Efficient Soft-Decision Decoding Algorithms for Linear Block Codes Using Algorithm A*

by

Yunghsiang Sam Han

Abstract of Dissertation

August, 1993

A class of novel and efficient maximum-likelihood soft-decision decoding algorithms, and a new class of efficient suboptimal soft-decision decoding algorithms for linear block codes are presented in this dissertation.

The approach used here converts the decoding problem into a problem of searching through a graph for an equivalent code of the transmitted code. Algorithm A^*, which uses a priority-first search strategy, is employed to search through this graph. This search is guided by an evaluation function f defined to take advantage of the information provided by the received vector and the inherent properties of the transmitted code. This function f is used to reduce the search space drastically and to adapt these decoding algorithms to the noise level.

Simulation results for the $(128, 64)$ binary extended BCH code show that, for most real channels of the 35,000 samples tried, the proposed maximum-likelihood soft-decision decoding algorithms are fifteen orders of magnitude more efficient in time and in space than that proposed by Wolf. Simulation results for the $(104, 52)$ binary extended quadratic residue code are also given.

The simulation results show that for the samples tried, the performance of a proposed suboptimal soft-decision decoding algorithm is at most within 0.25 dB of

the performance of a maximum-likelihood soft-decision decoding algorithm for the $(104, 52)$ binary extended quadratic residue code and within 0.5 dB for the $(128, 64)$ binary extended BCH code for very low signal-to-noise ratios.

We also determine an upper bound of the probability distribution of the computations performed in the proposed maximum-likelihood soft-decision decoding algorithms. The results of this bound show that these decoding algorithms are efficient for most practical communication systems.

Efficient Soft-Decision Decoding Algorithms for Linear Block Codes Using Algorithm A*

by

Yunghsiang Sam Han

B.S. National Tsing Hua University, Taiwan, 1984

M.S. National Tsing Hua University, Taiwan, 1986

Dissertation

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in School of Computer and Information Science
in the Graduate School of Syracuse University

August, 1993

Approved _____

Date _____

Acknowledgements

I would like to thank Dr. Carlos Hartmann for his encouragement and guidance. This work would not have been possible without his advice and commitment. I would also like to thank Dr. Kishan Mehrotra for his help in the analysis of algorithms. Thanks are also due to Dr. Harold Mattson, Jr., and Dr. Luther Rudolph for providing invaluable advice at a very early stage of this work. I am grateful to Mr. Chih-chieh Chen for implementing a prototype of the decoding algorithm in software and pointing out an important property of that decoding algorithm.

The School of Computer and Information Science supported me throughout my graduate career. I am also grateful to the Northeast Parallel Architectures Center for allowing me the use of their facilities. Dr. Hari Krishna, one of the best instructors I have ever had, deserves special mention for having helped make clear to me some important concepts in coding theory.

Special thanks are due to Mrs. Elaine Weinman for her invaluable help in the preparation of this manuscript. I also thank Dr. Chilukuri K. Mohan for his comments on this work.

The work presented in this dissertation was partially supported by the National Science Foundation under Contract NCR-9205422.

I would like to give special thanks to my family, my wife Chihching and my son Justin. This dissertation would not have been possible without my wife who

supported and encouraged me at every step of my long journey toward my degree.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The goal of any communication system is to transfer information correctly from source to destination. The medium used for transmission is called a *channel*. Since most of the channels used are subject to noise, for reliable transmission of information over a noisy channel, specific techniques are required to correct errors introduced by channels.

The error-control technique discussed in this dissertation originated with the publication of "A Mathematical Theory of Communication" by Shannon [34] in which he proposed the ideas of source coding and channel coding. He proved that without loss of generality we can treat source coding and channel coding as two separate problems, and information transmitted over a channel can be regarded as binary digits. Hence, the problem in channel coding is to find an encoder and a decoder such that the probability of error can be minimized for transmission. Shannon also proved that the probability of error in the information transmitted over a noisy channel can be reduced to any desired level if the data rate is within a calculable figure called the *channel capacity*. However, since his proof is non-constructive, he left open the question of finding a specific code that can be used on a channel to achieve any desired

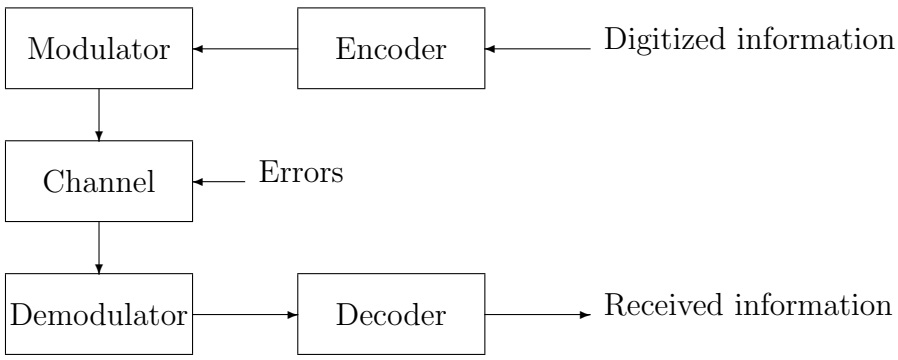

Figure 1.1: The digital communication system

low probability error. Thus, one of the important problems in coding theory is how to construct these codes. In practice, any code needs an efficient decoder. Hence, another significant problem in this field is how to find an efficient decoding algorithm for a specific code, the problem upon which this dissertation will focus. The following sections will review some basic concepts of a communication system incorporating coding. Some existing decoding algorithms are also briefly investigated.

1.1 Digital Communication System

A block diagram of a typical digital communication system incorporating coding is shown in Figure 1.1 [37]. The input of a channel encoder is a stream of digitized information, i.e., a stream of 0's and 1's, and we assume that every digit is equally likely to be a 0 or 1. The channel encoder divides the incoming data stream into sequences of fixed length k. There are 2^k binary sequences of length k, each of which is mapped to a distinct codeword by the encoder, with any codeword a sequence of

binary digits of length n. The mapping is a one-to-one correspondence between the k digits coming in to the encoder and the n digits going out from the encoder. The entire collection of codewords is called a *code*.

The n digits going out from the encoder are sent at discrete intervals to a modulator where they are transformed into specified waveforms (signals) for transmission over the channel.

The channel is a transmission medium that introduces a number of effects such as attenuation, distortion, interference, and noise [37]. These effects make it uncertain whether information will be received correctly.

The demodulator tries to decide on the values of the transmitted signals and pass those decisions to the decoder. If a demodulator quantizes each signal into two levels and decides on the values of the signal in terms of 0 or 1, it is called a *hard-decision demodulator*. If a modulator passes the analog output of filters matched to the signals to the decoder, it is called a *soft-decision demodulator* [37]. The n values coming from the demodulator constitute a *received vector* that will be used to estimate the transmitted codeword by the decoder.

A decoder tries to use some rule to estimate the transmitted codeword on a given received vector. Normally, a decoding rule should be chosen that will minimize error probability.

1.2 Channel Model

A transmission channel can be characterized in terms of the set of input variables, the set of output variables, and the probability of receiving an output element when an input element has been transmitted [21]. Many channel models have been analyzed in [43, 21]. This dissertation considers only the time-discrete memoryless channel,

since it is suitable for the channels to which coding technique is applied.

Definition 1.1 *[21] The time-discrete memoryless channel (TDMC) is a channel specified by an arbitrary input space A, an arbitrary output space B, and for each element a in A , a conditional probability measure on every element b in B that is independent of all other inputs and outputs.*

The channel input is a sequence of elements from A, the channel output is a sequence of elements from B, and each output element depends statistically only on the corresponding input element.

An example of TDMC is the *Additive White Gaussian Noise channel* (AWGN channel). We assume that antipodal signaling is used in the transmission of binary signals over the channel. A 0 is transmitted as $+\sqrt{E}$ and a 1 is transmitted as $-\sqrt{E}$, where E is the signal energy per channel bit. Thus, the input space is $A = \{0, 1\}$ and the output space is $B = \boldsymbol{R}$. When a sequence of input elements $(c_0, c_1, \ldots, c_{n-1})$ is transmitted, the sequence of output elements $(r_0, r_1, \ldots, r_{n-1})$ will be $r_j = (-1)^{c_j}\sqrt{E} + e_j$, $j = 0, 1, \ldots, n-1$, where e_j is a noise sample of a Gaussian process with single-sided noise power per hertz N_0. The variance of e_j is $N_0/2$ and the *signal-to-noise ratio* (SNR) for the channel is $\gamma = E/N_0$.

1.3 Binary Linear Block Code (BLBC)

Most definitions and results in this section are taken from McEliece [29]. Usually the input stream for a channel will consist of binary digits, and the source generates a 0 or 1 with the same probability. In block coding, the input stream is segmented into fixed length blocks, each of which is called a *message*. Each message contains k binary digits. The encoder transfers each message $\boldsymbol{u}$ to a binary n-tuple $\boldsymbol{c}$ with $n > k$

according to some rules. This binary n-tuple $\boldsymbol{c}$ is referred to as a *codeword* of the message $\boldsymbol{u}$, and there is one-to-one correspondence between $\boldsymbol{u}$ and its codeword $\boldsymbol{c}$. The collection of all codewords of messages is called a code $\boldsymbol{C}$. Before a well-known encoding rule that will generate a linear block code is introduced, we give a definition of the *Galois field of order 2* [25].

Definition 1.2 *The Galois field of order 2, denoted by* $\boldsymbol{GF(2)}$*, is a set of two elements 0 and 1, and the operations of modulo-2 addition* $\oplus$ *and modulo-2 multiplication* $\times$*.*

Definition 1.3 *An* (n, k) *binary linear block code is a* k*-dimensional subspace of the* n*-dimensional vector space* $\boldsymbol{V}_n = \{(x_0, x_1, \ldots, x_{n-1}) | \forall x_i \; x_i \in \boldsymbol{GF(2)}\}$*;* n *is called the length of the code,* k *the dimension.*

An (n, k) BLBC can be specified by any set of k linear independent codewords $\boldsymbol{c}_0$,$\boldsymbol{c}_1$,...,$\boldsymbol{c}_{k-1}$. Every codeword in $\boldsymbol{C}$ is one of the 2^k linear combinations $\sum_{i=0}^{k-1} a_i \boldsymbol{c}_i$, where $a_i \in \boldsymbol{GF(2)}$. If we arrange the k codewords into a $k \times n$ matrix $\boldsymbol{G}$, $\boldsymbol{G}$ is called a *generator matrix* for $\boldsymbol{C}$.

It is easy to encode the message by linear block codes. Let $\boldsymbol{u} = (u_0, u_1, \ldots, u_{k-1})$, where $u_i \in \boldsymbol{GF(2)}$. The corresponding codeword of $\boldsymbol{u}$ can be obtained by multiplying $\boldsymbol{G}$ to $\boldsymbol{u}$. That is, $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1}) = \boldsymbol{uG}$. If a BLBC maps u_i to c_i for $0 \leq i \leq k-1$, then the BLBC is called *systematic code*. For a given code $\boldsymbol{C}$ with generator matrix $\boldsymbol{G}$, it is well known that we have a systematic code with generator matrix $\boldsymbol{G}'$ where $\boldsymbol{G}'$ is obtained by permuting the columns of $\boldsymbol{G}$ and by doing some row operations on $\boldsymbol{G}$. In this case, the code generated by $\boldsymbol{G}'$ is an *equivalent code* of that generated by $\boldsymbol{G}$. Notice that, the generator matrix $\boldsymbol{G}'$ of a systematic code has the form of $[\boldsymbol{I}_k \boldsymbol{A}]$, where $\boldsymbol{I}_k$ is the $k \times k$ identity matrix.

An (n, k) linear block code can also be specified by another matrix called the parity-check matrix.

Let $\boldsymbol{C}$ be an (n, k) BLBC. A *parity check* for $\boldsymbol{C}$ is an equation of the form

$$a_0 c_0 \oplus a_1 c_1 \oplus \ldots \oplus a_{n-1} c_{n-1} = 0,$$

which is satisfied for any $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1}) \in \boldsymbol{C}$. The collection of all vectors $\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})$ forms a subspace of $\boldsymbol{V}_n$. It is denoted by $\boldsymbol{C}^{\perp}$ and is called the *dual code* of $\boldsymbol{C}$. The dimension of $\boldsymbol{C}^{\perp}$ is $n - k$ and $\boldsymbol{C}^{\perp}$ is an $(n, n - k)$ BLBC. Any generator matrix of $\boldsymbol{C}^{\perp}$ is a *parity-check matrix* for $\boldsymbol{C}$ and is denoted by $\boldsymbol{H}$. Hence, $\boldsymbol{H}\boldsymbol{c}^T = \boldsymbol{0}$ for any $\boldsymbol{c} \in \boldsymbol{C}$.

Definition 1.4 *The Hamming weight of a codeword $\boldsymbol{c}$, $W_H(\boldsymbol{c})$, is the number of 1's of the codeword. The Hamming distance between two codewords $\boldsymbol{c}$ and $\boldsymbol{c}'$ is defined as $d_H(\boldsymbol{c}, \boldsymbol{c}')$ = the number of components in which $\boldsymbol{c}$ and $\boldsymbol{c}'$ differ.*

From the above definition we observe that $d_H(\boldsymbol{c},\boldsymbol{0}) = W_H(\boldsymbol{c})$. That is, the Hamming weight of a codeword is the Hamming distance between it and the all-zero codeword. For a linear block code, two important properties regarding Hamming distance and Hamming weight are:

1. Let HW be the set of all distinct Hamming weights that codewords of $\boldsymbol{C}$ may have. Furthermore, let $HD(\boldsymbol{c})$ be the set of all distinct Hamming distances between $\boldsymbol{c}$ and any codeword. Then, $HW = HD(\boldsymbol{c})$ for any $\boldsymbol{c} \in \boldsymbol{C}$.

2. If $\boldsymbol{C}$ and $\boldsymbol{C}'$ are equivalent to each other, then the HW for $\boldsymbol{C}$ is the same as that for $\boldsymbol{C}'$.

The smallest nonzero element in HW is referred to as d_{min}, the minimum distance of the code $\boldsymbol{C}$; d_{min} is a very important parameter for a code since it determines the error-correcting capability of the code.

In the next section we will describe how to decode a received vector.

1.4 Soft-Decision Decoding

The premise of soft-decision decoding is to take into consideration the analog output of filters matched to the signals. That is, real numbers are used and associated with every component of the codeword in the decoding procedure. Soft-decision decoding can provide about 2 dB of additional coding gain when compared to hard-decision decoding [12].

In general, an optimal decoding rule of a linear block code is to estimate a codeword $\widehat{\boldsymbol{c}}$ such that $\widehat{\boldsymbol{c}}$ is the most likely transmitted codeword when the received vector $\boldsymbol{r}$ is given. That is, the optimal decoding rule minimizes error probability. Formally, an optimal decoding rule can be formulated as follows:

Let $\boldsymbol{C}$ be the transmitted (n, k) binary linear block code and $\boldsymbol{r}$ be a received vector:

$$\text{set } \widehat{\boldsymbol{c}} = \boldsymbol{c}_\ell \text{ where } \boldsymbol{c}_\ell \in \boldsymbol{C} \text{ and}$$

$$\boldsymbol{Pr}(\boldsymbol{c}_\ell|\boldsymbol{r}) \geq \boldsymbol{Pr}(\boldsymbol{c}|\boldsymbol{r}) \text{ for all } \boldsymbol{c} \in \boldsymbol{C}.$$

If all codewords of $\boldsymbol{C}$ have equal probability of being transmitted, then to maximize $\boldsymbol{Pr}(\boldsymbol{c}|\boldsymbol{r})$ is equivalent to maximizing $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c})$, where $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c})$ is the probability that $\boldsymbol{r}$ is received when $\boldsymbol{c}$ is transmitted, since

$$\boldsymbol{Pr}(\boldsymbol{c}|\boldsymbol{r}) = \frac{\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c})\boldsymbol{Pr}(\boldsymbol{c})}{\boldsymbol{Pr}(\boldsymbol{r})} \ .$$

Definition 1.5 *A maximum-likelihood decoding rule (***MLD** *rule), which minimizes error probability when each codeword is transmitted equiprobably, decodes a received vector* $\boldsymbol{r}$ *to a codeword* $\boldsymbol{c}_\ell \in \boldsymbol{C}$ *such that*

$$\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}_\ell) \geq \boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}) \text{ for all } \boldsymbol{c} \in \boldsymbol{C}.$$

For a time-discrete memoryless channel, the **MLD** rule can be formulated as

$$\text{set } \widehat{\boldsymbol{c}} = \boldsymbol{c}_\ell \text{ where } \boldsymbol{c}_\ell = (c_{\ell 0}, c_{\ell 1}, \ldots, c_{\ell(n-1)}) \in \boldsymbol{C} \text{ and}$$
$$\prod_{j=0}^{n-1} \boldsymbol{Pr}(r_j|c_{\ell j}) \geq \prod_{j=0}^{n-1} \boldsymbol{Pr}(r_j|c_j) \text{ for all } \boldsymbol{c} = (c_0, c_1, \ldots, c_{(n-1)}) \in \boldsymbol{C}.$$

Let $S(\boldsymbol{c},\boldsymbol{c}_\ell) \subseteq \{0, 1, \ldots, n-1\}$ be defined as $j \in S(\boldsymbol{c}, \boldsymbol{c}_\ell)$ iff $c_{\ell j} \neq c_j$. Then the **MLD** rule can be written as

$$\text{set } \hat{\boldsymbol{c}} = \boldsymbol{c}_\ell \text{ where } \boldsymbol{c}_\ell \in \boldsymbol{C} \text{ and}$$
$$\sum_{j \in S(\boldsymbol{c},\boldsymbol{c}_\ell)} \ln \frac{\boldsymbol{Pr}(r_j|c_{\ell j})}{\boldsymbol{Pr}(r_j|c_j)} \geq 0 \text{ for all } \boldsymbol{c} \in \boldsymbol{C}.$$

Following the formulation given in [23], we define the bit log-likelihood ratio of r_i as

$$\phi_i = \ln \frac{\boldsymbol{Pr}(r_i|0)}{\boldsymbol{Pr}(r_i|1)} .$$

Furthermore, let $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_{n-1})$. The absolute value of ϕ_i is called the *reliability* of position i of received vector. By [23, Theorem 5] the **MLD** rule can be written as

$$\text{set } \widehat{\boldsymbol{c}} = \boldsymbol{c}_\ell, \text{ where } \boldsymbol{c}_\ell \in \boldsymbol{C} \text{ and}$$
$$\sum_{j=0}^{n-1} (\phi_j - (-1)^{c_{\ell j}})^2 \leq \sum_{j=0}^{n-1} (\phi_j - (-1)^{c_j})^2 \text{ for all } \boldsymbol{c} \in \boldsymbol{C}. \tag{1.1}$$

Thus, we will say that $\boldsymbol{c}_\ell$ is the "closest" codeword to $\boldsymbol{\phi}$.

Let $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1})$ be the hard-decision of $\boldsymbol{\phi}$. That is

$$\begin{aligned} y_i &= 1 \text{ if } \phi_i < 0; \\ &= 0 \text{ otherwise.} \end{aligned}$$

Furthermore, let $\boldsymbol{s} = \boldsymbol{y}\boldsymbol{H}^T$ be the syndrome of $\boldsymbol{y}$ and let $E(\boldsymbol{s})$ be the collection of all error patterns whose syndrome is $\boldsymbol{s}$. Another useful form of **MLD** rule can be stated as

$$\text{set } \widehat{\boldsymbol{c}} = \boldsymbol{y} \oplus \boldsymbol{e}_\ell, \text{ where } \boldsymbol{e}_\ell \in E(\boldsymbol{s}) \text{ and}$$

$$\sum_{j=0}^{n-1} e_{\ell j}|\phi_j| \leq \sum_{j=0}^{n-1} e_j|\phi_j| \quad \text{for all } \boldsymbol{e} \in E(\boldsymbol{s}). \tag{1.2}$$

One straightforward way to implement the **MLD** rule is to calculate $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}) = \prod_{j=0}^{n-1} \boldsymbol{Pr}(r_j|c_j)$ for every codeword in $\boldsymbol{C}$ and select the codeword that maximizes it. In practice this can be done only for those codes with a small number of codewords, that is, low rate codes or middle-to-high rate codes with short block length. Indeed, the **MLD** soft-decision decoding problem of linear codes has been shown to be an NP-hard problem [7, 39, 17]. That means that it is extremely difficult to discover a polynomial time algorithm for **MLD** soft-decision decoding of linear codes. Thus, some suboptimal soft-decision decoding algorithms trade performance for decoding time and space complexities. In Section 1.6 we will describe existing decoding algorithms that implement the **MLD** rule and some that are suboptimal. A decoding algorithm which implements the **MLD** rule will be referred as an *optimal decoding algorithm.*

1.5 Trellis and Code Tree

In this section we demonstrate how to convert a decoding problem to a graph-search problem on a graph in which a path represents a codeword in code $\boldsymbol{C}$.

