CRC Aided List Decoding of Convolutional Codes

MINGZHE GUO
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

CRC Aided List Decoding of Convolutional Codes

Mingzhe Guo
mi2844gu-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Wei Zhou

Co-supervisor: Michael Lentmaier

Examiner: Thomas Johansson

September 6, 2019

© 2019
Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Designing for the best combination of channel codes and the decoding algorithm has always been a popular topic in channel coding. As a decoding algorithm for convolutional codes, Viterbi algorithm provides the most reliable performance with high complexity. List Viterbi algorithm(LVA) is an alternative solution to provide a good tradeoff between performance and complexity, and is widely used for high memory convolutional codes.

In this thesis, a coding scheme for combining LVA and CRC will be introduced, and the implementation of CRC aided list decoding will be explained. This thesis compares the complexity and performance of this proposed decoding algorithm with Viterbi algorithm and list decoding. Simulations show that the proposed CRC aided list decoding can beat a traditional list decoding, and that there are several parameters that will influence the performance. Furthermore, the thesis investigates the performance of the decoder with different implementations under this coding scheme, and analyses the error probability by analysing the simulations.

Keywords: Channel coding, convolutional code, list decoding, CRC, 5G.

Acknowledgments

First, I want to thank my supervisor Wei Zhou. He helps me with his knowledge and experience whenever the project meets with an obstacle. Without his help, this thesis is not possible to complete.

I am also grateful for my co-supervisor Michael Lentmaier. His advice also helps me a lot in the project. A lot of ideas in the thesis are inspired by his opinions.

I also want to thanks professor John B Anderson, who gave me some advice on list decoding algorithm. The sorting algorithm used in the program is based on his suggestion.

Finally, I want to thank my friends and my family for supporting me behind.

Popular Science Summary

Communication involves transferring data from one place to another. In the transmitter, the transmitted message is converted in a suitable form before going through the channel. At the receiver, the signal is estimated and reconstructed to match the transmitted signal. Digital communication is a mode of communication where the date source is converted in discrete format before modulated as analog waves and transferring to the receiver. One advantage of digital information is that it tends to be far more resistant to errors due to interference, noise and channel fading than information symbolized in an analog medium. Digital communication systems are becoming the most common communication solutions all around us.

Channel coding is a technique of detecting and correcting bit errors in digital communication systems. In transmitter, the information sequence is encoded, where some redundant information are added to form a codeword. In receiver, by analysing the received codeword, the decoder can detect and correct the bit error. There are different coding methods developed for different situations.

Polar codes received lots of research attention since introduced in a publication in 2008 by Erdal Arikan[1]. They can achieve Shannon’s capacity at infinite block length with low complexity decoding due to their simple structure. So polar codes with long block length have both advantages of having good performance and low complexity. However, polar codes do not provide good performance at short block length. A lot of researches focus on finding methods to improve polar codes. List decoding of polar codes with cyclic redundancy check(CRC) is one of them.

There are several researches revealing that the combination of listing decoding with CRC is able to improve the performance of short polar codes. List decoding builds a set of possible codewords, and the CRC is used at the last step to eliminate those paths that don’t satisfy CRC check from the list. With this combination, short polar codes are greatly improved from performing relatively bad to having the start-of-the-art performance among all other codes such as LDPC and turbo codes. It is interesting to see, in this thesis, that if the combination of list decoding and CRC can also be used to improve the performance of traditional convolutional codes.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Communication plays an important role in our daily life nowadays. During the transmission, the quality of the channel will influence the information accuracy. If the signal to noise ratio(SNR) is low, errors will occur frequently. To reduce the number of errors due to the channel, i.e., improve the transmission accuracy, channel coding is widely used in communication systems as a mechanism to enhance the reliability of the transmission.

Polar codes have been receiving lots of attention in the last decade and now become standard code in control channel of 5G systems. However, the asymptotic result of achieving the channel capacity with successive cancellation decoder for polar codes does not warrant a good performance in the short block length, especially in the region of low error rate. In fact, 5G system uses a successive cancellation list(SCL) decoding, and combine it with CRC code. The SCL decoder generates a list of most possible codewords during decoding, and CRC helps to check and eliminate the incorrect codewords in the final decoding step. With this combination, polar codes in short block can have a significant improvement.

Convolutional codes are the used in the thesis. The main feature of convolutional codes is that each output coded block does not only depend on current input block, but also on previous input block. It is a type of well-researched codes, and always used as a benchmark when decoded by Viterbi algorithm, which gives maximum likelihood performance.

List decoding is a sub-optimal decoding method compared to Viterbi algorithm. Instead of calculating all candidates and storing all of them, list decoding only focuses on a subset of candidates, so it can achieve low complexity even for decoding a high memory code. To improve a sub-optimal list decoding is always a popular topic in channel coding field. Inspired by the CRC aided SCL, it is also interesting to see that what may happen when combining CRC with list decoding. In CRC aided SCL decoding, the CRC check is the final step in the decoding. What would happen if we divide CRC bits into several parts, and spread them in between the codewords? Will it improve the performance? This thesis will aim at solving these questions, showing the realization of CRC aided list decoding, and introducing some parameters and different implementations that will affect the performance.

1.1 Goal of the Project

In the thesis, the performance of convolutional codes with short block length will be investigated. First, there will be a description about list decoding and a comparison of list decoding and Viterbi decoding to understand some basic concept in coding theory, and then CRC will be introduced to the code. A CRC aided list decoder will be shown and simulated. There will also be an explanation of why CRC aided decoder could have a better performance and how to optimize it. Some comparisons will be made to find a better implementation of the decoder for improving the performance.

The goal of the project is to construct a CRC aided list decoder, understand why it improves code performance, and possibly find some combinations to improve it.

1.2 Related Work

The interest of this project grows from research on polar codes. Tal and Vardy first proposed the use of CRC aided list decoding on polar codes[2]. In their article, they demonstrate that polar codes of block length 2048-bit can have a better performance than LDPC codes of length 2304 and rate 0.5. There is also an improved combination showing that short block length polar codes with 128 bits and bit rate 0.5 can achieve a block error rate down to 10^{-6}, outperforming LDPC and turbo code with the same block length and code rate.

Johansson[3] introduced that by dividing CRC into several shorter CRCs spreading out over the polar code, the performance of CRC aided polar code can be improved. Inspired by the research, this thesis discuss about spreading CRC check bits encoded in between the sub-blocks of information bits before feeding them into the convolutional code encoder.

1.3 Contributions

Previous researches on polar codes have demonstrated that with CRC aided, short block length polar code can reach a better performance. The thesis further investigates that the performance of CRC aided list decoding of convolutional codes. In the thesis, firstly, we have a comparison of Viterbi algorithm versus list Viterbi algorithm. Simulation shows that list Viterbi algorithm can outperform Viterbi algorithm given the same complexity. Then, we design a CRC aided decoder, which uses CRC to eliminate paths that don't satisfy CRC check. Puncturing is used to reduce some redundancy to maintain the code rate. Simulation result shows that the CRC aided decoding can achieve a better performance than list decoding for convolutional codes. We also make a comparison of different schemes of CRC aided list decoding, and analyze how they affect the performance. We also describe the situation where the decoder fails to decode the codeword. In order to solve this problem, we introduce two different implementations, and make comparison between them. After that, there will be an error analysis showing how CRC helps list decoding to achieve better performance.

All the investigations in the thesis are performed with simulations from code written in Matlab.

1.4 Thesis Outline

The thesis outline is as follows.

Chapter 2 introduces the basic concepts of coding theory, including block codes, convolutional codes, state transition diagram and trellis, Viterbi decoding, as well as a brief explanation of CRC. Then, Chapter 3 gives a more detailed explanation of list decoding, introduces the relevant decoding algorithm, as well as compares list decoding with Viterbi decoding in both complexity and performance. Puncturing is used to maintain the code rate. After that, Chapter 4 adds CRCs to the list decoding. Depending on the positions of CRCs within a codeword, different schemes of CRC aided list decoding are introduced. Two implementations on proposed decoding scheme is discussed to reduce errors that may happen during the decoding process. Finally, Chapter 5 shows all the simulation results based on Chapter 4, compare different decoding schemes, parameters and error types. Conclusion and future work are discussed in Chapter 6.

Chapter 2
Background

Error control coding is a well-known technique that increases the reliability in digital communication systems, where extra bits are added to the data at the transmitter to detect and correct error at the receiver. Convolutional codes were introduced in 1950s by Peter Elias and have been widely used in many systems[4]. They are one kind of error control codes where parity bits at any given time are generated based on input bits both at that time and also at previous times.

The Viterbi algorithm, which is a decoding method for convolutional codes that allows soft input, can achieve the optimal maximum likelihood decoding performance. Therefore, it is the most powerful decoding algorithm when the system can afford the complexity, but it will have rather high complexity when the encoder constraint length (the number of bits that need to be stored for encoding) is big.

When using stronger codes, other decoding methods such as list decoding can be more efficient. List decoding of convolutional codes corrects errors by extending a reduced set of most promising subpaths. It is an efficient technique that can achieve high coding gain with relatively low complexity compared to Viterbi algorithm. In other words, the list decoder can potentially have a better performance than the Viterbi algorithm when decoding the same convolutional codes at around same complexity.

2.1 Channel Coding

Communication is a fundamental need in our modern lives. We communicate over a variety of channels daily. Bit error can occur whenever a message is transferred in a channel. However, if the message is encoded properly, the receiver can detect and correct the error caused by noisy channel. The process of encoding, detecting and correcting bit errors in digital communication systems is called channel coding. It is a process of constructing code in encoder and detecting bit errors in decoder in digital communication systems.

Channel coding is also called error-control coding, since it controls the occurrences of errors so that the receiver can recover the original information if parts of it are corrupted. The basic idea is to judiciously introduce redundancy so that the original information can be recovered even when parts of the data have been corrupted.

Channel coding is performed both at the transmitter and at the receiver. At the transmitter side, the encoder maps incoming data sequence into a channel input sequence. Extra bits are added with the raw data before modulation. At the receiver side, channel coding enables decoder to recover the channel output sequence into an output data sequence. In this process, overall effect of the channel noise should be minimized.

A coding gain is expected to be achieved with properly coding. With coding, the energy per information bit change to $E_b = \frac{1}{R}E_s > E_s$, where R is the code rate and always smaller than 1. The coding gain is defined to be the overall gain E_b/N_0 of a coded system compared to an uncoded system. A big coding gain represents an increased coding efficiency.

A block diagram of a typical channel coding model is shown in Figure 2.1.

