(x_i|y_i)$ and

$l = lnL$. Therefore, we obtain:

$$lnL(x_1 \oplus ... \oplus x_l | y_1, ..., y_l) = ln \frac{1 + \prod_{i=1}^{l} (tanh(l_i/2)}{1 - \prod_{i=1}^{l} (tanh(l_i/2)} \tag{4.2}$$

; where $l_i = lnL(x_i|y_i)$. The belief propagation algorithm for LDPC codes can be derived from these two observations. In round 0, the check nodes send along all the outgoing edges their log-likelihoods conditioned on their observed value. For example, if the channel used is the BSC with error probability p, then the first message sent to all the check nodes adjacent to a message node is $ln(1-p) - lnp$ if the node's value is zero, and it is the negative of this value if the node's value is one. In all the subsequent rounds of the algorithm a check node c sends to an adjacent message node v a likelihood according to 4.2. A message node v sends to the check node c its log-likelihood conditioned on its observed value and on the incoming log-likelihoods from adjacent check nodes other than c using the relation 4.1.

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

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

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

; where C_v is the set of check nodes incident to message node v, and V_c is the set of message nodes incident to check node c.

The computations at the check nodes can be simplified further by performing them in the log- domain. Since the value of tanh(x) can be negative, we need to keep track of its sign separately. Let γ be a map from the real numbers $[-\infty, \infty]$ to F_2 x $[0, \infty]$ defined by $\gamma(x) := (sgn(x), -lntanh(|x|/2))$ (we set sgn(x) = 1 if $x \geq 1$ and sgn(x) = 0 otherwise.) It is clear that γ is bijective, so there exists an inverse function γ^{-1}. Moreover, $\gamma(xy) = \gamma(x) + \gamma(y)$, where addition is component-wise

in F_2 and in $[0, \infty]$. Then it is very easy to show that the last equation above is equivalent to:

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

We will use this representation when discussing density evolution later. In practice, belief propagation may be executed for a maximum number of rounds or until the passed likelihoods are close to certainty, whichever is first. A certain likelihood is a likelihood in which $lnL(x|y)$ is either ∞ or$-\infty$. If it is ∞,then $Pr[x = 0|y] = 1$, and if it is $-\infty$, then $Pr[x = 1|y] = 1$.

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

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

One question that might rise is about the relationship of belief propagation and maximum likelihood decoding. The answer is that belief propagation is in general less powerful than maximum likelihood decoding. In fact, it is easy to construct classes of LDPC codes for which maximum likelihood decoding can decode many more errors than belief propagation (one example is given by biregular bipartite graphs in which the common degree of the message nodes is very large).

4.6 Asymptotic Analysis of Belief Propagation and Density Evolution

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

for the l first rounds of the algorithm only if the neighborhood of a message node up to depth l is a tree.

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

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

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

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

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

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

If G is the group of real numbers with respect to multiplication, then $f \otimes g$ is the well-known convolution of real functions.

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

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

To assess the evolution of the densities at the check nodes, we need to use the operator γ introduced above. For a random variable X on $[-\infty, \infty]$ with density F let $\Gamma(F)$ denote the density of the random variable $\gamma(X)$. $\gamma(X)$ is defined on the group $G := F2$ x $[0, \infty]$. Therefore, the density of $\gamma(X) + \gamma(Y)$ is the convolution (over G) of $\Gamma(F)$ and $\Gamma(H)$, where H denotes the density of Y . Following the equation above and assuming independence via the independence assumption, we see that the common density g_i of the messages passed from check to message nodes at round i is $\Gamma - 1(\rho(\Gamma(f_i)))$, where $\rho(h) = \sum_d \rho_d h^{\otimes(d-)}$. All in all, we obtain the following recursion for the densities f_i:

$$f_{i+1} = f \otimes \lambda(\Gamma^{-1}(\rho(\Gamma(f_i)))) \tag{4.3}$$

This recursion is called density evolution. The reason for the naming should be obvious. I have not made the recursion very explicit. In fact, the operator Γ has not been derived at all.

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

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

4.7 Decoding on the BEC

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

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

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

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

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

1. Initialization: Initialize the values of all the check nodes to zero.
2. Direct recovery: For all message nodes v, if the node is received, then add its value to the values of all adjacent check nodes and remove v together with all edges emanating from it from the graph.
3. Substitution recovery: If there is a check node c of degree one, substitute its value into the value of its unique neighbor among the message nodes, add that value into the values of all adjacent check nodes and remove the message nodes and all edges emanating from it from the graph.

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

The decoding algorithm can be analyzed along the same lines as the full belief propagation. First, we need to find the expected density of the messages passed

at each round of the algorithm under the independence assumption. In this case, the messages are binary (either $+\infty$ or 0), hence we only need to keep track of one parameter, namely the probability p_i that the messages passed from message nodes to check nodes at round i of the algorithm is 0. Let q_i denote the probability that the message passed from check nodes to message nodes at round i of the algorithm is 0. Then, conditioned on the event that the message node is of degree d, we have $p_{i+1} = p \cdot q_i^{d-1}$. Indeed, a message from a message i node v to a check node c is 0 iff

$$waserasedandallthemessagescomingfromtheneighboringchecknodesotherthan$$

are 0, which is q_i^{d-1} under the independence assumption. Conditioned on i the event that the check node has degree d we have $q_i = 1 - (1 - p_i)^{d-1}$: the check node c sends a message $+\infty$ to the message node v iff all the neighboring message nodes except for v send a message $+\infty$ to c in the previous round. Under the independence assumption that probability is $(1 - p_i)^{d-1}$, which shows the identity.

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

$$p_{i+1} = p \cdot \lambda(1 - \rho(1 - p_i)) \tag{4.4}$$

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

$$p \cdot \lambda(1 - \rho(1 - x)) < x for x \in (0, p) \tag{4.5}$$

for successful decoding which was first proved and later reproduced. It is a useful and interesting exercise for the reader to show that 4.5 is identical to 4.3 in the case of the BEC.

Condition 4.5 was proved in a completely different way than explained here. A system of differential equations was derived whose solutions tracked the expected fraction of nodes of various degrees as the decoding process evolved. One of the solutions corresponds to the fraction of check nodes of reduced degree one during the algorithm. By keeping this fraction above zero at all times, it is guaranteed that in expectation there are always check nodes of degree one left to continue the decoding process. To show that the actual values of the random variables are

sharply concentrated around their computed expectations, a large deviation result was derived which is not unsimilar to Azuma's inequality for martingales.

Condition 4.5 can be used to calculate the maximal fraction of erasures a random LDPC code with given edge degree distributions can correct using the simple decoding algorithm. For example, consider a random biregular graph in which each message node has degree 3 and each check node has degree 6. (Such a graph is called a (3, 6)-biregular graph.) In this case $\lambda(x) = x^2$ and $\rho(x) = x^5$. What is the maximum fraction of erasures p? (In fact, this value is a supremum.) You can simulate the decoder on many such random graphs with a large number of message nodes. The simulations will show that on average around 42.9% erasures can be recovered. What is this value? According to 4.5 it is the supremum of all p such that $p(1-(1-x)^5)^2 < x$ on (0, p). The minimum of the function $x/(1-(1-x)^5)^2$ on(0,1) is attained at the unique root of the polynomial $9x^4-35x^3+50x^2-30x+5$ in the interval (0,1), and this is the supremum value for p. This value can be computed exactly, using formulas for the solution of the quartic.

As a side remark, I would like to mention an interesting result. First, it is not hard to see that the ratio r/n between the message and the check nodes equals $\int_0^1 \rho(x)dx/\int_0^1 \lambda(x)dx$. The rate of the code is at least $1-r/n$, and since the capacity of the erasure channel with erasure probability p is $1-p$, 4.5 should imply that $p \leq \int_0^1 \rho(x)dx/\int_0^1 \lambda(x)dx$ in a purely mechanical way (without using the interpretations above).

4.8 Hard Decision Decoding on the BSC

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

In both cases the messages passed between the message nodes and the check nodes consist of 0 and 1. Let me first describe the Gallager A algorithm: in round 0, the message nodes send their received values to all their neighboring check nodes. From that point on at each round a check node c sends to the neighboring message node v the addition (mod 2) of all the incoming messages from incident message

nodes other than v. A message node v sends the following message to the check node c: if all the incoming messages from check nodes other than c are the same value b, then v sends the value b to c; otherwise it sends its received value to c.

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

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

$$p_{i+1} = (1-p) \cdot \lambda(\frac{1-\rho(1-2p_i)}{2}) + p \cdot (1-\lambda(\frac{1+\rho(1-2p_i)}{2})) \tag{4.6}$$

If $\lambda(x), \rho(x)$, and p are such that p_i is monotonically decreasing, then decoding will be successful asymptotically with high probability, as long as the independence assumption is valid. For example, consider a (3,6)-biregular graph. In this case $\lambda(x) = x^2$ and $\rho(x) = x^5$, and the condition becomes:

$$(1-p) \cdot (\frac{1-(1-2x)^5}{2})^2 + p \cdot (1-(\frac{1+(1-2x)^5}{2})^2) < x \tag{4.7}$$

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

By Shannon's theorem the maximum error probability that a code of rate 1/2 can correct is the maximum p such that $1 + p \cdot log_2(p) + (1-p) \cdot log_2(1-p) =$

0.5. A numerical approximation shows that p is around 11%, which means that the Gallager A algorithm on the biregular (3, 6)-graph is very far from achieving capacity. Bazzi shows that for rate 1/2 the best graph for the Gallager A algorithm is the biregular (4, 8)-graph for which the maximum tolerable error probability is roughly 0.0475—still very far from capacity. This shows that this algorithm, though simple, is very far from using all the information that can be used.

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

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

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

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

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

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

The above one parameter recursions can be used to design codes that asymptotically perform very well for a given amount of noise. The method of choice in these cases is linear programming.

4.9 Completing the Analysis: Expander Based Arguments

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

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

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

To give the reader an idea of how these methods are used, I will exemplify them in the case of the BEC. Choose a graph with edge degree distributions given by $\lambda(x)$ and $\rho(x)$ at random. The analysis of the belief propagation decoder for the BEC implies that if condition 4.5 is true, then for any $\epsilon > 0$ there is an n_0 such that for all $n \geq n_0$ the erasure decoder reduces the number of erased message nodes below ϵn. The algorithm may well decode all the erasures, but the point is that the analysis does not guarantee that.

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

It is shown that for a random bipartite graph without message nodes of degree

one or two there is a constant ϵ depending on the rate of the induced code and on the degrees of the message nodes such that the graph is an $(\epsilon, 1/2)$-expander with high probability. On random graphs without message nodes of degrees one or two we see that the erasure decoding algorithm succeeds with high probability provided condition 4.5 is satisfied.

4.10 Achieving Capacity

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

There is no satisfactory answer to this question for arbitrary channels. What I mean by a satisfactory answer is an answer to the question whether subclasses of LDPC codes, for example LDPC codes with an appropriate degree distribution, will probably come arbitrarily close to the capacity of the channel. Optimization results for various channels, such as the Additive White Gaussian Noise (AWGN) channel and the BSC have produced specific degree distributions such that the corresponding codes come very close to capacity.

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

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

I believe that this question is one of the fundamental open questions in the asymptotic theory of LDPC codes. Some authors describe capacity-achieving sequences for the BEC for any erasure probability p. Let $\epsilon > 0$ be given, let $D := [1/\epsilon]$, and set:

$$\lambda(x) = \frac{1}{H(D)} \sum_{i=1}^{D} \frac{x^i}{i},\ \rho(x) = e^{\alpha(x-1)} \tag{4.8}$$

; where $\alpha = H(D)/p$. (Technically, $\rho(x)$ cannot define a degree distribution since

it is a power series and not a polynomial. But the series can be truncated to obtain a function that is arbitrarily close to the exponential.) We now apply 4.5:

$$p\lambda(1-\rho(1-x)) < -\frac{p}{H(D)}ln(\rho(1-x)) = \frac{\alpha p}{H(D)}x = x \tag{4.9}$$

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

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

The degree distribution above is called the Tornado degree distribution and the corresponding codes are called Tornado codes. These codes have many applications in computer networking which I will not mention here.

Tornado codes were the first class of codes that could provably achieve the capacity of the BEC using belief propagation. Since then many other distributions have been discovered.

For the ending, in the figure 4.3 is illustrated a comparison of a LDPC with other codes. As we can see, LDPC approximates very closely to Shannon limit.

[2]Actually, as was discussed before, 4.5 only shows that the fraction of erasures can be reduced to any constant fraction of the number of message nodes. To show that the decoding is successful all the way to the end, we need a different type of argument. Expansion arguments do not work for the corresponding graphs, since there are many message nodes of degree 2.