We now give a short description of a trellis [1] for code $\boldsymbol{C}$ where the search will be performed. Let $\boldsymbol{H}$ be a parity-check matrix of $\boldsymbol{C}$, and let $\boldsymbol{h}_i$, $0 \leq i \leq n-1$ be the column vectors of $\boldsymbol{H}$. Furthermore, let $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ be a codeword of $\boldsymbol{C}$. With respect to this codeword, we recursively define the states $\boldsymbol{s}_t$, $-1 \leq t \leq n-1$, as

$$\boldsymbol{s}_{-1} = \boldsymbol{0}$$

and

$$\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + c_t \boldsymbol{h}_t = \sum_{i=0}^{t} c_i \boldsymbol{h}_i, \quad 0 \leq t \leq n-1.$$

Clearly, $\boldsymbol{s}_{n-1} = \boldsymbol{0}$ for all codewords of $\boldsymbol{C}$. The above recursive equation can be used to draw a trellis diagram. In this trellis, $\boldsymbol{s}_{-1} = \boldsymbol{0}$ identifies the start node at level -1; $\boldsymbol{s}_{n-1} = \boldsymbol{0}$ identifies the goal node at level $n-1$; and each state $\boldsymbol{s}_t, 0 \leq t \leq n-2$ identifies a node at level t. Furthermore, each transition (arc) is labeled with the appropriate codeword bit c_t. Thus, there is a one-to-one correspondence between the codewords of $\boldsymbol{C}$ and the sequences of labels encountered when traversing a path in the trellis from the start node to the goal node. Note that the trellis defined here corresponds to the expurgated trellis of [44].

The best way to understand the construction of a trellis of code $\boldsymbol{C}$ is through an example.

Example 1.1 *Let*

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

be a generator matrix of $\boldsymbol{C}$ *and let*

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

be a parity-check matrix of $\boldsymbol{C}$*. A trellis of* $\boldsymbol{C}$ *is shown in Figure 1.2.*

A code tree is another way to represent every codeword of an (n, k) code $\boldsymbol{C}$ as a path through a tree containing $n+1$ levels . The code tree can be treated as an expanded version of the trellis, where every path is totally distinct from every other

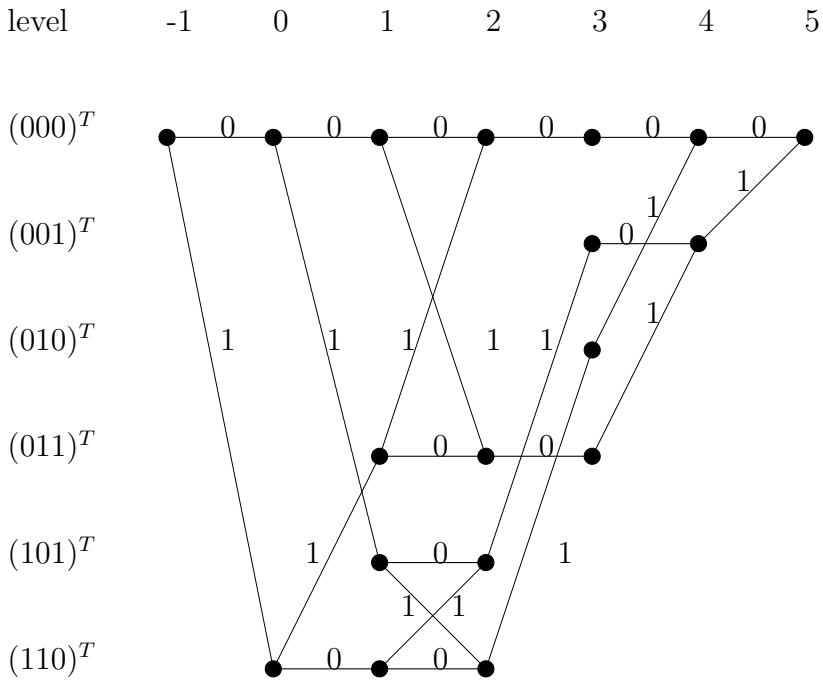

Figure 1.2: An example of a trellis of the $(6, 3)$ code

path. The leftmost node is called the *start node*. There are two branches, labeled by 0 and 1, respectively, that leave each node at the first k levels. After the k levels, there is only one branch leaving each node. The 2^k rightmost nodes are called *goal nodes*.

Next, we describe how to determine the sequence of labels encountered when traversing a path from a node at level k to a goal node.

Let $\boldsymbol{G}$ be a generating matrix of $\boldsymbol{C}$ whose first k columns form the $k \times k$ identity matrix. Furthermore, let $c_0, c_1, \ldots, c_{k-1}$ be the sequence of labels encountered when traversing a path from the start node to a node m at level $k-1$. Then $c_k, c_{k+1}, \ldots, c_{n-1}$, the sequence of labels encountered when traversing a path from node m to a goal node, can be obtained as follows:

$$(c_0, c_1, \ldots, c_k, c_{k+1}, \ldots, c_{n-1}) = (c_0, c_1, \ldots, c_{k-1})\boldsymbol{G}.$$

An example of a code tree follows:

Example 1.2 *Let*

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

be a generator matrix of $\boldsymbol{C}$*. The code tree for this code is shown in Figure 1.3.*

Now we describe how to convert a decoding problem to a graph-search problem.

From Inequality 1.1, we want to find a codeword $\boldsymbol{c}_\ell$ such that $\sum_{j=0}^{n-1}(\phi_j - (-1)^{c_{\ell j}})^2$ is the minimum among all the codewords. Thus, if we properly specify the arc costs on a trellis or a code tree, the **MLD** rule can be written as follows:

> Find a path from the start node to a goal node such that the cost of the path is minimum among all the paths from the start node to a goal node,

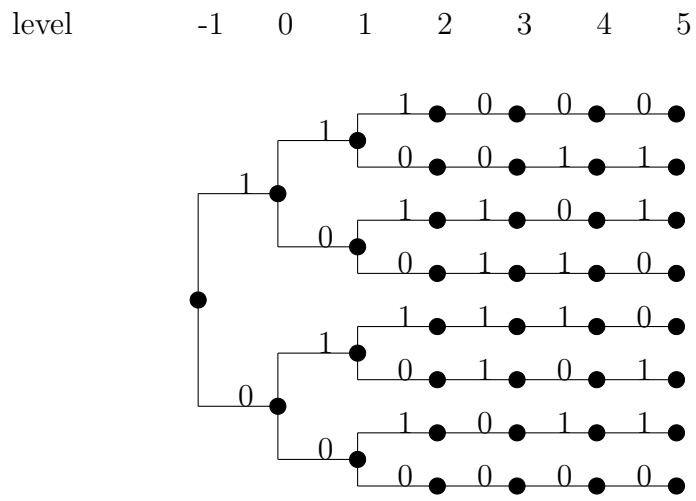

Figure 1.3: An example of a code tree of the $(6, 3)$ code

where a cost of a path is the summation of the cost of arcs in the path. Such a path is called an *optimal path*.

To explain in more detail, in the trellis of $\boldsymbol{C}$ the cost of the arc from $\boldsymbol{s}_{t-1}$ to $\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + c_t\boldsymbol{h}_t$ is assigned the value $(\phi_t - (-1)^{c_t})^2$. In the code tree of $\boldsymbol{C}$, the cost of the arc from a node at level $t-1$ to a node at level t is assigned the value $(\phi_t - (-1)^{c_t})^2$, where c_t is the label of the arc. Thus the solution of the decoding problem is converted into finding a path from the start node to a goal node in the trellis (code tree), that is, a codeword $\boldsymbol{c}= (c_0, c_1, \ldots, c_{n-1})$ such that $\sum_{j=0}^{n-1}(\phi_j - (-1)^{c_j})^2$ is minimum among all paths from the start node to a goal node.

1.6 Soft-Decision Decoding Algorithms

The goal of soft-decision decoding algorithms of linear block codes is to select the codeword closest to $\boldsymbol{\phi}$. There are two methods to construct a set of codewords from which the decoding algorithm can select such a codeword [12].

- Before the search procedure begins, a fixed number of codewords are selected. The algorithm then searches through all those codewords in order to determine the closest codeword to $\boldsymbol{\phi}$. The advantage of this type of decoding algorithm is that the computational effort to minimize the size of those codewords needed to be stored can be done in advance and then only those codewords will be stored in the decoder. Unfortunately, Bruck and Naor proved that, in general, it is unlikely that the size of those codewords will be bounded by a polynomial of k or $n-k$ [10]. We will discuss more algorithms of the next type whose time and space complexities are random variables of the received vector.

- In contrast with the above algorithms, decoding algorithms of this type generate the next codeword to be examined by observing the effect of the previous generated codeword or by utilizing any reliable information that may be available. Although the effort of decoding the next codeword to be examined of this type of decoding algorithm may be greater, the number of codewords examined can sometimes be less, especially for a high SNR channel. Unfortunately, it is difficult to analyze the performance of this type of algorithm since the number of codewords examined will be a random variable.

In 1978, Wolf proposed a block code version of the Viterbi algorithm to be used to decode linear block codes [44]. The decoding problem was first converted to a graph-search problem on a trellis derived from a parity-check matrix of the code.

The **MLD** rule could then be implemented by applying the Viterbi algorithm [42]. His decoding procedure for binary linear block codes is as follows.

Let $\boldsymbol{C}$ be an (n, k) linear block code with parity check-matrix $\boldsymbol{H}$. First, construct a trellis for $\boldsymbol{C}$ and then apply the Viterbi algorithm to the trellis. How to construct a trellis for $\boldsymbol{C}$ has been described in Section 1.5, and we will therefore concentrate here on how the Viterbi algorithm works.

Basically, the Viterbi algorithm is a dynamic programming algorithm that is executed as follows:

1. For every node at level ℓ, calculate the sum of arc costs of every path from the start node to this node.

2. Keep the path with the lowest sum of arc costs and discard the others.

3. Apply the same process to any node at every level until the goal node at level $n - 1$.

4. The surviving path from the start node to the goal node is an optimal path.

In Step 1 of the above procedure, the calculation of the sum of the arc costs of a path from the start node to another node can be obtained by adding the arc cost to a node in the previous level. Thus, the Viterbi algorithm can be performed level-by-level to minimize the number of computations needed. Consequently, Wolf's algorithm uses a breadth-first search strategy to accomplish the search of an optimal path through a trellis. Therefore, the time and space complexities of this algorithm are of $O(n \times min(2^k, 2^{n-k}))$ [14], since it traverses the entire trellis.

One way to improve Wolf's algorithm is to reduce the number of states in a trellis for $\boldsymbol{C}$ [20, 30, 5]. In [20], Forney gave a general procedure to reduce the number of states in a trellis.

Forney's method is as follows. Let $\boldsymbol{C}$ be an (n, k) linear block code. Let $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ be an n-tuple over $\boldsymbol{GF}(2)$, and let $\boldsymbol{c}_p^{(i)} = (c_0, c_1, \ldots, c_i)$ and $\boldsymbol{c}_f^{(i)} = (c_{i+1}, c_{i+2}, \ldots, c_{n-1}), -1 \leq i \leq n-1$. For code $\boldsymbol{C}$, let $\boldsymbol{C}_p^{(i)}$ be the linear code that consists of $\boldsymbol{c}_p^{(i)}$ for all $\boldsymbol{c} \in \boldsymbol{C}$ such that $\boldsymbol{c}_f^{(i)} = (0, 0, \ldots, 0)$. Similarly, let $\boldsymbol{C}_f^{(i)}$ be the linear code that consists of $\boldsymbol{c}_f^{(i)}$ for all $\boldsymbol{c} \in \boldsymbol{C}$ such that $\boldsymbol{c}_p^{(i)} = (0, 0, \ldots, 0)$. Denote the dimensions of $\boldsymbol{c}_p^{(i)}$ and $\boldsymbol{c}_f^{(i)}$ by k_{pi} and k_{fi}, respectively. Forney showed that the number of states at the i^{th} level is 2^{s_i}, where $s_i = k - k_{pi} - k_{fi}$. The minimal trellis size index, denoted as $s = \max_i(s_i)$, depends on the order of the columns in the generator matrix $\boldsymbol{G}$ or in the parity-check matrix $\boldsymbol{H}$. However, for long codes, it is difficult to find an order from which the minimal trellis size index is the smallest among all possible orders, since it depends on the code structure [5]. Algorithms to find such orders for the Golay code and the Reed-Muller codes are described in [20].

Another decoding algorithm that performs the **MLD** rule and searches through a trellis or code tree was proposed by Battail [2]. This algorithm can be treated as a branch-and-bound search algorithm that searches through a graph that is a trellis or a code tree for a code $\boldsymbol{C}^*$ equivalent to code $\boldsymbol{C}$.

Now we describe Battail's algorithm in more detail. Let $\boldsymbol{C}$ be an (n, k) linear block code with generator matrix $\boldsymbol{G}$. $\boldsymbol{C}^*$ is obtained from $\boldsymbol{C}$ by permuting the positions of codewords of $\boldsymbol{C}$ in such a way that the first k positions of codewords in $\boldsymbol{C}^*$ correspond to the "most reliable linearly independent" positions in the received vector $\boldsymbol{\phi}$. $\boldsymbol{G}^*$ is a generator matrix of $\boldsymbol{C}^*$ whose first k columns form the $k \times k$ identity matrix. Let $\boldsymbol{\phi}^* = (\phi_0^*, \phi_1^*, \ldots, \phi_{n-1}^*)$ be used as the "received vector." It is obtained by permuting the positions of $\boldsymbol{\phi}$ in the same manner in which the columns of $\boldsymbol{G}$ can be permuted to obtain $\boldsymbol{G}^*$.

Let us define

$$Z(\boldsymbol{\phi}^*, \boldsymbol{c}^m) = Z_{\boldsymbol{u}}(\boldsymbol{\phi}^*, \boldsymbol{c}^m) + Z_{\boldsymbol{s}}(\boldsymbol{\phi}^*, \boldsymbol{c}^m),$$

where

$$Z_{\boldsymbol{u}}(\boldsymbol{\phi}^*,\boldsymbol{c}^m) = \sum_{j=0}^{k-1} \left(\phi_j^* - (-1)^{c_j^m}\right)^2$$

and

$$Z_{\boldsymbol{s}}(\boldsymbol{\phi}^*,\boldsymbol{c}^m) = \sum_{j=k}^{n-1} \left(\phi_j^* - (-1)^{c_j^m}\right)^2.$$

Then from Inequality 1.1, optimal decoding consists of determining $\widehat{m}$ such that

$$Z(\boldsymbol{\phi}^*,\boldsymbol{c}^{\widehat{m}}) \leq Z(\boldsymbol{\phi}^*,\boldsymbol{c}^m) \text{ for any } \boldsymbol{c}^m \in \boldsymbol{C}^*.$$

The decoding procedure is now as follows.

Let $\boldsymbol{y}^* = (y_0^*, y_1^*, \ldots, y_{n-1}^*)$ be the hard-decision of $\boldsymbol{\phi}^*$. First calculate $\boldsymbol{c}^1 = (y_0^*, y_1^*, \ldots, y_{k-1}^*) \times \boldsymbol{G}^*$ and set a threshold $\sigma = Z(\boldsymbol{\phi}^*,\boldsymbol{c}^1)$, where σ is an upper bound on the cost of an optimal path. Then other information vectors $\boldsymbol{u}$, corresponding to vectors $\boldsymbol{c}$ that belong to the code $\boldsymbol{C}^*$, are tried in the order of non-decreasing $Z_{\boldsymbol{u}}(\boldsymbol{\phi}^*,\boldsymbol{c})$. $Z(\boldsymbol{\phi}^*,\boldsymbol{c})$, which corresponds to vectors $\boldsymbol{c}$, is then computed. If $Z(\boldsymbol{\phi}^*,\boldsymbol{c})$ is found for some $\boldsymbol{c}$ less than the current σ, then $Z(\boldsymbol{\phi}^*,\boldsymbol{c})$ becomes the new threshold, and $\boldsymbol{c}$ will be the best solution found so far. Continue the decoding procedure until $Z_{\boldsymbol{u}}(\boldsymbol{\phi}^*,\boldsymbol{c})$ exceeds the current threshold. The author did not address clearly how to order the paths according to $Z_{\boldsymbol{u}}(\boldsymbol{\phi}^*,\boldsymbol{c})$ and this could affect the efficiency of the algorithm.

Another way to perform the **MLD** rule is to find an error pattern that satisfies Inequality 1.2 [16, 36, 35, 24, 26]. A general decoding procedure for these algorithms is as follows [26].

Let $\boldsymbol{y}$ be the hard-decision of $\boldsymbol{\phi}$. Determine the syndrome $\boldsymbol{s}$ of $\boldsymbol{y}$. If $\boldsymbol{s} = 0$, then output $\widehat{\boldsymbol{c}} = \boldsymbol{y}$. Otherwise, find an error pattern $\boldsymbol{e}$ whose syndrome is $\boldsymbol{s}$ such that $\sum_{j=0}^{n-1} e_j|\phi_j|$ is minimal. Output $\widehat{\boldsymbol{c}} = \boldsymbol{y} \oplus \boldsymbol{e}$. Of these algorithms, [24] will be described in more detail, but we first need to describe Chase's algorithms [11].

The main idea of Chase's algorithms is to select a set of test patterns that are closed to $\boldsymbol{y}$, and for each of them use a hard-decision decoder to obtain an error pattern whose syndrome is the same as that of $\boldsymbol{y}$. Let $\boldsymbol{C}$ be an (n, k) code with minimum distance d_{min}. The basic procedure of Chase is as follows:

1. Find $\boldsymbol{y}$, the hard-decision of $\boldsymbol{\phi}$.

2. Introduce some patterns of errors, $\boldsymbol{t}$, to generate test pattern $\boldsymbol{t}_s = \boldsymbol{t} + \boldsymbol{y}$.

3. Decode each $\boldsymbol{t}_s$ to $\boldsymbol{c}$ using a hard-decision decoder if possible.

4. Compute $\sum_{j=0}^{n-1} (\phi_j - (-1)^{c_j})^2$ for all $\boldsymbol{c}$ obtained in Step 3 and select the $\boldsymbol{c}$ such that $\sum_{j=0}^{n-1} (\phi_j - (-1)^{c_j})^2$ is minimal.

Chase proposed three methods to introduce $\boldsymbol{t}$. The first generates $\boldsymbol{t}$, the all-zero pattern and all error patterns with Hamming weight $\lfloor d_{min}/2 \rfloor$; the second generates $\boldsymbol{t}$, all $2^{\lfloor d_{min}/2 \rfloor}$ combinations of values in the $\lfloor d_{min}/2 \rfloor$-least reliable positions and zeros in all other positions; the third generates $\boldsymbol{t}$, all the vector that have 1's in the i-least reliable positions and zeros elsewhere (for d_{min} odd, $i = 0, 2, 4, \ldots, d_{min} - 1$ and for d_{min} even, $i = 0, 1, 3, \ldots, d_{min} - 1$).

The first method will give the best performance, since it generates the largest number of error patterns. However, as this number is too large for most practical systems, this method can only be implemented for short codes or codes with small d_{min}. It is easy to see that all the methods proposed by Chase are suboptimal soft-decision decoding algorithms. In [24], the authors give a way to calculate the proper Hamming weight for the error patterns in Chase's second method in order to perform the **MLD** rule.

Suppose there is a t error-correcting hard-decision decoder available for code $\boldsymbol{C}$. Let $\boldsymbol{c}$ be the closest codeword to $\boldsymbol{\phi}$ found so far by the algorithm and $\boldsymbol{e} = \boldsymbol{c} \oplus \boldsymbol{y}$.

Furthermore, let $\boldsymbol{u}_o = (u_{00}, u_{01}, \ldots, u_{0(n-1)})$, the vector obtained by permuting the positions of $\boldsymbol{\phi}$ with non-decreasing reliability. Let $S(\boldsymbol{c}) = \{\phi_i |\ y_i = c_i, 0 \leq i \leq n-1\}$ and $\boldsymbol{v} = (v_0, v_1, \ldots . v_{n-W_H(\boldsymbol{e})-1})$ contain all the distinct elements in $S(\boldsymbol{c})$ as its components such that $|v_0| \leq |v_1| \leq \ldots \leq |v_{n-W_H(\boldsymbol{e})-1}|$. First, we calculate the smallest $j \in \{0, 1, 2, \ldots, n-1\}$ that satisfies the inequality

$$\sum_{i=0}^{n-1} e_i |\phi_i| < \sum_{i=0}^{d_{min}-W_H(\boldsymbol{e})-t-2} |v_i| + \sum_{i=0}^{t} |u_{i+j+1}|.$$

Then, to check if $\boldsymbol{c}$ is the closest codeword to $\boldsymbol{\phi}$, we need to check error patterns that are all 2^{j+1} combinations of values in the $j+1$-least reliable positions and the zeros in all other positions.

1.7 Synopsis of the Dissertation

This dissertation is organized in the following way. A detailed description of algorithm A^* is given in Chapter 2 and some important properties of algorithm A^* are also discussed.

In Chapter 3 we present a class of novel and efficient maximum-likelihood soft-decision decoding algorithms for linear block codes. The approach used here converts the decoding problem into a search problem through a graph that is a trellis or code tree for an equivalent code of the transmitted code. Algorithm A^* is employed to search through this graph. This search is guided by an evaluation function f defined to take advantage of the information provided by the received vector and the inherent properties of the transmitted code.

In Chapter 4 we investigate how to determine the computational performance of our decoding algorithms. Since the number of nodes visited and the number of codewords tried by our decoding algorithms are random variables, their probability

distributions are given.

In Chapter 5 we present a class of suboptimal soft-decision decoding algorithms that use a generalized algorithm A^*. In these algorithms we limit the number of nodes stored using two criteria.

In Chapter 6 we describe several conclusions and recommend some further research topics.

Chapter 2

Algorithm A^*

As mentioned in Section 1.5, when the decoding problem is converted into a graph-search problem, we are interested in finding a path from the start node representing the initial condition to a goal node that represents the termination condition. This path leads us to construct a codeword that maximizes $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c})$, where $\boldsymbol{c} \in \boldsymbol{C}$.

