On Algebraic Decoding of Algebraic-Geometric and Cyclic Codes

Ralf Kötter

Department of Electrical Engineering
Linköping University, S-581 83 Linköping, Sweden

Linköping 1996
## Contents

1 Introduction 1
   I  Background ................................. 1
   II  Why are cyclic and algebraic-geometric codes interesting? . 2
   III Outline of the thesis ......................... 3
   IV  Historical notes ............................ 5

2 Error-locating pairs for cyclic codes*
   I  Introduction ................................. 14
   II  Error-locating pairs ......................... 14
      A  An error-locating procedure ................. 14
      B  Error-locating functions .................... 18
   III Error-correcting pairs ....................... 20
   IV  Cyclic codes ............................... 22
      A  Notation .................................. 22

B Decoding BCH-codes . . . . . . . . . . . . . . . . 24
C Recurrences . . . . . . . . . . . . . . . . . . . . . . 27
D Correcting more errors . . . . . . . . . . . . . . . . 30
V Pairs from MDS-codes . . . . . . . . . . . . . . . . . . . 31
   A A class of MDS-codes . . . . . . . . . . . . . . . . 31
   B Construction of pairs . . . . . . . . . . . . . . . . 32
VI Complexity . . . . . . . . . . . . . . . . . . . . . . . . . 34
   A Short description of the Fundamental Iterative
      Algorithm . . . . . . . . . . . . . . . . . . . . . . 34
   B Reducing complexity . . . . . . . . . . . . . . . . 36
VII More on pairs . . . . . . . . . . . . . . . . . . . . . . . . 39
   A Error-location by hyperplanes . . . . . . . . . . . 39
   B Generalized Hamming weight . . . . . . . . . . . . 41
VIII Cyclic codes of length less than 63 . . . . . . . . . . . . . 42
IX Sequences of codes . . . . . . . . . . . . . . . . . . . . . 46
X Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . 48

3 A fast parallel implementation of a Berlekamp-Massey algorithm
   for algebraic-geometric codes† 53
I Introduction . . . . . . . . . . . . . . . . . . . . . . . 54
II Algebraic-geometric codes . . . . . . . . . . . . . . . . . 56
III The syndrome matrix . . . . . . . . . . . . . . . . . 61

†submitted to IEEE Transactions on Information Theory

iv
IV  The algorithm ................................................. 65
V   Majority voting ............................................ 74
VI  Implementation ............................................. 80
VII Error-erasure decoding ................................. 86
VIII Conclusions ................................................ 94

4  Fast generalized minimum distance decoding of algebraic-geometric and Reed-Solomon codes†
   I  Introduction ............................................... 100
   II An error-erasure-locating algorithm ................... 101
   III Generalized-Minimum Distance decoding ............. 105
   IV  The Fundamental Iterative Algorithm ................. 108
   V  Reed-Solomon codes ..................................... 111
   VI Algebraic-geometric codes ............................. 123
   VII Conclusions ............................................ 141

5  A Forney type formula for the determination of error values for Hermitian codes
   I  Introduction ............................................... 148
   II  The main idea ............................................ 149
   III Hermitian codes ......................................... 151
   IV Conclusions .............................................. 159

6  Conclusions .................................................. 163

†to appear in IEEE Transactions on Information Theory
Chapter 1

Introduction

I Background

The development of coding theory stems from the need to establish reliable transmission of information. A sender who tries to communicate a message to a receiver is often faced with a transmission channel that is subject to disturbances of various kinds. In many everyday life channels the signal that is picked up by the receiver is not identical to the signal that was sent by the transmitter.

This situation is tackled by encoding the set of messages into a prescribed set of codewords that are as different as possible with respect to the disturbances caused by the channel. As long as the corruption of the codeword caused by the channel is sufficiently small the receiver can still successfully guess which codeword was sent. The receivers task of choosing the codeword which is most similar to the received word is referred to as the decoding problem.

The decoding problem has a trivial solution which implies that the receiver simply compares the received word with all possible codewords. The receiver then can choose the codeword that best matches the received word. This may be a feasible strategy for codes with few codewords but becomes
very soon impractical with increasing code size. One of the crucial ques-
tions in the assessment of a coding scheme is the availability of efficient
algorithms to handle the encoding and decoding problem. A powerful code
that cannot be decoded in an efficient manner may be mathematically in-
teresting but is usually ruled out from engineering applications of coding
theory.

Many different coding principles have been developed that are suitable for
different channels. In this thesis we consider certain block codes, namely
cyclic codes and algebraic-geometric (AG) codes. In particular, we focus
on the development and investigation of efficient decoding algorithms for
these codes.

II Why are cyclic and algebraic-geometric codes
interesting ?

By a code we mean a finite subset of a metrical space as for example Eu-
clidean space or Hamming space. The distortion caused by a channel is of-
ten modelled as additive noise, which means that a channel adds a random
element of the space in consideration to the transmitted codeword. Based on
the assumption that the norm of the added error is likely to be small, the pri-
mary goal of code construction can be described as the problem of packing
a maximum number of spheres of a fixed diameter in the space of concern.
The resulting code then corresponds to the centers of these spheres. This
purely geometrical problem does not in principle impose any algebraic re-
strictions on the code. However, a code that is not carrying any algebraic
structure can usually only be described by an exhaustive list of codewords.
Moreover, the decoding problem can only be solved by performing the la-
borious work of comparing the received word to all codewords on the list.
In practical applications of coding theory, where the complexity of handling
a code is of crucial importance, this is usually not feasible.

In order to obtain codes that allow the development of efficient encoding
and decoding algorithms we are forced to impose an algebraic structure on
the code. The most important restriction on the codes that are investigated
in this thesis is the fact that these codes are linear spaces. This restriction allows the usage of the rich mathematical tool-box provided by linear algebra.

In the case of linear cyclic codes we moreover require that any cyclic shift of a codeword also results in a codeword. This property of a code turns out to yield a very strong algebraic structure. In particular, a linear cyclic code can be described as an ideal in a ring of polynomials over a finite field. Furthermore, it is possible to associate with every position in a codeword an element of a finite field. One important implication of this correspondence is that it is possible to derive a two step decoding algorithm. In the first step the algorithm determines the erroneous positions in the received word. The second step (if necessary) consists of calculating the corresponding error values. A second implication is that cyclic codes can be analyzed using properties of matrices of Vandermonde type. This turns out to be the most important mathematical tool in the treatment of linear cyclic codes.

Codes from algebraic geometry are obtained by evaluating a space of functions in a fixed set of points. The beautiful idea in the construction of these codes is to choose the point set for evaluation from an algebraic curve. This choice allows usage of the very well developed mathematical machinery from algebraic geometry. Although the algebraic structure of the codes differs according to the curve in consideration, very powerful and general mathematical tools, in particular the theorem of Riemann-Roch, are readily at hand to treat these codes. The decoding of AG codes follows the main idea of decoding cyclic codes, which separates the decoding problem into two steps. The positions in a codeword are now associated with the points on the curve.

### III. Outline of the thesis

This thesis is divided into 6 chapters. The core of the material is presented in chapters 2,3,4,5. These chapters are, up to editorial changes, identical with the following scientific articles.
Chapter 1. Introduction


In addition to the above four papers parts of the results of this thesis have been presented in [1][2][3][4] and [5].

Chapter 2 is a joint work with Iwan Duursma. The problem considered is the decoding of cyclic codes. The chapter explains how many earlier decoding procedures for cyclic codes can be derived in the framework of error-locating pairs. Decoding procedures are derived for all but four binary cyclic codes of length less than 63.

In chapter 3 the problem of efficiently solving the key equation in the decoding of AG codes is addressed. A parallel Berlekamp-Massey type algorithm for computing error-locating functions for one-point AG codes from non-singular, absolutely irreducible curves is derived.

Chapter 4 gives a fast generalized minimum-distance decoding algorithm in the framework of error-locating pairs. The approach is in principle applicable to any linear code. Special attention is paid to Reed-Solomon (RS) and AG codes.

The 5th chapter treats the computation of error values for the special case of Hermitian codes. The algorithm derived in this chapter is based on a generalization of Forney’s formula.

Finally in chapter 6 we summarize and give some concluding remarks.
IV  Historical notes

The development of cyclic codes was started by Prange [6] in 1957. Many important discoveries concerning cyclic codes were made around 1960. The BCH-codes, which are a subclass of cyclic codes, were discovered independently by Hocquenghem [7] and by Bose and Ray-Chaudhuri [8]. Reed and Solomon gave a first description of RS codes [9] and Gorenstein and Zierler observed that RS codes may be seen as a subclass of the BCH codes. The interpretation of cyclic codes as ideals was investigated by Peterson [10] and Kasami [11].

Since then a large number of papers have been written that deal with the characterization of cyclic codes and the description of particular classes of codes. Due to the main interest of this thesis we focus on the development of decoding algorithms.

The decoding of cyclic codes very soon concentrated on the decoding of the very important subclass of BCH codes. The first practical decoding procedure for binary BCH codes was given by Peterson [10]. Peterson’s algorithm was generalized by Gorenstein and Zierler [12] to non-binary cyclic codes. Thus an efficient decoding algorithm up to half the designed minimum distance of BCH codes was available around 1960. The algorithm is usually referred to as algebraic decoding. Numerous papers, dealing with implementation aspects and refinements of this algorithm, have appeared since then. We mention the work of Forney [13] and Chien [14] which derived a number of simplifications. A significant contribution was Berlekamp’s discovery of an algorithm that solves the key equation of algebraic decoding in an iterative fashion [15]. This algorithm was rederived and reinterpreted as a shift-register design problem by Massey in [16].

A number of algorithms have been developed that are similar to Peterson’s algorithm in the sense that they follow the idea of decoding by error location. Sugiyama, Kasahara, Hirasawa, and Namekawa employed the Euclidean algorithm in [17](1975). Berlekamp and Welch (1983) derived a decoding algorithm using interpolation techniques [18]. Many authors have improved the above algorithms in various ways so that today the decoding of BCH codes is very well investigated. Still the decoding of cyclic codes in general remains an unsolved problem. The concept of error-correcting
pairs, introduced by Pellikaan in 1992 [19], and the ideas of Feng and Rao [20] and Duursma [21] on majority coset decoding (both ideas were inspired by decoding procedures for AG codes) seem at this point the most promising approach for future research.

AG codes were discovered by V.D. Goppa in 1977 [22]. Goppa’s discovery opened a wide range of research problems connected with construction, description and generalization of AG codes as well as the application of AG codes as component codes in various code constructions. As in the case of cyclic codes we focus on the development of decoding algorithms. However we have to mention a result by Tsfasman, Vlăduţ, and Zink [23]. They proved, for sufficiently large alphabets (i.e. $q \geq 49$, $q$ is a square number) the existence of $q$-ary code sequences with asymptotic rate exceeding the Varshomov-Gilbert existence bound for codes. This fact is even more remarkable as the construction of these codes requires only polynomial complexity. (For a precise formulation of this see [24]).

Around 1989 the decoding problem was attacked in two different ways by Justesen, Larsen, Jensen, Havemose, and Høholt [25] and by Porter [26]. The two approaches can be seen as generalizations of Peterson’s decoding algorithm for BCH codes and the Euclidean algorithm for the decoding of classical Goppa codes, respectively. The essence of the algorithm by Justesen et al., which was formulated for plane irreducible curves, was translated into a formal algebraic geometric language by Skorobogatov and Vlăduţ [27] and is usually referred to as the basic algorithm. Porters algorithm was further developed by Ehrhard [28] and Porter, Shen, and Pellikaan [29].

The algorithms proposed by Justesen et al. and Porter had a substantial gap in the error correction capability in as much as the number of guaranteed correctable errors was bounded by $\lfloor (d - g - 1)/2 \rfloor$, where $d$ is the designed distance of the code and $g$ the genus of the curve used for code construction. This gap to the error correction capability of the code was decreased in [27] for the algorithm proposed by Justesen et al.. The corresponding algorithm is referred to as the modified algorithm. An extension of this algorithm is found in Duursma’s work [30]. A similar modification in order to improve Porters algorithm was given in [29].

In 1989 Pellikaan [31] proved for sufficiently large alphabets the existence of an effective decoding algorithm that consists of a number of applications
of the basic algorithm with different but fixed start parameters. At least one
of the decoding algorithms is then guaranteed to correct any error pattern
of weight at most half the minimum distance of the code. However, no
effective method for the determination of these start parameters is known in
the general case.

In 1993 Feng and Rao gave an effective decoding algorithm that achieved
error correction up to the half the designed minimum distance [20]. The
idea of their approach is to iteratively determine more and more informa-
tion about the error vector. The algorithm, which was formulated for plane
irreducible curves, was extended and reformulated in an algebraic geometric
language by Duursma [21].

Around the same time as Feng and Rao formulated their algorithm Ehrhard
[32] gave an algorithm that achieves error correction up to half the designed
minimum distance if the designed minimum distance is sufficiently large.
The main idea of Ehrhard’s algorithm is to iteratively determine the param-
eters of a basic algorithm for the particular error pattern in question. The
precise relation (if any) between Ehrhard’s algorithm and the algorithm of
Feng and Rao is still not determined.

The computationally efficient solution of the decoding problem has been an
active research area accompanying the development of the different decod-
ing algorithms. The fundamental work in this area was done by Justesen,
Larsen, Jensen, and Høholt [33]. They applied a modified two-dimensional
Berlekamp-Massey algorithm proposed by Sakata [34] to the decoding of
codes on plane irreducible curves. The ideas of Feng and Rao were incor-
porated into this approach in a series of papers that deal with similar
algorithms and refinements of these algorithms. A reference list of these
papers can be found in the introduction of chapter 3.

The reader that is interested in further details on the different decoding al-
gorithms is referred to the excellent survey paper by Pellikaan and Høholt
[35] and, of course, the original references.
Bibliography


Chapter 2

Error-locating pairs for cyclic codes*

Abstract  A general decoding method for linear codes is investigated for cyclic codes. The decoding consists of solving two systems of linear equations. All but four binary cyclic codes of length less than 63 can so be decoded up to their actual distance. A new family of codes is given for which the decoding needs only $O(n^2)$ operations.

Index Terms — Algebraic decoding, cyclic code, error-locating pair, MDS-code, QR-code.

---

I Introduction

The most successful methods for decoding of linear codes separate the decoding into the location of the error positions and the determination of the error values. Particular examples are the decoding of cyclic codes up to the BCH-bound and the basic algorithm for the decoding of algebraic-geometric codes. The methods allow a unified description that applies to any linear code. This was noticed by Pellikaan [17], who used it to describe the decoding of AG-codes. Independently but later, Kötter [12] gave a similar description. The location of the error positions is done with the help of an error-locating pair of vector spaces. To decode a particular linear code one has to assign such a pair to the code. We investigate how this can be done for cyclic codes. For a given error-locating pair, the decoding itself can be performed by solving two systems of linear equations.

We first recall the unified description. It applies to any linear code. Thus, it is presented with a minimum of assumptions and notation and the proofs can remain short. Section III investigates the determination of the error values. The later sections restrict to cyclic codes. In Sections IV and V we suggest general formats for error-locating pairs. Section VI treats complexity aspects of the decoding procedure. Section VII shows how in certain cases it is possible to extend the error-correction capability of error-correcting pairs. Section VIII gives pairs to decode all but four binary cyclic codes of length less than 63 up to their actual distance. In the last section, some sequences of codes are given with corresponding error-locating pairs.

II Error-locating pairs

A An error-locating procedure

The $n$-tuples defined over a field $\mathbb{F}$ form a vector space denoted by $\mathbb{F}^n$. For two vectors $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1})$ and $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$, we define a product $\mathbf{u} \star \mathbf{v} = (u_0v_0, u_1v_1, \ldots, u_{n-1}v_{n-1})$. For two subspaces $U, V \subset \mathbb{F}^n$, 
II. Error-locating pairs

Let $U \ast V$ denote the set of vectors $\{u \ast v : u \in U, v \in V\}$. For a linear code $C$ we denote the dimension by $k(C)$ and the minimum Hamming distance by $d(C)$, or by $k$ and $d$ respectively when no confusion arises. The following definition is crucial to treat the problem of error localization.

**Definition 2.1 ($t$-error-locating pair)** Let $U$, $V$ and $C$ be linear codes of length $n$ over the field $\mathbb{F}$. We call $(U, V)$ a $t$-error-locating pair for $C$ if the following conditions hold

\begin{align}
U \ast V &\subseteq C^\perp, \\ k(U) &> t, \\ d(V^\perp) &> t. 
\end{align}

Using this definition we will derive a $t$-error-locating procedure based on the following central observation.

**Theorem 2.1** Let $(U, V)$ be a $t$-error-locating pair for the code $C$. Let $y = c + e$ be a word in $\mathbb{F}^n$ with $c \in C$ and $e$ a vector of weight at most $t$. There exists a non-zero vector $u \in U$ such that

\begin{equation}
\sum_{i=0}^{n-1} y_i u_i v_i = 0, \quad \text{for all } v \in V. \tag{2.4}
\end{equation}

Moreover, any solution $u \in U$ of (2.4) satisfies

\begin{equation}
e \ast u = 0 \tag{2.5}
\end{equation}

**Proof.** In (2.4) we may replace $y$ by $e$ by condition (2.1) and the fact that $y_i = c_i + e_i$. Thus any vector $u \in U$ with property (2.5) is a solution to (2.4). Condition (2.2) guarantees the existence of a non-zero vector. This is because we impose at most $t$ linear conditions on $U$. To prove (2.5), we note that (2.4) has the equivalent formulation

\[ y \ast u \in V^\perp. \]

Again replacing $y$ by $e$ and using $\text{weight}(e) \leq t$ and condition (2.3) we find (2.5). \qed
Chapter 2. Error-locating pairs for cyclic codes

Assume we are given an error-locating pair \((U, V)\) and a received word \(y\). We have to find a solution \(u \in U\) to the homogeneous system of linear equations (2.4), which then by property (2.5) locates possible error positions with zeros. We will give the matrix defining this system. Let \(\text{diag}(y)\) denote the \(n \times n\) matrix which has the elements of \(y\) on its main diagonal and which is zero everywhere else. Equation (2.4) can thus be written as:

\[ v \cdot \text{diag}(y) \cdot u^T = 0, \quad \text{for all } v \in V. \]

Obviously, it is enough to consider a set of basis vectors in \(V\), forming a generator matrix \(G_V\) for \(V\) and we obtain

\[ G_V \cdot \text{diag}(y) \cdot u^T = 0. \]

To make this equation solvable with methods of linear algebra we replace \(u\) by \(u = \sigma G_U\), where \(G_U\) is a generator matrix for \(U\) and \(\sigma\) is an element of \(\mathbb{F}^{k(U)}\). Thus the key equation (2.4) can be rephrased as

\[ S(y) \cdot \sigma^T = 0, \quad \text{(2.6)} \]

where

\[ S(y) = G_V \cdot \text{diag}(y) \cdot G_U^T. \]

Any solution \(\sigma\) for (2.6) gives a solution \(u = \sigma G_U\) for (2.4) which now locates possible error positions with zeros. Thus we have described a \(t\)-error-locating procedure provided we have a \(t\)-error-locating pair. The problem of error-location is now to find the spaces \(U\) and \(V\) that satisfy conditions (2.1)-(2.3) for a maximal value of \(t\).

**Example 2.1** Let \(\mathbb{F}_7\) be the finite field with seven elements represented as the ring of integers modulo 7. Let \(C\) be a code of length 7 over \(\mathbb{F}_7\) with parity check matrix

\[
H_C = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
0 & 1 & 4 & 2 & 2 & 4 & 1 \\
0 & 1 & 1 & 6 & 1 & 6 & 6
\end{pmatrix}.
\]
The code $C$ is an extended Reed-Solomon code with minimum Hamming distance 5 and thus two error correcting. Let the codes $U$ and $V$ be defined by generator matrices

$$G_U = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 4 & 2 & 2 & 4 \end{pmatrix}, \quad G_V = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 \end{pmatrix}. $$

It is straightforward to verify that conditions (2.1) – (2.3) are satisfied for $t \leq 2$. Let a vector $y = c + e$ with $c \in C$ and $e = (0, 3, 0, 0, 1, 0, 0)$ be received. Matrix $S(y)$ equals $S(c) + S(e)$. Condition (2.1) guarantees that $S(c) = 0$, and we find

$$S(y) = \begin{pmatrix} 4 & 0 & 5 \\ 0 & 5 & 4 \end{pmatrix}.$$ 

A solution $\sigma$ to the key equation (2.6) is given by $\sigma = (4, 2, 1)$. This corresponds to a vector $u = \sigma G_U = (4, 0, 5, 5, 0, 4, 3)$ which is error-locating.

Remark 2.1 We are completely free in choosing bases for $U$ and $V$, i.e. in choosing the matrices $G_U$ and $G_V$, without affecting the space of solutions to the key equation. Nevertheless the choice of $G_U$ and $G_V$ determines the structure of the matrix $S(y)$. We will point out how this affects the computational complexity of solving (2.6) at a later stage.

Remark 2.2 Given a particular error vector $e$, it is clear from the proof of Theorem 2.1 that the following conditions are sufficient to obtain $u \in U \setminus 0$ with property (2.5):

$$C \ast U \subseteq V^\perp, \quad (2.7)$$

$$\exists u \in U \setminus 0 : e \ast u = 0, \quad (2.8)$$

$$\forall u \in U \setminus 0 : e \ast u \in V^\perp \Rightarrow e \ast u = 0. \quad (2.9)$$

Conditions (2.1) and (2.7) can both be phrased as

$$\sum_{i=0}^{n-1} c_i u_i v_i = 0, \quad \text{for all } c \in C, u \in U, v \in V.$$ 

Thus (2.7) is equivalent to (2.1). Conditions (2.8) and (2.9) are weaker than conditions (2.2) and (2.3) respectively due to the fact that they are formulated for a particular error pattern. We will have to refer to the weaker conditions in some cases where the conditions in Definition 2.1 are too strong.
Chapter 2. Error-locating pairs for cyclic codes

B Error-locating functions

Theorem 2.1 in the previous subsection gives a possibility to determine the error positions as zeros of a word \( u \in U \). This describes the general case. In some known algorithms, in particular for BCH-codes and AG-codes, an error-locating word \( u \) is associated in a natural way with an error-locating function. We will need this connection to make some properties of \( u \) and the corresponding error-locating function more transparent. Also the relation with functions is helpful in actually finding pairs \((U, V)\) because many codes have a clearer description in terms of functions. The rest of the section is devoted to this relation.

We have derived two sets of sufficient conditions for an error-locating pair. A pair with (2.1)-(2.3) locates all error patterns of a given weight. Such a pair is hard to find in general. Conditions (2.7)-(2.9) are weaker. They are formulated for a particular error pattern however and the verification for a large class of error patterns becomes cumbersome. We formulate a third set of conditions that can be seen as a compromise. The conditions depend on the positions of the errors but not on the particular error values.

Lemma 2.1 For an error vector \( e \), let \( E (\overline{E}) \) be the subspace of \( \mathbb{F}^n \) consisting of all vectors that have zero components in the error (non-error) positions. The following conditions are sufficient to locate the error positions with the pair \((U, V)\):

\[
\begin{align*}
C \ast U & \subseteq V^\perp \\
U \cap E & \neq 0 \\
V^\perp \cap \overline{E} & = 0.
\end{align*}
\]

Proof. The conditions imply (2.7)-(2.9). \qed

The conditions of the lemma can be expressed in terms of functions. We need the following.

Notation 2.1 For a field \( \mathbb{F} \) let \( S \) be the \( \mathbb{F} \)-algebra of \( n \)-tuples defined over \( \mathbb{F} \) with component-wise multiplication and addition. Let \( R \) be an
II. Error-locating pairs

\[ \mathbb{F}-\text{algebra with no zero-divisors such that there exists a surjective homomorphism } Ev : R \twoheadrightarrow S, \text{ with kernel } I. \]

For a code \( C \subset S \), let \( L(C) \subset R \) denote a \( \mathbb{F} \)-vector space such that the restriction of \( Ev \) to \( L(C) \) is a \( \mathbb{F} \)-vector space isomorphism from \( L(C) \) to \( C \). In particular \( L(C) \cap I = (0) \).

**Remark 2.3** \( R \) will be identified with a ring of functions. \( Ev \) is then the evaluation mapping, that means the evaluation of \( f \in R \) in a set of points. \( Ev \) induces naturally an \( \mathbb{F} \)-algebra isomorphism between \( R/I \) and \( S \).

**Example 2.2 (Example 2.1 continued)** In the situation of Example 2.1, we can identify \( R \) with \( \mathbb{F}_7[x] \), the ring of polynomials in \( x \) with coefficients from \( \mathbb{F}_7 \). \( Ev \) is the map that evaluates polynomials in points \( \{0, 1, 2, 3, 4, 5, 6\} \) and thus \( I \subset R \) is generated by \( x^7 - x \). The codes \( C, U \) and \( V \) can be defined as the images of the vector spaces of polynomials of degree at most 2, 2, and 1 respectively. Thus the vector \( u = (4, 0, 5, 5, 0, 4, 3) \) in Example 2.1 is the image of the polynomial \( x^2 + 2x + 4 = (x - 1)(x - 4) \).

**Example 2.3** For cyclic codes we take \( R = \mathbb{F}[x] \). Let \( \alpha \in \mathbb{F} \) be a primitive \( n \)-th root of unity. \( Ev \) is the evaluation map that evaluates polynomials in points \( \{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\} \), i.e.

\[ Ev(x) = (1, \alpha, \alpha^2, \ldots, \alpha^{n-1}). \]

The ideal \( I \subset R \) is generated by \( x^n - 1 \). We obtain the algebra isomorphism \( Ev : \mathbb{F}[x]/(x^n - 1) \overset{\sim}{\twoheadrightarrow} S \). For cyclic codes this differs from the well-known vector space isomorphism \( Id : \mathbb{F}[x]/(x^n - 1) \overset{\sim}{\twoheadrightarrow} S \), that identifies ideals in \( \mathbb{F}[x]/(x^n - 1) \) with cyclic codes in \( S \). For example:

\[ Ev(1 + x) = (1 + 1 + \alpha, 1 + \alpha^2, \ldots, 1 + \alpha^{n-1}), \]

\[ Id(1 + x) = (1, 1, 0, 0, \ldots, 0, 0). \]

**Example 2.4** For AG-codes an evaluation map \( Ev \) occurs in their definition [8].
Now, let an algebra-homomorphism $Ev : R \rightarrow S$ be given as in Notation 2.1. The following lemma is just a reformulation of Lemma 2.1 in terms of vector spaces of functions.

**Lemma 2.2**  Let $L(U), L(V^\perp)$ and $L(C)$ map to the codes $U, V^\perp$ and $C$ after evaluation. Let the maps be bijective as in Notation 2.1. For an error vector $e$, let $J(J)$ be the ideal in $R$ consisting of all elements that evaluate to zero at the error (non-error) positions. The following conditions are sufficient to locate the error positions with the pair $(U, V)$:

$$
L(C) \ast L(U) \subseteq L(V^\perp) + I, \\
L(U) \cap J \neq (0), \\
L(V^\perp) \cap J = (0).
$$

**Proof.**  Immediate from Lemma 2.1.  

The question arises whether an error-locating procedure can be formulated in terms of error-locating functions. This is indeed the case. The decoding procedures for BCH-codes [2, p.248] or AG-codes [11, 22] use this approach. $L(C), L(U)$ and $L(V^\perp)$ have here a natural interpretation.

### III  Error-correcting pairs

The previous section shows how an error-locating pair $(U, V)$ can be used to locate the error positions in a received word. This is the most important part of the decoding. Therefore error-locating pairs will play a major role in what follows. The key idea is that for a received word $y$, a vector $u$ can be obtained such that $u$ has zeros at the error positions.

**Remark 2.4**  The error vector $e$ satisfies the conditions

$$
y - e \in C, \\
e \ast u = 0.
$$

Thus $e$ can be obtained by solving a system of linear equations.
In general the vector \( u \) may have zeros at non-erroneous positions too. For the determination of the error values it is important that the set of zeros is not too large.

**Lemma 2.3**  
*For a code \( C \) with error-locating pair \((U, V)\), let \( u \in U \setminus 0 \) locate the error positions of the error vector \( e \), that is \( e * u = 0 \). The error values are uniquely determined by \( u \) if and only if*

\[
\forall c \in C : \quad c * u = 0 \quad \Rightarrow \quad c = 0. \tag{2.10}
\]

*Proof.* Assume we can write \( y \) in two different ways as \( y = e_1 + c_1 = e_2 + c_2 \), where \( c_1, c_2 \in C \) and \( e_1 * u = e_2 * u = 0 \). It follows that

\[
(e_1 - e_2) * u = 0, \quad \text{with} \quad e_1 - e_2 = c_2 - c_1 \in C.
\]

Condition (2.10) implies \( e_1 = e_2 \). If this condition fails, say \( c * u = 0 \) for \( c \neq 0 \), we find the two different solutions \( e, e - c \).

Lemma 2.3 is the basis for the definition of a \( t \)-error-correcting pair in [17]. See also [12].

**Definition 2.2 (\( t \)-error-correcting pair)**  
Let \((U, V)\) be a \( t \)-error-locating pair for the code \( C \) as in Definition 2.1. We call \((U, V)\) a \( t \)-error-correcting pair for the code \( C \) if in addition to the conditions (2.1),(2.2) and (2.3) the following is satisfied

\[
d(C) + d(U) > n, \tag{2.11}
\]

where \( n \) denotes the code length of \( C \).

**Remark 2.5**  
The definition is justified by the lemma since condition (2.11) implies

\[
\forall c \in C, \forall u \in U : \quad c * u = 0 \quad \Rightarrow \quad c = 0 \lor u = 0, \tag{2.12}
\]

which implies (2.10). In some cases we will prefer to use the weaker condition (2.12).
Chapter 2. Error-locating pairs for cyclic codes

**Remark 2.6** Recall that a pair $(U, V)$ needs to satisfy $C \ast U \subseteq V^\perp$ to be error-locating for a code $C$. By the lemma, an error-locating pair will be error-correcting if it satisfies

$$(C \setminus 0) \ast (U \setminus 0) \subseteq (V^\perp \setminus 0).$$

**Remark 2.7** In terms of functions a pair $(U, V)$ needs to satisfy $L(C) \ast L(U) \subseteq L(V^\perp) + I$ to be error-locating. By Lemma 2.3 it will be error-correcting if it satisfies

$L(C) \ast L(U) \subseteq L(V^\perp).$

Here we use the fact that $R$ has no zero-divisors and that $L(V^\perp) \cap I = (0)$. Let $\langle L(C) \ast L(U) \rangle$ denote the linear space spanned by all functions in $L(C) \ast L(U)$. The following conditions are then sufficient to guarantee error-correction with a pair $(U, V)$:

\begin{align*}
L(U) \cap J &\neq (0), \quad (2.13) \\
\langle L(C) \ast L(U) \rangle \cap \overline{J} &\neq (0). \quad (2.14)
\end{align*}

The dilemma in algebraic decoding is obvious. For (2.13) we want $L(U)$ to be big and for (2.14) we want $\langle L(C) \ast L(U) \rangle$ that means $L(U)$ to be small.

**IV Cyclic codes**

**A Notation**

From now on we restrict our attention to the class of cyclic codes. A cyclic code $C \subseteq \mathbb{F}^n$ is usually identified with an ideal in the ring $\mathbb{F}[x]/(x^n - 1)$ generated by a polynomial $g(x)$, which divides $x^n - 1$. A codeword $c = (c_0, c_1, \ldots, c_{n-1}) \in C$ is interpreted as a polynomial by the relation

$$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}, \quad \text{with } g(x) | c(x).$$

The code is determined by the zeros of $g(x)$. We assume $(\text{char } \mathbb{F}, n) = 1$, so that $x^n - 1$ has $n$ different zeros. Let the extension $\overline{\mathbb{F}}$ of $\mathbb{F}$ contain the $n$-th
roots of unity and let $\alpha \in \overline{F}$ be a primitive $n$-th root of unity. Let $m_i(x)$ be the minimal polynomial of $\alpha^i$ over $F$. If $g(x)$ equals $\text{lcm}\{m_i(x) : \alpha^i \in R\}$ then we call $R$ a defining set for $C$. If $R$ is the maximal defining set for $C$ we call $R$ complete. By abuse of standard notation we will describe the defining set by the exponents occurring in $R$. With $R = \{i_1, i_2, \ldots, i_l\}$ the matrix

$$M(R) = \begin{pmatrix}
(\alpha^{i_1})^0 & (\alpha^{i_1})^1 & \cdots & (\alpha^{i_1})^{n-1} \\
(\alpha^{i_2})^0 & (\alpha^{i_2})^1 & \cdots & (\alpha^{i_2})^{n-1} \\
\vdots & \vdots & \ddots & \vdots \\
(\alpha^{i_l})^0 & (\alpha^{i_l})^1 & \cdots & (\alpha^{i_l})^{n-1}
\end{pmatrix}$$

is a parity-check matrix for a code $C \subset \overline{F}^n$. The code $C$ is obtained as the subfield subcode of $\overline{C}$, that means $C = C \cap \overline{F}^n$. In order to distinguish codes over fields $F$ and $\overline{F}$ we use the notation $C/F$ and $C/\overline{F}$ respectively.

**Definition 2.3** Let $R$ be a defining set of a cyclic code $C/F$. $C$ is then defined as

$$C = \{c \in \overline{F}^n : M(R)c^T = 0\}.$$

We also like to refer to a matrix $M(R)$ as a generator matrix to describe the codes $U$ and $V$ in an error-locating pair $(U, V)$. To distinguish between the use of $M(R)$ as a parity-check matrix or as a generator matrix, we call $R$ a defining set in the former case and a generating set in the latter case. As it will be seen the notation of generating sets is very convenient to describe error-locating pairs for cyclic codes. We formally have the following definition.

**Definition 2.4** Let $I$ be a generating set of a cyclic code $U/\overline{F}$. $U$ is then defined as

$$U = \{u \in \overline{F}^n : u = \sigma M(I), \sigma \in \overline{F}^{|I|}\}.$$

In the following, generating sets for the codes $U$ and $V$ will be denoted by $I$ and $J$ respectively. We stress that both codes are defined over the large field $\overline{F}$. Thus, their dimensions follow immediately as $k(U) = |I|$ and
Chapter 2. Error-locating pairs for cyclic codes

Let \( k(V) = |J| \). Let \( I = \{i_1, \ldots, i_t\} \), where \( i_1 < \ldots < i_t \). We define \( T \) as the smallest set of consecutive integers containing \( I \), i.e.

\[
T = \{i_1, i_1 + 1, \ldots, i_t - 1, i_t\}.
\]

The following reformulation of the BCH-bound is obvious.

**Lemma 2.4** The minimum distance of a cyclic code of length \( n \) with generating set \( I \) is bounded below by

\[
d \geq n - |T| + 1.
\]

We will freely use the following observations. Let

\[
a(i) = (1, \alpha^i, \alpha^{2i}, \ldots, \alpha^{(n-1)i}).
\]

We have

\[
a(i) \perp a(j) \iff i + j \not\equiv 0 \pmod{n}.
\]

Let \( b + cR = \{b + ci \pmod{n} : i \in R\} \). The codes with defining sets \( R \) and \( b + cR \) are equivalent, for \( (c, n) = 1 \). Also let \( I + J = \{i + j \pmod{n} : i \in I, j \in J\} \). Let \( U, V \) and \( W \) be cyclic codes with generating sets \( I, J \) and \( I + J \) respectively. Then \( U \ast V \) is a subset of \( W \).

**B Decoding BCH-codes**

Theorem 2.1 provides a way to construct a decoding algorithm for a particular linear code. The bottle-neck in the construction is the search for an error-correcting pair \((U, V)\). As we shall see, there exists an obvious choice which enables us to decode up to the BCH-bound. We can do better however by using a correspondence between the pair of cyclic codes \((U, V)\) and the pair of defining sets \((A, B)\) in the lemma below.
IV. Cyclic codes

Lemma 2.5 (Roos-bound) (Theorem 3 [13]) If $A$ is a defining set for a cyclic code with minimum distance $d_A$ and if the set $B$ is such that $|B| \leq |B| + d_A - 2$, then the code with defining set $A + B$ has minimum distance $d \geq |B| + d_A - 1$.

Proof. After replacing $A$ and $B$ by sets of zeros see [13].

Corollary 2.1 Let $c_1$ and $c_2$ satisfy $(c_1, n) = (c_2, n) = 1$. In Lemma 2.5, the same bound on the distance holds for a code with defining set $c_1A + c_2B$.

Proof. First it is immediate from the proof in [13] that the constants play no essential role and can be taken equal to one. Also, we may restrict to the case $c_2 = 1$ by passing to an equivalent code. The lemma can now be applied with the sets $c_1A$ and $B$.

The following theorem relies on the lemma and provides a possibility to find error-locating pairs for a large class of cyclic codes.

Theorem 2.2 Let $s < t$. Let the generating sets $I, J$ and $K$ satisfy

\[
|I| = t + 1, \\
|J| = t - s, \\
|K| = s + 1, \\
|\mathcal{T}| = t - s, \\
|\mathcal{R}| \leq t.
\]

Let $(c_1, n) = (c_2, n) = (c_3, n) = 1$. Then the code $C/\mathbb{F}$ with defining set $R = b + c_1I + c_2J + c_3K$ has a $t$-error-locating pair $(U, V)$, where $U/\mathbb{F}$ is defined by the generating set $b + c_1I$ and $V/\mathbb{F}$ by the generating set $c_2J + c_3K$. For the distance of the code $C$ we have

\[
|\mathcal{T}| \leq 2t \Rightarrow d(C) \geq 2t + 1.
\]

The pair $(U, V)$ is $t$-error-correcting whenever

\[
|\mathcal{T}| \leq d(C).
\]
Proof. The verification of conditions (2.1) and (2.2) is straightforward. The distance \(d(V^\perp)\) can be estimated with the lemma. We use it with
\[
A = J, \quad d_A = t - s + 1 \quad \text{and} \quad B = K, \quad |B| \leq (s + 1) + (t - s + 1) - 2,
\]
and apply the corollary. It follows that \(d(V^\perp) \geq (s + 1) + (t - s + 1) - 1 = t + 1\) and (2.3) holds. The distance \(d(C)\) follows with another application of the lemma, this time with
\[
A = c_2 J + c_3 K, \quad d_A \geq t + 1 \quad \text{and} \quad B = bc_1^{-1} + I, \quad |B| \leq (t + 1) + (t + 1) - 2.
\]
Using the corollary we see that \(d(C) \geq (t + 1) + (t + 1) - 1 = 2t + 1\).

Example 2.5 (Example 1 [6]) Let \(C\) be the binary cyclic code of type \([39,15]\) defined by \(R \supseteq \{1,3\}\). In particular
\[
R \supseteq \{1,2,3,4,5,6,8,9,10,11,12\}.
\]
The BCH-bound yields \(d \geq 7\) (the Hartmann-Tzeng bound or the Roos bound do not improve on this). The actual distance equals 10 ([19],[13]). In the theorem we may choose \(I = \{1,2,3,8,9\}, J = \{0,1,2,3\}, K = \{0\}\) with \(t = 4\) and \(s = 0\). Since \(|I| > 2t\), the theorem does not yield a better estimate for the distance \(d(C)\). Using the knowledge that the distance is equal to 10, we see that \(|I| \leq d(C)\) and that the 4-error-locating pair \((U,V)\) is actually 4-error-correcting. The code \(C\) corresponds to entry 45 in Table 1.

A procedure to decode up to the Hartmann-Tzeng bound and in some cases up to the Roos bound is presented in [5]. We recall the two bounds, following [5], and show that the procedure is a special case of Theorem 2.2. Let the defining set for a cyclic code \(C\) contain \(b + c_1 I^* + c_2 J^*\), with \(I^* = \{1,2,\ldots,d_0 - 1\}\) and \(J^* = \{j_1,j_2,\ldots,j_{s+1}\}\), for \(j_1 < j_2 < \ldots < j_{s+1}\) and \(j_{s+1} - j_1 - s < d_0 - 1\). Also, let \((c_1,n) = (c_2,n) = 1\). Then
\[
d(C) \geq d_{\text{Roos}} = d_0 + s.
\]
The bound is a special case of Lemma 2.5. With the further restrictions

\[ s + 1 \leq d_0 - 1, \tag{2.15} \]
\[ j_{s+1} - j_1 < \frac{(d_0 + s - 1)}{2}. \tag{2.16} \]

the procedure in [5] decodes up to \( d_{Roos} \). Note that the Hartmann-Tzeng bound corresponds to \( j_h = h, h = 1, 2, \ldots, s + 1 \) and in this case the restrictions can always be fulfilled.

**Corollary 2.2 (Theorem 4 and Theorem 5 [5])** In decoding up to the Roos-bound, we may assume that \( s \) is such that \( d_{Roos} \) is odd and we write \( d_0 + s = 2t + 1 \). Let the generating sets \( I, J, K \) be defined as

\[ I = \{0, 1, 2, \ldots, t\}, \]
\[ J = \{1, 2, \ldots, t - s\}, \]
\[ K = \{j_1, j_2, \ldots, j_{s+1}\}. \]

The code \( C \) with defining set \( R = b + c_1I + c_1J + c_2K \) has distance \( d(C) \geq 2t + 1 \). A \( t \)-error-correcting pair is given by \( (U, V) \), where \( U \) has generating set \( b + c_1I \) and \( V \) has generating set \( c_1J + c_2K \).

**Proof.** The restriction (2.15) can be written as \( s < t \). In particular the set \( J \) is well-defined. The restriction (2.16) yields \( |K| - 1 < t \). Thus all conditions of the theorem are fulfilled. \( \square \)

### C Recurrences

Some error-locating pairs in Theorem 2.2 do not satisfy the condition \( |T| \leq d(C) \). In that case an error-locating word \( u \in U \) is obtained but it is not immediately clear whether the error values are uniquely determined or not. To investigate this we use the following lemma.