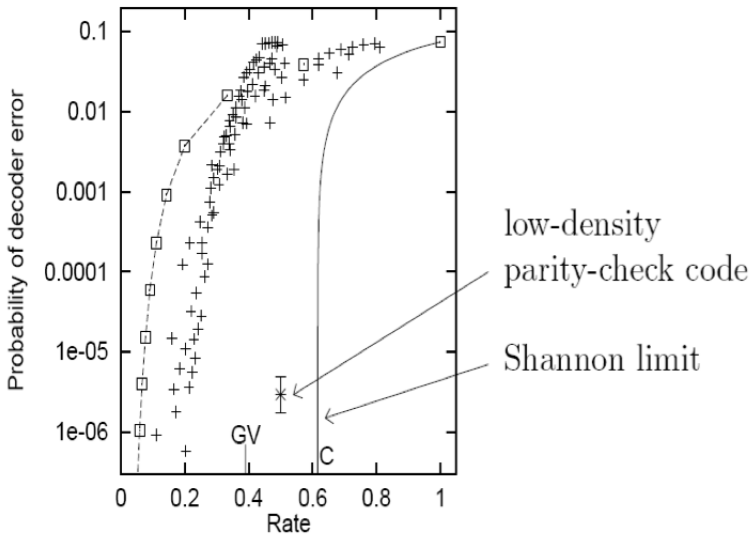


Figure 4.3: Example of approximation of LPDC to Shannon limit.

4.11 Graphs of Large Girth

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

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

$$1 + d(k-1)\frac{D^{\frac{l}{2}}-1}{D-1}$$

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

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

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

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

How does one construct bipartite graphs of large girth? There are a number of known techniques with origins in algebra and combinatorics. For example, it is very easy to construct optimal bipartite graphs of girth 6. Below I will give such a construction. Let C be a Reed-Solomon code of dimension 2 and length n over the field F_q. By definition, this code has q^2 codewords and the Hamming distance between any two distinct codewords is at least $n-1$. From C we construct a bipartite graph with q^2 message nodes and nq check nodes in the following way: The message nodes correspond to the codewords in C. The check nodes are divided in groups of q nodes each; the nodes in each such group corresponds to the elements of F_q. The connections in the graph are obtained as follows: A message node corresponding to the codewords $(x_1, ..., x_n)$ is connected to the check nodes corresponding to x_1 in the first group, to x_2 in the second group, ..., to x_n in the last group. Hence, all message nodes have degree n, and all check nodes have degree q, and the graph has in total nq^2 edges. Suppose that this graph has a cycle of length 4. This means that there are two codewords (corresponding to the two message nodes in the cycle) which coincide at two positions (corresponding to the two check nodes in the cycle). This is impossible by the choice of the code C, which shows that the girth of the graph is at least 6. To show the optimality of these graphs, we compare the number of check nodes to the above bound. Let $2l$ denote the girth of the graph. If $l = 3$, then $1+n(q-1) \leq q2$,which shows that $n \leq q+1$. By choosing $n = q+1$ we obtain optimal graphs of girth 6.

If $n < q + 1$, the graphs obtained may not be optimal, and their girth may be larger than 6. For $n \neq 2$ it is easy to see that the girth of the graph is indeed 6.

For $n = 2$ the girth is 8. (A cycle of length 6 in the graph corresponds to three codewords such that every two coincide in exactly one position. This is possible for $n > 2$, and impossible for n = 2.)

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

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

A discussion of graphs of large girth is not complete without at least mentioning Ramanujan graphs which have very large girth in an asymptotic sense. I will not discuss these graphs at all in this note.

4.12 Encoding Algorithms

An encoding algorithm for a binary linear code of dimension k and block length n is an algorithm that computes a codeword from k original bits $x_1, ..., x_k$. To compare algorithms against each other, it is important to introduce the concept of cost, or operations. For the purposes of this note the cost of an algorithm is the number of arithmetic operations over F_2 that the algorithm uses.

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

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

At this point there are at least two possible ways to go. One is to consider modifications of LDPC codes which are automatically equipped with fast encoding algorithms. The other is to try to find faster encoding algorithms for LDPC codes.

I will discuss both these approaches here, and outline some of the pro's and con's for each approach.

One simple way to obtain codes from sparse graphs with fast encoding is to modify the construction of LDPC codes in such a way that the check nodes have values, and the value of each check node is the addition of the values of its adjacent message nodes. (In such a case, it would be more appropriate to talk about redundant nodes, rather than check nodes, and of information nodes rather than message nodes. But to avoid confusion, I will continue calling the right nodes check nodes and the left nodes message nodes.) Figure 4.4 gives an example. The number of additions needed in this construction is upper bounded by the number of edges. So, efficient encoding is possible if the graph is sparse. The codewords in this code consist of the values of the message nodes, appended by the values of the check nodes.

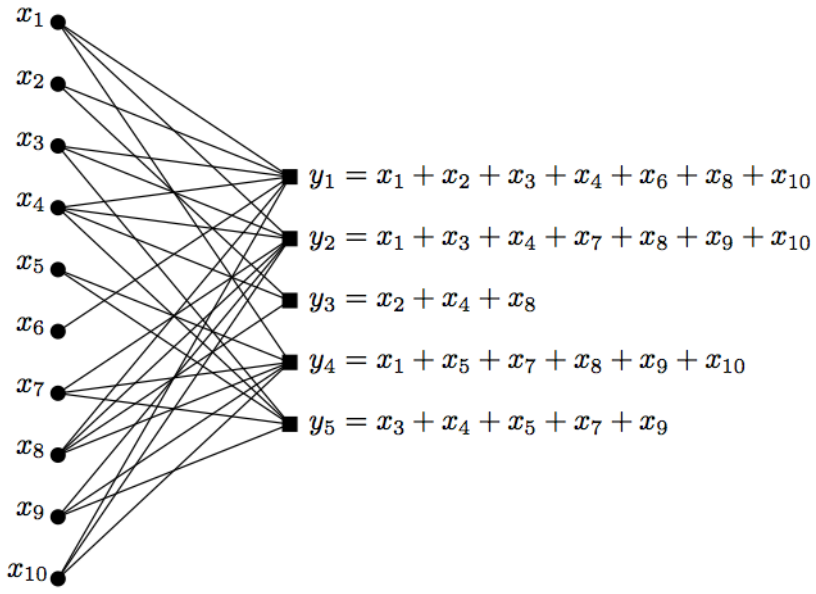


Figure 4.4: Construction with fast encoder

This construction leads to a linear time encoder, but it has a major problem with decoding. I will exemplify the problem for the case of the BEC. First, it is not clear that the belief propagation decoder on the BEC can decode all the erasures. This is because the check nodes can also be erased (in contrast to the case of LDPC codes where check nodes do not have a value per-se, but only keep track of the dependencies among the values of the message nodes). This problem is not an artifact of the non-optimal belief propagation decoder. Even the error probability of the maximum likelihood decoder is lower bounded by a constant in

this case. Let me elaborate. Suppose that a codeword is transmitted over a BEC with erasure probability p. Then an expected p-fraction of the message nodes and an expected p-fraction of the check nodes will be erased. Let Λ_d be the fraction of message nodes of degree d. Because the graph is random, a message node of degree d will have all its neighbors in the set of erased check nodes with probability p^d. This probability is conditioned on the event that the degree of the message node is d. So, the probability that a message node has all its neighbors within the set of erased check nodes is $\sum_d \Lambda_d p^d$, which is a constant independent of the length of the code. Therefore, no algorithm can recover the value of that message node.

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

The idea of using a cascade, though appealing in theory, is rather cumbersome in practice. For example, in the case of the BEC, the variance of the fraction of erasures per graph-layer will often be too large to allow for decoding. Moreover, maintaining all the graphs is rather complicated and may lead to deficiencies in the decoder.

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

More formally, the encoding process for RA codes is as follows. Consider an LDPC code whose graph has n message nodes and r check nodes. The value of the r check nodes is computed using the procedure introduced above, i.e., the value of each check node is the addition of the values of its adjacent message nodes. Let $(y_1, ..., y_r)$ denote the values of these check nodes. The redundant values $(s_1, ..., s_r)$ are now calculated as follows: $s_1 = y_1, s_2 = s_1 + y_2, ..., s_r = s_{r-1} + y_r$. (This explains

the phrase "accumulate.") An example of an RA code is given in figure 4.5 .

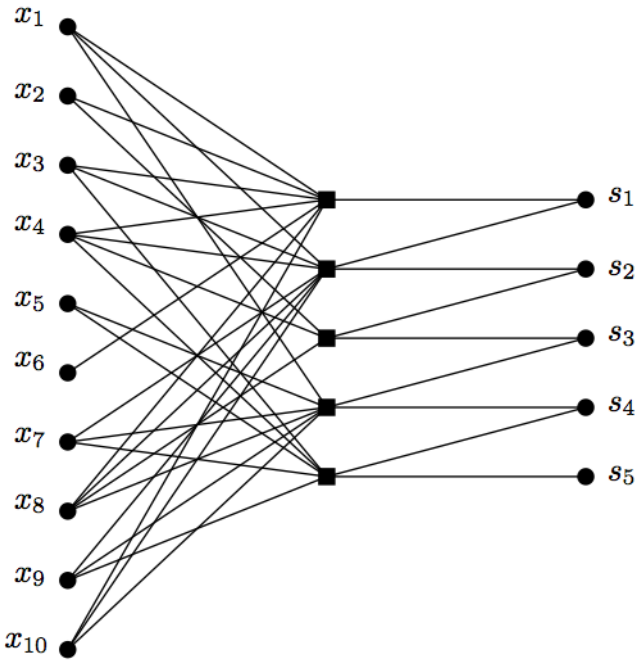


Figure 4.5: An irregular RA code. The left nodes are the information symbols, and the rightmost nodes are the redundant nodes. The squares in between are check nodes. Their values are computed as the addition of the values of their neighbors among the information nodes. The values of the redundant nodes are calculated so as to satisfy the relation that the values of the check nodes is equal to the addition of the values of the neighboring redundant nodes.

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

We conclude this section by mentioning the work of Richardson and Urbanke which provides an algorithm for encoding LDPC codes. They show that if the degree distribution $(\lambda(x), \rho(x))$ is such that $\rho(1-\lambda(x)) < x$ for $x \in (0,1)$, and such that $\lambda_2\rho'(1) > 1$, then the LDPC code can be encoded in linear time. The condition $\lambda_2\rho'(1) > 1$ has the following interpretation: consider the graph generated by the message nodes of degree 2. This graph induces a graph on the check nodes, by interpreting the message nodes of degree 2 as edges in that graph. Then $\lambda_2\rho'(1) > 1$ implies that this induced graph has a connected component whose number of vertices

is a constant fraction of the number of check nodes. This will be explained further, where this condition is actually used to devise a linear time encoding algorithm for a certain type of graphs.

4.13 Finite-Length Analysis

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

For these reasons it is important to develop other methods for analyzing the performance of message passing algorithms on small graphs. So far this has only started for the case of the BEC. In this case the analysis is of a combinatorial flavor. Given a bipartite graph, its associated code, and a set of erasures among the check nodes, consider the graph induced by the erased message nodes. A *stopping set* in this graph is a set of message nodes such that the graph induced by these message nodes has the property that no check node has degree one. The number of message nodes in the stopping set is called its size. It should be clear that belief propagation for the BEC stops prematurely (i.e., without recovering all the message nodes) if and only if this subgraph has a stopping set. Figure 4.6 gives some examples of graphs that are themselves stopping sets. Since unions of stopping sets are stopping sets, any finite graph contains a unique maximal stopping set (which may be the empty set). For a random bipartite graph the probability that belief propagation on the BEC has not recovered l message nodes at the point of failure (l can be zero) is the probability that the graph induced by the erased message nodes has a maximal stopping set of size l.

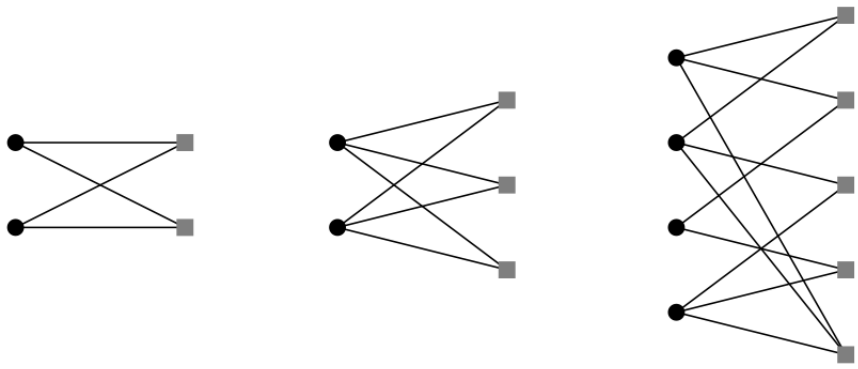


Figure 4.6: Examples of graphs that are stopping sets.

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

4.14 Examples

4.14.1 Example 1

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

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

First, let us see how many erasures this code can correct. The maximum fraction of erasures is 1 - R, where R is the rate, which is at least 1 - r/n. We should therefore

not expect to be able to correct more than an r/n-fraction of erasures, i.e., more than a $2/\alpha$-fraction. We now apply Condition 4.5: p is the maximum fraction of correctable erasures iff $p \cdot (1 - e^{-\alpha x}) < x$ for $x \in (0, p)$. Replacing x by px, this condition becomes:

$$1 - e^{-\beta x} < x,\ \beta = p\alpha \tag{4.10}$$

This latter condition has an interesting interpretation: the graph induced by the p-fraction of erasures is a random graph in the ensemble P(e,r), where e is the number of erasures, the expected value of which is en. For this graph the edge distribution from the point of view of the check nodes is $e^{-p\alpha(x-1)}$, and thus 4.5 implies 4.10.