Thus, the decoding problem is mapped to a more general graph-search problem. In this graph each arc is assigned a cost, and the cost of a path is the sum of the costs of the arcs connecting the nodes in this path. The problem is how to find an optimal path from the start node to a goal node, that is, a path with minimum (maximum) cost. Algorithm A^*, widely used in Artificial Intelligence, is an efficient procedure for finding an optimal path if one exists in a graph. We apply algorithm A^* only to finite graphs. In the following sections we will discuss algorithm A^*. Most definitions and results in this chapter are taken from Nilsson [31].

2.1 General Graph-Search Procedure

A *successor operator*, when applied to a node m, gives all the immediate successors of node m. We call this process of applying a successor operator to a node *expanding* the node. In order to describe algorithm A^* more easily, we first give a general graph-search procedure as presented in [31]:

Procedure **GRAPHSEARCH**

1. Create a *search graph*, $\mathcal{G}$, consisting solely of the start node, m_s. Put m_s on a list called *OPEN*.

2. Create a list called *CLOSED* that is initially empty.

3. LOOP: if *OPEN* is empty, exit with failure.

4. Select the first node on *OPEN*, remove it from *OPEN*, and put it on *CLOSED*. Call this node m.

5. If m is a goal node, exit successfully with the solution obtained by tracing a path along the pointers from m to m_s in $\mathcal{G}$. (Pointers are established in Step 7.)

6. Expand node m, generating the set, M, of its successors that are not ancestors of m. Install these members of M as successors of m in $\mathcal{G}$.

7. Establish a pointer to m from those members of M that were not already in $\mathcal{G}$ (i.e., not already on either *OPEN* or *CLOSED*). Add these members of M to *OPEN*.

   For each member of M that was already on *OPEN* or *CLOSED*, decide whether or not to redirect its pointer to m. For each member of M already on *CLOSED*, decide for each of its descendants in $\mathcal{G}$ whether or not to redirect its pointer.

8. Reorder the list *OPEN*, either according to some arbitrary scheme or according to heuristic merit.

9. Go LOOP.

This procedure maintains two lists of nodes of the given graph, namely, list CLOSED and list OPEN. List CLOSED contains the set of nodes that were expanded. List OPEN contains the set of nodes that were visited, but not expanded.

If the graph being searched is not a tree, it is possible that some of the elements of set M have already been visited—that is, they are already on list OPEN or list CLOSED. The problem of determining whether a newly generated node is on these lists can be computationally very expensive. For this reason we may decide to avoid making this test, in which case the search tree may contain several repeated nodes. These node repetitions lead to redundant successor computations and there is a trade-off between the computation cost for testing for repeated nodes and the computation cost for generating a larger search tree. In steps 6 and 7 of procedure **GRAPHSEARCH**, testing for repeated nodes is performed.

In an uninformed search procedure, no heuristic information from the problem has been used in reordering the list OPEN in Step 8. In this case, the two well-known search methods are the breadth-first and depth-first; however, these methods are exhaustive in nature, and thus in practice are applicable only to graphs with small numbers of nodes and paths.

In many cases it is possible to use some inherent properties of a problem to help reduce the search. The search procedure using this information is called a *heuristic search method*. In many situations it is possible to specify heuristics that reduce considerably the search effort without compromising the optimality of the solution.

2.2 The Optimality of Algorithm A^*

One well-known heuristic search method that guarantees to find the optimal solution if one exists is the algorithm A^* [31]. Algorithm A^* uses a cost function called *evaluation function* f to guide the search through the graph. This function f is computed for every node added to list OPEN in Step 7 of the procedure **GRAPHSEARCH**. In Step 8 of this procedure we reorder the list OPEN according to the value of the function f. From now on, in order to simplify the description of algorithm A^*, we assume that an optimal path is one that minimizes the cost function.

Definition 2.1 *For every node m, the evaluation function f is a function whose value $f(m)$ at node m estimates the cost of the minimum cost path that goes through node m; $f(m)$ is computed as*

$$f(m) = g(m) + h(m),$$

where $g(m)$ estimates the cost of the minimum cost path from start node m_s to node m, and $h(m)$ estimates the cost of the minimum cost path from node m to a goal node. Function h is called the heuristic function.

In algorithm A^*, the next node to be expanded is the one with the smallest value of f on the list OPEN since this node imposes the fewest severe constraints.

Definition 2.2 *Let f^* be a function such that $f^*(m)$ at any node m is the actual cost of a minimum cost path that goes through node m. Analogously,*

$$f^*(m) = g^*(m) + h^*(m),$$

where $g^*(m)$ is the actual cost of a minimum cost path from start node m_s to node m, and $h^*(m)$ is the actual cost of a minimum cost path from node m to a goal node.

An obvious choice for $g(m)$ is the cost of the path in the search tree from start node m_s to node m, given by summing all the arc costs encountered while constructing the minimum cost path from start node m_s to node m. Note that this path is the lowest cost path from start node m_s to node m found so far by the algorithm. The value of $g(m)$ may decrease if the search tree is altered in Step 7 of procedure **GRAPHSEARCH**. From now on we assume that function g is calculated in this way. In this case, $g(m) \geq g^*(m)$ for every node m of the graph. Furthermore, if $h(m) = 0$ for any node m, then algorithm A^* becomes a version of Dijkstra's algorithm [15].

In order to guarantee that algorithm A^* finds an optimal path if one exists, we impose the following condition on the heuristic function h.

Condition 1 *For every node m of the graph, $h(m) \leq h^*(m)$.*

Thus, all the function h used in algorithm A^* must satisfy the above condition. We now give the main theorem of algorithm A^*.

Theorem 2.1 *Algorithm A* will find an optimal path if one exists.*

The proof of Theorem 2.1 will be found in Appendix A.1.

Note that in order to define $h(m) \leq h^*(m)$, we use the properties of the problem. Next, we give a theorem that is useful for analyzing the behavior of algorithm A^*.

Theorem 2.2 *For any node m selected for expansion by algorithm A*, $f(m) \leq f^*(m_s)$.*

The proof of Theorem 2.2 will be found in Appendix A.2.

When we have more than one function h, the following theorem indicates which is the most efficient.

Theorem 2.3 *Let two evaluation functions* $f_1(m) = g_1(m) + h_1(m)$ *and* $f_2(m) = g_2(m) + h_2(m)$ *satisfy* $h_1(m) < h_2(m) \leq h^*(m)$ *for every non-goal node* m*. Algorithm A*, using evaluation function* f_2*, will never expand more nodes than algorithm A* using evaluation function* f_1*.*

The proof of Theorem 2.3 is given in Appendix A.3.

Furthermore, if two compared algorithms use the same tie-breaking rule that is independent of the values of g and h, then the above results hold when $h_1(m) \leq h_2(m) \leq h^*(m)$ is satisfied for every node m [32]. Thus, if $h(m) \geq 0$ for any node m, then algorithm A^*, using this function h, will never expand more nodes than the above version of Dijkstra's algorithm.

Another theorem to be used in our decoding algorithm to reduce the size of list OPEN follows.

Theorem 2.4 *Algorithm A* still finds an optimal path (if one exists) if it removes from list OPEN any node* m *for which* $f(m) > UB$*, where* UB *is an upper bound on the cost of an optimal path.*

The proof of Theorem 2.4 is given in Appendix A.4.

We next introduce an important restriction that can be imposed on function h to improve the efficiency of algorithm A^*.

2.3 The Monotone Restriction on Algorithm A*

The monotone restriction is a reasonable restriction that when imposed on function h can substantially decrease the computation time and storage of algorithm A^*.

Definition 2.3 *Function h is said to satisfy the monotone restriction if and only if for all nodes m_i and m_j, such that node m_j is a successor of node m_i,*

$$h(m_i) - h(m_j) \leq c(m_i, m_j),$$

where $c(m_i, m_j)$ is the arc cost between node m_i and node m_j.

Theorem 2.5 *If the monotone restriction is satisfied, then algorithm A* has already found a minimum cost path from the start node to the node it selects to expand.*

The proof of Theorem 2.5 is given in Appendix A.5.

An important consequence of the above theorem is that when the monotone restriction is satisfied, algorithm A^* does not need to update the minimum cost path from the start node to any node that is already on list CLOSED. Furthermore, when expanding node m, it does not need to update the minimum cost path from the start node to any descendant of an immediate successor of node m that is already on list CLOSED. Another useful theorem regarding monotone restriction is stated below.

Theorem 2.6 *Assume that the monotone restriction is satisfied. If node m_i is selected for expansion, then $f(m_i) \leq f(m_j)$, where m_j is an immediate successor of node m_i, which is not on list CLOSED.*

The proof of Theorem 2.6 is given in Appendix A.6.

From the description of algorithm A^*, it is clear that the most important factor in its efficiency is the selection of the heuristic function h and, consequently, the evaluation function f. Furthermore, Algorithm A^* can be considered as a branch-and-bound type algorithm. In general, it is difficult to give any idea of how well a branch-and-bound algorithm will perform on a given problem. Nevertheless, the technique is sufficiently powerful that it is often used in practical applications [9]. In the next section we will describe how to apply algorithm A^* to the decoding problem.

Chapter 3

Maximum-Likelihood Decoding Algorithm

In this chapter we present a class of novel and efficient maximum-likelihood soft-decision decoding algorithms for linear block codes. The approach used here converts the decoding problem into a search problem through a graph that is a trellis or code tree for an equivalent code of the transmitted code. Algorithm A^*, which is described in Chapter 2, is employed to search through this graph. This search is guided by an evaluation function f defined to take advantage of the information provided by the received vector and the inherent properties of the transmitted code. This function f is used to reduce drastically the search space and to make the decoding efforts of these decoding algorithms adaptable to the noise level.

For ease of explanation we will assume from now on that the received vector is $\boldsymbol{\phi}$ instead of $\boldsymbol{r}$.

3.1 Equivalent Code of the Transmitted Code

Our decoding algorithms, guided by evaluation functions f, search through a graph that is a trellis or a code tree for a code $\boldsymbol{C}^*$, which is equivalent to code $\boldsymbol{C}$. $\boldsymbol{C}^*$ is obtained from $\boldsymbol{C}$ by permuting the positions of codewords of $\boldsymbol{C}$ in such a way that the first k positions of codewords in $\boldsymbol{C}^*$ correspond to the "most reliable linearly independent" positions in the received vector $\boldsymbol{\phi}$. In Appendix B.1 we give an algorithm to obtain $\boldsymbol{G}^*$ from $\boldsymbol{G}$. $\boldsymbol{G}^*$ is a generator matrix of $\boldsymbol{C}^*$ whose first k columns form the $k \times k$ identity matrix. The time complexity of this algorithm is also discussed in this appendix.

In our decoding algorithms the vector $\boldsymbol{\phi}^* = (\phi_0^*, \phi_1^*, \ldots, \phi_{n-1}^*)$ is used as the "received vector." It is obtained by permuting the positions of $\boldsymbol{\phi}$ in the same manner in which the columns of $\boldsymbol{G}$ can be permuted to obtain $\boldsymbol{G}^*$. Furthermore, the description of decoding algorithms in the following sections is based only on the trellis. The description based on the code tree is similar to that of the trellis.

3.2 Evaluation Function

As we pointed out in Section 2.3, the selection of evaluation function f is of the utmost importance, since it determines the search effort of algorithm A^*. We now describe the function f we use in our decoding algorithms.

In order to define function f, we need first to specify the arc costs. As mentioned in Section 1.5, in the trellis of $\boldsymbol{C}^*$, the cost of the arc from $\boldsymbol{s}_{t-1}$ to $\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + c_t^* \boldsymbol{h}_t^*$ is assigned the value $(\phi_t^* - (-1)^{c_t^*})^2$. Thus the solution of the decoding problem is converted into finding a path from the start node to the goal node, that is, a codeword $\boldsymbol{c}^* = (c_0^*, c_1^*, \ldots, c_{n-1}^*)$ such that $\sum_{i=0}^{n-1} (\phi_i^* - (-1)^{c_i^*})^2$ is minimum among all paths from

the start node to the goal node.

Now we define function f for every node m in the trellis as

$$f(m) = g(m) + h(m).$$

As noted in Section 2.2, $g(m)$ is the lowest cost path from the start node to node m found so far by the algorithm, where the cost of a path from the start node to node m is obtained by summing all the arc costs encountered while constructing this path.

We now define a class of heuristic functions. Furthermore, if a function h belongs to this class it will satisfy $h(m) \leq h^*(m)$ for every node m. Recall that $h^*(m)$ is the cost of a minimum cost path from node m to the goal node. In order to define a function h that is a "good" estimator of h^*, we must use properties of the linear block code that are invariant under every permutation of the positions of the codewords.

As in Section 1.3, let $HW = \{w_i | 0 \leq i \leq I\}$ be the set of all distinct Hamming weights that codewords of $\boldsymbol{C}$ may have. Furthermore, assume $w_0 < w_1 < \cdots < w_I$. Thus, from the arguments in Section 1.3, HW for $\boldsymbol{C}^*$ is the same as that for $\boldsymbol{C}$. Moreover, the Hamming distance between any two codewords of $\boldsymbol{C}^*$ must belong to HW. Our heuristic functions are defined to take into consideration this fact and the linear property of $\boldsymbol{C}^*$.

Let S_{C^*} be a given subset of $\boldsymbol{C}^*$, and let $P_i(S_{C^*})$ be the set that contains all the subsets of S_{C^*} of cardinality i, $0 \leq i \leq |S_{C^*}|$, where $|S_{C^*}|$ is the cardinality of S_{C^*}. For a given S_{C^*} we now define our heuristic function, $h^{(i)}$, of order $i, 0 \leq i \leq |S_{C^*}|$.

1. For nodes at level $\ell, -1 \leq \ell < k-1$:

   Let m be a node at level ℓ, and let $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell$ be the labels of the lowest cost path $\boldsymbol{P}'_m$ from the start node to node m found so far by the algorithm.

   If $S_{C^*} = \emptyset$, then $h^{(0)}(m) = 0$. Otherwise, let $Y_i \in P_i(S_{C^*})$, and we now construct the set, $T(m, Y_i)$, of all binary n-tuples $\boldsymbol{v}$ such that their first $\ell + 1$ entries are

the labels of $\boldsymbol{P}'_m$ and $d_H(\boldsymbol{v}, \boldsymbol{c}^*) \in HW$ for all $\boldsymbol{c}^* \in Y_i$. That is,

$$\begin{aligned} T(m, Y_i) &= \{\boldsymbol{v} | \boldsymbol{v} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{\ell+1}, \ldots, v_{n-1}) \text{ and} \\ &\quad \forall \boldsymbol{c}^* \in Y_i, d_H(\boldsymbol{v}, \boldsymbol{c}^*) \in HW\}. \end{aligned}$$

Note that $T(m, Y_i) \neq \emptyset$. This can easily be seen by considering the binary k-tuple $\boldsymbol{u} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, 0, \ldots, 0)$ and noting that $\boldsymbol{uG}^* \in T(m, Y_i)$.

Finally, we define $h^{(i)}$ as

$$h^{(i)}(m) = \max_{Y_i \in P_i(S_{C^*})} \left\{ \min_{\boldsymbol{v} \in T(m, Y_i)} \left\{ \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2 \right\} \right\}.$$

2. For nodes at level $\ell, k-1 \leq \ell < n$:

   Because of the linear property of $\boldsymbol{C}^*$ and the fact that the first k columns of $\boldsymbol{G}^*$ are linearly independent, there is only one path from any node at level $k-1$ to the goal node. Furthermore, we can easily determine the labels $v_k^*, v_{k+1}^*, \ldots, v_{n-1}^*$ of this path using $\boldsymbol{G}^*$ and calculate its cost $\sum_{i=k}^{n-1} \left(\phi_i^* - (-1)^{v_i^*}\right)^2$. In view of the above fact, we define function $h^{(i)}$ as

   $$h^{(i)}(m) = \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v_i^*}\right)^2,$$

   where $v_{\ell+1}^*, v_{\ell+2}^*, \ldots, v_{n-1}^*$ are the labels of the only path $\boldsymbol{P}_m$ from node m to the goal node.

   Note that if node m is the goal node, then $h^{(i)}(m) = 0$. Furthermore, $h^{(i)}(m) = h^*(m)$, since there is only one path from node m to the goal node and $h^{(i)}(m)$ is the cost of this path.

Obviously, $h^{(i)}(m) \leq h^*(m)$ for any node m in the trellis.

For a given S_{C^*} and i, the i^{th} *order evaluation function* f is $f(m) = g(m) + h^{(i)}(m)$.

It is very important that the time complexity for calculating $h^{(i)}(m)$ be "reasonable," for otherwise the time taken by the decoding algorithm is spent on calculating $h^{(i)}(m)$, even though there are only a few nodes to be visited (open) in the trellis. In Appendix B.2 we present an algorithm to calculate $h^{(1)}(m)$ for node m at level $\ell, -1 \leq \ell < k-1$ whose time complexity is $O(|S_{C^*}| \times n)$.

We now give properties of heuristic function $h^{(i)}$ that will be used to speed up the decoding procedure. The proofs of properties 3.1 and 3.2 are given in Appendix C.1 and C.2, respectively. Properties 3.3 and 3.4 are immediate consequences of the definition of function $h^{(i)}$.

Property 3.1 *For a given* S_{C^*}

$$h^{(i)}(m_1) \leq h^{(i)}(m_2) + c(m_1, m_2),$$

where node m_2 *is an immediate successor of node* m_1*, and* $c(m_1, m_2)$ *is our arc cost from node* m_1 *to node* m_2*.*

Property 3.2 *For a given* S_{C^*} *and* i*, if nodes* $m_{\ell 1}$ *and* $m_{\ell 2}$ *are immediate successors of node* m_j*, then*

$$f(m_{\ell 1}) = f(m_j) \quad or \quad f(m_{\ell 2}) = f(m_j).$$

Now let S_{C^*} and S'_{C^*} be nonempty subsets of $\boldsymbol{C}^*$, and let $h^{(i)}$ be the i^{th} order heuristic function corresponding to S_{C^*}, and $h'^{(i)}$ for S'_{C^*}.

Property 3.3 *If* $S_{C^*} \subseteq S'_{C^*}$ *and* $0 \leq i \leq |S_{C^*}|$*, then*

$$h^{(i)}(m) \leq h'^{(i)}(m) \text{ for every node } m.$$

Property 3.4 *If* $0 \leq i \leq j \leq |S_{C^*}|$*, then*

$$h^{(i)}(m) \leq h^{(j)}(m) \text{ for every node } m.$$

We remark here that the decoding algorithm using function $h^{(j)}\left(h'^{(i)}\right)$ will not expand more nodes than the decoding algorithm using function $h^{(i)}\left(h^{(i)}\right)$. However, the time complexity for calculating $h^{(j)}(m)\left(h'^{(i)}(m)\right)$ will be higher than that of $h^{(i)}(m)\left(h^{(i)}(m)\right)$.

For the special case of $S_{C^*} = \{\mathbf{0}\}$, the first-order heuristic function is the heuristic function proposed in [22].

When a first-order heuristic function $h^{(1)}$ is used, the time and space complexities of the algorithm proposed here are $O(|S_{C^*}| \times n \times N(\boldsymbol{\phi}))$ and $O(n \times M(\boldsymbol{\phi}))$, respectively, where

$$\begin{aligned} N(\boldsymbol{\phi}) &= \text{the number of nodes visited during the decoding of } \boldsymbol{\phi}, \\ M(\boldsymbol{\phi}) &= \text{the maximum number of nodes that need to be stored} \\ &\quad \text{during the decoding of } \boldsymbol{\phi}. \end{aligned}$$

The derivation of these results is given in Appendix B.3.

We now give a geometric interpretation of the first-order evaluation function $f(m)$ for a node m at level $\ell, -1 \leq \ell < k-1$. Consider the set of all n-tuples over the real numbers. Figure 3.1 depicts $\boldsymbol{c}^*$, $\boldsymbol{\phi}^*$, and all the points $\boldsymbol{v} = (v_0, v_1, \ldots, v_{n-1})$ whose entries are 0 and 1 and $d_H(\boldsymbol{c}^*, \boldsymbol{v}) \in HW$. For calculating $f(m)$, we consider the lowest cost path $\boldsymbol{P}'_m$ from the start node to node m found so far by the algorithm. Let $\overline{v}_0, \overline{v} \ldots, \overline{v}_\ell$ be the labels of $\boldsymbol{P}'_m$. Now we consider only those points $\boldsymbol{v}$ defined above whose first $\ell+1$ entries are $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell$. In Figure 3.1, they are indicated by a check mark. From the points with check marks we select one that minimizes $\sum_{i=0}^{\ell}\left(\phi_i^* - (-1)^{\overline{v}_i}\right)^2 + \sum_{i=\ell+1}^{n-1}(\phi_i^* - (-1)^{v_i})^2$. This point is $\boldsymbol{v}' = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v'_{\ell+1}, \ldots, v'_{n-1})$ in Figure 3.1. Thus $f(m) = \sum_{i=0}^{\ell}\left(\phi_i^* - (-1)^{\overline{v}_i}\right)^2 + \sum_{i=\ell+1}^{n-1}\left(\phi_i^* - (-1)^{v'_i}\right)^2$.