**Lemma 2.6** Let the vector \( e \) have support in a set \( A \) and let \( R \) contain \( |A| \) consecutive integers. Let the syndromes

\[ S_i = \langle e, a(i) \rangle, \quad i \in R, \tag{2.17} \]
be known. Then the set of equations (2.17) determines \( e \) uniquely.

**Proof.** By the BCH-bound the difference of two solutions for \( e \) has weight at least \( |A| + 1 \) or zero. Hence, two solutions with support in \( A \) are identical. An efficient way to solve for \( e \) is given by Forney’s algorithm [19, p.297].

Recall from the computational scheme that in solving (2.4) for \( \mathbf{u} \in U \setminus \{0\} \) we actually find a vector \( \sigma \) solving the key equation (2.6). With \( I = \{i_1, \ldots, i_t\} \) as generating set for \( U \) the generator matrix of \( U \) is given by \( G_U = M(I) \). We are in the situation of Example 2.3 and for \( \sigma = (\sigma_{i_1}, \ldots, \sigma_{i_t}) \) we have

\[
\mathbf{u} = \sigma G_U = E^U(\sigma) \quad \text{for the polynomial}
\]

\[
\sigma = \sigma_{i_1}X^{i_1} + \ldots + \sigma_{i_t}X^{i_t} + \sigma_{i_1}X^{i_1}.
\]  

(2.18)

Having found an error-locating polynomial \( \sigma \), the zeros of \( \mathbf{u} \) are obtained as the zeros of \( \sigma \) by e.g. a Chien search. We denote this set with \( A \). The following lemma provides a method to obtain a consecutive syndrome set of size \( |A| \).

**Lemma 2.7** Let the polynomial \( \sigma \) (2.18) have the support of \( e \) among its zeros. Then

\[
\sigma_{i_1} S_{i_1+j} + \ldots + \sigma_{i_t} S_{i_t+j} + \sigma_{i_1} S_{i_1+j} = 0,
\]  

(2.19)

for all integers \( j \).

**Proof.**

\[
\begin{align*}
\sigma_{i_1} S_{i_1+j} + \ldots + \sigma_{i_t} S_{i_t+j} \\
= \langle \mathbf{e}, \sigma_{i_1} \mathbf{a}(i_1 + j) + \ldots + \sigma_{i_t} \mathbf{a}(i_t + j) \rangle \\
= \langle \mathbf{e}, (\sigma_{i_1} \mathbf{a}(i_1) + \ldots + \sigma_{i_t} \mathbf{a}(i_t)) \ast \mathbf{a}(j) \rangle \\
= \langle \mathbf{e}, \mathbf{u} \ast \mathbf{a}(j) \rangle = \langle \mathbf{e} \ast \mathbf{u}, \mathbf{a}(j) \rangle = 0.
\end{align*}
\]
Example 2.6  We consider the binary cyclic code $C$ of type $[45,15]$ with $R \supseteq \{1,3,7,15\}$. It corresponds to entry 84 in Table 1. We have

$$R \supseteq \{1,2,3,4,11,12,13,14,15,16,17,28,29,30,31\}.$$ 

The BCH-bound gives $d \geq 8$, while the actual distance $d = 9$ and four errors can be corrected. The choice

$$I = \{1,11,12,13,14\}, \quad J = \{0,1,2,3\},$$

defines an error-locating pair $(U,V)$ by Theorem 2.2. We can therefore compute

$$\sigma = \sigma_{14}X^{14} + \sigma_{13}X^{13} + \sigma_{12}X^{12} + \sigma_{11}X^{11} + \sigma_1X,$$

such that the error positions are zeros of $\sigma$. We may assume that there are four errors and therefore that $\sigma_1 \neq 0$. Using (2.19) with $j = 17$ we find $S_{18}$. The syndrome $S_5$ is then obtained with $j = 4$. By then $S_1, S_2, \ldots, S_{20}$ are all known and Lemma 2.6 applies.

Remark 2.8  When using recurrences of type (2.19) we need to know which syndromes have a nonzero coefficient. In general there can be zero coefficients and these cases have to be treated separately. As in the example, one may be able to show that some coefficients cannot be zero. The procedure is described in [6], but there zero coefficients are not considered. Thus, the procedure as described in [6] may fail for the entries 17,84 and 121 in Table 1.

Example 2.7  We consider the code $C$ of Example 2.6. The procedure in [6] corresponds to the choices

$$I = \{11,12,13,14,28\}, \quad J = \{0,1,2,3\}.$$ 

This defines an error-locating pair. In the case of four errors there will be a unique error-locating polynomial. The polynomial $X^{28} - X^{13}$ has fifteen zeros among the 45-th roots of unity. The zeros support a two-dimensional subcode of $C$ and the error values are not uniquely determined from the error positions. In fact, the unknown syndromes $\{S_5, S_{10}, S_{20}, S_{40}, S_{35}, S_{25}\}$ cannot be obtained from the known syndromes with a recurrence $S_{15+j} = S_j$. 


D Correcting more errors

Theorem 2.2 gives an error-correcting pair \((U, V)\) to correct errors up to the BCH-bound and in some cases beyond. To achieve the error-correction capability of some cyclic codes we recall a well-known ‘trick’. Considering binary cyclic codes, \(S_0\) has value either 0 or 1.

**Remark 2.9** Let the binary cyclic code \(C\) have distance \(d(C) \geq 2t + 1\). A \(t\)-error-correcting algorithm for the even weight subcode becomes a \(t\)-error-correcting algorithm for the code itself when used twice with two different values of \(S_0\).

**Example 2.8** We consider the cyclic code of length 33 with defining set \(R = \{1, 3\}\). The complete defining set contains the set \([-4, -3, -2, -1, 1, 2, 3, 4]\). The actual distance is equal to 10 and the even weight subcode can be decoded up to this distance with the pair \((U, V)\) defined with generating sets \(I = \{0, 1, 2, 3, 4\}\) and \(J = \{-3, -2, -1, 0\}\).

Feng and Tzeng [5] showed how the trick can be applied with reduced complexity. We recall briefly their argument applied to error-correcting pairs. In many cases we find error-correcting pairs, such that \(S_0\) occurs just once in the key matrix \(S(y)\) (2.6). Without loss of generality we can assume that \(S_0\) occurs in the last column. Let \(t\) be the maximal number of errors that we want to decode. If less than \(t\) errors have occurred we can find an error-locating polynomial \(\sigma\) from the leftmost columns, i.e. with vanishing coefficient at the last column. We only need to know \(S_0\) if we cannot find such a solution. But then by assumption precisely \(t\) errors have occurred which means \(S_0\) is equal to \(t \mod 2\).

**Example 2.9** We consider the \([31, 16, 7]\) binary cyclic code with defining set \(R = \{1, 5, 7\}\). An error-correcting pair is described by generator matrices \(G_V = M(\{1, 2, 0\})\) and \(G_U = M(\{7, 8, 18, 0\})\). The key equation is then given by

\[
\begin{pmatrix}
S_8 & S_9 & S_{19} & S_1 \\
S_9 & S_{10} & S_{20} & S_2 \\
S_7 & S_8 & S_{18} & S_0
\end{pmatrix}
\begin{pmatrix}
\sigma_7 \\
\sigma_8 \\
\sigma_{18} \\
\sigma_0
\end{pmatrix} = 0.
\]
If less than 3 errors occurred, we will find a vector $\sigma$ with $\sigma_0 = 0$ which locates the error positions. If we cannot find such a vector we assume three errors to have occurred and this means $S_0$ is equal to 1. So whenever $S_0$ is needed in order to calculate the error-locator polynomial, we know its value.

V  Pairs from MDS-codes

A  A class of MDS-codes

In this section we assume that the field $\mathbb{F}$ is finite of order $q$. Also let $(n, q) = 1$. As in the previous section let $\mathbb{F} \supset \mathbb{F}$ contain the $n$-th roots of unity.

**Theorem 2.3**  Let $C$ and $A$ denote two cyclic codes over $\mathbb{F}$ of length $n$. Let their defining sets be given by

$$R_C = \{1, q^l, q^{2l}, \ldots, q^{rl}\}, \quad \text{and} \quad R_A = \{1, q^l, q^{2l}, \ldots, q^{sl}\},$$

for $l, r, s > 0$ with $r < s$. Then

$$d(C) = \min \{ |R_C| + 1, d(A) \}.$$ 

In particular the code $C$ is MDS for $|R_C| < d(A)$.

**Proof.**  Clearly $d(C) \leq |R_C| + 1$ by the Singleton bound. It suffices to prove for a word $c \in C$ of weight $\text{wt}(c) \leq |R_C|$ that $c \in A$. The columns in $M(R_C)$ corresponding to the support of $c$ are dependent. Thus the submatrix of $M(R_C)$ formed by these columns has row-rank less than $|R_C|$. The submatrix of $M(R_A)$ formed by columns at the support of $c$ has a linear relation among its top $|R_C|$ rows, because this is just the corresponding submatrix of $M(R_C)$. Taking coefficients and rows to the power $q^l$ yields a relation on lower rows and it is seen that the two submatrices have the same row-space, which implies $c \in A$.  

**Remark 2.10**  The conclusion and the proof of the theorem remain the same when zero is added to the defining sets $R_C$ and $R_A$. 


Example 2.10  The code $C$ over $\mathbb{F} = GF(2^{11})$ of length $n = 23$ with defining set $R_C = \{0, 1, 4, \ldots, 4^r\}$ is MDS for $r \leq 5$, due to the fact that a code with defining set $R_A = \{0, 1, 4, \ldots, 4^{10}\}$ has minimum distance 8.

B  Construction of pairs

For a $t$-error-correcting BCH-code $C$ with defining set $R \supseteq \{1, 2, \ldots, 2t\}$ we have by Theorem 2.2 a $t$-error-correcting pair $(U, V)$. The codes $U$ and $V$ have generating sets $I = \{0, 1, \ldots, t\}$ and $J = \{1, 2, \ldots, t\}$ respectively. More general, we have

\textbf{Proposition 2.1}  Let the codes $U$ and $V$ be MDS of dimension $k(U) = t + 1$ and $k(V) = t$ respectively. A code $C$ with $C \perp U \ast V$ has distance $d(C) \geq 2t + 1$. Moreover it has the $t$-error-correcting pair $(U, V)$.

\textit{Proof.} Theorem 5 in [13] yields that $d(C) \geq 2t + 1$. The conditions (2.1)–(2.3) and (2.11) for an error-correcting pair follow immediately.  \hfill \Box

\textbf{Remark 2.11}  In case the code $U$ is not MDS, but otherwise the conditions on $U$ and $V$ are satisfied the pair $(U, V)$ in the lemma is still $t$-error-locating for a code $C$ with $C \perp U \ast V$.

We give two applications of MDS codes obtained with Theorem 2.3. In both cases, $U$ and $V$ are chosen such that the condition $C \perp U \ast V$ leads to a small defining set for $C$. Furthermore the key-equation (2.6) can be solved with complexity $O(t^2)$ in both cases. Here the complexity is estimated by the number of required multiplications in the field $\mathbb{F}$.

\textbf{Theorem 2.4} (first conjugacy format)  Let the codes $U/\overline{F}$ and $V/\overline{F}$ have generating sets $I = \{1, q^t, q^{2t}, \ldots, q^{lt}\}$ and $J = \{0, q^t, q^{2t}, \ldots, q^{(l-1)t}\}$, for $t \geq 2$. Let $t = t_i$ be maximal such that both $U$ and $V$ are MDS of dimension $k(U) = t + 1$ and $k(V) = t$ respectively. For $2 \leq t \leq t_i$, a code $C/\overline{F}$ with

$$R_C \supset \{1, 2, q^t + 1, q^{2t} + 1, \ldots, q^{(l-1)t} + 1\}$$
has the t-error-correcting pair \((U, V)\). The key equation (2.6) can be solved with complexity \(O(t^2)\).

**Proof.** The value of \(t_i\) can be obtained with Theorem 2.3. It is straightforward to verify that the complete defining set \(R\) for \(C\) satisfies \(R \supseteq I + J\) and thus \(C \perp U \ast V\). We may use Proposition 2.1. For the solving of the key equation see Section VI. \(\square\)

**Theorem 2.5 (second conjugacy format)** Let the codes \(U/\mathbb{F}^r\) and \(V/\mathbb{F}^r\) have generating sets \(I = \{0, 1, q^{2^l}, q^{4^l}, \ldots, q^{(2^l-2^t)}\}\) and \(J = \{0, q^l, q^{2^l}, \ldots, q^{(2^l-3^t)}\}\), for \(t \geq 2\). Let \(t = t_i\) be maximal such that both \(U\) and \(V\) are MDS of dimension \(k(U) = t + 1\) and \(k(V) = t\) respectively. For \(2 \leq t \leq t_i\), a code \(C/\mathbb{F}^r\) with

\[
R_C \supseteq \{0, 1, q^l + 1, q^{2^l} + 1, \ldots, q^{(2^l-3^t)} + 1\}
\]

has the t-error-correcting pair \((U, V)\). The key equation (2.6) can be solved with complexity \(O(t^2)\).

**Proof.** As in Theorem 2.4. \(\square\)

**Example 2.11** We consider codes of length \(n = 23\) over \(GF(2^{11})\). Let \(U\) and \(V\) be as in Theorem 2.5 with \(q = 2, l = 1, t = 3,\) or \(I = \{0, 1, 4, 16\}\) and \(J = \{0, 2, 8\}\). By Example 2.10 both \(U\) and \(V\) are MDS and we have found a 3-error-correcting pair for the even weight subcode \(C\) of the binary Golay code, since \(R_C \supseteq \{0, 1, 3, 9\}\).

**Lemma 2.8 (recurrences)** If a pair \((U, V)\) is t-error-locating and the generating sets \(I\) and \(J\) are of a conjugacy format, then the syndromes \(S_i = \langle e, a(i) \rangle\) can be determined for

\[
i = q^{sl} + 1, \quad \text{for } s \geq 1, \quad \text{first format (Theorem 2.4).}
\]

\[
i = q^{(2s-1)y} + 1 \quad \text{for } s \geq 1, \quad \text{second format (Theorem 2.5).}
\]
Proof. The case $s < t$ is obvious. For $s \geq t$ we use induction. For both formats we may assume that the error-locating word $u = \sum_{i \in I} \sigma_i a(i)$ has non-zero coordinate $\sigma_1$ at $a(1)$. We have, for the first format,

\[
0 = \langle e \ast u, a(q^l) \rangle \\
= \langle e, u \ast a(q^l) \rangle \\
= \sigma_1 \langle e, a(q^l + 1) \rangle + \text{known terms}.
\]

Similarly for the second format. \(\Box\)

Example 2.12 We consider codes of length $n = 39$ over $GF(2^{12})$. Let $U$ and $V$ be as in Theorem 2.4 with $q = 2, l = 1, t = 4$, or $I = \{1, 2, 4, 8, 16\}$ and $J = \{0, 2, 4, 8\}$. The code $V$ is MDS and we have found a 4-error-locating pair for the binary code $C$ with $R_C \supseteq \{1, 3, 5, 9\}$. It is of type $[39, 15, 10]$. By the lemma we can determine syndromes corresponding to the checks $17$ and $65 \equiv 26 \pmod{39}$ and the error values can be determined by Lemma 2.6.

VI Complexity

A Short description of the Fundamental Iterative Algorithm

In their paper [5], Feng and Tzeng proposed a fundamental iterative algorithm (FIA). For a matrix $A = \| A_{i,j} \|$, it gives the minimal set of dependent leading columns. It basically solves an arbitrary homogeneous system of linear equations and contains the Berlekamp-Massey algorithm as a special case. The algorithm is not required to make our decoding procedure work, but it seems to be a key algorithm in treating complexity aspects. We recall it in a form that allows us to complete the proof of Theorem 2.4 and Theorem 2.5.

Whenever it is necessary for reasons of dimension, we extend a vector with a suitable number of zeros. Let the matrix $A^{(a,b)}$ be the submatrix of $A$ consisting of the elements in the first $a$ rows and the first $b$ columns of $A$. For a
fixed \( b \), we consider column vectors \( \sigma \) with non-zero coordinate at position \( b \) that solve the equation
\[
A^{(a,b)} \sigma = 0.
\]
Let \( a = a^{(b)} \) be maximal such that a solution exists and let \( \sigma = \sigma^{(b)} \) be such a solution. To assure that a solution exists we use the convention \( A^{(0,b)} = 0^T \). For these \( a \) and \( \sigma \), let \( \Delta^{(b)} \) be defined as
\[
\Delta^{(b)} = \sum_{k=1}^{b} A_{a+1,k} \sigma_k,
\]
or as \( \Delta^{(b)} = 0 \) when \( A \sigma^{(b)} = 0 \). For given \( \{ (\sigma^{(b)}, \Delta^{(b)}, a^{(b)}) \}_{b<i} \) the idea of the FIA is now to calculate \( \sigma^{(i)} \) with help of the \( \sigma^{(b)} \), \( b < i \). Starting with any vector \( \sigma \) of length \( i \) and \( \sigma_i \) unequal to zero, this is achieved by subtracting suitable scalar multiples of the known \( \sigma^{(b)} \) from \( \sigma \) thereby obtaining a new \( \sigma \). More precisely, whenever \( \sigma \) solves
\[
A^{(i,j)} \sigma = 0,
\]
\[
d = \sum_{k=1}^{i} A_{j+1,k} \sigma_k \neq 0,
\]
and there exists a triple \( (\sigma^{(b)}, \Delta^{(b)}, j) \), we construct
\[
\sigma \leftarrow \sigma - \frac{d}{\Delta^{(b)}} \sigma^{(b)},
\]
which now solves
\[
A^{(j+1,i)} \sigma = 0.
\]
Finally we will obtain the triple \( (\sigma^{(i)}, \Delta^{(i)}, a^{(i)}) \). Constructing triples in the above way leads to a vector \( \sigma \) which solves the system \( A \sigma = 0 \). For details and proofs see [5].

Let us assume that \( A \) is a Hankel matrix. By starting the calculation of \( \sigma^{(i)} \) with a particular choice for \( \sigma \), namely a shifted version of \( \sigma^{(i-1)} \) with zero in the lowest position, we get the well-known Berlekamp-Massey algorithm. The calculations of most \( d \) are not necessary — they are zero or already known by the structure of Hankel matrices. This is the crucial point in
saving complexity. In the next section we will show how to apply the FIA to matrices $S(y)$ obtained with Theorem 2.4 and Theorem 2.5. It turns out that solving the linear systems described by these matrices is achieved with basically the same complexity as used for the Berlekamp-Massey algorithm.

\section{Reducing complexity}

In general, the complexity of the procedure described in Section II equals $O(n^3)$. This is due to the fact that only matrix inversions and multiplications are involved. We found two possibilities to improve on the number of computations. One approach uses regular structures of the matrix $S(y)$ and the other approach reduces the size of the field that contains the entries of $S(y)$.

In the case of the conjugacy format, the matrix $S(y)$ has a highly regular structure. We explicitly treat the first conjugacy format. For the second conjugacy format similar considerations hold. We write the generating sets defining $U$ and $V$ for the first conjugacy format in the following ordered form

\[ J = \{q^{(t-1)}, q^{(t-2)}, q^{(t-3)}, \ldots, q^t\}, \quad I = \{1, q^t, q^{2t}, \ldots, q^{ut}\}, \]

excluding the zero in $J$. We recall the definition of $S_i$ given in Lemma 2.6.

\[ S_i = \langle e, a(i) \rangle, \quad i \in R. \]

Obviously $S_i = \langle y, a(i) \rangle$ holds for all $i \in R$.

\textbf{Lemma 2.9} With generator matrices for $U$ and $V$ corresponding to the above ordering, the entries in $S(y)$ satisfy

\[ S(y)_{j,i} = (S(y)_{j+1,i-1})^{q^t}, \quad j = 1, \ldots, t-2, \quad i = 2, \ldots, t+1. \]

\textbf{Proof.} The entry $S(y)_{j,i}$ is equal to the syndrome $S_{h(j,i)}$ with

\[ h(j, i) = q^{(t-j)} + q^{(i-1)}. \]

We see that $h(j, i)$ is equal to $q^t h(j+1, i-1)$. The vector $y$ was assumed to be defined over a field $\mathbb{F}$ of cardinality $q$. Hence $S_{qh} = S_{q}^{h}$ and the lemma follows. \hfill $\square$
VI. Complexity

We see from Lemma 2.9 that the format of $S(y)$ is very similar to a Hankel matrix. Thus to find $S(y)$ we only have to calculate the entries in the first row and the last column. The other entries are found using the lemma. This structure is now used in finding the space of solutions to the key equation in the same way as in the Berlekamp-Massey algorithm. Recall that the complexity gain in using the Berlekamp-Massey algorithm was due to the fact that given a vector $\sigma^{(b)}$, which solves equation

$$A^{(a,b)} \sigma = 0,$$

we find a vector $\sigma$ that solves equation

$$A^{(a-1,b+1)} \sigma = 0$$

as a shifted version of $\sigma^{(b)}$ with zero in the lowest position.

**Proposition 2.2** Let $S(y)$ be the $(t-1) \times (t+1)$-matrix of Lemma 2.9. Solving

$$S(y) \sigma = 0$$

for $\sigma$ can be done with complexity $\mathcal{O}(t^2)$.

**Proof.** Consider a typical step in the FIA. Given a solution $\sigma^{(i-1)}$ to the equation $S(y)^{(a,i-1)} \sigma = 0$, we also have a solution to the equation $S(y)^{(a-1,i)} \sigma = 0$. The latter solution is obtained by taking all elements in $\sigma^{(i-1)}$ to the $q^j$-power and shifting them by one position. Using normal bases for the field, raising a number to the $q^j$-th power can be performed by a cyclic shift, not requiring computational complexity. Whenever possible, we now perform an update of $\sigma$. This is done by performing the following operation

$$\sigma \leftarrow \sigma - \frac{d}{\Delta^{(b)}} \sigma^{(b)}$$

with $d = (\Delta^{(i-1)}) q^j$,

and the calculation of a new $d$. The whole step requires at most $\mathcal{O}(t)$ operations and we find a solution to $S(y)^{(a,i)}$. Thus in every complexity demanding step, starting from a solution to the system $S(y)^{(a,b)}$ we find a solution to the system $S(y)^{(a',b')}$ such that $a' + b'$ is equal to $a + b + 1$. On the other hand $a' + b'$ is bounded by $2t$ which is the sum of the number of rows and the number of columns in $S(y)$. So we have to perform at most $2t$ times a calculation requiring $\mathcal{O}(t)$ operations. The proposition follows. □
Chapter 2. Error-locating pairs for cyclic codes

Remark 2.12 (on the proof of Theorem 2.4 and Theorem 2.5)
To complete the proof of Theorem 2.4, we have to add a row to $S(y)$ of Lemma 2.9. This row caused by the zero in $J$ does not fit into the quasi Hankel format. This causes one additional step in the FIA with complexity $O(t)$. Theorem 2.5 requires not only an additional row but also an additional column. We add this column as the rightmost column of $S(y)$. The FIA needs at most $O(t)$ operations for every position in this column. In both cases the overall complexity is still ruled by $O(t^2)$.

Example 2.13
The quadratic residue code of length 41 is a good example for the second conjugacy format. We find $I = \{1, 23, 37, 31, 0\}$ and $J = \{8, 20, 9, 0\}$ with $q^J = 23$. By the results of section 3 we can set $S_0 = 0$ whenever it is needed. An error-locating polynomial is found by solving the system

$$
\begin{pmatrix}
S_8^{5+1} & S_8^{2(3+1)} & S_8^{4(5+1)} & S_8^{5(0+1)} & S_8^5 \\
S_8^{3+1} & S_8^{2(8+1)} & S_8^{3(8+1)} & S_8^{2(8+1)} & S_8^3 \\
S_8^{7+1} & S_8^{2(8+1)} & S_8^{7(8+1)} & S_8^{7(8+1)} & S_8 \\
S_1 & S_8^2 & S_8^4 & S_8^6 & S_0
\end{pmatrix} \sigma = 0.
$$

We see that the submatrix $S(y)^{(3,1)}$ has a quasi Hankel structure which can be utilized to solve the system.

In considering cyclic codes of length $n$, in most cases codes $U$ and $V$ will be defined over the smallest field $\mathbb{F}$ containing an $n$-th root of unity. In some cases however $U$ and $V$ can be taken to be codes defined over a field $\mathbb{F} \subset \mathbb{F}$. This implies that $S(y)$ has entries from $\mathbb{F}$ rather than from $\overline{\mathbb{F}}$ which allows us to perform these operations faster.

Example 2.14
Let $n = 15$. Let $C$ be the double error-correcting BCH-code with $R_C = \{1, 2, 3, 4, 6, 8, 9, 12\}$. To show how the choice of $I$ and $J$ influences the decoding, we notice two possible choices. First we see that we can choose $J = \{1, 2\}$ and $I = \{0, 1, 2\}$ and this would correspond to the usual decoding as a subcode of a RS-code. A different choice is $I = \{2, 8, 0\}$ and $J = \{1, 4, 0\}$. This choice corresponds to cyclotomic cosets with respect to $GF(4)$. $S_0$ is only needed if two errors occurred which gives $S_0 = 0$. $S(y)$ will be a matrix over $GF(4)$ and all calculations will only involve computations over $GF(4)$. 
VII More on pairs

A Error-location by hyperplanes

In some cases we have a pair that does not satisfy condition (2.3). If the pair does satisfy the weaker condition (2.9), all solutions to the key equation are still error-locating. Example 2.18 treats such a situation. The situation becomes quite different when also condition (2.9) fails. The following proposition states an important property of the solution space to the key equation (2.4).

**Proposition 2.3** For a given code $C$, let the pair $(U, V)$ satisfy conditions (2.1),(2.2) and let $W \neq 0$, with

$$W = (e * U) \cap V^\perp.$$  

Then, for $y \in e + C$, the key equation (2.4)

$$\sum_{i=0}^{n-1} y_i u_i v_i = 0,$$  

for all $v \in V$,

has at least $m = k(W) + 1$ independent solutions $u_1, \ldots, u_m \in U$. Also, there exist $\lambda_1, \ldots, \lambda_m$ such that

$$e * (\lambda_1 u_1 + \ldots + \lambda_m u_m) = 0.$$  

(2.20)

**Proof.** We may replace $y$ in (2.4) with $e$. Clearly $u \in U$ is a solution whenever $e * u \in V^\perp$. In other words the space of solutions is the inverse image of $W$ under the linear map $U \rightarrow e * U$, $u \mapsto e * u$. The map has non-trivial kernel by condition (2.2), which proves (2.20). The vectors $e * u_1, \ldots, e * u_m$ are all in $W$ and hence are dependent. \qed

With the vectors $\{u_1, \ldots, u_m\}$ we associate the $n$ points $(u_{1i}, \ldots, u_{mi})$, $i = 1, \ldots, n$ in affine $m$-space. By the proposition all points corresponding to error positions are contained in a hyperplane through the origin

$$H : \lambda_1 X_1 + \ldots + \lambda_m X_m = 0.$$  

The most interesting situation is that condition (2.3) fails by one. In that case we can apply the following proposition.
Proposition 2.4  For a given code $C$, let the pair $(U, V)$ satisfy conditions (2.1),(2.2) and let $d(V^\perp)$ be equal to $t$. If the key equation (2.4) has a one-dimensional solution space spanned by $u_1 \in U$ then
\[ e \ast u_1 = 0 \]
is satisfied. If (2.4) has at least two independent solutions $u_1, u_2 \in U$ then the error points either lie in affine 2-space on a line through the origin excluding the origin or they coincide with the origin.

Proof. If the solution space is one-dimensional then by Proposition 2.3 $k(W)$ is equal to zero and condition (2.9) is satisfied. Otherwise $k(W)$ is not greater than one because the support of $W$ coincides with the support of $e$ and $d(W)$ is equal to $t$. Let $U(y)$ denote the space spanned by $u_1$ and $u_2$. If $W$ is contained in $e \ast U(y)$ then by Proposition 2.3 the error points lie on a line through the origin. At least one of $u_1$ and $u_2$ is unequal to zero at all error positions so the origin can be excluded. If $W$ is not contained in $e \ast U(y)$ both vectors $u_1$ and $u_2$ satisfy condition (2.9) and the error points coincide with the origin. \[ \square \]

Remark 2.13  In the above proposition we are looking for lines in affine space containing at least $t$ points. There can be no more than $n/t$ such lines. Thus solving for error values can be done in parallel, not affecting the time complexity. The computational complexity in this case is affected by a factor $n/t$.

Example 2.15  We consider the binary cyclic code $C$ of type $[31,11,11]$ with $\mathcal{R} = \{1, 3, 5, 11\}$. The complete defining set contains
\[ \{1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13\}. \]
We choose $I = \{1, 2, 3, 8, 9, 10\}$ and $J = \{0, 1, 2, 3\}$. Conditions (2.1), (2.2) and (2.11) are satisfied with $t = 5$, but $d(V^\perp) = 5$ and condition (2.3) is not satisfied. We only have $k(W) \leq 1$. We may obtain two independent solutions $u_1$ and $u_2$ to the key equation. Following the remark this gives 31 points in the affine plane. The linear combination of $u_1$ and $u_2$ that locates the error positions corresponds to a line passing through the origin and additional five of the 31 points or $u_1$ and $u_2$ are both error-locating. The code corresponds to entry 19 in Table 1 of Section VIII. Entries 47,119,124,137,140, 144,146 proceed likewise.
B Generalized Hamming weight

Assume now that conditions (2.1)-(2.3) are satisfied for a pair \((U, V)\) but condition (2.11) for error correction is not. Let \(U(y)\) denote the solutions of the key equation for a received word \(y\). Any \(u \in U(y)\) locates the error positions. Combination of several solutions reduces the possible error patterns. We use the concept of generalized Hamming weight [26].

\[ \text{Proposition 2.5} \quad \text{Let} \ (U, V) \text{ be a } t\text{-error-locating pair for the code } C \text{ as in Definition 2.1. Combination of } k(U) - t \text{ independent error-locating vectors } u \in U(y) \text{ determines the error values uniquely if the following is satisfied:} \]
\[ d(U, k(U) - t) + d(C) > n, \]
where \(d(U, i)\) denotes the \(i\)-th generalized Hamming weight of \(U\) and \(n\) denotes the code length of \(C\). 

\[ \text{Proof.} \quad \text{Immediate from the definition of generalized Hamming weight.} \]

\[ \square \]

\[ \text{Example 2.16} \quad \text{We consider binary Reed-Muller codes of length } n = 2^m, \text{ for } m > 3. \text{ See [15] for the definition and the main properties. The code } C = R(m - 3, m) \text{ has distance } d(C) = 8. \text{ The dual code is the second order Reed-Muller code, } C^\perp = R(2, m). \text{ For the decoding of } C \text{ we set } U = V = R(1, m). \text{ In particular } U \ast V \subseteq C^\perp. \text{ Furthermore } k(U) = m + 1 > 3 \text{ and } d(V^\perp) = d(R(m - 2, m)) = 4 > 3. \text{ Thus the pair } (U, V) \text{ satisfies Definition 2.1 and is } 3\text{-error-locating for the code } C. \text{ For a codeword } u \in U \]
\[ (u + 1) \ast R(m - 4, m) \subseteq C \]
holds. Thus the zero set of \(u\) supports a subcode of \(C\) and the pair \((U, V)\) is not 3-error-correcting. However we can reduce the possible error positions by combining \(k(U) - 3 = m - 2\) independent solutions to the key equation. We have \(d(U, i) = 2^m - 2^{m-i}\) and the combination reduces the number of possible error positions to four.

After puncturing, the code \(C\) is cyclic and the pair \((U, V)\) is defined with \(I = J = \{0, 1, 2, 4, \ldots, 2^{m-1}\}\). In this case there may appear only three possible error positions, the fourth coinciding with the punctured position.
Chapter 2. Error-locating pairs for cyclic codes

VIII Cyclic codes of length less than 63

Table 1 gives error-correcting pairs for binary cyclic codes which have error-correction capability exceeding the error-correction capability given by the BCH bound. We use the same numbering for codes as in [13]. Equivalent codes and subcodes with the same error correction capability are included in the table as remark. In four cases (no. 92, 123, 132, 146) we stay one short of the actual error-correction capability. All other pairs allow decoding up to half the actual minimum distance of the code.

To check conditions (2.1) and (2.2) of Definition 2.1 is straightforward. In all but four cases (no. 85, 106, 107, 137), the code $V$ is MDS. This follows either immediately using the BCH-bound or with Theorem 2.3. Thus, also condition (2.3) is easily verified in these cases. In the cases 85 and 137, the distance $d(V^\perp)$ is obtained with the Hartmann-Tzeng bound and Theorem 2.3 respectively. Cases 106 and 107 are treated in Example 2.18 and Example 2.19 respectively.

Usage of hyperplanes (Proposition 2.4) or usage of the unknown syndrome $S_0$ is indicated as remark. The remark FT indicates that the same error-correcting pair is given by Feng and Tzeng in [6].

To show that the pairs are error-correcting we use either the BCH-bound to show that condition (2.11) is satisfied or we use recurrences to determine unknown syndromes until we can apply Lemma 2.6. Whenever we use the conjugacy format, Lemma 2.8 provides us with a possibility to determine some unknown syndromes.

Cases that require other recurrences are listed in Table 2. For brevity we introduce the following notation. $\sigma_i \neq 0 : R(j) \rightarrow S_{i+j}$ means that we use equation (2.19) in Lemma 2.7,

$$\sigma_{i_1}S_{i_1+j} + \ldots + \sigma_{i_2}S_{i_2+j} + \sigma_{i_3}S_{i_3+j} = 0,$$

with the indicated $j$ and that $S_{i+j}$ will be the only unknown in the equation. Thus it can be obtained provided $\sigma_i \neq 0$. Recurrences separated by a comma can be computed in parallel. In Examples 2.17-2.20 the numbers in brackets refer to the numbering of the codes in the table.

Example 2.17 (26) The codes of type $[33, 11, 11]$ defined with $R = \{1, 3, 11\}$ and $R = \{3, 5, 11\}$ are equivalent. The given pairs are also equivalent. The codes $U$ and $V$ in the latter pair however have generator matrices
Table 1: Error-correcting pairs for binary cyclic codes of length less than 63.

<table>
<thead>
<tr>
<th>no.</th>
<th>n</th>
<th>k</th>
<th>d</th>
<th>R</th>
<th>J</th>
<th>I</th>
<th>d(V⁺)</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>17</td>
<td>9</td>
<td>5</td>
<td>[1]</td>
<td>[-1,+1]</td>
<td>[-3,0,+3]</td>
<td>3</td>
<td>GF(16)</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>9</td>
<td>8</td>
<td>[0,1,3,7]</td>
<td>[0,1,2]</td>
<td>[0,1,2,6]</td>
<td>4</td>
<td>FT</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>8</td>
<td>1,3,7,9</td>
<td>[0,1,2]</td>
<td>[1,2,6,7]</td>
<td>4</td>
<td>no.10;FT</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>23</td>
<td>12</td>
<td>7</td>
<td>[1]</td>
<td>[2,8,0]</td>
<td>[1,4,16,0]</td>
<td>4</td>
<td>no.12;S0=1</td>
</tr>
<tr>
<td>7</td>
<td>31</td>
<td>21</td>
<td>5</td>
<td>[1,5]</td>
<td>[1,4]</td>
<td>[0,1,4]</td>
<td>3</td>
<td>no.15</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>7</td>
<td>1,5,7</td>
<td>[0,1,2]</td>
<td>[0,7,8,18]</td>
<td>4</td>
<td>no.18;FT;S0=1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>11</td>
<td>1,3,5,11</td>
<td>[0,...,3]</td>
<td>[1,2,3,8,9,10]</td>
<td>5</td>
<td>no.21:H;Ex.2.15</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>33</td>
<td>13</td>
<td>10</td>
<td>[1,3]</td>
<td>[-2,...,+2]</td>
<td>[-2,...,+2]</td>
<td>5</td>
<td>GF(32),S0=0,1</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>1,3,11</td>
<td>[0,10,20,30,40]</td>
<td>[1,2,11,15,24,25]</td>
<td>6</td>
<td>no.27:FT</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>12</td>
<td>1,3,13</td>
<td>[0,...,3]</td>
<td>[1,2,3,8,9,10]</td>
<td>5</td>
<td>GF(32),no.27,Ex.2.17</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>35</td>
<td>16</td>
<td>7</td>
<td>[1,5,7]</td>
<td>[0,3,6]</td>
<td>[1,2,4,5]</td>
<td>4</td>
<td>no.37,38,39;FT</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>14</td>
<td>0,1,3,5</td>
<td>[0,...,5]</td>
<td>[31,...,34,0,1,8]</td>
<td>7</td>
<td>FT</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>39</td>
<td>26</td>
<td>6</td>
<td>[0,1]</td>
<td>[0,1]</td>
<td>[0,1,4]</td>
<td>3</td>
<td>no.44;FT;S0=0</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>10</td>
<td>1,3</td>
<td>[0,2,4,8]</td>
<td>[1,2,4,8,16]</td>
<td>5</td>
<td>no.46;Ex.2.12</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>13</td>
<td>12</td>
<td>1,3,13</td>
<td>[0,...,3]</td>
<td>[1,2,3,8,9,10]</td>
<td>5</td>
<td>no.48;H</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>41</td>
<td>21</td>
<td>9</td>
<td>[1]</td>
<td>[8,20,9,0]</td>
<td>[1,23,37,31,0]</td>
<td>5</td>
<td>no.50</td>
</tr>
<tr>
<td>19</td>
<td>43</td>
<td>29</td>
<td>6</td>
<td>[1]</td>
<td>[1,2]</td>
<td>[0,20,40]</td>
<td>3</td>
<td>FT</td>
</tr>
<tr>
<td>20</td>
<td>15</td>
<td>13</td>
<td>1,3</td>
<td>[-6,...,-1]</td>
<td>[0,...,6]</td>
<td>7</td>
<td>S0=0,1</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>45</td>
<td>23</td>
<td>7</td>
<td>[1,5,21]</td>
<td>[0,1,2]</td>
<td>[31,32,33,38]</td>
<td>4</td>
<td>no.74;FT</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>10</td>
<td>0,1,3,7</td>
<td>[0,...,3]</td>
<td>[-2,...,-1,11]</td>
<td>5</td>
<td>no.87,88;FT</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>15</td>
<td>9</td>
<td>1,3,15</td>
<td>[0,...,3]</td>
<td>[1,11,...,14]</td>
<td>5</td>
<td>no.87,89;Ex.2.6</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>15</td>
<td>10</td>
<td>1,7,9,15</td>
<td>[0,1,2,13,14,15]</td>
<td>[1,3,...,17]</td>
<td>5</td>
<td>no.86;FT</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>9</td>
<td>12</td>
<td>1,5,7,9,15</td>
<td>[13,...,17]</td>
<td>[0,1,2,13,14,15]</td>
<td>6</td>
<td>no.91;</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>47</td>
<td>24</td>
<td>11</td>
<td>[1]</td>
<td>[3,27,8,0]</td>
<td>[1,9,34,24,0]</td>
<td>5</td>
<td>no.93;S0=0</td>
</tr>
<tr>
<td>27</td>
<td>51</td>
<td>35</td>
<td>5</td>
<td>[1,9]</td>
<td>[0,1]</td>
<td>[1,8,15]</td>
<td>3</td>
<td>no.97,101,105;FT</td>
</tr>
<tr>
<td>28</td>
<td>34</td>
<td>6</td>
<td>[0,1,5]</td>
<td>[0,1]</td>
<td>[0,1,4]</td>
<td>3</td>
<td>no.104;FT</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>27</td>
<td>8</td>
<td>1,3,9</td>
<td>[0,2,8]</td>
<td>[0,1,4,16]</td>
<td>3</td>
<td>no.109,111,115; S0=1,-Ex.2.18</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>27</td>
<td>9</td>
<td>1,5,9</td>
<td>[0,2,7,8,13]</td>
<td>[0,2,7,8,13]</td>
<td>5</td>
<td>no.110,113,116; S0=0,-Ex.2.19</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>27</td>
<td>9</td>
<td>1,3,19</td>
<td>[-4,...,-1]</td>
<td>[0,...,4]</td>
<td>5</td>
<td>no.114,117;S0=0,1</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>19</td>
<td>14</td>
<td>1,3,5,9</td>
<td>[0,...,4]</td>
<td>[0,...,4,12,6]</td>
<td>6</td>
<td>no.120;S0=0,H</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>17</td>
<td>12</td>
<td>1,3,9,17,19</td>
<td>[-4,...,0]</td>
<td>[0,...,4,19]</td>
<td>6</td>
<td>no.125;S0,FT</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>17</td>
<td>14</td>
<td>1,3,5,17,19</td>
<td>[-4,...,1]</td>
<td>[0,...,6]</td>
<td>7</td>
<td>S0=0,1</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>17</td>
<td>14</td>
<td>1,5,9,17,19</td>
<td>[13,...,17]</td>
<td>[0,...,4,19]</td>
<td>6</td>
<td>no.126;Ex.2.20</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 2. Error-locating pairs for cyclic codes

Table 1 continued: Error-correcting pairs for binary cyclic codes of length less than 63.

