Soft Decision Decoding of Reed-Solomon Product Codes

EECS 229B Final Project Report

Anand Sarwate

May 13, 2005

1 Introduction

Reed-Solomon codes are a class of non-binary cyclic codes which have strong burst and erasure error correction capabilities. Their current uses include deep-space communications, compact disc recordings, cellular communications, and digital broadcast[1]. Their popularity stems in part from the existence of encoding and decoding algorithms that are simple to implement in hardware. In addition to their practical uses, these codes also have a nice theoretical structure for proving the feasibility of certain coding and storage schemes [2] [3].

Product codes are a method for constructing longer codes from shorter component codes. The information symbols can be thought of entering the encoder as a 2-dimensional array as shown in Figure 1 [4]. The rows are first encoded individually according to a code $\mathcal{C}_1$ and then the columns of the result are encoded according to a code $\mathcal{C}_2$. If the codes $\mathcal{C}_2$ and $\mathcal{C}_2$ have minimum distances d_1 and d_2 respectively, then the product code has minimum distance $d_1 d_2$. In this project, we investigate a very restricted class of product codes in which the row and column code are both Reed-Solomon codes. These Reed-Solomon Product Codes (RS-PC) are used in the encoding of data for DVDs.

Much of the recent research about Reed-Solomon codes has come from a new approach to decoding Reed-Solomon codes, developed initially by Sudan [5]. These algorithms attempt to bridge the often significant gap between maximum likelihood (ML) decoding and previous algebraic decoding techniques. In particular, the recent result of Kötter and Vardy [6] allows soft-decision

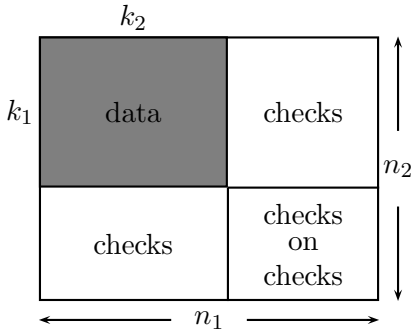

Figure 1: Product codes

decoding within in this improved framework. However, in their work they implicitly assume that the soft-information is coming from a certain channel model and modulation scheme.

The decoding of product codes is usually done in an iterative fashion by decoding columns and then rows or vice-versa. Reddy and Robinson [7] developed an algorithm to use the column decoder to provide soft information to the row decoder to improve the correction capabilities without having explicit soft information about the channel at the encoder. They use the number of errors estimated by the column decoders to provide a weight or reliability to the symbols of the decoder output. The row decoder can then use these weights to decode the rows using Forney's generalized minimum distance (GMD) algorithm [8].

The purpose of this project is to investigate the suitability of the Kötter-Vardy algorithm for the purpose of soft-decision decoding product Reed-Solomon codes. We propose a method of generating the channel soft information using the Guruswami-Sudan algorithm [9] and ideas from the Reddy-Robinson algorithm. Unfortunately, a full evaluation of the suitability of this list-decoding algorithm was not possible due to the cost of running time and its dependence on certain parameters of the algorithms.

This project thus involves two parts. The first part is an implementation of the Kötter-Vardy algorithm[1] and its constituent parts along with a summary and review of those sub-algorithms and their complexity. The second is the proposal for a way of using this algorithm to decode product codes and an example from simulation illustrating the decoding. In the next section we describe some of the mathematics that is used in the decoding algorithms. In section 3 we discuss the algorithms in detail and in section 4 we discuss our product code decoding framework.

2 Definitions

In this section we define some notation. Most of the definitions are the same as in the tutorial paper by McEliece [10].

Let $F = GF(q)$ denote the finite field with q elements. Let $F[x]$ be the ring of univariate polynomials with coefficients in F and $F[x, y]$ the ring of bivariate polynomials with coefficients in F. In this report we will assume $q = 2^m$. The information message can be thought of as a vector f of k elements in F^k:

$$f = (f_0, f_1, \ldots f_{k-1}) \ . \tag{1}$$

An (n, k) **Reed-Solomon (RS) code** over F maps the information vector f into an n-dimensional vector in F^n by polynomial evaluation. Let $\alpha_1, \alpha_2, \ldots \alpha_n$ be n nonzero elements of F. With a slight abuse of notation, we define the polynomial $f(x)$ by

$$f(X) = f_0 + f_1 X + f_2 X^2 + \ldots + f_{k-1} X^{k-1} \ . \tag{2}$$

The codeword is then

$$c = (f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n)) \ . \tag{3}$$

Most often we choose $n = q - 1$ and the α_i are the n nonzero elements of F.