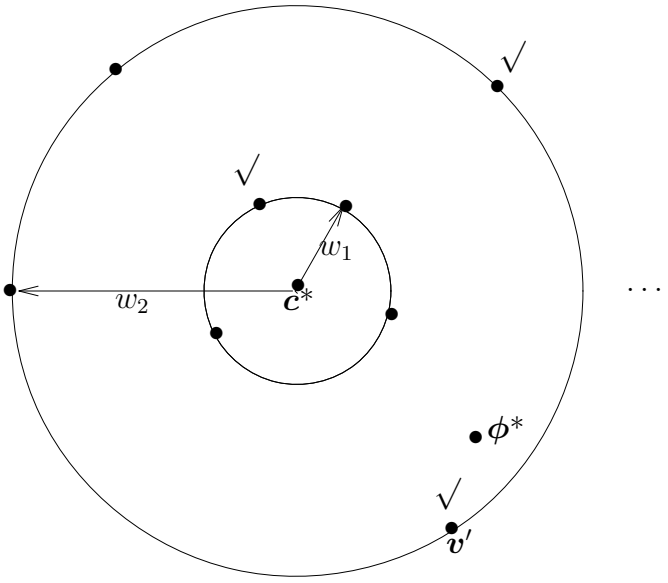

Figure 3.1: Geometric interpretation of $f(m)$

For long block codes it may be impossible to determine the set HW; however, our algorithm will still find the optimal solution even if in the computation of function h the algorithm considers all the Hamming weights of any superset of HW. The algorithm using a superset of HW may visit more nodes than that using HW. Furthermore, as pointed out in Section 2.2, the algorithm will not open fewer nodes if it uses a proper superset of HW instead of HW in the computation of heuristic function.

3.3 Speed-up Techniques

In this section we present some properties of the decoding algorithms that can be used to speed up the decoding procedure. In order to simplify the presentation of these techniques we assume that function h belongs to the class of heuristic functions defined above.

By Property 3.1, function h satisfies the monotone restriction,

$$h(m_i) \leq h(m_j) + c(m_i, m_j),$$

where node m_j is an immediate successor of node m_i and $c(m_i, m_j)$ is our arc cost from node m_i to node m_j. Then, as we pointed out in Section 2.3, we do not need to store the list CLOSED if we do not check for repeated nodes and we do not have to update the parentage in the search tree of any successors of the node that our algorithm selects to expand.

Let node m_1 at level $\ell < k-2$ be the node on list OPEN selected for expansion. Let $h(m_1) = \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2$. Consider now the path $\overline{\boldsymbol{P}}_{m_1}$ from node m_1 to node m_2 at level $k-2$, whose labels are $v_{\ell+1}, v_{\ell+2}, \ldots, v_{k-2}$. It is easily seen by Property 3.2 and the definition of our function h that the value of function f for every node in path $\overline{\boldsymbol{P}}_{m_1}$ is equal to $f(m_1)$. Furthermore, by Theorem 2.6 we can conclude that path

$\overline{\boldsymbol{P}}_{m_1}$ will be the path followed by the algorithm. Thus, we do not have to calculate the values of function f for the nodes of this path, which reduces considerably the time complexity of the algorithm.

Our algorithm will search the trellis only up to level $k-1$, since we can construct the only path from any node m at level $k-1$ to the goal node using $\boldsymbol{G}^*$. The labels of the combined paths from the start node to node m, and from node m to the goal node, correspond to a codeword. So the cost of this path, which is equal to $f(m)$, can be used as an upper bound on the cost of an optimal path. By Theorem 2.4, we can use this upper bound to reduce the size of list OPEN. Furthermore, since there is a codeword whose corresponding path in the trellis has cost equal to $f(m)$, then we need to keep only one node on list OPEN whose f value is equal to the upper bound.

The trellis search can be stopped at any time when we know that a codeword $\boldsymbol{c}_\ell^* = \left(c_{\ell 0}^*, c_{\ell 1}^*, \ldots, c_{\ell(n-1)}^*\right)$ generated satisfies Inequality 1.1. The following criterion can be used to indicate this fact.

Criterion 3.1 *If* $h^{(i)}(m_s) = \sum_{j=0}^{n-1} \left(\phi_j^* - (-1)^{c_{\ell j}^*}\right)^2$, *then* $\boldsymbol{c}_\ell^*$ *satisfies Inequality 1.1.*

Recall that m_s is the start node.

The validity of this criterion is based on the fact that, since $\boldsymbol{C}^* \subseteq T(m_s, Y_i)$, then $h^{(i)}(m_s) \leq \sum_{j=0}^{n-1} \left(\phi_j^* - (-1)^{c_{tj}^*}\right)^2$ for any $\boldsymbol{c}_t^* \in \boldsymbol{C}^*$.

Note that the decision criterion introduced in [38] is equivalent to the above criterion for the special case, $S_{C^*} = \left\{\boldsymbol{c}_\ell^*\right\}$. It is easy to show that if a codeword $\boldsymbol{c}_\ell^*$ satisfies the criterion given by Inequalities 3.7a and 3.7b in [19], then it will also satisfy the criterion given in [38].

It is important to mention that the set S_{C^*} does not need to be fixed during the decoding of $\boldsymbol{\phi}$. In the case where S_{C^*} is allowed to change, we have an adaptive decoding procedure. In order to avoid increasing computation time when S_{C^*} is changed

at some stage of the decoding procedure, we may not want to recalculate the values of function $h^{(i)}$ with respect to the set S_{C^*} for every node on list OPEN. Under these circumstances, nodes on list OPEN may have values of function $h^{(i)}$ calculated with respect to different sets; thus we can no longer guarantee that monotone restriction will be satisfied, and we cannot assure that when a node is selected for expansion the decoding algorithm has already found a minimum cost path from the start node to this node. Therefore, the decoding algorithm may not find an optimal path, but by not checking for repeated nodes it is ensured that the decoding algorithm will find an optimal path. This can easily be seen, since the procedure will now generate a code tree, and $h^{(i)}(m) \leq h^*(m)$ for every node of the tree, independently of the set S_{C^*} used to compute $h^{(i)}(m)$. Thus, $f(m) \leq g(m) + h^*(m)$. As the procedure is now generating a code tree, the cost of the minimum cost path from the start node to a goal node that goes through node m is $g(m) + h^*(m)$. If we do not check for repeated nodes, then the adaptive version of decoding algorithm will never delete all optimal paths during the search procedure.

The outline and complexities of the adaptive algorithm are given in Appendix B.3.

3.4 Simulation Results for the AWGN Channel

In this section we present simulation results for the $(104, 52)$ binary extended quadratic residue code and the $(128, 64)$ binary extended BCH code when these codes are transmitted over the Additive White Gaussian Noise (AWGN) channel described in Section 1.2. In order to account for the redundancy in codes of different rates, we used the SNR per transmitted information bit $\gamma_b = E_b/N_0 = \gamma n/k$ in our simulation.

We do not know HW for these two codes, so we use a superset for them. For (104,52) we know that $d_{\min} = 20$ and that the Hamming weight of any codeword is

divisible by 4 [27]. Thus, for this code the superset used is $\{x|(x$ is divisible by 4 and $20 \leq x \leq 84)$ or $(x = 0)$ or $(x = 104)\}$. For (128,64), the superset used is $\{x|$ (x is even and $22 \leq x \leq 106)$ or $(x = 0)$ or $(x = 128)\}$.

We have implemented our adaptive decoding algorithm for the case $i = 1$, that is, we use a first-order heuristic function. Furthermore, the set S_{C^*} has cardinality 1 and is updated according to the following rule: For every codeword $\boldsymbol{c}^*{}_1$ generated during the decoding of $\boldsymbol{\phi}$, if the value of $h^{(1)}(m_s)$ calculated with respect to $\boldsymbol{c}^*{}_1$ is greater than the value of $h^{(1)}(m_s)$ calculated with respect to the codeword in S_{C^*}, then set $S_{C^*} = \{\boldsymbol{c}^*{}_1\}$. The rationale behind this rule is that, for any node m, $h^{(1)}(m) \geq h^{(1)}(m_s)$ whenever these values are calculated with respect to the same set S_{C^*}.

Simulation results attested to the fact that the efficiency of this decoding algorithm depends strongly on the selection of the initial set S_{C^*}.

In our implementation this initial set is constructed by considering the codeword $\boldsymbol{c}^*$, obtained as follows. Let $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1})$ be the hard-decision of $\boldsymbol{\phi}^*$. Furthermore, let $\boldsymbol{u} = (u_0, u_1, \ldots, u_{k-1}) = (y_0, y_1, \ldots, y_{k-1})$. That is,

$$u_i = \begin{cases} 0 & \text{if } \phi_i^* \geq 0; \\ 1 & \text{if } \phi_i^* < 0; \end{cases},$$

and $\boldsymbol{\phi}^* = (\phi_0^*, \phi_1^*, \ldots, \phi_{k-1}^*, \phi_k^*, \ldots, \phi_{n-1}^*)$. Now we let $S_{C^*} = \{\boldsymbol{c}^*\}$, where $\boldsymbol{c}^* = \boldsymbol{u} \cdot \boldsymbol{G}^*$.

In the implementation of our decoding algorithm we decided not to check for repeated nodes. In this situation the graph becomes a code tree. Thus, we do not have to keep list CLOSED. Furthermore, list OPEN is always kept ordered according to the values f of its nodes. In this case, by the analysis in Appendix B.3, the time complexity and the space complexity of our algorithm are $O(n \times N(\boldsymbol{\phi}))$ and

$O(n \times M(\boldsymbol{\phi}))$, respectively. Recall that

$$\begin{aligned} N(\boldsymbol{\phi}) &= \text{the number of nodes visited during the decoding of } \boldsymbol{\phi}, \\ M(\boldsymbol{\phi}) &= \text{the maximum number of nodes that need to be stored} \\ &\quad \text{during the decoding of } \boldsymbol{\phi}. \end{aligned}$$

The values of $N(\boldsymbol{\phi})$ and $M(\boldsymbol{\phi})$ will strongly depend upon the SNR. In Chapter 4, we will give an upper bound on the average of $N(\boldsymbol{\phi})$ for a simple heuristic function. In the worst case, the time and space complexities of our algorithm are $O(n \times 2^k)$, which are, under the condition $k \leq (n-k)$, equal to those of Wolf's algorithm [44], which are $O(n \times \min(2^k, 2^{n-k}))$ [14].

First, we give simulation results for the (104,52) code. Quadratic residue codes are known to be very good codes that are very difficult to decode even when only hard-decision decoding is employed [6, 12, 8, 33]. Some quadratic residue codes have been decoded by using information-set decoding algorithms [3]. However, these algorithms are sub-optimal, that is, do not implement the **MLD** rule. Thus, the only two maximum-likelihood soft-decision decoding algorithms known to us that can be used to decode the (104,52) code are Wolf's algorithm [44] and Hwang's algorithm [23].

It is difficult to compare the performance of our algorithm with that of Hwang's, because he found the subset of codewords that must be stored for implementing the MLD rule only for very short codes [23, Table I]. However, we observe that the complexities of Wolf's algorithm are approximately the same as those of Hwang's for the codes presented in Table I of [23]. More evidence of this claim can be obtained by using the results presented in [13]. We will therefore compare the performance of our algorithm to that of Wolf's. We will assume for comparison purposes that the time and space complexities of Wolf's algorithm are of $O(n \times \min(2^k, 2^{n-k}))$, since it is difficult to find, using Forney's procedure [20], a trellis with minimum number of

states for the $(104, 52)$ code.

The simulation results for the $(104, 52)$ code for γ_b equal to 5 dB, 6 dB, 7 dB, and 8 dB are given in Table 3.1. These results were obtained by simulating 35,000 samples for each SNR. Note that the time and space complexities of Wolf's algorithm are proportional to $2^{52} \approx 4.50 \times 10^{15}$.

Since during simulation no decoding errors occurred for any of the above SNRs, the bit error probability is estimated using the formula [19],

$$n_d \sqrt{d_{\min}/(4\pi nk\gamma_b)}\; e^{-(kd_{\min}\gamma_b/n)}, \tag{3.1}$$

where n_d is the number of codewords of Hamming weight $d_{\min}$. The value of n_d was calculated using the results presented in [28]. Table 3.2 gives an estimate of the bit error probability and coding gain for above SNRs.

The distributions of $N(\boldsymbol{\phi})$, $C(\boldsymbol{\phi})$, and $M(\boldsymbol{\phi})$ for the $(104, 52)$ code for γ_b equal to 5 dB are given in Table 3.3.

We now give the simulation results for the (128,64) code. Since an algebraic decoder that corrects up to 10-bit errors can be constructed for this code, the maximum-likelihood soft-decision decoding algorithm recently proposed in [24] can be implemented. However, in this paper simulation results are given only for very short codes up to length 23. Sub-optimal decoding procedures for this code have been proposed in [16, 3]. Again, we will assume for comparison purposes that the time and space complexities of Wolf's algorithm are of $O(n \times \min(2^k, 2^{n-k}))$, since it is very difficult to find, using Forney's procedure [20], a trellis with the minimum number of states for the $(128, 64)$ code. Note that the time and space complexities of Wolf's algorithm are proportional to $2^{64} \approx 1.84 \times 10^{19}$.

The simulation results for the (128,64) code for γ_b equal to 5 dB, 6 dB, 7 dB, and 8 dB are given in Table 3.4. These results were obtained by simulating 35,000

γ_b	5 dB		6 dB		7 dB		8 dB	
	max	ave	max	ave	max	ave	max	ave
$N(\boldsymbol{\phi})$	142123	19	2918	1	221	1	0	0
$C(\boldsymbol{\phi})$	32823	5	519	2	35	2	1	1
$M(\boldsymbol{\phi})$	13122	4	1912	1	155	1	0	0

where

$N(\boldsymbol{\phi})$ = the number of nodes visited during the decoding of $\boldsymbol{\phi}$,

$C(\boldsymbol{r})$ = number of codewords constructed in order to decide on the closest codeword to $\boldsymbol{\phi}$;

$M(\boldsymbol{\phi})$ = the maximum number of nodes that need to be stored during the decoding of $\boldsymbol{\phi}$.

max = maximum value among 35,000 samples;

ave = average value among 35,000 samples;

γ_b = E_b/N_0.

Table 3.1: Simulation for the $(104, 52)$ code

γ_b	5 dB	6 dB	7 dB	8 dB
P_b	$2.028 \times 10^{-10^*}$	$5.023 \times 10^{-14^*}$	$1.494 \times 10^{-18^*}$	$3.079 \times 10^{-24^*}$
CG	7.90	8.35	8.80	9.05

P_b = bit error probability;

CG = coding gain (dB);

* Calculate using Formula 3.1.

Table 3.2: Bit error probability and coding gain for the $(104, 52)$ code

Interval	Frequencies		
	$N(\boldsymbol{\phi})$	$C(\boldsymbol{\phi})$	$M(\boldsymbol{\phi})$
0	34030	0	34196
1–2,000	953	34994	801
2,001–4,000	8	2	0
4,001–6,000	1	1	0
6,001–8,000	1	0	0
8,001–10,000	1	0	1
10,001–18,000	2	0	2
18,001–40,000	1	3	0
40,001–144,000	3	0	0
more than 144,000	0	0	0

Table 3.3: Distributions of $N(\boldsymbol{\phi})$, $C(\boldsymbol{\phi})$, and $M(\boldsymbol{\phi})$ for the $(104, 52)$ code for $\gamma_b = 5\ dB$

γ_b	5 dB		6 dB		7 dB		8 dB	
	max	ave	max	ave	max	ave	max	ave
$N(\boldsymbol{\phi})$	216052	42	13603	2	1143	1	0	0
$C(\boldsymbol{\phi})$	38219	8	1817	2	91	2	1	1
$M(\boldsymbol{\phi})$	16626	7	856	1	965	1	0	0

Table 3.4: Simulation for the $(128, 64)$ code

γ_b	5 dB	6 dB	7 dB	8 dB
P_b	$1.57 \times 10^{-12}*$	$1.71 \times 10^{-16}*$	$1.82 \times 10^{-21}*$	$1.02 \times 10^{-27}*$
CG	8.85	9.22	9.50	9.70

Table 3.5: Bit error probability and coding gain for the $(128, 64)$ code

samples for each SNR.

Table 3.5 gives only an estimate of the bit error probability and coding gain for above SNRs, because no decoding error occurred during simulation. When calculating P_b using Formula 3.1, the value of $n_d = 243,840$ was taken from [4].

The distributions of $N(\boldsymbol{\phi})$, $C(\boldsymbol{\phi})$, and $M(\boldsymbol{\phi})$ for the $(128, 64)$ code for γ_b equal to 5 dB are given in Table 3.6.

Simulation results for these codes show that for the 35,000 samples tried, a drastic reduction on the search space was achieved for most practical communication systems where the probability of error is less than 10^{-3} (γ_b greater than 6.8 dB) [12], even when the algorithm uses a superset of HW.

In order to verify the contribution of our heuristic function h to the efficiency

Interval	Frequencies		
	$N(\boldsymbol{\phi})$	$C(\boldsymbol{\phi})$	$M(\boldsymbol{\phi})$
0	33614	0	33893
1–2,000	1324	34988	1096
2,001–4,000	21	2	4
4,001–6,000	8	2	4
6,001–8,000	7	0	0
8,001–10,000	7	4	0
10,001–18,000	7	2	3
18,001–40,000	4	2	0
40,001–218,000	8	0	0
more than 218,000	0	0	0

Table 3.6: Distributions of $N(\boldsymbol{\phi})$, $C(\boldsymbol{\phi})$, and $M(\boldsymbol{\phi})$ for the $(128, 64)$ code for $\gamma_b = 5\ dB$

of our decoding algorithm, we implemented Dijkstra's algorithm with our speed-up techniques for the $(128, 64)$ code. Simulation results for 6 dB indicate that, for the two samples that did not satisfy Criterion 3.1 among the 35,000 samples, more than 350,000 nodes needed to be stored. On the other hand, our algorithm needed to store at most 856 nodes to decode these samples.

Simulation results showed that our adaptive decoding algorithm described in this section is at least one order of magnitude more efficient in time and space than that proposed in [22], where $S_{C^*} = \{\mathbf{0}\}$ during the entire decoding procedure.

Chapter 4

Analysis of the Performance of the Algorithm

In this chapter we investigate how to determine the computational performance of the decoding algorithms proposed in Chapter 3. Since the number of nodes visited, $N(\boldsymbol{\phi})$, and the number of codewords tried, $C(\boldsymbol{\phi})$, are random variables, the probability distributions of them will be given for the AWGN channel.

4.1 A Simple Heuristic Function h_s

In this section we define a simple heuristic function h_s. In the following section, this function h_s will be used by a simple version of the decoding algorithms proposed in Chapter 3 to determine these computational performances.

Let m be a node at level $\ell < k-1$, and let $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell$ be the labels of the lowest cost path $\boldsymbol{P}'_m$ from the start node to node m found so far by the algorithm. Define

h_s as

$$h_s(m) = \sum_{i=\ell+1}^{n-1} (|\phi_i| - 1)^2 .$$

For a node at a level greater than $k-2$, the function h_s will be defined as in Section 3.2.

We now show that if m is not start node m_s, and m is at level $\ell < k-1$, then the time complexity of calculating $h_s(m)$ is a constant. Let node m be an immediate successor of node m', which is on path $\boldsymbol{P}'_m$. Furthermore, let $\boldsymbol{y}$ be the hard-decision of $\boldsymbol{\phi}$. Then

$$h_s(m) = h_s(m') - (|\phi_\ell| - 1)^2.$$

Consequently,

$$f(m) = f(m') + (y_\ell \oplus \overline{v}_\ell)(4 \times |\phi_\ell|),$$

where $\overline{v}_\ell$ is the label of the arc between node m' and node m. Thus, the time complexity of calculating $f(m)$ is a constant when node m is not start node m_s.

It is easy to see that $h_s(m) \leq h(m)$ for every node m in the trellis or code tree, where h is the heuristic function proposed in Chapter 3.

4.2 Performance of Maximum-Likelihood Decoding Algorithm Using Function h_s

In this section we determine the computation performance of the decoding algorithms proposed in Chapter 3. In order to do this, we first derive the computational performance of a simplified version of them, which we denote by SDA. In this version:

1. Use function h_s as the heuristic function.

2. The search is performed on a code tree of the (n, k) code $\boldsymbol{C}$.

3. No ordering is performed on the positions of $\boldsymbol{\phi}$.

4. None of any speed-up technique mentioned in Section 3.3 is used.

We now state the main results of the computational performance of SDA when code $\boldsymbol{C}$ is transmitted over the AWGN channel described in Section 1.2. In order to account for the redundancy in codes of different rates, we use the SNR per transmitted information bit $\gamma_b = E_b/N_0 = \gamma n/k$ in this chapter.

Theorem 4.1 *Let $\overline{N}$ be the average number of nodes visited and let G be the standard normal distribution. Then*

$$\overline{N} \leq \tilde{N},$$

where

$$\begin{aligned}
\tilde{N} &= 2\left[k + \sum_{\ell=0}^{k-2}\sum_{\overline{d}=1}^{\ell+1}\binom{\ell+1}{\overline{d}} G\left(-\frac{\overline{\mu}(\ell,\overline{d})}{\overline{\sigma}(\ell,\overline{d})}\right)\right], \\
\overline{\mu}(\ell,\overline{d}) &= \sqrt{N_0}\left\{2\overline{d}\sqrt{\frac{k}{n}\gamma_b} + (n-\ell-1)\left[2\sqrt{\frac{k}{n}\gamma_b}G(-\sqrt{2\frac{k}{n}\gamma_b}) - \frac{1}{\sqrt{\pi}}e^{-\frac{k}{n}\gamma_b}\right]\right\}, \\
\overline{\sigma}^2(\ell,\overline{d}) &= N_0\left\{2\overline{d} + (n-\ell-1)\left[(4\frac{k}{n}\gamma_b + 2)G(-\sqrt{2\frac{k}{n}\gamma_b}) - 2\sqrt{\frac{\frac{k}{n}\gamma_b}{\pi}}e^{-\frac{k}{n}\gamma_b}\right.\right. \\
&\quad \left.\left. - \left(2\sqrt{\frac{k}{n}\gamma_b}G(-\sqrt{2\frac{k}{n}\gamma_b}) - \frac{1}{\sqrt{\pi}}e^{-\frac{k}{n}\gamma_b}\right)^2\right]\right\},
\end{aligned}$$

and

$$\gamma_b = \frac{E_b}{N_0}.$$

The proof of Theorem 4.1 is given in Appendix D.1.

