Reed-Solomon Codes: Construction & Decoding

Samuel J. Parsons

December 2007

1 Historical Background

In 1948, Claude E. Shannon published "Mathematical Theory of Communication" [1] beginning the field of coding theory. He explained and proved the existence of error-correcting codes with a theorem now known as Shannon's theorem. Not only did he introduce the concept of error-correcting by adding extra data, but also coined the word *bit* and reinforced the concept of transmitting information digitally by sending 0's and 1's through a channel.

> "Shannon was the person who saw that the binary digit was the fundamental element in all of communication. That was really his discovery, and from it the whole communications revolution has sprung." *R. G. Gallager*

Concurrently with Shannon's research, Richard Hamming "introduced an elegant approach to single-error correction and double-error detection" [2].[1] These two discoveries led to a plethora of research and experimentation with coding theory over the following years culminating in Irving Reed and Gustave Solomon's ground-breaking results in "Polynomial codes over certain finite fields" [4], published in 1960. The polynomial codes they described are known as Reed-Solomon codes. When Reed-Solomon codes were first introduced the technology was not yet available to implement them. However, as the technology advanced the application of the theory become more and more prevalent. Reed and Solomon's research laid the groundwork for many of the error-correcting techniques that we use today.

> "When you talk about CD players and digital audio tape and now digital television, and various other digital imaging systems that are coming – all of those need Reed-Solomon codes as an integral part of the system." - *Robert McEliece, Caltech* [2]

Irving Reed is now a professor of electrical engineering at the University of Southern California and continues to research coding theory. He has published more than 300 papers and several books, and has won numerous awards. "Reed was one of the first to recognize abstract algebra as the basis for error-correcting codes." [5, p. 538] Gustave Solomon met Reed at the MIT Lincoln labs where they published their paper

[1]See [3] for a simple explanation of single- and double-error detection.

together. Later, Solomon went to the Jet Propulsion Laboratory for NASA and taught at UCLA, UC Berkley, and Caltech before passing away in 1996. Reed-Solomon codes are still being used and active research continues. In the past 10 years, there have been significant improvements in decoding algorithms for Reed-Solomon codes with results from Sudan, Guruswami, and others.

2 Introduction

The goal of coding theory is to communicate data with fidelity across interference-ridden channels. We transform the data into a code using a set of rules (encoding) and then transmit the data across some channel. Once the transmitted data is received at the destination we attempt to determine the original data (decoding). In encoding, we aim to add just the right extra data so we can correct a reasonable number of errors while not drastically increasing the amount of information that must be transmitted or the complexity of the decoding algorithm. Although the concepts we will investigate indeed are abstract, they must be coupled with a realistic understanding of the constraints involved in real-world problems.

Most simply, a code is a mapping from one set of elements to another. In coding theory we choose to map our original data to a set of codewords that are longer than the original words. This length allows for parts of the words to be lost or compromised while leaving enough of the codeword intact that we can still identify what it represents. The challenge is to add enough data so that we have this error-correcting property and not so much that the code becomes cumbersome or increasing the likelihood of error. Over the past 50 years, researchers have developed various codes and improved upon them substantially; still, there is no one-size-fits-all code. Since we are always dealing with limitations in either time or space, tradeoffs must be made. The variables of a particular situation will suggest which code is ideal or perhaps that a new one needs to be developed. We will focus on a class of linear block codes called Reed-Solomon codes. The structure of these codes can be leveraged to efficiently correct errors that occur during transmission of data.

Before looking in detail at Reed-Solomon codes we will give some fundamental definitions and theorems that apply to linear block codes and coding theory in general.

2.1 Fundamentals

Throughout this paper, we will work over a finite field, $\mathbb{F}_q$ where q is the power of a prime. The simplest field is the binary field, $\mathbb{F}_2 = \{0, 1\}$, but as we will find, larger fields are often used to define Reed-Solomon codes.

Definition 2.1. A linear block code, C, of rate k/n, is a k-dimensional subspace of $\mathbb{F}_q^n$. We call this an (n, k) code.

Since a code is a vector space, we can represent any codeword c in the code as a linear combination of the basis vectors. For example, given the basis of C:

$$\mathbf{b}_1 = \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1n} \end{bmatrix}$$

$$\mathbf{b_2} = \begin{bmatrix} b_{21} & b_{22} & b_{23} & \dots & b_{2n} \end{bmatrix}$$

$$\dots$$

$$\mathbf{b_k} = \begin{bmatrix} b_{k1} & b_{k2} & b_{k3} & \dots & b_{kn} \end{bmatrix}$$

we can write an arbitrary codeword c as:

$$\mathbf{c} = d_1 \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1n} \end{bmatrix} + d_2 \begin{bmatrix} b_{21} & b_{22} & b_{23} & \dots & b_{2n} \end{bmatrix} + \dots + d_k \begin{bmatrix} b_{k1} & b_{k2} & b_{k3} & \dots & b_{kn} \end{bmatrix}$$

with $d_i \in \mathbb{F}_q$.

We can also construct a $k \times n$ *generator matrix*, G, using the basis vectors as rows. This allows us to make the mapping from $\mathbb{F}_q^k$ to $\mathbb{F}_q^n$ relatively easily in the form of matrix multiplications.

$$G = \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1n} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2n} \\ \dots & & & & \\ b_{k1} & b_{k2} & b_{k3} & \dots & b_{kn} \end{bmatrix}$$

Recall from linear algebra that generator matrices are not unique. A generator matrix must have an independent set of k columns which we call the information set and all other generator matrices of the same code will have the same information set. [6, p.7]

Definition 2.2. If C is a linear block code, then we define $C^{\perp}$ as the dual code of C where:

$$C^{\perp} = \{v \in \mathbb{F}^n | v \bullet c = 0, \forall c \in C\}.$$

A *parity check matrix* H of a code C is an $n \times (n-k)$ matrix whose rows generate the orthogonal complement of C. The rows of H generate the null space of the generator matrix G, and H is also a generator matrix for the code $C^{\perp}$.

When it comes to decoding it is important to be able to measure by how much two vectors or words differ. *Hamming distance* is a simple method of measuring this difference.

Definition 2.3. The *Hamming distance*, or simply *distance*, between two words of the same length, $\mathbf{w_1}$ and $\mathbf{w_2}$, denoted $D(\mathbf{w_1}, \mathbf{w_2})$, is the number of places in which the components of w_1 and w_2 differ. The *weight* of a word is the *Hamming distance* between it and the zero word.

The distance between any two distinct words is non-negative, and $D(\mathbf{w_1}, \mathbf{w_2}) = 0$ if and only if $\mathbf{w_1} = \mathbf{w_2}$. Clearly, by definition $D(\mathbf{w_1}, \mathbf{w_2}) = D(\mathbf{w_2}, \mathbf{w_1})$. Also, it can be shown with induction that it satisfies the triangle inequality. Together this shows that *Hamming distance* is a metric.

The error-correcting capability of a code is determined by the minimum distance between codewords.

Definition 2.4. The *minimum distance* of a code C is the largest integer d such that $D(\mathbf{c_1}, \mathbf{c_2}) \geq d$ for all codewords $\mathbf{c_1}, \mathbf{c_2} \in C$ with $\mathbf{c_1} \neq \mathbf{c_2}$.

Let's imagine that the codewords in a code are points in a three-dimensional space and draw a sphere of radius $r = d/2$ around each point, we will have set of spheres that may touch but they will not intersect since d is the minimum distance between any two points. Now, consider that we receive a codeword that has just been transmitted. It may contain some errors. We take its value and mark its coordinate in this imaginary three-dimensional space. If less than d errors occurred in transmission, then the coordinate will still be in the sphere belonging to the correct codeword. And so, although we encountered some errors, we are still able to determine the original data that was sent. This analogy helps make it clear that codes with a greater minimum distance are more desirable since they are able to correct more errors.

The *Singleton bound* establishes a useful relationship between n, k, and the minimum distance:

Theorem 2.1. *[7, p.1] The minimum distance, d, of an (n, k) code satisfies the following inequality: $d \leq n - (k - 1)$.*

Proof. Just project all the codewords on the first $(k-1)$ coordinates. Since there are q^k different codewords, by the pigeon-hole principle at least two of them should agree on these $(k-1)$ coordinates. But these then disagree on at most the remaining $n - (k-1)$ coordinates. And hence the minimum distance of the code C is $d \leq n - (k - 1)$. $\square$

Note that we are not guaranteed the existence of a code that satisfies the Singleton bound with equality for all n and k. In fact, we have a special name for those that do.

Definition 2.5. An (n, k) code is *maximum distance separable* (MDS) if it satisfies the Singleton bound with equality: $d = n - (k - 1)$.

2.2 How to use a code for error correction

Following is a basic method for using a code C for error correction [8, pg. 2]:

1. We are given some data to encode as some vector $\mathbf{u}$ of k elements from a field $\mathbb{F}$; $\mathbf{u} \in \mathbb{F}^k$.
2. We then encode this vector $\mathbf{u}$ by mapping it to a codeword $\mathbf{v} \in \mathbb{F}^n$. This can be accomplished by multiplying by a generator matrix G. For example, $\mathbf{v} = \mathbf{u}G$.
3. We transmit the codeword $\mathbf{v}$; errors may occur during transmission.
4. We receive a codeword $\mathbf{w} = \mathbf{v} + \mathbf{e}$ where $\mathbf{e} \in \mathbb{F}^n$ contains the errors that occurred during transmission.
5. We attempt to recover $\mathbf{v}$ given $\mathbf{w}$ by finding the closest codeword $\mathbf{c} \in \mathbb{F}^n$ to the vector $\mathbf{w}$.
6. The original data is determined by finding the $\mathbf{d} \in \mathbb{F}^k$ such that $\mathbf{c} = \mathbf{d}G$.

Finding the closest word often turns out to be non-trivial and so most improvements and enhancements are accomplished by making changes in how step 5 is computed.

3 Reed-Solomon Codes

Reed-Solomón codes are ideal for situations where the majority of errors occur in bursts as opposed to being uniformly distributed; this is a common attribute in various channels.