Another way of defining RS codes is as a type of non-binary BCH codes. In this case we say the (m, t) Reed-Solomon code is a $(2^m - 1, 2^m - 1 - 2t, 2t + 1)$ cyclic code generated by the polynomial

$$g(X) = (X - \alpha)(X - \alpha^2) \cdots (X - \alpha^{2t}) \ . \tag{4}$$

[1] We had hoped to use an existing implementation but were unable to obtain one.

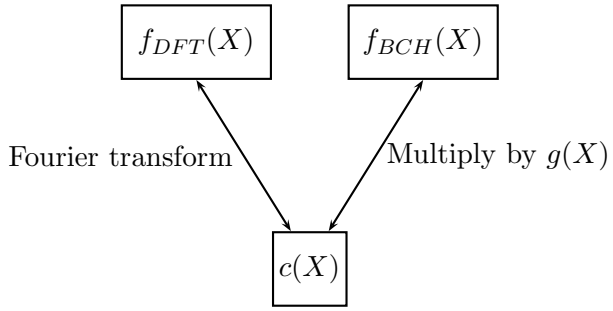

Figure 2: Relationship between systematic encoding and DFT encoding.

The minimum distance of this RS code is $2t + 1$ so it can clearly correct any received codeword with t or fewer errors.

A question of importance for the product algorithm that we propose is how to map systematically generated (in the BCH sense) codewords into the corresponding polynomial evaluation form. This mapping can be thought of in terms of the Fourier transform [11]. Consider ordering the nonzero elements of $GF(q)$ in cyclic order, so that $\alpha_k = \alpha^{k-1}$ for a primitive element α. Then evaluating the message polynomial $f(X)$ gives

$$c_j = \sum_{i=0}^{k-1} f_i \alpha^{j-1} \tag{5}$$

Since $GF(q)$ is cyclic we can think of α as an $q-1$-th root of unity. If we view the vector of coefficients of f as a n dimensional vector with $n-k$ zero components, then equation (5) is just the n-dimension discrete Fourier transform (DFT) of the coefficient vector. Thus encoding in the polynomial evaluation sense simply takes the DFT of the coefficients.

Since the RS code is the same whether generated in the BCH sense or the DFT sense, we immediately have a mapping between message polynomials. That is, given a message polynomial in the BCH sense, we can encode it systematically using the BCH generator polynomial. We can then decode the received codeword using an algorithm that considers polynomials in the DFT sense. The resulting messages can be mapped via the DFT and back using the standard BCH map. This is summarized in Figure 2.

A RS **product code** consists of two RS component codes $\mathcal{C}_1$ and $\mathcal{C}_2$, both of which are over $GF(q)$. Thus the blocklength of both components is the same, but they may differ in their minimum distances. Let $\mathcal{C}_1$ be an (n, k_1) code with minimum distance $d_1 = 2t_1 + 1$ and $\mathcal{C}_2$ be an (n, k_2) code with minimum distance $d_2 = 2t_2 + 1$. The information array $B \in F^{k_1 \times k_2}$ is encoded in the following way:

1. The rows are encoded one-by-one using $\mathcal{C}_2$ and the resulting codewords are put into the rows of a matrix B'.

2. The columns of B' are encoded one-by-one using $\mathcal{C}_1$ and the resulting codewords are put into the columns of a matrix W

The matrix W is called the **codeword matrix**.