From the above theorem and Chebyshev's inequality [18], we have the probability distribution of the number of nodes visited.

Theorem 4.2 *The probability distribution of the number of nodes visited is*

$$\boldsymbol{Pr}(N_s(\boldsymbol{\phi}) \geq L) \leq \frac{\tilde{N}}{L},$$

Figure 4.1: Average number of nodes visited for the $(48, 24)$ code

where $N_s(\boldsymbol{\phi})$ is the number of nodes visited when SDA decodes $\boldsymbol{\phi}$.

The values of $\tilde{N}$ for the $(48, 24)$ code for γ_b equal to 2 dB, 3 dB, 4 dB, 5 dB, 6 dB, 7 dB, and 8 dB are given in Figure 4.1. In this figure is also given the average number of nodes visited by the SDA, and by the decoding algorithm proposed in Section 3.4. These averages were obtained by simulating 10,000 samples. According to the results shown in Figure 4.1 the average number of nodes visited by the decoding algorithm proposed in Section 3.4 is much smaller than the number visited by the other two, especially for low SNRs.

Next, we give the results of the number of codewords tried.

Theorem 4.3 *Let $\overline{C}$ be the average number of codewords tried.*

$$\overline{C} \leq \tilde{C},$$

where

$$\begin{aligned}
\tilde{C} &= 2\left[1+\sum_{\overline{d}=1}^{k-1}\binom{k-1}{\overline{d}} G\left(-\frac{\overline{\mu}(\overline{d})}{\overline{\sigma}(\overline{d})}\right)\right], \\
\overline{\mu}(\overline{d}) &= \sqrt{N_0}\left\{2\overline{d}\sqrt{\frac{k}{n}\gamma_b}+(n-k+1)\left[2\sqrt{\frac{k}{n}\gamma_b}G(-\sqrt{2\frac{k}{n}\gamma_b})-\frac{1}{\sqrt{\pi}}e^{-\frac{k}{n}\gamma_b}\right]\right\}, \\
\overline{\sigma}^2(\overline{d}) &= N_0\left\{2\overline{d}+(n-k+1)\left[(4\frac{k}{n}\gamma_b+2)G(-\sqrt{2\frac{k}{n}\gamma_b})-2\sqrt{\frac{\frac{k}{n}\gamma_b}{\pi}}e^{-\frac{k}{n}\gamma_b}\right.\right. \\
&\quad \left.\left.-\left(2\sqrt{\frac{k}{n}\gamma_b}G(-\sqrt{2\frac{k}{n}\gamma_b})-\frac{1}{\sqrt{\pi}}e^{-\frac{k}{n}\gamma_b}\right)^2\right]\right\},
\end{aligned}$$

and

$$\gamma_b = \frac{E_b}{N_0}.$$

The proof of Theorem 4.3 is given in Appendix D.2.

From the above theorem and Chebyshev's inequality [18], we have the probability distribution of the number of codewords tried.

Theorem 4.4 *The probability distribution of the number of codewords tried is*

$$\boldsymbol{Pr}(C_s(\boldsymbol{\phi}) \geq L) \leq \frac{\tilde{C}}{L},$$

where $C_s(\boldsymbol{\phi})$ is the number of codewords tried when SDA decodes $\boldsymbol{\phi}$.

The values of $\tilde{C}$ for the $(48, 24)$ code for γ_b equal to 2 dB, 3 dB, 4 dB, 5 dB, 6 dB, 7 dB, and 8 dB are given in Figure 4.2. In this figure is also given the average number of codewords tried by the decoding algorithm proposed in Section 3.4. This average was obtained by simulating 10,000 samples. The average number of codewords tried

Figure 4.2: Average number of codewords tried for the $(48, 24)$ code

Figure 4.3: $\tilde{N}$ for the $(104, 52)$ code and the $(128, 64)$ code

by the SDA is not given in Figure 4.2, since they are very close to those calculated by the formula in Theorem 4.3. The results in Figure 4.2 show that the average number of codewords tried by the decoding algorithm proposed in Section 3.4 is much smaller than the average number for the other one, especially for low SNRs.

In Figures 4.3 and 4.4 we give the values of $\tilde{N}$ for the $(104, 52)$ code and the $(128, 64)$ code for γ_b from 2 dB to 8 dB, and $\tilde{C}$ for the $(104, 52)$ code and the $(128, 64)$ code for γ_b from 2 dB to 8 dB, respectively.

From Theorems 2.3, 4.2, and 4.4, we have the probability distributions of $N(\boldsymbol{\phi})$ and $C(\boldsymbol{\phi})$.

Figure 4.4: $\tilde{C}$ for the $(104, 52)$ code and the $(128, 64)$ code

Theorem 4.5

$$\boldsymbol{Pr}(N(\boldsymbol{\phi}) \geq L) \leq \frac{\tilde{N}}{L}.$$

Theorem 4.6

$$\boldsymbol{Pr}(C(\boldsymbol{\phi}) \geq L) \leq \frac{\tilde{C}}{L}.$$

From Theorem 4.5 the average number of nodes visited for the decoding algorithms proposed in Chapter 3 is smaller than or equal to the average numbers shown in Figure 4.3. Thus, the decoding algorithms proposed in Chapter 3 are efficient for codes of moderate lengths for most practical communication systems where the probability of error is less than 10^{-3} (γ_b greater than 6.8 dB).

Chapter 5

Suboptimal Decoding Algorithm

In Chapter 3 we proposed a class of maximum-likelihood soft-decision decoding algorithms for linear block codes using Algorithm A^*. These algorithms, guided by evaluation functions f, search through a trellis or a code tree. Each of these algorithms maintains a list OPEN of nodes of the trellis or the code tree that are candidates to be expanded. The algorithm selects for expansion the node on list OPEN with minimum values of function f. Function f is used to reduce drastically the search space and to make the decoding efforts of this decoding algorithm adaptable to the noise level; however, for low SNRs the number of nodes on list OPEN is still too large for the algorithm to have practical application.

The results of our simulations have shown that the number of nodes that need to be stored on list OPEN before the optimal path is found is much smaller than the total number of nodes stored before the algorithm stops. Thus we may limit the search with small degradations on the performance of the algorithm.

In this chapter we present a class of suboptimal soft-decision decoding algorithms that use a generalized algorithm A^*. In these algorithms we limit the size of list OPEN using two criteria that we will describe in the next section.

5.1 Criteria for Limiting the Size of List OPEN

The two criteria used to limit the size of list OPEN are:

1. If the probability that an optimal path goes through a node is smaller than a given parameter, then we do not store this node.

2. If a node needs to be stored on list OPEN when the size of list OPEN has reached a given upper bound, then we discard the node on list OPEN with the maximum value of function f.

Memory requirement is usually a crucial factor in the practical implementation of any decoding algorithm. Thus, in the second criterion we limit the size of list OPEN by giving an upper bound on the maximum number of nodes that can be stored on list OPEN.

To use the first criterion we need to calculate the probability that an optimal path goes through a node. We demonstrate below how to calculate this probability for the AWGN channel described in Section 1.2.

For any received vector $\boldsymbol{r}$, if an optimal decoding algorithm decodes it to a non-transmitted codeword, then it is very difficult for a suboptimal decoding algorithm to decode it to the transmitted codeword. Thus, when an optimal decoding algorithm decodes a received vector to a non-transmitted codeword we do not care which codeword a suboptimal decoding algorithm will decode to. Therefore, it is reasonable to consider only those received vectors that will be decoded to transmitted codewords by an optimal decoding algorithm. That is, when we derive the probability that an optimal path goes through a node, we may assume that no decoding error will occur if we employ an optimal decoding algorithm. Under this assumption we have the following theorem.

Theorem 5.1 *Let an (n, k) code $\boldsymbol{C}$ be transmitted over the AWGN channel. When no decoding error occurs, the probability distribution of $f^*(m_s)$ is approximately a normal distribution with mean μ and variance σ^2, where*

$$\begin{aligned} \mu &= n\frac{N_0}{2}, \\ \sigma^2 &= n\frac{N_0^2}{2}. \end{aligned}$$

The proof of Theorem 5.1 is given in Appendix E.

Let node m be a node in a trellis or the code tree of the transmitted (n, k) code $\boldsymbol{C}$ and let UB be the lowest upper bound on the cost of an optimal path found so far by the algorithm. If an optimal path goes through node m, then the cost of an optimal path, $f^*(m_s)$, is greater than or equal to $h(m)$ and less than or equal to UB. Thus, the probability that an optimal path goes through node m is less than or equal to $\boldsymbol{Pr}(h(m) \leq f^*(m_s) \leq UB)$. This leads us to the following theorem.

Theorem 5.2 *Let T be the probability that an optimal path goes through node m. Furthermore, let UB be an upper bound on the cost of an optimal path. Then*

$$T \leq \frac{1}{\sigma\sqrt{2\pi}} \int_{h(m)}^{UB} e^{-\frac{1}{2}(\frac{t-\mu}{\sigma})^2} dt,$$

where

$$\begin{aligned} \mu &= n\frac{N_0}{2}, \\ \sigma^2 &= n\frac{N_0^2}{2}. \end{aligned}$$

Thus, when a node is visited, the algorithm calculates T for this node. If T is less than a given threshold, then we will discard this node.

We now describe the outline of our decoding algorithms. In our suboptimal decoding algorithms we will fix the maximum number of nodes, M_B, allowed on list

OPEN. As in optimal decoding algorithms, list OPEN is always kept ordered. When a node, m, is visited, the algorithm calculates T for this node. If T is less than a given threshold δ, then we discard this node. Otherwise, we need to insert this node into list OPEN. If the number of nodes on list OPEN is equal to M_B, then the algorithm discards the node with larger f value between node m and the node with the largest f value on list OPEN. The algorithm inserts the remaining node into list OPEN.

We remark here that Criterion 3.1 can also be used with suboptimal decoding algorithms and, furthermore, all the speed-up techniques in Section 3.3 can also be applied to suboptimal decoding algorithms.

From the above outline of suboptimal decoding algorithms it can be seen that we must have an efficient memory management to insert nodes into and delete nodes from list OPEN in order to implement suboptimal decoding algorithms as well as optimal decoding algorithms. We implement list OPEN using B-tree, a data structure that deals efficiently with the insertion and deletion of nodes [40].

5.2 Simulation Results for the AWGN Channel

In order to verify the performance of our suboptimal decoding algorithms, we give simulation results for the $(104, 52)$ binary extended quadratic residue code and for the $(128, 64)$ binary extended BCH code when these codes are transmitted over the AWGN channel described in Section 1.2.

We have implemented a suboptimal version of the adaptive decoding algorithm that uses a first-order heuristic function. Since this suboptimal decoding algorithm is performed on low SNRs, the initial S_C^* is constructed by considering the 16 codewords as follows. Let $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1})$ be the hard-decision of $\boldsymbol{\phi}^*$. Furthermore, let $S = \{\boldsymbol{u} | \boldsymbol{u} = (u_0, u_1, \ldots, u_{k-1})$ and $u_i = y_i$ for $0 \leq i \leq k-5\}$. For every element $\boldsymbol{u}$

Figure 5.1: Performance of suboptimal decoding algorithm for the $(104, 52)$ code

in S, we get a codeword $\boldsymbol{c}^* = \boldsymbol{u} \cdot \boldsymbol{G}^*$. Now we let $S_{C^*} = \{\boldsymbol{c}^*\}$, where the value of $h^{(1)}(m_s)$, calculated with respect to $\boldsymbol{c}^*$, is the largest among all the 16 codewords. The rule of updating S_C^* is the same as that in Section 3.4.

The simulation results for the $(104, 52)$ code for γ_b equal to 1.5 dB, 1.75 dB, 2.0 dB, 2.25 dB, 2.5 dB, and 2.75 dB are given in Figure 5.1 and in Table 5.1 for three threshold values. M_B is equal to 3,000.

In Figure 5.1 we also give a lower bound on the bit error probability of the maximum-likelihood decoding algorithm. This lower bound is obtained as follows. For every sample, when suboptimal decoding algorithm terminates, we have a codeword that is obtained from the algorithm. If this codeword is closer to the received vector than the transmitted codeword is, then any optimal decoding algorithm will

threshold	1.5 dB	1.75 dB	2.0 dB	2.25 dB	2.5 dB	2.75 dB
0.0	26357	23909	18366	13240	10070	6698
0.25	10976	9643	6481	3980	2879	1579
0.5	3166	2827	1818	950	703	344

Table 5.1: The average number of nodes visited during the decoding of $(104, 52)$ code

also decode the received vector to a non-transmitted codeword. Thus, we assume the optimal decoding algorithm will decode to the codeword obtained from the suboptimal decoding algorithm and report if a decoding error occurs. Bit error probability of the uncoded data is also given in Figure 5.1.

From Figure 5.1, for the $(104, 52)$ code, the performance of the suboptimal decoding algorithm with $\delta = 0.0$ is within 0.25 dB of the performance of an optimal decoding algorithm; the performance of the suboptimal decoding algorithm with $\delta = 0.25$ is within 0.5 dB of the performance of an optimal decoding algorithm; and the performance of the suboptimal decoding algorithm with $\delta = 0.5$ is within 1 dB of the performance of an optimal decoding algorithm. Thus, for the samples tried, limiting the size of list OPEN to 3,000 nodes introduced only a small degradation on the performance of the algorithm for the $(104, 52)$ code; however, the average numbers of nodes visited for the samples tried is several orders of magnitude smaller than those given in Figure 4.3.

The simulation results for the $(128, 64)$ code for γ_b equal to 1.0 dB, 1.25 dB, 1.5 dB, 1.75 dB, and 2.0 dB are given in Figure 5.2 and in Table 5.2 for three threshold values. M_B is equal to 6,000.

In Figure 5.2 we also give a lower bound on the bit error probability of the maximum-likelihood decoding algorithm and bit error probability of the uncoded

Figure 5.2: Performance of suboptimal decoding algorithm for the $(128, 64)$ code

threshold	1.0 dB	1.25 dB	1.5 dB	1.75 dB	2.0 dB
0.0	88325	82650	75905	65223	55474
0.25	54416	41694	35613	29554	23162
0.5	22294	16705	13478	10389	6910

Table 5.2: The average number of nodes visited during the decoding of $(128, 64)$ code

data.

From Figure 5.2, for the $(128, 64)$ code, the performance of the suboptimal decoding algorithm with $\delta = 0.0$ is within 0.5 dB of the performance of an optimal decoding algorithm; the performance of the suboptimal decoding algorithm with $\delta = 0.25$ is within 0.6 dB of the performance of an optimal decoding algorithm; and the performance of the suboptimal decoding algorithm with $\delta = 0.5$ is within 0.75 dB of the performance of an optimal decoding algorithm. Thus, for the samples tried, limiting the size of list OPEN to 6,000 nodes introduced only a small degradation on the performance of the algorithm for the $(128, 64)$ code; however, the average numbers of nodes visited for the samples tried is several orders of magnitude smaller than those given in Figure 4.3.

Chapter 6

Conclusions and Further Research

6.1 Conclusions

In this dissertation we have proposed a novel decoding technique. Simulation results for the linear block codes in Section 3.4 show that, for the 35,000 samples tried, this decoding technique drastically reduced the search space, especially for most practical communication systems where the probability of error is less than 10^{-3} (γ_b greater than 6.8 dB) [12]. For example, the results of Table 3.4 at 6 dB show that, for the 35,000 samples tried, in the worst case this decoding algorithm is approximately 15 orders of magnitude more efficient in time and space than Wolf's algorithm.

We would like to emphasize here the flexibility of this decoding technique. For example, (1) it is applicable to any linear block code; (2) it does not require the availability of a hard decision decoder; (3) in order to make it more efficient to decode a particular code, we can design a heuristic function that takes advantage of the specific properties of this code; (4) any stopping criterion can be easily incorporated into it.

Furthermore, we would like to point out that the algorithms present in this dissertation are suitable for a parallel implementation, one reason being that when calculating $h(m)$ for node m, the algorithm has determined the labels of the path from node m to a node at level $k-2$ that it will follow, so the successors of the nodes in this path can be open simultaneously and processed independently. This will substantially reduce the idle time of processors and the overhead due to processor communication; thus we expect a very good speed-up from parallel versions of our algorithms.

In Chapter 4 we determine an upper bound of the probability distribution of the number of nodes visited by our decoding algorithms. From the figures shown in that chapter, these decoding algorithms are efficient for most practical communication systems where the probability of error is less than 10^{-3} (γ_b greater than 6.8 dB).

For low SNRs and for codes of moderate to long lengths, the number of nodes visited during the decoding procedure may be great and make the proposed decoding algorithm impractical. Thus in Chapter 5 we develop a class of suboptimal decoding algorithms based on a generalized algorithm A^*. From Figures 5.1 and 5.2, the performance of the suboptimal decoding algorithms is within 0.25 dB of the performance of an optimal decoding algorithm for the $(104, 52)$ binary extended quadratic residue code and within 0.5 dB for the $(128, 64)$ binary extended BCH code.

The decoding approach proposed in this dissertation may make an impact on both the theoretical and practical branches of coding theory. Theoreticians will be challenged to identify and construct classes of linear codes whose properties maximize the efficiency of this decoding procedure. And practitioners will want to find the most efficient way to implement this algorithm in a fast, single-purpose processor using sequential/parallel structures.

6.2 Further Research

Several research topics that might be chosen as follow-ups to the work presented in this dissertation include:

1. Using more properties of a specified linear block code in the design of a new function h in order to decode this code more efficiently.

2. Finding a tighter upper bound on the average number of nodes visited by our decoding algorithms.

3. Applying algorithm A^* to the decoding of convolutional codes.

4. Designing parallel versions of our decoding algorithms.

5. Designing a class of linear block codes that can be efficiently decoded by our decoding algorithms.

Appendix A

Proof of Theorems in Chapter 2

In this appendix we give all the proofs of theorems in Chapter 2.

A.1 Proof of Theorem 2.1

We prove by contradiction that when a goal node is selected for expansion, algorithm A^* has already found a minimum cost path from the start node to this goal node. Thus, when the algorithm selects to expand the goal node, it has found an optimal path.

Let node m_{g_1} be a goal node selected for expansion by algorithm A^*. Assume that the path from the start node to node m_{g_1} found so far by algorithm A^* is not an optimal path. Let $\boldsymbol{P}'^{*}_{m_{g_2}} = (m_s, m_0, \ldots, m_\ell, \ldots, m_{g_2})$ be an optimal path. Note that m_{g_1} and m_{g_2} may be two distinct goal nodes. Let node m_ℓ be the first node in this sequence of nodes that is on list OPEN. Furthermore, let $g^*(m)$ be the actual cost of a minimum cost path from the start node to node m. By Condition 1,

$$f(m_\ell) = g^*(m_\ell) + h(m_\ell) \leq g^*(m_\ell) + h^*(m_\ell) = g^*(m_{g_2}).$$

Since the cost, $g(m_{g_1})$, of the path from the start node to the goal node m_{g_1} found so far by algorithm A^* is larger than $g^*(m_{g_2})$, then $f(m_\ell) < f(m_{g_1})$. Contradiction.

A.2 Proof of Theorem 2.2

Let m be any node selected for expansion by algorithm A^*. If m is a goal node, we have $f(m) = f^*(m_s)$. Now, assume that m is not a goal node. Since m is not a goal node, then there is a node m' on list OPEN which is on an optimal path from m_s to a goal node with $f(m') \leq f^*(m_s)$. Thus, $f(m) \leq f^*(m_s)$.

A.3 Proof of Theorem 2.3

Let algorithm A_1 be algorithm A^* using function f_1, and let algorithm A_2 be algorithm A^* using function f_2. We prove this theorem using induction on the depth of a node in the search tree generated by algorithm A_2 at termination.

First, we prove that if algorithm A_2 expands a node m having zero depth in its search tree, then so will algorithm A_1. In this case, node m is node m_s. If m_s is a goal node, then neither algorithm expands any nodes. If m_s is not a goal node, both algorithms expand node m_s.

Now we assume (the induction hypothesis) that algorithm A_1 expands all the nodes expanded by algorithm A_2 that have depth less than or equal to k in the search tree generated by algorithm A_2. We need to show that any node m expanded by algorithm A_2, and of depth $k+1$ in the search tree generated by algorithm A_2, is also expanded by algorithm A_1.

By the induction hypothesis, any ancestor of node m in the search tree generated

by algorithm A_2 is also expanded by algorithm A_1. Thus,

$$g_1(m) \leq g_2(m).$$

Now assume that algorithm A_1 did not expand node m, which was expanded by algorithm A_2. Clearly, at the termination of algorithm A_1, node m must be on list OPEN for algorithm A_1 since algorithm A_1 expanded an immediate ancestor of node m. Since algorithm A_1 terminated in a minimum cost path without expanding node m, then

$$f_1(m) \geq f^*(m_s).$$

Thus,

$$g_1(m) + h_1(m) \geq f^*(m_s).$$

Since $g_1(m) \leq g_2(m)$ we have

$$h_1(m) \geq f^*(m_s) - g_2(m).$$

Since algorithm A_2 expanded node m, by Theorem 2.2

$$f_2(m) \leq f^*(m_s).$$

or

$$g_2(m) + h_2(m) \leq f^*(m_s)$$

or

$$h_2(n) \leq f^*(m_s) - g_2(m).$$

Thus,

$$h_2(m) \leq h_1(m).$$

Contradiction. So, any node m expanded by algorithm A_2, and of depth $k+1$ in the search tree generated by algorithm A_2, is also expanded by algorithm A_1.