<table>
<thead>
<tr>
<th>no.</th>
<th>condition</th>
<th>recurrence</th>
<th>syndromes</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>(\sigma_{18} \neq 0)</td>
<td>(R(25) \rightarrow S_{12})</td>
<td>(S_3, S_{15}, S_{11})</td>
</tr>
<tr>
<td></td>
<td>(\sigma_{18} = 0 \land \sigma_{8} \neq 0)</td>
<td>(R(7) \rightarrow S_{15})</td>
<td>(S_{15}, S_3, S_{11})</td>
</tr>
<tr>
<td>26</td>
<td>(R(10) \rightarrow S_7)</td>
<td>(S_7)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\sigma_{14} \neq 0)</td>
<td>(R(6) \rightarrow S_7)</td>
<td>(S_7)</td>
</tr>
<tr>
<td></td>
<td>(\sigma_{25} \neq 0)</td>
<td>(R(1) \rightarrow S_{25})</td>
<td>(S_7)</td>
</tr>
<tr>
<td></td>
<td>(\sigma_{2} \neq 0)</td>
<td>(R(24) \rightarrow S_{26})</td>
<td>(S_7)</td>
</tr>
<tr>
<td>73</td>
<td>(R(33) \rightarrow S_{26})</td>
<td>(S_7)</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>(R(5) \rightarrow S_5, R(18) \rightarrow S_{18})</td>
<td>(S_5, S_9)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\sigma_{0} = 0 \land \sigma_{11} \neq 0)</td>
<td>(R(25) \rightarrow S_{26}, R(10) \rightarrow S_{21}, R(14) \rightarrow S_{10})</td>
<td>(S_9, S_{21}, S_5)</td>
</tr>
<tr>
<td>90</td>
<td>(R(27) \rightarrow S_{26})</td>
<td>(S_9)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\sigma_{15} = 0 \land \sigma_{14} \neq 0)</td>
<td>(R(28) \rightarrow S_{22})</td>
<td>(S_9)</td>
</tr>
<tr>
<td></td>
<td>(\sigma_{15} = 0 \land \sigma_{14} = 0 \land \sigma_{13} \neq 0)</td>
<td>(R(20) \rightarrow S_{21})</td>
<td>(S_9)</td>
</tr>
<tr>
<td>96</td>
<td>(R(33) \rightarrow S_{14}, R(2) \rightarrow S_{35}, R(40) \rightarrow S_{41})</td>
<td>(S_5, S_3, S_{11})</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>(R(48) \rightarrow S_{43}, R(9) \rightarrow S_{10}, R(43) \rightarrow S_{44})</td>
<td>(S_9, S_5, S_{11})</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>(R(5) \rightarrow S_{12}, R(16) \rightarrow S_{23})</td>
<td>(S_3, S_{11})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\sigma_{7} = 0 \land \sigma_{8} \neq 0)</td>
<td>(R(16) \rightarrow S_{24}, R(3) \rightarrow S_{11})</td>
<td>(S_3, S_{11})</td>
</tr>
<tr>
<td>121</td>
<td>(R(14) \rightarrow S_{14}, R(23) \rightarrow S_{23})</td>
<td>(S_5, S_{11})</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\sigma_{0} = 0 \land \sigma_{19} \neq 0)</td>
<td>(R(46) \rightarrow S_{14}, R(1) \rightarrow S_{23})</td>
<td>(S_5, S_{11})</td>
</tr>
<tr>
<td>123</td>
<td>(R(31) \rightarrow S_{31}, R(48) \rightarrow S_{48})</td>
<td>(S_{11}, S_5)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\sigma_{0} = 0 \land \sigma_{10} \neq 0)</td>
<td>(R(12) \rightarrow S_{31}, R(29) \rightarrow S_{48})</td>
<td>(S_{11}, S_5)</td>
</tr>
<tr>
<td></td>
<td>(\sigma_{0} = 0 \land \sigma_{10} \neq 0)</td>
<td>(R(12) \rightarrow S_{31}, R(29) \rightarrow S_{48})</td>
<td>(S_{11}, S_5)</td>
</tr>
<tr>
<td>136</td>
<td>(R(32) \rightarrow S_{23}, R(28) \rightarrow S_{29}, R(19) \rightarrow S_{20})</td>
<td>(S_1, S_8, S_5)</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>(R(28) \rightarrow S_{29}, R(19) \rightarrow S_{20})</td>
<td>(S_3, S_5)</td>
<td></td>
</tr>
</tbody>
</table>

* This entry was incomplete in the article that was published in IEEE Transactions on Information Theory

Table 2. Recurrences for the error-correcting pairs from Table 1.
\(G_U\) and \(G_V\) respectively that are defined over \(GF(32)\). With this choice of \(G_U\) and \(G_V\) the key equation (2.6) can be solved over the field \(GF(32)\).

**Example 2.18 (106)** The code \(C\) is of type \([51, 27, 8]\) with \(R \supseteq \{1, 3, 9\}\). The pair \((U, V)\) is defined with \(I = \{2, 8, 12, 0\}\) and \(J = \{1, 4, 0\}\). We need \(S_0\) only when three errors occurred and may then set \(S_0 = 1\) (see section 3). For the error-location all conditions except (2.3) are obviously satisfied. In fact \(d(V^\perp) = 3\) and condition (2.3) does not hold for \(t = 3\). We prove the weaker condition (2.9): \((e \ast U) \cap V^\perp = 0\). Words of weight three in \(V^\perp/GF(256)\) are in the code with defining set \(R = \{1, 4, 16, 13, 0\}\) by Theorem 2.3. But \(\{1, 4\} + \{0, 12\} \subset R\), which implies by Theorem 5 in [13] that the support of a word of weight three must be of the form

\[\{\alpha, \rho \alpha, \rho^2 \alpha\},\]

for \(\alpha\) a 51-th root of unity and \(\rho\) a primitive third root of unity. Up to multiplication with a scalar the values at these positions are \((1, \rho, \rho^2)\). But the values of \(u \in U\) at these positions are a linear combination of \((1, 1, 1)\) and \((1, \rho^2, \rho)\) and (2.9) is satisfied. In [6] the pair \(I = \{0, 2, 8, 12\}\) and \(J = \{0, 1, 4\}\) is given without the above verification.

**Example 2.19 (107)** The code \(C\) is of type \([51, 27, 9]\) with \(R \supseteq \{1, 5, 9\}\). The pair \((U, V)\) is defined with \(I = J = \{8, 13, 2, 7, 0\}\). We need \(S_0\) only when four errors occurred and may then set \(S_0 = 0\) (see section 3). For the error-location we prove (2.3): \(d(V^\perp) > 4\). A word \(c \in V^\perp\) satisfies the checks \(\{8, 13, 2, 0\}\) and by Theorem 2.3 also checks \(\{8, 13, 2, 16, 26, 4, 32, 10\}\) if it is of weight four or less. Thus \(d(V^\perp) \geq 4\). Let \(c\) have non-zero values \((c_1, c_2, c_3, c_4)\). At the same support we have codewords with values \((c_1c_1, c_1c_2, c_1c_3, c_1c_4)\) and \((c_1^2, c_2^2, c_3^2, c_4^2)\). Thus \(c_1 = c_2 = c_3 = c_4\). Example 32 in [13] shows that a binary code with \(R \supseteq \{1, 7\}\) has no words of weight four, using \(R \supseteq \{0, 3\} + \{1, 2, 4\}\). The error-correction follows with the recurrences in Table 2.

**Example 2.20 (123)** There is an extended choice for \(U\) and \(V\), namely \(I = \{0, 1, 2, 3, 4, 19, 36\}\) and \(J = \{13, 14, 15, 16, 17, 32, 49\}\). Assume six errors have occurred and we find a space of polynomials with \(\sigma_{36}\) equal to
zero. Then we can apply the hyperplane method and the given recurrences. If there is no such solution then the error positions do not support a codeword in \( V^\perp \) and the solution with \( \sigma_{36} \neq 0 \) is error-locating. An error-locator of the form \( \sigma_2 X^2 + \sigma_{19} X^{10} + \sigma_{36} X^{35} \) can occur and then the zero set supports an eight dimensional subcode of \( C \). Solutions of another form determine the error values uniquely:

\[
\begin{align*}
\sigma_0 \neq 0 : & \quad R(31) \rightarrow S_{31}, R(48) \rightarrow S_{48}.
\sigma_1 \neq 0 : & \quad R(30) \rightarrow S_{31}, R(47) \rightarrow S_{48}.
\sigma_3 \neq 0 : & \quad R(34) \rightarrow S_{37}, R(0) \rightarrow S_3.
\sigma_4 \neq 0 : & \quad R(33) \rightarrow S_{37}, R(50) \rightarrow S_3.
\end{align*}
\]

**IX Sequences of codes**

In a certain range of the minimum distance, the conjugacy formats of Section V allow the construction of sequences of cyclic codes with the same designed distance and redundancy as BCH-codes.

**Proposition 2.6**  
Let \( n \) be equal to \( 2^m - 1 \), \( m = 2l + 1 \). For a binary cyclic code \( C \) with defining set \( R_C \supseteq \{1, 2^l + 1, 2^{l-1} + 1\} \) we have a 3-error-correcting pair \((U, V)\) with generating sets \( I = \{0, 1, 2^l, 2^{2l}\} \) and \( J = \{0, 2^l, 2^{3l}\} \).

**Proof.** The code is defined in [15, Ch.9 §11]. There it is also proved that the distance \( d = 7 \). For the even weight subcode we may write \( R \supseteq \{0, 1, 2^l + 1, 2^{3l} + 1\} \). Thus we find the formats of Theorem 2.5, except that the code \( U \) is not MDS. In fact the codes \( U \) and \( V \) are equivalent to codes with generating sets \( I' = 4 \cdot I = \{0, 4, 2, 1\} \) and \( J' = 4 \cdot 2^l \cdot J = \{0, 2, 1\} \), which shows that conditions (2.1)-(2.3) and (2.11) are fulfilled. Since \( S_0 \) occurs only once in the key matrix \( S(y) \), by the results of Section IV we can assume that \( S_{0} \equiv 3 \pmod{2} \). \( \Box \)

**Remark 2.14 (QR-codes)**  
It is clear from Table 1, that the second conjugacy format yields good pairs for the smaller binary QR-codes. The
conjugacy formats require defining sets of size $t$, while the BCH-format requires sets of size $2t$ (the largest set of consecutive quadratic residues is in general not formed by the residues $\{1, 2, \ldots, 2t\}$ and the usual argument that only $\{1, 3, \ldots, 2t - 1\}$ need to be in the defining set does not apply). With a uniform distribution of the quadratic residues, the conjugacy formats should correct about twice the number of errors of the BCH-format. Calculations for codes of length less than 1024 agree with this. For example, for the codes of length $n = 863$ and $n = 887$, we apply Theorems 2.4 and 2.5 with $q = 2$. They yield pairs to correct $t = 10 \ (l = 57)$ and $t = 7 \ (l = 62)$ errors for $n = 863$ and $t = 8 \ (l = 182)$ and $t = 11 \ (l = 206)$ errors for $n = 887$. For both values of $n$, the BCH-format corrects $t = 4$ errors. Also, it is clear that both the BCH-format and the conjugacy formats have a capability that is of order $\log(n)$ for large codelength $n$. This is way below the square root bound.

In addition to the sequences of codes satisfying the conjugacy format, we give the following sequences.

**Proposition 2.7** Let $C$ be a binary cyclic code of length $n$ where 3 does not divide $n$. Let $R_C$ contain the set $\{-1, 1\}$. Then a 2-error-correcting pair $(U, V)$ is given through generating sets $I = \{-3, 0, 3\}$ and $J = \{-1, 1\}$.

**Proof.** The complete defining set $R$ for $C$ satisfies $R \supseteq I + J$. 3 does not divide the length and it follows that $U$ and $V$ are both MDS. The proof follows from Proposition 2.1. \qed

**Example 2.21 (Zetterberg codes [24])** Let $n$ be equal to $2^{2m} + 1$. The Zetterberg code $C$ with defining set $R_C = \{1\}$ has the 2-error-correcting pair $(U, V)$ given in Proposition 2.7.

**Example 2.22 (Melas codes [25])** Let $n$ be equal to $2^m - 1$ and let $m$ be odd. The Melas code $C$ with defining set $R_C = \{1, -1\}$ has the 2-error-correcting pair $(U, V)$ given in Proposition 2.7.
The following sequence of reversible codes contains as members binary codes of type $[73, 37, \geq 11]$ and $[85, 45, \geq 11]$.

**Proposition 2.8**  Let $C$ be a binary cyclic code of length $n$ where 3 does not divide $n$. Let $R_C$ contain the set $\{-7, -5, -1, 1, 5, 7\}$. Then a 5-error-correcting pair $(U, V)$ is given through generating sets $I = \{-4, -2, -1, 1, 2, 4\}$ and $J = \{-6, -3, -0, 3, 6\}$.

**Proof.** Use Theorem 2.2. \qed

# Conclusions

In this paper we have presented algebraic decoding algorithms for linear cyclic codes. For binary codes of length less than 63, the full error-correction capability of the code is achieved in all but four cases. The decoding procedures depend heavily on the idea of error-correcting pairs which was first developed by Pellikaan in [17]. This idea unifies to a certain extent algebraic decoding techniques, and, as is shown in Section IV, former results obtained by Feng and Tzeng in [5] have a clear interpretation. As is shown in Section V, error-correcting pairs from MDS-codes are very often the key to the decoding of certain classes of cyclic codes. So the decoding of classical BCH-codes employs RS-codes in the corresponding error-locating pairs. We have derived a new class of MDS-codes which are used to describe a new family of codes. This new family resembles in many ways the classical BCH-codes and contains members like the $[23, 12, 7]$ Golay code and the $[41, 21, 9]$ QR-code. In correspondence with BCH-codes an efficient decoding procedure for the new family is derived which uses a Berlekamp Massey type algorithm. Thus the decoding of codes in this family needs only $O(n^2)$ operations.
Bibliography


Chapter 3

A fast parallel implementation of a Berlekamp-Massey algorithm for algebraic-geometric codes*

Abstract  We obtain a parallel Berlekamp-Massey type algorithm for determining error-locating functions for the class of one-point algebraic-geometric codes. The proposed algorithm has a regular and simple structure and is suitable for VLSI implementation. We give an outline for an implementation, which uses as main blocks γ copies of a modified one-dimensional Berlekamp-Massey algorithm, where γ is the size of the first non-gap in the function space associated with the code. Such a parallel implementation determines an error locating function in time $O(n^2)$ where $n$ is the length of the algebraic geometric code.

Index Terms — algebraic geometric codes, decoding, Berlekamp-Massey algorithm.

* submitted to IEEE Transactions on Information Theory
Chapter 3. A fast parallel implementation of a BMA for AG codes

I Introduction

The decoding problem in coding theory consists of finding a codeword in a given code that has minimum Hamming distance to a given received word $y$. One successful strategy to accomplish this task is decoding by error location. This means that the problem is separated into the two problems of first determining the support of the error vector and then determining the corresponding error values. Decoding by error location is the common principle of most decoding algorithms for algebraic geometric (AG) codes. The main problem in making AG codes practical is to find a way of performing the required computations in a fast and efficient way.

A computationally efficient algorithm for decoding of AG codes was first described by Justesen et al. in [1]. The approach is based on a two-dimensional Berlekamp-Massey type algorithm introduced by Sakata in [2]. In [3] Sakata et al. extended the algorithm to allow efficient decoding up to half the designed minimum distance. The algorithm in [3] treats in detail codes from Hermitian curves and needs a complexity, counted as the number of multiplication in the field of concern, that is upper bounded by a term $A n^{7/3}$ for a large enough constant $A$. Here $n$ denotes the length of the code from an Hermitian curve and the constant $A$ is independent of the length of the code.

In addition to [1][3] there is a considerable literature on the computationally efficient decoding of AG codes available [4][5][6][7][8][9][10][11][12]. The algorithms presented in these papers yield a decoding complexity which is essentially upper bounded by a term $B n^2$ for a large enough constant $B$. Again, $n$ denotes the length of the AG code, the constant $B$ is independent of the length of the code and $\gamma$ is the first non-gap in the non-gap sequence of the space associated with the AG code.

With exception of [11] hardware implementation issues are rarely treated in the above references. The main goal and achievement of the present paper is the development of an algorithm that is fast and can be efficiently implemented. The development of such an algorithm is crucial to make AG codes competitive when compared to non-binary BCH codes or concatenated code constructions. In this context one has to consider not only computational complexity but also steering logic overhead and space
requirements in a hardware implementation. In particular, a parallel, simple and regular implementation of the decoding algorithm is essential in order to make AG codes a reasonable option in engineering applications.

We propose an algorithm that offers such convenient features in a hardware implementation, while maintaining essentially the same computational burden as the algorithms mentioned above. A parallel implementation performs one run of the algorithm as fast as a one-dimensional Berlekamp-Massey algorithm (BMA) used for example in the decoding of Reed-Solomon (RS) codes. This is due to the fact that the basic building blocks of a parallel implementation of the presented algorithm are just modified implementations of a one-dimensional BMA. Using a serial implementation of these one-dimensional BMA the running time of our algorithm is $O(n^2)$, where as a time unit we use the time required for a multiplication in a finite field. We outline a parallel architecture for VLSI implementation that is based on Blahut's implementation of a one dimensional BMA [15]. Using a parallel implementation of the one-dimensional BMA it is possible to further speed up the algorithm. We achieve this running time while maintaining the space complexity $O(\gamma n)$, measured in terms of registers and arithmetical units. It is convenient to compare the cost of using algebraic geometric codes in an application to the costs of using an RS code. In the proposed parallel implementation the time requirement for an algebraic geometric code correcting $t$ errors and defined over a curve of genus $g$ is essentially the same as the time requirements of a Reed-Solomon code correcting $t + 2g$ errors. The space requirement is $\gamma$ times as large. For codes on Hermitian curves, $\gamma$ equals $n^{\frac{1}{2}}$. This has to be seen in the light of the Hermitian code being $\gamma$ times longer than an extended Reed Solomon code over the same alphabet.

We also give an algorithm for efficient error-erasure decoding. The main difference to the case of error-only decoding is a different initialization of the algorithm.

Given a nontrivial error-locating function it is known how to calculate the error values to complete the decoding of an AG code. In order to keep the paper short we concentrate on the problem of finding error-locating functions. For a treatment of the problem of finding error values in order to complete the decoding the reader is referred to e.g. [1][10][14].

When dealing with computational complexity, we prefer in the present paper to count the number of binary operations that are required to perform all
necessary multiplications in the finite field of concern. Whenever a statement in this paper concerns complexity, we assume that the finite fields of concern have characteristic two. Using this complexity measure the algorithm from [3] requires a complexity in the order $O((\log_2(q))^2n^{7/3})$ where $q$ denotes the size of the code alphabet. For a thorough treatment of complexity issues in connection with VLSI implementations we refer to Mastrovito [16].

In contrast to computational complexity, the time complexity is measured in the time required for a multiplication in a finite field. We use this time unit independently of the size of the field of concern. This is reasonable when we assume the use of parallel multiplier architectures and when we neglect the length of the critical path in a VLSI multiplier implementation [16].

II Algebraic-geometric codes

Let $\mathbb{F}_q$ be the finite field with $q$ elements and let $\mathcal{X}$ be a nonsingular, projective, absolutely irreducible curve over $\mathbb{F}_q$ with genus $g$. A rational function $f$ on $\mathcal{X}$ is characterized by the divisor of zeros and poles of $f$, denoted by $(f)$. For a given divisor $G$, $L(G)$ is defined as the space of all rational functions satisfying

$$L(G) = \{f : (f) + G \geq 0\}.$$ 

Let $D$ be a divisor obtained as the sum of $n$ distinct $\mathbb{F}_q$-rational points $P_i$, $i = 0 \ldots n - 1$ such that the supports of $G$ and $D$ are disjoint. There are two different ways of defining an AG code. The functional code $C_L(D,G)$ is defined as

$$C_L(D,G) = \{(f(P_0), f(P_1), \ldots, f(P_{n-1})) : f \in L(G)\}$$

and the residual code $C_\Omega(D,G)$ is defined as

$$C_\Omega(D,G) = (C_L(D,G))^\perp.$$ 

The term residual code originates from the fact that $C_\Omega(D,G)$ can be obtained by taking residues of differentials on $\mathcal{X}$ in points $P_i$. The parameters
II. Algebraic-geometric codes

of \( C_\Omega(D, G) \) can be found in e.g [17] as length \( n = \deg(D) \), dimension
\( k(C_\Omega) = n - \deg(G) + g - 1 \) and designed minimum Hamming distance
\( d^*(C_\Omega) = \deg(G) - 2g + 2 \) for \( n > \deg(G) > 2g - 2 \).

Let \( l(G) \) denote the dimension of \( L(G) \). The main tool for investigating AG codes is the Riemann-Roch theorem, which yields that
\[
l(G) \geq \deg(G) - g + 1
\]
with equality if \( \deg(G) > 2g - 2 \).

We will only consider the case of so called one-point codes, which means that \( G \) is a multiple of a \( \mathbb{F}_q \)-rational point \( P_\infty \). This is advantageous in an implementation, because the space \( L(mP_\infty) \) has a relatively regular structure compared to spaces associated with arbitrary divisors \( G \). We briefly recall the main ideas of Sakata's algorithm [2] when it is applied to the efficient decoding of AG codes, cf. [1][3][4][5][6][10].

The set of functions, that have poles only at a certain point \( P_\infty \), forms a ring. For simplicity we assume in this section, that this ring is generated over \( \mathbb{F}_q \) by two functions \( \psi_1, \psi_2 \) with respective pole orders \( v_1, v_2; v_1 < v_2 \) at \( P_\infty \). The value \( v_1 \) is crucial for complexity considerations and we denote it by \( \gamma \).

Let a vector \( y \) of length \( n \) be given such that \( y = c + e, c \in C_\Omega(D, G) \) where \( e \) is an error vector with Hamming weight \( t < d^*(C_\Omega)/2 \). The syndromes \( s_{i,j} \) are defined as
\[
s_{i,j} = \sum_{l=0}^{n-1} e_l \psi_1^j(P_l) \psi_2^j(P_l).
\]

It is clear from the definition of the code \( C_\Omega(D, mP_\infty) \) that we can calculate \( s_{i,j} \) from the vector \( y \) if \( iv_1 + jv_2 \) is less than or equal to \( m \). Let \( f \) be a function of the form
\[
f = \sum_{(i,j): iv_1 + jv_2 \leq w} f_{i,j} \psi_1^i \psi_2^j,
\]
which has pole order \( w \) at \( P_\infty \). \( f \) is called an error-locating function if \( f \) evaluates to zero in the error positions, i.e. the equation \( f(P_l)e_l = 0 \) holds
for all \( l \). The main observation that allows the determination of an error-locating function \( f \) is that \( f \) gives a linear recurrence on the two dimensional array of syndromes \( s_{i,j} \), cf.[18],

\[
\sum_{(i,j):i v_1 + j v_2 \leq w} f_{i,j} s_{i+i_0,j+j_0} = 0 \ \forall i_0, j_0 \geq 0. \tag{3.1}
\]

In [18] it is shown that we can determine an error-locating function from the known syndromes provided that the weight of the error vector is less than \( (m-3g+1)/2 = (d^*(C_\Omega) - g)/2 \). In order to achieve the full error correction capability of the code, we need to know all syndromes \( s_{i,j} \) with \( iv_1 + jv_2 \) less than or equal to \( m + g \). Feng and Rao [19] and Duursma [20] showed how to obtain the \( s_{i,j} \) with \( iv_1 + jv_2 > m \) from the known \( s_{i,j} \), provided that the error vector has weight less than \( d_{FR}(C_\Omega)/2 \), where \( d_{FR}(C_\Omega) \geq d^*(C_\Omega) \) denotes the Feng Rao distance of the code \( C_\Omega(D, mP_\infty) \), cf. [21]. Their result leads to an algorithm, which allows determination of an error-locating function \( f \) for any error vector of weight at most \( (d_{FR}(C_\Omega) - 1)/2 \). Once an error-locating function (i.e. a set of possible error positions) has been found we can find the error vector \( e \) by an erasure decoding algorithm. Given an error-locating function this step requires a complexity at most in the order of \( O((\log_q(q))^2 n^2) \) cf. [1][10][14], which is also the complexity required to calculate syndromes from the received word.

The overall complexity of a decoding algorithm for AG codes is usually dominated by the complexity of the algorithm used for the calculation of the error-locating function. Feng and Rao used Gaussian elimination to solve the set of linear equations (3.1) resulting in a complexity \( O((\log_q(q))^2 n^3) \).

Our algorithm requires a computational complexity \( O((\log_q(q))^2 n^2) \), which is essentially the complexity required by any of the algorithms proposed for efficient decoding of AG codes. However, our algorithm offers in addition to this computational complexity a simple and easily parallelized structure. In order to be able to compare our algorithm with Sakata type algorithms we describe the operations required to perform a typical iteration step in the modified Sakata algorithm. For a detailed treatment and the required modifications of the algorithm we refer to [1][2][10][22] and [23].

At the starting point of a typical iteration of Sakatas algorithm we have two collections of functions \( F_u \) and \( G_u \) with the following properties. The
functions $f \in F_u$ satisfy equations (3.1) for all numbers $(i_0, j_0)$ such that $i_0 v_1 + j_0 v_2 + w$ is less than or equal to $u$. The set $G_u$ consists of functions $g$, that failed to satisfy the equations of (3.1) in an earlier step of the algorithm. The goal of one iteration step is to construct corresponding sets $\tilde{F}_{u+1}$ and $G_{u+1}$. This is accomplished by performing the following operations.
1. For any \( f \in F_u \) check whether \( f \) is also an element in \( F_{u+1} \).

2. Given the pole orders of functions in \( F_u \) and \( G_u \), the pole order of functions in \( F_{u+1} \) is calculated according to a number of different cases described in [2]. The corresponding functions \( f' \in F_{u+1} \) are then constructed by selecting suitable pairs of functions \((f, g), f \in F_u \) and \( g \in G_u \) and combining them in a proper way. This implies an operation of the form

\[
f' \leftarrow \psi_1^a \psi_2^b f + \delta \psi_1^{a'} \psi_2^{b'} g
\]

where the numbers \( a, b, a', b', \delta \) are determined by the different cases described in [2].

3. The elements of \( G_{u+1} \) are chosen among the elements of \( G_u \) and the functions of \( F_u \) such that certain requirements associated with the set \( F_{u+1} \) are satisfied.

One of the key problems in finding an efficient way of implementing Sakata’s algorithm is that in step 2 of an iteration a function \( f \in F_u \), \( g \in G_u \) may be used for the construction of several different functions in \( F_{u+1} \). Moreover, the pair of functions \( f \) and \( g \) that are needed to construct a function \( f' \in F_{u+1} \) is impossible to predict a priori. We quote the following statement from [24] which points out just this difficulty in a software implementation of the algorithm from [3]:

‘At any point \( q \) we unfortunately need access to every polynomial in the auxiliary set \( G \) and \( F \) when we want to construct a polynomial for the new \( F \)-set. ... . For the sake of hardware-implementation it would be nice if we do not have to use the polynomials in \( F \) and \( G \) more than once when we construct the new \( F \)-set.’

It is just this difficulty that we overcome in our algorithm by using two sets of functions that maintain a constant size \( \gamma \) throughout the algorithm. A similar idea was used by Sakata in [12] to develop a parallel algorithm in the framework of his algorithm. The most important features of our algorithm are listed below.
• The algorithm is easily parallelized where the main building blocks of an implementation are \( \gamma \) modified implementations of a one-dimensional BMA. Thus the time required for one run of the algorithm is essentially equal to the time required by a BMA for RS codes.

• The different cases of Sakata's algorithm are reduced to one condition.

• No selection procedure for pairs of functions is needed. The pair of functions used to construct a new function is in every iteration step determined \textit{a priori}.

• No additional multiplication of functions with monomials of the form \( \psi^a \psi^b \) (\( \psi_1^a \psi_2^b \)) is required.

### III The syndrome matrix

Our algorithm is best described using a matrix notation. We translate the linear conditions, that are imposed on an error-locating function by equation (3.1), into a matrix language, cf. [7][8]. We start by characterizing some properties of the space \( L(mP_{\infty}) \), that are needed to fix a suitable basis of \( L(mP_{\infty}) \).

Let \( o_j \) be a non-negative integer such that \( l(o_j P_{\infty}) \neq l((o_j - 1)P_{\infty}) \). This means that there exists a function \( \phi_{o_j} \) with a pole of order \( o_j \) at \( P_{\infty} \). The number \( o_j \) is called a non-gap of \( P_{\infty} \). The constant function has no poles so that \( o_0 = 0 \) and \( \phi_0 = 1 \). As a consequence of the Riemann-Roch Theorem the non-gaps satisfy

\[
0 = o_0 < o_1 < o_2 < \cdots < o_{g-1} < 2g, \quad o_i = i + g, \text{ for } i \geq g.
\]

\( \phi_{o_1} \) is a non-constant function of lowest pole order at \( P_{\infty} \) and it follows that \( \gamma \) equals \( o_1 \). \( P_{\infty} \) is called a \textit{Weierstrass point}, if \( \gamma \) is less than \( g + 1 \). It is clear that when \( o_i \) and \( o_j \) are non-gaps then \( o_i + o_j \) is also a non-gap. The set of integers \( i \in \mathbb{Z} \) such that \( l(iP_{\infty}) = l((i - 1)P_{\infty}) \) holds is denoted by \( G \). The following definition gives a basis of \( L(mP_{\infty}) \) that is suitable for the proposed algorithm. Functions that are elements in \( L(mP_{\infty}) \) are always assumed to be expanded in this basis.
**Definition 3.1** A standard basis \( B(m) = \{1, \phi_\gamma, \ldots, \phi_m \} \) for \( L(mP_\infty) \) has the property that if \( \phi_j \) is an element of \( B(m) \) and \( l \) is the least positive non-gap in \( B(m) \) such that \( j - l \) is a non-gap then
\[
\phi_j = \phi_l \phi_{j-l}.
\]

The most important consequence of Definition 3.1 is that for any \( \phi_l \in B(m) \) with \( l \leq m - \gamma \) the function \( \phi_l \phi_{l'} \), too, is an element of \( B(m) \). In other words \( B(m) \) is closed under multiplication with \( \phi_{l} \) as long as this is consistent with the maximal pole order \( m \). In order to get a structurally nice formulation of the algorithm in section IV, we formally define \( \phi_i = 0 \) if \( i \in G \), cf. [7][8]. We can uniquely express any function \( f \in L(mP_\infty) \) as
\[
f = \sum_{i=0}^{m} f_i \phi_i, \quad f_i = 0 \forall i \in G,
\]
and can thus identify vectors \( \mathbf{f} = (f_0, f_1, \ldots, f_m) \) with functions. We will freely use this identification and say e.g. that a function \( f \) solves a linear system of equations if the corresponding coefficient vector does.

**Example 3.1** We consider the Klein quartic curve over \( \mathbb{F}_{2^3} \cong \mathbb{F}_2[\alpha]/\langle \alpha^3 + \alpha + 1 \rangle \) given by the equation
\[
\mathcal{X} : X^3 Y + Y^3 Z = Z^3 X.
\]
\( \mathcal{X} \) has genus \( g = 3 \) and contains 24 \( \mathbb{F}_{2^3} \)-rational points \( P_i \). We choose the point \( P_\infty = (0:1:0) \) and construct the space of functions \( L(13P_\infty) \). The intersection divisors of the coordinate functions are
\[
(\bar{X}) = 3(0:0:1) + (0:1:0) \\
(\bar{Y}) = 3(1:0:0) + (0:0:1) \\
(\bar{Z}) = 3(0:1:0) + (1:0:0).
\]
The functions \( \phi_3 = Y/Z, \phi_5 = XY/Z^2 \) and \( \phi_7 = Y^3/Z^2 X \) have pole order 3, 5, 7 at \( P_\infty \), respectively. Moreover \( \phi_3, \phi_5, \phi_7 \) generate the ring of functions with poles in \( P_\infty \). The gaps are \( (1, 2, 4) \) and \( \gamma \) equals 3. A standard basis for \( L(13P_\infty) \) is given as
\[
1, \phi_3, \phi_5, \phi_3^2, \phi_7, \phi_3 \phi_5, \phi_3^3, \phi_3 \phi_7, \phi_3^2 \phi_5, \phi_3^4, \phi_3^2 \phi_7.
\]
The code $C_\Omega(D, 13 P_\infty)$ has length 23, dimension 12 and minimum distance 9.

For later use we note the following relations for functions on $X$, which can easily be verified:

\[
\phi_6^2 = \phi_{10} + \phi_3, \quad \phi_7^2 = \phi_{14} + \phi_7, \quad \phi_5 \phi_7 = \phi_{12}.
\]

Using these relations we find

\[
\phi_i \phi_j = \begin{cases} 
0 & i \in \{1, 2, 4\} \lor j \in \{1, 2, 4\} \\
\phi_{i+j} + \phi_{i+j-7} & i \not\in \{1, 2, 4\} \land j \not\in \{1, 2, 4\} \land i \equiv j \not\equiv 0 \mod 3 \\
\phi_{i+j} & \text{otherwise.}
\end{cases}
\]

As before let a vector $y$ of length $n$ be given such that $y = c + e, c \in C_\Omega(D, m P_\infty)$. We define the entries of a semi-infinite matrix $S = ||S_{i,j}||, 0 \leq i, j$ by the equation

\[
S_{i,j} = \sum_{l=0}^{n-1} e_l \phi_i(P_l) \phi_j(P_l).
\]

The linear conditions, that the matrix $S$ imposes on a vector $\sigma$ in a system of linear equations $S \sigma^T = 0$, are in fact just a reformulation of the linear conditions described by the recursions of equation (3.1). Thus any solution to the equation $S \sigma^T = 0$ corresponds to the coefficient vector of an error-locating function.

**Example 3.2 (Example 3.1 continued)** We consider the setup of Example 3.1. For a received vector $y$ let the following entries in $S$ be calculated.

\[
S_{0,0} = \alpha^4, \quad S_{3,0} = \alpha, \quad S_{5,0} = \alpha^3, \quad S_{6,0} = \alpha^6, \quad S_{7,0} = 1, \quad S_{8,0} = \alpha^6, \\
S_{9,0} = 0, \quad S_{10,0} = \alpha^5, \quad S_{11,0} = \alpha^2, \quad S_{12,0} = \alpha^4, \quad S_{13,0} = \alpha^5.
\]

The syndromes $S_{i,0}, i \leq 13$ correspond to an error vector of weight 4 with errors at points $(1 : \alpha : 1), (1 : \alpha^2 : 1), (\alpha^2 : \alpha^4 : 1), (\alpha^2 : \alpha^5 : 1)$.
and corresponding error values $\alpha$, $1$, $\alpha^2$. We can calculate the known part of the matrix $S$ with help of the following relations, which are a direct consequence of the relations among functions $\phi_i$ given in Example 3.1,

$$S_{i,j} = \begin{cases} 
0 & i \in \{1,2,4\} \vee j \in \{1,2,4\} \\
S_{i+j,0} + S_{i+j-7,0} & i \notin \{1,2,4\} \wedge j \notin \{1,2,4\} \wedge i \equiv j \not\equiv 0 \mod 3 \\
S_{i+j,0} & \text{otherwise}.
\end{cases}$$

Our knowledge of the matrix $S$ can be expressed as follows:

$$S = \begin{pmatrix} 
\alpha^4 & 0 & 0 & \alpha & 0 & \alpha^3 & \alpha^6 & 1 & \alpha^6 & 0 & \alpha^5 & \alpha^2 & \alpha^4 & \alpha^5 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\alpha & 0 & 0 & \alpha^6 & 0 & \alpha^6 & 0 & \alpha^5 & \alpha^2 & \alpha^4 & \alpha^5 & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\alpha^3 & 0 & 0 & \alpha^6 & 0 & \alpha^6 & \alpha^2 & \alpha^4 & \alpha & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\alpha^6 & 0 & 0 & 0 & 0 & \alpha^2 & \alpha^4 & \alpha^5 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
1 & 0 & 0 & \alpha^5 & 0 & \alpha^4 & \alpha^5 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\alpha^6 & 0 & 0 & \alpha^2 & 0 & \alpha & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 0 & \alpha^4 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\alpha^5 & 0 & 0 & \alpha^5 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\alpha^2 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\alpha^4 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\alpha^5 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\end{pmatrix}.$$

Our goal is to calculate an error-locating function with low complexity from the matrix $S$. An algorithm for accomplishing this task is given in section IV, where we assume for clarity that all entries in $S$ are known. This is reasonable because the entries of matrix $S$ that are unknown when starting the algorithm can be determined whenever they are needed to continue the algorithm [19][20]. The inclusion of a procedure to determine the unknown entries in $S$ in our algorithm is postponed to section V.
IV  The algorithm

We denote by $S^{(a,b)}_i$ the submatrix of $S$ consisting of the elements in the $a+1$ first rows in the $b+1$ first columns of $S$, i.e.: $S^{(a,b)}_i = \|S_{ij}\|, 0 \leq i \leq a, 0 \leq j \leq b$.

Suppose we are given a function $\sigma \in L(bP_\infty) \setminus L((b-1)P_\infty)$ that solves the system of linear equations

$$S^{(a-1,b)}_i \sigma^T = 0, \quad \sum_{i=0}^b \sigma_i S_{a,i} = \Delta \neq 0.$$  

We then say that the function $\sigma$ gives a discrepancy with value $\Delta$ at position $(a,b)$. If a function $\sigma \in L(bP_\infty) \setminus L((b-1)P_\infty)$ solves the linear system of equations $S\sigma^T = 0$, we formally say that $\sigma$ gives a discrepancy in position $(\infty,b)$.

The idea of the proposed algorithm is to iteratively find nonzero functions $\sigma$ that solve an equation $S^{(a,b)}_i \sigma^T = 0$ for increasing numbers $r = a + b$. The following lemma provides a possibility to combine partial solutions.

**Lemma 3.1** Let a function $\sigma$ give a discrepancy with value $\Delta$ in position $(a,b)$. Moreover, suppose we are given a function $\lambda$ that gives a discrepancy with value 1 in position $(a,b'), b' \leq b$. Then the function $\sigma' = \sigma - \Delta \lambda$ gives a discrepancy in position $(a',b)$ with $a' > a$.

**Proof.** Clearly the equations

$$\sum_{i=0}^b S_{j,i} \sigma'_i = 0, \quad 0 \leq j < a$$

hold. Moreover, we have

$$\sum_{i=0}^b S_{a,i} \sigma'_i = \sum_{i=0}^b S_{a,i} \sigma_i - \Delta \sum_{i=0}^{b'} S_{a,i} \lambda_i = \Delta - \Delta = 0.$$

$\square$
The following lemma is one of the keys to a Berlekamp-Massey type algorithm for saving computational complexity.

**Lemma 3.2** Suppose we are given a function $\sigma$ that gives a discrepancy with value $\Delta$ in position $(a, b)$. Then the function $\phi_{\gamma} \sigma$ gives a discrepancy in position $(a', b + \gamma)$ with value $\Delta'$ and

$$a' = \begin{cases} > a - \gamma & a - \gamma \in G \\ a - \gamma & \text{otherwise.} \end{cases}$$

Moreover, if $a - \gamma \notin G$ holds we have $\Delta' = \Delta$.

**Proof.** With equation (3.2) and $\sigma = \sum_{l=0}^{b} \sigma_l \phi_l$ the assumption of the lemma implies

$$\sum_{j=0}^{n-1} e_j \sigma(P_j) \phi_i(P_j) = 0 \text{ for } i < a.$$ 

This means that

$$\sum_{j=0}^{n-1} e_j (\phi_{\gamma}(P_j) \sigma(P_j)) \phi_i(P_j) = 0 \text{ for } i < a - \gamma.$$ 

If $a - \gamma \notin G$ holds then the relation $\phi_a = \phi_{\gamma} \phi_{a - \gamma}$ is satisfied. The equation

$$\sum_{j=0}^{n-1} e_j \sigma(P_j) \phi_a(P_j) = \Delta$$

is rewritten as

$$\sum_{j=0}^{n-1} e_j (\phi_{\gamma}(P_j) \sigma(P_j)) \phi_{a - \gamma}(P_j) = \Delta.$$ 

Thus the claim $\Delta' = \Delta$ holds if $a - \gamma \notin G$. \qed

Lemma 3.1 and 3.2 can now be combined to give the basic induction step in the proposed algorithm.
**Lemma 3.3** Let $\sigma$ and $\lambda$ be two functions that give discrepancies in positions $(a, b)$ and $(a', b')$ with values $\Delta$ and 1. Moreover, let the numbers $a, b, a', b'$ satisfy the relations $a' + b' < a + b$ and $a' \equiv a \pmod{\gamma}$. We can construct a function $\sigma'$ that gives a discrepancy in position $(a'', b'')$ where $a'' + b'' > a + b$ by letting

$$\sigma' = \begin{cases} 
\sigma - \Delta \phi_{\gamma}^{(a'-a)/\gamma} \lambda & a \leq a' \\
\phi_{\gamma}^{(a-a')/\gamma} \sigma - \Delta \lambda & a' < a \end{cases}.$$  

(3.3)

**Proof.** We first consider the case when $a$ is less than or equal to $a'$. The multiplication of function $\lambda$ with $\phi_{\gamma}^{(a'-a)/\gamma} \lambda$ gives a discrepancy in position $(a, b' + (a' - a))$. On the other hand, $b > (b' + (a' - a))$ holds by assumption and we can invoke Lemma 3.1, which proves the claim. The proof of the case $a' < a$ is similar. $\phi_{\gamma}^{(a-a')/\gamma} \sigma$ gives a discrepancy in position $(a', b + (a - a'))$ which has value $\Delta$. The condition $b + (a - a') > b'$, needed in order to invoke Lemma 3.1, holds again by assumption. \qed

Lemma 3.2 suggests a partitioning of functions $\sigma$ into equivalence classes, where two functions are in the same equivalence class if their pole order at $P_{\infty}$ is equal modulo $\gamma$. On the other hand Lemma, 3.3 motivates a partitioning of functions $\lambda$ into equivalence classes. Here two functions are in the same equivalence class if they give discrepancies in rows with indices that are equivalent modulo $\gamma$. We formally define two sets of functions $\sigma^{(r,i)}$ and $\lambda^{(r,i)}$.