Next, I will show that the maximum value of β for which Condition 4.10 holds is $\beta = 1$. For the function $1 - e^{-\beta x} - x$ to be less than 0 in (0, 1), it is necessary that the derivative of this function be non-positive at 0. The derivative is $\beta e^{-\beta x} - 1$, and its value at 0 is $\beta - 1$. Hence, $\beta \leq 1$ is a necessary condition for 4.10 to hold. On the other hand, if $\beta = 1$, then 4.10 is satisfied. Therefore, the maximum fraction of correctable erasures for a code in the ensemble P(n, r) is r/(2n), i.e., the performance of these codes is at half the capacity. So, the ensemble P(n,r) is not a very good ensemble in terms of the performance of belief propagation on the BEC.

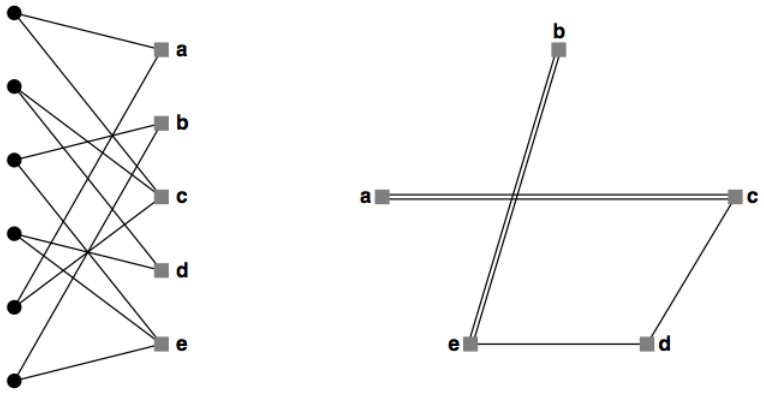


Figure 4.7: A graph with left degree 2 and its induced graph on the check nodes.

Before I go further in the discussion of codes in the ensemble P(n, r), let me give a different view of these codes. A bipartite graph with n message nodes and r check nodes in which each message node has degree 2 defines a (multi-)graph on the set of check nodes by regarding each message node as an edge in the graph in the obvious way. Multi-graphs and bipartite graphs with message degree 2 are in one-to-one correspondence to each other. In the following we will call the graph formed on the

check nodes of a bipartite graph G with message degree 2 induced by G. Figure 4.7 gives an example.

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

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

It is a well-known fact that a giant 2-core exists with high probability in a random graph with E edges in m nodes iff the average degree of a node is larger than 1, i.e., iff $E \geq m$. (A giant 2-core in the graph is a 2-core whose size is linear in the number of vertices of the graph.) Condition 4.10 is a new proof for this fact, as it shows that if the average degree of the induced graph is smaller than 1, then with high probability the graph does not contain a 2-core of linear size. It is also a well-known fact that this is precisely the condition for the random graph to contain a *giant component*, i.e., a component with linearly many nodes. Therefore, condition 4.10 can also be viewed as a condition on the graph not having a large component. (This condition is even more precise, as it gives the expected fraction of unrecovered message nodes at the time of failure of the decoder: it is p times the unique root of the equation $1 - x - e^{-\beta x}$ in the interval (0, 1); incidentally, this is exactly the expected size of the giant component in the graph, as is well-known in random graph theory.)

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

The condition $p\rho'(1) < 1$ is equivalent to the statement that the graph induced on the check nodes by the bipartite graph has a giant component. According to that paper, if a graph is chosen randomly on n nodes subject to the condition that for each d the fraction of nodes of degree d is essentially R_d (see the paper for a precise definition), then the graph has almost surely a giant component iff $\sum_d d(d-2)R_d > 0$. Consider the graph obtained from the restriction of the message nodes to a p-fraction, and consider the graph induced by this smaller graph on the

check nodes. Then, it is not hard to see that the degree distribution for this graph is $R(px+1-p)$, where $R(x) = c\int \rho(x)d$ and c is the average degree of the check nodes in the smaller bipartite graph. Therefore, the condition for the induced graph to have a giant component equals $pR''(1) < c$, where $R''(x)$ is the second derivative of $R(x)$. This is precisely equal to $p\rho'(1) < 1$, i.e., the stability condition is equivalent to the statement that the induced graph has a giant component. Incidentally, this is also equivalent to the condition that the graph does not have a giant 2-core, since stopping sets are equivalent to 2-cores in this setting. The fraction of nodes in the giant 2-core (if it exists) is equal to the unique solution of the equation $1 - \rho(1 - px) - x = 0$ in (0, 1).

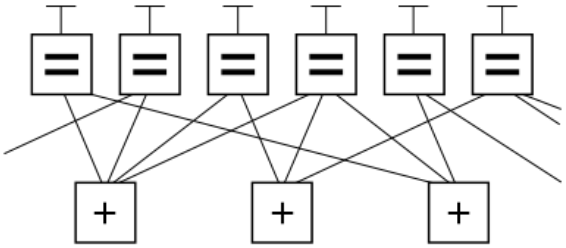
LDPC codes from graphs with left degree 2 play an important role. It states that small amounts of noise are correctable by belief propagation for an LDPC code with degree distribution given by $\lambda(x)$ and $\rho(x)$ if and only if $\lambda_2\rho'(1) < (\int_{-\infty}^{+\infty} f(x)e^{-x/2}dx)^{-1}$,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

4.14.2 Example 2

4.14.2.1 Encoding

As we said before, LDPC codes are defined by a sparse parity-check matrix. This sparse matrix is often randomly generated, subject to the sparsity constraints. These codes were first designed by Gallager in 1962.

Below is a graph fragment of an example LDPC code using Forney's factor graph notation. In this graph, n variable nodes in the top of the graph are connected to $(n - k)$ constraint nodes in the bottom of the graph. This is a popular way of graphically representing an (n, k) LDPC code. The bits of a valid message, when placed on the T's at the top of the graph, satisfy the graphical constraints. Specifically, all lines connecting to a variable node (box with an '=' sign) have the same value, and all values connecting to a factor node (box with a '+' sign) must sum, modulo two, to zero (in other words, they must sum to an even number).



Ignoring any lines going out of the picture, there are 8 possible 6-bit strings corresponding to valid codewords: (i.e., 000000, 011001, 110010, 101011, 111100, 100101, 001110, 010111). This LDPC code fragment represents a 3-bit message encoded as six bits. Redundancy is used, here, to increase the chance of recovering from channel errors. This is a (6, 3) linear code, with $n = 6$ and $k = 3$.

Once again ignoring lines going out of the picture, the parity-check matrix representing this graph fragment is:

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

In this matrix, each row represents one of the three parity-check constraints, while each column represents one of the six bits in the received codeword. In this example, the eight codewords can be obtained by putting the parity-check matrix H into this form $[-P^T|I_{n-k}]$ through basic row operations:

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

From this, the generator matrix G can be obtained as $[I_k|P]$ (noting that in the special case of this being a binary code $P = -P$), or specifically:

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Finally, by multiplying all eight possible 3-bit strings by G, all eight valid codewords are obtained. For example, the codeword for the bit-string '101' is obtained by:

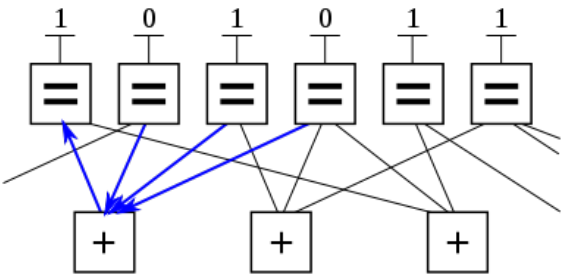
$$\begin{pmatrix}1 & 0 & 1\end{pmatrix}\begin{pmatrix}1 & 0 & 0 & 1 & 0 & 1\\ 0 & 1 & 0 & 1 & 1 & 1\\ 0 & 0 & 1 & 1 & 1 & 0\end{pmatrix} = \begin{pmatrix}1 & 0 & 1 & 0 & 1 & 1\end{pmatrix}.$$

4.14.2.2 Decoding

As with other codes, optimally decoding an LDPC code on the binary symmetric channel is an NP-complete problem, although techniques based on iterative belief propagation used in practice lead to good approximations. In contrast, belief propagation on the binary erasure channel is particularly simple where it consists of iterative constraint satisfaction.

For example, consider that the valid codeword, 101011, from the example above, is transmitted across a binary erasure channel and received with the first and fourth bit erased to yield ?01?11. Since the transmitted message must have satisfied the code constraints, the message can be represented by writing the received message on the top of the factor graph.

In this example, the first bit cannot yet be recovered, because all of the constraints connected to it have more than one unknown bit. In order to proceed with decoding the message, constraints connecting to only one of the erased bits must be identified. In this example, either the second or third constraint suffices. Examining the second constraint, the fourth bit must have been 0, since only a 0 in that position would satisfy the constraint. This procedure is then iterated. The new value for the fourth bit can now be used in conjunction with the first constraint to recover the first bit as seen below. This means that the first bit must be a 1 to satisfy the leftmost constraint.



Thus, the message can be decoded iteratively. For other channel models, the

messages passed between the variable nodes and check nodes are real numbers, which express probabilities and likelihoods of belief. This result can be validated by multiplying the corrected codeword r by the parity-check matrix H:

$$\mathbf{z} = \mathbf{Hr} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

Because the outcome z (the syndrome) of this operation is the 3×1 zero vector, the resulting codeword r is successfully validated.

4.15 Comparative: LDPC vs TC

A brief comparison of Turbo codes and LDPC codes will be given in this section, both in term of performance and complexity. The reader can find further information in appendix B (It is not included in the thesis since Turbo-codes are not used in the DVB standards). In order to give a fair comparison of the codes, we use codes of the same input word length when comparing. The rate of both codes is R = 1/2.

Figures 4.8 and 4.9 show the performance of Turbo codes and the LDPC codes with information length 1784 and 3568 respectively. The Turbo codes defined in the CCSDS (see bibliography) were used for obtaining the performance curves. For both code lengths the LDPC codes are better until a certain S/N is reached, respectively Eb/No = 1.15 and 1.1 dB. This characteristic 'error floor' or 'knee' was also typical of Turbo codes in earlier years, but this has become less of a problem lately due to the definition of good permutation matrices.

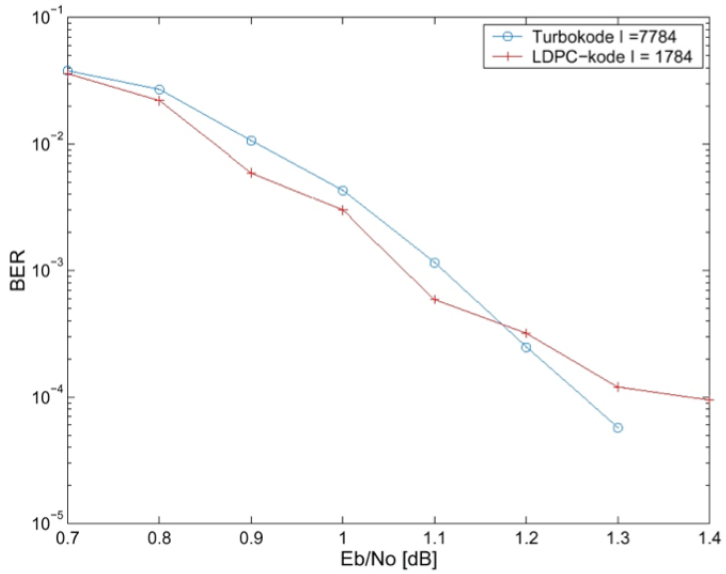


Figure 4.8: Turbo code and LDPC codes: I = 1784.

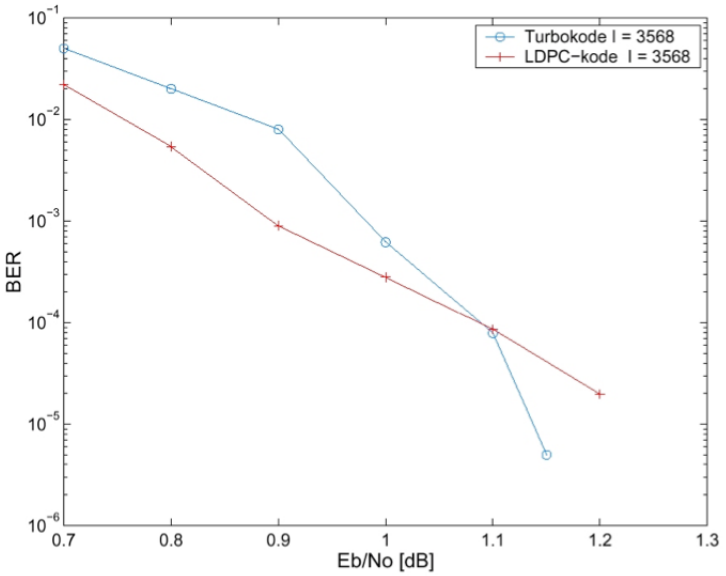


Figure 4.9: Turbo code and LDPC codes: I = 3568.

In order to compare the complexity of the codes, the number of multiplications, additions and complex operations like $log, tanh, e$ and $tanh^{-1}$ were counted in both codes. Figure 4.10 shows the number of operations per iteration for the LDPC and

Turbo codes of length I = 1784 and 3568.