A.4 Proof of Theorem 2.4

Consider an optimal path $\boldsymbol{P}^* = (m_s, m_0, \ldots, m_\ell, \ldots, m_g)$. Let node m_ℓ be the first node in this sequence of nodes that is on list OPEN. Thus

$$f(m_\ell) = g^*(m_\ell) + h(m_\ell),$$

where $g^*(m)$ is the actual cost of a minimum cost path from the start node to node m. By Condition 1,

$$f(m_\ell) = g^*(m_\ell) + h(m_\ell) \leq g^*(m_\ell) + h^*(m_\ell).$$

Since $g^*(m_\ell) + h^*(m_\ell) < UB$, then

$$f(m_\ell) < UB.$$

Thus, node m_ℓ will not be deleted from list OPEN.

A.5 Proof of Theorem 2.5

We prove by contradiction that if the monotone restriction is satisfied, then when a node is selected for expansion, algorithm A^* has already found a minimum cost path from the start node to this node.

Let node m_t be any node selected for expansion by algorithm A^*. Assume that the path from the start node to node m_t found so far by algorithm A^* is not a minimum cost path from the start node to node m_t. Let $\boldsymbol{P}'^*_{m_t} = (m_s, m_0, \ldots, m_\ell, \ldots, m_{t-1}, m_t)$ be a minimum cost path from the start node to node m_t. Let node m_ℓ be the first node in this sequence of nodes that is on list OPEN. Furthermore, let $g^*(m)$ be the actual cost of a minimum cost path from the start node to node m. By the monotone

restriction,

$$g^*(m_{t-1}) + h(m_{t-1}) \leq g^*(m_{t-1}) + h(m_t) + c(m_{t-1}, m_t) = g^*(m_t) + h(m_t).$$

By transitivity we have that

$$g^*(m_\ell) + h(m_\ell) \leq g^*(m_t) + h(m_t).$$

Since $g^*(m_\ell) + h(m_\ell) = f(m_\ell)$ and $g^*(m_t) < g(m_t)$, then $f(m_\ell) < f(m_t)$. Contradiction.

A.6 Proof of Theorem 2.6

Assume node m_i is selected for expansion. We now consider two cases:

Case 1. m_j was not visited. In this case

$$f(m_j) = g^*(m_i) + c(m_i, m_j) + h(m_j),$$

where $g^*(m)$ is the actual cost of a minimum cost path from the start node to node m. By the monotone restriction, $f(m_j) \geq g^*(m_i) + h(m_i) = f(m_i)$.

Case 2. m_j is on list OPEN. In this case, since m_j is not selected for expansion or by Case 1, then $f(m_j) \geq f(m_i)$.

Appendix B

Algorithms in Chapter 3

In this appendix we give algorithms in Chapter 3.

B.1 Reordering the Positions of Received Vector

Let $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_{n-1})$ be the received vector. If $|\phi_i| > |\phi_j|$, then we consider that ϕ_i is more "reliable" than ϕ_j, where $|x|$ is the absolute value of x. Let $\boldsymbol{\phi}' = (\phi'_0, \phi'_1, \ldots, \phi'_{n-1})$ be a vector obtained by permuting the positions of $\boldsymbol{\phi}$ such that $|\phi'_i| \geq |\phi'_{i+1}|$ for $0 \leq i < n-1$. The $k \times n$ matrix $\boldsymbol{G}'$ is obtained from $\boldsymbol{G}$ by applying this same permutation to the columns of $\boldsymbol{G}$. In order to give an algorithm to obtain $\boldsymbol{G}^*$, the generator matrix of $\boldsymbol{C}^*$, from $\boldsymbol{G}'$, we first introduce some definitions.

Let $\boldsymbol{A}$ be an $r \times m$ matrix. Given a set $S = \{i_1, i_2, \ldots, i_s\} \subset \{0, 1, 2, \ldots, m-1\}$ we say that S is a sub-information set of $\boldsymbol{A}$ iff the columns of $\boldsymbol{A}$ indexed by $i_1, i_2, \ldots, i_s$ are linearly independent. Furthermore, we define the SW operator. For $0 \leq i, j < m$, $SW(\boldsymbol{A}, i, j)$ is the $r \times m$ matrix obtained from $\boldsymbol{A}$ by swapping columns i and j of $\boldsymbol{A}$.

The following is an algorithm to obtain $\boldsymbol{G}^*$ from $\boldsymbol{G}'$ for $2 \leq k < n$.

1. $i \leftarrow 1; j \leftarrow 1; S = \{0\}$; $\boldsymbol{G}_1^* \leftarrow \boldsymbol{G}'$.

2. If $S \cup \{j\}$ is a sub-information set of $\boldsymbol{G}_1^*$, then $\boldsymbol{G}_1^* \leftarrow SW(\boldsymbol{G}_1^*, i, j)$;

   else

   $j \leftarrow j + 1$;

   go to 2.

3. $S \leftarrow S \cup \{i\}$.

4. If $|S| = k$, then stop;

   else

   $i \leftarrow i + 1$;

   $j \leftarrow j + 1$;

   go to 2.

5. Transform $\boldsymbol{G}_1^*$ into $\boldsymbol{G}^*$ by row operation such that the

   first k columns of $\boldsymbol{G}^*$ form a $k \times k$ identity matrix.

The time complexity of the procedure to construct $\boldsymbol{G}^*$ is $O(k^2 \times n)$; however, many of the operations performed during this construction can be done in parallel. In this case, the time complexity becomes $O(k \times n)$.

B.2 Algorithm to Calculate $h^{(1)}(m)$

We present an algorithm to calculate $h^{(1)}(m)$ for node m at level $\ell, -1 \leq \ell < k - 1$, whose time complexity is $O(|S_{C^*}| \times n)$.

First, we present this algorithm for the special case $S_{C^*} = \{\mathbf{0}\}$. Then we show how this algorithm can be applied to calculate $h^{(1)}(m)$ for the case $S_{C^*} = \{\boldsymbol{c}^*\}$ by modifying $\boldsymbol{\phi}^*$. Finally, it is immediately clear how to calculate $h^{(1)}(m)$ in general. For ease of notation, we denote $h^{(1)}(m)$ by $h(m)$.

B.2.1 Algorithm for the Case $S_{C^*} = \{\mathbf{0}\}$

We will show that to calculate $h(m)$ we need to construct at most two vectors belonging to $T(m, \{\mathbf{0}\})$.

Consider a node m at level ℓ and let $\boldsymbol{P}'_m$ be the lowest cost path from the start node to node m found so far by the algorithm. Furthermore, let $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell$ be the labels of $\boldsymbol{P}'_m$. Since $f(m) = g(m) + h(m) = \sum_{i=0}^{\ell} \left(\phi_i^* - (-1)^{\overline{v}_i}\right)^2 + h(m)$, then $h(m)$ depends only on the values of $\phi^*_{\ell+1}, \phi^*_{\ell+2}, \ldots$, and ϕ^*_{n-1}.

Let $\boldsymbol{u}_\ell = \left(u_{\ell(\ell+1)}, u_{\ell(\ell+2)}, \ldots, u_{\ell(n-1)}\right)$ be obtained by permuting the positions of $\left(\phi^*_{\ell+1}, \phi^*_{\ell+2}, \ldots, \phi^*_{n-1}\right)$ in such a manner that $u_{\ell i} \leq u_{\ell(i+1)}$ for $(\ell+1) \leq i \leq (n-2)$. We remark here that we can easily construct $\boldsymbol{u}_\ell$ from $\boldsymbol{u}_{-1}$.

Recall that

$$T(m, \{\mathbf{0}\}) = \{\boldsymbol{v} | \boldsymbol{v} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{\ell+1}, \ldots, v_{n-1}),\ W_H(\boldsymbol{v}) \in HW\},$$

and

$$h(m) = \min_{\boldsymbol{v} \in T(m, \{\mathbf{0}\})} \left\{ \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v_i}\right)^2 \right\}.$$

Because of the definition of $T(m, Y_i)$, we can compute $h(m)$ using $\boldsymbol{u}_\ell$ instead of $\left(\phi^*_{\ell+1}, \phi^*_{\ell+2}, \ldots, \phi^*_{n-1}\right)$.

Let $\boldsymbol{v}_p = \left(\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{p(\ell+1)}, v_{p(\ell+2)}, \ldots, v_{p(\ell+w)}, v_{p(\ell+w+1)}, \ldots, v_{p(n-1)}\right)$ and $\boldsymbol{v} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{\ell+1}, \ldots, v_{n-1})$ belong to $T(m, \{\mathbf{0}\})$ such that $W_H(\boldsymbol{v}_p) = W_H(\boldsymbol{v})$. Furthermore, let $v_{p(\ell+i)} = 1$ for $1 \leq i \leq w$ and $v_{p(\ell+i)} = 0$ for $(w+1) \leq i \leq (n-1)$. Thus

$$\boldsymbol{v}_p = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, 1, 1, \ldots, 1, 0, \ldots, 0).$$

Lemma B.1

$$\sum_{i=\ell+1}^{n-1} (u_{\ell i} - (-1)^{v_{pi}})^2 \leq \sum_{i=\ell+1}^{n-1} (u_{\ell i} - (-1)^{v_i})^2.$$

PROOF.

$$\begin{aligned} D_1 &= \sum_{i=\ell+1}^{n-1} (u_{\ell i} - (-1)^{v_{pi}})^2 - \sum_{i=\ell+1}^{n-1} (u_{\ell i} - (-1)^{v_i})^2 \\ &= 2 \sum_{i=\ell+1}^{n-1} \left\{ u_{\ell_i} \left((-1)^{v_i} - (-1)^{v_{pi}} \right) \right\}. \end{aligned}$$

Let $S = \{x | v_x = 0 \text{ and } \ell + 1 \leq x < w + \ell + 1\}$ and $S' = \{x | v_x = 1 \text{ and } w + \ell + 1 \leq x < n\}$. Since $\boldsymbol{v}_p = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, 1, 1, \ldots, 1, 0, \ldots, 0)$ and $W_H(\boldsymbol{v}) = W_H(\boldsymbol{v}_p)$, then $|S| = |S'|$. So $D_1 = 4 \left(\sum_{i \in S} u_{\ell i} - \sum_{i \in S'} u_{\ell i} \right) \leq 0$ since $|S| = |S'|$, $u_{\ell i} \leq u_{\ell j},\ i \in S$ and $j \in S'$. □

By Lemma B.1, when calculating the value of $h(m)$ we need only to consider vectors in $T(m, \{\mathbf{0}\})$ with patterns such as $\boldsymbol{v}_p$. Thus we need to consider only a subset of $T(m, \{\mathbf{0}\})$, $T'(m, \{\mathbf{0}\})$, such that it contains only vectors with patterns such as $\boldsymbol{v}_p$. Note that $T'(m, \{\mathbf{0}\}) \neq \emptyset$.

Let w_ℓ be the number of components in $\boldsymbol{u}_\ell$ that are negative and let $\boldsymbol{v}'_p = \left(\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v'_{p(\ell+1)}, v'_{p(\ell+2)}, \ldots, v'_{p(\ell+w')}, v'_{p(\ell+w'+1)}, \ldots, v'_{p(n-1)} \right) \in T'(m, \{\mathbf{0}\})$. Furthermore, let $v'_{p(\ell+i)} = 1$ for $1 \leq i \leq w'$ and $v'_{p(\ell+i)} = 0$ for $(w'+1) \leq i \leq (n-1)$.

Lemma B.2 *If $w' < w \leq w_\ell$, then*

$$\sum_{i=\ell+1}^{n-1} (u_{\ell i} - (-1)^{v_{pi}})^2 < \sum_{i=\ell+1}^{n-1} \left(u_{\ell i} - (-1)^{v'_{pi}} \right)^2.$$

PROOF.

$$\begin{aligned} D_2 &= \sum_{i=\ell+1}^{n-1} (u_{\ell i} - (-1)^{v_{pi}})^2 - \sum_{i=\ell+1}^{n-1} \left(u_{\ell i} - (-1)^{v'_{pi}} \right)^2 \\ &= 4 \sum_{i=\ell+w'+1}^{\ell+w} u_{\ell i} < 0, \end{aligned}$$

since $u_{\ell i} < 0,\ \ell + 1 \leq i < \ell + w_\ell + 1$. □

Lemma B.3 *If $w_\ell < w < w'$, then*

$$\sum_{i=\ell+1}^{n-1} (u_{\ell i} - (-1)^{v_{pi}})^2 \leq \sum_{i=\ell+1}^{n-1} \left(u_{\ell i} - (-1)^{v'_{pi}}\right)^2.$$

The proof of this lemma is similar to that in Lemma B.2.

We now consider three different patterns of $\boldsymbol{u}_\ell$.

Case 1. All components of $\boldsymbol{u}_\ell$ are negative.

In this case, by Lemma B.2,

$$h(m) = \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2,$$

where $\boldsymbol{v} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{\ell+1}, \ldots, v_{n-1})$ is the vector in $T'(m, \{\mathbf{0}\})$ with maximum Hamming weight.

Case 2. All components of $\boldsymbol{u}_\ell$ are greater than or equal to zero.

In this case, by Lemma B.3,

$$h(m) = \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2,$$

where $\boldsymbol{v} = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v_{\ell+1}, \ldots, v_{n-1})$ is the vector in $T'(m, \{\mathbf{0}\})$ with minimum Hamming weight.

Case 3. $\boldsymbol{u}_\ell$ has at least one negative and one positive component.

Let $\overline{w}_\ell = W_H((\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell))$. By Lemma B.2 and Lemma B.3 we can easily show that at most two vectors in $T'(m, \{\mathbf{0}\})$ must be inspected to calculate $h(m)$. These vectors are:

1. $\boldsymbol{v}' = \left(\overline{v}_0, \overline{v}_1, \overline{v}_\ell, v'_{\ell+1}, \ldots, v'_{n-1}\right)$, such that $W_H(\boldsymbol{v}')$ is the largest value among all the vectors $\boldsymbol{v}$ belonging to $T'(m, \{\mathbf{0}\})$ satisfying $W_H(\boldsymbol{v}) \leq \overline{w}_\ell + w_\ell$.

2. $\boldsymbol{v}'' = \left(\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v''_{\ell+1}, \ldots, v''_{n-1}\right)$, such that $W_H(\boldsymbol{v}'')$ is the smallest value among all vectors $\boldsymbol{v}$ belonging to $T'(m, \{\mathbf{0}\})$ satisfying $W_H(\boldsymbol{v}) > \overline{w}_\ell + w_\ell$.

We remark here that if $\boldsymbol{u}_\ell$ belongs to Case 3, then at least one of the vectors $\boldsymbol{v}'$ or $\boldsymbol{v}''$ will always exist.

If both $\boldsymbol{v}'$ and $\boldsymbol{v}''$ exist, then

$$h(m) = \min\left\{\sum_{i=\ell+1}^{n-1}\left(\phi_i^* - (-1)^{v'_i}\right)^2, \sum_{i=\ell+1}^{n-1}\left(\phi_i^* - (-1)^{v''_i}\right)^2\right\}.$$

Otherwise, $h(m)$ is calculated using the vector that exists.

B.2.2 Algorithm for the Case $S_{C^*} \neq \{\mathbf{0}\}$

Now we show that we can use the procedure presented in Section B.2.1 to calculate $h(m)$ for any $\boldsymbol{c}^* = (c_0^*, c_1^*, \ldots, c_\ell^*, c_{\ell+1}^*, \ldots, c_{n-1}^*)$. In order to differentiate the heuristic function calculated with respect to $\{\mathbf{0}\}$ and $\{\boldsymbol{c}^*\}$, we denote the heuristic function, calculating with respect to $\{\mathbf{0}\}$ by $h_{\mathbf{0}}$.

Consider a fictitious node m' such that a path $\boldsymbol{P}_{m'}$ from the start node to node m' has labels $\overline{v}_0 \oplus c_0^*, \overline{v}_1 \oplus c_1^*, \ldots, \overline{v}_\ell \oplus c_\ell^*$.

Lemma B.4

$$h(m) = \min_{\boldsymbol{v} \in T(m', \{\mathbf{0}\})}\left\{\sum_{i=\ell+1}^{n-1}\left((-1)^{c_i^*}\phi_i^* - (-1)^{vi}\right)^2\right\},$$

where

$$\begin{aligned} T(m', \{\mathbf{0}\}) &= \{\boldsymbol{v} | \boldsymbol{v} = (\overline{v}_0 \oplus c_0^*, \overline{v}_1 \oplus c_1^*, \ldots, \overline{v}_\ell \oplus c_\ell^*, v_{\ell+1}, \ldots, v_{n-1}) \quad \textit{and} \\ &\qquad d_H(\boldsymbol{v}, \mathbf{0}) \in HW\}. \end{aligned}$$

Note that this value is $h_{\mathbf{0}}(m')$ when we assume that the received vector is $\left((-1)^{c_0^*}\phi_0^*, (-1)^{c_1^*}\phi_1^*, \ldots, (-1)^{c_{n-1}^*}\phi_{n-1}^*\right)$.

PROOF.

$$h(m) = \min_{\boldsymbol{v}' \in T(m,\{\boldsymbol{c}^*\})} \left\{ \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v_i'}\right)^2 \right\}$$

$$= \min_{\boldsymbol{v}' \in T(m,\{\boldsymbol{c}^*\})} \left\{ \sum_{i=\ell+1}^{n-1} \left((-1)^{c_i^*}\phi_i^* - (-1)^{c_i^* \oplus v_i'}\right)^2 \right\}.$$

Let $\boldsymbol{v} = \boldsymbol{c}^* \oplus \boldsymbol{v}'$. Thus $\boldsymbol{v}' = \boldsymbol{v} \oplus \boldsymbol{c}^*$. We must show that $\boldsymbol{v}' \in T(m, \{\boldsymbol{c}^*\})$ iff $\boldsymbol{v} \in T(m', \{\boldsymbol{0}\})$.

$$\begin{aligned}
\boldsymbol{v}' \in T(m,\{\boldsymbol{c}^*\}) \quad \text{iff} \quad & \boldsymbol{v}' = (\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v'_{\ell+1}, \ldots v'_{n-1}), \\
& \text{and } d_H(\boldsymbol{v}', \boldsymbol{c}^*) \in HW \\
\text{iff} \quad & \boldsymbol{v} \oplus \boldsymbol{c}^* = \left(\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell, v'_{\ell+1}, \ \ldots, v'_{n-1}\right), \\
& \text{and } d_H(\boldsymbol{c}^* \oplus \boldsymbol{v}', \boldsymbol{0}) \in HW \\
\text{iff} \quad & \boldsymbol{v} = (\overline{v}_0 \oplus c_0^*, \overline{v}_1 \oplus c_1^*, \ldots, \overline{v}_\ell \oplus c_\ell^*, v'_{\ell+1} \oplus c_{\ell+1}^*, \ldots, v'_{n-1} \oplus c_{n-1}^*), \\
& \text{and } d_H(\boldsymbol{v}, \boldsymbol{0}) \in HW \\
\text{iff} \quad & \boldsymbol{v} \in T(m', \{\boldsymbol{0}\}).
\end{aligned}$$

Since $\boldsymbol{v}' \in T(m, \{\boldsymbol{c}^*\})$ iff $\boldsymbol{v} \in T(m', \{\boldsymbol{0}\})$, then we may consider minimization over vectors in $T(m', \{\boldsymbol{0}\})$ instead of in $T(m, \{\boldsymbol{c}^*\})$. Thus

$$h(m) = \min_{\boldsymbol{v} \in T(m',\{\boldsymbol{0}\})} \left\{ \sum_{i=\ell+1}^{n-1} \left((-1)^{c_i^*}\phi_i^* - (-1)^{v_i}\right)^2 \right\}.$$

□

Since the time complexity to find $\boldsymbol{v}'$ and $\boldsymbol{v}''$ in Case 3 of Section B.2.1 is $O(n)$, we can conclude that the time complexity to calculate $h(m)$ is $O(n)$. Thus the time complexity to calculate $h^{(1)}(m)$ is $O(|S_{C^*}| \times n)$.

B.3 Outline and Complexity Analysis of the Decoding Algorithm in Chapter 3

In this section we give an outline of our decoding algorithm. Recall that we do not check for repeated nodes, thus we do not have to store list CLOSED. We also give the orders of time and space complexities of this algorithm.

Given $\boldsymbol{\phi}$:

1. Construct $\boldsymbol{G}^*$ and $\boldsymbol{\phi}^*$; store the permutation used to construct $\boldsymbol{\phi}^*$ from $\boldsymbol{\phi}$.
2. Calculate the initial S_{C^*}, RESULT_f and RESULT_c^*.
3. If RESULT_c^* satisfies Criterion 3.1, then go to Step 12.
4. $f(m_s) = h(m_s)$ and construct $\overline{\boldsymbol{P}}_{m_s}$, which is the path the algorithm will follow.
5. Create OPEN, containing only m_s and RESULT_c^*.
6. LOOP: Select the first node on OPEN, remove it from OPEN. Call this node m. (Note that whenever node m_2 is inserted into OPEN in Step 10(b), then this node can always be selected as node m.)
7. If the level of node m is $k-1$, then go to Step 12.
8. Expand node m, generating nodes m_1 and m_2, the successors of node m.
9. If m is at level $k-2$, then for $i = 1$ to 2.
   (a) Construct the codeword $\boldsymbol{c}^*$ whose information bits are given by the labels of the path from the start node to node m_i.

(b) Calculate $g(m_i)$ and $h(m_i)$:

$$f(m_i) \longleftarrow g(m_i) + h(m_i).$$

(c) If $\boldsymbol{c}^*$ satisfies Criterion 3.1, then

RESULT_$c^* \longleftarrow \boldsymbol{c}^*$;

go to Step 12.