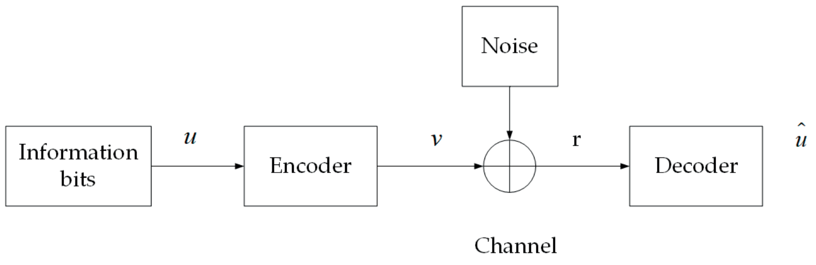

Figure 2.1: Block diagram of channel coding model

2.2 Block Codes

Block code is introduced by Richard Hamming in 1950. It is one type of error control codes in which a fixed number of bits, k, are taken into the encoder, and then the codeword consisting of a larger number n of bits is generated as output.

A length n block code $\mathcal{C}$ over a field $\mathbb{F}$ is a set of code words $\mathbf{v} \in \mathbb{F}^n$. A linear code consists of codewords such that

$$\mathbf{v}_1, \mathbf{v}_2 \in \mathcal{C} \Rightarrow \ \alpha\mathbf{v}_1 + \beta\mathbf{v}_2 \in \mathcal{C}, \ \alpha, \beta \in \mathbb{F}.$$

The encoding of a linear code can be described by a $k \times n$ generator matrix $\mathbf{G}$:

$$\mathbf{v} = \mathbf{uG}, \mathbf{G} \in \mathbb{F}^{k \cdot n}.$$

In a linear block code, any set of k linearly independent codewords in $\mathcal{C}$ can be used as rows of $\mathbf{G}$.

2.3 Channel Models

To present the effects of a communication channel mathematically and analyse transmit error properly, channel model is introduced to channel coding field. A channel model is an essential piece of the physical layer communication simulation.

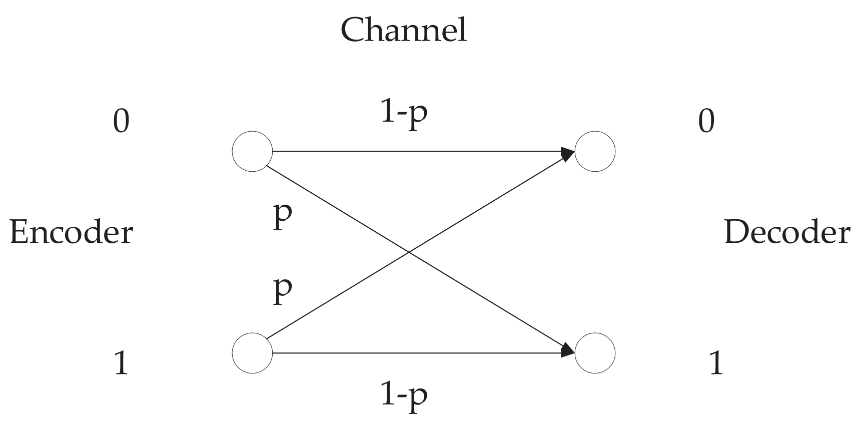

Figure 2.2: BSC channel model

Assume for a binary channel, the input bit is x and output bit is y, and the transmission probability is $p(y|x)$.

Some typical channel definitions are explained as follows:

Binary symmetric channel(BSC): Transmitter sends a bit and receiver receives a bit. In the channel, the bit is flipped when error occurs in a probability p_e. The channel is symmetric, so error probability $p(0|1) = p(1|0) = p_e$, and the probability of a correct transfer $p(0|0) = p(1|1) = 1 - p_e$. Figure 2.2 is a model of BSC.

Binary erasure channel(BEC): A binary channel where transmitter sends a bit and receiver receives a bit. The receiver either receives the bit or it receives a message that the bit is not received ("erased"). When the receiver gets a bit, it can be certain whether the bit is received or erased. If it is received, it is sure that the bit is received correctly. This simplified channel is often used for analyzing code performance.

Additive white Gaussian noise(AWGN): A widely used channel model to mimic the effect of many random processes that occur in nature. The word "white" refers to that it has uniform power across the frequency band for the information system. The Gaussian noise used in the channel has a normal distribution in the time domain with an average value of zero.

The capacity of AWGN channel can be represented by

$$C = W \log_2(1 + \frac{S}{N_0 W}) bits/sec.$$

This is called Shannon limit or Shannon capacity stated by Claude Shannon in 1948. C is channel capcity, W is the bandwidth, and N_0 is noise density. It shows the theoretical maximum information transfer rate of the channel, if given a particular noise level. In other words, it is possible to transmit information nearly

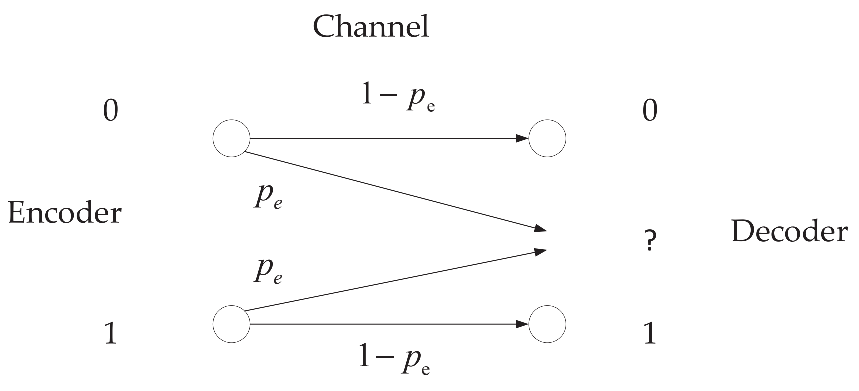

Figure 2.3: BEC channel model

without error at any rate below a limiting rate C. If a code is close to Shannon limit, then it can be indicated that it is a strong code.

All the channels used in the thesis are AWGN channels, which means noise is generated across all the frequency band.

2.4 Convolutional Codes

Convolutional codes have a different structure compared to block codes[5]. During each unit of time, the input of convolutional code encoder can be considered to be k-bit message blocks, and the corresponding output can also be considered as n-bit message blocks. The main difference between convolutional codes and block codes is that each coded n-bit output block depends not only the corresponding k-bit input message at the same time, but also on the m previous message blocks.

2.4.1 Introduction to Convolutional Codes

The fundamental idea of convolutional codes is to express each code block $\mathbf{v}_t$ as a function of both present and previous m input blocks, i.e.,

$$\mathbf{v}_t = f(\mathbf{u}_t, \mathbf{u}_{t-1}, ..., \mathbf{u}_{t-m}),$$

where m is called memory of the code, it defines the number of previous input blocks used to generate the current block.

2.4.2 Encoding

In the convolutional code encoder, each information sequence is encoded into a code sequence. All possible code sequences produced by the encoder are called codewords of an (n, k, m) convolutional code. The parameter m is called the memory of the code. The ratio $R = k/n$ is called the code rate. The redundancy bits in each code block is small. However, more redundancy bits are added by increasing the memory m of the code while k and n remain fixed. The encoding can be described by

$$\mathbf{v} = \mathbf{u}\mathbf{G},$$

where $\mathbf{u}$ is input sequence and $\mathbf{v}$ is output sequence. $\mathbf{G}$ is called generator matrix. Typically a generator matrix for a convolutional has the structure:

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & ... & \mathbf{G}_m & & \\ & \mathbf{G}_0 & \mathbf{G}_1 & ... & \mathbf{G}_m & \\ & & \ddots & & & \ddots \end{bmatrix}, \mathbf{G}_i \in \mathbb{F}_2^{k \cdot n}.$$

The output $v_t^{(j)}, j = 1, ..., n$ can be written as:

$$v_t^{(j)} = \sum_{i=1}^{k} \sum_{l=0}^{m} u_{t-l}^{(i)} g_{i,l}^{j}.$$

As described on the equation, the output $v_t^{(j)}$ is related to each input $u^{(i)}$ by a convolution with the corresponding generator $g_i^{(j)}$. The code is specified by a set of generator sequences of length $m + 1$,

$$\mathbf{g}^{(n)} = (g_0^{(n)}, g_1^{(n)}, g_2^{(n)}, ..., g_m^{(n)}).$$

The encoder is actually a discrete linear system.

2.4.3 State and Trellis Diagram

Since the encoder is a linear sequential circuit, a state diagram can be used to describe its behaviour. The encoder executes a state transition when the information bit is shifted into the encoder register. The current state is

$$\mathbf{s}_l = (c_{l-1}, c_{l-2}, ..., c_{l-m}),$$

and when the next bit c_l is shifted into the encoder, it moves to the next state

$$\mathbf{s}_{l+1} = (c_l, c_{l-1}, ..., c_{l-m+1}).$$

An example of state diagram of a (2, 1, 2) code is shown in Figure 2.4. There are $2^2 = 4$ states, and one bit input will result in two bits output. Every input sequence $\mathbf{u} = (u_0, u_1, ...)$ defines a path in the state trellis in the encoder.

By expanding the state diagram along the time, it results in an encoder trellis. Figure 2.5 is a trellis of a (2,1,2) code based on the disgram.

After the entire sequence is coded, the encoder must return to its starting state. This is called the termination of the trellis. It is done by appending m zeros to the message sequence.