For any parameters $n, k,$ and $d = n - k + 1$ with $(1 \leq k \leq n)$ and a finite field $\mathbb{F}_q$, there exists a (n, k, d) Reed-Solomon (RS) code over $\mathbb{F}_q$ so long as $n \leq q + 1$. [9, p. 103]. Since $n \leq q + 1$, codes over the binary field $\mathbb{F}_2$ are limited to length three and are not particularly interesting, so we usually consider non-binary RS codes over the field $\mathbb{F}_q$ where q can be quite large. [9, p. 103]. In channels where it is convenient to transmit binary m-tuples, we let $q = 2^m$, with $\mathbb{F}_q$ forming an extension field of $\mathbb{F}_2$.

RS codes form a large class of MDS codes; also for any (n, k) MDS code over some field $\mathbb{F}_q$ there exists an extended (or doubly-extended) RS code with the same parameters. [9, p. 103].

Reed-Solomon codes can be represented in several different ways. We will present them as evaluation codes which will be useful for the purpose of this paper, but they can also be represented as duals of evaluation codes and as cyclic codes. The other presentations are useful for other decoding and encoding systems.

3.1 RS codes as evaluation codes

A RS code is a mapping from words which are k-tuples in the field $\mathbb{F}_q^k$ to codewords which are n-tuples in the field $\mathbb{F}_q^n$. The standard (or punctured) code is found by setting $n = q - 1$. Codes with $n = q$, and $n = q + 1$ are called extended, and doubly-extended RS codes, respectively. Using a $k \times n$ generator matrix like the following is one method of performing the mapping from $\mathbb{F}_q^k$ to $\mathbb{F}_q^n$.

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & \dots & \dots & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \dots & \dots & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \dots & \dots & \dots & \alpha^{n-2} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & \alpha^{k-1} & \alpha^{2k-2} & \alpha^{3k-3} & \dots & \dots & \dots & \alpha^{n-k+1} \end{bmatrix} \tag{3.1}$$

where α is a primitive element in the field $\mathbb{F}_q$. [8, p.3] A primitive element in a field is a generator for non-zero field elements – the field $\mathbb{F}_q^*$. The benefit of using this particular generator matrix is that we can easily compute the matrix multiplication by means of simple polynomial evaluation at the field elements. By noticing that we may obtain the ith row of G by evaluating the polynomial x^i at the field elements α_j, we arrive at a succinct description of the codewords independent of the generator matrix:

$$RS[n = q - 1, k, d] = \{(f(\alpha^0), f(\alpha^1), f(\alpha^2), ..., f(\alpha^{n-1})) : f \in \mathbb{F}_q[z] \text{ and } \deg f(z) < k\}$$

There is a unique polynomial $f(z)$ corresponding to each element of $\mathbb{F}_q^k$, but the field elements at which we evaluate the polynomials will remain fixed. This is summarized in the following theorem.

Theorem 3.1. *[9, p.104] The q^k n-tuples generated by the mappings $f(z) \mapsto \{f(\alpha^j), 0 \leq j < n\}$ form a linear ($n = q$-1, k, d= n-k+1) MDS code over $\mathbb{F}_q$, where α is a primitive element in $\mathbb{F}_q$ and the polynomials $f(z) \in \mathbb{F}_q[z]$ have degree less than k.*

Proof. The code is linear because the sum of the codewords corresponding to two polynomials $f_1(z)$ and $f_2(z)$ is the codeword corresponding to the polynomial $f_1(z) + f_2(z)$, and the multiple of the codeword corresponding to $f(z)$ by $\beta \in \mathbb{F}_q$ is the codeword corresponding to the polynomial $\beta f(z)$.

A codeword has a zero symbol in the coordinate corresponding to β_i if and only if $f(\beta_i) = 0$; ie, if and only if β_i is a solution of the equation $f(z) = 0$. By the fundamental theorem of algebra, if $f(z) \neq 0$, then since $\deg f(z) \leq k-1$, this equation can have at most $k-1$ roots in $\mathbb{F}_q$. therefore a nonzero codeword can have at most $k-1$ symbols equal to zero, so its weight is at least $n-k+1$. Since the code is linear, this implies that its minimum distance is at least $d \geq n-k+1$. But by the Singleton bound, $d \leq n-k+1$; thus $d = n-k+1$. □

Example. Let's look at a standard (punctured) RS code where $n = q-1$. Let $q = 4$, then $n = 3$, and we'll let $k = 2$. Since RS codes are MDS, $d = n-k+1 = 2$, and so we have a $(3, 2, 2)$ RS code. We will be working over the field $\mathbb{F}_4$. Recall that addition and multiplication in this field can be defined by the following tables.

$+$	0	1	α	α^2
0	0	1	α	α^2
1	1	0	α^2	α
α	α	α^2	0	1
α^2	α^2	α	1	0

$*$	0	1	α	α^2
0	0	0	0	0
1	0	1	α	α^2
α	0	α	α^2	1
α^2	0	α^2	1	α

Taking vectors of length $k = 2$, the following mapping yields codewords of length n.

$$RS[3, 2, 2] : f(z) = f_0 + f_1 z \mapsto (f(\alpha^0), f(\alpha^1), f(\alpha^2))$$

where $\alpha \in \mathbb{F}_4$ is primitive, $f(z) \in \mathbb{F}_4[z]$ and $\deg f(z) < 2$. The following is equivalent:

$$(f_0, f_1) \mapsto (f_0 + f_1\alpha^0, f_0 + f_1\alpha^1, f_0 + f_1\alpha^2)$$

Using the two generators $g_0 = (1, 1, 1)$ and $g_1 = (1, \alpha, \alpha^2)$, which are a basis for the vector space we wish to create, we can build the following generator matrix.

$$G = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \end{bmatrix}$$

This code has 16 codewords since each of f_0 and f_1 can range over the four elements of $\mathbb{F}_4$. By multiplying the possible input vectors by the generator matrix we arrive at the following coding map.

$$\begin{array}{llcllllllcllll}
(0, & 0) & \mapsto & (0, & 0, & 0) & \qquad & (\alpha, & 0) & \mapsto & (\alpha, & \alpha, & \alpha) \\
(0, & 1) & \mapsto & (1, & \alpha, & \alpha^2) & & (\alpha, & 1) & \mapsto & (\alpha^2, & 0, & 1) \\
(0, & \alpha) & \mapsto & (\alpha, & \alpha^2, & 1) & & (\alpha, & \alpha) & \mapsto & (0, & 1, & \alpha^2) \\
(0, & \alpha^2) & \mapsto & (\alpha^2, & 1, & \alpha) & & (\alpha, & \alpha^2) & \mapsto & (1, & \alpha^2, & 0) \\
(1, & 0) & \mapsto & (1, & 1, & 1) & & (\alpha^2, & 0) & \mapsto & (\alpha^2, & \alpha^2, & \alpha^2) \\
(1, & 1) & \mapsto & (0, & \alpha^2, & \alpha) & & (\alpha^2, & 1) & \mapsto & (\alpha, & 1, & 0) \\
(1, & \alpha) & \mapsto & (\alpha^2, & \alpha, & 0) & & (\alpha^2, & \alpha) & \mapsto & (1, & 0, & \alpha) \\
(1, & \alpha^2) & \mapsto & (\alpha, & 0, & \alpha^2) & & (\alpha^2, & \alpha^2) & \mapsto & (0, & \alpha, & 1)
\end{array}$$

This map tells us how to map every input vector into a codeword which we can transmit.

4 The Guruswami-Sudan Algorithm

The decoding algorithm that initially brought popularity to the RS codes was the Berlekamp-Massey algorithm which is algebraic in approach and is probably the most famous of those available. Since then many others have been developed. We will look in depth at the Guruswami-Sudan (GS) decoding algorithm devised initially by Sudan [10] and then further developed in collaboration with Guruswami [11]. The GS algorithm is particularly interesting because it can decode RS codes often significantly beyond the conventional error-correcting bounds.

Typically, given an RS code with minimum distance d, we are guaranteed to successfully decode up to $t_0 = \lfloor (d-1)/2 \rfloor$ errors. Recall the previous analogy of the spheres. If we have a minimum distance d, then the centers of these spheres are at least d apart, and we can safely use $t_0 = \lfloor (d-1)/2 \rfloor$ as the radius of the spheres. However, some spheres could have a larger radius and still not intersect other spheres. What is the probability that by increasing the radius of the sphere we would decode words incorrectly? Sudan and Guruswami [11] found that in many situations this probability can be very small and thus gives us the ability to correct up to $t_{GS} = n - 1 - \lfloor \sqrt{n(k-1)} \rfloor$ errors without making the probability of incorrectly decoding prohibitively large [11, p.12].

4.1 Algorithm Sketch

Assume that we have transmitted a codeword $\mathbf{c} = (f(\alpha^0,), f(\alpha^1), ..., f(\alpha^{n-1}))$ and we received the word $\mathbf{r} = (\beta_0, \beta_1, ..., \beta_{n-1})$. The original codeword was found by evaluating some polynomial $f(z)$ of degree less than k at the n non-zero field elements, $\alpha^0, \alpha^1, ...\alpha^{n-1}$. By definition the coefficients in the polynomial $f(z)$ correspond directly to the components of the original vector in $\mathbb{F}_q^k$ that we wish to encode. Given $\mathbf{r}$, the algorithm indicates that we can find a polynomial $p(x)$, so that the following holds.

$$|\{i : p(\alpha^i) \neq \beta_i\}| \leq t_m,$$

where t_m is the number of errors that we can decode, and $t_m \geq t_0 = \lfloor d/2 \rfloor$. The GS decoder has a variable m that specifies the multiplicity to be used in the interpolation step. The number of errors that can be corrected, t_m, corresponds with this variable. In [12] McEliece identifies the two main parts of the algorithm:

1. The interpolation step: Given our received word, $\mathbf{r} = (\beta_0, \beta_1, ..., \beta_{n-1})$, we construct a two-variable polynomial

$$Q(x,y) = \sum_{i,j} a_{i,j} x^i y^j$$

so that Q has a zero of multiplicity m at each point (α^i, β_i) and the $(1, k-1)$ weighted degree (which we will define) of $Q(x,y)$ is as small as possible.