**Definition 3.2** A collection of functions $\sigma^{(r,i)}$ and $\lambda^{(r,i)}$ of pole order $b^{(r,i)}_{\sigma}$ and $b^{(r,i)}_{\lambda}$ is called valid at pole order $r$ if the following conditions hold.

- The functions $\sigma^{(r,i)}$ give discrepancies at positions $(a, b^{(r,i)}_{\sigma})$ where $b^{(r,i)}_{\sigma}$ is minimal such that $b^{(r,i)}_{\sigma} \not\in G$, $b^{(r,i)}_{\sigma} \equiv i \mod{\gamma}$ and $a + b^{(r,i)}_{\sigma} > r$. 


Moreover, we define numbers $a^{(r,i)}_{\sigma}$, $b^{(r,i)}_{\sigma}$ and $\Delta^{(r+1,i)}$ as
\[
\begin{align*}
a^{(r,i)}_{\sigma} &= r + 1 - b^{(r,i)}_{\sigma} \\
b^{(r,i)}_{\sigma} &= a^{(r,i)}_{\sigma} \mod \gamma \\
\Delta^{(r+1,i)} &= \begin{cases} 
\sum_{t=0}^{\sigma_{\gamma}} a^{(r,i)}_{\sigma} \cdot S_{a^{(r,i)}_{\sigma}} & a^{(r,i)}_{\sigma} \notin G \\
0 & a^{(r,i)}_{\sigma} \in G.
\end{cases}
\end{align*}
\]

- The functions $\lambda^{(r,j)}$ give discrepancies with value 1 at position $(a^{(r,j)}_{\lambda}, b^{(r,j)}_{\lambda})$, where $a^{(r,j)}_{\lambda}$ is maximal such that $a^{(r,j)}_{\lambda} \equiv j \mod \gamma$ and $a^{(r,j)}_{\lambda} + b^{(r,j)}_{\lambda} \leq r$. If no such function exists we define $\lambda^{(r,j)} = 0$ and $a^{(r,j)}_{\lambda}$ as
\[
a^{(r,j)}_{\lambda} = \max_{l \in G} \{ l : l \equiv j \mod \gamma \}.
\]

The relevance of the above definition of $\sigma^{(r,i)}$ for the decoding of AG codes becomes clear in the following lemma.

**Lemma 3.4** Let for a code of type $C_{\Omega}(D, mP_{\infty})$ an error vector $e$ of Hamming weight $t < (m - 2g + 2)/2 = d^*(C_{\Omega})/2$ be given. The space of error-locating functions has a basis $\phi^i_{\frac{1}{2}}(\sigma^{(m+2g+\gamma-1,j)})$ where $i \geq 0$ and $0 \leq i < \gamma$. Moreover, the error-locating function of lowest pole order at $P_{\infty}$ is given as the function of lowest pole order in the set $\{\sigma^{(m+g,j)}\}_{i=0}^{\gamma-1}$.

**Proof.** We first prove the second statement of the lemma. To this end, let $f$ be the function of lowest pole order in $\{\sigma^{(m+g,j)}\}_{i=0}^{\gamma-1}$. As a consequence of the Riemann-Roch Theorem the pole order of this function is less than $t + g + 1$ and $f$ solves the equation $S^{(m-t,t+g)}f = 0$. We have to prove that $f$ is error-locating. To this end, let $M(a)$ be the $(a + 1) \times n$ matrix with elements $M_{i,j} = \phi_i(P_j)$, $0 \leq i \leq a$, $0 \leq j \leq n-1$. Matrix $S^{(m-t,t+g)}$ can be decomposed into
\[
S^{(m-t,t+g)} = M(m-t) \text{diag}(e) M(t+g)^T
\]
where $\text{diag}(e)$ is the diagonal matrix with the vector $e$ on its main diagonal. The expression $\text{diag}(e) M(t+g)^T f$ is now identified as the column
IV. The algorithm

vector obtained by the component-wise multiplication of the vector $e^T$ and $(f(P_0), f(P_1), \ldots, f(P_{n-1}))^T$. On the other hand, it is a codeword in the code with parity check matrix $M(m - t)$. This code is an AG code of type $C_{O}(D, (m - t)P_{\infty})$ and has minimum distance $m - t - 2g + 2$, which by assumption is greater than $t$. It follows that the component-wise product of the vector $e$ and $(f(P_0), f(P_1), \ldots, f(P_{n-1}))$ is the all zero word and therefore $f$ is error-locating.

The first statement is proved by a similar argument. For an error vector of weight $t$ the maximum pole order of a function $\sigma^{(r,i)}$ equals $t + 2g - 1 + \gamma$ for any $r \geq 0$. This is a consequence of the the Riemann-Roch Theorem and the fact that there may not exist a function of pole order $t + 2g - 1$ that has the error positions as zeros. However, there is always such a function of pole order $t + 2g - 1 + \gamma$. Having bounded the maximal pole order of all $\sigma^{(r,i)}$ in this way, the proof proceeds as before. The statement that the space of error-locating functions has a basis $\phi_i^{l_i} \sigma^{(m+2g+\gamma-1,i)}$ where $l_i \geq 0$ and $0 \leq i \leq \gamma$ follows from the definition of the functions $\sigma^{(m+2g+\gamma-1,i)}$.

Lemma 3.2 and Definition 3.2 are the two cornerstones in the proposed algorithm. Given a collection of functions $\sigma^{(r,i)}$ and $\lambda^{(r,j)}$ we can find a collection of $\sigma^{(r+1,i)}$ and $\lambda^{(r+1,j)}$ by pairing together suitable $\sigma^{(r,i)}$ and $\lambda^{(r,j)}$ and applying Lemma 3.3, where the numbers $i$ and $j$ are chosen such that $a^{(r,i)}_\sigma \equiv a^{(r,j)}_\lambda \mod \gamma$. For any $0 \leq i < \gamma$ and $0 \leq j < \gamma$ there exist functions $\sigma^{(r,i)}$ and $\lambda^{(r,j)}$. It follows that we can make all necessary pairings. The important observation, in order to get a parallel algorithm, is that no two pairs of functions have a function in common. Lemma 3.3 then yields functions $\sigma^{(r+1,i)}$ directly and functions $\lambda^{(r+1,j)}$ are given as either $\lambda^{(r,j)}$ or a scalar multiple of $\sigma^{(r,i)}$.

In order to get a concise notation we associate to the functions $\sigma^{(r,i)}$ and $\lambda^{(r,j)}$ polynomials in one variable $z$ by letting

$$\tilde{\sigma}^{(r,i)}(z) = \sum_{l=0}^{b^{(r,i)}_\sigma} \sigma^{(r,i)}_l z^{b^{(r,i)}_\sigma - l}, \quad \tilde{\lambda}^{(r,i)}(z) = z^{r+1-(a^{(r,i)}_\lambda+b^{(r,i)}_\lambda)} \sum_{l=0}^{b^{(r,i)}_\lambda} \lambda^{(r,i)}_l z^{b^{(r,i)}_\lambda - l}.$$

We reformulate and extend Lemma 3.3 using these polynomials.
Lemma 3.5 Let two functions $\sigma^{(r,i)}$, and $\lambda^{(r,j)}$ with associated values $a_\sigma^{(r,i)}$, $a_\lambda^{(r,j)}$ and $\Delta^{(r+1,i)}$ be given such that the relation $a_\sigma^{(r,i)} \equiv a_\lambda^{(r,j)} \mod \gamma$ holds. From the corresponding polynomials $\bar{\sigma}^{(r,i)}(z)$, and $\bar{\lambda}^{(r,j)}(z)$ we can find $\bar{\sigma}^{(r+1,i)}(z)$ and $\bar{\lambda}^{(r+1,j)}(z)$ as

$$\bar{\sigma}^{(r+1,i)}(z) = \bar{\sigma}^{(r,i)}(z) - \Delta^{(r+1,i)} \bar{\lambda}^{(r,i)}(z)$$

and

$$\bar{\lambda}^{(r+1,j)}(z) = \begin{cases} (\Delta^{(r+1,i)})^{-1} z\bar{\sigma}^{(r,i)}(z) & (\Delta^{(r+1,i)} \neq 0) \land (a_\lambda^{(r,j)} < a_\sigma^{(r,i)}) \\ z\bar{\lambda}^{(r,j)}(z) & \text{otherwise.} \end{cases}$$

Proof. We must show that the polynomial $\bar{\sigma}^{(r+1,i)}(z)$ does indeed correspond to a function $\sigma^{(r+1,i)}$. Lemma 3.3 gives a possibility to combine functions $\sigma^{(r,i)}$ and $\lambda^{(r,j)}$. We use the notation of Lemma 3.3 and identify $\sigma^{(r,i)}$, $\lambda^{(r,j)}$ and $\sigma^{(r+1,i)}$ with $\sigma$, $\lambda$ and $\sigma'$. The pole-order difference of $\sigma$ and $\phi_{\gamma}^{(a'-a)/\gamma} \lambda$ in the case $a \leq a'$ equals the difference of pole-orders of $\phi_{\gamma}^{(a'-a)/\gamma} \sigma$ and $\lambda$ in the case $a' < a$. This difference is in both cases equal to $a + b - (a' + b')$. Let $\beta$ be the pole order of $\sigma$ or $\phi_{\gamma}^{(a'-a)/\gamma} \sigma$ respectively in the two cases of Lemma 3.3. Thus for the coefficients of the vector expansions of $\sigma'$, $\sigma$ and $\lambda$ we have

$$\sigma'_{b-l} = \sigma_{b-l} \text{ for } l = 0, 1, \ldots, a + b - (a' + b') - 1$$

and

$$\sigma'_{b-l} = \sigma_{b-l} - \Delta \lambda_{b-l} \text{ for } l = a + b - (a' + b'), a + b - (a' + b') - 1, \ldots, \beta.$$
IV. The algorithm

The definition of \( \bar{\lambda}^{(r,i)}(z) \) already incorporates the multiplication with \( z^{a+b-(a'+b')} \), which equals \( z^{r+1-a'+b'} \) as for the corresponding function \( \sigma^{(r,i)}(z) \) we have \( a^{(r,i)}_\sigma + b^{(r,i)}_\sigma = r + 1 \). Lemma 3.3 implies that the function \( \sigma' \) gives a discrepancy at a position \( (a, \beta) \) with \( a + \beta > r + 1 \). Moreover, the conditions \( \beta \not\in G \) and \( \beta \equiv i \mod \gamma \) hold. In order to prove equation (3.4) we have to show that \( \beta \) is the least number which allows such a solution and that consequently \( \bar{\sigma}^{(r+1,i)} \) can be chosen as \( \bar{\sigma}' \). We only have to consider the case \( a^{(r,i)}_\sigma > a^{(r,i)}_\lambda \) and \( \Delta^{(r+1,i)} \neq 0 \), as in all other cases we have \( b^{(r,i)}_\sigma = b^{(r+1,i)}_\sigma \). The constructed polynomial \( \sigma' \) corresponds to a function that gives a discrepancy in position \( (a'', r + 1 - a^{(r,i)}_\lambda) \), where we have \( a'' > a^{(r,i)}_\lambda \). Assume there exists a function \( \sigma'' \) with maximal pole order \( \beta'' \) with \( b^{(r,i)}_\sigma < \beta'' < r + 1 - a^{(r,i)}_\lambda \) that satisfies the required conditions for \( \sigma^{(r+1,i)} \). We then could construct a function \( \sigma'' + \theta^{(r+1,i)}(b^{(r+1,i)}_\sigma) \gamma^{(r,i)} \) which for a suitable choice of the constant \( \theta \) has pole order less than \( \beta'' \) and which gives a discrepancy at position \( (r + 1 - \beta'', < \beta'') \). This contradicts the definition of \( \lambda^{(r,i)} \).

Equation (3.5) follows from the fact that the only two possible candidates for \( \lambda^{(r+1,i)} \) are \( \lambda^{(r,i)} \) or a scalar multiple of \( \sigma^{(r,i)} \). Moreover, the multiplication with \( z \) required in the corresponding polynomial \( \bar{\lambda}^{(r,i)} \) is performed. \( \square \)

Let numbers \( l_i \) be defined as \( l_i = \min \{l : (l \equiv i \mod \gamma) \land (l \not\in G)\} \). We state the main algorithm as a set of recursive equations.

**Algorithm 3.1**

**Initialization:**

\[
\begin{align*}
\bar{\sigma}^{(-1,i)}(z) &= 1, \quad a^{(-1,i)}_\sigma = -l_i, \quad 0 \leq i < \gamma, \quad (3.6) \\
\bar{\lambda}^{(-1,j)}(z) &= 0, \quad a^{(-1,j)}_\lambda = l_j - \gamma, \quad 0 \leq j < \gamma, \quad (3.7)
\end{align*}
\]
Chapter 3. A fast parallel implementation of a BMA for AG codes

Iteration step:

\[ j^{(r,i)} = a^{(r,i)} \mod \gamma \]  \hspace{1cm} (3.8)

\[ \Delta^{(r+1,i)} = \begin{cases} 
0 & a^{(r,i)} \in G \\
\sum_{h=0}^{r+1-a^{(r,i)}} \bar{\sigma}^{(r,i)}_h S_{a^{(r,i)}_{r+1-a^{(r,i)}-h}} & \text{otherwise} 
\end{cases} \]  \hspace{1cm} (3.9)

\[ \delta^{(r,i)} = \begin{cases} 
1 & (\Delta^{(r+1,i)} \neq 0) \land (a^{(r,j^{(r,i)})}_\lambda < a^{(r,i)}_\sigma) \\
0 & \text{otherwise} 
\end{cases} \]  \hspace{1cm} (3.10)

\[ \begin{bmatrix} \bar{\sigma}^{(r+1,i)}(z) \\ \bar{\lambda}^{(r+1,j^{(r,i)})}(z) \end{bmatrix} = \begin{bmatrix} 1 & \Delta^{(r+1,i)} \\ \delta^{(r,i)} \Delta^{(r+1,i)-1} & 1 - \delta^{(r,i)} \end{bmatrix} \begin{bmatrix} \bar{\sigma}^{(r,i)}(z) \\ \bar{\lambda}^{(r,j^{(r,i)})}(z) \end{bmatrix} \]  \hspace{1cm} (3.11)

The quotient \( \delta^{(r,i)}/\Delta^{(r+1,i)} \) is understood to equal zero if \( \Delta^{(r+1,i)} \) equals zero.

\[ a^{(r+1,i)}_\sigma = (1 - \delta^{(r,i)}) a^{(r,i)}_\sigma + \delta^{(r,i)} a^{(r,j^{(r,i)})}_\lambda + 1 \]  \hspace{1cm} (3.12)

\[ a^{(r+1,j^{(r,i)})}_\lambda = (1 - \delta^{(r,i)}) a^{(r,j^{(r,i)})}_\lambda + \delta^{(r,i)} a^{(r,i)}_\sigma. \]  \hspace{1cm} (3.13)

**Theorem 3.1** The set of recursive equations in Algorithm 3.1 can be used to calculate a collection of polynomials \( \bar{\sigma}^{(s,i)}(z) \), \( 0 \leq i < \gamma \) and \( \bar{\lambda}^{(s,j^{(s,i)})}(z) \), \( 0 \leq j < \gamma \) for any \( s \geq 0 \). The corresponding functions \( \sigma^{(s,i)} \) and \( \lambda^{(s,j)} \) are valid at pole order \( s \).

**Proof.** The main part of the proof follows directly from Lemma 3.5. \( \delta^{(r,i)} \) is a steering variable used to distinguish the two cases of equation (3.5). Equations (3.12) and (3.13) follow from the proof of Lemma 3.5. The initialization of the algorithm is consistent with Definition 3.2. \( \square \)

Algorithm 3.1 has some interesting properties. The calculations of the \( \gamma \) polynomials \( \bar{\sigma}^{(s,i)} \) in iteration step \( s + 1 \) are independent of each other and can be performed in parallel. Given \( \gamma \) similar processing units capable of performing the calculations in equations (3.8) to (3.13), we obtain all polynomials \( \bar{\sigma}^{(s,i)} \) in \( s + 1 \) iterations. The complexity, that is required by any
Table 3.1: Table of intermediate results during one run of Algorithm 3.1

<table>
<thead>
<tr>
<th>r</th>
<th>$a^{(r,i)}_x$</th>
<th>$a^{(r,i)}_\lambda$</th>
<th>$\bar{\sigma}^{(r,i)}$</th>
<th>$\bar{\lambda}^{(r,i)}$</th>
<th>$\Delta^{(r+1,i)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>[0,-5,-7]</td>
<td>[-3,4,2]</td>
<td>[1,1,1]</td>
<td>[0,0,0]</td>
<td>$[\alpha^4,0,0]$</td>
</tr>
<tr>
<td>0</td>
<td>[-2,-4,-6]</td>
<td>[0,4,2]</td>
<td>[1,1,1]</td>
<td>$[z\alpha^3,0,0]$</td>
<td>[0,0,0]</td>
</tr>
<tr>
<td>1</td>
<td>[-1,-3,-5]</td>
<td>[0,4,2]</td>
<td>[1,1,1]</td>
<td>$[z^2\alpha^3,0,0]$</td>
<td>[0,0,0]</td>
</tr>
<tr>
<td>2</td>
<td>[0,-2,-4]</td>
<td>[0,4,2]</td>
<td>[1,1,1]</td>
<td>$[z^3\alpha^3,0,0]$</td>
<td>[0,0,0]</td>
</tr>
<tr>
<td>3</td>
<td>[1,-1,-3]</td>
<td>[0,4,2]</td>
<td>[1 + $z^2\alpha^4,1,1$]</td>
<td>$[z^4\alpha^3,0,0]$</td>
<td>[0,0,0]</td>
</tr>
<tr>
<td>4</td>
<td>[2,0,-2]</td>
<td>[0,4,2]</td>
<td>[1 + $z^3\alpha^4,1,1$]</td>
<td>$[z^5\alpha^3,0,0]$</td>
<td>$[0,\alpha^3,0]$</td>
</tr>
<tr>
<td>5</td>
<td>[3,1,-1]</td>
<td>[0,4,2]</td>
<td>$[1 + z \alpha^5,1 + z^5 \alpha^6,1]$</td>
<td>$[z^6\alpha^3,0,0]$</td>
<td>$[0,\alpha^3,0]$</td>
</tr>
<tr>
<td>6</td>
<td>[1,2,0]</td>
<td>[3,4,2]</td>
<td>$[1 + z^3 \alpha^4 + z^5 \alpha^6,1 + z^5 \alpha^6,1]$</td>
<td>$[z^6\alpha^5 + z^4 \alpha^3,0,0]$</td>
<td>[0,0,1]</td>
</tr>
<tr>
<td>7</td>
<td>[2,3,1]</td>
<td>[3,4,2]</td>
<td>$[1 + z^3 \alpha^4 + z^5 \alpha^6,1 + z^5 \alpha^6,1]$</td>
<td>$[z^2 \alpha^6 + z^5 \alpha^3,0,0]</td>
<td>[0,\alpha^2,0]</td>
</tr>
<tr>
<td>7</td>
<td>[3,4,2]</td>
<td>[3,4,2]</td>
<td>$[1 + z^3 \alpha^4 + z^6 \alpha^4,1 + z^5 \alpha^6]$</td>
<td>$[z^3 \alpha^6 + z^6 \alpha^3,0,0]$</td>
<td>$[0,\alpha^2,0]$</td>
</tr>
<tr>
<td>9</td>
<td>[4,5,3]</td>
<td>[3,4,2]</td>
<td>$[1 + z^3 \alpha^2 + z^6 \alpha^2]$</td>
<td>$[z^4 \alpha^6 + z^7 \alpha^3,0,0$</td>
<td>[0,0,0]</td>
</tr>
<tr>
<td>10</td>
<td>[5,3,4]</td>
<td>[3,4,5]</td>
<td>$[1 + z^3 \alpha^2 + z^6 \alpha^2]$</td>
<td>$[z^5 \alpha^6 + z^8 \alpha^3,0]$</td>
<td>$[\alpha^6,0,0]$</td>
</tr>
<tr>
<td>11</td>
<td>[6,4,5]</td>
<td>[3,4,5]</td>
<td>$[1 + z^4 \alpha^5 + z^3 + z^6 \alpha^2]$</td>
<td>$[z^6 \alpha^6 + z^9 \alpha^3,0]$</td>
<td>$[\alpha^6,0,\alpha^4]$</td>
</tr>
<tr>
<td>12</td>
<td>[4,5,6]</td>
<td>[6,4,5]</td>
<td>$[1 + z^5 \alpha^5 + z^3 + z^6 \alpha^3 + z^9 \alpha^2]$</td>
<td>$[z \alpha + z^6 \alpha^2]$</td>
<td>[0,1,0,0]</td>
</tr>
<tr>
<td>13</td>
<td>[5,6,7]</td>
<td>[6,4,5]</td>
<td>$[1 + z^5 \alpha^5 + z^3 + z^6 \alpha^3 + z^9 \alpha^2]$</td>
<td>$[\alpha^6 \alpha^2 z^3 + z^4 \alpha^2]$</td>
<td>$[\alpha^6 \alpha^2 z^3 + z^4 \alpha^2]$</td>
</tr>
</tbody>
</table>

**Example 3.3 (Example 3.1.3.2 continued)** We consider the setup of Example 3.1 and 3.2 and let Algorithm 3.1 operate on the matrix $S$, that is given in example 3.2. The intermediate results during the calculation of polynomials $\sigma^{(i)}$, $i = 0, 1, 2$ are given in table IV. The example is continued in section V where we determine the values of $S_{a,b}$ with $a + b$ $\geq$ 14, that are necessary to continue the algorithm. □
processing unit in one iteration step, is in the order \( O((\log_2(q))^2 s) \). Due to the fact that we have to perform \( s \) iterations in \( \gamma \) processing units we find that the overall complexity, in order to find all polynomials \( \sigma^{(s,i)} \), is not greater than \( O((\log_2(q))^2 \gamma s^2) \). In a parallel implementation the time requirements are determined by the time requirements for one of the \( \gamma \) processing units. Provided that these units work in a serial fashion the time requirements are in the order \( O(s^2) \) and thus essentially the same as for a serially implemented one-dimensional BMA designed to correct \( s \) errors in a codeword of an RS code. Any parallelization of the underlying architecture of the one-dimensional BMA helps speeding up the algorithm. We outline an implementation for codes on Hermitian curves in section VI, where also a more detailed treatment of complexity issues can be found.

V  Majority voting

The set of recursive equations provided in Algorithm 3.1 allows the determination of functions \( \sigma^{(r,i)} \) for all \( r \geq 0 \) presumed that we know the complete matrix \( S \). However, this is not the case as we only know entries \( S_{a,b} \) with \( a + b \leq m \). In [19] Feng and Rao give a procedure to iteratively determine \( S_{a,b} \) for \( a + b > m \) from the known part of \( S \), if the weight of the error vector does not exceed the error correction capability of the code. One key observation in this algorithm is that all \( S_{a,b} \) with \( a + b = m + w, w \geq 1 \) can be expressed in the form

\[
S_{a,b} = \sum_{i=0}^{m+w} \varepsilon_i^{(a,b)} S_{i,0}
\]

(3.14)

where the coefficients \( \varepsilon_i^{(a,b)} \) are known and determined by the underlying curve (cf. Example 3.2). Moreover, we have the following lemma.

**Lemma 3.6** Let \( B(m) = \{1, \phi_\gamma, \ldots, \phi_m\} \) be a standard basis for \( L(m) \). \( B(m) \) can be chosen in such a way that

\[
S_{a,b} = S_{a+b,0} + \sum_{i=0}^{a+b-1} \varepsilon_i^{(a,b)} S_{i,0}
\]

(3.15)
holds for \( a + b \leq m \) and all numbers \( a, b \notin G \).

**Proof.** Any function of pole order at most \( m \), and thus in particular the product \( \phi_a \phi_b \), can be expanded in the basis \( B(m) \), i.e.

\[
\phi_a \phi_b = \varepsilon_{a+b}^{(a,b)} \phi_{a+b} + \sum_{i=0}^{a+b-1} \varepsilon_i^{(a,b)} \phi_i.
\] (3.16)

We have to show that we can choose the functions \( \phi_i \) in such a way that \( \varepsilon_i^{(a,b)} \) equals one or equivalently that the expression \( \phi_a \phi_b - \phi_{a+b} \) has a pole order at \( P_\infty \) that is less than \( a + b \) for all non-gaps \( a \) and \( b \). Equation (3.15) follows then directly from the definition of the syndromes in equation (3.2).

Let \( t \) be a fixed local parameter at \( P_\infty \) that means a function with a simple zero at \( P_\infty \), cf.\([25, \text{ ch.2}]\). It is known \([25, \text{ ch.2}]\) that the functions \( \phi_i, i \notin G \) can uniquely be written as \( t^{-i}u_i \) where \( u_i \) is a rational function that has neither a zero nor a pole at \( P_\infty \). We choose the basis in such a way that all functions \( u_i \) evaluate to one at \( P_\infty \). This choice does not conflict with the defining properties of a standard basis. Now we can write \( \phi_a \phi_b - \phi_{a+b} \) as \( t^{-(a+b)}(u_a u_b - u_{a+b}) \). Clearly the expression \( u_a u_b - u_{a+b} \) evaluates to zero at \( P_\infty \) and can therefore be written as \( tl^'u' \), where \( l \) is a positive integer and \( u' \) a function with neither zeros nor poles at \( P_\infty \). Thus the pole order at \( P_\infty \) of the expression \( \phi_a \phi_b - \phi_{a+b} \) is less than \( a + b \) and the claim is proved. \( \Box \)

In the light of Lemma 3.6 we from now on can assume that the coefficients \( \varepsilon_i^{(a,b)} \) equal one in equation (3.14). We will explicitly treat the case when \( w \) in equation (3.14) equals 1. The cases \( w > 1 \) are iteratively treated in a completely similar fashion.

The second key observation is the fact that the rank of \( S \) equals the weight \( t \) of the error vector. To see this, let \( M \) be a matrix with entries defined as \( M_{i,j} = \phi_i(P_j) \). \( S \) can be described as \( S = M \text{diag}(e) M^T \), where \( \text{diag}(e) \) is a diagonal matrix which contains the vector \( e \) on its main diagonal. The rank of \( S \) equals \( t \), as we have \( \text{rank}(M) = n \) and \( \text{rank}(\text{diag}(e)) = t \).

In order to describe the procedure of Feng and Rao we use the notion of a matrix discrepancy. We say that \( S \) has a matrix discrepancy at position
(a, b) if the following three conditions are satisfied

\[
\text{rank}(S^{(a,b)}) = \text{rank}(S^{(a-1,b)}) + 1 \\
= \text{rank}(S^{(a,b-1)}) + 1 \\
= \text{rank}(S^{(a-1,b-1)}) + 1.
\]

**Lemma 3.7** The rank of \(S^{(a,b)}\) equals the number of matrix discrepancies at positions \((a', b')\), \(a' \leq a, b' \leq b\). Moreover, any column and row of \(S\) contains at most one matrix discrepancy.

**Proof.** The rank of \(S^{(a,b)}\) is equal to the number of rows (columns) in \(S^{(a,b)}\) that contain a matrix discrepancy. This follows from the observation that any fixed row (column) does not contain a matrix discrepancy if and only if it is a linear combination of the previous rows (columns). It follows that the number of columns and rows that hold a matrix discrepancy is equal for any matrix \(S^{(a',b')}\) with \(a' \leq a, b' \leq b\). This in turn implies that any row and any column contains only one matrix discrepancy.

Discrepancies that are given by functions in Algorithm 3.1 and matrix discrepancies are related in the following lemma.

**Lemma 3.8** Let two polynomials \(\tilde{\sigma}^{(r,i)}(z)\) and \(\tilde{\lambda}^{(r,j)}(z)\) with \(j \equiv a^{(r,i)} \mod \gamma\) and \(\Delta^{(r+1,i)} \neq 0\) be given. Then \(S\) has

\[
\max\{0, \gamma^{-1}(a^{(r,i)} - a^{(r,j)})\}
\]

matrix discrepancies in positions

\[
(a^{(r,i)} - l\gamma, r + 1 - a^{(r,i)} + l\gamma) \text{ for } l = 0, 1, \ldots, \gamma^{-1}(a^{(r,i)} - a^{(r,j)}) - 1.
\]

Moreover, \(S\) has no matrix discrepancies in positions \((a, r + 1 - a), a \equiv a^{(r,i)} \mod \gamma\) outside the positions given in equation (3.17).

**Proof.** We first consider the general situation that a function \(\sigma\) gives a discrepancy at position \((a, b)\). This implies on the one hand that the \(a\)-th row of \(S^{(a,b)}\) is not a linear combination of the rows of \(S^{(a-1,b)}\), on the other hand
the $b$-th column of $S^{(a-1,b)}$ is linear dependent on the columns of $S^{(a-1,b-1)}$ with a relation described by $\sigma$. Thus we have

$$\text{rank}(S^{(a,b)}) = \text{rank}(S^{(a-1,b)}) + 1 = \text{rank}(S^{(a-1,b-1)}) + 1.$$ 

We note that the relation $\text{rank}(S^{(a,b)}) = \text{rank}(S^{(a-1,b)}) + 1$ implies that $S^{(a,b)}$ has a matrix discrepancy in the $a$-th row.

Now by the definition of $\sigma^{(r,i)}$ we can not have a matrix discrepancy in positions $(a, r + 1 - a), a \equiv a^{(r,i)}_\sigma, a > a^{(r,i)}_\sigma$. In positions $(a, r + 1 - a), a \equiv a^{(r,i)}_\sigma, a \leq a^{(r,i)}_\sigma, a \notin G$ we have a function which gives a discrepancy, namely $\phi^{(r,i)}_\sigma$. Thus it follows that the rows in $S$ pointed out by these discrepancies contain matrix discrepancies. Moreover, we have matrix discrepancies in positions $(a, r + 1 - a), a \equiv a^{(r,i)}_\sigma, a^{(r,i)}_\sigma < a \leq a^{(r,i)}_\sigma$, because all other positions of matrix discrepancies in the $a$-th row are contradicted by the definition of $a^{(r,i)}_\sigma$. The definition of $\lambda^{(r,i)}$ now implies that we have matrix discrepancies in positions $(a, < r + 1 - a), a \equiv a^{(r,i)}_\sigma, a < a^{(r,i)}_\sigma$. Because every row can hold at most one matrix discrepancy the claim of the lemma is proved.

Assume that Algorithm 3.1 has run up to iteration step $m$. In order to continue the algorithm we need to know the entries $S_{a,b}$ of $S$ with $a+b = m+1$. The strategy in determining $S_{m+1,0}$ and thus all $S_{a,b}$ with $a+b = m+1$ is to minimize the rank of $S$ or, equivalently, the number of matrix discrepancies in positions $(a,b)$ with $a+b = m+1$. Lemma 3.8 provides the necessary tool for doing this. It is proved in [19][20] that indeed more than half of the matrix discrepancies that may appear in the matrix $S$ at positions $(a,b)$ with $a+b = m+1$ can be removed. This is done by choosing $S_{m+1,0}$ in such a way that most of the discrepancy values given by functions $\sigma^{(r,i)}$ are forced to zero. The proposed procedure starts by choosing an estimate $\hat{S}_{m+1,0} = 0$, which according to equation (3.14) at the same time induces all other estimates $\hat{S}_{a,b}$ with $a+b = m+1$. For all $i$ such that $a^{(r,i)}_\sigma \notin G$ we then calculate numbers

$$\hat{\Delta}^{(m+1,i)} = \hat{S}_{a^{(m,i)}_\sigma,m+1-a^{(m,i)}_\sigma} + \sum_{h=1}^{m+1-a^{(m,i)}_\sigma} \phi^{(m,i)}_h S_{a^{(m,i)}_\sigma,m+1-a^{(m,i)}_\sigma-h}.$$ (3.18)
We remark that the constant term in polynomials $\bar{s}^{(r,i)}(z)$ always equals 1 when Algorithm 3.1 is properly initialized. Comparing equation (3.18) with the calculation of $\Delta^{(r,i)}$ in equation (3.9) and using the expansion of $S_{\alpha,b}$ in syndromes $S_{r,0}$, $0 \leq r \leq a + b$ given in equation (3.14) we can write

$$\hat{\Delta}^{(m+1,i)} = \Delta^{(m+1,i)} - S_{m+1,0}.$$ 

It follows that $\Delta^{(m+1,i)}$ equals $\hat{\Delta}^{(m+1,i)} + S_{m+1,0}$ and we invoke a majority scheme to determine $S_{m+1,0}$ such that most $\Delta^{(m+1,i)}$ equal zero, where we count each $\Delta^{(m+1,i)}$ with multiplicity $\max\{0, a^{(m,i)}_\sigma - a^{(m,i)}_\lambda\}$ according to Lemma 3.8. We formalize the procedure in the following algorithm.

**Algorithm 3.2**

1. Calculate $\hat{\Delta}^{(m+1,i)}$ for all $i$ where $a^{(m,i)}_\sigma \not\in G$ and let $D$ be the set of numbers $\Delta^{(m+1,i)}$.

2. Find the element $\hat{S} \in D$ with greatest associated value

$$\max_{\hat{\Delta}^{(m+1,i)} = \hat{S}} \{0, a^{(r,i)}_\sigma - a^{(r,i)}_\lambda\}.$$

3. Calculate $S_{m+1,0}$ and $\Delta^{(r,i)}$ as

$$S_{m+1,0} = -\hat{S}, \quad \Delta^{(r,i)} = \hat{\Delta}^{(m+1,i)} + S_{m+1,0}.$$

**Theorem 3.2** Let all entries $S_{\alpha,b}$ in $S$ with $a + b \leq m$ and a collection of polynomials $\bar{s}^{(m,i)}$ and $\bar{\lambda}^{(m,i)}$ with associated values $a^{(m,i)}_\sigma, a^{(m,i)}_\lambda$ be given. Algorithm 3.2 can be used to determine $S_{m+1,0}$ and $\Delta^{(m+1,i)}$ if the condition $t < (m - 2g + 2)/2 = d^*(C_{t})/2$ holds.

**Proof.** The relation $\Delta^{(m+1,i)} = \hat{\Delta}^{(m+1,i)} + S_{m+1,0}$ together with Lemma 3.8 yields that, in order to avoid matrix discrepancies in positions $(a, m+1-a)$, $a \equiv a^{(m+1,i)} \mod \gamma$, we must choose $S_{m+1,0}$ as $-\hat{\Delta}^{(m+1,i)}$. The second step of the algorithm can be described as choosing the unknown $S_{m+1,0}$ in such a way that the observed rank of $S$ is minimized given that we know all $S_{\alpha,b}$ with $a + b \leq m$ and are free to choose $S_{m+1,0}$. For a proof that this strategy uniquely determines $S_{m+1,0}$ we refer to [19]. In the third step $S_{m+1,0}$ and $\Delta^{(r,i)}$ are explicitly calculated. □
Remark 3.1 A proof of Theorem 3.2 that does not rely on the results of [19] is postponed to section VII, where the proof of a similar theorem is given that covers the case of error-erasure decoding and contains Theorem 3.2 as special case.

Theorem 3.2 opens the possibility to calculate the set $\sigma^{(r,i)}$ for arbitrary $r \geq 0$ using Algorithm 3.1, as we iteratively can determine any entry in $S$ when it is needed. In order to do this we have to replace equation (3.9) in Algorithm 3.1 with Algorithm 3.2. The complexity of the majority voting algorithm is mainly determined by finding the maximum in a set of at most $\gamma$ elements. This does not affect the overall complexity of the algorithm, which is still upper bounded by $O((\log_2(q))^2 \gamma m^2)$.

Example 3.4 (Example 3.1, 3.2, 3.3 continued) We continue the algorithm of example 3.3. In example 3.3 we could not determine the numbers $\Delta^{(14,i)}$ as we did not know the syndrome $S_{14,0}$. According to Algorithm 3.2 we assume that $\hat{S}_{14,0}$ equals zero. The only syndrome $\hat{S}_{a,b}$ with $a + b = 14$ that is nonzero under this assumption is $\hat{S}_{7,7}$, which equals $S_{7,0} = 1$. With polynomials $\sigma^{(13,i)}$ as given in table IV we calculate the numbers $\hat{\Delta}^{(14,0)} = \alpha$, $\hat{\Delta}^{(14,1)} = \alpha$ and $\hat{\Delta}^{(14,2)} = \alpha$. Thus there is only one element in the set $D$ in Algorithm 3.2 and its associated value

$$\sum_{i : \Delta^{(14,i)} = \alpha} \max \{0, a^{(14,i)}_\sigma - a^{(14,j^{(14,i)})}_\lambda\}$$

equals $(5 - 5) + (6 - 6) + (7 - 4) = 3$. It follows that $S_{14,0}$ equals $\alpha$ and $\Delta^{(14,0)} = \Delta^{(14,1)} = \Delta^{(14,2)} = 0$. We can add the following rows to table IV

<table>
<thead>
<tr>
<th>$r$</th>
<th>$a^{(r,i)}_\sigma$</th>
<th>$a^{(r,j)}_\lambda$</th>
<th>$\sigma^{(r,j)}$</th>
<th>$\chi^{(r,j)}$</th>
<th>$\Delta^{(r+1,i)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>$[\ldots, \ldots]$</td>
<td>$[\ldots, \ldots]$</td>
<td>$[\ldots, \ldots]$</td>
<td>$[\ldots, \ldots]$</td>
<td>$[0, 0, 0]$</td>
</tr>
<tr>
<td>14</td>
<td>$[6, 7, 8]$</td>
<td>$[6, 4, 5]$</td>
<td>$[1 + z^5 \alpha^5 + z^3 \alpha^3 \alpha^2 + z^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 + z \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2 \alpha^2</td>
<td>$ $[?, ?, ?]$</td>
<td></td>
</tr>
</tbody>
</table>

In a similar fashion we determine $S_{15,0}$ and the polynomials $\sigma^{(15,i)}$, which
correspond to functions $\sigma^{(15,i)}$, as

$$\begin{align*}
\hat{\sigma}^{(15,0)} &= 1 + \alpha^5 z + \alpha^9 z^2 + \alpha^4 z^3 + \alpha^6 z^4 + z^5 \\
\sigma^{(15,0)} &= \phi_3 + \alpha^5 \phi_5 + \alpha^2 \phi_5 + \alpha^6 \phi_3 + 1 \\
\hat{\sigma}^{(15,1)} &= 1 + \alpha z^2 + \alpha^6 z^2 + \alpha^4 z^3 + \alpha^9 z^4 + z^5 \\
\sigma^{(15,1)} &= \phi_8 + \alpha \phi_8 + \alpha^6 \phi_5 + \alpha^2 \phi_3 + \alpha^6 \\
\hat{\sigma}^{(15,2)} &= 1 + \alpha^6 z^2 + \alpha^2 z^4 + z^7 \\
\sigma^{(15,2)} &= \phi_7 + \alpha^6 \phi_5 + \alpha^2 \phi_3 + 1.
\end{align*}$$

The functions $\sigma^{(15,i)}$, $i = 0, 1, 2$ are error-locating and their common zero set are precisely the error points.

VI Implementation

The equation-set of Algorithm 3.1 is surprisingly similar to the equation-set of a one-dimensional BMA. This makes it possible to design an implementation based on $\gamma$ copies of a modified one dimensional BMA. For exemplification we choose the serial implementation of a BMA proposed by Blahut. For details on this implementation see [15, p.189]. It should be observed, however, that to a certain extent any implementation of a one-dimensional BMA can serve as a building block and that parallel implementations of a BMA further reduce the time requirements of the algorithm.

As remarked in sections IV and V the implementation of the algorithms of Theorems 3.1 and 3.2 differ because the underlying curves are different. Here we exemplify the implementation for Hermitian curves over a field of characteristic 2. The modifications needed for other curves are discussed at the end of this section.

The Hermitian curve $\mathcal{H}$ over $\mathbb{F}_{q^2}$ may be described by the equation

$$\mathcal{H} : X^{q+1} = ZY^q + Z^q Y.$$  \hspace{1cm} (3.19)

$\mathcal{H}$ has genus $g = \frac{(q-1)q}{2}$, contains one point at infinity $P_\infty = (0 : 1 : 0)$ and $q^3$ affine points $P_i$. Using the notation $x = X/Z$ and $y = Y/Z$ it
VI. Implementation

is known [26], that a basis for \(L(mP_\infty)\) is given by functions \(x^iy^j\) where \(iq + j(q + 1) \leq m\) and \(0 \leq i, 0 \leq j < q\). We can define functions \(\phi_l\) in the following way

\[
\phi_l = \begin{cases} 
  x^iy^j & x^iy^j \in L(lP_\infty) \setminus L((l-1)P_\infty), \quad j < q \\
  0 & L(lP_\infty) = L((l-1)P_\infty).
\end{cases}
\]

For Hermitian curves the number \(\gamma\) equals \(q\). In the rest of this section we will write \(\gamma\) in a statement if it is valid for any algebraic curve and \(\gamma\) if the value is specific for Hermitian curves.

It is desirable to avoid using the complete matrix \(S\). This is possible as only the entries \(S_{a,0}\) and knowledge of the curve \(\mathcal{H}\) is needed for reconstructing the desired part of \(S\) at every step of the algorithm. For Hermitian curves we have the following lemma.