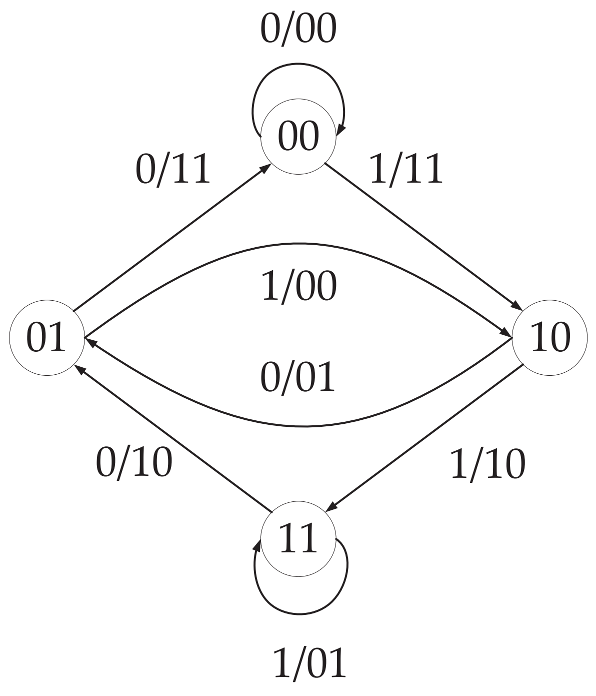

Figure 2.4: State diagram of a (2, 1, 2) code

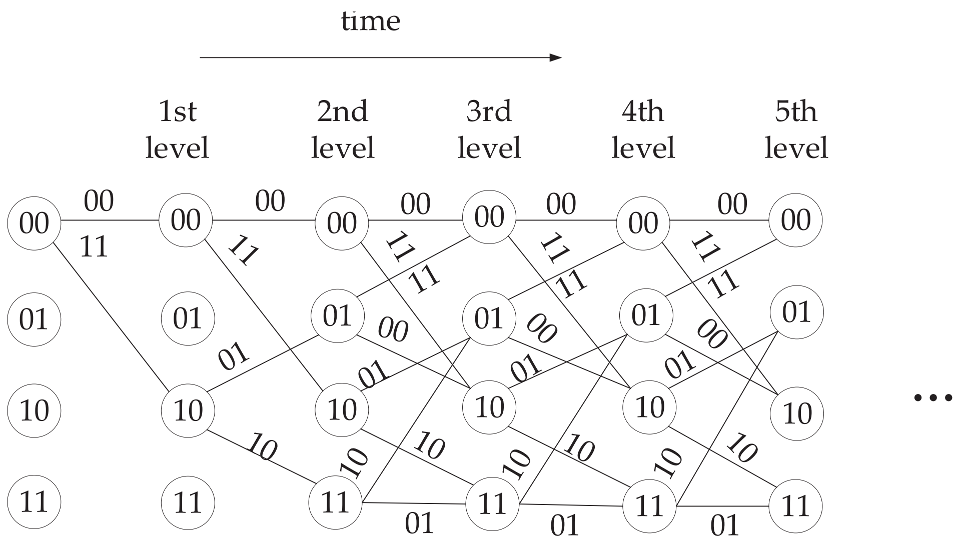

Figure 2.5: Trellis of a (2, 1, 2) code

When the first "0" is shifted into the encoder register, the state turns to

$$(0, c_{l-1}, c_{l-2}, ..., c_{l-m+1}).$$

When all m zeros are shifted into the register, the encoder is back to all zero state. During the termination process, the number of states is reduced by half in each step.

The polynomial to generate the codeword for convolutional code is picked from the optimum code table shown in Table 2.1 to achieve better efficiency.

Based on the optimum codes, the polynomial used to generate convolutional code in encoder is shown in Table 2.2 and Table 2.3.

2.4.4 Minimum Distance

The minimum free distance d_{free} of a convolutional code is the minimum Hamming distance between any two code sequences. It is also the minimum weight of all the code sequences.

Suppose Q_l is a set of paths in the trellis at time l that diverges from all-zero path and re-merge to all-zero path later. Let $\mathbf{z}$ to be a path in Q_l. Then the minimum distance can be computed by the equation:

$$d_{free} = \min_{l>m}\{\min w(\mathbf{z}) : \mathbf{z} \in Q_l\}.$$

The minimum distance of the (2,1,2) convolutional code shown as an example above is $d_{free} = 5$.

m	$g^{(0)}$	$g^{(1)}$	d_{free}
1	3	1	3
2	5	7	5
3	13	17	6
4	27	31	7
5	53	75	8
6	117	155	10
7	247	371	10
8	561	753	12
9	1131	1537	12
10	2473	3217	14
11	4325	6747	15
12	10627	16765	16

Table 2.1: Optimum rate $R = 1/2$ convolutional codes.

m	$P^{(0)}(x)$	$P^{(1)}(x)$
2	101	111
3	001011	001111
4	010111	011001
5	101011	111101
6	001001111	001101101
7	010100111	011111001
8	101110001	111101011
10	010100111011	011010001111
12	001000110010111	001110111110101

Table 2.2: Binary polynomial used for $R = 1/2$ convolutional codes.

v	$P^{(0)}$	$P^{(1)}$
2	$1+D^2$	$1+D+D^2$
3	$1+D^2+D^3$	$1+D+D^2+D^3$
4	$1+D^2+D^3+D^4$	$1+D+D^4$
5	$1+D^2+D^4+D^5$	$1+D+D^2+D^3+D^5$
6	$1+D^3+D^4+D^5+D^6$	$1+D+D^3+D^4+D^6$
7	$1+D^2+D^5+D^6+D^7$	$1+D+D^2+D^3+D^4+D^7$
8	$1+D^2+D^3+D^4+D^8$	$1+D+D^2+D^3+D^5+D^7+D^8$
10	$1+D^2+D^5+D^6+D^7+D^9+D^{10}$	$1+D+D^3+D^7+D^8+D^9+D^{10}$
12	$1+D^4+D^5+D^8+D^{10}+D^{11}+D^{12}$	$1+D+D^2+D^4+D^5+D^6+D^7+D^8+D^{10}+D^{12}$

Table 2.3: Polynomial used for $R = 1/2$ convolutional codes.

2.4.5 Maximum Likelihood Decoding

For an (n, k, m) convolutional code, each code sequence is considered to be a path in the code trellis. For an information sequence $\mathbf{c}$ consisting of L message blocks,

$$\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1, ..., \mathbf{c}_l, ..., \mathbf{c}_{L-1}).$$

After encoding, each coded sequence $\mathbf{v}$ is a path of $L + m$ branches long in the trellis

$$\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_l, ..., \mathbf{v}_{L+m-1}),$$

Suppose that after the channel, the received sequences becomes

$$\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, ..., \mathbf{r}_l, ..., \mathbf{r}_{L+m-1}),$$

and the l-th received block is

$$\mathbf{r}_l = (r_l^1, r_l^2, ..., r_l^n).$$

The maximum likelihood decoding finds the path $\mathbf{v}$ in all the paths so that the correct transmission probability $p(\mathbf{r}|\mathbf{v})$ is the largest.

For a binary code, a Q-ary output discrete memory-less channel(DMC), and a binary sequence $\mathbf{v}$, the conditional probability $p(\mathbf{r}|\mathbf{v})$ can be calculated as follows[6]:

$$p(\mathbf{r}|\mathbf{v}) = \prod_{l=0}^{L+m-1} p(\mathbf{r}_l|\mathbf{v}_l), \tag{2.1}$$

where $p(\mathbf{r}_l|\mathbf{v}_l)$ is branch condition probability. It can be calculated by:

$$p(\mathbf{r}_l|\mathbf{v}_l) = \prod_{i=1}^{n} p(r_l^{(i)}|v_l^{(i)}), \tag{2.2}$$

where $p(r_l^{(i)}|v_l^{(i)})$ is channel transition probability.

Define the log-likelihood function of path $\mathbf{v}$:

$$M(\mathbf{r}|\mathbf{v}) \triangleq \log p(\mathbf{r}|\mathbf{v}), \tag{2.3}$$

which is also called the metric of path $\mathbf{v}$.

Based on (2.2) and (2.3), we have

$$M(\mathbf{r}_l|\mathbf{v}_l) = \log p(\mathbf{r}_l|\mathbf{v}_l), \tag{2.4}$$

which is called branch metric. If combined with (2.2), we can have

$$M(\mathbf{r}_l|\mathbf{v}_l) = \sum_{i=1}^{n} p(r_l^{(i)}|v_l^{(i)}), \tag{2.5}$$

which is called bit metric.

For a binary systemic channel, the log-likelihood function turns into:

$$\log(p(\mathbf{r}|\mathbf{v})) = d(\mathbf{r}|\mathbf{v})\log(\frac{p}{1-p}) + (L+m)n\log(1-p), \tag{2.6}$$

where $d(\mathbf{r}|\mathbf{v})$ is the Hamming distance between r and v. If $d(\mathbf{r}|\mathbf{v})$ is minimized, $\log(p(\mathbf{r}|\mathbf{v}))$ is maximized. That's the main idea of the maximum likelihood decoding.

2.5 Viterbi Algorithm

First introduced by Viterbi in 1967 and recognized by Forney in 1973, Vitrebi algorithm is an implementation of maximum likelihood function. Viterbi algorithm is well-known in coding theory as an optimal algorithm that maximizes the performance for a long time. It is used for finding the most likely sequence of hidden states.

Viterbi algorithm acts like a sequence matcher, it exhaustively searches the possible sequence by using trellis to find the closest sequence with respect to the received sequence. In Viterbi algorithm, the decoder goes through the trellis level by level, and computes the metrics of all partial paths. Then, it compares all metrics entering into the same node, stores the path with largest metric and eliminates others paths. That path stored by this step is called survivor. At each level of the code trellis, there are 2^{km} nodes, and also 2^{km} survivors. When the code begins to terminate, the number of survivors will reduce. The last survivor at the end will be the maximum likelihood path for the algorithm to choose.

Viterbi algorithm can be roughly described with three steps:

Step 1. Starting at level $l = m$ in the trellis. The decoder computes the partial metric of the paths entering into the l-th order node. Store the survivor and metric for each node.

Step 2. Increasing l by 1. Compute partial metric for all paths entering a $(l+1)$-th order node. Add the branch metric to the metric of previous l-th order node where the survivor comes from. Store the metric for the largest metric for each $(l+1)$-th node, and eliminate all the other paths.

Step 3. If $l < L + m$, repeat Step 2. Otherwise, stop.

Viterbi algorithm stores the path with maximum probability in each node, and also stores back pointers to recover the optimal path. An implementation of Viterbi algorithm and a comparison with list decoding will be presented later.

2.6 Cyclic Redundancy Check

Cyclic redundancy check (CRC) is an error-detecting code commonly used in digital communication areas such as IEEE 802.1 and storage devices in order to ensure the correctness of transmission. Generally, CRC is classified into CRC4, CRC16, CRC32, CRC64 according to CRC length[7]. Among them, CRC32c is a well-known code that applies to lots of communication systems, such as iSCI. CRC has been widely used on many communication protocols, including Ethernet, asynchronous transfer mode, and fiber distributed data interface.

Typically for a CRC detecting process, when an error is detected, the receiver usually sends a "negative acknowledgment" back to the transmitter, and the transmitter sends the message again. But CRC technique only detects error, it doesn’t help to correct errors.

A polynomial is required to calculate CRC, since it is a polynomial based technique. Assume a polynomial in GF(2)(Galois field of two elements), it is a polynomial in which the coefficient x is either 0 or 1. Assume the length of the CRC bits is n_c. To calculate the CRC bits, the message polynomial, $m(x)$, is first right padded by n_c zeros, and then divided by the generator polynomial $p(x)$. For example, the information bits are 110101, they correspond to the polynomial $m(x) = x^5 + x^4 + x^2 + 1$. If the CRC generator polynomial $p(x) = x^3 + x^2 + 1$, then the polynomial division is $r(x) = \frac{(x^5+x^4+x^2+1)\times x^3}{x^3+x^2+1}$, which will be treated as CRC polynomial. The remainder is appended to the message to form the new message. The new message will have the structure:

$$\mathbf{m}' = [\mathbf{m}\ \mathbf{r}],$$

where $\mathbf{m}'$ is the generated new sequence with CRC appended.