(d) Update S_{C^*}.

(e) If $f(m_i) <$ RESULT_f, then

RESULT_$c^* \longleftarrow \boldsymbol{c}^*$;

RESULT_$f \longleftarrow f(m_i)$;

remove all nodes on OPEN

whose f values are equal to or

greater than RESULT_f;

insert m_i into OPEN.

(f) If $f(m_i) \geq$ RESULT_f, then discard node m_i.

10. If m is at level $\ell < k-2$, then

(a) Select the node that is not on $\overline{\boldsymbol{P}}_m$,

call it m_1;

calculate $g(m_1)$ and $h(m_1)$;

if $\ell < k-3$, then construct $\overline{\boldsymbol{P}}_{m_1}$;

$f(m_1) \longleftarrow g(m_1) + h(m_1)$;

if $f(m_1) <$ RESULT_f then

insert node m_1 into OPEN;

otherwise, discard node m_1.

(b) $f(m_2) \longleftarrow f(m)$;
if $\ell < k-3$, then construct $\overline{\boldsymbol{P}}_{m_2}$ by deleting node m from $\overline{\boldsymbol{P}}_m$;
insert m_2 into OPEN as first node.

11. Go LOOP.

12. Construct $\widehat{\boldsymbol{c}}$ from RESULT_c^* by applying the inverse permutation that was used to construct $\boldsymbol{\phi}^*$ from $\boldsymbol{\phi}$.

We now apply the algorithm to a numerical example in which $|S_{C^*}| = 1$. In order for this example to be more illustrative we do not use Criterion 3.1, so we do not perform steps 3 and 9 (c) of the algorithm. Furthermore, the rule of updating S_{C^*} is the same as that in Section 3.4.

Example B.1 *Let $\boldsymbol{C}$ be the (8,4) binary extended Hamming code with $w_0 = 0$, $w_1 = 4$, and $w_2 = 8$. $\boldsymbol{C}$ is transmitted over an* **AWGN** *channel with $E = 1$. The generator matrix of $\boldsymbol{C}$ is*

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

Assume that the received vector is $\boldsymbol{\phi} = (-3, -2, -2, 1, 4, -1, 0, 0)$. We identify a node m at level ℓ, $0 \le \ell \le k-1$ in the tree by $m(\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell)$, where $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell$ are the labels associated with the path $\boldsymbol{P}'_m$ from the start node to node m.

STEP 1*:*

$$\boldsymbol{\phi}' = (4, -3, -2, -2, 1, -1, 0, 0),$$

$$\boldsymbol{G}' = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{G}_1^* = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{G}^* = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{\phi}^* = (4, -3, -2, 1, -2, -1, 0, 0),$$

and $\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 4 & 3 & 0 & 5 & 6 & 7 \end{pmatrix}$, *where* π *is a permutation used to construct* $\boldsymbol{\phi}^*$ *from* $\boldsymbol{\phi}$.

STEP 2*:*

$$\boldsymbol{c}^* = (0, 1, 1, 0, 0, 0, 1, 1),$$
$$S_{C^*} = \{(0, 1, 1, 0, 0, 0, 1, 1)\},$$
$$RESULT\_f = 29,$$
$$RESULT\_c^* = \boldsymbol{c}^*.$$

STEP 4*:*

By Lemma B.4, we first modify $\boldsymbol{\phi}^*$ *to* $(4, 3, 2, 1, -2, -1, 0, 0)$. *By Case 3 in Appendix B.2.1,*

$$v' = (0, 0, 0, 0, 0, 0, 0, 0) \text{ and}$$
$$v'' = (0, 0, 0, 0, 1, 1, 1, 1).$$

Thus,

$$\begin{aligned}\sum_{i=0}^{7}\left(\phi_i^* - (-1)^{v_i'}\right)^2 &= (4-1)^2+(3-1)^2+(2-1)^2+(1-1)^2+(-2-1)^2\\ &\quad +(-1-1)^2+(0-1)^2+(0-1)^2 = 29,\\ \sum_{i=0}^{7}\left(\phi_i^* - (-1)^{v_i''}\right)^2 &= (4-1)^2+(3-1)^2+(2-1)^2+(1-1)^2+(-2+1)^2\\ &\quad +(-1+1)^2+(0+1)^2+(0+1)^2 = 17.\end{aligned}$$

So $h(m_s) = f(m_s) = 17$*, and we obtain the labels of* $\overline{\boldsymbol{P}}_{m_s}$ *as* $<0,1,1>$.

STEP 5*:*

$OPEN = <m_s, m(0,1,1,0)>$.

LOOP 1*:*

STEP 6*: Consider node* m_s.

STEP 8*: Expand node* m_s *to obtain node* $m(0)$ *and node* $m(1)$.

STEP 10(a)*:*

$m_1 = m(1)$.

By Lemma B.4, we first modify $\boldsymbol{\phi}^*$ *to* $(4,3,2,1,-2,-1,0,0)$*. By Case 3 in Appendix B.2.1,*

$$v'' = (1,0,0,0,1,1,1,0).$$

Thus,

$$\begin{aligned}\sum_{i=1}^{7}\left(\phi_i^* - (-1)^{v_i''}\right)^2 &= (3-1)^2+(2-1)^2+(1-1)^2+(-2+1)^2\\ &\quad +(-1+1)^2+(0+1)^2+(0-1)^2 = 8.\end{aligned}$$

$g(m_1) = (4+1)^2 = 25$*. So* $f(m_1) = 25+8 = 33$*. Since* $33 > 29$*, we discard node* m_1.

STEP 10(b)*:*

$$\overline{\boldsymbol{P}}_{m(0)} = <1, 1> \textit{ and } f(m(0)) \ = 17.$$
$$OPEN = <m(0), m(0,1,1,0)>.$$

LOOP 2*:*

STEP 6*: Consider node* $m(0)$.

STEP 8*: Expand node* $m(0)$ *to obtain node* $m(0,0)$ *and node* $m(0,1)$.

STEP 10(a)*:*

$m_1 = m(0,0)$.

By Lemma B.4, we first modify $\boldsymbol{\phi}^*$ *to* $(4,3,2,1,-2,-1,0,0)$. *By Case 3 in Appendix B.2.1,*

$$v'' \ = (0,1,0,0,1,1,1,0).$$

Thus,

$$\sum_{i=2}^{7} \left(\phi_i^* - (-1)^{v_i''}\right)^2 \ = (2-1)^2 + (1-1)^2 + (-2+1)^2$$
$$+(-1+1)^2 + (0+1)^2 + (0-1)^2 = 4.$$

$g(m_1) = (4-1)^2 + (-3-1)^2 = 25$. *So* $f(m_1) = 25+4 = 29$. *Since* $29 = 29$ *we discard node* m_1.

STEP 10(b)*:*

$$\overline{\boldsymbol{P}}_{m(0,1)} = <1> \textit{ and } f(m(0,1)) \ = 17.$$
$$OPEN = <m(0,1), m(0,1,1,0)>.$$

LOOP 3:

STEP 6: *Consider node* $m(0,1)$.

STEP 8: *Expand node* $m(0,1)$ *to obtain node* $m(0,1,0)$ *and node* $m(0,1,1)$.

STEP 10(a):

$m_1 = m(0,1,0)$.

By Lemma B.4, we first modify $\boldsymbol{\phi}^*$ *to* $(4,3,2,1,-2,-1,0,0)$. *By Case 3 in Appendix B.2.1,*

$$v'' = (0,0,1,0,1,1,1,0).$$

Thus,

$$\sum_{i=3}^{7} \left(\phi_i^* - (-1)^{v_i''}\right)^2 = (1-1)^2 + (-2+1)^2 + (-1+1)^2 + (0+1)^2 + (0-1)^2 = 3.$$

$g(m_1) = (4-1)^2 + (-3+1)^2 + (-2-1)^2 = 22$. *So* $f(m_1) = 22+3 = 25$. *Since the level of* $m(0,1)$ *is 1, we do not need to calculate the labels of* $\overline{\boldsymbol{P}}_{m_1}$. *Furthermore, since* $25 < 29$, *then* $OPEN = <m(0,1,0), m(0,1,1,0)>$.

STEP 10(b)*:*

$$f(m(0,1,1)) = 17.$$
$$OPEN = <m(0,1,1), m(0,1,0), m(0,1,1,0)>.$$

LOOP 4:

STEP 6: *Consider node* $m(0,1,1)$.

STEP 8: *Expand node* $m(0,1,1)$ *to obtain node* $m_1 = m(0,1,1,0)$ *and node* $m_2 = m(0,1,1,1)$.

STEP 9(a): *Pick node* m_1,

$$\boldsymbol{c}^* = (0,1,1,0,0,0,1,1).$$

STEP 9(b):

$$\begin{aligned} f(m_1) &= (4-1)^2 + (-3+1)^2 + (-2+1)^2 + (1-1)^2 + (-2-1)^2 \\ &\quad +(-1-1)^2 + (0+1)^2 + (0+1)^2 = 29. \end{aligned}$$

STEP 9(d):

$S_{C^*} = \{(0,1,1,0,0,0,1,1)\}$.

STEP 9(f):

Discard node m_1.

STEP 9(a):*Pick node* m_2,

$\boldsymbol{c}^* = (0,1,1,1,0,1,0,0)$.

STEP 9(b):

$$\begin{aligned} f(m_2) &= (4-1)^2 + (-3+1)^2 + (-2+1)^2 + (1+1)^2 \\ &\quad +(-2-1)^2 + (-1+1)^2 + (0-1)^2 + (0-1)^2 = 29. \end{aligned}$$

STEP 9(f):

Discard node m_2.

LOOP 5:

STEP 6: *Consider node* $m(0,1,0)$.

STEP 8: *Expand node* $m(0,1,0)$ *to obtain node* $m_1 = m(0,1,0,0)$ *and node* $m_2 = m(0,1,0,1)$.

STEP 9(a): *Pick node* m_1,

$$\boldsymbol{c}^* = (0,1,0,0,1,1,0,1).$$

STEP 9(b):

$$\begin{aligned} f(m_1) &= (4-1)^2+(-3+1)^2+(-2-1)^2+(1-1)^2 \\ &\quad +(-2+1)^2+(-1+1)^2+(0-1)^2+(0+1)^2 = 25. \end{aligned}$$

STEP 9(d):

$S_{C^*} = \{(0,1,1,0,0,0,1,1)\}$.

STEP 9(e):

$RESULT\_c^* = (0,1,0,0,1,1,0,1)$.

$RESULT\_f = 25$.

$OPEN = <m(0,1,0,0)>$.

STEP 9(a): *Pick node* m_2,

$\boldsymbol{c}^* = (0,1,0,1,1,0,1,0)$.

STEP 9(b):

$$\begin{aligned} f(m_2) &= (4-1)^2+(-3+1)^2+(-2-1)^2+(1+1)^2 \\ &\quad +(-2+1)^2+(-1-1)^2+(0+1)^2+(0-1)^2 = 33. \end{aligned}$$

STEP 9(d):

$S_{C^*} = \{(0,1,1,0,0,0,1,1)\}$.

STEP 9(f):

Discard node m_2.

LOOP 6:

STEP 6: *Consider node* $m(0,1,0,0)$.

STEP 7: *Go to STEP 12.*

STEP 12:

$$\hat{\boldsymbol{c}} = (1, 0, 1, 0, 0, 1, 0, 1).$$

In Table B.1 we give the orders of time and space complexities for each step of the algorithm. One way of implementing list OPEN is to use a B-tree [40]. The time complexities of steps 6, 9(e), and 10(a) given in Table B.1 assume that this data structure is used. These complexities are obtained by noticing that the maximum number of nodes visited during decoding of $\boldsymbol{\phi}$ are upperbounded by $2^{k+1} - 1$, the number of nodes in a complete binary tree of height k. Thus, the time complexities of these steps are $O(k)$.

We remark here that Step 9(a) is not performed for all the nodes visited (open) during the decoding procedure. It is performed only for those nodes visited at level $k - 1$. Furthermore, in this step some operations performed during the construction of a codeword $\boldsymbol{uG}^*$ can be done in parallel. So, the time complexity of this step becomes $O(n)$, which we will assume to be the case in the following analysis. In Step 2 we also assume that the time complexity of constructing a codeword is $O(n)$.

In order to give the time and space complexities of our algorithm, we define the following quantities:

$O_t(h)$ = the time complexity of function h;

$O_s(h)$ = the space complexity of function h;

$N(\boldsymbol{\phi})$ = the number of nodes visited during the decoding of $\boldsymbol{\phi}$;

$M(\boldsymbol{\phi})$ = maximum |OPEN| that will occur during the decoding of $\boldsymbol{\phi}$, where |OPEN| denotes the number of nodes on list OPEN.

Based on the information given in Table B.1, we have

(a) time complexity is $O(k \times n) + (O_t(h) + O(n)) \times N(\boldsymbol{\phi})$;

(b) space complexity is $O(k \times n) + (O_t(h) + O(n)) \times M(\boldsymbol{\phi})$.

Complexities		
Step	Time	Space
1	$O(k \times n) + O(n \times \log n)$	$O(k \times n)$
2	$O(\|S_C^*\| \times n)$	$O(\|S_C^*\| \times n)$
3	$O_t(h)$	$O_s(h)$
4	$O_t(h)$	$O_s(h)$
5	$O(1)$	$O(n)$
6	$O(k)$	$O(n)$
7	$O(1)$	$O(1)$
8	$O(k)$	$O(n)$
9(a)	$O(n)$	$O(n)$
9(b)	$O(n)$	$O(1)$
9(c)	$O_t(h)$	$O_s(h)$
9(d)	$O_t(h)$	$O_s(h)$
9(e)	$O_t(h) + O(k)$	$O_s(h)$
9(f)	$O(1)$	$O(1)$
10(a)	$O_t(h) + O(k)$	$O_s(h)$
10(b)	$O(1)$	$O(1)$
11	$O(1)$	$O(1)$
12	$O(n)$	$O(1)$

Table B.1: Order of complexities

Since a lower bound for $N(\boldsymbol{\phi})$ and $M(\boldsymbol{\phi})$ is k if we do not use Criterion 3.1, then we may write the time complexity as $(O_t(h) + O(n)) \times N(\boldsymbol{\phi})$ and the space complexity as $(O_t(h) + O(n)) \times M(\boldsymbol{\phi})$.

Appendix C

Proof of Properties in Chapter 3

C.1 Proof of Property 3.1

Let node m_2 at level ℓ be an immediate successor of node m_1. Furthermore, let c_ℓ^* be the label of the arc from node m_1 to node m_2 and $c(m_1, m_2) = \left(\phi_\ell^* - (-1)^{c_\ell^*}\right)^2$. We now prove that $h^{(i)}(m_1) \leq h^{(i)}(m_2) + c(m_1, m_2)$.

1. $\ell < k-1$. Let $Y_i \in P_i(S_{C^*})$. Furthermore, let $\boldsymbol{v}' = (\overline{v}'_0, \overline{v}'_1, \ldots, \overline{v}'_\ell, v'_{\ell+1}, v'_{\ell+2}, \ldots, v'_{n-1}) \in T(m_2, Y_i)$ such that

$$\min_{\boldsymbol{v} \in T(m_2, Y_i)} \left\{ \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v_i}\right)^2 \right\} = \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right)^2.$$

   Since $\boldsymbol{v}' \in T(m_2, Y_i)$, then $\left(\overline{v}'_0, \overline{v}'_1, \ldots, \overline{v}'_{\ell-1}, c_\ell^*, v'_{\ell+1}, v'_{\ell+2}, \ldots, v'_{n-1}\right) \in T(m_1, Y_i)$. Thus

$$\min_{\boldsymbol{v} \in T(m_2, Y_i)} \left\{ \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v_i}\right)^2 \right\} + c(m_1, m_2) \geq \min_{\boldsymbol{v} \in T(m_1, Y_i)} \left\{ \sum_{i=\ell}^{n-1} \left(\phi_i^* - (-1)^{v_i}\right)^2 \right\}.$$

   Thus, $h^{(i)}(m_2) + c(m_1, m_2) \geq h^{(i)}(m_1)$.

2. $\ell = k-1$. $h^{(i)}(m_1) \leq h^*(m_1)$ and $h^{(i)}(m_2) = h^*(m_2)$. Since $h^*(m_1) - c(m_1, m_2) \leq h^*(m_2)$, then $h^{(i)}(m_1) \leq h^*(m_2) + c(m_1, m_2) = h^{(i)}(m_2) + c(m_1, m_2)$.

3. $\ell > k-1$. $h^{(i)}(m_1) = h^*(m_1)$ and $h^{(i)}(m_2) = h^*(m_2)$. Since $h^*(m_1) - c(m_1, m_2) = h^*(m_2)$, then $h^{(i)}(m_1) = h^{(i)}(m_2) + c(m_1, m_2)$.

C.2 Proof of Property 3.2

Consider node m_ℓ at level ℓ, $-1 \leq \ell < k-2$. Furthermore, let

$$h^{(i)}(m_\ell) = \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right)^2,$$

where $(\overline{v}'_0, \overline{v}'_1, \ldots, \overline{v}'_\ell, v'_{\ell+1}, v'_{\ell+2}, \ldots, v'_{n-1}) \in T(m_\ell, Z)$ for some $Z \in P_i(S_{C^*})$. Now consider the path $\overline{\boldsymbol{P}}_{m_\ell} = (m_\ell, m_{\ell+1}, \ldots, m_{k-2})$ from node m_ℓ to node m_{k-2} at level $k-2$ whose labels are $v'_{\ell+1}, v'_{\ell+2}, \ldots, v'_{k-2}$. We now show that if $m_{\ell+1}$ is a node in this path at level $\ell+1$, then $f(m_\ell) = f(m_{\ell+1})$.

By definition

$$\begin{aligned} f(m_\ell) &= g(m_\ell) + h^{(i)}(m_\ell) \\ &= g(m_\ell) + \left(\phi_{\ell+1}^* - (-1)^{v'_{\ell+1}}\right)^2 + \sum_{i=\ell+2}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right)^2 \\ &= g\left(m_{\ell+1}\right) + \sum_{i=\ell+2}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right). \end{aligned}$$

Since $(\overline{v}'_0, \overline{v}'_1, \ldots, \overline{v}'_\ell, v'_{\ell+1}, v'_{\ell+2}, v'_{\ell+3}, \ldots, v'_{n-1}) \in T(m_{\ell+1}, Z)$, then

$$\sum_{i=\ell+2}^{n-1} \left(\phi_i^* - (-1)^{v'_i}\right)^2 = \min_{\boldsymbol{v} \in T(m_{\ell+1,Z})} \left\{ \sum_{i=\ell+2}^{n-1} (\phi_i^* - (-1)^{v_i})^2 \right\},$$

otherwise

$$h^{(i)}(m_\ell) > \min_{\boldsymbol{v} \in T(m_\ell, Z)} \left\{ \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2 \right\}.$$

Analogously, we can conclude that

$$h^{(i)}(m_{\ell+1}) = \sum_{i=\ell+2}^{n-1} \left(\phi_i^* - (-1)^{v_i'}\right)^2.$$

Appendix D

Proof of Theorems in Chapter 4

In this appendix we give the proofs of theorems in Chapter 4.

D.1 Proof of Theorem 4.1

Let an (n, k) code $\boldsymbol{C}$ be transmitted over an AWGN channel. Thus,

$$\boldsymbol{Pr}(r_i|o) = \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(r_i-\sqrt{E})^2}{N_0}}$$

and

$$\boldsymbol{Pr}(r_i|1) = \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(r_i+\sqrt{E})^2}{N_0}}.$$

Since

$$\phi_i = \ln \frac{\boldsymbol{Pr}(r_i|0)}{\boldsymbol{Pr}(r_i|1)} = \frac{4\sqrt{E}}{N_0} r_i,$$

then

$$\boldsymbol{\phi} = \frac{4\sqrt{E}}{N_0} \boldsymbol{r}.$$

Thus, for fixed SNR $\frac{4\sqrt{E}}{N_0}$ can be treated as a positive constant. Since any positive constant multiplied to $\boldsymbol{\phi}$ will not affect the decoding procedure, we can substitute $\boldsymbol{r}$

for $\boldsymbol{\phi}$ in our decoding algorithm when $\boldsymbol{C}$ is transmitted over an AWGN channel [12]. Furthermore, without loss of generality we can assume that $\mathbf{0}$ is transmitted over an AWGN channel.