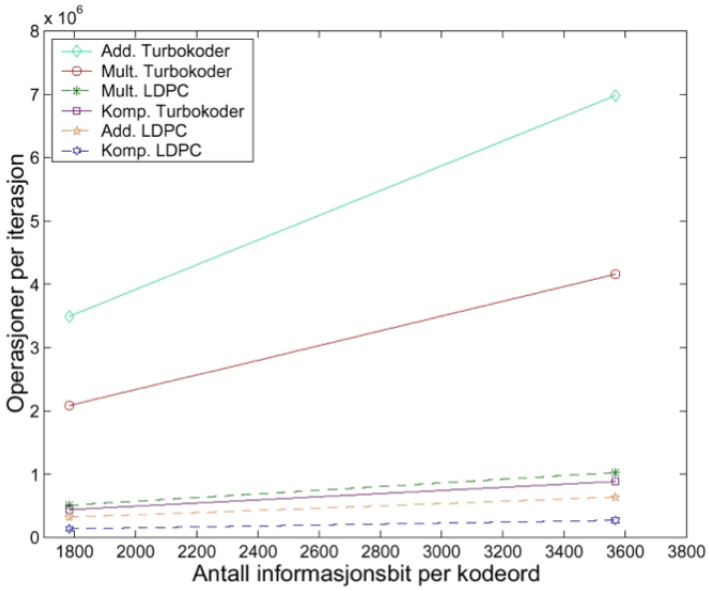


Figure 4.10: Complexity: The figure shows Additions (Add.), Multiplications (Mult.) and Complex operations (Komp.) per iteration for LDPC and Turbo codes for I = 1784 and I = 3568.

The number of iterations used may vary in the case of the LDPC codes, as the decoding quality may easily be checked, i.e. that if a valid code word is decoded, the decoding may be stopped. Hence, there will be less and less iterations performed the higher the S/N. Hence, in order to characterize the complexity, a representative number of operations per code word was computed at Eb/No = 1 dB. The maximum number of iterations was set to 100 for the smaller code and to 45 for the larger code. This was chosen in order to give a fair comparison with Turbo codes: With the above parameters the bit error rate for the LDPC code was 0.0038 and 0.00054 for the I = 1784 and I = 3568 codes respectively, whereas for the Turbo code the bit error rate was 0.0043 and 0.00062 respectively, i.e. the error rates were sufficiently similar for a fair comparison to be given. In the case of Turbo codes, the computation of complexity is simpler since the number of iterations used is pre-determined.

Figure 4.11 shows the number of operations per code word, separated into the different categories of operations. The total number of operations per code word is given in figure 4.12.

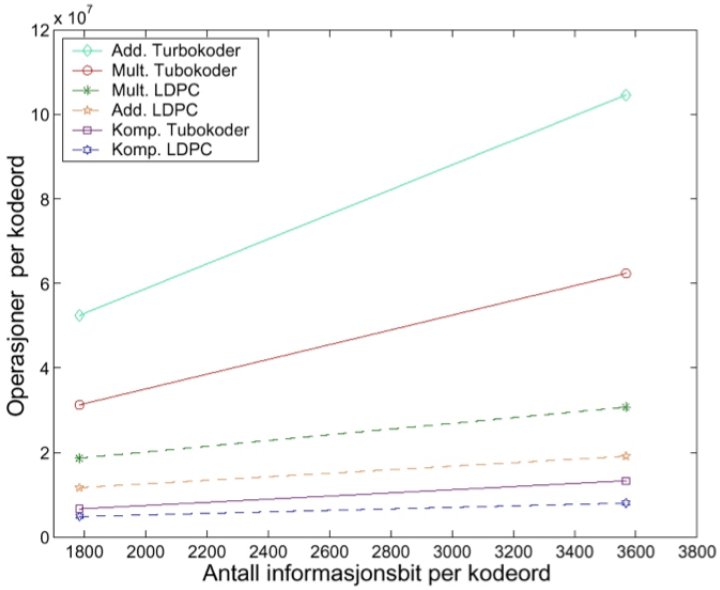


Figure 4.11: Complexity: The figure shows Additions (Add.), Multiplications (Mult.) and Complex operations (Komp.) per code word for LDPC and Turbo codes for I = 1784 and I = 3568.

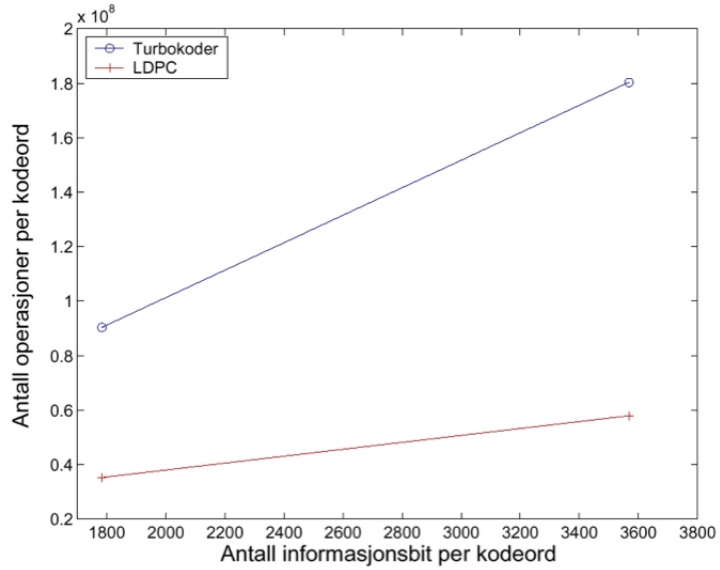


Figure 4.12: Complexity: The Figure shows the total number of operations (Additions + Multiplications + Complex operations) per code word for LDPC and Turbo codes for I = 1784 and I = 3568.

4.15.1 Conclusions

The results in the previous section show that LDPC codes have a significantly lower complexity than Turbo codes at the code lengths, performance and S/N that were considered here.

Turbo codes have a fixed number of iterations in the decoder. This implies that the time spent in the decoding and the bit rate out of the decoder, are constant entities. In contrast, the LDPC decoder stops when a legal code word is found, implying that there is potential for significantly reducing the amount of work to be done relative to Turbo codes. This also implies that the bit rate out of the decoder will vary, and a buffer system must be designed in order to make the bit rate constant. The LDPC decoder will become faster the higher the S/N.

An advantage with LDPC codes is that the decoders may be implemented in parallel. This has significant advantages when considering long codes.

4.16 Comparative: LDPC vs DBTC (Duo-Binary Turbo codes)

The performance comparison (figure 4.13) underlines the fact that the selection of an appropriate coding technique depends crucially on the target block length. For a code rate of 1/2, DBTC[3] outperform LDPC for block lengths up to 1728 (0.2 dB gain over LDPC for N = 576). Then LDPC start to progressively outperform DBTC (0.1 dB better for N = 4308).

[3]DBTC differ from classical by the fact that the information bits are encoded pair wise.

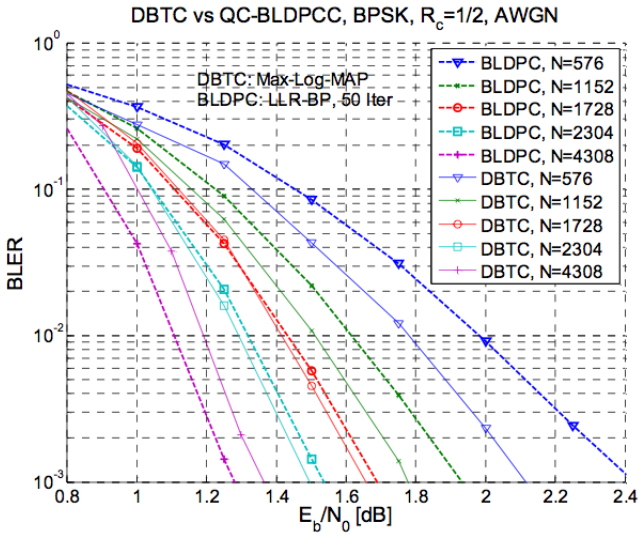


Figure 4.13: Performance comparison between DBTC and QC-BLDPCC, Rc=1/2.

In general, the performance loss by going from large to small block sizes is lower for DBTC than for LDPC. The threshold (in terms of block length) that separates these two regimes, however, depends on the code rate. When increasing the code rate to Rc = 3/4, a block length of 1152 is sufficient for the BDLPCC to achieve the same performance as the DBTC, and the difference observed for N = 576 is very small (figure 4.14).

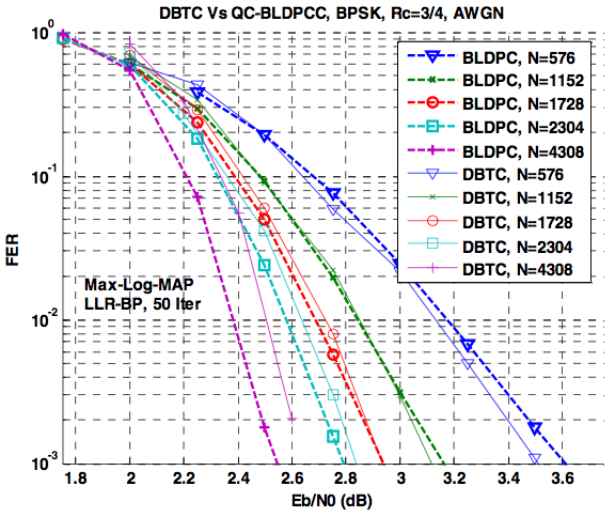


Figure 4.14: Performance comparison between DBTC and QC-BLDPCC, Rc=3/4.

4.16.1 Complexity-Performance Trade-Off

In the following figures 4.15 and 4.16, a packet length of N = 2304 bits is represented by purple markers, and N = 576 bits by blue markers. Diamond markers indicate results for DBTC. In the case of LDPC, we have to differentiate between the maximum complexity (maximum number of iterations, 20 in the considered setup) which will be represented by a circle marker. A triangle marker is used for the average complexity (average number of iterations).

Energy and Cycles are distinguished then by the use of non- filled or filled marker, respectively. For DBTC these figures coincide, as only simple arithmetic operations (ADD, etc.) are used. This will be illustrated by the use of a green edge color.

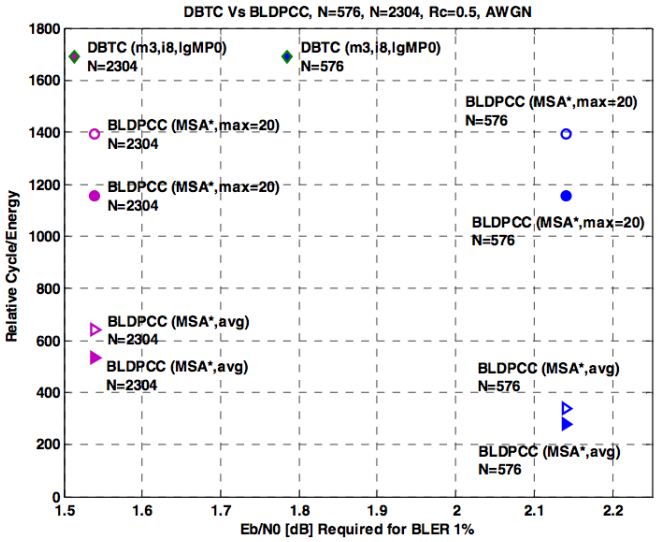


Figure 4.15: Complexity-Performance Trade-Off for QC-LDPCC and DBTC, Rc=1/2.

For the case of Rc=1/2, depicted in figure 4.15 above, DBTC offer a better complexity-performance trade-off than LDPC for low block sizes (N = 576), as they perform better at only slightly more energy consumption (w.r.t. the maximum iteration case for LDPC). However, for a higher block length (N = 2304), LDPC become more suitable, since their energy consumption saving is achieved at the expense of only minor performance degradation.

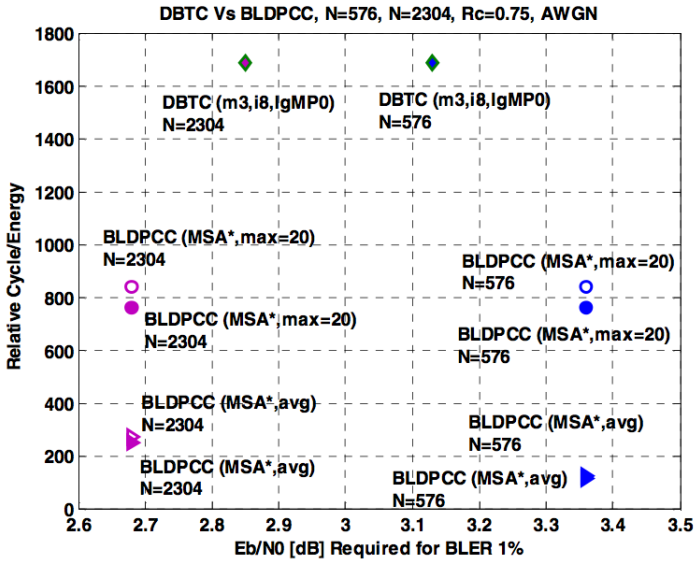


Figure 4.16: Complexity-Performance Trade-Off for QC-LDPCC and DBTC, Rc=3/4.