**Lemma 3.9** Let the curve \(\mathcal{H}\) be defined by the equation \(\mathcal{H} : X^{q+1} = ZY^q + Z^qY\). The entries \(S_{a,b}\) of the matrix \(S\) satisfy the relations

\[
S_{a,b} = \begin{cases} 
  0 & (a \in G) \lor (b \in G) \\
  S_{a+b,0} & (a \mod q) + (b \mod q) < q \\
  S_{a+b,0} - S_{a+b-q^2+1,0} & \text{otherwise}.
\end{cases}
\]

**Proof.** In cases \((a \in G)\) or \((b \in G)\) this follows from the fact that \(\phi_a = 0\) or \(\phi_b = 0\). In the other cases the functions \(\phi_a\) and \(\phi_b\) satisfy

\[
\phi_a = x^{(a-q+1)(a \mod q)}/y^{(a \mod q)}
\]

and

\[
\phi_b = x^{(a+1)(b \mod q)}/y^{(b \mod q)}.
\]

If the condition \((a \mod q) + (b \mod q) < q\) holds, we have \(\phi_a\phi_b = \phi_{a+b}\) and the second case follows. If the condition \((a \mod q) + (b \mod q) \geq q\) holds, \(\phi_a\phi_b\) equals

\[
\phi_a\phi_b = x^{(a+b-(q+1))(a \mod q)+(b \mod q)}/y^{(a \mod q)+(b \mod q)}
\]

and the exponent of \(y\) lies in the range \(q, q + 1, \ldots, 2(q - 1)\). Using the defining equation of \(\mathcal{H}\) in the affine form, \(y^q = x^{q+1} - y\), we reduce the exponent of \(y\), which yields the relation \(\phi_a\phi_b = \phi_{a+b} - \phi_{a+b-q^2+1}\). The third case follows from this observation. \(\Box\)
We define a $q$-periodic sequence $h(a, b)$ in the following way

$$h(a, b) = \begin{cases} 0 & (a \mod q) + (b \mod q) < q \\ 1 & \text{otherwise}. \end{cases}$$

The calculation of $\Delta^{(r+1,i)}$ in Theorem 3.1 can now be rewritten as

$$\Delta^{(r+1,i)} = \sum_{l=0}^{r+1-a^{(r,i)}_{\nu}} \sigma^{(r,i)}_l (S_{r+1-l,0} - h(a^{(r,i)}_{\nu}, b^{(r,i)}_{\nu} - l)S_{r-q^2+2-l,0}). \quad (3.20)$$

We note that not all the quantities $S_{r-q^2+2-l,0}$ are defined. However, if $r - q^2 + 2 - l$ becomes negative either $h(a^{(r,i)}_{\nu}, b^{(r,i)}_{\nu} - l)$ or $\sigma^{(r,i)}_l$ equals zero. Equation (3.20) can be realized in the shift register circuit outlined in figure 3.1, where the switch is steered by the sequence $h(a^{(r,i)}_{\nu}, b^{(r,i)}_{\nu} - l)$, $l = 0, 1, \ldots, b^{(r,i)}_{\nu}$.

The same circuit is used for calculating $\tilde{\Delta}^{(r+1,i)}$ in Theorem 3.2. The only difference to the calculation of $\Delta^{(r+1,i)}$ is that initially $S_{r+1,0}$ is set to zero.

The equation set of Algorithm 3.1 shows a second main difference to a one-dimensional BMA. The polynomials $\bar{\lambda}^{(r,j)}$ used in the $(r + 1)$th iteration for updating $\bar{\sigma}^{(r,i)}$ change in the next iteration according to the definition of $j^{(r,i)}$. However they change in a very regular way as becomes clear in the following lemma.
Lemma 3.10 The numbers \( j^{(r,i)} \) satisfy the relation
\[
j^{(r+1,i)} = j^{(r,(i+1)) \mod \gamma}
\]

Proof. \( j^{(r,i)} \) is defined as \( a_{\sigma}^{(r,i)} \mod \gamma \). From equation (3.12) we have
\[
a_{\sigma}^{(r+1,i)} \equiv a_{\sigma}^{(r,i)} + 1 \mod \gamma.
\]
The lemma follows.

Lemma 3.10 implies that the updating polynomials \( \tilde{\lambda}^{(r,j)} \) are passed cyclically between the processing units for calculating \( \tilde{\sigma}^{(r,i)} \). Figure 3.2 gives the outline of a circuit that can be used to calculate the polynomials \( \tilde{\sigma}^{(2\ell-1,i)} \).

In this outline knowledge of \( S_{a,0} \) for \( 0 \leq a \leq 2\ell - 1 \) is presumed. The syndrome register in the implementation has length \( 2\ell + 1 \) and is initially loaded with the sequence \( \{S_{0,0}, 0, S_{2\ell-1,0}, S_{2\ell-2,0}, \ldots, S_{1,0}\} \). The registers for the coefficients of \( \tilde{\sigma}^{(r,i)} \) have length \( \ell \) and are loaded with a single one in the leftmost position. The registers for the coefficients of \( \tilde{\lambda}^{(r,j)} \) have length \( \ell + 1 \) and are loaded with zeros. For simplicity we assume that the length of all registers is sufficient to hold all coefficients of the polynomials \( \tilde{\sigma}^{(r,i)} \) and \( \tilde{\lambda}^{(r,j)} \). A typical iteration starts by calculating all \( \Delta^{(r+1,i)} \). This requires \( \ell \) clock cycles. During the following \( \ell \) clock cycles the updating of \( \tilde{\sigma}^{(r,i)} \) is performed. At the same time, according to Lemma 3.10, the contents of the \( \tilde{\lambda}^{(r,j)} \) registers are replaced cyclically. The multiplication of \( \tilde{\lambda}^{(r,j)} \) respectively \( \tilde{\sigma}^{(r,i)} \) by \( z \) is realized simultaneously as the \( \tilde{\lambda}^{(r,j)} \) registers have length \( \ell + 1 \). A similar trick applied to the syndrome register of length \( 2\ell + 1 \) yields that after \( 2\ell \) clock cycles the contents of all registers are in the proper state to start the next iteration.

In order to include the majority voting algorithm of Theorem 3.2 the circuit of figure 3.2 has to be modified as shown in figure 3.3. We want to determine polynomials \( \tilde{\sigma}^{(2\ell-1,i)} \) with initial knowledge of \( S_{a,0} \) for \( 0 \leq a \leq m \) where \( m < 2\ell - 1 \). We initialize the syndrome register with all known \( S_{a,0} \) and additional \( 2\ell - m \) zeros i.e. the sequence \( \{S_{0,0}, 0, \ldots, 0, S_{m,0}, S_{m-1,0}, \ldots, S_{1,0}\} \). During the first \( m \) iterations the circuit works just as the circuit in figure 3.2. In the \( (m + 1) \)th iteration the set of \( \tilde{\Delta}^{(m+1,i)} \) is calculated automatically and fed to a majority voting device which calculates and stores \( S_{m+1,0} \). The \( \tilde{\Delta}^{(m+1,i)} \) are then modified according to the equation
\[
\Delta^{(m+1,i)} = \tilde{\Delta}^{(m+1,i)} + S_{m+1,0}
\]
to equal \( \Delta^{(m+1,i)} \) and the updating is performed as before. Moreover, the correct value of \( S_{m+1,0} \) is supplied for the calculation of \( \tilde{\Delta}^{(m+2,i)} \) when it is needed and subsequently fed to the syndrome register.
Remark 3.2  The register lengths of the implementations outlined in Figures 3.2 and 3.3 are chosen in order to simplify the timing of the circuit. On the other hand the syndromes $S_{a \beta}$ are known to be zero for all $a \in G$. At the expense of a more complicated timing the syndrome register can be made shorter in order to avoid these expendable syndromes. Similar considerations allow the use of shorter registers for the coefficients of the polynomials $\tilde{\sigma}^{(r,j)}$ and $\tilde{\lambda}^{(r,j)}$.

The outlined circuits describe an implementation for codes on Hermitian curves. The differences in implementation for codes from other curves are
relatively minor. We have a different initialization of the $a_{\beta}^{(-1,i)}$ and $a_{\lambda}^{(-1,i)}$ due to a different set of gaps $G$. The implementation is affected through the number of parallel processing units. A more substantial difference is found in the number of relations that is needed to describe the matrix $S$ given the entries $S_{a,\beta}$, $a \geq 0$. In the case of codes on Hermitian curves we only have one nontrivial relation given in Lemma 3.9. In the general case we have at most $\gamma(\gamma - 1)/2$ (usually much less) different relations between the entries in $S_{a,\beta}$. 

Figure 3.3: Outline for an implementation of a Berlekamp-Massey type algorithm for codes on Hermitian curves with included majority voting.
VII Error-erasure decoding

Let a vector $y = c + e \in \mathbb{F}_q^n$, $c \in C_\Omega(D, mP_\infty)$ be given, where $e$ is a vector of weight at most $t + \rho$. Moreover, we assume knowledge of a set $\mathcal{R}$ of $\rho$ positions in $y$ such that the weight of $e$ outside $\mathcal{R}$ is at most $t$. The positions in $\mathcal{R}$ are called erasure positions. The problem of error-erasure decoding consists of finding the error vector $e$ of minimal weight outside $\mathcal{R}$. It has important applications in concatenated coding schemes and soft decoding algorithms for block codes. It is well known that a unique minimum weight error vector $e$ exists if the relation $2t + \rho < d(C_\Omega(D, mP_\infty))$ is satisfied. We assume that the values in erasure positions of $y$ are all defined. If this is not the case, which might occur in certain applications, we set the corresponding values to zero. For AG codes it is natural to introduce an erasure divisor $\mathcal{R}$ defined as $\mathcal{R} = \sum_{i \in \mathcal{R}} P_i$.

Again our goal is to find in an efficient way nontrivial functions that locate errors. Let a matrix $S$ be defined as before. All functions considered are still expanded in the standard basis of Definition 3.1.

Lemmata 3.1-3.3 hold without further requirements on the functions $\sigma$ and $\lambda$. Especially if functions $\sigma$ and $\lambda$ in Lemma 3.3 have a common zero set $\mathcal{R}$ the constructed function $\sigma'$ also has zeros in $\mathcal{R}$. As these three lemmata formulate the fundamental observations for Theorem 3.1 we can construct a similar theorem for the case of error-erasure decoding. The following definition of functions $\mu^{(r,i)}$ and $\nu^{(r,i)}$ replaces in this case Definition 3.2.

**Definition 3.3** A collection of functions $\mu^{(r,i)}$ and $\nu^{(r,i)}$ of pole order $t^{(r,i)}_\mu$ and $t^{(r,i)}_\nu$ is called valid at pole order $r$ for a set of erasure positions $\mathcal{R}$, if the following conditions hold.

- $\mu^{(r,i)}(P_i) = 0$ and $\nu^{(r,i)}(P_i) = 0$ for all $i \in \mathcal{R}$.
- The functions $\mu^{(r,i)}$ give discrepancies at positions $(a, b^{(r,i)}_\mu)$ where $b^{(r,i)}_\mu$ is minimal such that $b^{(r,i)}_\mu \equiv i \mod \gamma$ and $a + b^{(r,i)}_\mu > r$. Moreover,
we define numbers \( a^{(r,i)}_\mu \), \( j^{(r,i)} \) and \( \Delta^{(r+1,i)} \) as
\[
\begin{align*}
    a^{(r,i)}_\mu &= r + 1 - b^{(r,i)}_\mu \\
    j^{(r,i)} &= a^{(r,i)}_\mu \mod \gamma \\
    \Delta^{(r+1,i)} &= \begin{cases} 
        \sum_{l=0}^{b^{(r,i)}_\mu} \mu^{(r,i)}_l S_{a^{(r,i)}_\mu} \quad a^{(r,i)}_\mu \notin G \\
        0 & a^{(r,i)}_\mu \in G
    \end{cases}
\end{align*}
\]

- The functions \( \nu^{(r,j)} \) give discrepancies at positions \( (a^{(r,j)}_\nu, b^{(r,j)}_\nu) \) with value 1 where \( a^{(r,j)}_\nu \) is maximal such that \( a^{(r,j)}_\nu \equiv j \mod \gamma \) and \( a^{(r,j)}_\nu + b^{(r,j)}_\nu \leq r \). If no such function exists we define \( \nu^{(r,j)} = 0 \) and \( a^{(r,j)}_\nu \) as
\[
a^{(r,j)}_\nu = \max \{ l : l \equiv j \mod \gamma \}.
\]

**Lemma 3.11** Let \( \mathbf{e} \) be an error vector of weight \( t < (m - 2g + 2 - \rho)/2 \) outside positions \( \mathcal{R} \). The space of error-erasure-locating functions has a basis \( \phi_{\mu}^{(m+2g+\gamma-1,i)} \) where \( l_i \geq 0 \) and \( 0 \leq i < \gamma \). Moreover, the error-erasure-locating function with lowest pole order at \( P_\infty \) is given as the function of lowest pole order in the set \( \{ \mu^{(m+g,i)} \}_{i=0}^{\gamma-1} \).

**Proof.** After replacing \( \sigma^{(r,i)} \) with \( \mu^{(r,i)} \) and some obvious modifications, the proof is identical to the proof of Lemma 3.4 and therefore omitted.

To functions \( \mu^{(r,i)} \) and \( \nu^{(r,j)} \) we again associate polynomials in one variable in the same way as we did for \( \sigma^{(r,i)} \) and \( \lambda^{(r,j)} \). The similarity of all involved definitions and lemmata in the case of error-only and error-erasure decoding suggests that essentially the same algorithm as in Theorem 3.1 can be used to determine error-erasure-locating functions. In order to guarantee that functions \( \mu^{(r,j)} \) and \( \nu^{(r,j)} \) have zeros in the erasure positions, we have to initialize the algorithm with functions that have zeroes in the points of \( \mathcal{R} \). To this end, let \( \eta^{(i)} \) be the nontrivial function of lowest pole order \( b^{(i)} \) such that \( \eta^{(i)} \in L(mP_\infty - R) \), \( b^{(i)} \equiv i \mod \gamma \). Without loss of generality we
assume that the coefficient $\eta_{h(i)}$ in the vector expansion of $\eta^{(i)}$ equals one. The polynomial $\tilde{\eta}^{(i)}(z)$ is then defined as

$$\tilde{\eta}^{(i)}(z) = \sum_{l=0}^{b^{(i)}} \tilde{\eta}_l^{(i)} z^{b^{(i)}-l}.$$ 

Let $r_0$ be defined as the minimum over the values $b^{(i)}$, $0 \leq i < \gamma$. We recall that numbers $l_i$ were defined as $l_i = \min\{l : (l \equiv i \mod \gamma) \land (l \not\in G)\}$. The following set of recursive equations states an algorithm for the calculation of an error-erasure-locating function.

**Algorithm 3.3**

**Initialization:**

$$\tilde{\mu}^{(r_0-1,i)}(z) = \tilde{\eta}^{(i)}(z), \quad a^{(r_0-1,i)}_\mu = r_0 - b^{(i)}, \quad 0 \leq i < \gamma,$$

$$\tilde{\nu}^{(r_0-1,j)}(z) = 0, \quad a^{(r_0-1,j)}_\nu = l_j - \gamma, \quad 0 \leq j < \gamma,$$

**Iteration step:**

$$j^{(r,i)} = a^{(r,i)} \mod \gamma$$

$$\Delta^{(r+1,i)} = \begin{cases} 0 & a^{(r,i)}_\mu \in G \\ \sum_{h=0}^{r+1-a^{(r,i)}_\mu} \tilde{p}^{(r,i)}_h a^{(r,i)}_{\mu-h} & \text{otherwise} \end{cases}$$

$$\delta^{(r,i)} = \begin{cases} 1 & (\Delta^{(r+1,i)} \neq 0) \land (a^{(r,j)^{(r,i)}}_\nu < a^{(r,i)}_\mu) \\ 0 & \text{otherwise} \end{cases}$$

$$[\begin{array}{c} \tilde{\mu}^{(r+1,i)}(z) \\ \tilde{\nu}^{(r+1,j)^{(r,i)}}(z) \end{array}] = \begin{bmatrix} 1 & -\Delta^{(r+1,i)} \\ \delta^{(r,i)} \Delta^{(r+1,i)-1} z & (1 - \delta^{(r,i)})z \end{bmatrix} \begin{bmatrix} \tilde{\mu}^{(r,i)}(z) \\ \tilde{\nu}^{(r,j)^{(r,i)}}(z) \end{bmatrix}$$

The quotient $\delta^{(r,i)}/\Delta^{(r+1,i)}$ is understood to equal zero if $\Delta^{(r+1,i)}$ equals zero.

$$a^{(r+1,i)}_\mu = (1 - \delta^{(r,i)})a^{(r,i)}_\mu + \delta^{(r,i)} a^{(r,j)^{(r,i)}}_\nu + 1$$

$$a^{(r+1,j)^{(r,i)}}_\nu = (1 - \delta^{(r,i)})a^{(r,j)^{(r,i)}}_\nu + \delta^{(r,i)}a^{(r,i)}_\mu.$$
Theorem 3.3 The set of recursive equations in Algorithm 3.3 can be used to calculate a collection of $\tilde{\mu}^{(s,i)}(z)$, $0 \leq i < \gamma$ and $\tilde{\nu}^{(s,j)}(z)$, $0 \leq j < \gamma$ for any $s \geq r_0$. The corresponding functions $\mu^{(s,i)}$ and $\nu^{(s,j)}$ are valid at pole order $s$ for a set of erasures $\mathcal{R}$.

Proof. Lemma 3.5 including its proof remains valid after replacing every $\sigma$ and $\lambda$ with $\mu$ and $\nu$. The essence of the theorem relies only on this modified form of Lemma 3.5. The initialization is consistent with the definition of functions $\mu^{(r,i)}$ and $\nu^{(r,j)}$. □

An implementation of a BMA for error-erasure decoding does not differ from an implementation of error-only decoding given in figure 3.2. The only difference is the initialization of the algorithm. It follows immediately that the required complexity is the same as in the case of error-only decoding. In order to estimate the complexity for the determination of an error-erasure-locating function, we need an algorithm to find the required functions $\eta^{(i)}$.

Proposition 3.1 Let a divisor of erasures $R = P_{i_1} + P_{i_2} + \cdots + P_{i_p}$ be given. We can find functions $\eta^{(i)}$ with associated values $b^{(i)}$, $0 \leq i < \gamma$ as $\eta^{(i)} = \eta^{(\rho,i)}$ and $b^{(i)} = b^{(\rho,i)}$, where $\eta^{(\rho,i)}$ and $b^{(\rho,i)}$ are calculated by the following set of recursive equations.

\[
\begin{align*}
\eta^{(0,i)} &= \phi_{i}, \quad b^{(0,i)} = l_i \\
\beta^{(r+1)} &= \left(\min_{0 \leq i < \gamma} \{b^{(i)}(\eta^{(r,i)}(P_{i_{r+1}}) \neq 0)\}\right) \mod \gamma \\
\tau^{(r+1)} &= \frac{\eta^{(r,\beta^{(r+1)}}(P_{i_{r+1}})}{\eta^{(r,\beta^{(r+1)}}(P_{i_{r+1}})} \\
\eta^{(r+1,i)} &= \begin{cases} \\
\left(\phi_{\gamma} - \phi_{\gamma}(P_{i_{r+1}})\right)\eta^{(r,i)} & i = \beta^{(r+1)} \\
\eta^{(r,i)} - \eta^{(r,i)}(P_{i_{r+1}})\tau^{(r+1)} & \text{otherwise} \\
\end{cases} \\
\eta^{(r+1,i)} &= \begin{cases} \\
\eta^{(r,i)} + \gamma & i = \beta^{(r+1)} \\
\eta^{(r,i)} & \text{otherwise} \\
\end{cases}
\end{align*}
\]

Proof. Let $R_i$ equal $P_{i_1} + P_{i_2} + \cdots + P_{i_p}$ and $R_0$ equal 0. The proof works by induction. Obviously the claim is true for $R_0$. Suppose that it
is true for a divisor $R_i$. From the construction of functions $\eta^{(r+1,i)}$ follows that the functions $\eta^{(r+1,i)}$ have $P_{i_1}, \ldots, P_{i_r}$ as zeros. Moreover, the pole orders of all functions $\eta^{(r+1,i)}$ equal the pole orders of functions $\eta^{(r,i)}$ except for the case that $i$ equals $\beta^{(r+1)}$. In this case there does not exist a function of pole order $b^{(r,i)}$ which has the required zero in $P_{i_{r+1}}$. Finally $\beta^{(r+1)}$ is always defined as otherwise $L(mP_{\infty} - R_r)$ would be equal to $L(mP_{\infty} - R_{r+1})$ for all $m > 0$, which is not possible.

The equation set of Proposition 3.1 requires in every iteration the evaluation of $\gamma$ functions in a certain erasure point and the subsequent updating of $\gamma$ functions. The complexity for these operations is $O((\log_2(q))^2 \gamma \rho)$ and we have to perform $\rho$ iteration steps. Thus the total complexity for the algorithm of Proposition 3.1 is in the order $O((\log_2(q))^2 \gamma \rho^2)$. The operations can be performed in $\gamma$ parallel working devices which results in a time required to find functions $\eta^{(i)}$ in the order $O(\rho^2)$.

So far, the treatment of error-erasure decoding is a straight forward adoption of the theorems and proofs of the error-only case. In order to include a majority voting scheme, which is necessary to achieve the full error-erasure correction capability of the code, the content of Theorem 3.2 and the setup leading to this Theorem can be directly translated to error-erasure decoding, but the proof can not. This is because the proof of Theorem 3.2 relies on the maximal possible rank of matrix $S$. In the case of error-erasure decoding it is not the rank of matrix $S$ but the number of errors outside the set of erasure positions that is to be minimized. We reformulate Algorithm 3.2 and Theorem 3.2 for the case of error-erasure decoding and give a proof that does not rely on the rank of $S$. Although the essence of the proof relies on ideas from [19] and [20] it is formulated in the framework of our algorithm and the resulting theorem is slightly stronger than the corresponding theorems of [19] and [20]. The theorem contains Theorem 3.2 as a special case for a trivial divisor of erasures $R = 0$.

**Algorithm 3.4**

1. Calculate $\hat{\Delta}^{(m+1,i)}$ for all $i$ where $a_{(m,i)\not\in G}$ and let $D$ be the set of numbers $\hat{\Delta}^{(m+1,i)}$ for those $i$. 
2. Find the element \( \hat{S} \in D \) with greatest associated value

\[
\sum_{i: \Delta^{(m+1,i)} = \hat{S}} \max \{0, a_{\mu}^{(m,i)} - a_{\nu}^{(m,j^{(m,i)})}\}.
\]

3. Calculate \( S_{m+1,0} \) and \( \Delta^{(r,i)} \) as

\[
S_{m+1,0} = -\hat{S}, \quad \Delta^{(r,i)} = \hat{\Delta}^{(m+1,i)} - \hat{S}
\]

**Theorem 3.4** Let all entries \( S_{a,b} \) in \( S \) with \( a + b \leq m \) be given and let, for an erasure position set \( R \) of size \( p \), a collection of functions \( \mu^{(m,i)} \) and \( \nu^{(m,j)} \) with associated values \( a_{\mu}^{(m,i)}, a_{\nu}^{(m,j)} \) be given. For an error vector of weight \( t \) outside \( R \), Algorithm 3.4 can be used to determine \( S_{m+1,0} \) and \( \Delta^{(m+1,i)} \) provided the condition

\[
2t < d_{\Omega}^a(D, mP_{\infty}) - \rho + \sum_{i=0}^{\gamma-1} \max \{0, \frac{a_{\nu}^{(m,j^{(m,i)})} - a_{\mu}^{(m,i)}}{\gamma} \}
\]

holds.

**Proof.** Let \( E \) be the divisor of errors,

\[
E = \sum_{P_i : i \in \text{supp}(e) \setminus R} P_i, \quad \text{deg}(E) \leq t.
\]

Our primary goal is to determine the function of least pole order in the space of error-erasure-locating functions \( L(mP_{\infty} - E - R) \). By the Riemann-Roch theorem there exist \( \text{deg}(E) + \rho + g \) nonnegative numbers \( l \) such that \( L(lP_{\infty} - E - R) \) equals \( L((l-1)P_{\infty} - E - R) \). The set of these numbers is called \( E + R \) gaps at \( P_{\infty} \) or simply \( E + R \) gaps and we denote it by \( B \). \( B \) contains the set of \( R \) gaps, i.e. all \( l \) such that \( L(lP_{\infty} - R) = L((l-1)P_{\infty} - R) \). We have an \( R + E \) gap at pole order \( l \) if and only if there exist a function \( \nu \in L(mP_{\infty} - R) \) that gives a discrepancy at position \( (a < \infty, l) \) and there does not exist a function of lower pole order \( l' \) that gives a discrepancy at position \( (a, l') \). Conversely, given a function \( \nu \in L(mP_{\infty} - R) \), which gives a discrepancy at a position \( (a, l) \), we know that there must exist an \( R + E \) gap for some \( l' \leq l \), which is not an \( R \) gap. This observation holds independently for all different positions \( (a, l) \), where
we encountered a discrepancy. The idea of the theorem is to choose the unknown \( S_{m+1,0} \) in such a way that the number of \( R + E \) gaps is minimized as this minimizes the degree of \( E \). Assume we have run Algorithm 3.3 up to \( r = m \) and we thus know the set of numbers \( a_{(m,j)}^{(m_0,j)} \) and \( a_{(m,j)}^{(m_0,j)} \). Let \( \tau \) be the number of \( E + R \) gaps that are not \( R \) gaps and that we encountered so far. We can calculate \( \tau \) in two different ways as

\[
\tau = \sum_{j=0}^{\gamma-1} \frac{a_{(m,j)}^{(m_0,j)} - a_{(m,j)}^{(m_0,j-1)}}{\gamma}
\]

and

\[
\tau = \sum_{j=0}^{\gamma-1} \frac{b_{(m,j)}^{(m_0,j)} - b_{(m,j)}^{(m_0,j-1)}}{\gamma}.
\]

From the above equations and using the relation \( a_{(m,j)}^{(m_0,j)} + b_{(m,j)}^{(m_0,j)} = m + 1 \) we obtain that

\[
2\tau = \sum_{j=0}^{\gamma-1} \frac{m + 1 - a_{(m,j)}^{(m_0,j)} - b_{(m,j)}^{(m_0,j-1)} + a_{(m,j)}^{(m_0,j)} - a_{(m,j)}^{(m_0,j-1)}}{\gamma},
\]

\[
\sum_{i=0}^{\gamma-1} \frac{a_{(m,i)}^{(m_0,i)} - a_{(m,i)}^{(m_0,i-1)}}{\gamma} = m + 1 - \sum_{j=0}^{\gamma-1} \frac{b_{(m,j)}^{(m_0,j)} + a_{(m,j)}^{(m_0,j)}}{\gamma} - 2\tau,
\]

or

\[
\sum_{i=0}^{\gamma-1} \max \left\{ 0, \frac{a_{(m,i)}^{(m_0,i)} - a_{(m,i)}^{(m_0,i-1)}}{\gamma} \right\} = \quad (3.21)
\]

\[
m + 1 - \sum_{j=0}^{\gamma-1} \frac{b_{(m,j)}^{(m_0,j)} + a_{(m,j)}^{(m_0,j)}}{\gamma} - 2\tau + \sum_{i=0}^{\gamma-1} \max \left\{ 0, \frac{a_{(m,i)}^{(m_0,i)} - a_{(m,i)}^{(m_0,i-1)}}{\gamma} \right\}.
\]

From the two relations

\[
\sum_{j=0}^{\gamma-1} \frac{a_{(m_0,j)}^{(m_0,j)}}{\gamma} + \frac{j}{\gamma} = g - 1
\]

\[
\sum_{j=0}^{\gamma-1} \frac{b_{(m_0,j)}^{(m_0,j)}}{\gamma} - \frac{j}{\gamma} = g + \rho
\]
we deduce that the right hand side of equation (3.21) equals
\[
d_t^*(D, mP_\infty) - 2\tau - \rho + \sum_{i=0}^{\gamma-1} \max\{0, \frac{a_{\mu}^{(m,i)} - a_{\mu}^{(m,i)}}{\gamma}\}.
\]

The left hand side of (3.21) equals the number of positions \((a, b)\) with \(a+b = m+1\), which, depending on the choice of \(S_{m+1,0}\), may give rise to an \(E+R\) gap. On the other hand we know that the number of \(E+R\) gaps is at most \(t - \rho + g\) and we already have encountered \(\tau + \rho + g\) gaps. So when determining \(S_{m+1,0}\) not more than \(t - \tau\) new gaps can be encountered. In order to meet the assumption on the weight of the error vector outside \(R\) we have to choose \(S_{m+1,0}\) in such a way that less than half of the candidates for \(E+R\) gaps really will give gaps. This means we can by majority voting determine \(S_{m+1,0}\). In particular the assumption of the Theorem implies that the set of \(E+R\) gap candidates has at least one element. \(\square\)

We conclude from theorems 3.4 and 3.3 that an implementation of a BMA for error-erasure decoding algorithm with included majority voting does not differ from an implementation of error-only decoding.

**Remark 3.3** The term
\[
d_t^*(D, mP_\infty) - 2\tau - \rho + \sum_{i=0}^{\gamma-1} \max\{0, \frac{a_{\mu}^{(m,i)} - a_{\mu}^{(m,i)}}{\gamma}\}
\]
(3.22)
in Theorem 3.4 indicates that in certain cases the determination of \(S_{m+1,0}\) and \(\Delta^{(m+1,i)}\) is possible even when the weight of the error vector outside the erasure set exceeds the designed error correction capability \(d_t^*(D, mP_\infty) - \rho\). In the error-only case it is possible to bound expression (3.22) from below which leads to a bound on the minimum distance that in certain cases is larger than the designed minimum distance \(d_t^*(D, mP_\infty)\). This improved bound on the minimum distance is called the Feng-Rao distance. For a detailed treatment of this topic we refer to Pellikaan and Kirsch [21].
In this paper we have proposed an efficient algorithm to calculate error-locating functions for the class of one point AG codes $C_\Omega(D, mP_\infty)$. The algorithm is described by a recursive set of equations and works for both error only and error-erasure decoding. In the latter case a different initialization of the algorithm is necessary. The complexity of the algorithm is in the order of $O((\log_2(q))^2 \gamma m^2)$ binary operations, where the alphabet $\mathbb{F}_q$ of the code is a finite field of characteristic 2 and $\gamma$ is the size of the first non-gap in the space of functions associated with the code. The main result of this paper is an error-locating algorithm, which is convenient for implementation. Furthermore the structure of the algorithm naturally suggests a parallel implementation, which does not require more memory than a sequential implementation. The algorithm requires a running time that is essentially equal to the running time required by the BMA for an RS code. These two observations are important in order to make AG codes a competitive and reasonable alternative to RS codes in many applications.
Bibliography


Chapter 4

Fast generalized minimum distance decoding of algebraic-geometric and Reed-Solomon codes*

Abstract  Generalized Minimum Distance (GMD) decoding is a standard soft decoding method for block codes. We derive an efficient general GMD decoding scheme for linear block codes in the framework of error-correcting pairs. Special attention is paid to Reed-Solomon (RS) codes and one-point algebraic geometric (AG) codes. For RS codes of length $n$ and minimum Hamming distance $d$, the GMD decoding complexity turns out to be in the order $O(nd)$, where complexity is counted as the number of multiplications in the field of concern. For AG codes, the GMD decoding complexity is highly dependent on the curve in consideration. It is shown that we can find all relevant error-erasure-locating functions with complexity $O(o_3nd)$, where $o_3$ is the size of the first non-gap in the function space associated with the code. A full GMD decoding procedure for a one-point AG code can be performed with complexity $O(dn^2)$.

Index Terms — GMD decoding, Reed-Solomon codes, algebraic geometric codes, efficient algorithms

*to appear in IEEE Transactions on Information Theory
Chapter 4. Fast GMD decoding of AG and RS codes

I Introduction

The always returning task in applications of coding theory is to find the most likely word in a code, given a received word that is known to be a distorted code word. In many practical situations it is possible to utilize additional information about the reliability of the positions in the received word. Algorithms that make use of such side information have important applications in concatenated schemes, where a so called outer code over a large alphabet is concatenated with an inner code that is often a binary code or a code in Euclidean space. Here the decoding of the inner code provides the reliability information as the weight of the error vector that corrupted the corresponding inner code word. Generalized Minimum Distance (GMD) decoding of block codes, introduced by Forney in [1], gives a possibility to utilize reliability information on positions of the received word. Efficient schemes to perform such a GMD decoding procedure were proposed by Sorger [2], Araki, Takada and Morii [3], Berlekamp [4], Taipale and Seo [5], and Kötter [6][7]. However, the approaches of [2][3][4] and [5] are restricted to Reed-Solomon (RS) codes. The required complexity, counted as the number of multiplications in the field of concern, of these approaches is in the order $O(n^2 d^5)$, where $d$ denotes the length of the RS-code and $d$ the minimum Hamming distance. This is to be compared with earlier schemes requiring a computational complexity in the order $O(n^2)$.

The approach proposed here is not restricted to RS codes but provides a scheme that is in principal applicable to any linear code. The basic idea relies on the concept of error-correcting pairs which was introduced by Pellikan in [8]. The complexity of this approach is highly dependent on the structure of the code. In the case of RS codes the complexity lies in the order $O(n^4 d)$, For AG codes the determination of all error-erasure-locating functions requires complexity $O(n_1 nd)$, where $n_1$ is the size of the first non-gap in the function space associated with the code. The goal of our paper is to first give the general description of this scheme and then to investigate in detail how this scheme applies to two prominent classes of linear codes, namely RS codes and AG codes. In particular, we focus on efficiently combining the general idea of fast GMD decoding with Berlekamp-Massey type algorithms that are used in the decoding of RS and AG codes.

The outline of the paper is as follows. In section II, III and IV we give the
II. An error-erasure-locating algorithm

basic ideas of an efficient GMD decoding scheme. Sections V and VI treat RS and AG codes respectively.

II An error-erasure-locating algorithm

Let \( \mathbb{F}_q \) denote the finite field with \( q \) elements and \( \mathbb{F}_q^n \) the vector space of all \( n \)-tuples \( \mathbf{u} = \{u_0, u_1, \ldots, u_{n-1}\} \) over \( \mathbb{F}_q \). As a convention in this paper all vector and matrix indices are counted starting from zero. A linear code \( C \) over \( \mathbb{F}_q \) is a linear subspace of \( \mathbb{F}_q^n \) with dimension \( k(C) \) and minimum Hamming distance \( d(C) \) between any two different words in \( C \). With respect to the standard inner product

\[
(\mathbf{u}, \mathbf{v}) = \sum_{i=0}^{n-1} u_i v_i \quad \mathbf{u}, \mathbf{v} \in \mathbb{F}_q^n,
\]

we have the dual linear space to \( C \) which is denoted by \( C^\perp \). A generator matrix of \( C \) is denoted by \( G_C \) and a parity check matrix by \( H_C \).

For two vectors \( \mathbf{u} = (u_0, u_1, \ldots, u_{n-1}) \) and \( \mathbf{v} = (v_0, v_1, \ldots, v_{n-1}) \), we define a component-wise product \( \mathbf{u} * \mathbf{v} = (u_0v_0, u_1v_1, \ldots, u_{n-1}v_{n-1}) \). For two subspaces \( U, V \subset \mathbb{F}_q^n \) let \( U * V \) denote the set of vectors \( \{\mathbf{u} * \mathbf{v} : \mathbf{u} \in U, \mathbf{v} \in V\} \).

In the standard decoding situation we are given a vector \( \mathbf{y} = \mathbf{c} + \mathbf{e} \) and we want to find the codeword \( \mathbf{c} \in C \) such that the Hamming weight of the error vector \( \mathbf{e} \) is minimized. Following the main idea of algebraic decoding, we consider a two step algorithm which first tries to find the nonzero positions in \( \mathbf{e} \) and in a second step determines the error values in these positions. The following definition, which is equivalent to Definition 1 in [9], is central for the error-location problem.

**Definition 4.1** (\( \ell \)-error-locating pair) Let \( U, V \) and \( C \) be linear codes of length \( n \) over the field \( \mathbb{F}_q \). We call \( (U, V) \) a \( \ell \)-error-locating pair for \( C \) if the following conditions hold

\[
\begin{align*}
C * U &\subseteq V, \\
k(U) &> \ell, \\
d(V) &> \ell.
\end{align*}
\]
Chapter 4. Fast GMD decoding of AG and RS codes

Our first goal will be to find a vector \( u \in U \) such that \( u \ast e = 0 \). Clearly \( u \) then indicates possible error positions with zeros. Let \( \text{diag}(e) \) denote the \( n \times n \) matrix which has the elements of \( e \) on its main diagonal and which is zero everywhere else. The starting point of our paper is the following theorem from [10], compare also [8][9].

**Theorem 4.1** Let \( (U, V) \) be a \( t \)-error-locating pair for the code \( C \). Let \( y = c + e \) be a word in \( \mathbb{F}_q^n \) with \( c \in C \) and \( e \) a vector of weight at most \( t \). Any solution \( \sigma \) to the system of linear equations defined by

\[
H_V \cdot \text{diag}(y) \cdot G_U^T \sigma^T = 0. \tag{4.4}
\]

gives a vector \( u = \sigma G_U \) which satisfies

\[
u \ast e = 0.
\]

Moreover, there exists a nontrivial such \( \sigma \).

**Proof.** With \( y = c + e \) we can rewrite equation (4.4) as

\[
H_V \cdot \text{diag}(c) \cdot G_U^T \sigma^T + H_V \cdot \text{diag}(e) \cdot G_U^T \sigma^T = 0.
\]

By condition (4.1) the first term on the left hand side of this equation is equal to the zero vector for all \( \sigma \). Thus the solution space to (4.4) depends only on the error vector \( e \). For any \( \sigma \) that solves (4.4) we have a vector \( u = (\sigma G_U) \) such that

\[
u \ast e \in V.
\]

Clearly the Hamming weight of \( u \ast e \) can not exceed \( t \) which by condition (4.3) implies

\[
u \ast e = 0.
\]

To show the existence of a nontrivial solution \( \sigma \), we observe that any nonzero vector \( u \), that is zero in all error positions, gives rise to a nontrivial \( \sigma \) which solves (4.4). To find such an \( u \) we have to impose at most \( t \) linearly independent conditions on \( U \). Thus the existence of a nontrivial vector \( u \) with the error-locating property is guaranteed by condition (4.2). \( \square \)
In many practical applications of codes we have access to additional information concerning the reliability of the positions in a received vector. We can utilize this information by presuming that errors have occurred in the \( \rho \) least reliable positions. We call these positions erasure positions. Let a vector \( \mathbf{y} \) and a set \( \mathcal{R} \), containing precisely the erasure positions, be given. The goal of error-erasure decoding is to find the codeword \( \mathbf{c} = \mathbf{y} - \mathbf{e} \) that minimizes the Hamming weight of the error vector \( \mathbf{e} \) outside the set \( \mathcal{R} \). The idea of error-erasure decoding relies on the well known fact that we can correct any combination of \( \rho \) erasures and \( t \) errors as long as the condition \( 2t + \rho < d(C) \) is satisfied.

Here we assume that estimates exist for all erasure positions, otherwise we may set \( y_i = 0 \) for all \( i \in \mathcal{R} \). Again our goal is to find a vector \( \mathbf{u} \) such that \( \mathbf{u} \ast \mathbf{e} = 0 \), but in addition we now require also that the linear conditions \( u_i = 0 \) hold for all \( i \in \mathcal{R} \).

**Definition 4.2 ((t, \rho)-error-erasure-locating pair)** Let \( U, V \) and \( C \) be linear codes of length \( n \) over the field \( \mathbb{F}_q \). We call \( (U, V) \) a \((t, \rho)\)-error-erasure-locating pair for \( C \) if the following conditions hold

\[
C \ast U \subseteq V, \quad k(U) > t + \rho, \quad d(V) > t. \tag{4.5, 4.6, 4.7}
\]

**Corollary 4.1 (to Theorem 4.1)** Let \( (U, V) \) be a \((t, \rho)\)-error-erasure-locating pair for the code \( C \). Let \( \mathbf{y} = \mathbf{c} + \mathbf{e} \) be a given word in \( \mathbb{F}_q^n \) with \( \mathbf{c} \in C \) and let \( \mathcal{R} \) be a set of erasure positions with \( |\mathcal{R}| \leq \rho \). Let the vector \( \mathbf{e} \) have weight at most \( t \) outside the erasure positions. Then any solution \( \mathbf{\sigma} \) that solves the system of linear equations defined by

\[
H_V \cdot \text{diag}(\mathbf{y}) \cdot G_U^T \mathbf{\sigma}^T = 0, \quad \tag{4.8}
\]

and, with \( \mathbf{u} = \mathbf{\sigma}G_U \),

\[
u_i = 0 \text{ for all } i \in \mathcal{R}, \quad \tag{4.9}
\]

gives a vector \( \mathbf{u} \) which satisfies

\[
\mathbf{u} \ast \mathbf{e} = 0.
\]
Proof. To satisfy condition (4.9) let $W$ be the the subspace of $U$ that consists of all vectors in $U$ which have zero components in $\mathcal{R}$. Clearly $k(W)$ is greater than $t + \rho - |\mathcal{R}|$ and therefore greater than $t$. $(W, V)$ is a $t$-error-locating pair for $C$. The theorem gives the corollary.

Remark 4.1 The existence of a $(t, \rho)$-error-erasure-locating pair $(U, V)$ for a code $C$ does in general not imply the existence of a $(t-1, \rho+2)$-error-erasure-locating pair for $C$. However, we can consider the code $C'$ that is obtained by puncturing $C$ in one erasure position. Let $(U', V')$ be the error-erasure-locating pair that is obtained from the pair $(U, V)$ by puncturing the same position as in $C'$. Provided that $d(V)$ is larger than 1, $(U', V')$ is a $(t-1, \rho+1)$-error-erasure-locating pair for $C'$, because we have $d(V') \geq d(V) - 1 > t - 1$ and $k(U') = k(U) > (t - 1) + (\rho + 1)$ and condition (4.5) clearly is satisfied. In this sense a $(t, \rho)$-error-erasure-locating pair $(U, V)$ for a code $C$ provides an error-erasure location scheme for $t-1$ errors and $\rho+2$ erasures, where one of the erased positions corresponds to the punctured position.