2. The factorization step: We then find all factors of $Q(x,y)$ of the form $y - p(x)$, where $p(x)$ is a polynomial of degree $k-1$ or less. The list of all these factors is

$$\mathcal{L} = \{p_1(x), ..., p_L(x)\}.$$

$\mathcal{L}$ is a set of polynomials each corresponding with a vector or codeword in the $\mathbb{F}_q^n$; there are three types:

   (a) The transmitted, or causal, codeword

   (b) Plausible codewords which are within Hamming distance t_m from $\mathbf{r}$

   (c) Implausible codewords – those at a distance $> t_m$ from $\mathbf{r}$.

If less than t_m errors have occurred, then the list $\mathcal{L}$ will include the original word. If there is only one codeword (or polynomial) in $\mathcal{L}$, we decode the received word as that. If there is more than one codeword in $\mathcal{L}$, then we must decide which one to decode as. We will now fill in some background theory before stating the algorithm more completely and substantiating these claims.

4.2 Some Theory of Two-Variable Polynomials

4.2.1 Monomial Orderings

If we have a two-variable function $Q(x,y) \in \mathbb{F}[x,y]$, we can write

$$Q(x,y) = \sum_{i,j \geq 0} a_{i,j} x^i y^j. \tag{4.1}$$

It is clear that $Q(x,y)$ is a sum of monomials and is two dimensional. As part of the algorithm we have to build a two-variable polynomial and we have to solve a system of constraints for the coefficients of the polynomial. It is helpful to have a one-dimensional ordering on the monomials in order to solve this system. Also, the notion of monomial orderings will come up throughout our discussion. We denote the set of all monomials of two variables as:

$$\mathbb{M}[x,y] = \{x^i y^j : i,j \geq 0\} \tag{4.2}$$

The set $\mathbb{M}[x,y]$ is isomorphic to $\mathbb{N}^2$ under the bijection $(x^iy^j) \mapsto (i,j)$.

Definition 4.1. [12, 3-1] The following properties hold for a monomial ordering '<' over a set $\mathbb{M}$, with $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{M}$.[2]

1. If $a_1 \leq b_1$ and $a_2 \leq b_2$, then $(a_1, a_2) \leq (b_1, b_2)$, where $\mathbf{a} = (a_1, a_2)$ and $\mathbf{b} = (b_1, b_2)$
2. If $\mathbf{a}$, $\mathbf{b}$ are distinct, either $\mathbf{a} < \mathbf{b}$ or $\mathbf{b} < \mathbf{a}$ (total ordering).
3. If $\mathbf{a} \leq \mathbf{b}$, then $\mathbf{a} + \mathbf{c} \leq \mathbf{b} + \mathbf{c}$

We will use a type of monomial ordering called weighted degree (WD) orderings. To construct a two-variable WD ordering we need a vector with two non-negative integers, $\mathbf{w} = (u, v)$, at least one of which must be greater that zero.

Definition 4.2. [12, 3-1] For a fixed $\mathbf{w} = (u, v)$, the $\mathbf{w}$-degree of the monomial x^iy^j is:

$$\deg_w x^iy^j = ui + vj$$

The following example makes it clear that if we were to use the standard $\mathbf{w}$-degree on $\mathbb{M}[x,y]$ we would only achieve a partial ordering with distinct elements having the same order.

Example. Let's look at the $\mathbf{w}$-degree of some polynomials for $\mathbf{w} = (1,3)$ and $\mathbf{w} = (0,1)$ respectively.

monomial	$deg_{1,3}$	$deg_{0,1}$
x	1	0
x^2	2	0
x^3	3	0
y	3	1
xy	4	1
x^4	4	0
x^2y	5	1
x^5	5	0
x^6	6	0
x^3y	6	1
y^2	6	2

We can see that $\deg_{1,3} x^3 = \deg_{1,3} y$ and $\deg_{0,1} x = \deg_{0,1} x^2$ so these orderings are not monomial orderings since they are only partial. There are two ways that we can modify $\mathbf{w}$-degree to satisfy the requirement that it be a total ordering: $\mathbf{w}$-lexicographic ($\mathbf{w}$-lex) order, and $\mathbf{w}$-reverse lexicographic ($\mathbf{w}$-revlex) order.

[2]Assume that the dimension of $\mathbb{M}$ is 2. The general case is analogous.

Definition 4.3. [12, 3-2] **w**-lex order is defined as follows.

$$x^{i_1}y^{j_1} < x^{i_2}y^{j_2}$$

if either $ui_1 + vj_1 < ui_2 + vj_2$, or $ui_1 + vj_1 = ui_2 + vj_2$ and $i_1 < i_2$. **w**-revlex order is similar, except that the rule for breaking ties is $i_1 > i_2$. (In the special case $\mathbf{w} = (1, 1)$, these orderings are called graded-lex, or grlex, and reverse graded-lex, or grevlex, respectively.)

Example. In the previous example, with $\mathbf{w} = (1, 3)$, the **w**-lex ordering is

$$x < x^2 < y < x^3 < xy < x^4 < x^2y < x^5 < y^2 < x^3 < x^6...$$

and the **w**-revlex ordering is

$$x < x^2 < x^3 < y < x^4 < xy < x^5 < x^2y < x^6 < x^3y < y^2 < ...$$

With $\mathbf{w} = (0, 1)$ (referred to as a *y-ordering*), the **w**-lex ordering is

$$x < x^2 < ... < y < xy < x^2y < ... < y^2 < xy^2 < x^2y^2 < ...$$

and the **w**-revlex ordering is

$$... < x^2 < x < ... < x^3y < x^2y < xy < y < ... < x^3y^2 < x^2y^2 < xy^2 < y^2 < ...$$

Be defining a monomial ordering $1 = \phi_0(x, y) < \phi_1(x, y) < ...$, where each ϕ_k corresponds to a unique monomial, a two-variable polynomial can be defined in terms of it. For example:

$$Q(x, y) = \sum_{j=0}^{J} a_j \phi_j(x, y)$$

Under this ordering, the leading monomial (LM), the one of highest degree, is $LM(Q) = \phi_J(x, y)$. In this context $\deg Q(x, y) = \deg LM(Q) = \deg \phi_J(x, y)$, and therefore *rank* of $Q(x, y)$ is J.

Definition 4.4. The index of some monomial $\phi = x^i y^j$, denoted Ind(ϕ), in the ordering $\phi_0 < \phi_1 < ...$ is the integer k such that $\phi_k = x^i y^j$.

Example. Let's look at the index of the first 20 monomials of the **w**-revlex order for $\mathbf{w} = (1, 3)$.

(i,j)	monomial	$\deg_{1,3}$	$\operatorname{Ind}(x^iy^j)$	(i,j)	monomial	$\deg_{1,3}$	$\operatorname{Ind}(x^iy^j)$
$(1,0)$	x	1	1	$(0,2)$	y^2	6	11
$(2,0)$	x^2	2	2	$(7,0)$	x^7	7	12
$(3,0)$	x^3	3	3	$(4,1)$	x^4y	7	13
$(0,1)$	y	3	4	$(1,2)$	xy^2	7	14
$(4,0)$	x^4	4	5	$(8,0)$	x^8	8	15
$(1,1)$	xy	4	6	$(5,1)$	x^5y	8	16
$(5,0)$	x^5	5	7	$(2,2)$	x^2y^2	8	17
$(2,1)$	x^2y	5	8	$(9,0)$	x^9	9	18
$(6,0)$	x^6	6	9	$(6,1)$	x^6y	9	19
$(3,1)$	x^3y	6	10	$(3,2)$	x^3y^2	9	20

In this case: $\operatorname{Ind}(x^5y) = 16$ and $\phi_{16} = x^5y$. Also $\operatorname{Ind}(x^3y^2) = 20$ and $\phi_{20} = x^3y^2$; and $\operatorname{Ind}(x^2y) = 8$ and $\phi_8 = x^2y$.

It turns out that in the $(1,v)$-revlex ordering the numbers $\operatorname{Ind}(x^K)$ and $\operatorname{Ind}(y^L)$ are important, so we introduce the following notation[3].

$$A(K,v) = \operatorname{Ind}(x^K) \tag{4.3}$$

$$B(L,v) = \operatorname{Ind}(y^L) \tag{4.4}$$

where we use an underlying $(1,v)$-revlex order. Since x^K is the first monomial of $(1,v)$-degree K, and y^L is the last monomial of $(1,v)$-degree vL, the following also hold.

$$A(K,v) = |\{(i,j) : i+vj < K\}| \tag{4.5}$$

$$B(L,v) = |\{(i,j) : i+vj \leq Lv\}| - 1. \tag{4.6}$$

Theorem 4.1. *[12, 3-5] For $K \geq 0$, let $r = K \mod v$. Then*

$$A(K,v) = \frac{K^2}{2v} + \frac{K}{2} + \frac{r(v-r)}{2v} \tag{4.7}$$

$$B(L,v) = \frac{vL^2}{2} + \frac{(v+2)L}{2} \tag{4.8}$$

Proof. The equality (4.8) can be proved by induction. With $L = 1$,

$$B(1,v) = |\{(i,j) : i+vj \leq v\}| - 1 = (v+2) - 1 = v+1. \frac{v}{2} + \frac{v+2}{2} = \frac{1}{2}(2v+2) = v+1$$

We are given the induction hypothesis,

$$B(L-1,v) = \frac{v(L-1)^2}{2} + \frac{(v+2)(L-1)}{2}$$

[3]This notation is due to [12]

and we can write

$$B(L,v) = B(L-1,v) + |\{(i,j) : (L-1)v + 1 \le i + vj \le Lv\}|.$$

A little algebra yields the desired result.