Since all the operations are in GF(2), if the new message $\mathbf{m}'$ is transmitted correctly, it should be dividable by the generator $p(x)$. If the receiver finds that $\mathbf{m}'$ is not dividable by $p(x)$, which means that the reminder of the polynomial division is not zero, then it means error occurs during the transmission. This is a brief explanation of how CRC detects error.

2.6.1 Serial CRC Encoder

Assume the generator polynomial has the form

$$p(x) = p_0 + p_1 x + ... + p_n x^n,$$

where n is the number of CRC bits. In a GF(2) field, the parameter p_n is either 0 or 1, which decides the connection or disconnection between the XOR gate and the feedback path in structure Figure 2.6. The state of each flip flop is represented by $c_i (i = 0, 1, 2, ..., n)$. In serial CRC encoder, the data stream must be bit-serial, so the CRC bits are generated step by step with time[8].

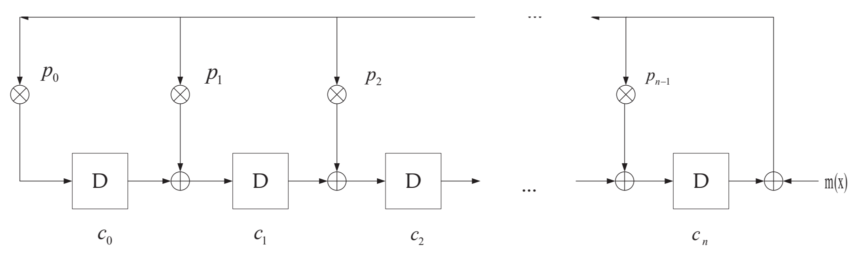

Figure 2.6: Serial CRC encoder

2.6.2 Parallel CRC Encoder

Different from serial CRC encoder, a parallel CRC encoder can calculate multiple CRC bits in one clock cycle, thus make the process more efficient, and the throughout is improved.

The structure of parallel CRC encoder is shown in Figure 2.7. The 64-bit data are divided into 4 16-bit blocks, and the CRC bits are generated and then added to the final CRC[8]. Four execution units are used to calculate the polynomial division.

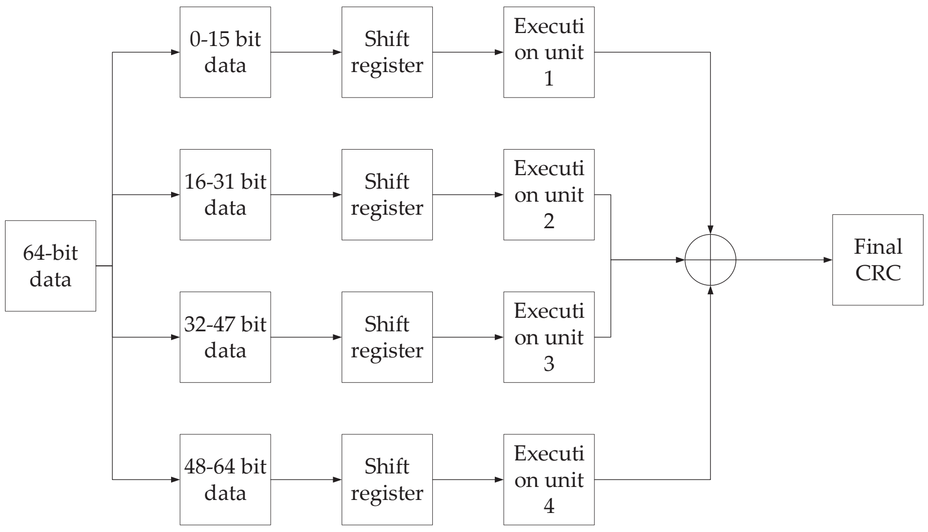

Figure 2.7: Parallel CRC encoder

Chapter 3

List Decoding of Convolutional Codes

Since Viterbi algorithm becomes popular, researches have introduced various generalizations of it, and list decoding is one of them. List decoding, introduced by Elias in the 1950s[4], is an alternative method of decoding for error control code. Instead of keeping all the candidates and calculating all the possible paths in each step, list decoding allows the decoder to choose and store certain number of paths from all the candidates in each step. Calculations are also limited into a certain number that only depends on the list size. Typically, list decoding is a sub-optional decoding compared to Viterbi decoding, but since it takes less storage and has much lower complexity, list decoding is more suitable for higher-memory codes. The complexity of list decoding and the comparison of performance of list decoder and Viterbi decoder with different memory will be described separately later in the thesis.

In the sections below, we will discuss more about list Viterbi decoding algorithm.

3.1 Summary of Viterbi Algorithm

As explained before, Viterbi algorithm searches the best sequence by using trellis to find the closest sequence from the received sequence from the channel. Let m be the memory of the convolutional code, then the number of states is 2^m. Let N be the total number of trellis sections, a typical Viterbi trellis is shown in Figure 3.1.

Let the minimum cost to reach state j at time t from the starting state to be $\Phi_t(j)$, and the cost from state i to state j be $c_t(j, i)$. Let the history of the best path to be stored in array $A_t(j)$. Thus the Viterbi algorithm recursion step can be described as[9]:

$$\begin{aligned}\phi_t(i) &= \min_{1\le j\le 2^m}[\phi_{t-1}(j) + c_t(j, i)],\\ A_t(i) &= \arg\min_{1\le j\le 2^m}[\phi_{t-1}(j) + c_t(i, j)],\\ &\qquad 1 \le i \le 2^m, 1 < t < N.\end{aligned}$$

Before we introduce the list decoding algorithm which the thesis uses, there will be some demonstration of other list decoding algorithms.

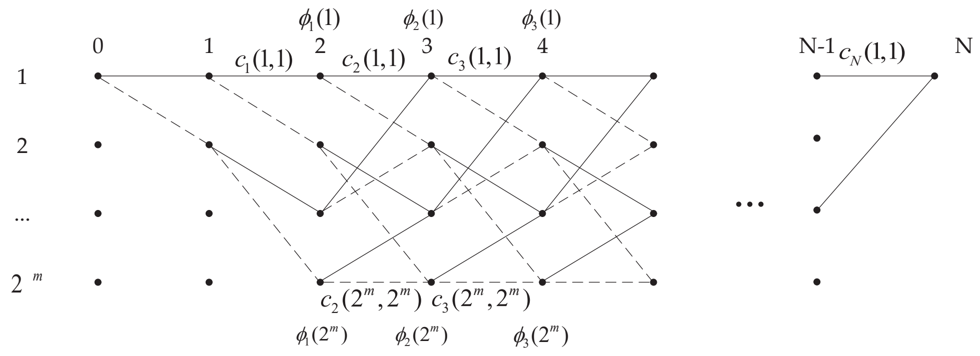

Figure 3.1: Trellis with 2^m states in Viterbi algorithm

3.2 Different List Decoding Algorithms

3.2.1 Parallel List Decoding

By computing l best paths into each state, the parallel list decoding finds l best paths simultaneously. In each state at each time, the algorithm compute $2^m l$ costs and store l paths with larger metric. It requires a matrix with $2^m l \times N$ to store the history for each time instant[9].

3.2.2 Serial List Decoding

The serial list decoding algorithm finds l most likely paths once in a time. The main advantage of the algorithm is that k-th best candidate is only computed if the previous $k-1$ candidates are determined to be erroneous. By avoiding a lot of unexpected calculations, the complexity is reduced significantly[9].

3.2.3 The M-Algorithm

The M-algorithm is a breadth-first sorting algorithm. It only keeps a fixed number l paths, and deletes all other paths according to the criterion. Assume one state in the trellis extends to two paths in each step. M-algorithm proceeds by extending the l paths into $2l$ new paths. Then a metric comparison is performed to find the best l paths out of all new paths. l paths are remained, all other l paths are deleted before moving to next step.

To sum up, the basic idea of M-algorithm is to extend two paths form each remaining path, order the list to find M best paths, and delete the rest of paths[10].

3.3 List Decoding in the Thesis

The list decoding algorithm used in the thesis is similar to M-algorithm. But in order to fit CRC to the list decoding, some changes have been made. The number of states that remain in the list per step is called list size l. Like M-algorithm, each state in the list extends to two new paths, resulting in totally $2l$ paths. After calculating the metric, the algorithm then sorts and compares all the paths. Then, instead of selecting best paths, it selects l best states. If k-th path and $(k+1)$-th best paths are expanded from the same state, then both paths history are stored for backtracking afterwords, but the state only takes one candidate place. As a result, the list with l states can contain l_m paths$(l \leq l_m \leq 2l)$. This is because after CRC is introduced, we want to have more candidates in the list, and CRC detecting is more suitable for a state-based algorithm. The algorithm can be summarized as follows:

Step 1. Initialization:

Define a matrix with size $l \times N$ to store calculated metrics. l is list size, and N corresponds to the the length of the received sequence. Then define a matrix with the same size to store the previous state of a chosen path for backtrack, which is called backtrack matrix.

Define an arrays of size $2l$ to store the best $2l$ states in each step. The arrays will be updated at each loop.

Step 2. Recursion:

At time $t < N - 2^m$, calculates metric $c_t(j,i)$ and add to the previous calculated metric. Let $\phi_t(i,k)$ to be the k-th previous calculated metric where $1 < k < l$. We have:

$$\phi_t(i,k) = \min_{1 \leq j \leq 2^m} [\phi_{t-1}(j,k^*) + c_t(j,i)], \tag{3.1}$$

where k^* is the previous state of the k-th best path.

Step 3. Termination:

At time $t = N - 2^m$, all the information bits have been processed and the remaining are termination bits with length m. At the end of the termination bits, only one candidate is supposed to survive, and the list size at time $t = 2^m$ is 1. At $N - 2^m < t < N$, the list length l_t will reduce to $l_t = \lceil l \times 0.5^{t-N+2^m} \rceil$.

Step 4. Backtracking:

If the i-th path is selected to be the best path, then a backtrack is performed by selecting i-th row and 2^m-th column in the backtracking matrix. The previous state is stored in the matrix. Take the value as new row and move back to $(2^m - 1)$-th column. Loop back until it returns to column 1.

List size is an essential factor for list decoder. With a small list size, a list decoder can avoid calculating large number of states in each step even if memory is large, thus reducing decoding complexity significantly. But as a price, the probability that the proper decoding path eliminated during the decoding process and not included in the list will be increased. If the proper path is not able to survive in the list in one step, there will be no chance to get a correct decoding result for the entire package. So a good balance between complexity and performance is required for a list decoder.