From Theorem 4.1 and the Corollary 4.1 we find the set of possible error positions which we denote by $E$. The second step is to find the error values. The error vector satisfies the two sets of linear conditions

$H_C(y^T - e^T) = 0$

$e \ast u = 0.$

The error values can be found solving a system of linear equations, if the set of possible error positions is not too large. Not too large means that we can always find the error values if

$|E| < d(C),$ \hspace{1cm} (4.10)

because in that case the system of linear equations has maximal possible rank $|E|$.

We call an error-erasure-locating pair of codes which allows determination of the error values an error-erasure-correcting-pair. The size of $E$ does not exceed $n - d(U)$. This fact, together with condition (4.10), justifies the following definition, cf. also [8][9].
III. Generalized-Minimum Distance decoding

Definition 4.3 ((t, \rho)-error-erasure-correcting pair) Let \( (U, V) \) be a \((t, \rho)\)-error-erasure-locating pair for the code \( C \). We call \( (U, V) \) a \( t \)-error-correcting pair for the code \( C \) if in addition to the conditions (4.5),(4.6) and (4.7) the following inequality is satisfied

\[
d(C) + d(U) > n,
\]

where \( n \) denotes the code length of \( C \).

Remark 4.2 The definition of an error-erasure-correcting pair is rather strong but fully sufficient for the codes considered in this paper. For a more thorough treatment of the subject the reader is referred to [9], where weaker conditions for error correction are formulated.

Remark 4.3 The problem of error correction of linear codes comes actually down to finding an \( t \)-error-correcting pair that maximizes \( t \) for a given code \( C \). Having found such a pair the decoding can be performed by solving linear equations and matrix multiplications. The matrix dimensions of all involved matrices are less than or equal to \( n \). Thus the complexity of such a procedure is of order \( O(n^3) \). For many prominent classes of codes, such as BCH-codes [11, p.248] and AG codes [17], there are natural choices for the pair \( (U, V) \).

III Generalized-Minimum Distance decoding

In addition to the vector \( y \), let some kind of reliability information on the positions be given. GMD decoding, proposed by Forney in [12], gives the possibility to utilize the reliability information in an algebraic decoding algorithm. For a description of this algorithm we assume that the reliability information induces a complete ordering on positions. In Forney’s GMD procedure, \( \left\lfloor \frac{q}{2} \right\rfloor \) decoding attempts are performed, where an increasing number of least reliable positions are declared erasure. More precisely, the \( i \)-th decoding attempt employs an error-erasure-locating algorithm, where the \( 2i \) least reliable positions in \( y \) are treated as erasure. These decoding attempts result in a list of tentative code words, and we choose from this list the code
word that is most reliable. A number of different criteria to guarantee successful GMD decoding is investigated in [13]. For a performance analysis of GMD decoding we refer to [13], because the main interest of this paper concerns the efficiency of algorithms. We only note that the criteria, formulated in [13], take into account whether an error occurred or not, so knowledge of the error positions is enough to choose the most likely candidate from the list of tentative codewords.

Let a sequence \( \{ R_i \}_{i=0}^I \) of erasure position sets with \( |R_i| = \rho_i \) and \( R_i \subseteq R_{i+1} \) be given. From the previous section we see that all we need, in order to make a GMD procedure work, is a corresponding set of \((t_i, \rho_i)\)-error-erasure-correcting pairs. From Remark 4.3 we see that such an ad hoc scheme requires \( O(\alpha n^3) \) operations due to the fact that we employ at most \( \lceil \frac{I}{2} \rceil \) independent decoding attempts, each requiring a complexity \( O(n^3) \). One of the results of this paper is that we can do better by introducing a relation between the error-erasure-locating pairs for different numbers of erasures. Explicitly, if a sequence of error-erasure correcting pairs, satisfying the desired relation, exists we can find the set of error-erasure-locating vectors for increasing sets of erasure positions with essentially the same complexity as required by one error-erasure-locating algorithm.

Let \( \{ U_i, V_i \}_{i=0}^I \) be a collection of \((t_i, \rho_i)\)-error-erasure-locating pairs. We assume \( U_i \subseteq U_{i+1} \) and \( V_i \subseteq V_{i+1} \). In fact, given a code \( C \), it suffices to claim this for the spaces \( U_i \) because we then define the \( V_i \) as the spaces spanned by all vectors in \( C \cdot U_i \). Without loss of generality we assume that the generator matrix \( G_{V_{i+1}} \) contains \( G_{U_i} \) in the first \( k(U_i) \) rows. In the same way the parity check matrix \( H_{V_i} \) is contained in the first \( n - k(V_i) \) rows of \( H_{V_{i+1}} \). The corresponding set of key equations (4.8) is

\[
\Gamma_i \sigma^T = 0, \quad \sigma \in \mathbb{F}_q^{k(U_i)} \quad \text{where} \quad \Gamma_i = H_{V_i} \text{diag}(y) G_{U_i}^T.
\]  

Each key equation has a space of solutions which we denote by \( \Sigma_i \)

\[
\Sigma_i = \{ \sigma : \Gamma_i \sigma^T = 0 \}.
\]

The idea is now to obtain in one step all spaces \( \Sigma_i \) together with the corresponding spaces

\[
W_i = \{ u : u = \sigma G_{U_i}, \sigma \in \Sigma_i \}.
\]
In a second step we will find the subspaces of $W_i$ which are zero in the desired erasure positions so as to satisfy condition (4.9) of Corollary 4.1.

We define a matrix $\Gamma$ by

$$\Gamma = H_{V_0} \text{diag}(\gamma) G_{U_i}^T$$

(4.14)

and denote the sub-matrix of $\Gamma$ consisting of the elements in the first $a + 1$ rows and first $b + 1$ columns as $\Gamma^{(a,b)}$. This definition of $\Gamma^{(a,b)}$ is chosen in order to make the numbers $a, b$ consistent with the row and column indices of $\Gamma$. The following relation holds by the nested property of codes $U_i$ and $V_i$ and the choice of matrices $G_{U_i}$ and $H_{V_0}$

$$\Gamma^{(n-k(U_i)-1,k(U_i)-1)} = \Gamma_i.$$  

(4.15)

Thus the necessary information for all decoding attempts is contained in $\Gamma$. The efficiency of the proposed procedure originates from the fact that we can find the solution spaces $\Sigma_i$ of all key equations (4.12) by dealing with only one matrix $\Gamma$. In the next section we will describe an efficient algorithm to accomplish this.

**Lemma 4.1** The vector spaces $W_i$ satisfy

$$W_i \subseteq W_{i+1}, \text{ for all } i < l$$

**Proof.** Consider a vector $\sigma$ that solves $\Gamma_i \sigma^T = 0$. We see from equation (4.15) that the same vector $\sigma$ extended by $k(U_{i+1}) - k(U_i)$ zeros also solves $\Gamma_{i+1} \sigma^T = 0$. Thus we have for all $i < l$ that $\Sigma_i$ is contained in $\Sigma_{i+1}$, where vectors in $\Sigma_i$ are extended by a suitable number of zeros. The lemma follows now from equation (4.13). \qed

**Remark 4.4** From Lemma 4.1 we obtain a generator matrix $G_{W_i}$ for $W_i$ which contains generator matrices for $W_i$, $i \leq l$ in the leading rows. By simple row operations, we obtain $G_{W_i}$ in such a form that for all $i$ there exists a row vector that belongs to $W_i$ and has zeros in positions $R_i$. This vector is then the proper error-erasure-locating vector from the $(t_i, \rho_i)$-error-erasure-locating pair. Obtaining $G_{W_i}$ in this form requires $O(d^2 n)$ operations due to the fact that we have to perform at most $O(d^2)$ row operations.
In the general case, the described approach results in an efficient procedure for obtaining the error-erasure-locating vectors in an GMD-decoding scheme based on algebraic decoding. Nevertheless there are inherent restrictions. It is by no means clear whether, for a certain code \( C \), a sequence of \( (t_i, \rho_i) \)-error-erasure-correcting pairs with the desired nested properties exists such that \( 2t_i + \rho_i \) is reasonably close to \( d(C) \) for each pair in the sequence. Assuming we have found such a sequence we are confronted with the next problem, namely that the sets of possible error positions in the general case might contain more positions than the weight of the corresponding error pattern. Moreover, we have at this stage of the algorithm no indication whether the erasure positions really were erroneous. On the other hand, checking the criteria for GMD-decoding investigated in [13] requires perfect knowledge of error positions in order to decide on the most likely codeword from the list of tentative codewords. In this situation there are two possible strategies. Applying the first strategy we could simply accept the uncertainty in error positions and make a decision based on the zero sets found. This is feasible, if we know that not too many non error positions are among the possible error positions. The other strategy implies that we complete the decoding procedures by finding the error vectors that correspond to the sets of possible error positions. In this setup we obtain a list of tentative error vectors and we can pick the most likely one from this list. If we choose the second alternative we can be sure to realize the decoding capability of the GMD procedure, but we also increase complexity. In either case the finding of the error-erasure-locating vectors has to be performed, and the described procedure helps in reducing computational complexity. We will return to these problems in Sections V and VI where we restrict ourselves to RS and AG codes.

IV The Fundamental Iterative Algorithm

Equation (4.12) in the previous section provides us with a set of key equations for different numbers of set erasures. These equations can be solved by any standard method from linear algebra. An especially convenient form of Gaussian elimination is the Fundamental Iterative Algorithm (FIA) which was formulated by Feng and Tzeng in [14]. We recall here a slightly modified form of this algorithm that is suited for our situation. For a given \( M \times N \)
matrix $A$,

\[ A = \| A_{i,j} \|, \quad i = 0, 1, \ldots, M - 1, \quad j = 0, 1, \ldots, N - 1, \]

the FIA gives a minimal set of linearly dependent leading columns. It basically solves an arbitrary homogeneous system of linear equations and contains the Berlekamp-Massey algorithm (BMA) as a special case. We recall a brief description of the basic idea. Details and proofs can be found in [14]. Whenever it is necessary for reasons of dimension we extend a vector with a suitable number of zeros. Again let the matrix $A^{[a,b]}$ be the sub-matrix of $A$ consisting of the elements in the first $a + 1$ rows in the first $b + 1$ columns of $A$. For a fixed $b$ we consider column vectors $\sigma$ with non-zero coordinate at the $b$-th position that solve the equation

\[ A^{[a,b]} \sigma = 0. \]

Let $a = a(b)$ be maximal such that a solution exists and let $\sigma = \sigma^{(b)}$ be such a solution. For these $a$ and $\sigma$ let $\Delta^{(b)}$ be defined as

\[ \Delta^{(b)} = \sum_{k=0}^{b} A_{a+1,k} \sigma_k, \]

or as $\Delta^{(b)} = 0$ when $A\sigma^{(b)} = 0$. If $\Delta^{(b)} \neq 0$ we say that we have a discrepancy at position $(a + 1, b)$.

For a given sequence \{ $(\sigma^{(b)}, \Delta^{(b)}, a(b))$ \}$_{b<i}$ the idea of the FIA is now to calculate $\sigma^{(i)}$ with help of the $\sigma^{(b)}$, $b < i$. Starting with any vector $\sigma$ of length $i + 1$ and $\sigma_i$ unequal to zero, this is achieved by subtracting suitable scalar multiples of the known $\sigma^{(b)}$ from $\sigma$, thereby obtaining a new $\sigma$. More precisely, whenever $\sigma$ solves

\[ A^{[j,i]} \sigma = 0, \]

with

\[ \delta = \sum_{k=0}^{i} A_{j+1,k} \sigma_k \neq 0 \]

and there already exists a discrepancy at position $(j + 1, b)$, $b < i$ we use the triple $(\sigma^{(b)}, \Delta^{(b)}, j)$ to construct a new $\sigma$ by the rule

\[ \sigma \leftarrow \sigma - \frac{\delta}{\Delta^{(b)}} \sigma^{(b)}. \]
Chapter 4. Fast GMD decoding of AG and RS codes

This new $\sigma$ satisfies

$$A^{(j+1,i)} \sigma = 0.$$ 

Finally we will obtain the triple $(\sigma^{(i)}, \Delta^{(i)}, a(i))$. Repeating this process we eventually find the desired $\sigma$ that solves $A \sigma = 0$. We emphasize the fact that we can freely choose the starting vector to investigate the $i$-th column with the only restriction that the $i$-th component is unequal to zero. This is the key to an efficient algorithm of Berlekamp-Massey type. If we, due to the structure of the matrix, can find a good starting vector, that saves some operations, this will lead to faster algorithms. This is the basic observation that will be utilized for RS and AG-codes in section $V$ and $VI$.

In the previous section we considered the task of solving simultaneously systems of linear equations that were defined in equation (4.15) by sub-matrices of $\tilde{\Gamma}$. The FIA produces in one run the solution spaces for all $\Gamma_j$. The only modification to the original algorithm is that now we are not only interested in the first combination of linearly dependent columns but in all such relations. Having found the first solution that solves a matrix $\Gamma_j$ for a minimal $j$ we simply consider the next column in $\tilde{\Gamma}$. After a complete run of the FIA we thus have for all $i, 0 \leq i < N$ triples $(\sigma^{(i)}, \Delta^{(i)}, a(i))$ where $a(i)$ is given either as the maximal $a$ such that $\tilde{\Gamma}^{[a,j]}$ has a solution or as $n - k(V_j) - 1$ if $k(U_{j-1}) \leq i < k(U_j)$ and $\Gamma_j \sigma^{(i)} = 0$.

**Proposition 4.1** Let a matrix $\tilde{\Gamma}$ be given with sub-matrices

$$\Gamma_j = \tilde{\Gamma}^{(n-k(V_j)-1,k(U_j)-1)},$$

describing systems of linear equations. The solution spaces $\Sigma_j$ for all $\Gamma_j$ are found by the FIA in one run where a set of vectors generating $\Sigma_j$ is given by

$$\Sigma_j = \langle \sigma^{(i)} : a(i) \geq n - k(V_j) - 1 \rangle.$$

**Proof.** Follows from the construction of the triples $(\sigma^{(i)}, \Delta^{(i)}, a(i))$ and the proof of Lemma 4.1.

We summarize our results obtained so far in the following theorem.
Theorem 4.2 Let a linear code $C$ together with a sequence of $(t_i, \rho_i)$-error-erasure-locating pairs $\{ (U_i, V_i) \}_{i=0}^t$ be given. For a vector $y = c + e$ with $c \in C$ let a sequence $\{ R_i \}_{i=0}^t$ of erasure position sets with $|R_i| = \rho_i$ and $R_i \subseteq R_{i+1}$ be given. If the pairs $(U_i, V_i)$ satisfy

$$U_i \subseteq U_{i+1}$$

and

$$V_i \subseteq V_{i+1}$$

then we can find the corresponding error-erasure-locating vectors from all pairs $(U_i, V_i)$ with complexity $O(n^3)$.

Proof. Using equation (4.14) we can calculate matrix $\tilde{\Gamma}$ from the vector $y$ with complexity $O(n^3)$. Applying the FIA to $\tilde{\Gamma}$ requires complexity $O(n^3)$. The theorem follows then from Proposition 4.1 and Remark 4.4.

V Reed-Solomon codes

We recall one of the standard descriptions of RS codes over $\mathbb{F}_q$. Let $\alpha$ be a primitive element in $\mathbb{F}_q$ and $\mathbb{F}_q[x]$ be the ring of polynomials in $x$ with coefficients from $\mathbb{F}_q$. The RS code $C$ of length $n = q - 1$, dimension $k(C)$ and minimum distance $d(C) = n - k + 1$ is defined as

$$C = \{ (f(\alpha), f(\alpha^1), \ldots, f(\alpha^{q-2})) : \deg(f) < k, f \in \mathbb{F}_q[x] \}.$$

The $i$th position in a codeword of $C$ is naturally associated with $\alpha^i \in \mathbb{F}_q$.

We assume that $d(C)$ is odd and define $\tau$ as

$$\tau = \frac{d(C) - 1}{2}.$$

A sequence of $(t_i, \rho_i)$-error-erasure-locating pairs is described in the following proposition.
Proposition 4.2 Let $C$ be the RS code of length $n$, dimension $k$ and distance $2\tau + 1$. Let $U_i$ and $V_i$ be the RS codes of length $n$ with dimensions $\tau + 1 + i$ and $k + \tau + i$, respectively, for $i = 0, 1, \ldots, \tau$. Then the pairs $(U_i, V_i)$ give a sequence of $(t_i, \rho_i)$-error-erasure-correcting pairs with $U_i \subset U_{i+1}$ and $V_i \subset V_{i+1}$ such that

$$t_i = \tau - i$$

(4.16)

$$\rho_i = 2i.$$ 

(4.17)

Proof. For every pair $(U_i, V_i)$ we check conditions (4.5)-(4.7) for error-erasure correction. $C$ and $U_i$ consist of the evaluation vectors of polynomials of highest degree $k-1$ and $\tau + i$ respectively. The component-wise product of vectors in condition (4.5) corresponds to the usual product of polynomials before evaluating them in the $q - 1$ points $\alpha^j, 0 \leq j < q - 1$. Thus $V_i$ is obtained by the evaluation of polynomials of degree at most $(k-1) + (\tau+i)$ and it follows that $V_i$ is the RS code of dimension $k + \tau + i$. This shows (4.5). For a pair $(U_i, V_i)$ we have $k(U_i) = \tau + i + 1$ and $d(V_i) = \tau - i + 1$, and we see that conditions (4.6) and (4.7) are satisfied with $t_i$ equal to $\tau - i$ and $\rho_i$ equal to $2i$. The nested property of codes $U_i$ and $V_i$ is obvious from the construction of pairs $(U_i, V_i)$. Finally, in order to check condition (4.11), we note that $d(U_i) + d(C)$ equals $n + \tau - i + 1 > n$. 

With the above proposition we directly can apply the results of the previous sections to RS codes. In particular, the requirements in order to apply Theorem 4.2 are satisfied. However, we can do better by combining a Berlekamp-Massey type algorithm with the idea of a fast GMD decoding scheme. The way this is done is described in the rest of this section.

For an RS code with distance $2\tau + 1$ let $S_i$ be the usual syndrome component derived from a vector $y$

$$S_i = \sum_{j=0}^{n-1} y_j \alpha^{ji} \quad \text{for } i = 1, 2, \ldots, 2\tau.$$

Using generator and parity check matrices for RS codes defined as

$$\{G_C\}_{i,j} = \alpha^{ij}, \quad i = 0, \ldots, k-1, \quad j = 0, \ldots, n-1$$

$$\{H_C\}_{i,j} = \alpha^{(i+1)j}, \quad i = 0, \ldots, n-k-1, \quad j = 0, \ldots, n-1,$$
we can work out expression (4.14) for $\tilde{\Gamma}$. $\tilde{\Gamma}$ has the usual Hankel structure i.e.

$$\tilde{\Gamma}_{i,j} = S_{i+j+1} \text{ for } 0 \leq i + j < 2\tau.$$ 

We see that the interesting part of $\tilde{\Gamma}$ is as a triangular matrix with $2\tau$ columns and rows. Comparing this definition of $\tilde{\Gamma}$ to equation (4.14) we have entries in $\tilde{\Gamma}$ even for row-indices exceeding $n - k(V_0) - 1$. That does not change the solution spaces $\Sigma_i$ but is preferable for the description of a decoding algorithm for RS codes.

When dealing with Hankel matrices, it is convenient to identify vectors $\sigma$ with polynomials $\sigma(x)$ such that $\sigma$ is interpreted as the coefficient vector of a polynomial $\sigma(x)$. We say that a polynomial $\sigma(x)$ solves a linear equation if the coefficient vector $\sigma$ does. We point out that our notation slightly differs from the one that is used in most books on the subject. Instead of error-locators of the form $\prod (1 - \alpha^i x)$ [11, p.168] we use error-locating polynomials of the form $\prod (x - \alpha^i)$, which appears to be more natural in connection with error-locating-pairs. When applied to Hankel matrices in a suitable form, the FIA leads to the well known BMA described in e.g. [11]. There the BMA is formulated as an iterative procedure for the construction of an autoregressive filter. The output of the BMA are two polynomials $\Lambda(x)$ and $B(x)$ which describe recursive relations on the syndromes $S_1, S_2, \ldots, S_{2\tau}$. $\Lambda(x)$ is the feedback-connection polynomial of least degree that generates the sequence of syndromes, whereas $B(x)$ describes the feedback-connection polynomial with least degree, less than the degree of $\Lambda$, that generated the largest initial sequence of the syndromes. A detailed description of the BMA can be found in [11]. According to our notation, let $\sigma(x)$ and $\lambda(x)$ be the reciprocal polynomials to $\Lambda(x)$ and $B(x)$, respectively. The recurrence properties of $\Lambda(x)$ and $B(x)$ are translated to the following properties of $\sigma(x)$ and $\lambda(x)$.

- $\sigma(x)$ is the polynomial of minimal degree $l$ such that the system of equations
  
  $$\tilde{\Gamma}^{(2\tau-l-1,l)} \sigma^T = 0$$

  is satisfied.
Chapter 4. Fast GMD decoding of AG and RS codes

- The polynomial $\lambda(x)$ of degree $b < \deg(\sigma)$ solves for a maximal $a$ the system of equations

$$\tilde{\Gamma}^{[a,b]} \lambda^T = 0.$$ 

Two additional properties of the polynomials $\sigma(x)$ and $\lambda(x)$ are formulated in the following two lemmata as they are not explicitly proved in [11].

**Lemma 4.2** Let $a$ be maximal such that the polynomial $\lambda(x)$ solves the system of equations

$$\tilde{\Gamma}^{[a,b]} \lambda^T = 0.$$ 

For this $a$ the relation

$$a = \deg(\sigma) - 2$$

holds.

**Proof.** The lemma is essentially a reformulation of Lemma 7.4.3 from [11, p.181], which states that the length of the shift register described by $\Lambda(x)$ equals $r - L_B$, where $L_B$ is the length of the shift register described by $B(x)$ and $B(x)$ generated the sequence of syndromes $S_1, S_2, \ldots, S_{r-1}$ but not $S_1, S_2, \ldots, S_r$. In our notation this is expressed in the two equations

$$\deg(\sigma) = r - \deg(\lambda) \text{ and } a + 1 + \deg(\lambda) = r - 1,$$

which combined prove the lemma. \hfill $\Box$

**Lemma 4.3** The polynomials $\sigma(x)$ and $\lambda(x)$ do not have a common factor.

**Proof.** The lemma follows from the iterative construction of $\sigma(x)$. There exists a polynomial $\lambda^{-}(x)$, with degree less than the degree of $\lambda$, such that

$$\sigma(x) = A(x)\lambda(x) + \delta \lambda^{-}(x)$$

holds for some polynomial $A(x)$ and some nonzero constant $\delta$. A common factor of $\sigma(x)$ and $\lambda(x)$ would also be a factor of $\lambda^{-}(x)$. We can in a similar way trace back all operations of the BMA and find that all intermediate polynomials would contain this common factor. This contradicts the fact that the BMA is started with the constant polynomial. \hfill $\Box$
Let a syndrome polynomial \( S(x) \) be defined as
\[
S(x) = \sum_{l=0}^{2\tau-1} S_{2\tau-l}x^l.
\]

We define two polynomials \( \nu(x) \) and \( \mu(x) \) as
\[
\nu(x) = S(x)\sigma(x) \mod x^{2\tau} \\
\mu(x) = S(x)\lambda(x) \mod x^{2\tau}.
\]

If we have a polynomial \( \sigma(x) \), that is zero at all error positions, and the corresponding associated polynomial \( \nu(x) \), the error values in these positions are given by a reformulation of Forneys formula [11, p.184]
\[
e_i = \begin{cases} 
-\frac{\alpha^{-i(l(2\tau+1)\mu(\sigma^l))}}{\sigma'(\alpha^i)} & \text{for all } i \text{ s.th. } \sigma(\alpha^i) = 0 \\
0 & \text{otherwise},
\end{cases} \quad (4.18)
\]
where \( \sigma'(x) \) denotes the formal derivative of \( \sigma(x) \). The derivation of this formula is essentially the same as the proof of Theorem 7.5.2. in [11, p.184] and therefore omitted.

The results of section III would now suggest to first determine a proper basis of the spaces \( \Sigma_i \) and then, corresponding to accumulated erasure positions, in a second step to impose successively linear conditions on the spaces \( \Sigma_i \). For RS codes it is simpler to combine the two steps. Let \( \mathcal{R} \) be a set of erasure positions in a word \( y \). We need the notation of polynomials \( \sigma^{(\mathcal{R})}(x) \), \( \lambda^{(\mathcal{R})}(x) \) and associated polynomials \( \nu^{(\mathcal{R})}(x) \) and \( \mu^{(\mathcal{R})}(x) \), that play a similar role for error-erasure decoding as \( \sigma(x) \) and \( \lambda(x) \) and their associated polynomials for error-only decoding. The polynomials should satisfy the following properties.

- The greatest common divisor of \( \sigma^{(\mathcal{R})}(x) \) and \( \lambda^{(\mathcal{R})}(x) \) equals
\[
\prod_{i,j \in \mathcal{R}} (x - \alpha^i).
\]
\[
\tilde{\Gamma}^{(2\tau - 1 - l, l)} \sigma^T = 0
\]

is satisfied.

- The polynomial \( \lambda^{(R)}(x) \) satisfies property (4.19) and solves the system of equations

\[
\tilde{\Gamma}^{(a, b)} \lambda^T = 0
\]

for a maximal \( a < 2\tau - 1 - \deg(\lambda^{(R)}) \) such that

\[
\tilde{\Gamma}^{(a+1, b)} \lambda^T \neq 0.
\]

- The polynomials \( \nu(x)^{(R)} \) and \( \mu(x)^{(R)} \) satisfy

\[
\nu^{(R)}(x) = S(x)\sigma^{(R)}(x) \mod x^{2\tau}
\]

\[
\mu^{(R)}(x) = S(x)\lambda^{(R)}(x) \mod x^{2\tau}.
\]

We note that the polynomials \( \sigma(x), \lambda(x), \nu(x) \) and \( \mu(x) \) satisfy properties (4.19)-(4.24) for the empty set of erasure positions. We are interested in a nested sequence of sets of erasure positions. Thus let \( \{R_j\}_{j=0}^{2\tau} \) be a sequence of erasure position sets with \( R_0 = \{\} \) and \( R_j = R_{j-1} \cup \{i_j\} \). For polynomials \( \sigma^{(R_j)}(x) \) and \( \lambda^{(R_j)}(x) \), that solve equations \( \tilde{\Gamma}^{(2\tau - 1 - l, l)} \sigma^T = 0 \) and \( \tilde{\Gamma}^{(a, b)} \lambda^T = 0 \), we define a number \( s^{(j)} \) as

\[
s^{(j)} = (2\tau - 1 - l) - a.
\]

It follows from Lemma 4.2 that \( s^{(0)} \) equals \( d(C) - 2\deg(\sigma) \). Whenever \( s^{(j)} \) is non-positive, we can add an arbitrary multiple of \( \lambda^{(R_j)}(x) \) to \( \sigma^{(R_j)}(x) \) with neither changing the degree of \( \sigma^{(R_j)}(x) \) nor the system of equations solved by \( \sigma^{(R_j)}(x) \). In our case, we use this degree of freedom in order to prescribe certain zeros of \( \sigma^{(R_j)}(x) \). The following recursive set of equations describes an algorithm to iteratively determine the relevant polynomials \( \sigma^{(R_j)}(x) \) and \( \nu^{(R_j)}(x) \).
Algorithm 4.1
Initialization:

\[
\sigma^{(R_0)}(x) = \sigma(x), \quad \nu^{(R_0)}(x) = \nu(x), \\
\lambda^{(R_0)}(x) = \lambda(x), \quad \mu^{(R_0)}(x) = \mu(x), \\
s^{(0)} = d(C) - 2\deg(\sigma).
\]

Iteration step:

\[
\delta^{(j)} = \begin{cases} 
1 & (\sigma^{(R_j)}(\alpha^{i+1}) = 0) \lor \left( (s^{(j)} \leq 0) \land (\lambda^{(R_j)}(\alpha^{i+1}) \neq 0) \right) \\
0 & \text{otherwise.}
\end{cases}
\]

\[
\begin{bmatrix}
\sigma^{(R_{j+1})}(x) & \nu^{(R_{j+1})}(x) \\
\lambda^{(R_{j+1})}(x) & \mu^{(R_{j+1})}(x)
\end{bmatrix}
= 
\begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix}
\delta^{(j)} + 
\begin{bmatrix}
(1 - \delta^{(j)})(x - \alpha^{i+1}) & \delta^{(j)}(x - \alpha^{i+1}) \\
\lambda^{(R_j)}(\alpha^{i+1}) & -\sigma^{(R_j)}(\alpha^{i+1})
\end{bmatrix}
\begin{bmatrix}
\sigma^{(R_j)}(x) & \nu^{(R_j)}(x) \\
\lambda^{(R_j)}(x) & \mu^{(R_j)}(x)
\end{bmatrix}
\tag{4.26}
\]

\[
s^{(j+1)} = s^{(j)} + 2\delta^{(j)} - 1
\]

Theorem 4.3 Let a word \( y = c + e \in \mathbb{F}_q^n, c \in C \) and a nested sequence of erasure position sets \( \{R_j\}_{j=0}^{2\tau} \) be given. Moreover, let polynomials \( \sigma^{(R_0)}(x), \lambda^{(R_0)}(x), \nu^{(R_0)}(x), \mu^{(R_0)}(x) \) be given, as provided by the BMA. Algorithm 4.1 calculates the polynomials \( \sigma^{(R_j)}(x) \) and \( \nu^{(R_j)}(x) \) for all \( 0 \leq j \leq 2\tau \) with complexity \( O(d^2) \). If the weight of the vector \( e \) outside the erasure set \( R_j \) is less than \( (d(C) - |R_j|)/2 \) then \( \sigma^{(R_j)}(x) \) is error-locating.

Proof. We have to prove that the polynomials, constructed in Algorithm 4.1, satisfy properties (4.19)-(4.24). For the empty set of erasures \( R_0 \) we can verify this from the properties of the BMA. Due to the inductive nature of the algorithm, we assume that properties (4.19)-(4.24) are satisfied for polynomials \( \sigma^{(R_j)}(x), \lambda^{(R_j)}(x), \nu^{(R_j)}(x), \mu^{(R_j)}(x) \) and that \( s^{(j)} \) satisfies equation (4.25). It suffices to prove that \( \sigma^{(R_{j+1})}(x), \lambda^{(R_{j+1})}(x), \nu^{(R_{j+1})}(x), \mu^{(R_{j+1})}(x) \) and \( s^{(j+1)} \) satisfy properties (4.19)-(4.24). From the construction
it is clear that $\sigma^{(R_j+1)}(x)$ and $\lambda^{(R_j+1)}(x)$ both have a root at $\alpha^{i_j+1}$. Moreover, they are both nonzero functions with greatest common divisor

$$\prod_{i_j \in R_{j+1}} (x - \alpha^{i_j}),$$

which follows from equation (4.26) and property (4.19) for polynomials $\sigma^{(R_j)}(x)$ and $\lambda^{(R_j)}(x)$. To prove properties (4.20) and (4.21), we distinguish the following events:

1. $\sigma^{(R_j)}(\alpha^{i_j+1}) = 0$

   In this case $\delta^{(j)}$ equals 1 and $\sigma^{(R_j+1)}(x)$ is a scalar multiple of $\sigma^{(R_j)}(x)$ and thus of minimal degree satisfying property (4.20). Moreover, $\lambda^{(R_j)}(\alpha^{i_j+1}) \neq 0$ holds, and the maximal $a$ such that property (4.21) holds for an erasure set $R_{j+1}$ is less than the corresponding number for an erasure set $R_j$. From the structure of Hankel matrices, it follows that $\lambda^{(R_{j+1})}(x)$, as calculated in the algorithm, satisfies (4.21). In this case $s^{(j+1)}$ equals $s^{(j)} - 1$ according to equation (4.25).

2. $\sigma^{(R_j)}(\alpha^{i_j+1}) \neq 0 \land \lambda^{(R_j)}(\alpha^{i_j+1}) \neq 0 \land s^{(j)} \leq 0$

   The situation is almost the same as in case 1. We only note that the degree of $\lambda^{(R_j)}(x)$ is less than the degree of $\sigma^{(R_j)}(x)$ as otherwise $s^{(j)}$ can not be less than or equal to zero. $\sigma^{(R_{j+1})}(x)$ is obtained by a suitable linear combination of $\sigma^{(R_j)}(x)$ and $\lambda^{(R_j)}(x)$ and has degree equal to the degree of $\sigma^{(R_j)}(x)$.

3. $\sigma^{(R_j)}(\alpha^{i_j+1}) \neq 0 \land s^{(j)} > 0$

   If $s^{(j)}$ is greater than zero we need to increase the degree of $\sigma^{(R_j)}(x)$ in order to find $\sigma^{(R_j+1)}(x)$. To see this, note that the existence of a polynomial $\sigma^{(R_{j+1})}(x)$ of the same degree as $\sigma^{(R_j)}(x)$ would admit the construction of a polynomial that solves the equation $\tilde{\Gamma}(2r - 1 - \deg(\sigma^{(R_j)})) \lambda^T = 0$ for $l < \deg(\sigma^{(R_j)})$. The constructed polynomial would contradict the definition of $\lambda^{(R_j)}(x)$ and $s^{(j)}$.

   The polynomial $\lambda^{(R_{j+1})}(x)$ can now be constructed as a suitable linear combination of $\sigma^{(R_j)}(x)$ and $\lambda^{(R_j)}(x)$ such that $\lambda^{(R_{j+1})}(\alpha^{i_j+1})$ equals
zero. \( \lambda^{(\mathcal{R}_{j+1})}(x) \) of some degree \( b' \) constructed in that way solves the equation \( \hat{\Gamma}(a, b')\lambda^T = 0 \) for the same \( a \) as \( \lambda^{(\mathcal{R}_j)}(x) \), though the degree of \( \lambda^{(\mathcal{R}_{j+1})}(x) \) need not be equal to the degree of \( \lambda^{(\mathcal{R}_j)}(x) \). Thus the function \( \lambda^{(\mathcal{R}_{j+1})}(x) \) satisfies property (4.21). In this case \( s^{(j+1)} \) equals \( s^{(j)} + 1 \) according to equation (4.25).
4. \(\sigma^{(R_j)}(\alpha^{i_j+1}) \neq 0 \land \lambda^{(R_j)}(\alpha^{i_j+1}) = 0 \land s(j) < 0\)

The case is similar to case 3. We only need to show that the degree of polynomial \(\sigma^{(R_j+1)}(x)\) is greater than the degree of \(\sigma^{(R_j)}(x)\). Assume the degree would be equal. As in case 3, we can construct a polynomial \(\lambda(x)\) which solves the equation \(\tilde{\Gamma}^{(2r-1-\deg(\sigma)/l)}\lambda^T = 0\) for \(l < \deg(\sigma^{(R_j)}(x))\) and which is nonzero in the point \(\alpha^{i_j+1}\). For a suitable choice of a polynomial \(A(x)\) we can construct a polynomial \(A(x)\lambda^{(R_j)}(x) + \lambda'(x)\) which either contradicts the definition of \(\sigma^{(R_j)}(x)\) or \(\lambda^{(R_j)}(x)\).

The four cases cover all possible situations because the case \((\sigma^{(R_j)}(\alpha^{i_j+1}) = 0) \land (\lambda^{(R_j)}(\alpha^{i_j+1}) = 0)\) can not occur. It can be checked directly that Algorithm 4.1 performs precisely the operations required in the four cases.

The error-locating property follows either from the results of section III or immediately, as the constructed polynomials \(\sigma^{(R_j)}(x)\) are scalar multiples of the error-erasure-locating functions found in the standard BMA for the error-erasure case. The functions \(\nu^{(R_j)}(x)\) and \(\mu^{(R_j)}(x)\) satisfy properties (4.23) and (4.24) due to the fact that the definition of \(\nu^{(R_j)}(x)\) and \(\mu^{(R_j)}(x)\) describes a ring homomorphism from \(\mathbb{F}_q[x]\) to the ring of polynomials modulo the ideal generated by \(x^{2r}\). It thus suffices to notice that \(\nu^{(R_j)}(x)\) and \(\mu^{(R_j)}(x)\) are subject to the same operations as \(\sigma^{(R_j)}(x)\) and \(\lambda^{(R_j)}(x)\). Finally, we note that every iteration step requires two linear combinations of polynomials of degree less than \(d\) and two multiplications of polynomials of degree less than \(d\) with a linear factor. The complexity of an iteration step is thus in the order \(O(d)\) and we have to perform at most \(d\) iteration steps. It follows that we can find all relevant polynomials with complexity \(O(d^2)\). \(\Box\)

**Remark 4.5** Theorem 4.3 shows that we can find all error-erasure locating polynomials together with their associated polynomials with complexity \(O(d^2)\). If the error vector has weight less than \(\tau - |R_j|/2\) outside the erasure positions, \(\sigma^{(R_j)}\) is error-locating. Thus we can apply Forney’s formula (4.18). In particular we can determine the support of the error vector found in the \(j\)th decoding attempt as the zero set of \(\sigma^{(R_j)}(x)\) minus the zero set of \(\nu^{(R_j)}(x)\). If the \(j\)th decoding attempt fails then \(\sigma^{(R_j)}(x)\) has less than \(\deg(\sigma^{(R_j)})\) roots in \(\mathbb{F}_q\). The determination of the zero sets of these polynomials is a crucial operation. The standard technique to do this is to test all
elements of \( \mathbb{F}_q \), which requires for each polynomial a complexity proportional to \( nd \). Thus the complexity for finding all relevant zero sets using this technique is proportional to \( nd^2 \). This is just the complexity required by a GMD decoding procedure using \( \lceil \frac{n}{2} \rceil \) independent decoding attempts each of complexity \( O(nd) \). We circumvent this large complexity by translating all operations performed in Algorithm 4.1 to vector operations, where the vectors \( \sigma, \lambda, \nu \) and \( \mu \) are the evaluation vectors of the corresponding polynomials in the points \( x \in \mathbb{F}_q \), \( 0 \leq i \leq n-1 \). Multiplication of a polynomial with \( x \) is thus translated to the component-wise multiplication of an evaluation vector with the vector \( (\alpha^0, \alpha^1, \ldots, \alpha^{n-1}) \). Performing the operations of Algorithm 4.1 with vectors does not increase the number of required operations. As every vector operation requires complexity \( O(n) \) instead of \( O(d) \), the complexity required by Algorithm 4.1, performed on vectors, is \( O(dn) \).

Given an RS-code \( C \), a received vector \( y = c + e \), and an ordering of positions in \( y \) according to some reliability information, a fast GMD decoding algorithm thus implies the following steps:

**Algorithm 4.2**

1. Calculation of the syndromes \( S_i \) for \( 1 \leq i \leq 2\tau \).
2. Application of the BMA to the syndrome array to find polynomials \( \sigma(x), \lambda(x), \nu(x) \) and \( \mu(x) \).
3. Evaluation of polynomials \( \sigma(x), \lambda(x), \nu(x) \) and \( \mu(x) \) in \( q-1 \) points.
4. Application of Algorithm 4.1 to find the set of polynomials \( \sigma^{(R_i)}(x) \) and \( \nu^{(R_i)}(x) \), alternatively their evaluation vectors \( \sigma^{(R_i)}(x) \) and \( \nu^{(R_i)}(x) \), \( i = 0, 2, \ldots, 2\tau \).
5. Determination of the best estimate for the codeword \( c \) based on the zero sets of polynomials \( \sigma^{(R_i)}(x) \) and \( \nu^{(R_i)}(x) \).
6. Completion of the decoding by finding the error values.

We summarize the results of this section in the following theorem.
Chapter 4. Fast GMD decoding of AG and RS codes

Theorem 4.4 Let an RS code $C$ of length $n$ and minimum Hamming distance $d$ be given. For a given word $y = c + e, c \in C$, let a reliability information on the positions in $y$ be given. We can perform a GMD-decoding of the word $y$ with complexity $O(nd)$.

Proof. We estimate the complexity of every step in the GMD decoding algorithm 4.2. Finding all the syndromes can be done with complexity $O(nd)$. The BMA requires at most $O(d^2)$ operations in order to find polynomials $\sigma(x)$ and $\lambda(x)$ together with their associated polynomials [11, p.180]. The evaluation of these polynomials needs complexity $O(nd)$. Algorithm 4.1, together with Remark 4.5, yields that the complexity required to find the zero sets of $\sigma^{R_i}(x)$ and $\nu^{R_i}(x)$ lies in the order $O(nd)$. At this point we can decide upon the support of the most probable error positions, which requires $O(nd)$ operations. We complete the decoding using Forney’s formula. To this end we have to calculate

$$\sigma^{(R_i)}(\alpha^l) = \prod_{j \neq l} (\alpha^l - \alpha^j)$$

for any found error position $l$, where $j$ runs through the zero set of $\sigma^{(R_i)}(x)$ with exception of $l$. This calculation demands at most $d - 2$ multiplications for any of the at most $d - 1$ error positions and thus requires a computational complexity in the order $O(d^2)$. The theorem follows, as the complexity of all steps is upper bounded by $O(nd)$. □

Remark 4.6 The above theorem remains valid with minor obvious modifications when considering codes that are derived from RS codes, such as BCH codes and generalized RS codes.