$$B(L,v) = \frac{v(L-1)^2}{2} + \frac{(v+2)(L-1)}{2} + vL + 1.$$

$$= \frac{vL^2 + 2vL + v}{2} + \frac{(v+2)(L-1)}{2} + \frac{2vL}{2} + \frac{2}{2}$$

$$= \frac{vL^2}{2} + \frac{(v+2)(L-1)}{2} + \frac{v}{2} + \frac{2}{2} - \frac{2vL}{2} + \frac{2vL}{2}$$

$$= \frac{vL^2}{2} + \frac{(v+2)L}{2}.$$

The validity of (4.7) follows from (4.5): for each j such that $vj < K$, i must be in the range $0 \le i < K - vj$, so that

$$A(K,v) = \sum_{j=0}^{\lfloor K/v \rfloor} (K - jv)$$

$$= v \sum_{j=0}^{\lfloor T \rfloor} (T - j), \text{ where } T = K/v.$$

Now we apply Euler's summation formula, which implies:

$$\sum_{j=0}^{\lfloor T \rfloor} (T - j) = \int_0^T (T - x)dx + \frac{T}{2} - \int_0^T \{x - \frac{1}{2}\}dx$$

$$= \frac{T^2}{2} + \frac{T}{2} + \frac{\{T\}(1 - \{T\})}{2} \tag{4.9}$$

where $\{x\} = x - \lfloor x \rfloor$ is the fractional part of x. Eq (4.9) is equivalent to (4.7), since $\{T\} = r/v$. □

4.2.2 Degree and Rank

As noted earlier, we can write a two-variable polynomial in terms of a monomial ordering, for example, $Q(x,y) = \sum_{j=0}^{J} a_j \phi_j(x,y)$, where the coefficients $a_0, a_1, \ldots$ are from some field $\mathbb{F}$. If we wish to find the $(1,v)$-degree and $(0,1)$-degree or y-degree of $Q(x,y)$, we recall that the degree of a polynomial is simply the degree of the leading monomial in the corresponding ordering. We use the following notation to refer to this. [12, 3-4]

$$D(u,v;J) = \max\{\deg_{u,v} \phi_j(x,y) : j = 0, \ldots, J\} \tag{4.10}$$

With this notation we can succinctly state the upper bounds for the $(1,v)$-degree and y-degree of $Q(x,y)$ as

$$\deg_{1,v} Q(x,y) \le D(1,v;J) \tag{4.11}$$

$$\deg_{0,1} Q(x,y) \le D(0,1;J) \tag{4.12}$$

For an increasing sequence of integers $A = \{0 = a_0 < a_1 < a_2 < ...\}$ and a real number $x \geq 0$, we may want to find the greatest integer in the sequence A that is less than or equal to the number x. We refer to the index of this integer as the *rank of apparition*[4] of x with respect to A, denoted as $r_A(x) = K$ such that $a_k \leq x < a_{k+1}$. Alternately:

$$r_A(x) = \max\{K : a_K \leq x\} = \min\{L : x < a_{L+1}\} \tag{4.13}$$

Theorem 4.2. *[12, 3-4] With v fixed, define sequences $\{a_K = A(K, v)\}$ and $\{b_L = B(L, v)\}$. Then*

$$D(1, v : J) = r_A(J) \tag{4.14}$$

$$D(0, 1 : J) = r_B(J) \tag{4.15}$$

Proof. This is just a matter of observing the fact that x^K is the first monomial of $(1, v)$-degree K and that y^L is the first monomial of $(0, 1)$-degree L. □

4.2.3 Bounds

We will now find some bounds for $A(K, v)$ and $r_A(x)$.

Theorem 4.3. *[12, 3-5] For $v \geq 1$, $K \geq 0$,*

$$\frac{K^2}{2v} \leq A(K, v) \leq \frac{(K + v/2)^2}{2v}. \tag{4.16}$$

Proof. First note that $A(K, v) = \frac{K^2}{2v} + \frac{K}{2} + \frac{r(v-r)}{2v}$ by result (4.7), and so the first inequality is true if $\frac{K}{2} + \frac{r(v-r)}{2v} \geq 0$ which is true for all $K \geq 0$ and $v \geq 1$. Moreover a strict inequality holds for $K > 0$.

Since $A(K, v) = \frac{K^2}{2v} + \frac{K}{2} + \frac{r(v-r)}{2v}$ and $\frac{(K+v/2)^2}{2v} = \frac{K^2}{2v} + \frac{K}{2} + \frac{v^2/4}{2v}$, the second inequality is true if $\frac{r(v-r)}{2v} \leq \frac{v^2/4}{2v}$. Since $\frac{r(v-r)}{2v}$ reaches it's max when $r = v/2$, $\frac{r(v-r)}{2v} \leq \frac{(v/2)(v-v/2)}{2v} = \frac{v^2/4}{2v}$.

□

Theorem 4.4. *[12, 3-6] For $v \geq 1$, $J \geq 0$,*

$$\lfloor \sqrt{2vJ} - \frac{v}{2} \rfloor \leq r_A(J) \leq \lfloor \sqrt{2vJ} \rfloor - 1$$

Proof. This follows from combining the result (4.16) from Theorem 4.3 with the result (4.18) in Lemma 4.6 to follow. □

Theorem 4.5. *[12, 3-6] For $v \geq 1$, $J \geq 0$*

$$r_B(J) = \left\lfloor \sqrt{\frac{2J}{v} + \left(\frac{v+2}{2v}\right)^2} - \left(\frac{v+2}{2v}\right) \right\rfloor$$

[4] McEliece reports that this name was coined by Basil Gordon of UCLA.

and so

$$\left\lfloor \sqrt{\frac{2J}{v}} - \frac{v+2}{2v} \right\rfloor \leq r_B(J) \leq \left\lfloor \sqrt{\frac{2J}{v}} \right\rfloor.$$

Proof. These facts follow from combining the result (4.8) with (4.17) in Lemma 4.6 to follow. □

Lemma 4.6. *[12, 3-6]*

If $a_k = f(K)$ where $f(x)$ is a continuous increasing function of $x > 0$, then

$$r_A(x) = \lfloor f^{-1}(x) \rfloor. \tag{4.17}$$

More generally, if $g(K) \leq a_K \leq f(K)$, where $f(x)$ and $g(x)$ are both continuous, increasing functions of $x > 0$, then

$$\lfloor f^{-1}(x) \rfloor \leq r_A(x) \leq \lfloor g^{-1}(x) \rfloor. \tag{4.18}$$

Proof. Suppose $K = r_A(x)$. Then by definition,

$$g(K) \leq a_K \leq x < a_{K+1} \leq f(K+1).$$

Thus $K \leq g^{-1}(x)$ and $f^{-1}(x) < K+1$, ie. $r_A(x) \leq g^{-1}(x)$ and $f^{-1}(x) < r_A(x) + 1$, ie,

$$f^{-1}(x) - 1 < r_A(x) \leq g^{-1}(x).$$

The desired result (4.18) follows immediately, if we recall that $r_A(x)$ is an integer. □

4.3 Multiplicity of zeros in two-variable functions.

As you might guess, it's somewhat difficult to visualize the graph of a polynomial over a finite field in a traditional 3D diagram with Cartesian coordinates and so we must be satisfied with a rather abstract understanding. A two-variable polynomial $Q(x,y)$ has a zero at a point (α,β) if $Q(\alpha,\beta) = 0$. Guruswami and Sudan found that if we require multiple zeros, or singularities as they call them [11, p. 5], at certain points when interpolating our polynomial, we can correct more errors with the same information. However, increases in multiplicity correspond with an increase in the time it takes to process the resulting polynomial.

Definition 4.5. [12, 4-1] We say that $Q(x,y) = \sum_{i,j} a_{i,j} x^i y^j \in F[x,y]$ has a zero of multiplicity, or order, m at $(0,0)$, and write

$$ord(Q:0,0) = m, \tag{4.19}$$

if $Q(x,y)$ involves no term of total degree less than m, or $a_{i,j} = 0$ if $i+j < m$. Similarly, we say that $Q(x,y)$ has a zero of order m at (α,β), and write

$$ord(Q:\alpha,\beta) = m \tag{4.20}$$

if $Q(x+\alpha, y+\beta)$ has a zero of order m at $(0,0)$.

If we were to have a single-variable polynomial, we could quite easily determine the multiplicity at a given point. However, when we move to two-variables this becomes more challenging. One method of discovering the multiplicity of zeros at a given point is to use Hasse derivatives.

Theorem 4.7. *[12, 4-1] Let* $Q(x,y) = \sum_{i,j} a_{i,j} x^i y^j \in F[x,y]$. *For any* $(\alpha,\beta) \in F^2$, *we have*

$$Q(x+\alpha, y+\beta) = \sum_{r,s} Q_{r,s}(\alpha,\beta) x^r y^s \tag{4.21}$$

where

$$Q_{r,s}(x,y) = \sum_{i,j} \binom{i}{r}\binom{j}{s} a_{i,j} x^{i-r} y^{j-s} \ \forall\ r,s\ s.t. 0 \le r+s < m \tag{4.22}$$

which is called the (r,s)th Hasse (mixed partial) derivative of $Q(x,y)$. *Note that:*

$$Q_{r,s}(\alpha,\beta) = Coeff_{x^r,y^s} Q(x+\alpha, y+\beta). \tag{4.23}$$

Proof. By the binomial theorem we have

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

If $Q(x,y) = \sum_{i,j} a_{i,j} x^i y^j$, then

$$\begin{aligned} Q(x+\alpha, y+\beta) &= \sum_{i,j} a_{i,j} (x+\alpha)^i (y+\beta)^j \\ &= \sum_{i,j} a_{i,j} \left[\sum_{r=0}^{i} \binom{i}{r} x^r \alpha^{i-r}\right] \left[\sum_{s=0}^{j} \binom{j}{s} y^s \beta^{j-s}\right] \\ &= \sum_{i,j} \sum_{r=0}^{i} \sum_{s=0}^{j} x^r y^s \binom{i}{r}\binom{j}{s} a_{i,j} \alpha^{i-r} \beta^{j-s} \\ &= \sum_{r,s} x^r y^s \sum_{i,j} \binom{i}{r}\binom{j}{s} a_{i,j} \alpha^{i-r} \beta^{j-s} \\ &= \sum_{r,s} Q_{r,s}(\alpha,\beta) x^r y^s. \end{aligned}$$