Figure 3.2 shows an example of this list decoding algorithm. In the example, code memory $m = 2$, code rate $R = 1/2$, and list size $l = 2$. In the 3rd step, two states expand to 4 states, and each of the 8 branches are calculated. Then, instead of storing all 4 states, the decoder only stores 2 states with highest metric. These

2 states are considered to be survivors, and the rest of two states are discarded. Then, in the 4th step, only 4 branches are expanded from the two survivors, reaching 4 different states, and still two of them will be selected as survivors. It proceeds until termination.

Simulations of list decoding algorithm with fixed list size and fixed memory are shown in the Figure 3.3 and 3.4. In Figure 3.3, the list size is fixed to be 8, and the performance of convolutional codes with different memory are compared. As expected, with list size fixed, the performance of list decoder increases with memory. In Figure 3.4, the memory of the codes are fixed to be 5, and different list sizes are used for comparison. Simulation shows that the performance improves with increasing list size.

3.4 Complexity

The complexity of list decoder is independent of the code memory. The decoder only stores and calculates two times of the list size, so the memory is not affecting the complexity for list decoding.

In Viterbi algorithm, for a rate $R = 1/2$ code, at each level in recursion step, metric of all the 2^{m+1} branches are calculated, then all metric are added into previous states. Considering two bits are taking into calculation at each level for a binary channel, it takes $2 \times 2^{m+1}$ multiplications and 2^m additions for each level. Besides, because there are 2^m states in the state diagram of the encoder, the decoder must reserve 2^m states to storage the survivor. Since the storage usage goes exponentially with code memory m, it is not feasible to use codes with large memory for Viterbi algorithm in practice.

In the list decoding algorithm, at each level, there are $2 \times l$ branches being calculated, and the metric are added to l previous states. It uses $2 \times 2 \times l$ multiplications and l additions for each level. Then, a sorting step is taken to sort l best states out of $2l$ states candidates in the list. The typical complexity of sorting $2l$ elements is $2l \log(2l)$. But in the algorithm, there is no need to sort all elements in an order, since it just requires to sort out top l best elements, so the sorting complexity is reduced to l[10].

3.5 Comparison with Viterbi Decoding

With proper choice of list size, a list decoder can achieve similar performance as Viterbi decoder with lower memory. Figure 3.5 compares the performance of list decoders with fixed memory $m = 10$ but different list sizes with memory-5 and 10 Viterbi decoder. Memory $m = 10$ indicates that there are 1024 states in the trellis, which is a pretty large number for Viterbi algorithom which requires calculating all the states in each step, but since list decoding only processes those candidates which exist in the list, the complexity doesn’t really increase with memory. In this case, list decoding will save large amount of time during the simulation. For comparison, the performance of Viterbi decoder with memory $m = 5, 10$ are also shown in the figure.

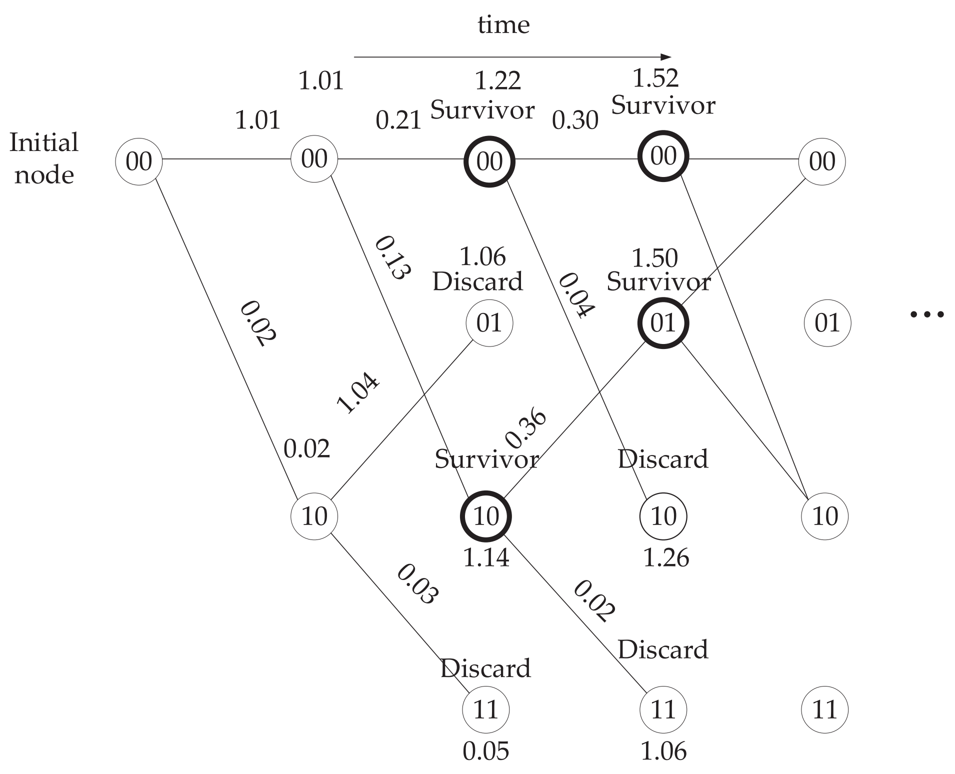

Figure 3.2: An example of list decoding algorithm with $l = 2$

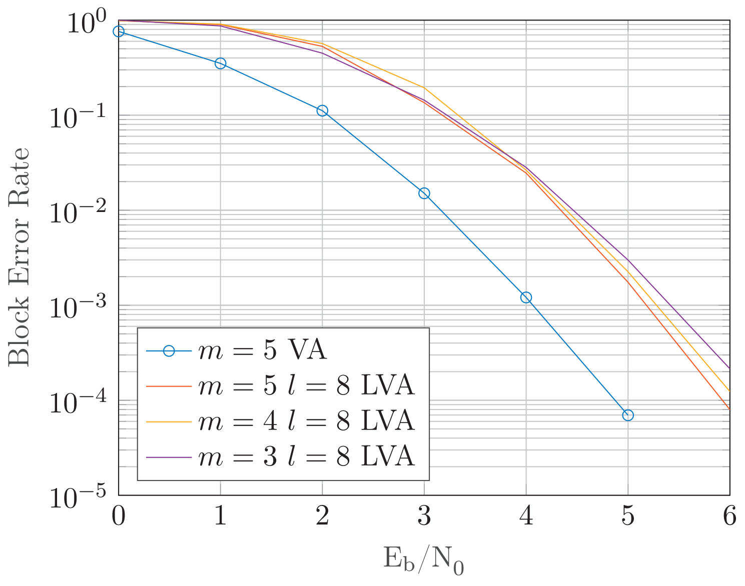

Figure 3.3: List decoding with fixed $l = 8$

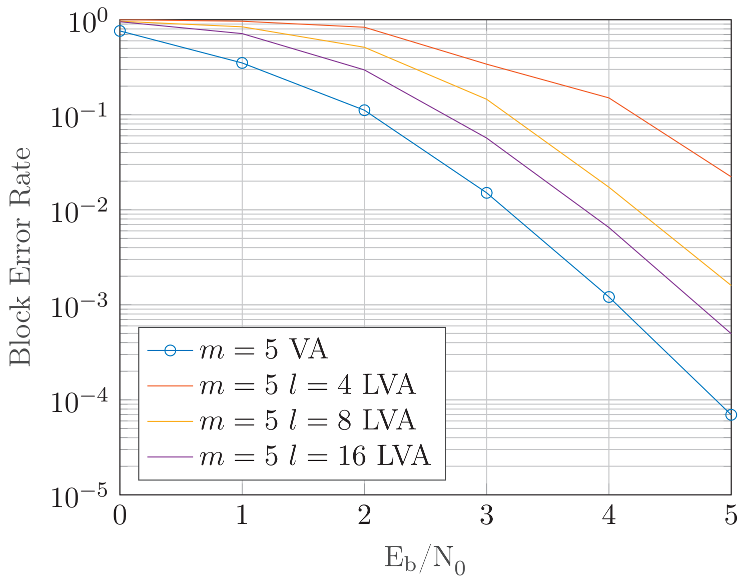

Figure 3.4: List decoding with fixed $m = 5$

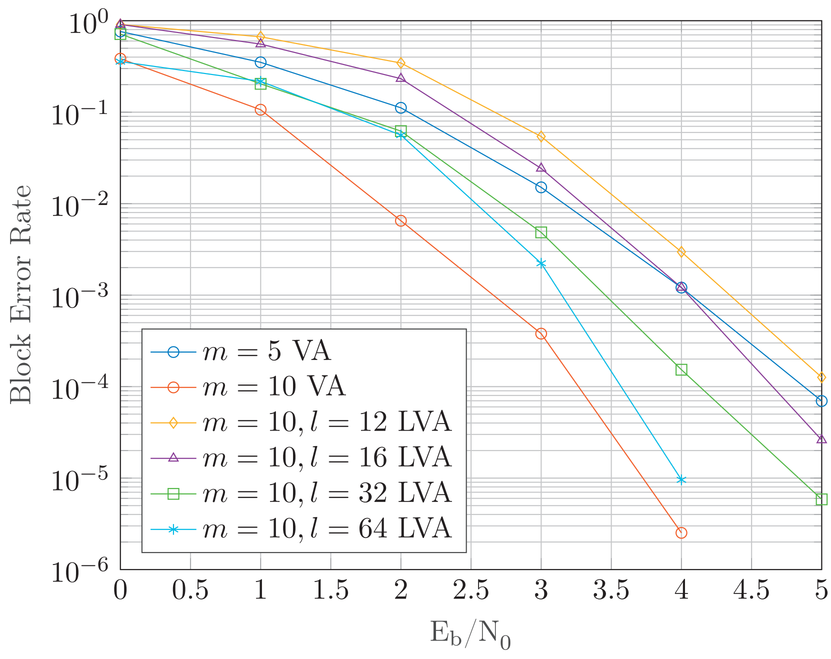

Figure 3.5: List decoding with same memory and different list sizes

When list size increases, list decoder can have a significant improvement in performance. With list size 16, a list decoder for a memory-10 convolutional code has a better performance than a Viterbi decoder for a memory-5 convolutional code. Also noticeable is the fact that a list decoder with list size 32 actually calculates $2l = 64$ paths per step, which is exactly the same number of paths as a memory-5 Viterbi decoder will calculate in each step. So by choosing list size and memory wisely, a list decoder can really over-perform a Viterbi decoder with approximately the same complexity.

It is interesting to see that a balance between lower complexity and better performance can be achieved with large memory and relatively small list size. In order to reduce the possibility of losing correct path in the list and to achieve better performance, CRC will be introduced, and CRC aided list decoder will be fully explained in the following chapter.