These two trends are reinforced throughout the remaining results, in figure 4.16 above. Indeed, the higher the code rate the less energy/cycles are required by LDPC (provided that the high code rate is achieved by using a code of a different rate rather than puncturing the rate 1/2 code). It might be worth mentioning that DBTC still outperform LDPC for lower block lengths, but their higher energy consumption in the present case of 8 internal decoder iterations seems to be the price to be paid for such performance. The presented complexity/performance trade-off is a necessary step towards fair comparison of these two main channel coding candidates DBTC and LDPC.

Although these results are quite informative, they are not yet sufficient for a further technological decision. Indeed, the number of gates and the memory size requirements, together with robustness to quantization (fixed point simulations), still have to be investigated. Based on current assessment and know-how, these two channel coding techniques look more like complementary than concurrent solutions (w.r.t. block length).

4.16.2 Conclusions

Codes based on sparse graphs and iterative decoding have made near Shannon limit error correction performance available also for practical applications. A large number of possible options have been identified and discussed in research and

standardization activities. However, the huge amount of codes available is rendering a selection difficult, not the least because practical implementation constraints (parallelization, fixed point implementation) are often neglected in the code design. Comparison results for different types of codes are also scarce.

This contribution provides a detailed comparison of the two techniques in terms of complexity, performance, and a number of implementation issues (parallelization, suitability for H-ARQ, etc.). However, we have to stress that iterative decoding schemes still suffer from quite low performance at short block lengths. Therefore, convolutional codes should still considered an option with block lengths below 200 information bits. Further improved DBTC or non-binary LDPCC might replace them in the future.

All codes show the same high degree of flexibility in terms of block sizes and code rates and support the construction of rate compatible code sets.

The domain of suitability (for a target BLER of 1%) of all candidates is summarized in figure 4.17 above. LDPCC are favored over DBTC for large block sizes due to their superior error correction performance in this regime (lower required bit SNR to achieve BLER of 1%). DBTC or alternatively CC should be used for small block sizes.

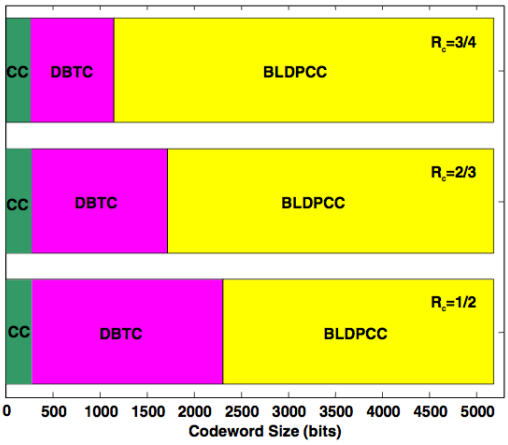


Figure 4.17: Complexity-Performance Trade-Off for QC-LDPCC and DBTC, Rc=3/4.

Recent improvements in the code design have increased the flexibility of LDPC and DBTC in terms of block sizes and code rates to an extent that the impact of choosing one or another of these FEC technique on the overall design of a wireless is becoming more and more limited, as soft-input soft-output versions of the decoder algorithms also exist for all techniques presented in figure 4.17. The FEC

en/decoding subblocks can hence be regarded as a “black box” by the rest of the system, and system designers can decide in favor of the coding technique that best suits the desired performance- (implementation) complexity trade-off only at the end of the design process.

Chapter 5

Functionality of BCH-LDPC in the new DVB standards

As we saw in chapter 2, all the new specifications of the new DVB standards make use of LDPC (Low-density parity-check) codes in combination with BCH (Bose-Chaudhuri-Hocquengham) to protect against high noise levels and interference (see figure 5.1). In comparison, the old standards, which made use of convolutional coding and Reed-Solomon, two further code rates have been added. Let's see how they function.

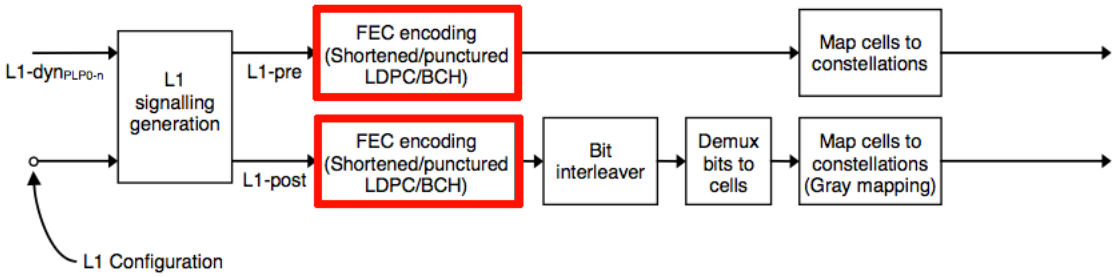


Figure 5.1: Bit Interleaved Coding and Modulation (BICM) in DVB-T2.

The FEC sub-system shall perform outer coding (BCH), Inner Coding (LDPC) and Bit interleaving. The input stream shall be composed of BBFRAMEs and the output stream of FECFRAMEs.

Each BBFRAME (K_{bch} bits) shall be processed by the FEC coding subsystem, to generate a FECFRAME (N_{ldpc} bits). The parity check bits (BCHFEC) of the

systematic BCH outer code shall be appended after the BBFRAME, and the parity check bits (LDPCFEC) of the inner LDPC encoder shall be appended after the BCHFEC field, as shown in figure 5.2.

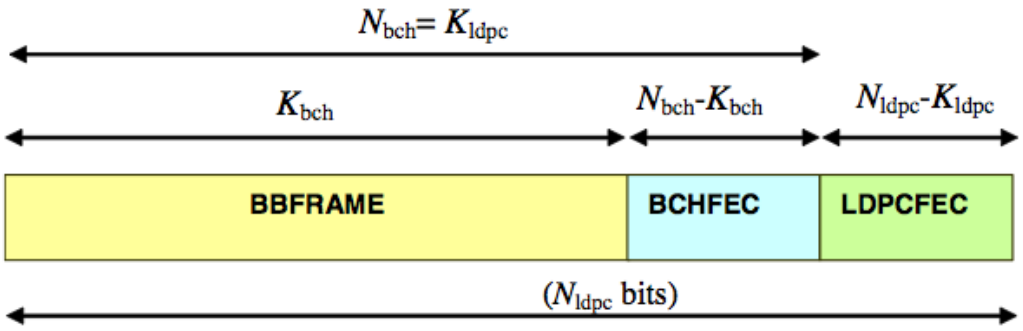


Figure 5.2: Format of data before bit interleaving (N_{ldpc} = 64 800 bits for normal FECFRAME, N_{ldpc} = 16 200 bits for short FECFRAME).

Table 5.3 gives the FEC coding parameters for the normal FECFRAME (N_{ldpc} = 64 800 bits) and table 5.4 for the short FECFRAME (N_{ldpc} = 16 200 bits).

LDPC Code	BCH Uncoded Block K_{bch}	BCH coded block N_{bch} LDPC Uncoded Block K_{ldpc}	BCH t-error correction	N_{bch}-K_{bch}	LDPC Coded Block N_{ldpc}
1/2	32 208	32 400	12	192	64 800
3/5	38 688	38 880	12	192	64 800
2/3	43 040	43 200	10	160	64 800
3/4	48 408	48 600	12	192	64 800
4/5	51 648	51 840	12	192	64 800
5/6	53 840	54 000	10	160	64 800

Figure 5.3: Coding parameters (for normal FECFRAME N_{ldpc} = 64 800).

LDPC Code identifier	BCH Uncoded Block K_{bch}	BCH coded block N_{bch} LDPC Uncoded Block K_{ldpc}	BCH t-error correction	N_{bch}-K_{bch}	Effective LDPC Rate K_{ldpc}/16 200	LDPC Coded Block N_{ldpc}
1/4 (see note)	3 072	3 240	12	168	1/5	16 200
1/2	7 032	7 200	12	168	4/9	16 200
3/5	9 552	9 720	12	168	3/5	16 200
2/3	10 632	10 800	12	168	2/3	16 200
3/4	11 712	11 880	12	168	11/15	16 200
4/5	12 432	12 600	12	168	7/9	16 200
5/6	13 152	13 320	12	168	37/45	16 200
NOTE: This code rate is only used for protection of L1-pre signalling and not for data.						

Figure 5.4: Coding parameters (for short FECFRAME $N_{ldpc} = 16\ 200$).

Note: For $N_{ldpc} = 64\ 800$ as well as for N_{ldpc} =16 200 the LDPC code rate is given by K_{ldpc}/N_{ldpc}. In table 5.3 the LDPC code rates for $N_{ldpc} = 64\ 800$ are given by the values in the 'LDPC Code' column. In table 5.4 the LDPC code rates for $N_{ldpc} = 16\ 200$ are given by the values in the 'Effective LDPC rate' column, i.e. for $N_{ldpc} = 16\ 200$ the 'LDPC Code identifier' is not equivalent to the LDPC code rate.

5.1 Outer encoding (BCH)

A t-error correcting BCH (N_{bch}, K_{bch}) code shall be applied to each BBFRAME to generate an error protected packet. The BCH code parameters for $N_{ldpc} = 64\ 800$ are given in table 5.3 and for $N_{ldpc} = 16\ 200$ in table 5.4. The generator polynomial of the t error correcting BCH encoder is obtained by multiplying the first t polynomials in table 5.5 for $N_{ldpc} = 64\ 800$ and in table 5.6 for $N_{ldpc} = 16\ 200$.

$g_1(x)$	$1+x^2+x^3+x^5+x^{16}$
$g_2(x)$	$1+x+x^4+x^5+x^6+x^8+x^{16}$
$g_3(x)$	$1+x^2+x^3+x^4+x^5+x^7+x^8+x^9+x^{10}+x^{11}+x^{16}$
$g_4(x)$	$1+x^2+x^4+x^6+x^9+x^{11}+x^{12}+x^{14}+x^{16}$
$g_5(x)$	$1+x+x^2+x^3+x^5+x^8+x^9+x^{10}+x^{11}+x^{12}+x^{16}$
$g_6(x)$	$1+x^2+x^4+x^5+x^7+x^8+x^9+x^{10}+x^{12}+x^{13}+x^{14}+x^{15}+x^{16}$
$g_7(x)$	$1+x^2+x^5+x^6+x^8+x^9+x^{10}+x^{11}+x^{13}+x^{15}+x^{16}$
$g_8(x)$	$1+x+x^2+x^5+x^6+x^8+x^9+x^{12}+x^{13}+x^{14}+x^{16}$
$g_9(x)$	$1+x^5+x^7+x^9+x^{10}+x^{11}+x^{16}$
$g_{10}(x)$	$1+x+x^2+x^5+x^7+x^8+x^{10}+x^{12}+x^{13}+x^{14}+x^{16}$
$g_{11}(x)$	$1+x^2+x^3+x^5+x^9+x^{11}+x^{12}+x^{13}+x^{16}$
$g_{12}(x)$	$1+x+x^5+x^6+x^7+x^9+x^{11}+x^{12}+x^{16}$

Figure 5.5: BCH polynomials (for normal FECFRAME $N_{ldpc} = 64\,800$).

$g_1(x)$	$1+x+x^3+x^5+x^{14}$
$g_2(x)$	$1+x^6+x^8+x^{11}+x^{14}$
$g_3(x)$	$1+x+x^2+x^6+x^9+x^{10}+x^{14}$
$g_4(x)$	$1+x^4+x^7+x^8+x^{10}+x^{12}+x^{14}$
$g_5(x)$	$1+x^2+x^4+x^6+x^8+x^9+x^{11}+x^{13}+x^{14}$
$g_6(x)$	$1+x^3+x^7+x^8+x^9+x^{13}+x^{14}$
$g_7(x)$	$1+x^2+x^5+x^6+x^7+x^{10}+x^{11}+x^{13}+x^{14}$
$g_8(x)$	$1+x^5+x^8+x^9+x^{10}+x^{11}+x^{14}$
$g_9(x)$	$1+x+x^2+x^3+x^9+x^{10}+x^{14}$
$g_{10}(x)$	$1+x^3+x^6+x^9+x^{11}+x^{12}+x^{14}$
$g_{11}(x)$	$1+x^4+x^{11}+x^{12}+x^{14}$
$g_{12}(x)$	$1+x+x^2+x^3+x^5+x^6+x^7+x^8+x^{10}+x^{13}+x^{14}$

Figure 5.6: BCH polynomials (for short FECFRAME $N_{ldpc} = 16\,200$).

The bits of the baseband frame form the message bits $M = (m_{K_{bch}-1}, m_{K_{bch}-2}, ..., m_1, m_0)$ for BCH encoding, where $m_{K_{bch}-1}$ is the first bit of the BBHEADER and m_0 is the last bit of the BBFRAME (or padding field if present). BCH encoding of information bits $M = (m_{K_{bch}-1}, m_{K_{bch}-2}, ..., m_1, m_0)$ onto a codeword is achieved as follows:

- Multiply the message polynomial $m(x) = m_{K_{bch}-1}x^{k_{bch}-1} + m_{K_{bch}-2}x^{k_{bch}-2} + ... + m_1x + m_0$ by $x^{N_{bch}-K_{bch}}$.
- Divide $x^{N_{bch}-K_{bch}}m(x)$ by $g(x)$, the generator polynomial. Let $d(x) = d_{N_{bch}-K_{bch}-1}x^{N_{bch}-K_{bch}-1} + ... + d_1x + d_0$ be the remainder.