□

Theorem 4.8. *[12, 4-3] We also have*

$$Q(x,y) = \sum_{r,s} Q_{r,s}(\alpha,\beta)(x-\alpha)^r (y-\beta)^s$$

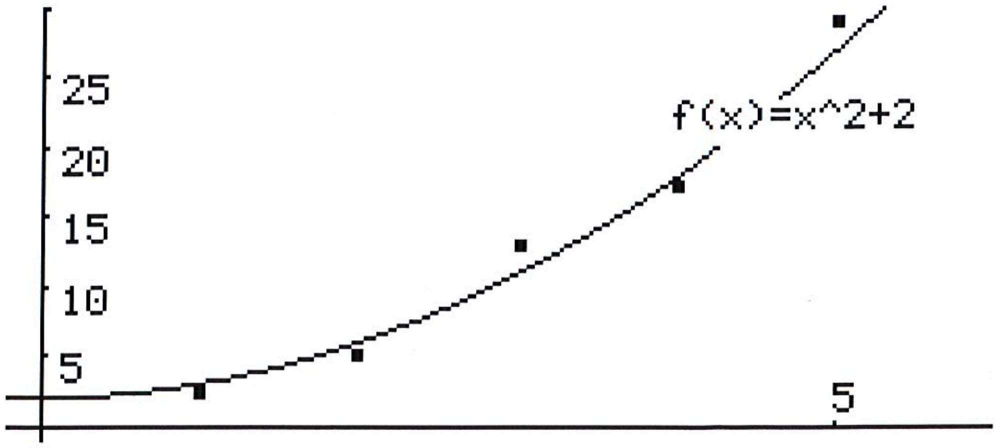

Figure 1: A simple example of interpolation.

Theorem 4.9. *[12, 4-3] The polynomial $Q(x, y)$ has a zero of order m at (α, β) if and only if*

$$Q_{r,s}(\alpha, \beta) = 0 \text{ for all } r \text{ and } s \text{ such that } 0 \leq r + s < m. \tag{4.24}$$

Proof. By definition, $ord(Q : \alpha, \beta) \geq m$ iff $Q(x + \alpha, y + \beta)$ has a zero of order m at $(0,0)$. But by Theorem 4.7, $Q(x + \alpha, y + \beta)$ has a zero of order m at $(0,0)$ iff $Q_{r,s}(\alpha, \beta) = 0$ for all $0 \leq r + s < m$. □

4.4 Interpolation Theorem

When we start decoding we are given the vector or word $(\beta_0, \beta_1, ..., \beta_{n-1})$ which is the sent vector $(f(\alpha^0), f(\alpha^1), ..., f(\alpha^{n-1}))$ after it has been transmitted and which may now contain some errors. Our goal is to find the word corresponding to the polynomial $f(x)$ that we used initially to attain the codeword. We will achieve this by finding a list of polynomials so that when we evaluate them at the field elements we are close to the received vector $(\beta_0, \beta_1, ..., \beta_{n-1})$. First we will need to build a two-variable polynomial over $\mathbb{F}_q$ by interpolating on the points $\{(\alpha^0, \beta_0), (\alpha^1, \beta_1), ..., (\alpha^{n-1}, \beta_{n-1})\}$.

The basic idea of interpolating is to find a function (and in our case a polynomial) that closely matches a given set of data points. For example, given the points $\{(1, 2.2), (2, 5), (3, 13), (4, 17.1), (5, 29)\}$, we can see in Figure 1 that the function $f(x) = x^2 + 2$ closely fits the data.

The Interpolation Theorem 4.10. *[12, 5-1] Let $m(\alpha, \beta) : (\alpha, \beta) \in \mathbb{F}_q^2$ be a multiplicity function and let $\phi_0 < \phi_1 < ...$ be an arbitrary monomial order. Then there exists a nonzero polynomial $Q(x, y)$ of the form*

$$Q(x, y) = \sum_{i=0}^{C} a_i \phi_i(x, y) \tag{4.25}$$

where

$$C = \sum_{\alpha,\beta} \binom{m(\alpha,\beta)+1}{2}$$

which has a zero of multiplicity $m(\alpha,\beta)$, at $(x,y) = (\alpha,\beta)$, for all $(\alpha,\beta) \in \mathbb{F}_q^2$.

Proof. By Theorem 4.9, $Q(x,y)$ has a zero of multiplicity m at (α,β) if and only if

$$Q_{r,s}(\alpha,\beta) = 0 \text{ for all } (r,s) \text{ such that } 0 \leq r+s < m(\alpha,\beta) \tag{4.26}$$

There are $\binom{m(\alpha,\beta)+1}{2}$ choices for (r,s) in (4.26) and by (4.22), each such choice imposes one homogeneous linear constraint on the coefficients a_i. In total there are C such linear constraints imposed on the $C+1$ coefficients $a_0, a_1, ..., a_C$. It follows that there must be at least one nonzero solution to this set of equations, which corresponds to a nonzero polynomial $Q(x,y)$ of the form (4.25) with the required multiplicities. □

Corollary 4.11. *[12, 5-2] For any $(1,v)$, there is a nonzero polynomial $Q(x,y)$ with the required zero multiplicities whose $(1,v)$-degree is strictly less that $\sqrt{2vC}$*

Proof. Take $\{\phi(x,y)\}$ to be $(1,v)$-revlex order. Then by (4.25),

$$\deg_{1,v} Q(x,y) \leq \max\{\deg_{1,v} \phi_j(x,y) : j = 0, \ldots, C\} = \deg_{1,v} \phi_C(x,y) = r_A(C),$$

where $A = (a_k)$ is the sequence $\text{Ind}(x^K)$, for $(1,v)$-revlex order. But $r_A(C) < \sqrt{2vC}$, by a straightforward generalization of Theorem 4.4 □

4.5 Factorization Theorem

Definition 4.6. [12, 5-2] We define the *Q-Score* of a function $f \in \mathbb{F}[x]$ in relation to $Q(x,y) \in \mathbb{F}[x,y]$ as

$$S_Q(f) = \sum_{\alpha \in \mathbb{F}} ord(Q : \alpha, f(\alpha)).$$

For our purposes the Q-Score gives us a measure of how close $(f(\alpha^0), f(\alpha^1), ..., f(\alpha^{n-1}))$ is to $(\beta_0, \beta_1, ...\beta_{n-1})$ for any f we should choose. Because of how we've built the polynomial $Q(x,y)$, we can determine how close a given codeword is to the received word by measuring the Q-Score.

The Factorization Theorem 4.12. *[12, 5-2]*

Suppose $f(x) \in \mathbb{F}[x]$ of degree less than v, $Q(x,y) \in \mathbb{F}[x,y]$, and

$$S_Q(f) > deg_{1,v} Q.$$

Then $y - f(x)$ is a factor of $Q(x,y)$.

Proof. Let $Q(x,y) = \sum_{i,j} a_{i,j}x^i y^j$. Then $Q(x, f(x))$ is a polynomial in x:

$$Q(x, f(x)) = \sum_{i,j\geq 0} a_{i,j}x^i f(x)^j. \tag{4.27}$$

Lemma 4.13. *If $f(x) \in \mathbb{F}[x]$ of degree less than v, then* $\deg Q(x, f(x)) \leq \deg_{1,v} Q(x,y)$.

Proof. For $a_{i,j} \neq 0$, $\deg(x^i f(x)^j) \leq \deg(x^i x^{vj}) = i + vj \leq \max(i + vj : a_{i,j} \neq 0) = \deg_{1,v} Q(x,y)$. $\square$

Lemma 4.14. $Q(x, f(x)) = 0$ *if and only if* $(y - f(x))|Q(x,y)$.

Proof. Let us view $Q(x,y)$ as a polynomial in y over the rational field $F(x)$. Then by the division algorithm, we can write

$$Q(x,y) = Q_0(x,y)(y - f(x)) + r(x), \tag{4.28}$$

where $r(x) \in F(x)$. Substituting $f(x)$ for y in (4.28), we obtain

$$Q(x, f(x)) = r(x),$$

so that $Q(x, f(x)) = 0$ if and only if $r(x) = 0$, which is equivalent to the stated result. $\square$

Lemma 4.15. *If ord$(Q : \alpha, \beta) = K$, and $f(\alpha) = \beta$, then*

$$(x - \alpha)^K | Q(x, f(x)).$$

Proof. Using Theorem 4.8, we can express $Q(x,y)$ as a polynomial in $(x - \alpha)$ and $(y - \beta)$:

$$Q(x,y) = \sum_{i,j} b_{i,j}(x - \alpha)^i (y - \beta)^j.$$

Substituting $f(x)$ for y, we have

$$Q(x, f(x)) = \sum_{i,j} b_{i,j}(x - \alpha)^i (f(x) - \beta)^j. \tag{4.29}$$

We know that $f(\alpha) = \beta$, and so it follows that $(x - \alpha)$ divides $(f(x) - \beta)$. Moreover, $(x - \alpha)^{i+j}$ divides $(x - \alpha)^i (f(x) - \beta)^j$. Since $\text{ord}(Q : \alpha, \beta) = K$, the coefficient $b_{i,j} = 0$ for all $i + j < K$. Therefore $(x - \alpha)^K$ divides every non-zero term of $Q(x, f(x))$, and so $(x - \alpha)^K$ divides $Q(x, f(x))$. $\square$

We can now complete the proof of the Factorization Theorem 4.12. By Lemma 4.15, we know that $\prod_{\alpha\in\mathbb{F}}(x - a)^{\text{ord}(Q:\alpha,f(\alpha))}$ divides $Q(x, f(x))$. But by Lemma 4.13, the degree of $Q(x, f(x)$ is (at most) $\deg_{1,v} Q(x,y)$, and the degree of $\prod_{\alpha\in\mathbb{F}}(x - \alpha)^{\text{ord}(Q:\alpha,f(\alpha))}$ is $S_Q(f)$. Thus if $S_Q(f)$ exceed $\deg_{1,v} Q$, it follows that $Q(x, f(x) = 0$, and so by Lemma 4.14, $y - f(x)$ divides $Q(x,y)$.