Let $\boldsymbol{P'_0}$ be the path from start node m_s to a goal node whose labels are all zero. Let us define the cost of the path $\boldsymbol{P'_0}$ as $g(\boldsymbol{P'_0})$. That is

$$g(\boldsymbol{P'_0}) = \sum_{i=0}^{n-1}(r_i - 1)^2.$$

From the definition of $f^*(m_s)$, we have

$$g(\boldsymbol{P'_0}) \geq f^*(m_s).$$

Now let node m be a node at level ℓ in the code tree and the labels of path $\boldsymbol{P'}_m$, the path from node m_s to node m found so far by the algorithm, are $\overline{v}_0, \overline{v}_1, \ldots, \overline{v}_\ell$. Let $S' = \{i | \overline{v}_i = 1, 0 \leq i \leq \ell\}$ and $|S'| = \overline{d}$. From the definition of function f

$$\begin{aligned} f(m) &= g(m) + h_s(m) \\ &= \sum_{i=0}^{\ell}\left(r_i - (-1)^{\overline{v}_i}\right) + \sum_{i=\ell+1}^{n-1}\left(|r_i| - 1|\right)^2. \end{aligned}$$

Now we want to calculate the probability that node m is expanded by the algorithm. From Theorem 2.2, this probability will be less than or equal to the probability that $f(m) \leq f^*(m_s)$, i.e., $\boldsymbol{Pr}(f(m) \leq f^*(m_s))$. Since

$$g(\boldsymbol{P'_0}) \geq f^*(m_s),$$

then

$$\boldsymbol{Pr}(f(m) \leq f^*(m_s)) \leq \boldsymbol{Pr}(f(m) \leq g(\boldsymbol{P'_0})).$$

Furthermore

$$f(m) \leq g(\boldsymbol{P'_0}) \quad \text{iff} \quad \sum_{i=0}^{\ell}\left(r_i - (-1)^{\overline{v}_i}\right)^2 + \sum_{i=\ell+1}^{n-1}\left(|r_i| - 1|\right)^2 \leq \sum_{i=0}^{n-1}(r_i - 1)^2$$

$$\text{iff} \quad \sum_{i \in S'} 4r_i + \sum_{i=\ell+1}^{n-1} 2(r_i - |r_i|) \leq 0$$

$$\text{iff} \quad \sum_{i \in S'} 2r_i + \sum_{i=\ell+1}^{n-1} (r_i - |r_i|) \leq 0.$$

Now let us define two new random variables Z_i and Z_i' as

$$Z_i = 2r_i \text{ and } Z_i' = r_i - |r_i|.$$

Since $\mathbf{0}$ is transmitted,

$$\boldsymbol{Pr}(r_i) = \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(r_i - \sqrt{E})^2}{N_0}}.$$

Let $E(X)$ be the mean of random variable X and let $Var(X)$ be the variance of X. Thus, $E(r_i)$ is $\sqrt{E}$ and $Var(r_i)$ is $\frac{N_0}{2}$. Then

$$E(Z_i) = 2\sqrt{E}$$

and

$$\begin{aligned} Var(Z_i) &= E(Z_i^2) - E^2(Z_i) \\ &= E(4r_i^2) - 4E^2(r_i) \\ &= 4[E(r_i^2) - E^2(r_i)] \\ &= 4Var(r_i) \\ &= 2N_0 \text{ [41].} \end{aligned}$$

Now let us calculate $E(Z_i')$ and $Var(Z_i')$:

$$E(Z_i') = \frac{1}{\sqrt{\pi N_0}} \int_{-\infty}^{0} 2te^{-\frac{(t-\sqrt{E})^2}{N_0}} dt.$$

Let $x = \frac{t-\sqrt{E}}{\sqrt{\frac{N_0}{2}}}$, then $dx = \frac{dt}{\sqrt{\frac{N_0}{2}}}$. Thus

$$E(Z_i') = \frac{\sqrt{\frac{N_0}{2}}}{\sqrt{\pi N_0}} \int_{-\infty}^{-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}} 2(\sqrt{\frac{N_0}{2}}x + \sqrt{E}) e^{-\frac{x^2}{2}} dx$$

$$\begin{aligned} &= \frac{2\sqrt{\frac{N_0}{2}}}{\sqrt{2\pi}} \int_{-\infty}^{-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}} x e^{-\frac{x^2}{2}} dx + \frac{2\sqrt{E}}{\sqrt{2\pi}} \int_{-\infty}^{-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}} e^{-\frac{x^2}{2}} dx \\ &= \frac{2\sqrt{\frac{N_0}{2}}}{\sqrt{2\pi}} \int_{-\infty}^{-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}} x e^{-\frac{x^2}{2}} dx + 2\sqrt{E} G(-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}), \end{aligned}$$

where G is the standard normal distribution.

Let $y = x^2$, then $dy = 2xdx$. Thus,

$$\begin{aligned} E(Z_i') &= \frac{\sqrt{\frac{N_0}{2}}}{\sqrt{2\pi}} \int_{\infty}^{\frac{E}{\frac{N_0}{2}}} e^{-\frac{y}{2}} dy + 2\sqrt{E} G(-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}) \\ &= 2\sqrt{E} G(-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}) - \sqrt{\frac{N_0}{\pi}} e^{-\frac{E}{N_0}}. \end{aligned}$$

Similarly,

$$\begin{aligned} Var(Z_i') &= E(Z_i^2) - E^2(Z_i) \\ &= \frac{1}{\sqrt{\pi N_0}} \int_{-\infty}^{0} (2t)^2 e^{-\frac{(t-\sqrt{E})^2}{N_0}} dt - E^2(Z_i) \\ &= 2(2E + N_0) G(-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}) - 2\sqrt{\frac{EN_0}{\pi}} e^{-\frac{E}{N_0}} - E^2(Z_i) \\ &= 2(2E + N_0) G(-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}) - 2\sqrt{\frac{EN_0}{\pi}} e^{-\frac{E}{N_0}} \\ &\quad - \left(2\sqrt{E} G(-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}) - \sqrt{\frac{N_0}{\pi}} e^{-\frac{E}{N_0}} \right)^2. \end{aligned}$$

Now let us define a new random variable X as

$$X = \sum_{i \in S'} 2r_i + \sum_{i=\ell+1}^{n-1} (r_i - |r_i|).$$

By the central limit theorem [41] , when $n \geq 10$, the probability distribution of X is approximately a normal distribution with mean $\overline{\mu}(\ell, \overline{d})$ and variance $\overline{\sigma}^2(\ell, \overline{d})$, where

$$\overline{\mu}(\ell, \overline{d}) = \overline{d} E(Z_i) + (n - \ell - 1) E(Z_i')$$

$$
\begin{aligned}
&= 2\overline{d}\sqrt{E} + (n-\ell-1)\left\{2\sqrt{E}G(-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}) - \sqrt{\frac{N_0}{\pi}}e^{-\frac{E}{N_0}}\right\},\\
\overline{\sigma}^2(\ell,\overline{d}) &= \overline{d}Var(Z_i) + (n-\ell-1)Var(Z_i')\\
&= 2\overline{d}N_0 + (n-\ell-1)\left\{2(2E+N_0)G(-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}) - 2\sqrt{\frac{EN_0}{\pi}}e^{-\frac{E}{N_0}}\right.\\
&\quad \left. - \left(2\sqrt{E}G(-\frac{\sqrt{E}}{\sqrt{\frac{N_0}{2}}}) - \sqrt{\frac{N_0}{\pi}}e^{-\frac{E}{N_0}}\right)^2\right\}.
\end{aligned}
$$

Thus,

$$\boldsymbol{Pr}(f(m) \le f^*(m_s)) \le \boldsymbol{Pr}(X \le 0) = G(-\frac{\overline{\mu}(\ell,\overline{d})}{\overline{\sigma}(\ell,\overline{d})}).$$

Since $f(m) \le g(\boldsymbol{P_0'})$ for any node m on path $\boldsymbol{P_0'}$, we can assume that node m will be expanded. We now consider those nodes that are not on this path. It is easy to see that, for any node that is not on path $\boldsymbol{P_0'}$, the labels of the path from node m_s to it will contain at least one 1. Since the first k positions of any codeword are information bits, the average number of nodes expanded by the algorithm is less than or equal to

$$\left[k + \sum_{\ell=0}^{k-2}\sum_{\overline{d}=1}^{\ell+1}\begin{pmatrix}\ell+1\\ \overline{d}\end{pmatrix} G\left(-\frac{\overline{\mu}(\ell,\overline{d})}{\overline{\sigma}(\ell,\overline{d})}\right)\right],$$

where

$$
\begin{aligned}
\overline{\mu}(\ell,\overline{d}) &= \sqrt{N_0}\left\{2\overline{d}\sqrt{\frac{k}{n}\gamma_b} + (n-\ell-1)\left[2\sqrt{\frac{k}{n}\gamma_b}G(-\sqrt{2\frac{k}{n}\gamma_b}) - \frac{1}{\sqrt{\pi}}e^{-\frac{k}{n}\gamma_b}\right]\right\},\\
\overline{\sigma}^2(\ell,\overline{d}) &= N_0\left\{2\overline{d} + (n-\ell-1)\left[(4\frac{k}{n}\gamma_b+2)G(-\sqrt{2\frac{k}{n}\gamma_b}) - 2\sqrt{\frac{\frac{k}{n}\gamma_b}{\pi}}e^{-\frac{k}{n}\gamma_b}\right.\right.\\
&\quad \left.\left. - \left(2\sqrt{\frac{k}{n}\gamma_b}G(-\sqrt{2\frac{k}{n}\gamma_b}) - \frac{1}{\sqrt{\pi}}e^{-\frac{k}{n}\gamma_b}\right)^2\right]\right\},
\end{aligned}
$$

and

$$\gamma_b = \frac{E_b}{N_0} = \frac{n}{k}\frac{E}{N_0}.$$

Since when a node is expanded by the algorithm, the algorithm will visit two nodes, the average number of nodes visited is less than or equal to

$$2\left[k+\sum_{\ell=0}^{k-2}\sum_{\overline{d}=1}^{\ell+1}\left(\begin{array}{c}\ell+1\\ \overline{d}\end{array}\right)G\left(-\frac{\overline{\mu}(\ell,\overline{d})}{\overline{\sigma}(\ell,\overline{d})}\right)\right],$$

where

$$\begin{aligned}
\overline{\mu}(\ell,\overline{d}) &= \sqrt{N_0}\left\{2\overline{d}\sqrt{\frac{k}{n}\gamma_b}+(n-\ell-1)\left[2\sqrt{\frac{k}{n}\gamma_b}G(-\sqrt{2\frac{k}{n}\gamma_b})-\frac{1}{\sqrt{\pi}}e^{-\frac{k}{n}\gamma_b}\right]\right\},\\
\overline{\sigma}^2(\ell,\overline{d}) &= N_0\left\{2\overline{d}+(n-\ell-1)\left[(4\frac{k}{n}\gamma_b+2)G(-\sqrt{2\frac{k}{n}\gamma_b})-2\sqrt{\frac{\frac{k}{n}\gamma_b}{\pi}}e^{-\frac{k}{n}\gamma_b}\right.\right.\\
&\quad\left.\left.-\left(2\sqrt{\frac{k}{n}\gamma_b}G(-\sqrt{2\frac{k}{n}\gamma_b})-\frac{1}{\sqrt{\pi}}e^{-\frac{k}{n}\gamma_b}\right)^2\right]\right\},
\end{aligned}$$

and

$$\gamma_b = \frac{E_b}{N_0} = \frac{n}{k}\frac{E}{N_0}.$$

D.2 Proof of Theorem 4.3

When a node at level $k-2$ is expanded, the algorithm will generate two codewords. Thus, to calculate the average number of codewords tried, we consider only those nodes that are at level $k-2$. From the argument in Appendix D.1, the average number of codewords tried is less than or equal to

$$2\left[1+\sum_{\overline{d}=1}^{k-1}\left(\begin{array}{c}k-1\\ \overline{d}\end{array}\right)G\left(-\frac{\overline{\mu}(\overline{d})}{\overline{\sigma}(\overline{d})}\right)\right],$$

where

$$\overline{\mu}(\overline{d}) = \sqrt{N_0}\left\{2\overline{d}\sqrt{\frac{k}{n}\gamma_b}+(n-k+1)\left[2\sqrt{\frac{k}{n}\gamma_b}G(-\sqrt{2\frac{k}{n}\gamma_b})-\frac{1}{\sqrt{\pi}}e^{-\frac{k}{n}\gamma_b}\right]\right\},$$

$$\begin{aligned}\overline{\sigma}^2(\overline{d}) &= N_0\left\{2\overline{d}+(n-k+1)\left[(4\frac{k}{n}\gamma_b+2)G(-\sqrt{2\frac{k}{n}\gamma_b})-2\sqrt{\frac{\frac{k}{n}\gamma_b}{\pi}}e^{-\frac{k}{n}\gamma_b}\right.\right.\\ &\quad\left.\left.-\left(2\sqrt{\frac{k}{n}\gamma_b}G(-\sqrt{2\frac{k}{n}\gamma_b})-\frac{1}{\sqrt{\pi}}e^{-\frac{k}{n}\gamma_b}\right)^2\right]\right\},\end{aligned}$$

and

$$\gamma_b = \frac{E_b}{N_0}.$$

Appendix E

Proof of Theorems in Chapter 5

In this appendix we give a proof of Theorem 5.1.

Let an (n, k) code $\boldsymbol{C}$ be transmitted over an AWGN channel. Since we assume that no decoding error occurs, then

$$\begin{aligned} f^*(m_s) &= \sum_{i=0}^{n-1} \left((-1)^{c_i}\sqrt{E} - (e_i + (-1)^{c_i}\sqrt{E})\right)^2 \\ &= \sum_{i=0}^{n-1} e_i^2, \end{aligned}$$

where,

$$\boldsymbol{Pr}(e_i) = \frac{1}{\sqrt{\pi N_0}} e^{-\frac{e_i^2}{N_0}}.$$

Now we define a new random variable Z_i as

$$Z_i = e_i^2.$$

Let $E(X)$ be the mean of random variable X and $Var(X)$ be the variance of X. Then,

$$E(Z_i) = \frac{1}{\sqrt{\pi N_0}} \int_{-\infty}^{\infty} x^2 e^{-\frac{x^2}{N_0}} dx$$

$$\begin{aligned} &= -\frac{\sqrt{N_0}}{2\sqrt{\pi}} x e^{-\frac{x^2}{N_0}} \Big|_{-\infty}^{\infty} + \frac{\sqrt{N_0}}{2\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-\frac{x^2}{N_0}} dx \\ &= \frac{N_0}{2}. \end{aligned}$$

$$\begin{aligned} Var(Z_i) &= E(Z_i^2) - E^2(Z_i) \\ &= \frac{1}{\sqrt{\pi N_0}} \int_{-\infty}^{\infty} x^4 e^{-\frac{x^2}{N_0}} dx - \frac{N_0^2}{4} \\ &= -\frac{\sqrt{N_0}}{2\sqrt{\pi}} x^3 e^{-\frac{x^2}{N_0}} \Big|_{-\infty}^{\infty} + \frac{3\sqrt{N_0}}{2\sqrt{\pi}} \int_{-\infty}^{\infty} x^2 e^{-\frac{x^2}{N_0}} dx - \frac{N_0^2}{4} \\ &= \frac{N_0^2}{2}. \end{aligned}$$

By the central limit theorem [41], when $n \geq 10$, the probability distribution of $f^*(m_s)$ is approximately a normal distribution with mean μ and variance σ^2, where

$$\begin{aligned} \mu &= n\frac{N_0}{2}, \text{ and} \\ \sigma^2 &= n\frac{N_0^2}{2}. \end{aligned}$$

Bibliography

[1] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Transactions on Information Theory*, pp. 284–287, March 1974.

[2] G. Battail, "Simplified Optimal Soft Decoding of Linear Block Codes," *IEEE International Symposium on Information Theory*, St Jovite, Québac, Canada, 1983.

[3] L. D. Baumert and R. J. McEliece, "Soft Decision Decoding of Block Codes," DSN Progress Report 42–47, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, July and August 1978.

[4] L. D. Baumert and L. R. Welch, "Minimum-Weight Codewords in the (128,64) BCH Code," DSN Progress Report 42–42, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, September and October 1977.

[5] Y. Berger and Y. Be'ery, "Bound on the Trellis Size of Linear Block Codes," *IEEE Transactions on Information Theory*, pp. 203–209, January 1993.

[6] E. R. Berlekamp, *Algebraic Coding Theory*. New York, NY: McGraw-Hill Book Co., 1968.

[7] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg, "On the Inherent Intractability of Certain coding Problems," *IEEE Transactions on Information Theory*, pp. 384–386, May 1978.

[8] R. E. Blahut, *Theory and Practice of Error Control Codes*. Reading, MA: Addison-Wesley Publishing Co., 1983.

[9] G. Brassard and P. Bratley, *Algorithmics Theory and Practice*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1988.

[10] J. Bruck and M. Naor, "The Hardness of Decoding Linear Codes with Preprocessing," *IEEE Transactions on Information Theory*, pp. 381–385, March 1990.

[11] D. Chase, "A Class of Algorithms for Decoding Block Codes with Channel Measurement Information," *IEEE Transactions on Information Theory*, pp. 170–181, January 1972.

[12] G. C. Clark, Jr. and J. B. Cain, *Error-Correction Coding for Digital Communications*. New York, NY: Plenum Press, 1981.

[13] G. P. Cohen, P. J. Godlewski, and T. Y. Hwang, "Generating Codewords in Real Space: Applications to Decoding," *Proceedings of the* 3rd *International Colloquium on Coding Theory and Applications*, pp. 114–122, 1988.

[14] J. H. Conway and N. J. A. Sloane, "Soft Decoding Techniques for Codes and Lattices, Including the Golay Code and the Leech Lattice," *IEEE Transactions on Information Theory*, pp. 41–50, January 1986.

[15] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: The MIT Press, 1991.

[16] B. G. Dorsch, "A Decoding Algorithm for Binary Block Codes and J-ary Output Channels," *IEEE Transactions on Information Theory*, pp. 391–394, May 1974.

[17] J. Fang, G. Cohen, P. Godlewski, and G. Battail, "On the Inherent Intractability of Soft Decision Decoding of Linear Codes," *Proceedings of the* 2^{nd} *International Colloquium on Coding Theory and Applications*, pp. 141–149, 1986.

[18] W. Feller, *An Introduction to Probability Theory and its Applications*. New York, NY: John Wiley and Sons, 1966.

[19] G. D. Forney, Jr., *Concatenated Codes*. Cambridge, MA: The M.I.T. Press, 1966.

[20] G. D. Forney, Jr., "Coset Codes–Part II: Binary Lattices and Related Codes," *IEEE Transactions on Information Theory*, pp. 1152–1187, September 1988.

[21] R. G. Gallager, *Information Theory and Reliable Communication*. New York, NY: John Wiley and Sons, 1968.

[22] Y. S. Han, C. R. P. Hartmann, and C.-C. Chen, "Efficient Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes Using Algorithm A^*," Technical Report SU-CIS-91-42, School of Computer and Information Science, Syracuse University, Syracuse, NY 13244, December 1991.

[23] T.-Y. Hwang, "Decoding Linear Block Codes for Minimizing Word Error Rate," *IEEE Transactions on Information Theory*, pp. 733–737, November 1979.

[24] T. Kancko, T. Nishijima, H. Inazumi, and S. Hirasawa, "An Efficient Maximum-Likelihood-Decoding Algorithm for Linear Block Codes with Algebraic Decoder," submitted for publication to *IEEE Transactions on Information Theory*.

[25] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.

[26] N. J. C. Lous, P. A. H. Bours, and H. C. A. van Tilborg, "On Maximum Likelihood Soft-Decision Decoding of Binary Linear Codes," *IEEE Transactions on Information Theory*, pp. 197–203, January 1993.

[27] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. New York, NY: Elsevier Science Publishing Company, Inc., 1977.

[28] C. L. Mallows and N. J. A. Sloane, "An Upper Bound for Self-Dual Codes," *Information and Control*, vol. 22, pp. 188–200, 1973.

[29] R. J. McEliece, *The Theory of Information and Coding*. Advanced Book Program Reading, MA: Addison-Wesley Publishing Company, 1982.

[30] D. J. Muder, "Minimal Trellises for Block Codes," *IEEE Transactions on Information Theory*, pp. 1049–1053, September 1988.

[31] N. J. Nilsson, *Principle of Artificial Intelligence*. Palo Alto, CA: Tioga Publishing Co., 1980.

[32] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley Publishing Company, 1984.

[33] I. S. Reed, T. K. Truong, X. Chen, and X. Yin, "The Algebraic Decoding of the $(41, 21, 9)$ Quadratic Residue Code," *IEEE Transactions on Information Theory*, pp. 974–986, May 1992.

[34] C. E. Shannon, "A Mathematical Theory of Communication," *Bell Sys. Tech. J.*, vol. 27, pp. 379–423, July 1948.

[35] J. Snyders, "Reduced Lists of Error Patterns for Maximum Likelihood Soft Decoding," *IEEE Transactions on Information Theory*, pp. 1194–1200, July 1991.

[36] J. Snyders and Y. Be'ery, "Maximum Likelihood Soft Decoding of Binary Block Codes and Decoders for the Golay Codes," *IEEE Transactions on Information Theory*, pp. 963–975, September 1989.

[37] P. Sweeney, *Error Control Coding: An Introduction*. Hertfordshire: Prentice-Hall International (UK) Ltd., 1991.

[38] D. J. Taipale and M. B. Pursley, "An Improvement to Generalized-Minimum-Distance Decoding," *IEEE Transactions on Information Theory*, pp. 167–172, January 1991.

[39] N. N. Tendolkar and C. R. P. Hartmann, "Generalization of Chase Algorithm for Soft Decision Decoding of Binary Linear Codes," *IEEE Transactions on Information Theory*, pp. 714–721, September 1984.

[40] A. M. Tenenbaum and M. J. Augenstein, *Data Structures Using Pascal*. Englewood Cliffs, NJ: Prentice-Hall, Inc., second edition, 1986.

[41] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Englewood Cliffs, NJ: Prentice-Hall Inc., 1982.

[42] A. J. Viterbi, "Error Bound for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Transactions on Information Theory*, pp. 260–269, April 1967.

[43] A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*. New York, NY: McGraw-Hill Book Company, 1979.

[44] J. K. Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis," *IEEE Transactions on Information Theory*, pp. 76–80, January 1978.

Biographical Data

Name:	Yunghsiang Sam Han
Date and Place of Birth:	April 24, 1962 Taipei, Taiwan
College:	National Tsing Hua University Shinchu,Taiwan B.S., Electrical Engineering, 1984 M.S., Electrical Engineering, 1986
Graduate Work:	Syracuse University Syracuse, New York Teaching Assistant: 1989-92 Research Assistant: 1992-93