- Construct the output codeword I, which forms the information word I for the LDPC coding, as follows:
  $I = (i_0, i_1, ..., i_{N_{bch}-1}) =$
  $(m_{K_{bch}-1}, m_{K_{bch}-2}, ..., m_1, m_0, d_{N_{bch}-K_{bch}-1}, d_{N_{bch}-K_{bch}-2}, ..., d_1, d_0)$.

Note: The equivalent codeword polynomial is $c(x) = x^{N_{bch}-K_{bch}} m(x) + d(x)$.

5.2 Inner encoding (LDPC)

The LDPC encoder treats the output of the outer encoding, $I = (i_0, i_1, ..., i_{K_{ldpc}-1})$, as an information block of size $K_{ldpc} = N_{BCH}$, and systematically encodes it onto a codeword $\wedge$ of size N_{ldpc}, where:

$$\wedge = (\lambda_0, \lambda_1, \lambda_2, ..., \lambda_{N_{ldpc}-1}) = (i_0, i_1, ..., i_{K_{ldpc}-1}, p_0, p_1, ...p_{N_{ldpc}-K_{ldpc}-1}).$$

The LDPC code parameters (N_{ldpc}, K_{ldpc}) are given in tables 5.3 and 5.4.

5.2.1 Inner coding for normal FECFRAME

The task of the encoder is to determine $N_{ldpc}-K_{ldpc}$ parity bits $(p_0, p_1, ..., p_{n_{ldpc}-k_{ldpc}-1})$ for every block of k_{ldpc} information bits, $(i_0, i_1, ..., i_{K_{ldpc}-1})$. The procedure is as follows:

- Initialize $p_0 = p_1 = p_2 = ... = p_{N_{ldpc}-K_{ldpc}-1} = 0$

- Accumulate the first information bit, i_0, at parity bit addresses specified in the first row of the given tables. For example, for rate 2/3 (see table 5.7), (all additions are in Galois Field):

317 2255 2324 2723 3538 3576 6194 6700 9101 10057 12739 17407 21039	10574 11268 17932
1958 2007 3294 4394 12762 14505 14593 14692 16522 17737 19245 21272 21379	15442 17266 20482
127 860 5001 5633 8644 9282 12690 14644 17553 19511 19681 20954 21002	390 3371 8781
2514 2822 5781 6297 8063 9469 9551 11407 11837 12985 15710 20236 20393	10512 12216 17180
1565 3106 4659 4926 6495 6872 7343 8720 15785 16434 16727 19884 21325	4309 14068 15783
706 3220 8568 10896 12486 13663 16398 16599 19475 19781 20625 20961 21335	3971 11673 20009
4257 10449 12406 14561 16049 16522 17214 18029 18033 18802 19062 19526 20748	9259 14270 17199
412 433 558 2614 2978 4157 6584 9320 11683 11819 13024 14486 16860	2947 5852 20101
777 5906 7403 8550 8717 8770 11436 12846 13629 14755 15688 16392 16419	3965 9722 15363
4093 5045 6037 7248 8633 9771 10260 10809 11326 12072 17516 19344 19938	1429 5689 16771
2120 2648 3155 3852 6888 12258 14821 15359 16378 16437 17791 20614 21025	6101 6849 12781
1085 2434 5816 7151 8050 9422 10884 12728 15353 17733 18140 18729 20920	3676 9347 18761
856 1690 12787	350 11659 18342
6532 7357 9151	5961 14803 16123
4210 16615 18152	2113 9163 13443
11494 14036 17470	2155 9808 12885
2474 10291 10323	2861 7988 11031
1778 6973 10739	7309 9220 20745
4347 9570 18748	6834 8742 11977
2189 11942 20666	2133 12908 14704
3868 7526 17706	10170 13809 18153
8780 14796 18268	13464 14787 14975
160 16232 17399	799 1107 3789
1285 2003 18922	3571 8176 10165
4658 17331 20361	5433 13446 15481
2765 4862 5875	3351 6767 12840
4565 5521 8759	8950 8974 11650
3484 7305 15829	1430 4250 21332
5024 17730 17879	6283 10628 15050
7031 12346 15024	8632 14404 16916
179 6365 11352	6509 10702 16278
2490 3143 5098	15900 16395 17995
2643 3101 21259	8031 18420 19733
4315 4724 13130	3747 4634 17087
594 17365 18322	4453 6297 16262
5983 8597 9627	2792 3513 17031
10837 15102 20876	14846 20893 21563
10448 20418 21478	17220 20436 21337
3848 12029 15228	275 4107 10497
708 5652 13146	3536 7520 10027
5998 7534 16117	14089 14943 19455
2098 13201 18317	1965 3931 21104
9186 14548 17776	2439 11565 17932
5246 10398 18597	154 15279 21414
3083 4944 21021	10017 11269 16546
13726 18495 19921	7169 10161 16928
6736 10811 17545	10284 16791 20655
10084 12411 14432	36 3175 8475
1064 13555 17033	2605 16269 19290
679 9878 13547	8947 9178 15420
3422 9910 20194	5687 9156 12408
3640 3701 10046	8096 9738 14711
5862 10134 11498	4935 8093 19266
5923 9580 15060	2667 10062 15972
1073 3012 16427	6389 11318 14417
5527 20113 20883	8800 18137 18434
7058 12924 15151	5824 5927 15314
9764 12230 17375	6056 13168 15179
772 7711 12723	3284 13138 18919
555 13816 15376	13115 17259 17332

Figure 5.7: Rate 2/3 ($N_{ldpc} = 64\,800$).

$$p_{317} = p_{317} \oplus i_0 \qquad p_{6700} = p_{6700} \oplus i_0$$
$$p_{2255} = p_{2255} \oplus i_0 \qquad p_{9101} = p_{9101} \oplus i_0$$
$$p_{2324} = p_{2324} \oplus i_0 \qquad p_{10057} = p_{10057} \oplus i_0$$
$$p_{2723} = p_{2723} \oplus i_0 \qquad p_{12739} = p_{12739} \oplus i_0$$
$$p_{3538} = p_{3538} \oplus i_0 \qquad p_{17407} = p_{17407} \oplus i_0$$
$$p_{3576} = p_{3576} \oplus i_0 \qquad p_{21039} = p_{21039} \oplus i_0$$
$$p_{6194} = p_{6194} \oplus i_0$$

- For the next 359 information bits, $i_m, m = 1, 2, ..., 359$ accumulate i_m parity bit addresses $\{x + (m \bmod 360) \times Q_{ldpc}\} \bmod (N_{ldpc} - K_{ldpc})$ where x denotes the address of the parity bit accumulator corresponding to the first bit i_0, and Q_{ldpc} is a code rate dependent constant specified in table 5.8. Continuing with the example, $Q_{ldpc} = 60$ for rate 2/3. So for example for information bit i_1, the following operations are performed:

Code Rate	Q_{ldpc}
1/2	90
3/5	72
2/3	60
3/4	45
4/5	36
5/6	30

Figure 5.8: Q_{ldpc} values for normal frames.

$$p_{377} = p_{377} \oplus i_1 \qquad p_{6760} = p_{6760} \oplus i_1$$
$$p_{2315} = p_{2315} \oplus i_1 \qquad p_{9161} = p_{9161} \oplus i_1$$
$$p_{2384} = p_{2384} \oplus i_1 \qquad p_{10117} = p_{10117} \oplus i_1$$
$$p_{2783} = p_{2783} \oplus i_1 \qquad p_{12799} = p_{12799} \oplus i_1$$
$$p_{3598} = p_{3598} \oplus i_1 \qquad p_{17467} = p_{17467} \oplus i_1$$
$$p_{3636} = p_{3636} \oplus i_1 \qquad p_{21099} = p_{21099} \oplus i_1$$
$$p_{6254} = p_{6254} \oplus i_1$$

- For the 361st information bit i_{360} , the addresses of the parity bit accumulators are given in the second row of the given tables. In a similar manner the addresses of the parity bit accumulators for the following 359 information bits $i_m, m = 361, 362, ..., 719$ are obtained using the formula $\{x + (m \bmod 360) \times Q_{ldpc}\} \bmod (N_{ldpc} - K_{ldpc})$ where x denotes the address of the parity bit accumulator corresponding to the information bit i_{360} , i.e. the entries in the second row of the given tables.

- In a similar manner, for every group of 360 new information bits, a new row from the given tables are used to find the addresses of the parity bit accumulators.

After all of the information bits are exhausted, the final parity bits are obtained as follows:

- Sequentially perform the following operations starting with $i = 1$.

$$p_i = p_i \oplus p_{i-1},\ i = 1, 2, ..., N_{ldpc} - K_{ldpc} - 1$$

- Final content of p_i, $i = 0, 1, .., N_{ldpc} - K_{ldpc} - 1$ is equal to the parity bit p_i.

5.2.2 Inner coding for short FECFRAME

K_{ldpc} BCH encoded bits shall be systematically encoded to generate N_{ldpc} bits as described before, replacing table 5.8 with table 5.9, and the table 5.7 with the corresponding given table.

Code Rate	Q_{ldpc}
1/4	36
1/2	25
3/5	18
2/3	15
3/4	12
4/5	10
5/6	8

Figure 5.9: Q_{ldpc} values for short frames.

5.3 Example with DVB-S2

In this section, simulation results are presented that illustrate the performance of the DVB-S2 LDPC code. Figure 5.10 shows the frame error rate (FER) performance of the short frame size and figure 5.12 shows the FER performance of normal frame size. In each case, up to 100 iterations of the log-domain sum-product algorithm are executed. Table5.11 shows the Eb/No required to achieve a FER of 10^{-3} for each rate and frame size. Because of the large size of the normal frame size code and the steepness of the corresponding FER curve, results could not be simulated all the way down to a FER 10^{-3} for every code rate. Thus, extrapolated results are given for rates r = 1/3, 1/2, 2/3 and 5/6. Note that the results presented here are only for the LDPC code. The outer BCH code used by DVB-S2 helps to clean up additional errors at the output of the LDPC decoder and will improve the overall performance.

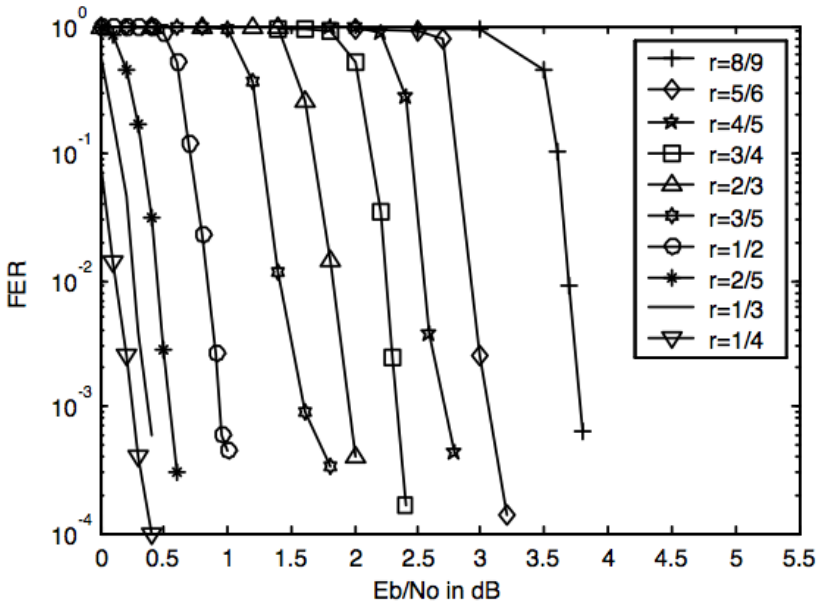


Figure 5.10: Frame error rate performance of the n = 16,200 bit (short frame) LDPC code used in DVB-S2. The decoder uses 100 iterations of the log-domain sum-product algorithm.

rate	short	normal
9/10	N/A	3.78 dB
8/9	3.78 dB	3.68 dB
5/6	3.06 dB	3.03* dB
4/5	2.72 dB	2.68 dB
3/4	2.33 dB	2.18 dB
2/3	1.95 dB	1.86* dB
3/5	1.59 dB	1.36 dB
1/2	0.93 dB	0.85* dB
2/5	0.55 dB	0.54 dB
1/3	0.37 dB	0.22* dB
1/4	0.25 dB	0.13 dB

Figure 5.11: The Eb/No required to achieve FER = 10^{-3} for the LDPC codes used in DVB-S2. Values marked with an asterisk (*) are extrapolated from figure 5.12.