$\square$

4.6 GS Decoder

McEliece [12, 6] introduces the following notation that will be useful in our discussion.

The following two functions report in how many places $(f(\alpha^0), f(\alpha^1), ..., f(\alpha^{n-1}))$ agree and disagree with $(\beta_0, \beta_1, ..., \beta_{n-1})$.

$$K(f, \beta) = |\{i : f(\alpha_i) = \beta_i\}| \tag{4.30}$$

$$D(f, \beta) = |\{i : f(\alpha_i) \neq \beta_i\}| = n - K(f, \beta) \tag{4.31}$$

We use the number $C(n, m)$ as the upper range of the two-variable polynomial we will construct and in a couple other definitions.

$$C(n, m) = n\binom{m+1}{2} = \frac{nm(m+1)}{2} \tag{4.32}$$

We also define the following three useful numbers:

$$K_m(n, k) = min\{K : A(mK, v) > C(n, m)\} = 1 + \lfloor r_A(C)/m \rfloor \tag{4.33}$$

$$t_m(n, k) = n - K_m(n, k) = n - 1 - \lfloor r_A(C)/m \rfloor \tag{4.34}$$

$$L_m(n, k) = max\{L : B(L, v) \leq C(n, m)\} = r_B(C). \tag{4.35}$$

Using Theorem 4.4 and 4.5 we have the following bounds for the K_m, t_m, and L_m.

$$\left\lfloor \sqrt{vn\frac{m+1}{m}} - \frac{v}{2m} \right\rfloor + 1 \leq K_m \leq \left\lfloor \sqrt{\frac{vn(m+1)}{m}} \right\rfloor - 1 \tag{4.36}$$

$$n - \left\lfloor \sqrt{vn\frac{m+1}{m}} \right\rfloor \leq t_m \leq n - 1 - \left\lfloor \sqrt{vn\frac{m+1}{m}} - \frac{v}{2m} \right\rfloor \tag{4.37}$$

$$L_m = \left\lfloor \sqrt{\frac{n}{v}m(m+1) + \left(\frac{v+2}{2v}\right)^2} - \frac{v+2}{2v} \right\rfloor < (m + \frac{1}{2})\sqrt{\frac{n}{v}}. \tag{4.38}$$

The GS(m) Decoder. [12, 6-2] The GS(m) decoder constructs a nonzero two-variable polynomial of the form

$$Q(x, y) = \sum_{j=0}^{C(n,m)} a_j \phi_j(x, y),$$

where $\phi_0 < \phi_1 < ...$ is $(1, v)$-revlex monomial order, such that $Q(x, y)$ has a zero of order m at each of the n points (α_i, β_i), for $i = 1, ..., n$ (the Interpolation Theorem 4.10) guarantees that such a polynomial exists.) The output of the algorithm is the list of y-roots of $Q(x, y)$, i.e,

$$\mathcal{L} = \{f(x) \in F[x] : (y - f(x))|Q(x, y)\}.$$

Theorem 4.16. *[12, 6-2] The output list, $\mathcal{L}$ contains every polynomial of degree $\leq v$ such that $K(f,\beta) \geq K_m$. Furthermore, the number of polynomials in the list is at most L_m.*

Proof. By (4.14) $\deg_{1,v} Q(x,y) \leq \max\{\deg_{1,v} \phi_i(x,y) : i = 0, ..., C\} = r_A(C)$. Hence by Theorem 4.12, any polynomial $f(x)$ of degree $\leq v$ such that $mK(f,\beta) > r_A(C)$, will be a y-root of $Q(x,y)$. In other words, if $K(f,\beta) \geq 1 + \lfloor r_A(C)/m \rfloor = K_m$, $f(x)$ will be on the list.

On the other hand, by (4.15), the y-degree of $Q(x,y)$ is $\leq r_B(C(n,m)) = L_m$. Since the number of y-roots of $Q(x,y)$ cannot exceed its y-degree it follows that the output list contains at most L_m polynomials. □

As we can see, if $\leq t_m$ errors occur in transmission then the $GS(m)$ decoder should include the correct word in the list $\mathcal{L}$ that it returns. The list $\mathcal{L}$ shouldn't be too large, having length at most L_m, which does increase directly with m. Although as m increases we can decode more errors, we must also perform more computation since the size of the two variable polynomial must increase, and so there is a practical limit on how many errors we might decode with the algorithm.

4.7 Decoding Algorithm

Let us pull together the previous results. Following is a more exhaustive and detailed statement of the algorithm and how it fits together. We are given n, k, and a field $\mathbb{F}_q$ and a multiplicity m. The decoder receives the input: $\beta = (\beta_0, \beta_1, ..., \beta_{n-1}) \in \mathbb{F}_q^n$. We decode as follows.

1. We build a two-variable polynomial $Q(x,y)$ with zeros of multiplicity $m(\alpha,\beta) = m$ (constant given above) at the points (α^0, β_0), (α^1, β_1),..., $(\alpha^{n-1}, \beta_{n-1})$; at every other point, $m(\alpha,\beta) = 0$. By increasing the multiplicity at each of the given points the function $Q(x,y)$ we increase the number of errors that can be corrected. As mentioned before, the notion of multiplicity forces the function Q to pass through the given point more times, or have a singularity, at that point.

   The Interpolation Theorem indicates how to build the function $Q(x,y)$.

   $$Q(x,y) = \sum_{i=0}^{c} a_i \phi(x,y)$$

   where

   $$c = \sum_{\alpha,\beta} \binom{m(\alpha,\beta)+1}{2} = n\binom{m+1}{2},$$

   and where $\phi(x,y)$ is the $(1, k-1)$-revlex monomial ordering.

   We now have the basic structure of the polynomial but not the coefficients. By Theorem 4.9 we know that our function $Q(x,y)$ will have multiplicities at the above points if and only if:

   $$Q_{r,s}(\alpha,\beta) = \sum_{i,j} \binom{i}{r}\binom{j}{s} a_{i,j} x^{i-r} y^{j-s} = 0 \qquad \forall r,s \text{ s.t. } 0 \leq r+s < m.$$

Clearly there are $\binom{m+1}{2}$ equations defined by each point and there are n points for a total of $c = n\binom{m+1}{2}$ equations. Since we have c independent equations and $c+1$ unknowns (the coefficients) and the system is homogeneous, there is at least one non-trivial solution to this system of equations. And so we can find the coefficients of our function $Q(x, y)$ corresponding to each monomial.

2. Now, we take the polynomials $f(x) \in \mathbb{F}_q[x]$ with degree less than k and find their *Q-score.* If less than t_m errors have occurred, then one of these polynomials represents the original data that was sent. The *Q-score* gives the number of points at which the function $f(x)$ agrees with the one with which we interpolated to attain $Q(x, y)$.

   We will compare the *Q-Score* of each polynomial $f(x)$ to the $(1, v)$-weight of $Q(x, y)$: $\deg_{1,v} Q(x, y) = \deg_{1,v} \phi_c(x, y) = \deg_{1,v} x^I y^J = I+vJ$. For each $f(x) \in \mathbb{F}_q[x]$ with degree less than k, if $S_Q(f) > I+vJ$, then we add $f(x)$ to the list $\mathcal{L}$.

   By Theorem 4.16, if fewer than t_m errors have occurred then the list $\mathcal{L}$ is guaranteed to include the original word that was sent and few, if any, others. The algorithm returns $\mathcal{L}$ as its result.

5 Conclusion

The original Guruswami-Sudan algorithm was designed to decode as many errors as possible and was not optimized to reduce computing complexity, although it is proven to be polynomial time (see [11]). Kötter [13] and Roth and Ruckenstein [14] have taken the theory further in the area of computing complexity, significantly reducing the amount of computing time and space required to perform the calculations. Research is still continuing in this area, although perhaps the most inspired discoveries have already been made.

The two pairs of researchers on whose work this paper is primarily based, Reed and Solomon and Guruswami and Sudan, have had an enormous impact on the the field of coding theory. Both pairs possessed a unique ability to see crucial connections between the problem at hand and another already established area of mathematics, demonstrating a pragmatic and creative flare for problem-solving. In particular, Guruswami and Sudan's discovery should give aspiring mathematicians hope that even after a particular area appears to have been exhausted, significant discoveries can yet be made.

A Example

We will work though a complete example from start to finish over the very simple $(3,2,2)$ RS code described in Section 3. Since $n = q-1$, we will be working over the field $\mathbb{F}_4$. The code is not guaranteed to decode any errors, since the conventional error-correction bound is $\lfloor \frac{d-1}{2} \rfloor = 0$. However, we will find that if 1 error occurs we can narrow down the result to a list of three codewords. The multiplicity, m, is used throughout the decoding algorithm; for our example we will set $m = 2$.

A.1 Definitions

The first step is to find the monomial ordering that we will be using. Since $v = k-1 = 1$, we will use $(1,1)$-*revlex*:

$$1 < x < y < x^2 < xy < y^2 < x^3 < x^2y < xy^2 < y^3 < \dots$$

By Theorem 4.2, we can find the sequences r_A and r_B:

$$r_A = D(1,1) = \{0,1,1,2,2,2,3,3,3,3,\dots\},$$

$$r_B = D(0,1) = \{0,0,1,0,1,2,0,1,2,3,\dots\}.$$

In Section 4.6 we introduce the following values:

$$C(n,m) = n\binom{m+1}{2} = 9$$

$$t_m(n,k) = n - 1 - \lfloor r_A(C)/m \rfloor = 2 - \lfloor r_A(9)/2 \rfloor = 2 - \lfloor 3/2 \rfloor = 1$$

$$L_m = r_B(C) = r_B(9) = 3$$

A.2 Encoding and Transmission

We will be encoding vectors of the form $(\alpha_0, \alpha_1) \in \mathbb{F}_4^2$ giving us 16 possible words and so each can correspond with a hexadecimal digit. Consider that we want to send the hexadecimal digit $0x7$. In binary, we have $0b0111$, and so we encode the vector $(1, \alpha^2)$. The encoding map in the Section 3 indicates that $(1,\alpha^2) \mapsto (\alpha, 0, \alpha^2)$. This can be established by representing $(1,\alpha^2)$ as a polynomial $f(z) = 1 + (\alpha^2)z$ and evaluating it at the non-zero elements of $\mathbb{F}_4$: $(f(\alpha^0), f(\alpha), f(\alpha^2)) = (\alpha, 0, \alpha^2)$.