Remark 4.7 It is worth mentioning that steps 1, 2, 3 and 6 or equivalent steps have to be performed by a decoding procedure that corrects only errors. The additional complexity required for the use of reliability information is thus the complexity required by steps 4 and 5. Step 5 is common to all GMD decoding schemes and lower bounds the asymptotic complexity of such schemes. This step includes calculation of the generalized distance from the received vector to every vector on the list of tentative codewords. This calculation implies $O(n)$ operations for any of the $\left\lfloor \frac{d}{2} \right\rfloor$ possible solutions. Therefore complexity $O(nd)$ is asymptotically optimal for GMD schemes as was remarked by Sorger in [2].
VI Algebraic-geometric codes

Algebraic-geometric codes, which were first introduced by Goppa in [15], can be seen as a generalization of RS codes. For a brief introduction into the topic we refer to the book of van Lindt and van de Geer [16], where standard terminology for the treatment of AG codes is defined. Let $\mathcal{X}$ be a nonsingular, absolutely irreducible, projective curve over $\mathbb{F}_q$ with genus $g$ and let $D$ be the divisor obtained as the sum of $n$ distinct rational points $P_i$, $i = 0 \ldots n-1$, on $\mathcal{X}$. A rational function $f$ on $\mathcal{X}$ may be characterized, up to multiplication with a constant, by a divisor of zeros and poles of $f$ which is denoted by $(f)$. For a divisor $G$ let $L(G)$ denote the space of rational functions $f$ on $\mathcal{X}$ that satisfy

$$L(G) = \{ f : (f) + G \geq 0 \}. \quad (4.27)$$

Let $l(G)$ denote the dimension of $L(G)$. From the Riemann-Roch theorem we have the relation

$$l(G) \geq \deg(G) - g + 1,$$

with equality if $\deg(G) > 2g - 2$. The algebraic geometric functional code $C_L(D, G)$ is defined as

$$C_L(D, G) = \{ (f(P_0), f(P_1), \ldots, f(P_{n-1})) : f \in L(G) \}.$$

Estimates of the parameters of $C_L(D, G)$ can be found in e.g [16] as length $n = \deg(D)$, minimum distance $d(C_L) \geq n - \deg(G)$ and dimension $k(C_L) = l(G) - l(G - D) \geq \deg(G) - g + 1$ for $\deg(G) < n$.

The residual code $C_{\Omega}(D, G)$ may be defined as

$$C_{\Omega}(D, G) = (C_L(D, G))^\perp$$

and has parameters $n = \deg(D)$, designed minimum distance $d^*(C_{\Omega}) = \deg(G) - 2g + 2$ and $k(C_{\Omega}) \geq n - \deg(G) + g - 1$ for $\deg(G) > 2g - 2$, cf. [16]. The designed minimum distance of the code is also simply denoted as $d$ when no confusion can arise.

We will only consider the case that $G$ is a multiple of a distinguished rational point $P_\infty$ that is not in the support of $D$, i.e. $G = mP_\infty$. We assume that $d(C_{\Omega}(D, mP_\infty)) - g$ is odd and define $\tau$ as

$$\tau = \frac{d(C_{\Omega}(D, G)) - g - 1}{2}.$$
Chapter 4. Fast GMD decoding of AG and RS codes

The parameter $\tau$ reflects the guaranteed error correction capability of the so called basic decoding algorithm [17]. Efficient schemes for decoding errors up to half the designed distance were given by Feng and Rao [18], Duursma [19], and Ehrhard [20]. The following proposition describes a sequence of error-erasure locating pairs for AG codes, that realizes the GMD error correction capability of the basic algorithm. The approach will later be extended to guarantee error-erasure-location up to the designed minimum distance.

**Proposition 4.3** Let $C$ be the code $C_D(D_m P_\infty)$ with $D = P_0 + P_1 + \ldots + P_{k-1}$. Let $U_i$ and $V_i$ be the AG codes $C_L(D, (\tau + g + i)P_\infty)$ and $C_D(D, (m - \tau - g - i)P_\infty)$ respectively. The pairs $(U_i, V_i)$, $i = 0, 1, \ldots, \tau$ give a sequence of $(t_i, \rho_i)$-error-erasure-correcting pairs with $U_i \subseteq U_{i+1}$ and $V_i \subseteq V_{i+1}$ such that

$$t_i \geq \tau - i,$$

$$\rho_i \geq 2i.$$  \hspace{1cm} (4.28)

**Proof.** The proof follows the proof of Proposition 4.2. Conditions (4.6), (4.7) and (4.11) can be checked with the above estimates on minimum distances and dimensions of AG codes. To prove that (4.5) holds we prove the equivalent statement

$$U_i \ast V_i^\perp \subseteq C^\perp.$$

The spaces $U_i$ and $V_i^\perp$ are obtained as evaluation of function spaces $L((\tau + g + i)P_\infty)$ and $L((m - (\tau + g + i))P_\infty)$. Thus, the product of a function $f \in L((\tau + g + i)P_\infty)$ and $g \in L((m - (\tau + g + i))P_\infty)$ is a function in $L(mP_\infty)$, which corresponds to a codeword in $C^\perp$. \hfill \Box

The above proposition gives the possibility to directly use the general setup described in section III for the GMD decoding of AG codes. It follows from Theorem 4.2 that we can determine all error-erasure-locating functions with complexity $O(n^3)$. We can do even better, however, by combining fast decoding procedures, that were first proposed by Justesen at al. in [21], with our scheme in the same way as we used a BMA for RS codes. The rest of this section will treat this approach and deals with the question how to
VI. Algebraic-geometric codes

125

combine the output of such a BMA for AG codes in order to get an efficient GMD decoding procedure. For the BMA itself, applied to AG codes, the reader is referred to [23].

A BMA type algorithm for AG codes requires the introduction of some additional structure to the matrix $\Gamma$. The rows of the generator matrices $G_{\Gamma_i}$ and parity check matrices $H_{\Gamma_0}$ are obtained by evaluating basis functions of spaces $L(lP_\infty)$ for suitable $l$. The choice of basis functions for $L(lP_\infty)$ naturally influences the structure of the matrix $\Gamma$. In order to fix a basis for $L(lP_\infty)$ that permits a Berlekamp-Massey type algorithm, we recall some properties of vector spaces $\mathcal{V}_i$.

Let $o_j$ be a non-negative integer such that $L(o_jP_\infty) \neq L((o_j - 1)P_\infty)$. The number $o_j$ is called a non-gap at $P_\infty$ or simply a non-gap, which means that there exists a function $\phi_{o_j}$ with a pole of order $o_j$ at $P_\infty$. The constant function has no poles and it follows that $o_0 = 0$ and $\phi_0 = 1$. The set of non-negative integers $l$ such that $L(lP_\infty) = L((l - 1)P_\infty)$ is called the gap sequence at $P_\infty$ and denoted by $G$. As a consequence of the Riemann-Roch Theorem the non-gaps satisfy

$$0 = o_0 < o_1 < o_2 < \cdots < o_{g-1} < 2g$$

and

$$o_i = i + g, \text{ for } i \geq g.$$ 

The value of $o_1$ is crucial for complexity considerations. $P_\infty$ is called a Weierstrass point, if $o_1$ is less than $g + 1$. It is clear that when $o_i$ and $o_j$ are non-gaps then $o_i + o_j$ is also a non-gap. This property of the non-gaps allows choosing a basis $\{1, \phi_{o_1}, \phi_{o_2}, \ldots\}$ for $L(lP_\infty)$ such that $\phi_{o_j + o_i}$ equals $\phi_{o_i} \phi_{o_j}$ if $o_j$ is a non-gap. In other words we assume the basis to be closed under multiplication with $\phi_{o_1}$ as long as this is consistent with the maximal pole order $l$. We introduce virtual basis functions $\phi_i = 0$ for $i \in G$. This streamlines notations as we can mimic a very regular structure of the spaces $L(lP_\infty)$. A function $f$ in $L(lP_\infty)$ can be thought of as uniquely being expanded as

$$\sum_{i=0}^{l} f_i \phi_i, \quad f_i = 0 \forall i \in G. \quad (4.30)$$
For a vector \( \mathbf{y} = \mathbf{c} + \mathbf{e}, \mathbf{c} \in C_\Omega(D, mP_{\infty}) \) we define syndromes \( S_\mathbf{y}(i, j) \) as

\[
S_\mathbf{y}(i, j) = \sum_{l=0}^{n-1} y_l \phi_i(P_l) \phi_j(P_l). \tag{4.31}
\]

It is clear from the definition of the code \( C_\Omega(D, mP_{\infty}) \) that \( S_\mathbf{y}(i, j) \) equals \( S_\mathbf{e}(i, j) \) if \( i + j \leq m \) holds. With the chosen basis for the spaces \( L(I P_{\infty}) \), the matrix \( \tilde{\Gamma} \) has entries

\[
\tilde{\Gamma}_{i,j} = S_\mathbf{e}(i, j) \text{ for } i + j \leq m. \tag{4.32}
\]

The entries \( S_\mathbf{e}(i, j) \) for \( i + j > m \) are unknown except for the case that \( i \) or \( j \) is a gap in which case \( S_\mathbf{e}(i, j) \) equals zero. The entries of \( \tilde{\Gamma} \) satisfy the property

\[
\tilde{\Gamma}_{i,j} = \tilde{\Gamma}_{i-o_1,j+o_1}, \text{ if } (i - o_1 \not\in G) \lor (j \not\in G), \tag{4.33}
\]

which can be seen from equation (4.31) and the fact that \( \phi_i = \phi_{i-o_1} \phi_{o_1} \) and \( \phi_{j+o_1} = \phi_j \phi_{o_1} \). Relation (4.33), which resembles the defining relation of Hankel matrices, is the key to a Berlekamp-Massey type algorithm. Such an algorithm was first derived by Justesen et al. in [21], based on an algorithm proposed by Sakata in [22]. Here we invoke an algorithm suggested in [23], which is different from the algorithm proposed in [21], though it naturally relies on similar observations. The following lemma, which is essentially equal to Lemma 2 from [23] constitutes one of the cornerstones of this algorithm and it will be needed in some subsequent proofs. Similar to the case of RS codes, where we identified vectors with polynomials, here we identify vectors \( \mathbf{\sigma} \) with functions \( \mathbf{\sigma} \) such that the vector \( \mathbf{\sigma} \) is the coefficient vector for the expansion of \( \mathbf{\sigma} \) according to (4.30). We will say that a function \( \mathbf{\sigma} \) solves a system of linear equations if the corresponding vector \( \mathbf{\sigma} \) is a solution.

**Lemma 4.4** Let a function \( \mathbf{\sigma} \) solve the system of linear equations

\[
\tilde{\Gamma}^{(a,j)} \mathbf{\sigma} = \mathbf{0}. \tag{4.34}
\]

Then the function \( \mathbf{\sigma}' = \phi_{o_1} \mathbf{\sigma} \) solves

\[
\tilde{\Gamma}^{(a-o_1,j+o_1)} \mathbf{\sigma} = \mathbf{0}.
\]
Proof. With equations (4.31), (4.32) and \( \sigma = \sum_{i=0}^{j} \alpha_i \phi_i \) equation (4.34) implies

\[
\sum_{j=0}^{n-1} y_j \sigma(P_j) \phi_i(P_j) = 0 \quad \text{for } i \leq a.
\]

Due to the structure of the chosen basis we have

\[
\sum_{j=0}^{n-1} y_j \sigma(P_j) \phi_0(P_j) \phi_i(P_j) = 0 \quad \text{for } i \leq a - o_1
\]

from which the lemma follows. \( \square \)

Here we do not intend to give a description of the algorithm from [23]. For our purposes it is sufficient to note that given a matrix \( \tilde{\Gamma} \) it calculates two sets of functions \( \{\sigma^{(i,m)}\}_{i=0}^{n-1} \) and \( \{\lambda^{(j,m)}\}_{j=0}^{o_1-1} \) together with associated numbers \( a_{\sigma}^{(m,i)} \) and \( a_{\lambda}^{(m,j)} \) such that the following properties are satisfied.

**Property 1** \( \sigma^{(i,m)} \) is a function of least pole order \( l \equiv i \mod o_1 \) that solves the system of equations

\[
\tilde{\Gamma}^{(m-l,j)} \sigma^T = 0.
\]

The number \( a_{\sigma}^{(m,i)} \) equals \( m - l + 1 \).

**Property 2** A function \( \lambda^{(j,m)} \) of pole order \( b \) solves the system of equations

\[
\tilde{\Gamma}^{(a-1,b)} \lambda^T = 0, \quad \sum_{l=0}^{b} \tilde{\Gamma}_{a,l} \lambda_l = \Delta \neq 0
\]

such that \( a \) is the greatest number smaller than \( m - b \) that satisfies \( a \equiv j \mod o_1 \). The number \( a_{\lambda}^{(m,j)} \) equals \( a \). If no such function exists then \( \lambda^{(j,m)} \) is set to zero.

The interpretation of functions \( \sigma^{(i,m)} \) and \( \lambda^{(j,m)} \) is analogous to the roll that polynomials \( \sigma \) and \( \lambda \) play in the BMA for RS codes. The functions \( \sigma^{(i,m)} \) solve a system of linear equations that describes multi-dimensional recurrence relations on the known syndromes, whereas the functions \( \lambda^{(j,m)} \) failed
to satisfy a recurrence relation at an earlier step of the algorithm. The function \( \sigma^{(i,m)} \) of lowest pole order is in fact the error-locating function proposed by the basic algorithm, which directly follows from Proposition 4.3. The main difference to a one-dimensional BMA is the fact that two sets of functions each of size \( o_1 \) are considered. The numbers \( a_{\sigma}^{(m,i)} \) and \( a_{\lambda}^{(m,j)} \) characterize the state of the algorithm by indicating which system of equations is solved by the corresponding function.

Let \( \mathcal{R} \) be a set of erasure positions. We want to determine sets of functions for error-erasure decoding that are similar to functions \( \sigma^{(i,m)} \) and \( \lambda^{(j,m)} \). More precisely, our goal is to find sets of functions

\[
\{ \sigma^{(i,m,\mathcal{R})} \}_{i=0}^{o_1-1}, \quad \{ \lambda^{(j,m,\mathcal{R})} \}_{j=0}^{o_1-1}
\]

and corresponding numbers \( a_{\sigma}^{(m,i,\mathcal{R})} \) and \( a_{\lambda}^{(m,j,\mathcal{R})} \) that satisfy the following properties.

**Property 1’** \( \sigma^{(i,m,\mathcal{R})} \) is a function of least pole order \( l \equiv i \mod o_1 \) that solves the system of equations

\[
\tilde{\Gamma}^{(m-l,l)} \sigma^T = 0 \\
\sigma^{(i,m,\mathcal{R})}(P_l) = 0 \forall l \in \mathcal{R}.
\]

The number \( a_{\sigma}^{(m,i,\mathcal{R})} \) equals \( m - l + 1 \).

**Property 2’** A function \( \lambda^{(j,m,\mathcal{R})} \) of any pole order \( b \) solves the system of equations

\[
\tilde{\Gamma}^{(a-1,b)} \lambda^T = 0, \quad \sum_{l=0}^{b} \tilde{\Gamma}_{a,l} \lambda_l = \Delta \neq 0 \\
\lambda^{(j,m,\mathcal{R})}(P_l) = 0 \forall l \in \mathcal{R},
\]

such that \( a \) is the greatest number smaller than \( m - b \) that satisfies \( a \equiv j \mod o_1 \). The number \( a_{\lambda}^{(m,j,\mathcal{R})} \) equals \( a \). If no such function exists then \( \lambda^{(j,m,\mathcal{R})} \) is set to zero.

The function \( \sigma^{(i,m,\mathcal{R})} \) of lowest pole order is just the error-erasure-locating function that is obtained from the basic algorithm for the set of erasure positions \( \mathcal{R} \), cf. Proposition 4.3.
VI. Algebraic-geometric codes

In the general setup of section III we could find the error-erasure-locating vectors for nested sets of erasures from the spaces $\Sigma_i$. In the case of AG codes we can bypass the spaces $\Sigma_i$, as we did for RS codes, and find the functions $\sigma(\mathcal{R}_i)$ and $\lambda(\mathcal{R}_i)$ directly from the functions $\sigma(i,m)$ and $\lambda(j,m)$. Let $\mathcal{R}_i \subset \mathcal{R}_{i+1}$ be the set of erasure positions $\{i_1, i_2, \ldots, i_l\}$. The following algorithm gives the basic iteration step to find functions $\sigma(\mathcal{R}_{i+1})$ and $\lambda(\mathcal{R}_{i+1})$ from the functions $\sigma(\mathcal{R}_i)$ and $\lambda(\mathcal{R}_i)$.

**Algorithm 4.3**

1. **Input**: Functions $\sigma(i,m,\mathcal{R}_i)$ and $\lambda(j,m,\mathcal{R}_i)$, numbers $a_{\sigma}(m,i,\mathcal{R}_i)$, $a_{\lambda}(m,j,\mathcal{R}_i)$ and the new erasure position $i_{l+1} = \mathcal{R}_{l+1} \setminus \mathcal{R}_l$.

2. Find the function $\xi$ among the functions $\sigma(i,m,\mathcal{R}_i)$ and $\lambda(j,m,\mathcal{R}_i)$ which solves the equation

$$\Gamma(a,b)\xi = 0$$

for a maximal $a$ such that $\xi(P_{i+1}) \neq 0$. If two such functions exist let $\xi$ be the one of lower pole order.

3. Construct the functions $\sigma(i,m,\mathcal{R}_{i+1}), \lambda(j,m,\mathcal{R}_{i+1})$ according to

$$\sigma(i,m,\mathcal{R}_{i+1}) = \begin{cases} 
(\phi_{0i} - \phi_{01}(P_{i+1}))\xi & \sigma(i,m,\mathcal{R}_i) = \xi \\
\xi(P_{i+1})\sigma(i,m,\mathcal{R}_i) - \sigma(i,m,\mathcal{R}_i)(P_{i+1})\xi & \text{otherwise},
\end{cases}$$

and

$$\lambda(j,m,\mathcal{R}_{i+1}) = \begin{cases} 
(\phi_{0j} - \phi_{01}(P_{i+1}))\xi & \lambda(j,m,\mathcal{R}_i) = \xi \\
\xi(P_{i+1})\lambda(j,m,\mathcal{R}_i) - \lambda(j,m,\mathcal{R}_i)(P_{i+1})\xi & \text{otherwise}.
\end{cases}$$

4. The numbers $a_{\sigma}(m,i,\mathcal{R}_{i+1})$ and $a_{\lambda}(m,j,\mathcal{R}_{i+1})$ are modified according to

$$a_{\sigma}(i,m,\mathcal{R}_{i+1}) = \begin{cases} 
a_{\sigma}(i,m,\mathcal{R}_i) - o_1 & \sigma(i,m,\mathcal{R}_i) = \xi \\
a_{\sigma}(i,m,\mathcal{R}_i) & \text{otherwise},
\end{cases}$$

and

$$a_{\lambda}(j,m,\mathcal{R}_{i+1}) = \begin{cases} 
a_{\lambda}(j,m,\mathcal{R}_i) - o_1 & \lambda(j,m,\mathcal{R}_i) = \xi \\
a_{\lambda}(j,m,\mathcal{R}_i) & \text{otherwise}.
\end{cases}$$
Proof. We have to show that the calculated functions \( \sigma^{(i,m,\mathcal{R}_{t+1})} \) and \( \lambda^{(j,m,\mathcal{R}_{t+1})} \) satisfy the required properties 1′ and 2′ for an erasure set \( \mathcal{R}_{t+1} \). Step 3 of the algorithm ensures that they have zeros in points indicated by the set \( \mathcal{R}_{t+1} \). With exception of the function that was chosen as \( \xi \), the functions \( \sigma^{(i,m,\mathcal{R}_{t+1})} \) and \( \lambda^{(j,m,\mathcal{R}_{t+1})} \) satisfy also the required properties for \( \sigma^{(i,m,\mathcal{R}_{t})} \) and \( \lambda^{(j,m,\mathcal{R}_{t})} \) which proves the validity of the algorithm for these functions.

We distinguish the two cases that \( \xi \) is a function \( \lambda^{(j,m,\mathcal{R}_{t})} \) or \( \sigma^{(i,m,\mathcal{R}_{t})} \), respectively. Assume first that \( \xi \) equals \( \lambda^{(j,m,\mathcal{R}_{t})} \). Then there does not exist a function \( \lambda^{(j,m,\mathcal{R}_{t+1})} \) that solves the system of equations

\[
\tilde{\Gamma}(a_{\lambda}^{(j,m,\mathcal{R}_{t})}b)X^T = (0, 0, \ldots, 0, \Delta \neq 0)^T, \lambda^{(j,m,\mathcal{R}_{t+1})}(P_{t+1}) = 0,
\]

because this would contradict the definition of \( \xi \). It follows that \( a_{\lambda}^{(j,m,\mathcal{R}_{t+1})} \) equals \( a_{\lambda}^{(j,m,\mathcal{R}_{t})} - o_1 \) as we have \( a_{\lambda}^{(j,m,\mathcal{R}_{t+1})} \equiv a_{\lambda}^{(j,m,\mathcal{R}_{t})} \mod o_1 \). Thus the function \( \lambda^{(j,m,\mathcal{R}_{t+1})} \) may be chosen as \( (\phi o_1 - \phi o_1(P_{t+1}))\lambda^{(j,m,\mathcal{R}_{t})} \), which guarantees property 2′.

If \( \xi \) equals \( \sigma^{(i,m,\mathcal{R}_{t})} \) then the pole order of \( \sigma^{(i,m,\mathcal{R}_{t+1})} \) equals the pole order of \( \sigma^{(i,m,\mathcal{R}_{t})} \) plus \( o_1 \) for a similar reason. Thus \( \sigma^{(j,m,\mathcal{R}_{t+1})} \), chosen as \( (\phi o_1 - \phi o_1(P_{t+1}))\sigma^{(j,m,\mathcal{R}_{t+1})} \), satisfies the required property 1′.

We estimate the complexity of Algorithm 4.3. There are all together \( 2o_1 \) functions to be considered. Step 2 and 3 involve the evaluation of the functions in a certain point, which requires complexity \( O(o_1d) \). The set of numbers \( a_{\lambda}^{(i,m,\mathcal{R}_{t})} \) and \( a_{\lambda}^{(j,m,\mathcal{R}_{t})} \) is then sufficient to choose the function \( \xi \). The construction of \( 2o_1 \) functions in step 3 also requires complexity \( O(o_1d) \). This implies that Algorithm 4.3 requires a complexity \( O(o_1d^2) \).

The set \( \mathcal{R}_0 \) denotes the empty set of erasure positions. For the empty set of erasure positions, the properties 1.2 and 1′,2′ coincide. Thus the algorithm from [23] provides functions \( \sigma^{(i,m,\mathcal{R}_0)} \) and \( \lambda^{(j,m,\mathcal{R}_0)} \) directly. We have to perform \( d \) iterations of Algorithm 4.3 in order to successively include all erasure positions. Thus the overall complexity to find the set of all interesting error-erasure-locating functions from functions \( \sigma^{(i,m,\mathcal{R}_0)} \) and \( \lambda^{(j,m,\mathcal{R}_0)} \) is in the order \( O(o_1d^2) \).
**Remark 4.8** In correspondence to Remark 4.5, we note that Algorithm 4.3 might as well be performed on vectors which are the evaluation of the functions $\sigma(i, m, \mathcal{R}_i)$ and $\lambda(j, m, \mathcal{R}_j)$ in points $P_l$, $0 \leq l \leq n - 1$. Using this setup, we do not have to evaluate the set of possible error-erasure-locating functions for different sets of erasures later on in order to get the information about possible error positions. The multiplication of a function with $\phi_0$, now corresponds to component-wise multiplication of the evaluation vectors. The number of updates in Algorithm 4.3 is still proportional to $o_1$, whereas the evaluation of functions becomes trivial. Every update is performed with vectors of length $n$ so that the computational complexity of one iteration step described by Algorithm 4.3 is in the order $O(o_1n)$.

We summarize the results obtained so far in the following theorem.

**Theorem 4.5** Let an AG code of type $C = C_\Omega(D, mP_\infty)$ be given. For a given word $y = c + e$, $c \in C$, let the syndromes $S_y(i, j)$ for $i + j \leq m$ and a reliability information on the positions in $y$ be given. We can find all tentative error-erasure-locating functions of minimal pole order for a GMD-decoding scheme, that realizes the error correction capability of the basic algorithm, with complexity $O(o_1d^2)$.

**Proof.** The theorem is a consequence of Algorithm 4.3. The functions $\sigma(i, m)$ and $\lambda(j, m)$ can be calculated with complexity $O(o_1d^2)$, which is proved in [23]. These functions are the starting point for the iterative determination of the functions $\sigma(i, m, \mathcal{R}_i)$ and $\lambda(j, m, \mathcal{R}_j)$ with Algorithm 4.3. Each of the less than $d$ iteration step of algorithm 4.3 requires complexity $O(o_1d)$, which results in an overall complexity $O(o_1d^2)$.  

Theorem 4.5 guarantees the error-erasure-locating property of functions $\sigma(i, m, \mathcal{R}_i)$ only up to the error correction capability of the basic algorithm. However, many error-erasure patterns of higher weight are located by this setup, cf. [25][24, Remark 4.6]. In order to guarantee error-erasure-location up to the full correction capability of the code, we can employ the majority voting scheme proposed by Feng and Rao in [18]. The idea of this scheme is to iteratively determine syndromes $S_e(i, j)$ with $i + j > m$. The main difficulty when applying this scheme is that the iterative determination of more
than one unknown syndrome is a nonlinear operation in the known syndromes. This implies that we can not treat the decoding attempts for nested sets of erasures in parallel by dealing with nested vector spaces. We have to treat the decoding attempts separately from now on. However, we can utilize the sets of functions $\sigma^{(i,m,R_i)}$ and $\lambda^{(j,m,R_i)}$ obtained so far. The functions $\sigma^{(i,m,R_i)}$ give recursions on the known syndromes. They also provide the possibility to determine a tentative value of a syndrome $S_n(i,j)$ with $i + j = m + 1$, that is consistent with the recursion described by $\sigma^{(i,m,R_i)}$. In this way it is possible to determine a similar tentative syndrome value for each found recursion on the known syndromes. Under the assumption that the weight of the error vector does not exceed the error correction capability of the code, Feng and Rao showed that, when counted properly, most of these tentative syndrome values are correct in the sense that they correspond to the syndrome value calculated from the unknown error vector. For details on the majority voting scheme the reader is referred to [18]. A fast algorithm that uses such a majority scheme in the context of error-erasure decoding is found in [23]. The algorithm formulated there works in an iterative fashion for a given set of erasure positions. An iteration determines, from two sets of function of type $\sigma^{(i,m,R_i)}$ and $\lambda^{(j,m,R_i)}$, the value of the syndrome $S_n(i,j)$ with $i + j = m + 1$ and two new sets of functions $\sigma^{(i,m+1,R_i)}$ and $\lambda^{(j,m+1,R_i)}$. Every iteration step requires a complexity $O(t_1d)$, [23]. For details of this algorithm the reader is referred to [23].

The question arises how many additional syndromes we have to determine until we can rely on the found recursions. To this end we assume that we iteratively could determine for any erasure set the syndromes $S_n(i,j)$ with $i + j \leq m + g$. These syndromes may be different for different sets of erasures. On the other hand knowing these syndromes is equivalent to decoding a code with a larger minimum distance. We can think of the code $C_D(D, (m + g)P_\infty)$ being decoded up to the error correction capability of the basic algorithm and we can apply Theorem 4.5, which yields that we indeed find the required tentative error-erasure-locating function of lowest pole order. We summarize the situation in the following theorem.

**Theorem 4.6** Let an AG code of type $C = C_D(D, mP_\infty)$ be given that is derived from a curve of genus $g > 0$. For a given word $y = c + e$, $c \in C$, let the syndromes $S_y(i,j)$ for $i + j \leq m$ and a reliability information on
VI. Algebraic-geometric codes

the positions in $\mathcal{V}$ be given. We can find all tentative error-erasure-locating functions of minimal pole order for a GMD-decoding scheme, that guarantees the full error correction capability of the code, with complexity in the order of $O(\alpha_1 g d^2)$.

Proof. From the proof of Theorem 4.5 it follows that we can find functions $\sigma^{(i,m,R_i)}$ and $\lambda^{(j,m,R_i)}$ with complexity $O(\alpha_1 d^2)$. The determination of syndromes $S_\mathcal{E}(a,b)$ for $m < a + b \leq m + g$ and functions $\sigma^{(i,m+g,R_i)}$ and $\lambda^{(j,m+g,R_i)}$ requires for a particular set of erasure positions $g$ iterations of the algorithm formulated in [23], where each iteration requires a complexity of order $O(\alpha_1 d^2)$ cf. [23]. We have at most $d$ sets of erasure positions $R_i$. The statement of the theorem follows.

Remark 4.9 Given a vector $\mathbf{y} = \mathbf{c} + \mathbf{e}$, $\mathbf{c} \in C_\Omega(D, m P_\infty)$ a particular erasure set $R_i$ does not always lead to an error-erasure-locating function $\sigma^{(i,m+g,R_i)}$, which locates the support of an error vector $\mathbf{e}$ such that $e_j \sigma^{(i,m+g,R_i)}(P_j)$ is zero for all $j$ and $\mathbf{y} - \mathbf{e}$ is a codeword in $C_\Omega(D, m P_\infty)$. The majority voting scheme of Feng and Rao often provides the possibility to identify such erasure sets at an early stage of the algorithm. The typical (but not general) situation is that, at a certain step of the algorithm, we will run out of votes to determine a new syndrome. If this situation occurs we can disregard of such an erasure set, which decreases the complexity of the algorithm.

Error values

In the previous section, an algorithm is given to determine, for different sets of erasure positions, a tentative error-erasure-locating function. In order to complete a GMD decoding, we need to determine the support of the corresponding error vector. We have to find a list of possible error vectors associated with the set of tentative error erase-locating functions. The approach proposed here is similar to the approach first proposed by Justesen et al. in [21] for codes from certain plane curves.

Let us for a moment assume that we know for an error vector $\mathbf{e}$ all syndromes $S_\mathcal{E}(i,0)$ for $0 \leq i < n + 2g$. The vector of syndromes is the image of a linear map from $\mathbb{F}_q^n$ to $\mathbb{F}_q^{n+2g}$ that sends $\mathbf{e}$ to $T \mathbf{e}^T$ where $T$ is the
(n + 2g) \times n matrix with entries \(T_{i,j} = \phi_i(P_j)\). The matrix \(T\) has rank \(n\) as a consequence of the fact that the vector space \(L((n + 2g - 1)P_\infty) / L((n + 2g - 1)P_\infty - D)\) has dimension \(n\). Thus the syndrome vector is in one to one correspondence with the vector \(e\). We can find the corresponding error vector with the inverse mapping \(T^{-1}\) from the set of \(n\) syndromes \(S_e(i, 0)\) such that the \(i\)th row of \(T\) is not a linear combination of proceeding rows of \(T\). The matrix describing \(T^{-1}\) is then a known \(n \times n\) matrix, and we can find the error vector \(e\) with computational complexity in the order \(O(n^2)\) in the general case. For many interesting families of curves, this complexity can be significantly decreased by using fast transform techniques, see e.g. [21] for codes on Hermitian curves.

The problem of finding the error vector \(e\) is thus reduced to finding the corresponding set of syndromes. These syndromes can be determined in an iterative fashion. To this end, we assume that we know for some number \(w > m\) all syndromes \(S_e(i, j)\) with \(i + j < w\). Given an error-erasure-locating function \(\sigma\) we show how to obtain all syndromes \(S_e(i, j)\) with \(i + j = w\). Repeating this process for increasing values of \(w\) clearly gives an iterative algorithm to determine all interesting syndromes. We need the following lemma.

**Lemma 4.5** Let for an error vector \(e\) all syndromes \(S_e(i, j)\) with \(i + j < w\) and one syndrome \(S_e(a, b)\) with \(a + b = w\) be given. Then we can find all syndromes \(S_e(i, j)\) with \(i + j = w\) with complexity in the order \(O(o_1 g)\).

**Proof.** Any syndrome \(S_e(i, j)\) with \(i + j = w\) can be expressed as a linear combination

\[
S_e(i, j) = \sum_{k=0}^{w} a_k^{(i,j)} S_e(k, 0),
\]

which is a consequence of equation (4.31) and the fact that the function \(\phi_i \phi_j\) can be expressed as

\[
\phi_i \phi_j = \sum_{k=0}^{w} a_k^{(i,j)} \phi_k.
\]  

(4.35)

The coefficients \(a_k^{(i,j)}\) are known in advance. The question arises how many of the coefficients \(a_k^{(i,j)}\) are nonzero. Due to the choice of the functions \(\phi_i\),
\(\phi_i\) contains a factor \(\phi_{i_0}^j\) such that \(i' = i - l_0o_1\) is a non-gap and \(i - (l_0 + 1)o_1\) is a gap, i.e. \(\phi_i = \phi_{i_0}^j\phi_1\). Similarly, the function \(\phi_j\) has the form \(\phi_j = \phi_{i_0}^j\phi_1\). We consider the equation

\[
\phi_i \phi_j = \sum_{k=0}^{i+j'} a_k(i', j') \phi_k. \tag{4.36}
\]

The maximal pole order of a function \(\phi_i\), which does not contain \(\phi_{i_0}\) as a factor, is \(2g - 1 + o_1\). It follows that \(\phi_i \phi_j\) is an element of the space \(L(2(2g - 1 + o_1)P_\infty)\) and equation (4.36) has at most \(l(2(2g - 1 + o_1)P_\infty) = 3g + 2o_1 - 1\) coefficients. Equation (4.35) is then obtained by multiplying (4.36) with \(\phi_{i_0}^j\).

From equation (4.33) we know that we have to determine at most \(o_1\) different syndromes \(S_e(i, j)\) with \(i + j = w\). Each determination of such a syndrome involves a linear equation with at most \(3g + 2o_1 - 1\) coefficients. It follows that the number of \(\mathbb{F}_q\)-multiplications in order to find all \(S_e(i, j)\) with \(i + j = w\) is at most \(o_1(3g + 2o_1 - 1)\) and the lemma follows. \(\square\)

**Remark 4.10** Lemma 4.5 is a rather conservative estimate on the required complexity. In many interesting cases, such as codes on Hermitian curves, this complexity is much less due to additional conditions that we can impose on the functions \(\phi_i\). In the case of codes on Hermitian curves, we could e.g. replace \(O(o_1g)\) in Lemma 4.5 with \(O(1)\), cf. [23].

Given an error-erasure-locating function

\[
\sigma = \sum_{i \geq 0} \sigma_i \phi_i,
\]

the following relation holds for all non-negative \(j\)

\[
\sum_{i \geq 0} \sigma_i S_e(j, i) = 0, \tag{4.37}
\]

which is a consequence of the fact that \(e_l \sigma(P_l)\) is equal to zero for all \(l\). Now let for some \(w\) all \(S_e(j, i)\) with \(j + i \leq w - 1\) and an error-erasure-locating function \(\sigma\) be given. Let \(\phi_l\) be the term with highest pole order in
the expansion of \( \sigma \). We rewrite equation (4.37) as

\[
S_e(j, l) = -\frac{1}{\sigma_l} \sum_{i=0}^{l-1} \sigma_i S_e(j, i).
\]

Provided \( j \) is a non-gap, in which case equation (4.38) is nontrivial, we can determine a syndrome \( S_e(j, l) \) with \( j + l = w \) from syndromes \( S_e(j, i) \) with \( j + i < w \). The number \( j \) is always a non-gap if \( w - l \) is larger than \( 2g - 2 \). This situation is always given in a GMD decoding scheme, where the minimal \( w \) equals \( m + 1 \) for the basic algorithm and \( m + g + 1 \) for an algorithm employing the majority voting scheme of Feng and Rao.

Equation (4.38) and Lemma 4.5 together yield the following Lemma.

**Lemma 4.6** Let for an error vector \( e \) the syndromes \( S_e(j, i) \) with \( j + i \leq m + g \) and an error-locating function \( \sigma \) be given. We can determine the set of syndromes \( S_e(i, 0) \) for \( 0 \leq i < n + 2g \) and the corresponding error vector \( e \) with complexity \( O(n(n + o_1g)) \).

**Proof.** Equation (4.38) and the proof of Lemma 4.5 give an iterative algorithm to determine the required syndromes. Every iteration step requires \( O(d) \) multiplications of elements in \( \mathbb{F}_q \) in order to compute equation (4.38) and \( O(o_1g) \) \( \mathbb{F}_q \)-multiplications in order to apply Lemma 4.5. We have to perform \( O(n - d) \) iterations which leads to \( O((n - d)(d + o_1g)) \) multiplications in order to calculate \( S_e(i, 0) \) for \( 0 \leq i < n + 2g \). The error vector \( e \) is found by the inverse map \( T^{-1} \), which requires \( O(n^2) \) \( \mathbb{F}_q \)-multiplications. The statement of the lemma follows.

**Remark 4.11** Lemma 4.6 allows us to find an error vector \( e \) provided that \( \sigma \) is error-locating in the sense that there exist a vector \( e \) such that \( e_l \sigma(P_l) \) equals zero for all \( l \) and \( e - y \) is a codeword in \( C(\Omega(D), mP_\infty) \). However, we can not always guarantee this error-locating property of \( \sigma \). On the other hand if \( \sigma \) is not error-locating we still can iteratively determine a set of syndromes which then points out a feasible error vector (of usually high weight). In a GMD decoding scheme context, where we have a list of tentative error-erasure-locating functions, which are not necessarily all error-locating, the above transformation procedure still leads to a list of tentative error vectors such that \( e - y \in C(\Omega(D), mP_\infty) \) for any vector \( e \) on the list.
Theorem 4.7 Let an AG code of type $C = C_{\Omega}(D, mP_\infty)$ with length $n$ and minimum distance $d$ be given. For a given word $y = c + e, c \in C$, let a reliability information on the positions in $y$ be given. A GMD decoding of the word $y$, that guarantees the full error correction capability of the code, can be performed with complexity not exceeding $O(dn(n + o_1g))$.

Proof. The first step in the proposed decoding procedure consists of calculating the syndromes $S_e(j, i)$ with $j + i \leq m$. We can determine these syndromes iteratively from equation (4.31) and Lemma 4.5 which requires a complexity $O((d + g)(n + o_1g))$. From Theorem 4.6 it follows that we then can find all tentative error-erasure-locating functions with complexity in the order $O(o_1gd^2)$. Given the set of tentative error-erasure locating functions we have to find the corresponding error vectors. From Lemma 4.6 and Remark 4.11 it follows that we can find a list of at most $\lceil d/2 \rceil$ feasible error vectors with complexity $O(dn(n + o_1g))$. Thus the complexity in every step is upper bounded by $O(dn(n + o_1g))$ and the theorem follows. \hfill \Box

An example - Hermitian Codes

Codes on Hermitian curves are one of the best investigated classes of AG codes. The curves are defined over $\mathbb{F}_{q^2}$ and can be described by the homogeneous equation [27]

$$\mathcal{H} : Y^qZ + YZ^q = X^{q+1}.$$ 

$\mathcal{H}$ is a nonsingular, plane curve with genus $g = q(q - 1)/2$. It has one point at infinity $P_\infty = (0 : 1 : 0)$ and $q^3$ other rational points over $\mathbb{F}_{q^2}$. Let $x$ and $y$ be defined as the rational functions $\frac{X}{Z}$ and $\frac{Y}{Z}$. With $D$ chosen as the sum of the $q^3$ affine points, it suffices to consider the affine equation of the curve

$$\mathcal{H} : y^q + y = x^{q+1}. \quad (4.39)$$

It is shown in [27] that the vector space $L(mp_\infty)$ has a basis

$$L(mp_\infty) = \langle x^i y^j : i \geq 0, 0 \leq j < q, qi + (q + 1)i \leq m \rangle. \quad (4.40)$$

In particular it follows that the non-gap sequence is equal to

$$0, q, q + 1, 2q, 2q + 1, 2q + 2, 3q, \ldots, (q - 1)q + q - 1, q^2, q^2 + 1, \ldots.$$
The first non gap \( o_1 \) is therefore equal to \( q \) and \( \phi_{o_1} \) is equal to \( x \). From equation (4.40) also follows that we can choose \( \phi_i \) as

\[
\phi_i = \begin{cases} 
0 & \text{if } i \text{ is a gap} \\
y(i \mod q)x^{(i-(q+1)(i \mod q))/q} & \text{otherwise.}
\end{cases}
\]

Given a vector \( y = c + e \), syndromes \( S_{c}(i, j) \) are defined by equation (4.31). For the chosen functions \( \phi_i \) the content of Lemma 4.5 is reduced to the following simple lemma, which we cite from [23].

**Lemma 4.7 (Lemma 8 from [23])** Let the curve \( \mathcal{H} \) be defined by the equation \( \mathcal{H} : X^{q+1} = ZY^q + Z^qY \). The syndromes \( S_{a,b} \) satisfy the relations

\[
S_{a,b} = \begin{cases} 
0 & (a \in G) \vee (b \in G) \\
S_{a+b,0} & (a \mod q) + (b \mod q) < q \\
S_{a+b,0} - S_{a+b-q^2+1,0} & \text{otherwise.}
\end{cases}
\]

**Proof.** This is a direct consequence of the affine equation of the curve \( \mathcal{H} \). For a full proof see [23].

We illustrate the error-localization procedure of an efficient GMD decoding procedure in the following example.