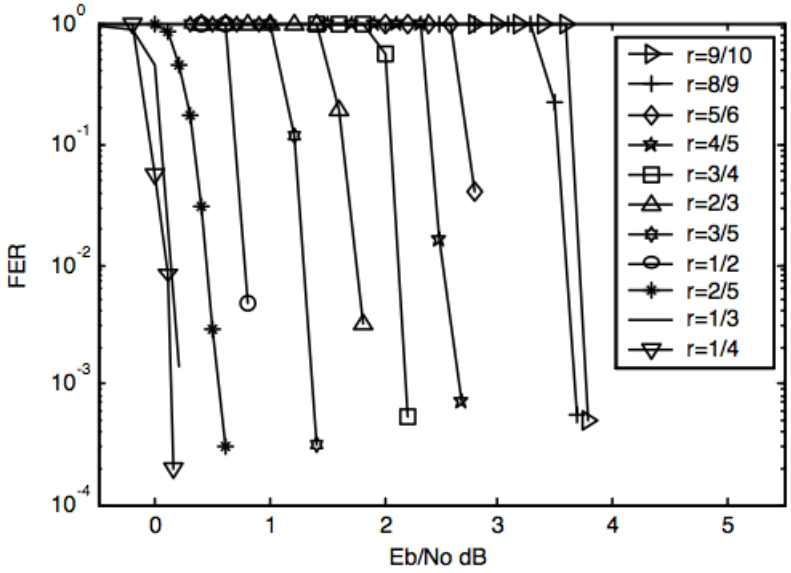


Figure 5.12: Frame error rate performance of the n = 64, 800 bit (normal frame) LDPC code used in DVB-S2. The decoder uses 100 iterations of the log-domain sum-product algorithm.

The DVB-S2 standard, which uses LDPC codes, represents a significant improvement in the satellite downlink. However, for these technological improvements to be a complete success several hurdles remain. Turbo and LDPC codes are still more

complex than their convolutional and Reed Solomon brethren, and therefore significant advances in implementation must still come to fruition. In addition, iteratively decodable codes are more sensitive to channel estimation and synchronization errors and therefore these issues must be dealt with carefully.

Chapter 6

Conclusions and future lines

Once finalized the thesis, we can sum up some of the main ideas exposed and extract conclusions.

About the introduction to the world of error-correcting codes, we can say:

- When encoding information, the minimal Hamming distance (d_{min}) among the available codewords is recommended to be as large as possible in order to detect and correct the maximum number of errors in the transmission.
  - Detectable errors $\leq d_{min}$
  - Correctible errors (t): $2t + 1 \leq d_{min}$
- The cost of decreasing the level of error of a message, is the increase of the redundancy bits (lower efficiency), latency due to more complex algorithms, and the economic factor.
- The fact of implementing an interleaver in the outcome of the encoding helps to fight against burst errors.
- Cyclic redundancy checks (CRCs) is the type of coding that detects a larger number of errors due to its use of polynomials and modular arithmetic.
- Block codes are less complex than convolutional codes, since they work on fixed-size packets.
- Shannon theorem affirms that it is possible to transmit the information almost without error in any way under a limiting rate, C, but it does not say how. Actually, this maximum capacity is not reachable, since the Shannon formula assumes some conditions which in practice do not exist. It does not take into account the impulsive noise, neither the attenuation or distortion.

About the overview of the new DVB standards:

- They all use a concatenation of BCH outer codes and LDPC inner codes in their FEC encoding block, although there exists a variation among the different rates used by them, since the conditions of the channels are very different (terrestrial, cable and satellite).

About Bose and Chauduri Hocquenghem (BCH):

- It is a multilevel cyclic variable-length digital error-correcting code used to multiple random error patterns.
- Its principal advantage is the ease with which it can be decoded, by the algebraic method known as *syndrome decoding*.
- It allows very simple electronic hardware, and it is also high flexible, allowing control over block length and acceptable error thresholds.
- It works with Galois fields, and the optimum generator polynomial must be primitive, minimal and taken from a combination of several polynomials corresponding to several powers of a primitive element in $GF(2^m)$. For its selection there exist some tables implemented by Lin and Costello.
- Locating errors implies an algorithm by Berlekamp that builds the error locator polynomial iteratively. That is why it is not too attractive when considered for high-speed applications.
- In order to ensure a high performance of the whole encoding, it is recommended to concatenate BCH with LDPC.

About Low-density parity-check (LDPC):

- LDPC are linear codes obtained from sparse bipartite graphs. This graph has n left nodes (message) and r right nodes (check), and its sparsity allows for the algorithmic efficiency.
- The graph can also be represented as a parity check matrix, which creats the available codewords.
- The decoder finds the correct codeword by a complex method called *belief propagation*, by Gallager. This iterative algorithm treats the messages as random variables and is based on some probability theorems such as Bayes' rule.

  - One of the main advantages is its running time (due to the sparsity), and the fact that the number of operations is linear in the number of message nodes.
  - It is independent of the channel used.
  - On the contrary, it is in general less powerful than maximum likelihood decoding.

- Comparing to Turbo Codes, we can say:
  - LDPC has significantly lower complexity. TC has a fixed number of iterations in the decoder, but in contrast, LDPC decoder stops when a legal codeword is found, reducing the amount of work to be done. However, this also implies the need of a buffer system in order to make the bit rate constant.
  - LDPC decoders may be implemented in parallel, which is good when considering long blocks.
  - TC offers a better complexity-performance for low block sizes (due to the iterative decoding in LDPC), but for a higher block length LDPC becomes more suitable.
  - The higher the code rate the less energy/cycles are required by LDPC. TC outperforms for lower block length but its higher energy consumption is the price to be paid.

BCH and LDPC codes, represent a significant improvement in the channel downlink. However, for these technological improvements to be a complete success several hurdles remain. Turbo and LDPC codes are still more complex than their convolutional and Reed Solomon brethren, and therefore significant advances in implementation must still come to fruition. In addition, iteratively decodable codes are more sensitive to channel estimation and synchronization errors and therefore these issues must be dealt with carefully.

Bibliography

Introduction

[1] Digital Television Advisor. *Proceso de Video - Parte 2*. Códigos de control de error

[2] Wikipedia. Códigos detectores y correctores de error

[3] Mundivia: Introducción a los códigos correctores de errores: `http://personales.mundivia.es/jtoledo`

[4] Divulgaciones.net: Teoría de códigos

[5] C. E. SHANNON. *The Bell System Technical Journal*. A Mathematical Theory of Communication

[6] WorldLingo. Teorema de la codificación del Ruidoso-canal

[7] Spread Spectrum Scene. Eb/No Explained

[8] Bell System Technical Journal 29. *Error detecting and error correcting codes*. Hamming, R. W.

[9] Thomson Delmar Learning. *Introduction to Telecommunications (2nd ed.)* ISBN 1401856489

Overview

[10] ETSI. *Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2)*. ETSI EN 302 755 V1.1.1

[11] ETSI. *Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital transmission system for cable systems (DVB-C2)*. ETSI EN 302 769

[12] ETSI. *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2).* ETSI EN 302 307 V1.2.1

Bose and Chaudhuri Hocquenghem (BCH)

[13] Claude E. Shannon and Warren Weaver. *University of Illinois Press.* The Mathematical Theory of Communication

[14] Shu Lin and Daniel J. Costello, Jr. *Englewood Cliffs, New Jersey: Prentice Hall.* Error Control Coding, Chapter 1

[15] Golay, Marcel J.E. *Proceedings of the IRE, Vol. 37.* Notes on Digital Coding

[16] Oliver Pretzel. *Oxford University Press.* Error Correcting Codes and Finite Fields

[17] Vera Pless. *New York: John Wiley & Sons.* Introduction to the Theory of Error Correcting Codes

[18] James DeTar. *Investor's Business Daily.* Mathematician Claude Shannon

[19] Lin and Costello. *New York: McGraw-Hill.* Error Coding Cookbook

Low-Density Parity-Check (LDPC)

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

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

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

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

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

[25] J. Rosenthal and P. Vontobel. *Construction of LDPC codes using Ramanujan graphs and ideas from Margulis*. Control, and Computing, pp. 248–257, 2000.

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

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

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

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

Functionality of BCH-LDPC in the new DVB standards

[30] DigiTAG. *Key technical, business, & regulatory implications*. Understanding DVB-T2

[31] Digital Video Broadcasting (DVB). *Frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2)*. ETSI EN 302 755 V1.1.1 (2009-09)

Turbo Coding (TC)

[32] Consultative Committee on Space Data Systems (CCSDS). *Draft ccsds recommendation for telemetry channel coding*. Revision 4

[33] C. Berrou, A. Glavieux, and P. Thitimajshima. *Near shannon limit error-correcting coding and decoding: Turbo-codes, Proceedings ICC*. IEEE, May 1993, pp. 1064–1070

[34] University of South Australia: Turbo coding research group: `http://www.itr.unisa.edu.au/~steven/turbo`

[35] C. Berrou, A. Glavieux, and P. Thitimajshima. *Near Shannon limit error- correcting coding and decoding: Turbo codes.* Geneva, Switzerland, May 2003.

[36] Third Generation Partnership Project (3GPP). *Multiplexing and Channel Coding (FDD).* March 2005. TS 25.212 Version 6.4.0.

[37] M. C. Valenti and J. Sun. *Turbo codes.* In F. Dowla, editor, Handbook of RF and Wireless Technologies. Newnes, 2004.

Appendix A

GF(2^m) Field Generator Computer Program

```
/* GENFIELD.C
This program generates fields of 2^m elements using polynomial arithmetic.
Hank Wallace 09-Mar-01 Revision 09-Mar-01
/* #define src_file 1
#include \process.h" #include \string.h" #include \conio.h" #include \math.h"
#include \ctype.h" #include \dos.h" #include \stdio.h" #include \stdlib.h"
typedef unsigned char typedef signed char typedef unsigned int typedef
unsigned long uchar; schar; uint; ulong;
/* ======================================================= */
char *tobinary(int number, int digits, char *s) /* This function converts an
integer  to its binary representation and places it in in string s. */ {
int i;
number<<=16-digits; *s=0; for (i=0; i<digits; i++)
{
if (number & 0x8000) strcat(s,"1");
else strcat(s,"0");
number<<=1; return(s);
}
/* ======================================================= */
char *topoly(int number, int digits, char *s) /* This function converts an
integer  to its polynomial representation and places it in in string s. */ {
int i;
number<<=16-digits; *s=0; for (i=0; i<digits; i++)
{
if (number & 0x8000) sprintf(&s[strlen(s)],"a^%d \, digits-i-1);
number<<=1; return(s);
}
/* ======================================================= */
}
}
void main(int argument_count, char *argument[]) {
```

```
}
int i,// loop index order, // order of the generator polynomial x,// the
current field element n; // number of elements in the field
long poly, // polynomial entered by the user l; // scratch
char s[100]; // for string operations
// look for command line arguments if (argument_count != 2)
{
}
printf(\GF(2^m) Field Element Generator\n\n"); printf(\Usage:
GeneratorPoly(a^n...a^0)\n\n"); exit(0);
// read polynomial coefficients // polynomial is assumed to have a root alpha
not in GF(2) poly=atol(argument[1]);
// determine order of polynomial order=31; l=poly; while ((l & 0x80000000L) == 0)
{
}
order--; l<<=1;
// compute number of elements in the field n=1 << order;
// generate and print the field elements printf(\Field of %d elements
with generator polynomial %s\n",
n,topoly(poly,order+1,s));
// print the ever present zero and one elements printf(\0 %s\n",
tobinary(0,order,s)); printf(\1 %s\n",tobinary(1,order,s));
x=1; // initialize the current field element for (i=0; i<n-2; i++)
{
}
x<<=1; // multiply by the root, alpha
if (x & n) // arithmetic is modulo the polynomial {
// subtract (exclusive OR) the generator polynomial x^=poly;
}
printf(\a^%-2d %s \,i+1,tobinary(x,order,s));
printf(\%s\n",topoly(x,order,s));
```

Appendix B

Turbo Codes

B.1 Introduction

This part of the appendix is an introductory tutorial on turbo codes, a new technique of error- correction coding developed in the 1990s. The reader is expected to be familiar with the basic concepts of channel coding, although we briefly and informally review the most important terms.

The paper starts with a short overview of channel coding and the reader is re- minded the concept of convolutional encoding. Bottlenecks of the traditional approach are described and the motivation behind turbo codes is explained. After examining the turbo codes design more in detail, reasons behind their efficiency are brought out. Finally, a real-world example of the turbo code used in the third generation Universal Mobile Telecommunications System (UMTS) is presented.

The paper is mainly based on two excellent tutorials by Valenti and Sun, and Barbulescu and Pietrobon. The scope of this paper does not cover implementation-specific issues such as decoder architecture, modulation techniques and the like.

B.2 Channel coding

The task of channel coding is to encode the information sent over a communication channel in such a way that in the presence of channel noise, errors can be detected and/or corrected. We distinguish between two coding methods:

- *Backward error correction (BEC):* requires only error detection: if an error is detected, the sender is requested to retransmit the message. While this

method is simple and sets lower requirements on the code's error-correcting properties, it on the other hand requires duplex communication and causes undesirable delays in transmission.

- *Forward error correction (FEC):* requires that the decoder should also be capable of correcting a certain number of errors, i.e. it should be capable of locating the positions where the errors occurred. Since FEC codes require only simplex communication, they are especially attractive in wireless communication systems, helping to improve the energy efficiency of the system. In the rest of this paper we deal with binary FEC codes only.

Next, we briefly recall the concept of conventional convolutional codes. Convolutional codes differ from block codes in the sense that they do not break the message stream into fixed-size blocks. Instead, redundancy is added continuously to the whole stream. The encoder keeps M previous input bits in memory. Each output bit of the encoder then depends on the current input bit as well as the M stored bits. Figure B.1 depicts a sample convolutional encoder. The encoder produces two output bits per every input bit, defined by the equations.