To simulate transmitting the encoded word across a noisy channel we will modify the second component of the vector, yielding $\mathbf{r} = (\alpha, 1, \alpha^2)$.

A.3 Decoding

We receive the word $\mathbf{r} = (\alpha, 1, \alpha^2)$. We will closely follow the steps indicated in Section 4.7 to determine a list of possible words. We must build the polynomial $Q(x,y)$ by interpolating at the points $(1, \beta_0)$, (α, β_1), and (α^2, β_2) where $\mathbf{r} = (\beta_0, \beta_1, \beta_2)$. Each point corresponds to a non-zero element in $\mathbb{F}_4$. In our case the points are: $\{(1, \alpha), (\alpha, 1), (\alpha^2, \alpha^2)\}$. Our goal is to find all polynomials of degree less that k such that when they are evaluated at the non-zero elements of $\mathbb{F}_4$ the result 'closely matches' our received vector $\mathbf{r}$. To do this we build the polynomial $Q(x, y)$ so that it has a zero of multiplicity $m = 2$ at each of the interpolating points. We can write the polynomial $Q(x, y)$ as

$$Q(x,y) = \sum_{i=0}^{C(n,m)} a_i \phi_i(x,y)$$

$$= a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2 + a_6 x^3 + a_7 x^2 y + a_8 xy^2 + a_9 y^3$$

Each monomial ϕ_i is the ith monomial in the $(1,1)$-*revlex* ordering listed above. We must find the coefficients. Theorem 4.9 states that $Q(x, y)$ has a zero of multiplicity m at each of these interpolating points if the following is satisfied:

$$Q_{r,s}(\alpha, \beta) = \sum_{i,j} \binom{i}{r} \binom{j}{s} a_{i,j} x^{i-r} y^{j-s} = 0 \qquad \forall r, s \text{ s.t. } 0 \leq r + s < m(\alpha, \beta) \qquad \forall \alpha, \beta \in \mathbb{F}_4$$

Where $(i, j) \in \{(I, J) : 0 \leq \text{Ind}(x^I, y^J) < C(n, m) = 9\}$.

Since $m(\alpha, \beta) = 0$ for every point except the interpolating points, we will use just those interpolating points. At each of those three points, there are three choices for r and s: $\{(0,0), (1,0), (0,1)\}$ yielding a total of nine equations: $Q_{0,0}(1, \alpha) = 0$, $Q_{0,0}(\alpha, 1) = 0$, $Q_{0,0}(\alpha^2, \alpha^2) = 0$, $Q_{0,1}(1, \alpha) = 0$, $Q_{0,1}(\alpha, 1) = 0$, $Q_{0,1}(\alpha^2, \alpha^2) = 0$, $Q_{1,0}(1, \alpha) = 0$, $Q_{1,0}(\alpha, 1) = 0$, $Q_{1,0}(\alpha^2, \alpha^2) = 0$. The following matrix represents the equations, with each column corresponding to a coefficient, and each row corresponding to one of the previous equations.

$$\begin{bmatrix}
1 & 1 & \alpha & 1 & \alpha & \alpha^2 & 1 & \alpha & \alpha^2 & 1 & 0 \\
1 & \alpha & 1 & \alpha^2 & \alpha & 1 & 1 & \alpha^2 & \alpha & 1 & 0 \\
1 & \alpha^2 & \alpha^2 & \alpha & \alpha & \alpha & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & \alpha^2 & 0 \\
0 & 0 & 1 & 0 & \alpha & 0 & 0 & \alpha^2 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & \alpha^2 & 0 & 0 & \alpha & 0 & \alpha & 0 \\
0 & 1 & 0 & 0 & \alpha & 0 & 1 & 0 & \alpha^2 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & \alpha^2 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & \alpha^2 & 0 & \alpha & 0 & \alpha & 0 & 0
\end{bmatrix}$$

In reduced echelon form, we have:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \alpha^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

We have one free coefficient, a_9. Since this is a homogeneous system we have a trivial solution (let $a_9 = 0$) and nontrivial solutions for $a_9 = 1, a_9 = \alpha, a_9 = \alpha^2$. We will let $a_9 = \alpha$. This yields the following values for the coefficients:

$$[\alpha, 0, 0, 0, 1, 0, \alpha, 0, 0, \alpha]$$

So we can write $Q(x, y)$ as:

$$Q(x, y) = (\alpha) + xy + (\alpha)x^3 + (\alpha)y^3$$

Having built $Q(x, y)$, we will find a list of possible sent words: they are the polynomials $f(x)$ such that $y - f(x)$ is a factor of $Q(x, y)$. The Factorization Theorem states that such polynomials will have a *Q-score* $> deg_{1,1}Q(x, y) = 3$. We take each vector in $\mathbb{F}_4^2$ and find the *Q-score* of the corresponding polynomial. If $S_Q(f) > 3$, we add the vector to our list $\mathcal{L}$. Remember that the max size of the list is $L_m = 3$.

elements $\in \mathbb{F}_4^2$	$f(x)$	$S_Q(f)$
$(0, 0)$	0	0
$(0, \alpha)$	αx	2
$(0, \alpha^2)$	$\alpha^2 x$	2
$(0, 1)$	x	2
$(\alpha, 0)$	α	2
(α, α)	$\alpha + \alpha x$	4
(α, α^2)	$\alpha + \alpha^2$	0
$(\alpha, 1)$	$\alpha + 1$	0
$(\alpha^2, 0)$	α^2	2
(α^2, α)	$\alpha^2 + \alpha x$	0
(α^2, α^2)	$\alpha^2 + \alpha^2 x$	0
$(\alpha^2, 1)$	$\alpha^2 + x$	4
$(1, 0)$	1	2
$(1, \alpha)$	$1 + \alpha x$	0
$(1, \alpha^2)$	$1 + \alpha^2 x$	4
$(1, 1)$	$1 + x$	0

So our decoder will return the following list of vectors:

$$\mathcal{L} = \{(\alpha, \alpha), (\alpha^2, 1), (1, \alpha^2)\}$$

As you can see the word that we sent, $(1, \alpha^2)$ is in the list, demonstrating the error-correcting capability of the code. Remember that in this case, we are using a code that conventionally cannot correct any errors. In larger codes the results can be much better than this – yielding exactly the word that was sent.