Chapter 4

CRC Aided List Decoding

In previous chapter, we have shown that for large memory codes with small list size, list decoding can outperform Viterbi decoding with same complexity. This chapter will focus on exploring the method to add CRC to the decoder, and explain the factors that could possibly affect coding performance with the CRC added, such as the position to insert CRC, and puncturing to remain the code rate. A CRC aided decoding scheme can be described as follows:

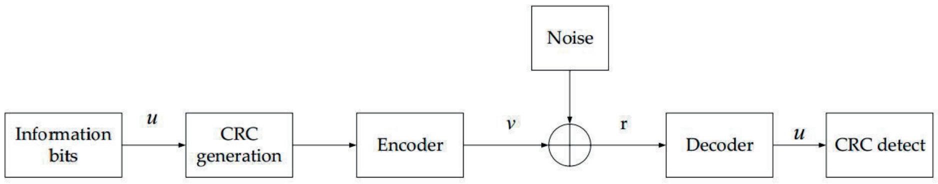

Figure 4.1: CRC aided decoding scheme

In the CRC aided decoding scheme, parity bits are generated based on the information bits u. These parity bits are attached to the information bits, before processed by the encoder. The codeword v are generated and sent to the channel, with noise added. The receiver then decodes the received codeword r, then sends the most likely information $\widehat{u}$ to CRC detector.

With CRC aided, whether or not the convolutional codes can get stronger, and can correct the block errors caused by the sub-optimal list decoder, are the main topics that the thesis is focusing on.

4.1 CRC Implementation

The project uses systematic cyclic redundancy check codes. The encoder creates parity bits according to a specified generator polynomial and appends them to the information bits, thus fixed-length check values are added to the information bits.

The main idea is to generate CRC based on the information bits of a convolutional code in the encoder, and eliminate those paths that are not satisfying CRC check on the decoder, and to choose the most likely path remaining in the list.

Whenever a block error occurs in a list decoder, generally there are two kinds of reasons. One is that the proper path is eliminated from the list, the other is when choosing the best path at the final step, the decoder choose a wrong path which has a better metric than the correct one. When the list size is small, the first situation occurs frequently, and becomes the main reason of causing a block error in list decoder. The error will be analysed later on the thesis. And when the list size is large, the second reason becomes the dominating factor for block errors.

CRC can actually help reduce both errors, especially the error caused by losing correct path in the list, if CRC bits are positioned separately in the code trellis. Following are some more discussions about CRC positions.

4.2 CRC Inner Encoding

The code can have a different performance if CRC bits are added to different positions in code trellis. A straightforward implementation is to add all CRC bits at the end of the code. In this case, the decoder will function normally until reaching the end of the information bit, and then perform CRC check to all the candidates in the list. Those paths that not satisfying CRC will be eliminated. The correct path can always survive CRC check, but other paths may not. For n_c CRC bits, the chance of a random path passing CRC check is $1/2^{n_c}$. The probability of block error caused by choosing a wrong path with a larger metric than that of a true path can be reduced by adding CRC check in this way.

Another implementation is to spread CRC bits into different parts of the trellis. Whenever running into a CRC checking step, the decoder will perform a CRC check to all the candidates in the list, and only keep those paths that can pass CRC check. Typically the remaining paths are smaller than the list size, so next few steps the branches will extend several times until the list is full again. By performing separate CRCs along with the trellis, the correct path will be more likely to survive and will have more chance to extend to following states, thus reducing block error caused by losing correct path in the list.

The two implementations are shown in Figure 4.2. The first one is adding all CRC bits after the information bits, the second one is to divide information bits into different subsequences, and generate CRC for each subsequences.

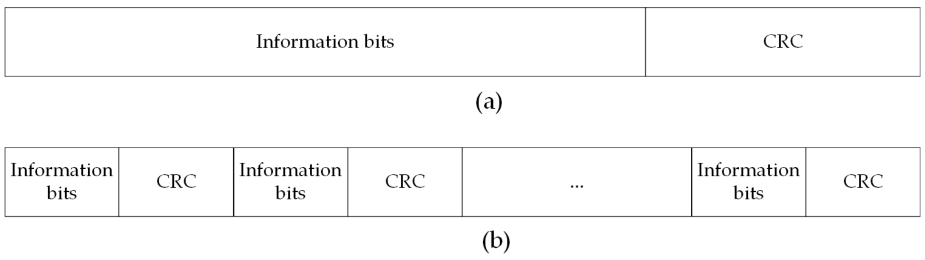

Figure 4.2: Adding CRC into different position
(a) Scheme 1: Adding CRC to the end of the sequence
(b) Scheme 2: Spreading CRC between the sequences

4.3 Puncturing of Convolutional Codes

In coding theory, puncturing is the process of removing some partiy bits after encoding. It has the same effect as encoding with an error-correction code with a higher rate, or less redundancy. In fact, the same Viterbi or list decoder can be applied to the original trellis structure regardless of the number of bits that have been punctured, as a consequence puncturing considerably increases the flexibility of the system without significantly increasing its complexity.

In the simulation, information sequence of $n = 96$ bits are generated. Then $n_c = 4$ bits CRC are added in between the information bits, as shown in Figure 4.3. After that, the information sequence are encoded by the convolutional code encoder. Then, termination bits are added at end of the message. For a $(2, 1, 10)$ convolutional code, there are $2m = 20$ termination bits added. The total codeword length is 220 bits in this example. The overall code rate $R = \frac{n}{(n+n_c)/R_c+m/R_c}$, where $R_c = 0.5$ is rate of the convolutional code. If more CRC bits are added, the overall code rate will decrease and it will influence the performance.

48 bits Information	2 bits CRC	48 bits Information	2 bits CRC

Figure 4.3: Sequence structure

Adding CRC changes code rate. In order to have a fair comparison between different decoding methods, puncturing a limit number of bits is required to maintain the code rate.

Minimum distance is typically considered to be the minimum code weight for which a corresponding path that returns to initial state in the fewest steps. That path can always be found easily given a certain trellis. By puncturing out the zeros in that certain path, the code weight is not affected, which means the minimum free distance of the code is not reduced, but the code rate is actually increased. The same method is also used in the project.

Figure 4.4 shows a path with minimum distance from a memory $m = 10$ convolutional code, as well as the bits to be punctured. The generator polynomial is chosen from Table 2.3. The minimum distance $d_{free} = 14$, which is also equal to the minimum code weight of all return-to-zero paths. Puncturing the zeros from this path will not reduce the minimum distance. From this example, to compensate the 4 CRC bits added on the information bits, it is appropriate to choose and puncture 4 bits from the locations of those 8 zeros in the path with minimum free distance.

4.4 Realization of CRC Aided List Decoding

As described before, when applying CRC check in the decoder, all the paths are checked and those don’t satisfy CRC check will be eliminated. When realizing it in program, there is one important issue to consider: the correct path might have

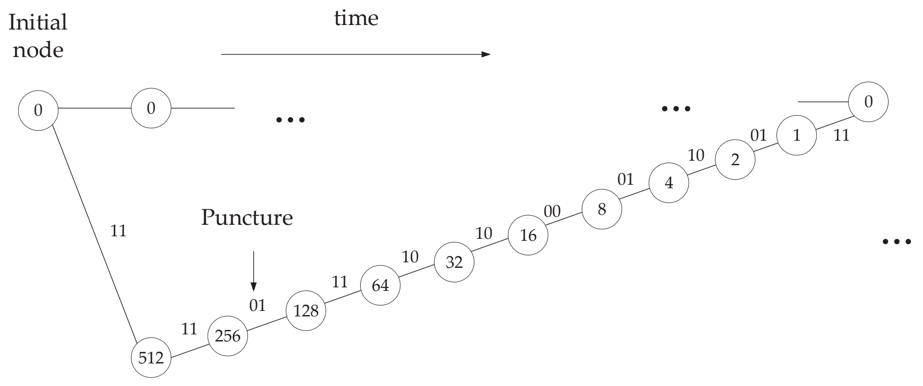

Figure 4.4: Puncturing for an $m = 10$ convolutional code

lost before decoding process reaches CRC check step, and all the candidates in the list are not satisfying CRC check. This situation can happen frequently at low SNR range, so it is not trivial to consider how to realize CRC check properly to avoid it. In the project, two methods are used to make sure that there are always some paths in the list.

4.4.1 Increase List Size Before Reaching CRC States

The most straightforward method is to avoid the situation of losing correct path before CRC check. This is a complex topic, since it is also the main cause of frame error when list size is small. By just increasing list size for some certain steps, the possibility of not losing the correct path will increase. The increase of list size is only a few steps before reaching CRC state, otherwise it will be unfair when comparing complexity. Generally, the list size only increases a few steps before CRC check. It starts at n_e steps before CRC check. At that step, $2l$ paths are calculated, and instead of choosing l of them to remain in the list, they are all kept. Next step, the list size increase to $2l$, $4l$ candidates are calculated, and still all stored in the list. Until the step before CRC check, the list contains $2^{n_e}l$ candidates, where n_e is the number of steps before CRC check when list size begins to increase. By increasing list size in this way, although we can't make sure that the correct path is still in the list, but the chance that neither of the candidates in the list satisfy CRC check is significantly reduced.

Figure 4.5 shows the performance of increasing list size different steps before CRC check. Extending list size $n_e = 1$ step before the CRC is significantly better than the performance without extending, and extending more steps earlier would make the performance better, but the gap between them decreases.

With increasing list size for a few steps, a real list size l_r is calculated as an average list size for each step. Given $l = 16$, $m = 10$ and that 8 bit CRC bits are spread into 4 different locations between the 100 bit information sequence, Table

4.1 shows l_r of extending list size from different n_e. From the table, when n_e is relatively small, we can still consider it having the same complexity with a normal list decoder.

n_e	l_r
0	16
2	16.697
3	18.782
4	20

Table 4.1: Average list size l_r for different n_e

4.4.2 Generate CRC in Decoder

There is another method to avoid the situation. Instead of applying a CRC check, the decoder now generates CRC bits, making all survivors reach a certain state based on the path corresponding to CRC bits.

An example is shown in Figure 4.6. Assume the information is divided into parts, and CRC bits are generated at end of each subsequences. In the CRC check step, a backtrack is made for each state surviving in the list, a most-likely information sequence is decoded for each state. Afterwards, CRC bits are generated according to the information sequences of each state using the same CRC generator in the encoder. For the marked path, based on the detected information sequence 000100, two CRC bits 11 are generated, thus leading the state to transition from current state 00 to state 01. The metric of this step is calculated normally and added to the metric from previous step. Afterwards, the decoding process will return to normal and then continue until the next CRC step.

The advantage of the method is that since CRC bits are generated based on previous information subsequences decoded up to current state, it is not possible to encounter the situation where a large number of survivors can not pass CRC check.

A simulation is made to compare the block error rate of both methods. The result is shown in Figure 4.6.