**Example 4.1** Let \( \mathcal{X} \) be the Hermitian curve

\[ \mathcal{X} : x^5 = y^4 + y \]

over \( \mathbb{F}_{16} \cong \mathbb{Z}_2[z]/(z^4 + z + 1) \). Let \( \alpha \) be a root of \( z^4 + z + 1 \). We investigate the code \( C_\Omega(D, 18P_\infty) \) with \( D \) as the sum of 64 affine \( \mathbb{F}_{16} \)-rational points \( P_1 = (x_1, y_1) \). \( C_\Omega(D, 18P_\infty) \) has dimension \( k = 51 \) and designed minimum distance \( d = 8 \). The error correction capability that is guaranteed by the basic algorithm corresponds to a distance equal to two. Let a vector \( y = c + e, c \in C_\Omega(D, 18P_\infty) \) be given such that five errors have occurred in points

<table>
<thead>
<tr>
<th>error point</th>
<th>error value</th>
<th>error point</th>
<th>error value</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\alpha^3, \alpha^3))</td>
<td>(\alpha)</td>
<td>((\alpha, \alpha^3))</td>
<td>(\alpha^4)</td>
</tr>
<tr>
<td>((0, \alpha^5))</td>
<td>(\alpha)</td>
<td>((\alpha^3, \alpha^2))</td>
<td>(\alpha).</td>
</tr>
</tbody>
</table>
VI. Algebraic-geometric codes

We can calculate syndromes $S_e(i, 0)$ according to equation (4.31) for $0 \leq i \leq 18$ as

\[
\begin{align*}
S_{0,0} &= \alpha^{14} & S_{1,0} &= 0 & S_{2,0} &= 0 & S_{3,0} &= 0 & S_{4,0} &= \alpha^5 \\
S_{5,0} &= \alpha^6 & S_{6,0} &= 0 & S_{7,0} &= 0 & S_{8,0} &= \alpha^8 & S_{9,0} &= \alpha^5 \\
S_{10,0} &= \alpha^4 & S_{11,0} &= 0 & S_{12,0} &= \alpha^{12} & S_{13,0} &= \alpha^{10} & S_{14,0} &= \alpha^{13} \\
S_{15,0} &= \alpha^5 & S_{16,0} &= \alpha^{11} & S_{17,0} &= \alpha^9 & S_{18,0} &= \alpha^{10}
\end{align*}
\]

Using Lemma 4.7, we then can calculate any syndrome $S_e(i, j)$ with $i + j \leq 18$. Thus we know all syndromes occurring in matrix $\tilde{\Gamma}$ as defined in equation (4.32). This matrix reads

\[
\begin{pmatrix}
\alpha^{14} & 0 & 0 & 0 & \alpha^5 & \alpha^6 & 0 & 0 & \alpha^8 & \alpha^5 & \alpha^4 & 0 & \alpha^{12} & \alpha^{10} & \alpha^{13} & \alpha^{11} & \alpha^9 & \alpha^{10} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\alpha^5 & 0 & 0 & 0 & \alpha^8 & \alpha^5 & 0 & 0 & \alpha^{12} & \alpha^{10} & \alpha^{13} & 0 & \alpha^{11} & \alpha^9 & \alpha^{10} & ? & ? & ? & ? \\
\alpha^6 & 0 & 0 & 0 & \alpha^5 & \alpha^1 & 0 & 0 & \alpha^{10} & \alpha^{13} & \alpha^5 & 0 & \alpha^9 & \alpha^{10} & ? & ? & ? & ? \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\alpha^8 & 0 & 0 & 0 & \alpha^{12} & \alpha^{10} & 0 & \alpha^{11} & \alpha^9 & \alpha^{10} & 0 & ? & ? & ? & ? & ? & ? & ? & ? \\
\alpha^5 & 0 & 0 & 0 & \alpha^{10} & \alpha^{13} & 0 & \alpha^9 & \alpha^{10} & ? & 0 & ? & ? & ? & ? & ? & ? & ? & ? \\
\alpha^4 & 0 & 0 & 0 & \alpha^{13} & \alpha^5 & 0 & 0 & \alpha^{10} & ? & 0 & ? & ? & ? & ? & ? & ? & ? & ? \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

We apply the BMA proposed in [23] to the matrix $\tilde{\Gamma}$ in order to obtain the functions $\sigma_{i,18}$ and $\lambda_{i,18}$. The functions are listed below.

\[
\begin{array}{c|c|c}
 i & \sigma_{i,18} & \lambda_{18,i} \\
0 & \alpha^{12} + \alpha^{13}x + \alpha^x2 + \alpha^{14}xy + y^3 & 7 \\
1 & \alpha^9 + \alpha^8x + \alpha^4y + \alpha^5x^2 + xy + \alpha^7x^3 + x^2y & 6 \\
2 & \alpha^4 + \alpha^{12}x + \alpha^{13}x^2 + \alpha^{12}xy + y^2 & 9 \\
3 & \alpha^8x + \alpha^x2 + \alpha^3xy + \alpha^5x^3 + y^3 & 4 \\
\end{array}
\]
<table>
<thead>
<tr>
<th>$i$</th>
<th>$\lambda^{(i,18)}$</th>
<th>$a^{(18,i)}_\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\alpha^2 + \alpha^9 x + y + \alpha^2 x^2$</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>$\alpha^2 x + \alpha^{10} y + \alpha^2 x^2 + \alpha^{10} xy$</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

It is straightforward to verify that the functions $\sigma^{(i,18)}$ and $\lambda^{(j,18)}$ satisfy the required properties. Starting from these functions, we have to iteratively determine functions $\sigma^{(i,18,\mathcal{R}_1)}$ and $\lambda^{(i,18,\mathcal{R}_1)}$. To this end, we assume that the point $P_{j_1} = (\alpha^5, \alpha^3)$ indicated the least reliable position in $\mathcal{Y}$ and we thus choose the first erasure position set $\mathcal{R}_1 = \{j_1\}$. Applying Algorithm 4.3, we find that the function $\xi$ equals $\lambda^{(1,18)}$, and we calculate the following tables.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\sigma^{(i,18,\mathcal{R}_1)}$</th>
<th>$a^{(18,i,\mathcal{R}<em>1)}</em>\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\alpha^6 + \alpha^6 y + \alpha^6 x + \alpha^{11} x y + \alpha^9 x^2 + \alpha^3 x^3$</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>$\alpha + \alpha^{10} y + \alpha^2 x y + \alpha^3 x^2 + \alpha^3 x^2 y + \alpha^{10} x^3$</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>$\alpha + \alpha^7 y + \alpha^3 y^2 + \alpha^7 x + \alpha^9 x y + \alpha^7 x^2$</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>$\alpha^2 + \alpha^3 y^3 + \alpha^{10} y + \alpha^{12} x + \alpha^7 x y + \alpha^{10} x^2 + \alpha^8 x^3$</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$j$</th>
<th>$\lambda^{(j,18,\mathcal{R}_1)}$</th>
<th>$a^{(18,j,\mathcal{R}<em>1)}</em>\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\alpha^3 + \alpha^4 y + \alpha^5 x + \alpha^3 x y + \alpha^4 x^2$</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>$\alpha^7 + y + \alpha x + \alpha^5 x y + \alpha^9 x^2 + \alpha^{10} x^2 y + \alpha^2 x^3$</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

We assume that the second least reliable position is indicated by $P_{j_2} = (\alpha, \alpha^9)$ and thus we have $\mathcal{R}_2 = \{j_1, j_2\}$. Because the function $\sigma^{(2,18,\mathcal{R}_1)}$ has $P_{j_2}$ as zero, we have to choose $\xi$ as $\lambda^{(0,18,\mathcal{R}_1)}$. In the following iteration of Algorithm 4.3 we find the following tables.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\sigma^{(i,18,\mathcal{R}_2)}$</th>
<th>$a^{(18,i,\mathcal{R}<em>2)}</em>\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\alpha^2 + \alpha^{12} y + \alpha^9 x + \alpha^{11} x y + x^4$</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>$\alpha^{10} + \alpha^5 y + \alpha^{14} x + \alpha^{11} x y + \alpha^7 x^2 + x^2 y + \alpha^7 x^3$</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>$\alpha^{13} + \alpha^4 y + y^2 + \alpha^4 x + \alpha^6 x y + \alpha^4 x^2$</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>$\alpha^3 y + y^3 + \alpha^{14} x + \alpha x y + \alpha x^2 + \alpha^5 x^3$</td>
<td>4</td>
</tr>
</tbody>
</table>
In a similar way such tables are found for erasure position sets $\mathcal{R}_i$ with $i$ larger than 2. Here we only consider the branches corresponding to erasure sets $\mathcal{R}_j$, $j = 0, 1, 2$. It is a remarkable fact that the function $\sigma^{(2,18,\mathcal{R}_1)}$ is error-locating, that means evaluates to zero at all five error points. This underlines the error correction capabilities of the basic algorithm. Besides $\sigma^{(2,18,\mathcal{R}_1)}$, the functions $\sigma^{(i,18,\mathcal{R}_2)}$, $i = 0, 1, 2$ are also error-locating. A subsequent application of the majority voting of Feng and Rao runs out of votes for the error-only case and yields $\sigma^{(2,18,\mathcal{R}_1)}$ as tentative error-locating function for erasure sets $\mathcal{R}_1$ and $\mathcal{R}_2$.

### VII Conclusions

The decoding scheme proposed in section III gives a general efficient procedure for GMD decoding of linear codes. Provided a sequence of error-erasure-locating pairs exists, the general procedure finds all error-erasure-locating vectors with a complexity in the order $O(n^3)$ where $n$ is the length of the code. In the case of RS codes, the procedure is performed by introducing two additional steps in a decoding procedure that corrects errors only. One of these steps concerns the iterative solution of a system of linear equations, and the other is the check for the most likely error vector, common to all GMD decoding schemes. Due to the regular structure of all linear equations involved, we have given an algorithm that accomplishes the GMD decoding problem for RS codes requiring complexity $O(nd)$, where $d$ is the minimum Hamming distance of the code. This has to be compared with earlier GMD decoding schemes that do not utilize the dependence structure between the different decoding attempts. The complexity of such schemes is given by $O(nd^2)$, due to the fact that we employ $O(d)$ decoding attempts each requiring complexity $O(nd)$. 
The complexity of the algorithm applied to algebraic geometric codes is highly dependent on the chosen curve. We consider nonsingular, absolutely irreducible, projective curves and a class of so-called *one-point* codes. We can find all tentative error-locating functions, in a setup that guarantees the error correction capability of the basic algorithm, with complexity $O(o_1 nd)$, where $o_1$ is the first non-gap of the function space associated with the code. The inclusion of the majority voting scheme of Feng and Rao [18], in order to guarantee the full GMD decoding capability of the code, can be done with complexity $O(o_1 g^2 d^2)$. However, it has been observed by Høholt and Pellikaan in [24, Remark 4.6], that the performance of the basic algorithm is close to the performance of an algorithm that guarantees the error correction capability up to half the designed minimum distance, see also [25]. This observation suggests that it very often may be sufficient to use the setup of the basic algorithm in an application.

The list of tentative error-erasure-locating functions provides some information about the error positions of the corresponding tentative error values. However, in contrast to the case of RS codes we can not determine the exact error positions from this list. This is, on the one hand, due to the fact that an error-erasure-locating functions may have more zeros than the weight of the corresponding error vector suggests, and on the other hand we are lacking an efficient Forney formula to check whether an erased position in the received word is erroneous.

For any particular set of erased positions, we can determine the error vector corresponding to an error-erasure-locating function with complexity $O(n(n + o_1 g))$. Thus, using the setup of the basic algorithm and if we decide on the most likely error vector based on the zero sets of the corresponding error-locating functions, a soft decision decoding of an AG code can be performed with complexity $O(o_1 n^2)$. In order to guarantee the full error correction capability of a GMD decoding scheme, we have to find a tentative error vector for any tentative error-erasure-locating function. Such a scheme would require complexity $O(dn(n + o_1 g))$. This compares with the complexity required by decoding attempts that do not use the dependency structure proposed in section III. Such an approach requires $O(o_1 nd^2)$ operations due to the fact that we employ $\lceil \frac{d}{2} \rceil$ decoding attempts, each requiring complexity $O(o_1 nd)$, [23].
Bibliography


Chapter 4. Fast GMD decoding of AG and RS codes


Chapter 5

A Forney type formula for the determination of error values for Hermitian codes

Abstract A generalization of Forney’s algorithm to one-point codes on Hermitian curves is derived. The proposed algorithm uses as input the set of known syndromes and an error-locating function \( \sigma \) together with the set of zeros of \( \sigma \). For an Hermitian code defined over an alphabet \( \mathbb{F}_{q^2} \), the number of required \( \mathbb{F}_{q^2} \) multiplications, in order to determine all error values, is less than \( 10(t+g)^2 \). This makes the algorithm especially well suited for high-rate codes, where the code length is large compared to the number of correctable errors.

Index Terms — Hermitian codes, error values, Forney’s formula, efficient algorithms
I Introduction

The decoding problem in coding theory can be described as finding the codeword in a code $C$ at minimum Hamming distance from a received word $y$, which is the corrupted version of a codeword in $C$. The most successful approach to the problem of decoding an algebraic geometric (AG) code stems from algebraic decoding. The strategy of algebraic decoding is to first calculate the set of erroneous positions in $y$ followed by the determination of the corresponding error values. Usually the set of error positions is indicated by the zero set of an error-locating function $\sigma$. Justesen et al. showed in [5] that an error-locating function gives a recursion satisfied by the syndromes, which allows determination of $\sigma$. Fast and efficient methods to calculate the function $\sigma$ and thus the set of possibly erroneous positions for AG codes were first given by Justesen et al. in [7]. Using results of Feng and Rao [9] the algorithm was extended by e.g. Sakata et al. in [6]. A parallel version of a similar algorithm was given by Kötter in [8]. When considering an AG code over a finite field $\mathbb{F}_q$, the complexity of such an algorithm, counted as the number of multiplications over $\mathbb{F}_q$, is in the order $O(\gamma d^2)$, where $d$ is the minimum distance of the code and $\gamma$ is an invariant of the underlying curve. The standard approach to the second step in algebraic decoding of AG codes, namely determining the error values, uses the recursion relations satisfied by $\sigma$ in order to determine a syndrome vector that completely describes the error vector. Once a sufficient set of syndromes is known we can find the error vector by a suitable linear transformation, see e.g. [7]. For low rate codes this method is very convenient. However, for high rate codes there are a couple of drawbacks. The number of syndromes that are to be determined is for high rate codes close to $n$, which makes the determination of error values often the most complex step in a decoding algorithm. Another drawback is that we always have to determine a complete set of syndromes. If few errors have occurred or if a systematic encoding procedure is used, it would be preferable to determine only the interesting error values and to disregard the others. For Reed-Solomon (RS) codes these drawbacks are overcome with the use of a formula derived by Forney in [1]. The goal of the present paper is to derive a generalization of Forney’s formula to the case of Hermitian codes. Once an error-locating function $\sigma$ together with its zero set is known, the complexity of the proposed algorithm, for Hermitian codes, is bounded above by $10(t + g)^2$, where $t$ denotes the number
II. The main idea

Let $\mathbb{F}_q$ denote the finite field with $q$ elements. A linear code $C$ of length $n$ over $\mathbb{F}_q$ is a subspace of $\mathbb{F}_q^n$. The minimum Hamming distance and the dimension of $C$ are denoted $d(C)$ and $k(C)$. A linear code may be described by an $(n - k(C)) \times n$ parity check matrix $H$ with column vectors $h_0, h_1, \ldots, h_{n-1}$ such that

$$C = \{ c \in \mathbb{F}_q^n | cH^T = 0 \}.$$

Throughout the paper the components of vectors and matrices are indexed starting from index 0. The inner product of two vectors $a$ and $b$ is denoted $(a, b) = \sum_{i=0}^{n-1} a_i b_i$, and the Hamming weight of a vector $a$ is denoted by $w(a)$. Let a vector $y = c + e$, $c \in C$ be given. The syndrome $s$ of $y$ is defined as $s = yH^T$. Clearly the syndrome $s$ only depends on the error vector $e$. Let a set $I = \{i_1, i_2, \ldots, i_{j_1}\} \subset \{0, 1, \ldots, n - 1\}$ be given such that the support of $e$ is a subset of $I$. The following simple lemma allows determination of the error values in the positions indicated by $I$.

**Lemma 5.1** Let a vector $\lambda^{(j)} \in \mathbb{F}_q^{n-k}$ be given such that

$$(\lambda^{(j)}, h_l) = \begin{cases} 0 & l \in I \setminus \{i_j\} \\ \neq 0 & l = i_j \end{cases}$$

holds. The error value in position $i_j \in I$ is given as

$$e_{i_j} = \frac{(\lambda^{(j)}, s)}{(\lambda^{(j)}, h_{i_j})}.$$

**Proof.** The lemma follows from the equation chain

$$(\lambda^{(j)}, s) = (\lambda^{(j)}, eH^T) = (\lambda^{(j)}, \sum_{i \in I} e_i h_i) = e_{i_j} (\lambda^{(j)}, h_{i_j}).$$
In the light of Lemma 5.1 efficient determination of error values comes
down to efficient calculation of vectors $\lambda^{(j)}$ for all $i_j \in I$. For a general
linear code this is achieved by solving the system of equations $eH^T = s,$
supp$(e) \subseteq I$. For structured codes, as e.g. RS and AG codes, we can
employ Lemma 5.1 in a more direct form. The result of the following ex-
ample is the well known Forney formula [1] to determine error values for
RS codes. The derivation of Forney’s formula given here reveals the main
idea behind the reasoning applied to the case of Hermitian codes in the next
section.

**Example 5.1 (RS codes)** Let $\alpha$ be a primitive element in $\mathbb{F}_q$. An RS
code $C$ of length $n = q - 1$, dimension $k = n - r$ and minimum distance
d $d = r + 1$ may be described by an $r \times n$ parity check matrix $H$ with $H_{i,j} = 
\alpha^{ij},$ $i = 0, \ldots, r - 1, j = 0, \ldots, n - 1$. Given a vector $y = c + e, e \in C,$
such that $w(e) = t$ is less than $(r + 1)/2$, the first step in algebraic decoding
is to calculate an error-locating function

$$
\sigma(x) = \prod_{i \in \text{supp}(e)} (x - \alpha^i),
$$

which points out the set of error positions $I$, cf. [2, ch.7]. From $\sigma(x)$ we
can easily obtain the vectors $\lambda^{(j)}$ as coefficient vectors of the polynomi-
als $\lambda^{(j)}(x) = \sigma(x)/(x - \alpha^{ij})$. The inner product $(\lambda^{(j)}, h_{i,j})$ then equals
$\lambda^{(j)}(\alpha^{ij})$, which in turn equals $\sigma'(\alpha^{ij})$, where $\sigma'(x)$ denotes the formal
derivative of $\sigma(x)$. We define an error-evaluator polynomial $\omega(x)$ of degree
t $-1$ with coefficients

$$
\omega_i = \sum_{j=0}^{t-1-i} s_j \sigma_{j+i+1}, \; i = 0, 1, \ldots, t - 1. \quad (5.1)
$$

From the relation $\sigma(x) = (x - \alpha^{ij})\lambda^{(j)}(x)$ it follows that $\sigma_I = \lambda^{(j)}_{I-1} - \alpha^{ij} \lambda^{(j)}_{I}$
and we can rewrite equation (5.1) as

$$
\omega_i = \sum_{l=0}^{t-1-i} s_l \lambda^{(j)}_{l+i+1} - \alpha^{ij} \sum_{l=0}^{t-1-(i+1)} s_l \lambda^{(j)}_{l+i+1}, \; i = 0, 1, \ldots, t - 1.
$$

From these equations one can easily derive the following expression

$$(\lambda^{(j)}, s) = \sum_{l=0}^{t-1} s_l \lambda^{(j)}_l = \omega_0 + \alpha^{ij}(\omega_1 + \alpha^{ij}(\omega_2 + \ldots)) = \omega(\alpha^{ij}).$$
It follows from Lemma 5.1 that the error value \( e_{ij} \) is given as
\[
e_{ij} = \frac{(\lambda^{(j)}, s_j)}{(\lambda^{(j)}, h_j)} = \frac{\omega(\alpha^{ij})}{\lambda^{(j)}(\alpha_2^i)} = \frac{\omega(\alpha^{ij})}{\sigma'(\alpha_2^i)}.
\]

III Hermitian codes

Codes on Hermitian curves are one of the most thoroughly investigated classes of AG codes. The curves are defined over \( \mathbb{F}_{q^2} \) and can be described by the equation \[ \mathcal{H} : Y^q Z + Y Z^q = X^{q+1}. \]
\( \mathcal{H} \) is a nonsingular, plane curve with genus \( g = \frac{(q^2-1)}{2} \), one point at infinity \( P_\infty = (0 : 1 : 0) \), and \( q^3 \) other points \( P_i = (x_i : y_i : 1), \) that have coordinates in \( \mathbb{F}_{q^2} \). The vector space of rational functions on \( \mathcal{H} \) that have a pole of multiplicity at most \( m \) in \( P_\infty \) is denoted \( L(mP_\infty) \). Let \( x \) and \( y \) be defined as the rational functions \( \frac{x}{y^q} \) and \( \frac{y}{x^q} \). A function \( f \in \mathbb{F}_{q^2}[x, y] \) can be interpreted as either a polynomial in the indeterminates \( x \) and \( y \) or as a rational function on \( \mathcal{H} \) in indeterminates \( X, Y, \) and \( Z \). When \( f \) is interpreted as a rational function, it will often be characterized by a divisor \( (f) \) of zeros and poles of \( f \). Functions \( x \) and \( y \) have poles of order \( q \) and \( q+1 \) only at \( P_\infty \), and thus the vector space \( L(mP_\infty) \) has a basis \[ (5.2) \]

An Hermitian code \( C(m) \) of length \( n = q^3 \) over \( \mathbb{F}_{q^2} \) may be defined as
\[ C(m) = \{ c \in \mathbb{F}_{q^2}^n : \sum_{i=0}^{n-1} f(P_i) c_i = 0 \quad \forall f \in L(mP_\infty) \}. \]

We assume that \( m \) is less than \( n \). Then \( C(m) \) has dimension at least \( n - m - 1 + g \) and minimum Hamming distance at least \( m - 2g + 2 \), see e.g. [4]. The rows of a parity check matrix \( H \) for \( C \) are obtained by evaluating monomials \( x^a y^b, \) \( a \geq 0, 0 \leq b < q, qa + (q + 1)b \leq m \) in points \( P_i \).

Given a word \( y = c + e, \) \( c \in C(m) \), we define syndromes \( S_{ab} \) as
\[ S_{ab} = \sum_{l=0}^{n-1} e_l x_l^a y_l^b. \]
Clearly, we can calculate $S_{a,b}$ for $a \geq 0, 0 \leq b < q, qa + (q+1)b \leq m$ by replacing $e_i$ with $y_i$ in (5.4). As in the case of RS codes the first step in algebraic decoding of codes from Hermitian curves is to calculate an error-locating function $\sigma \in \mathbb{F}_q[x,y]$ of least pole order at $P_\infty$ [5]. Efficient schemes to accomplish this are e.g. given in [6] and [8]. These schemes employ the majority voting scheme of Feng and Rao [9] or an equivalent scheme and determine syndromes $S_{a,b}$ for $a \geq 0, 0 \leq b < q, qa + (q+1)b \leq m + g$. We note that, for an error vector of weight $t$, $\sigma$ has at most $t + 1$ nonzero coefficients and pole order at most $t + g$, cf.[10].

After this step of the decoding algorithm we have the following data at our disposal.

- An error-locating function $\sigma \in \mathbb{F}_q[x,y]$.
- The set of possible error positions $I$ indicated as a set of zeros of a polynomial $\sigma$.
- Syndromes $S_{a,b}$ for $a \geq 0, 0 \leq b < q, qa + (q+1)b \leq m + g$.

As in the case of RS codes we are confronted with the task of determining functions $\lambda^{(j)} \in \mathbb{F}_q[x,y]$ the zero position sets of which are a superset of $I \setminus \{i_j\}$. Moreover we need this function in a suitable form i.e.

$$\lambda^{(j)}(x,y) = \sum_{(a,b) \in \mathbb{Z}_0^2, b < q} \lambda^{(j)}_{a,b} x^a y^b,$$

where $\mathbb{Z}_0$ denotes the set of non-negative integers. In contrast to the case of RS codes this is not a straightforward task for AG codes. The main difficulty is the fact that for AG codes the zero set of $\sigma$ is neither in one to one correspondence with the factors of $\sigma$ nor does it suggest any factorization of $\sigma$. In order to obtain the set of functions $\lambda^{(j)}$ starting from $\sigma$, we formulate some simple geometric properties of $\mathcal{H}$ in the following lemma.

**Lemma 5.2** Let $\mathcal{H} : Y^aZ + YZ^a = X^{a+1}$ be the Hermitian curve over $\mathbb{F}_q$ with rational points $P_i = (x_i : y_i : 1)$ and $P_\infty = (0 : 1 : 0)$. 

III. Hermitian codes

1. The tangent line $L_i : Y + y_i^q Z - x_i^q X$ at $P_i$ intersects $H$ only at $P_i$ with multiplicity $q + 1$.

2. The line $X - x_i Z$ intersects $H$ in points $P_j = (x_j : y_j : 1)$ and in $P_\infty$ with multiplicity one where $y_j$ runs through the $q$ different solutions for $y$ of the equation $y^q + y = x_i^{q+1}$.

3. The product of tangent lines at points $P_i$ with the same $x_i = \hat{x}$ is equal to

$$
\prod_{i : x_i = \hat{x}} L_i = Y^q + Y Z^{q-1} - \hat{x}^q X Z^{q-1} - \hat{x} X^q + \hat{x}^{q+1} Z^q.
$$

**Proof.** The first two statements are easily verified directly. For statement two see also [3]. In order to prove the third statement we consider the rational function $(x - x_i)^{q+1} = x^{q+1} - x_i^q x - x_i x^q + x_i^{q+1}$ which coincides on $H$ with the rational function $y^q + y - x_i^q x - x_i x^q + x_i^{q+1}$. The associated divisor of this function is

$$(q + 1) \sum_{i : x_i = \hat{x}} P_i - q(q + 1) P_\infty$$

which, as a consequence of statement one and two, is equal to the divisor of the function

$$
\prod_{i : x_i = \hat{x}} y + y_i^q - x_i^q x.
$$

This proves that the product $\prod_{i : x_i = \hat{x}} y + y_i^q - x_i^q x$ coincides on $H$ with the rational function $y^q + y - \hat{x}^q x - \hat{x} x^q + \hat{x}^{q+1}$. The function $\prod_{i : x_i = \hat{x}} y + y_i^q - x_i^q x$, considered as element of $\mathbb{F}_q[x, y]$, has degree $q$ in the indeterminate $y$ and the term $y^q$ occurs with coefficient 1. It follows that we have

$$
\prod_{i : x_i = \hat{x}} y + y_i^q - x_i^q x = y^q + y - \hat{x}^q x - \hat{x} x^q + \hat{x}^{q+1}.
$$

The third statement is just the homogeneous equivalent of this equation. □

The following lemma states the main observation for the construction of the functions $\lambda^{(j)}$. 

Lemma 5.3 Let a function $\sigma \in \mathbb{F}_{q^2}[x, y]$ be given that has a zero position set $I$ including a zero of multiplicity $l \leq q + 1$ at a point $P_{ij}, i_j \in I$. The function $\tilde{\sigma}$ constructed as

$$\tilde{\sigma} = \frac{\sigma(y^q + y - x_{ij}^q x - x_{ij} x^q + x_{ij}^{q+1})}{(y + y_{ij}^q - x_{ij}^q x)}$$

has a polynomial representation in $\mathbb{F}_{q^2}[x, y]$. Moreover, after reducing the powers of $y$ modulo the affine equation of the curve, $y^q = x^{q+1} - y$, $\tilde{\sigma}$ has the form

$$\tilde{\sigma} = (x - x_{ij})^l \lambda^{(j)}, \quad (5.5)$$

such that the zero position set of the function $\lambda^{(j)} \in \mathbb{F}_{q^2}[x, y]$ is a superset of $I \setminus \{i_j\}$ and $\lambda^{(j)}(x_{ij}, y_{ij}) \neq 0$.

Proof. By Lemma 5.2 $(y + y_{ij}^q - x_{ij}^q x)$ is a factor of $(y^q - y - x_{ij}^q x - x_{ij} x^q + x_{ij}^{q+1})$, which implies that $\tilde{\sigma}$ can be represented as polynomial in $\mathbb{F}_{q^2}[x, y]$. The divisor of zeros and poles of $\sigma$ equals $(lP_{ij} + A - (l + \deg(A))P_{\infty})$ for some effective divisor $A$. This implies that the divisor of $\tilde{\sigma}$ equals $l((x - x_{ij}) + B - (\deg(B)P_{\infty})$ for an effective divisor $B$ such that $B - A$ also is effective. From the construction of the function $\tilde{\sigma}$ we know, that $B - (\deg(B)P_{\infty})$ is the divisor of a rational function $\lambda^{(j)}$. Any function with only poles at $P_{\infty}$ has an unique representation as a polynomial in $x$ and $y$ such that the highest occurring power of $y$ is $q - 1$. Equation (5.5) follows now from the unique representation of $\tilde{\sigma}$ and $\lambda^{(j)}$. $\Box$

Lemma 5.3 implicitly outlines an algorithm to calculate the required functions $\lambda^{(j)}$. We want to estimate the complexity of this algorithm. As a complexity measure we choose the number of multiplications over $\mathbb{F}_{q^2}$, the field of definition of the code. In order to carefully count the number of $\mathbb{F}_{q^2}$-multiplications we formulate the algorithm explicitly.
Algorithm 5.1

**Input:** The error-locating function $\sigma$, together with its zero position set $I \supseteq \text{supp}(e)$.

**Construction of the $\lambda^{(j)}$: For all $i_j \in I$ perform the following steps:**

1. Calculate the polynomial
   \[
   \tilde{\sigma} = \sigma(y^q + y - x_{i_j}^q x - x_{i_j} x^q + x_{i_j}^{q+1}) = \sum_{l=0}^{2q-1} \tilde{\sigma}_l(x)y^l.
   \]

2. Calculate the polynomial
   \[
   \hat{\sigma} = \tilde{\sigma} / (y - (x_{i_j}^q x - y_{i_j}^q)) = \sum_{l=0}^{2q-2} \tilde{\sigma}_l(x)y^l,
   \]
   where polynomials $\tilde{\sigma}_l(x)$ are calculated recursively as
   \[
   \tilde{\sigma}_{l-1}(x) = \tilde{\sigma}_l(x) + (x_{i_j}^q x - y_{i_j}^q)\tilde{\sigma}_l(x) \quad (5.6)
   \]

3. Calculate the polynomial
   \[
   \hat{\sigma} = \hat{\sigma} \mod y^q - (x^{q+1} - y) = \sum_l \hat{\sigma}_l(y)x^l
   \]

4. Calculate the polynomial
   \[
   \lambda = \hat{\sigma} / (x - x_{i_j}) = \sum_l \lambda_l(y)x^l
   \]
   where polynomials $\lambda_l(y)$ are calculated recursively as
   \[
   \lambda_{l-1}(y) = \hat{\sigma}_l(y) + x_{i_j}\lambda_l(y). \quad (5.7)
   \]

5. If $\lambda(x_{i_j}, y_{i_j})$ equals zero then let $\tilde{\sigma}$ be equal to $\lambda$ and go to step 4. Otherwise let $\lambda^{(j)} = \lambda$.  

Lemma 5.4  Given the error-locating function $\sigma$ of lowest pole order together with its zero position set $I$, Algorithm 5.1 calculates the polynomials $\lambda^{(j)}$ and the numbers $\lambda^{(j)}(x_i, y_i)$ for all $i_j \in I$. The number of $\mathbb{F}_q$-multiplications required by Algorithm 5.1 is less than $(t+g)(9(t+g) - g)$.

Proof. The claim that Algorithm 5.1 calculates the functions $\lambda^{(j)}$ follows directly from Lemma 5.3. The recursive equations (5.6) and (5.7) are just a reformulation of the standard division algorithm. We only have to prove the claim concerning the number of multiplications. We estimate the number of multiplications $M_2$ required by step 1 of the algorithm for a fixed $i_j \in I$. The polynomial $\alpha(x)$ does not have more than $\min\{a+q, 2q-2\}$ nonzero coefficients so step 1 does not require more than $\min\{a+q, 2q-2\}$ multiplications. Step 2 needs a careful investigation. In particular we need to bound the degree of the polynomials $\alpha(x)$. To this end let a number $a$ be defined as $a = \lfloor (t+g)/q \rfloor$ if $t \geq g$ and as the $a$ satisfying $a(a+1) \leq 2t < (a+1)(a+2)$ if $t < g$. Note that $a$ as a function of $t$ is non-decreasing. We can thus distinguish the two cases $t < g$ and $t \geq g$ also by $a \leq q-2$ respectively $a > q-2$. The number $a$ provides a bound on the degree of polynomial $\alpha(x, 0)$. The degrees of polynomials $\alpha_i$ satisfy the inequality

$$\deg(\alpha_i) \leq a + q - l,$$

which in turn implies

$$\deg(\alpha_i) \leq a + q - l - 1.$$

In step 2 of the algorithm every polynomial $\alpha_i$, $l = 1, 2, \ldots, 2q - 2$, has to be multiplied with a factor $(x_i^a, x - y_i^a)$ and the number of multiplications can thus be estimated as

$$M_2 = \min\{a+q, 2q-2\} \sum_{l=1}^{\min\{a+q, 2q-2\}} (a + q - l - 1) + 1 = 2 \sum_{l=1}^{\min\{a+q, 2q-2\}} a + q - l$$

$$= \begin{cases} 
(a + q)(a + q - 1) & a \leq q - 2 \\
2((2q-2)(a + q) - (q - 1)(2q - 1)) & a > q - 2
\end{cases}$$

We consider the two cases separately. If $a > q - 2$ or equivalently $t \geq g$ holds, we find

$$M_2 \leq 2(2qa + q - 1 - 2a) \leq 2(2(t+g) - 1 + q - 2\lfloor (t+g)/q \rfloor) \leq 4(t+g).$$
In the case $a \leq q - 2$, we have
\[ \mathcal{M}_2 \leq a(a+1)+q^2-q+2a(q-1) \leq 4(t+g)+(2a(q-1)-2t-2g) < 4(t+g), \]
where the last inequality follows from the fact that $2t > a(a + 1)$ and the fact that $(2a(q-1) - a(a + 1) - q(q - 1))$ is negative for $q > 1$. Step 3 of the algorithm does not require any multiplications. In the light of equation (5.7) the number of multiplications required in step 4 is bounded above by the number of nonzero coefficients in the polynomial $\lambda$. $\lambda$ has pole order at most $t + g + q^2 - q - 1 = t + 3g - 1$ which implies that $\lambda$ has at most $t + 2g$ nonzero coefficients. It follows that $\mathcal{M}_4$ is less than $t + 2g$. The evaluation of $\lambda$ in a point $P_i$, required in step 5 of the algorithm, can be done with $t + 2g$ multiplications, i.e. $\mathcal{M}_5 \leq t + 2g$. In the worst case the number of multiplications is obtained if $\sigma$ has its maximal number of $t + g$ different zeros at points $P_i$ as in this case we have to perform steps 1-4 for all $t + g$ zeroes. Thus the number of multiplications is upper bounded by $(t + g)(\mathcal{M}_1 + \mathcal{M}_2 + \mathcal{M}_4 + \mathcal{M}_5) < (t + g)(9(t + g) - g)$, which proves the lemma. \( \Box \)

We summarize the main results of this paper in the following theorem.

**Theorem 5.1** Let a vector $y = e + c$, $c \in C(m)$ be given such that the Hamming weight of $e$ does not exceed half the minimum distance of the code $C(m)$. Moreover, let the error-locating function $\sigma$ of lowest pole order together with its zero position set $I \supseteq \text{supp}(e)$ and syndromes $S_{a,b}$ for $a \geq 0, 0 \leq b < q, qa + (q + 1)b \leq m + g$ be given. The error values $e_{ij}$ in positions, indicated by $I$, are given as
\[ e_{ij} = \frac{\sum_{(a,b)} \lambda_{a,b}^{(j)} S_{a,b}}{\lambda^{(j)}(x_{i,j}, y_{i,j})}, \]
where the functions $\lambda^{(j)}$ are calculated by algorithm 5.1. The number of required multiplications over $\mathbb{F}_q^2$ in order to determine the error vector $e$ is less than $10(t + g)^2$.

**Proof.** Most of the claim is a direct consequence of Lemmata 5.1 and 5.3. The function $\sigma$ does not have a zero of multiplicity larger than $q + 1$ in
any point $P_i$ as a consequence of the assumption that $\sigma$ has minimal possible pole order. Thus the coefficient vector of $\lambda^{(j)}$ satisfies the properties that are required in Lemma 5.3 for the vector $\Theta^{(j)}$. Moreover, $(\Theta^{(j)}, h_{ij})$ equals $\lambda^{(j)}(x_{ij}, y_{ij})$. Having determined the set of functions $\lambda^{(ij)}$ in Algorithm 5.3, we have to show that the given syndromes are sufficient to calculate the sum

$$\sum_{(a,b)} \lambda^{(j)}_{a,b} S_{a,b}.$$  \hspace{1cm} (5.8)

The pole order of the functions $\lambda^{(ij)}$ is less than $t + 3g$. This follows from the facts that the pole order of $\sigma$ is at most $t + g$ and the pole order of the function

$$\frac{(y^q - y - x_i^q x - x_i x^q + x_i^{q+1})}{(x_i^q x + y + y^q)(x - x_i)}$$

in $P_\infty$ is $q^2 - q - 1 = 2g - 1$. The assumption on the weight of the error vector implies $2t + 3g - 2 < m + g$, which implies $t + 3g < m + g$ for $t \geq 1$. The number of multiplications required to calculate the sum (5.8) is at most $t + 2g$ and we have to calculate this sum at most $t + g$ times. This observation together with Lemma 5.4 proves the theorem. \hfill $\square$

**Remark 5.1** In order to determine the zero set of $\sigma$, the following strategy can be used. We first determine the univariate polynomials $\sigma(x_i, y)$ for the $q^2$ possible values of $x_i \in \mathbb{F}_q$. This step requires at most $q^2t$ multiplications because $\sigma$ has at most $t + 1$ nonzero coefficients and we need not multiply the constant term in $\sigma$. Every polynomial $\sigma(x, y)$ of degree $q - 1$ has to be evaluated for $q$ values of $y_i$ which requires $q(q-1)$ multiplications. Thus the total number of multiplications required to calculate the zero set of $\sigma$ is upper bounded by a term $q^2t + q^2q(q-1) = q^2(t + 2g)$. Using fast transformation techniques this complexity can usually be significantly decreased. However, the efficiency of such techniques is dependent on the prime factorization of $q^2 - 1$.

**Remark 5.2** In [11] Leonard derives a different generalization of Forney’s formula, where he utilizes the theory of Grobner bases applied to the ideal of error-locating functions. The approach is applicable to any curve as it does not use particular geometric properties of the curve in consideration. A complexity analysis of this approach remains to be done.
IV Conclusions

We have proposed an algorithm for calculating error values for Hermitian codes. The algorithm can be interpreted as a generalization of Forney’s formula for RS codes. The algorithm calculates the desired error values from the set of known syndromes and the error-locating function $\sigma$. The complexity for calculating all error values is less than $10(t + g)^2$ in terms of multiplications over the field used for the definition of the code, where $t$ denotes the number of errors, and $g$ is the genus of the Hermitian curve used in the code construction. This complexity is for long ($n \geq 2^{15}$), high rate Hermitian codes less than the complexity of an algorithm relying on the inverse transform of a complete vector of syndromes.
Chapter 5. A Forney type formula for Hermitian codes
Bibliography


Chapter 6

Conclusions

Algebraic decoding is subject to a permanent evolution leading towards algorithms that successively better and better match the requirements of practical data transmission systems. The present thesis can be seen as part of this evolution process.

Classical algebraic coding theory is to a large extent concentrated on the decoding of BCH codes. For those codes, some very well investigated and perfected decoding algorithms are available. With exception of some outstanding codes, such as the binary Golay code, comparably little attention has been paid to decoding algorithms for other codes. In this context, the results of our work on cyclic codes have two interesting aspects. On a specific level, simple algebraic decoding algorithms are given for a number of very good cyclic codes including the quadratic residue code of length 41 and sequences of codes such as the Zetterberg and Melas codes. On the other hand, the decoding procedures derived in chapter 2 underline the important role that MDS codes play in algebraic decoding. It is just the discovery of a new type of cyclic MDS codes that allows the derivation of a large number of decoding algorithms for cyclic binary codes, which can not be treated as BCH-codes.

Although it is known that the parameters of AG codes are, in a certain range, superior to any other known code family, they face some very well developed and powerful competitors in practical applications, such as RS codes,
non-binary BCH codes, and codes from concatenated constructions. For a specific application the requirements such as e.g. available complexity, maximal delay and performance, can often be traded against each other. The choice of coding scheme is to a large extent determined by the complexity and delay of the corresponding decoding algorithm. A computationally efficient algorithm for the decoding of AG codes is known, but the algorithm is still not refined enough to be considered practical for a wide range of applications.

The results of this thesis contribute to the evolution and refinement process of decoding algorithms for AG codes in a number of ways. The range of problems addressed covers implementation aspects of a decoding algorithm as well as the development of a computationally efficient soft decoding algorithm and the determination of error values. Our main results on these topics can be phrased as follows.

- The structure of a Berlekamp-Massey type algorithm that is used for the decoding of one-point AG codes allows a simple and parallel implementation. The space requirements of such an implementation do not increase compared to serial implementations of the algorithm. The time requirements are comparable to the time requirements of a decoding algorithm for an RS code correcting $2g$ more errors, where $g$ is the genus of curve used for the construction of the AG code.

- Soft decoding of AG codes in the framework of Forney’s GMD decoding procedure can be efficiently performed using nested sequences of error-locating pairs. The main conclusion of our results is that, at least for RS, BCH and AG codes, the order of complexity for soft decoding is not necessarily greater than for hard decoding.

- Forney’s formula for the determination of error values in the case of RS codes can be generalized to Hermitian codes. For large code lengths and high rate codes, the proposed procedure improves on complexity compared to earlier schemes.

A more detailed assessment of the above results in comparison to earlier results is found in the concluding sections of chapters 3, 4 and 5.