A.4 SAGE Code

Following is some code that can be used in SAGE, an open source computer algebra system, to compute some of the more tedious parts of the example in the previous section. In particular, building and solving the system of equations needed to find the coefficients of $Q(x, y)$ can expedited in this manner. Also included is the output of the code.

```
sage: # settings
sage: q = 4
sage: n = 3
sage: m = 2
sage: k = 2
sage: v = k -1 # this will be used in t he 1-v degrees.
sage: c = n * binomial(m+1, 2)
sage: w = [1,v]
sage: degrees =
sage: for i in range((m+q)**2):
...       for j in range((m+q)**2):
...           if not degrees.has_key(i*w[0] + j*w[1]) :
...               degrees[i*w[0] + j*w[1]] = []
...           degrees[i*w[0] + j*w[1]].append([i,j])
...
sage: #print degrees
sage: monomial_order_1v = []
sage: for deg in degrees.itervalues():
...       for pair in sorted(deg,key=operator.itemgetter(1)):
...           monomial_order_1v.append(pair)
...
sage: #print monomial_order_1v
sage: #estimates:
sage: #number of max errors to decode
sage: # we can write the r_A(j) as this (see pg. 12)
sage: # monomial_order_1v[j][0]*1 + monomial_order_1v[j][1]*v
sage: # we can write the r_B(j) as this (see pg. 12)
sage: # monomial_order_01[j][0]*1 + monomial_order_01[j][1]*v
sage: t_m = n - 1 - floor( (monomial_order_1v[c][0]*1 + monomial_order_1v[c][1]*v) /m ); t_m # see pg 18
sage: print "t_m, (max number of errors we are able to decode): ", t_m
sage: t_m_lower = n - floor( sqrt( v*n*(m+1)/m ) )
sage: t_m_upper = n - 1 - floor( sqrt( v*n*(m+1)/m - v/ 2*m ) )
sage: print float(t_m_lower)," <= t_m <= ", float(t_m_upper)
```

```
sage: print
sage: L_m = monomial_order_01[c][0]*1 + monomial_order_01[c][1]*v; L_m
sage: print "L_m (max number of codewords in the list that's returned)" #:", L_m
sage: L_m = int(floor( sqrt( (n*m*(m+1)/v) + ((v+2)/(2*v))**2 ) - (v+2)/(2*v)  ))
sage: L_m_upper = float((m + .5)) * float(sqrt(n/v))
sage: print "L_m = ", L_m , " < ", L_m_upper
t_m, (max number of errors we are able to decode):  1
1.0  <= t_m <=  1.0

L_m (max number of codewords in the list that's returned)
L_m =  3  <  4.33012701892
sage: F = GF(q,'b'); F  # this puts the polynomial in terms of 'b'
Finite Field in b of size 2^2
sage: a = F.gen();a
b
sage: data = vector(F,k,[1,a**2]); data
(1, b + 1)
sage: data_encoded = vector(F,n,[ data[0]*1 + data[1]*1, data[0] + data[1]*a, data[0] + data[1]*a**2  ]); data_encoded
(b, 0, b + 1)
sage: # the error will simulate transmitting over a noisy channel.
sage: error = vector(F,n,[0,1, 0]); error;
(0, 1, 0)
sage: data_received = vector(F,n); data_received = data_encoded + error; data_received;
(b, 1, b + 1)
sage: # now build the function Q(x,y)
sage: # first, we'll use the following points to interpolate at.
sage: # for a received word (beta_0, beta_1, beta_2), the points are (1,beta_0), (a, beta_1), and (a**2, beta_2).
sage: interpolating_points = [ vector(F,2,[ 1, data_received[0] ]), vector(F,2,[a, data_received[1] ]),
   vector(F,2,[a**2,data_received[2]]) ]
sage: interpolating_points
[(1, b), (b, 1), (b + 1, b + 1)]
sage: # to build Q(x,y), we need to find the coefficients of the monomials. We know that they must satisfy the
   following equations:
sage: # Q_r,s(alpha,beta) = 0 for 0 <= r+s < m. (Q_r,s is defined in the thesis.)
sage: # with m = 2,
sage: rs = []
sage: for r in range(m):
...       for s in range(m):
...           if (r+s < m):
...               rs.append([r,s])
...
sage: rs
[[0, 0], [0, 1], [1, 0]]
sage: #so we will have a set of equations. each polynomial will be represented by a vector over F, we will then put
   these together into a matrix and solve the system. each component of the vector corresponds with a particular coefficient.
sage: # what we're trying to find, there are n*binomial(m+1,2)+1 coefficients (from 0 to c = n*binomial(m+1,2)).
sage: coefficients = vector(F,n*binomial(m+1,2)+1)
sage: # coefficients.degree() -- number of coefficients.
sage: # get a mapping from monomial index to the exponent powers. we need this to build the polynomial Q(x,y)
sage: w = [1,1]
sage: degrees = {}
sage: for i in range((m+q)**2):
```

```
...        for j in range((m+q)**2):
...            if not degrees.has_key(i*w[0] + j*w[1]) :
...                degrees[i*w[0] + j*w[1]] = []
...            degrees[i*w[0] + j*w[1]].append([i,j])
...
sage: #print degrees
sage: ij = []
sage: for deg in degrees.itervalues():
...        for pair in sorted(deg,key=operator.itemgetter(1)):
...            ij.append(pair)
...
sage: #print ij
sage: # build the matrix, there's a row for each equation, and a column for each coefficient.
sage: matrix_to_solve = matrix(F,n*binomial(m+1,2), n*binomial(m+1,2)+1 );
sage: eq = 0
sage: for rs_pair in rs:
...        for point in interpolating_points:
...            for c in range(coefficients.degree()):
...                i = ij[c][0]
...                j = ij[c][1]
...                r = rs_pair[0]
...                s = rs_pair[1]
...                alpha = point[0]
...                beta = point[1]
...
...                # let 0^0 = 1 in the finite field:
...                #print alpha, beta, i, j, r, s
...                #print binomial(j,s), binomial(i,r)
...                if (binomial(i,r) == 0 or binomial(j,s)==0 ):
...                    #print "binomail is 0"
...                    val = 0
...                elif (alpha==0 and (i-r) == 0 and beta==0 and j-s == 0):
...                    #print "exception1"
...                    val =  binomial(i,r) *binomial(j,s)
...                elif (alpha==0 and i-r == 0):
...                    #print "exception2"
...                    val =  binomial(i,r) *binomial(j,s) * (beta)**(j-s)
...                elif (beta==0 and j-s == 0):
...                    #print "exception3"
...                    val =  binomial(i,r) *binomial(j,s) * (alpha)**(i-r)
...                else:
...                    val =  binomial(i,r) *binomial(j,s) * (alpha)**(i-r) * (beta)**(j-s)
...                #print val
...                #print (alpha)**(i-r) * (beta)**(j-s)
...                #print eq,c
...                matrix_to_solve[eq,c] = val
...            eq = eq + 1
...
sage: print matrix_to_solve
[    1     1     b     1 b b + 1     1 b b + 1     1]
[    1     b 1 b + 1     b     1 1 b + 1     b     1]
[    1 b + 1 b + 1     b     b     b     1     1     1     1]
[    0     0     1     0     1     0     0     1     0 b + 1]
[    0     0     1     0     b     0     0 b + 1     0     1]
[    0     0     1     0 b + 1     0     0     b     0     b]
[    0     1     0     0     b     0     1     0 b + 1     0]
[    0     1     0     0     1     0 b + 1     0     1     0]
[    0     1     0     0 b + 1     0     b     0     b     0]
sage: # Now, let's solve the matrix.
sage: matrix_solved = matrix_to_solve.echelon_form(); matrix_solved
[    1     0     0     0     0     0     0     0     0     1]
[    0     1     0     0     0     0     0     0     0     0]
[    0     0     1     0     0     0     0     0     0     0]
[    0     0     0     1     0     0     0     0     0     0]
[    0     0     0     0     1     0     0     0     0 b + 1]
[    0     0     0     0     0     1     0     0     0     0]
[    0     0     0     0     0     0     1     0     0     1]
[    0     0     0     0     0     0     0     1     0     0]
[    0     0     0     0     0     0     0     0     1     0]
sage: #now put together the values of the Q(x,y) polynomial
sage: #let a_9 be something, then define the rest of the coefficients in terms of it.
sage: coefficients[coefficients.degree()-1] = a;
sage: for c in range(coefficients.degree()-1):
...       coefficients[c] = matrix_solved[c,coefficients.degree()-1]*coefficients[coefficients.degree()-1]
...
sage: coefficients
(b, 0, 0, 0, 1, 0, b, 0, 0, b)
sage: # find the deg_1,1 of Q(x,y)
sage: # the degree of a polynomial is the highest non-zero, so start at the end of coefficients, until you find a
   non-zero coefficient, then find the
sage: # degree of the polynomial corresponding with that coefficient -- see the ij list.
sage: ind_Q_xy = coefficients.degree()-1
sage: while coefficients[ind_Q_xy] == 0:
...       ind_Q_xy = ind_Q_xy -1
...
sage: deg_Q_xy = ij[ind_Q_xy][0]*w[0] + ij[ind_Q_xy][1]*w[1]; deg_Q_xy
3
sage: #find the q-score of each of the possible input vectors (over F_q^k). if > deg_1,1 of Q(x,y), then add to our list.
sage: print interpolating_points
sage: #output list for collecting plausible codewords.
sage: L = []
sage: for i in F:
...       for j in F:
...           # this gives us each possible input vector in F_q^k
...           q_score_f = vector(F,2,[i,j]); q_score_f
...           # we must now figure out the q-score for this vector.
...           q_score = 0
...           # to find the q-score of the word / poly f, we need to evaluate the poly at the field elements
...           # this will give us q points of the form (0, f(0)), (a**0, f(a**0)), ..., (a**(q-1), f(a**(a-1)) )
...           # if this point is in the interpolating points, then it has a score of m, otherwise one of zero, we add
  up these q-scores to get the total q-score for the poly f.
...           for el in F:
...               q_score_point = vector(F,2,[el, q_score_f[0] + q_score_f[1]*el])
...               for pt in interpolating_points:
```

```
...                     if pt == q_score_point:
...                         q_score = q_score + m
...             # we now have a q-score for this poly / word f.
...             print q_score_f
...             print q_score
...             # we know that if the q-score of a poly is greater that deg_1,1 of Q(x,y), then it is y- f(x) is a factor of
    Q(x,y) and so this f is a plausible codeword. if this is true, we add it to our output list, L
...             if q_score > deg_Q_xy:
...                 L.append(q_score_f)
[(1, b), (b, 1), (b + 1, b + 1)]
(0, 0)
0
(0, b)
2
(0, b + 1)
2
(0, 1)
2
(b, 0)
2
(b, b)
4
(b, b + 1)
0
(b, 1)
0
(b + 1, 0)
2
(b + 1, b)
0
(b + 1, b + 1)
0
(b + 1, 1)
4
(1, 0)
2
(1, b)
0
(1, b + 1)
4
(1, 1)
0
sage: print "The sent word is one of:", L
The sent word is one of: [(b, b), (b + 1, 1), (1, b + 1)]
```

References

[1] C. E. SHANNON, *Bell System Technical Journal* **27** (1948).

[2] B. A. CIPRA, *SIAM News* **26** (1993).

[3] R. J. MCELIECE, *Scientific America* **252**, 88 (1985).

[4] I. REED and G. SOLOMON, *SIAM Journal on Applied Math* **8**, 300 (1960).

[5] J. A. GALLIAN, *Contemporary Abstract Algebra*, Houghton Mifflin Co., New York, 5th edition, 2002.

[6] V. PLESS, *Introduction to the Theory of Error-Correcting Codes*, Wiley, New York, 2 edition, 1989.

[7] M. SUDAN, 6.897 Algorithmic Introduction to Coding Theory - Lecture 4, Lecture notes from a class at MIT, 2002.

[8] M. E. O'SULLIVAN, Math 696 - Coding Theory, Lecture Notes from a class at San Diego State University, 2006.

[9] D. FORNEY, 6.451 Principles of Digital Communication II - Chapter 8, Lecture notes from a class at MIT, available at http://ocw.mit.edu, 2005.

[10] M. SUDAN, *Journal of Complexity* **13**, 180 (1997).

[11] V. GURUSWAMI and M. SUDAN, Improved Decoding of Reed-Solomon and Algebraic-Geometric Codes, in *IEEE Symposium on Foundations of Computer Science*, pp. 28–39, 1998.

[12] R. J. MCELIECE, The Guruswami-Sudan Decoding Algorithm for Reed-Solomon Codes, A tutorial discussion, 2003.

[13] R. KÖTTER and A. VARDY, Algebraic soft-decision decoding of Reed-Solomon codes, 2000.

[14] R. ROTH and G. RUCKENSTEIN, *IEEE Transactions on Information Theory* **46**, 246 (2000).

[15] J. VAN LINT, *Introduction to Coding Theory*, Springer, third edition, 1999.