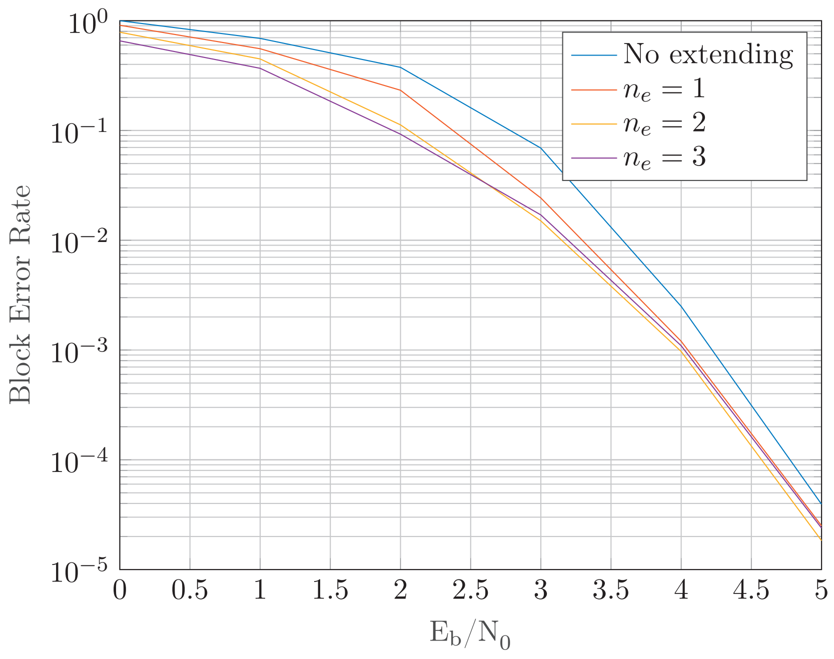

Figure 4.5: The performance of increasing list size several steps before CRC check

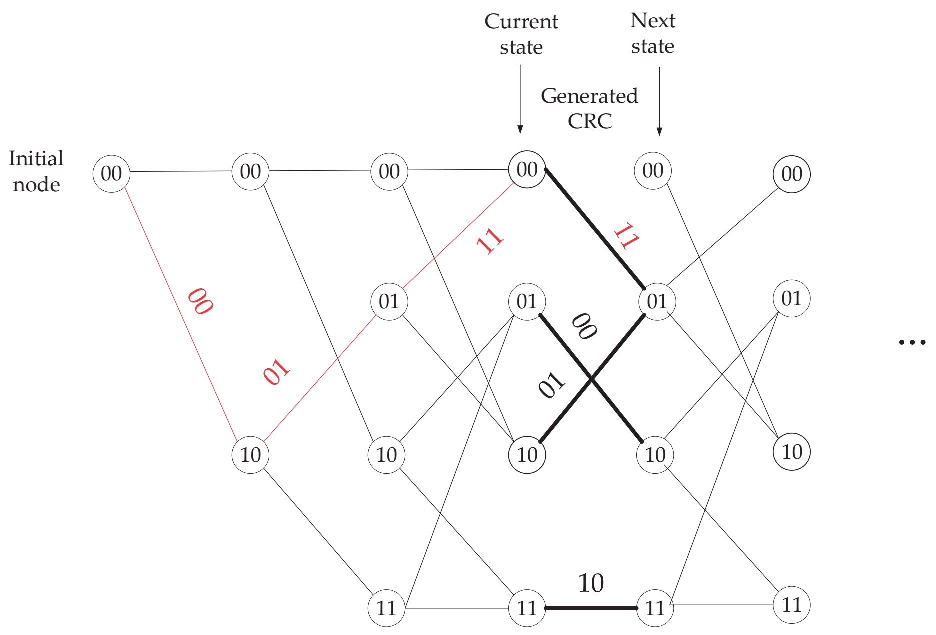

Figure 4.6: CRC bits generation in decoder

Chapter 5 Result and Analysis

The error probability achieved in the simulation depends on the code, the code rate, its free distance, channel SNR, the signal quality, and so on. In the thesis, the channel remains the same, information bits are generated randomly under the same criterion.

5.1 Simulation Result

Figure 5.1 is a performance comparison with a convolutional code with $m = 5$ using Viterbi algorithm, an $l = 16$ list decoding algorithm, and an $l = 16, n_c = 4$ CRC aided decoding algorithm, where n_c is the length of the CRC bits. It uses separate CRCs, which means the information is divided into two parts, and each part has a 2-bit CRC for checking. It uses the first introduced method in Chapter 4, where the list size is increased before reaching CRC states. In Figure 5.1, the dash lines are CRC-aided LVA, where CRC parity bits have length $n_c = 4$ and $n_c = 8$, and are spread into different locations in the subsequence. It is clear that a CRC aided list decoder has a better performance than a typical list decoder. In other words, CRC helps to reduce the block error probability for convolutional code.

After that, a few changes are made for comparison. All simulations below use the same $m = 10$ convolutional code, and in the decoder, list size l is fixed to be 16. Figure 5.2 shows the performance of adding all CRC bits in the end of the information sequence(Scheme 1), as well as spreading them into the sequence(Scheme 2). In the figure, the length of CRC bits $n_c = 4$, and in Scheme 1, the CRC bits are separated into 2 sections. As is shown in the figure, Scheme 1 has a slightly better performance than Scheme 2. Figure 5.3 is a slightly different comparison, where $n_c = 8$, and in Scheme 1, CRC bits are separated into 4 sections. The improvement is more obvious in this figure.

Figure 5.4 shows the performance of the first implementation(referred to as Im 1), which is extending list size before CRC check, and second implementation(referred to as Im 2), which is generating CRC bits in decoder to specify trellis path. The length of CRC bits n_c is fixed to be 4. Figure 5.5 is a similar figure, where the length of CRC bits is set to 8.

From the simulation above, it can be analysed that Scheme 1 with separated CRCs has a better performance than Scheme 2 that adds all CRC bits in the