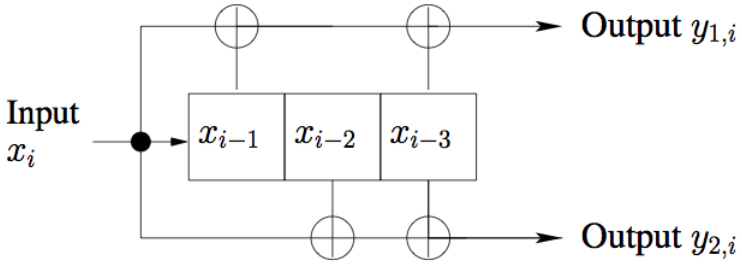


Figure B.1: A convolutional encoder.

$$y_{1,i} = x_i + x_{i-1} + x_{i-3}$$

$$y_{2,i} = x_i + x_{i-2} + x_{i-3}$$

For this encoder, M = 3, since the ith bits of output depend on input bit i, as well as three previous bits $i-1, i-2, i-3$. The encoder is nonsystematic, since the input bits do not appear explicitly in its output.

An important parameter of a channel code is the code rate. If the input size (or message size) of the encoder is k bits and the output size (the code word size) is n bits, then the ratio k/n is called the code rate r. Since our sample convolutional

en- n coder produces two output bits for every input bit, its rate is 1/2 . Code rate expresses the amount of redundancy in the code—the lower the rate, the more redundant the code.

Finally, the Hamming weight or simply the weight of a code word is the number of non-zero symbols in the code word. In the case of binary codes, dealt with in this paper, the weight of a code word is the number of ones in the word.

B.3 A need for better codes

Designing a channel code is always a tradeoff between energy efficiency and band- width efficiency. Codes with lower rate (i.e. bigger redundancy) can usually correct more errors. If more errors can be corrected, the communication system can operate with a lower transmit power, transmit over longer distances, tolerate more interference, use smaller antennas and transmit at a higher data rate. These properties make the code energy efficient. On the other hand, low-rate codes have a large overhead and are hence more heavy on bandwidth consumption. Also, decoding complexity grows exponentially with code length, and long (low-rate) codes set high computational requirements to conventional decoders. According to Viterbi, this is the central problem of channel coding: encoding is easy but decoding is hard.

For every combination of bandwidth (W), channel type, signal power (S) and received noise power (N), there is a theoretical upper limit on the data transmission rate R, for which error-free data transmission is possible. This limit is called channel capacity or also Shannon capacity (after Claude Shannon, who introduced the notion in 1948). For additive white Gaussian noise channels, the formula is:

$$R < W \cdot log_2(1 + S/N) \text{ [bits/second]}$$

In practical settings, there is of course no such thing as an ideal error-free channel. Instead, error-free data transmission is interpreted in a way that the bit error probability can be brought to an arbitrarily small constant. The bit error probability, or bit error rate (BER) used in benchmarking is often chosen to be 10^{-5} or 10^{-6}.

Now, if the transmission rate, the bandwidth and the noise power are fixed, we get a lower bound on the amount of energy that must be expended to convey one bit of information. Hence, Shannon capacity sets a limit to the energy efficiency of a code. Although Shannon developed his theory already in the 1940s, several decades later the code designs were unable to come close to the theoretical bound. Even in the beginning of the 1990s, the gap between this theoretical bound and

practical implementations was still at best about 3dB. This means that practical codes required about twice as much energy as the theoretical predicted minimum.

Keeping these design methods in mind, we are now ready to introduce the concept of turbo codes.

B.4 Encoding with interleaving

The first turbo code, based on convolutional encoding, was introduced in 1993 by Berrou. Since then, several schemes have been proposed and the term "turbo codes" has been generalized to cover block codes as well as convolutional codes. Simply put, a turbo code is formed from the parallel concatenation of two codes separated by an interleaver.

The generic design of a turbo code is depicted in figure B.2. Although the general concept allows for free choice of the encoders and the interleaver, most designs follow the ideas presented in:

- The two encoders used are normally identical.
- The code is in a systematic form, i.e. the input bits also occur in the output (see Figure B.2).
- The interleaver reads the bits in a pseudo-random order.

The choice of the interleaver is a crucial part in the turbo code design . The task of the interleaver is to "scramble" bits in a (pseudo-)random, albeit predetermined fashion. This serves two purposes. Firstly, if the input to the second encoder is interleaved, its output is usually quite different from the output of the first encoder. This means that even if one of the output code words has low weight, the other usually does not, and there is a smaller chance of producing an output with very low weight. Higher weight, as we saw above, is beneficial for the performance of the decoder. Secondly, since the code is a parallel concatenation of two codes, the divide-and-conquer strategy can be employed for decoding. If the input to the second decoder is scrambled, also its output will be different, or "uncorrelated" from the output of the first encoder. This means that the corresponding two decoders will gain more from information exchange.

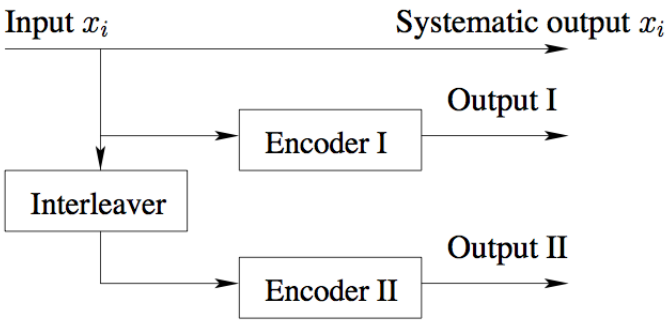


Figure B.2: The generic turbo encoder.

We now briefly review some interleaver design ideas, stressing that the list is by no means complete. The first three designs are illustrated in Figure B.3 with a sample input size of 15 bits.

1. A "row-column" interleaver: data is written row-wise and read column- wise. While very simple, it also provides little randomness.

2. A "helical" interleaver: data is written row-wise and read diagonally.

3. An "odd-even" interleaver: first, the bits are left uninterleaved and en- coded, but only the odd-positioned coded bits are stored. Then, the bits are scrambled and encoded, but now only the even-positioned coded bits are stored. Odd-even encoders can be used, when the second encoder produces one output bit per one input bit.

4. A pseudo-random interleaver defined by a pseudo-random number generator or a look-up table.

There is no such thing as a universally best interleaver. For short block sizes, the odd-even interleaver has been found to outperform the pseudo-random interleaver, and vice versa. The choice of the interleaver has a key part in the success of the code and the best choice is dependent on the code design.

Input				
x_1	x_2	x_3	x_4	x_5
x_6	x_7	x_8	x_9	x_{10}
x_{11}	x_{12}	x_{13}	x_{14}	x_{15}

Row-column interleaver output														
x_1	x_6	x_{11}	x_2	x_7	x_{12}	x_3	x_8	x_{13}	x_4	x_9	x_{14}	x_5	x_{10}	x_{15}

Helical interleaver output														
x_{11}	x_7	x_3	x_{14}	x_{10}	x_1	x_{12}	x_8	x_4	x_{15}	x_6	x_2	x_{13}	x_9	x_5

Odd-even interleaver output														
Encoder output without interleaving														
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
y_1	-	y_3	-	y_5	-	y_7	-	y_9	-	y_{11}	-	y_{13}	-	y_{15}
Encoder output with row-column interleaving														
x_1	x_6	x_{11}	x_2	x_7	x_{12}	x_3	x_8	x_{13}	x_4	x_9	x_{14}	x_5	x_{10}	x_{15}
-	z_6	-	z_2	-	z_{12}	-	z_8	-	z_4	-	z_{14}	-	z_{10}	-
Final output of the encoder														
y_1	z_6	y_3	z_2	y	z_{12}	y_7	z_8	y_9	z_4	y_{11}	z_{14}	y_{13}	z_{10}	y_{15}

Figure B.3: Interleaver designs.

B.5 Some notes on decoding

In the traditional decoding approach, the demodulator makes a "hard" decision of the received symbol, and passes to the error control decoder a discrete value, either a 0 or a 1. The disadvantage of this approach is that while the value of some bits is determined with greater certainty than that of others, the decoder cannot make use of this information.

A soft-in-soft-out (SISO) decoder receives as input a "soft" (i.e. real) value of the signal. The decoder then outputs for each data bit an estimate expressing the probability that the transmitted data bit was equal to one. In the case of turbo codes, there are two decoders for outputs from both encoders. Both decoders provide estimates of the same set of data bits, albeit in a different order. If all

intermediate values in the decoding process are soft values, the decoders can gain greatly from exchanging information, after appropriate reordering of values. Information exchange can be iterated a number of times to enhance performance. At each round, decoders re-evaluate their estimates, using information from the other decoder, and only in the final stage will hard decisions be made, i.e. each bit is assigned the value 1 or 0. Such decoders, although more difficult to implement, are essential in the design of turbo codes.

B.6 Performance

We have seen that the conventional codes left a 3dB gap between theory and prac- tice. After bringing out the arguments for the efficiency of turbo codes, one clearly wants to ask: how efficient are they?

Already the first rate 1/3 code proposed in 1993 made a huge improvement: the gap between Shannon's limit and implementation practice was only 0.7dB, giving a less than 1.2-fold overhead. (In the authors' measurements, the allowed bit error rate BER was 10^{-5}). In practice, the code rate usually varies between 1/2 and 1/6. Let the allowed bit error rate be 10^{-6}. For code rate 1/2, the relative increase in energy consumption is then 4.80dB for convolutional codes, and 0.98dB for turbo codes. For code rate 1/6, the respective numbers are 4.28dB and -0.12dB[1]. It can also be noticed, that turbo codes gain significantly more from lowering the code rate than conventional convolutional codes.

B.7 The UMTS Turbo Code

The UMTS turbo encoder closely follows the design ideas presented in 1993. The starting building block of the encoder is the simple convolutional encoder depicted in Figure B.1. This encoder is used twice, once without interleaving and once with the use of an interleaver, exactly as described above.

In order to obtain a systematic code, desirable for better decoding, the following modifications are made to the design. Firstly, a systematic output is added to the encoder. Secondly, the second output from each of the two encoders is fed back to the corresponding encoder's input. The resulting turbo encoder, depicted in Figure B.4, is a rate 1/3 encoder, since for each input bit it produces one systematic output bit and two parity bits.

[1]Although the relative value is negative, it does not actually violate the Shannon's limit. The negative value is due to the fact that we allow for a small error, whereas Shannon's capacity applies for perfect error-free transmission.

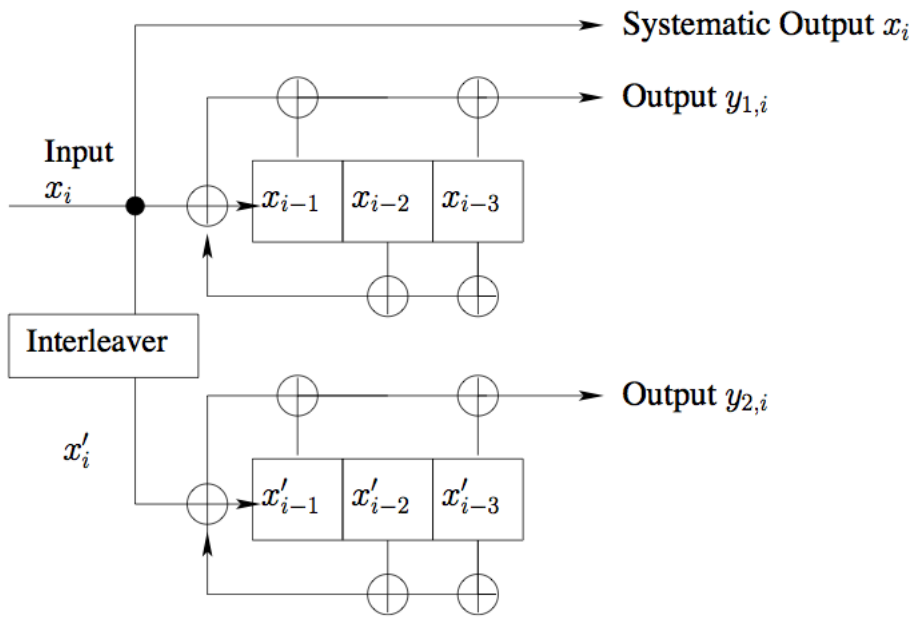


Figure B.4: The UMTS turbo encoder.

B.8 Conclusions

Turbo codes are a recent development in the field of forward-error-correction channel coding. The codes make use of three simple ideas: parallel concatena- tion of codes to allow simpler decoding; interleaving to provide better weight distribution; and soft decoding to enhance decoder decisions and maximize the gain from decoder interaction.

While earlier, conventional codes performed—in terms of energy efficiency or, equivalently, channel capacity—at least twice as bad as the theoretical bound suggested, turbo codes immediately achieved performance results in the near range of the theoretically best values, giving a less than 1.2-fold overhead. Since the first proposed design in 1993, research in the field of turbo codes has produced even better results. Nowadays, turbo codes are used in many commercial applications, including both third generation cellular systems UMTS and cdma2000.