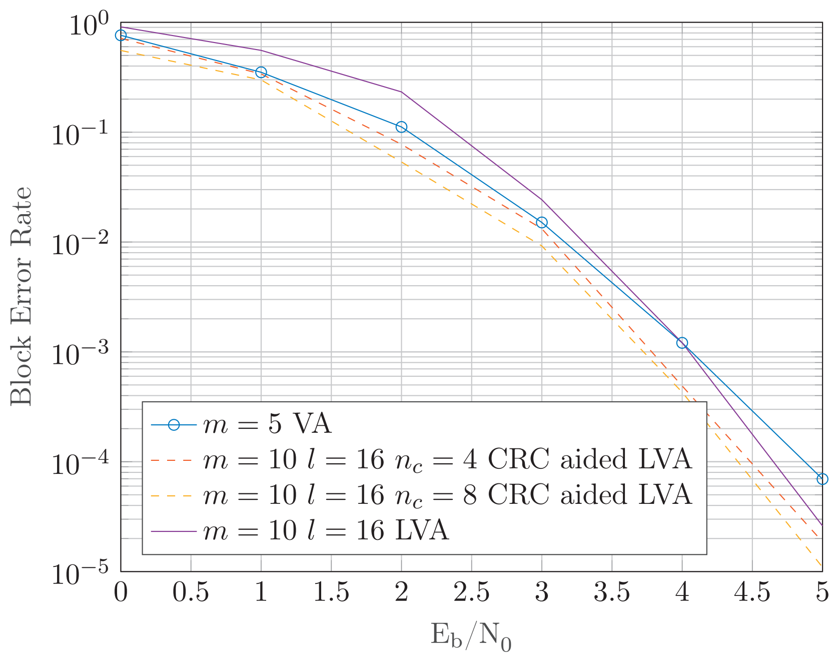

Figure 5.1: Comparison with LVA and CRC aided LVA

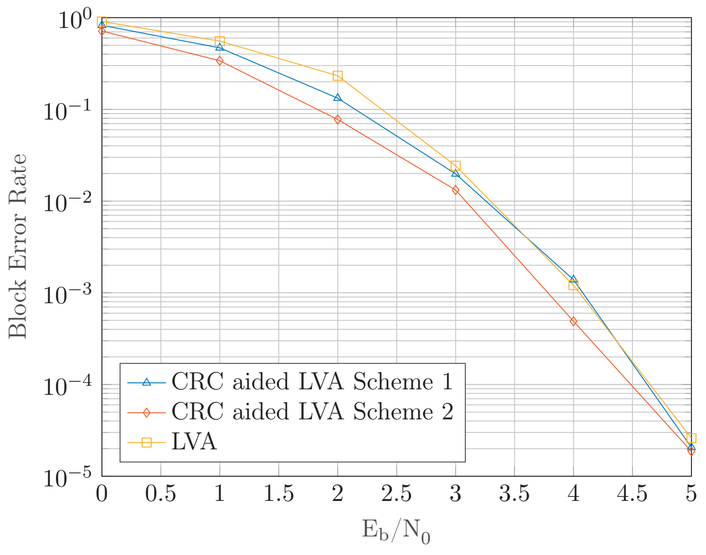

Figure 5.2: Comparison of different CRC position with $n_c = 4$

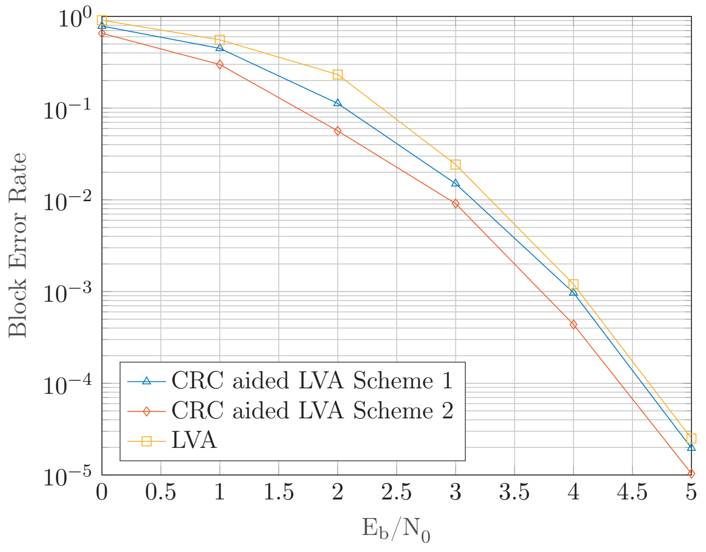

Figure 5.3: Comparison of different CRC position with $n_c = 8$

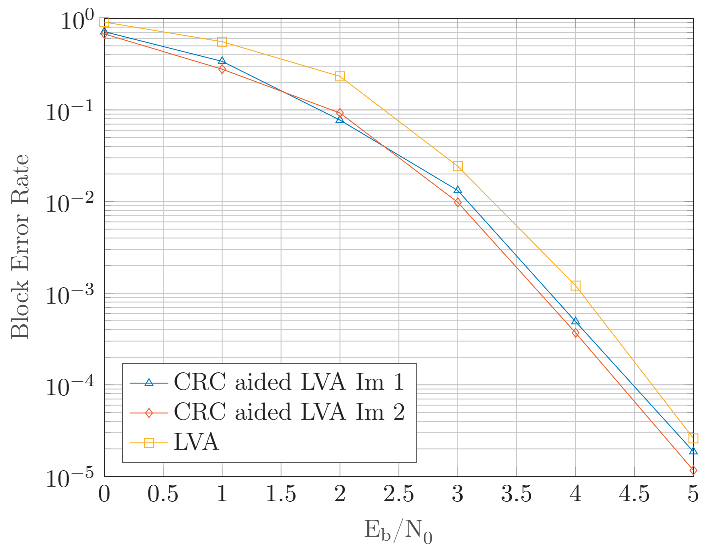

Figure 5.4: Comparison of different CRC implementations, $n_c = 4$

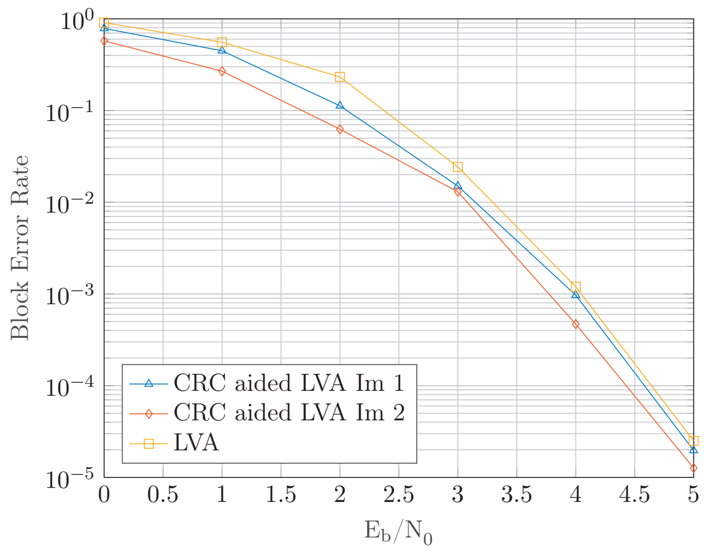

Figure 5.5: Comparison of different CRC implementations, $n_c = 8$

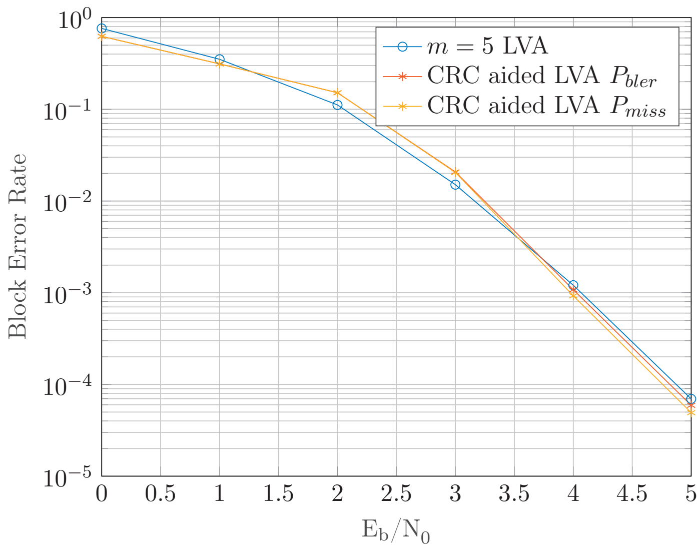

Figure 5.6: Comparison of block error and P_{miss}

end, especially for lower SNR. The second implementation with CRC generated in decoder is a bit more better than the first implementation, and it is also more efficient than increasing list size before CRC check.

5.2 Error probability Analysis

Block error probability can be divided into two parts. One is the probability denoted by P_{miss}, which is the probability that the correct path is lost from the list. The other one P_r is the probability that the decoder chooses a wrong path that has a better metric than a correct one. So block error probability can be expressed as:

$$P_{bler} = P_{miss} + (1 - P_{miss})P_r \tag{5.1}$$

Usually, for a convolutional code with small list size, P_{miss} is larger than P_r, which the main reason leading to the block error. Figure 5.6 shows that P_{miss} is almost identical to P_{bler}, proving that it is the main reason of the block error.

To be more specific, CRC does reduce both probabilities when is applied to a list decoder.

A simulation is made to show the comparison. The maximum number of errors is set to 10, meaning the simulation stops when the decoder detects 10 errors caused by one of each situation.

Figure 5.7 and Figure 5.8 are two figures showing the influences of CRC on both P_{miss} and P_r. The memory of the convolutional code used in the figure is 5.

For the CRC aided list decoder, $l = 8$, and $n_c = 4$. Both simulations use Scheme 2 and Implementation 1 introduced above.

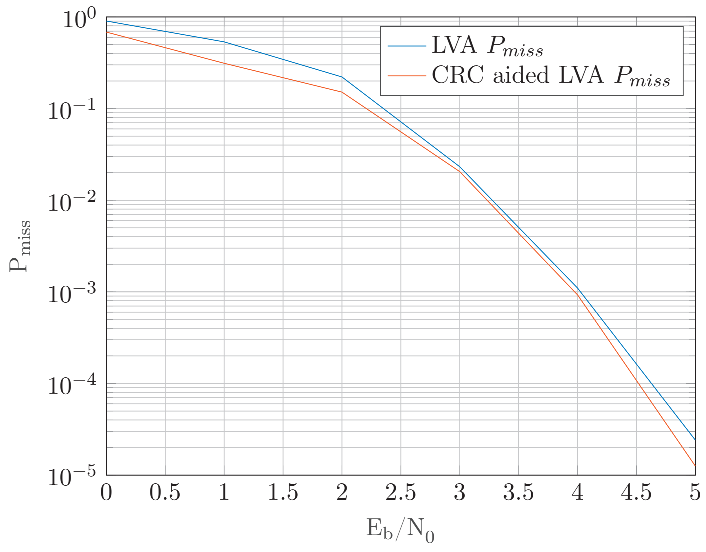

Figure 5.7: P_{miss} Comparison

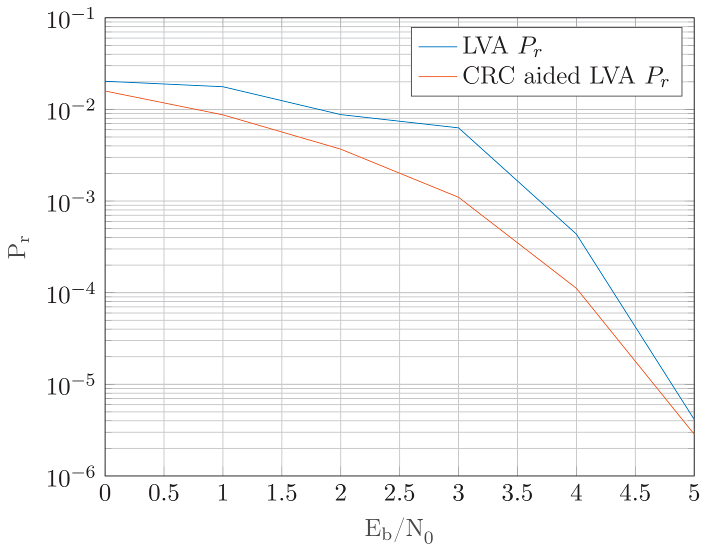

Figure 5.8: P_r Comparison

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Throughout this thesis we have studied an efficient realization of a sub-optimal list decoding, how CRC helps to minimize error probability, and two implementations of CRC aided list decoder. After that, a simulation is made to analysis error probability.

6.2 Future Work

Applying CRC check to improve coding performance is a deep area, and there is still a lot of work can be done to continue the research.

This thesis ran simulation mostly on an $m = 10$ convolutional code, and the list decoding results is always compared with $m = 5$ and $m = 10$ Viterbi algorithm. The simulation takes more time when simulating higher memory Viterbi codes, which makes the result lack of comparison. Future research will make more comparison with a larger variety on code memory.

Another potential extension is code rate. The thesis only uses a $R = 0.5$ rate convolutional code, and when CRC is added, we apply puncturing to maintain the code rate back to 0.5. It will also be interesting to see if we make some changes at code rate, and how the resulting performance is compared with the one without CRC but of the same code rate.

Also, the simulation of the two different errors P_{miss} and P_r are only tested on a few situations, since P_r is usually small and it takes a lot of time to get a smooth figure out of it. If given more time, there might be some more improvement in error probability analysis.

In short, there are still a lot things to be investigated and many tests with different parameters should be simulated, to fully understand and find a better way to optimize the performance when using the combination of convolutional codes with list decoding and CRC.

References

[1] I. Tal, A. Vardy, *How to Construct Polar Codes*, IEEE Transactions on Information Theory, vol. 59, no. 10, pp. 6562-6582, Oct. 2013.

[2] I. Tal, A. Vardy, *List Decoding of polar Codes*, IEEE International Symposium on Information Theory Proceedings, 2011.

[3] E. Johansson, *List Decoding of Polar Codes*, Master Thesis in Department of Electrical and Information Technology, Faculty of Engineering, Lund University, Oct. 2017.

[4] P. Elias, *List decoding for noisy channels*, Technical Report 335, Research Laboratory of Electronics, MIT, 1957.

[5] Rolf Johannesson, K.S. Zigangirov, *Fundamentals of Convolutional Coding*, IEEE Press, Piscataway, N.J., 1999.

[6] A. J. Viterbi, *Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm*, IEEE Transactions on Information Theory, IT-13, pp. 260-269, Apr. 1967.

[7] Q. Al-Doori, O. Alani, *A multi polynomial CRC circuit for LTE-Advanced communication standard*, 2015 7th Computer Science and Electronic Engineering Conference (CEEC), Sep. 2015.

[8] L. S. Prabha, R. S. Geethu, *Architecture of Parallel CRC Encoder Using State Space Transformations*, 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), July 2018.

[9] N. Seshadri, C. E. W. Sundberg, *List Viterbi Decoding Algorithms with Applications*, IEEE Transactions on Communication, vol. 42, no.2/3/4/, pp. 313-323. 1994.

[10] J. B. Anderson and S. Mohan, *Sequential Coding Algorithms: A Survey and Cost Analysis*, IEEE Transactions on Communications, vol. 32, pp. 169-176, Feb. 1984.

[11] R. Wang, W. Zhao, and G. Giannakis, *CRC-assisted error correction in a convolutionally coded system*, IEEE Transactions on Communication, vol. 56, no. 11, pp. 1807–1815, Nov. 2008.

Printed by Tryckeriet i E-huset, Lund 2019

LUND UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2019-724
http://www.eit.lth.se