We now restate some definitions which are needed to describe the component algorithms. Let

$$P(x) = \sum_i p_i x^i \tag{6}$$

$$Q(x,y) = \sum_{i,j} q_{ij} x^i y^j \tag{7}$$

be polynomials in $F[x]$ and $F[x,y]$ respectively. We say the polynomial $Q(x,y)$ has a **zero of multiplicity** m **at** $(0,0)$ if $q_{ij} = 0$ for all i,j such that $i+j < m$. We say $Q(x,y)$ has a **zero of multiplicity** m **at** (α,β) if $Q(x+\alpha, y+\beta)$ has a zero of multiplicity m at $(0,0)$.

The **Hasse derivative** of a univariate polynomial $P(x)$ is

$$D_H^{(1)}[P(x)] = \sum_{i=1}^{\deg P} \binom{i}{1} p_i x^{i-1} \ . \tag{8}$$

The $u-th$ Hasse derivative is

$$D_H^{(u)}[P(x)] = \sum_{i=u}^{\deg P} \binom{i}{u} p_i x^{i-u} \ . \tag{9}$$

The $(u,v)-th$ Hasse derivative of $Q(x,y)$ is

$$D_H^{(u,v)}[Q(x,y)] = \sum_{i=u}^{\deg_x(Q)} \sum_{j=v}^{\deg_y(Q)} \binom{i}{u}\binom{j}{v} q_{ij} x^{i-u} y^{j-v} \tag{10}$$

Since we are assuming F has characteristic 2, all we care about is the parity of the binomial coefficients. We write $D_H^{(u,v)}[Q(\alpha,\beta)]$ for the evaluation of (10) at the point (α,β).

Our interest in the Hasse derivative comes from two crucial facts. Firstly, $Q(x,y)$ has a zero of multiplicity m at (α,β) if and only if $D_H^{(u,v)}[Q(\alpha,\beta)]$ for all (u,v) such that $0 \le u+v < m$. Secondly, if

$$Q(x,y) = \sum_i x^i q_i(y) \tag{11}$$

Then

$$Q(x, xy+a) = \sum_{i,j} x^i y^j D_H^{(j)}[q_{i-j}(y)] \ . \tag{12}$$

The first fact will describe how to formulate certain constraints in the Guruswami-Sudan algorithm. The second fact is used to find factors of the form $(y-f(x))$ in a bivariate polynomial.

The (r,s)-weighted degree of $Q(x,y)$ is

$$\deg^{(r,s)}(Q) = \max\{ri + sj \ : \ q_{ij} \neq 0\} \ . \tag{13}$$

Of interest to us is the $(1, k-1)$-weighted degree, which assigns high weight to polynomials with high-degree terms in y.

3 Description of algorithms

The algorithm we propose for decoding these product codes is a combination of an older approach for decoding product codes and a recent innovation in soft-decision decoding for Reed-Solomon codes. The older algorithm is that of Reddy and Robinson [7] and the more recent results are those of Kötter and Vardy [6], which is based on the work of Guruswami and Sudan [9][5]. We will first describe these two algorithms and then in the next section we will describe the specific way in which we combine them.

3.1 Reddy-Robinson algorithm for product codes

The Reddy-Robinson algorithm for decoding product codes decodes one of the component codes and passes soft information reflecting the number of errors in that column to a decoder for the rows. The row decoder can use this soft information to improve its decoding performance. Reddy and Robinson propose to set weights in the first stage that can be used by Forney's generalized minimum distance (GMD) algorithm [8]. Let $\mathcal{C}_1$ be a $(n, k_1, 2t_1 + 1)$ Reed-Solomon code for the columns and $\mathcal{C}_2$ be a $(n, k_2, 2t_2 + 1)$ Reed-Solomon code for the rows.

Generalized minimum distance decoding is a simple way of using reliability information about the received codeword to improve decoding using an errors-and-erasures decoder. The algorithm is succinctly outlined in [4] but essentially works by iteratively erasing the received symbols with lowest reliability and attempting to decode. The algorithm proceeds until it reaches a given optimality condition. This "iterate until acceptance" decoding structure is used by other errors-and-erasures soft-decision decoding techniques. One recent paper in this vein is by Kamiya [12]. Forney's algorithm relies on a result that says that if the there exists a codeword within a certain weighted distance of the received vector, then there exists some K for which erasing the K least reliable symbols and doing errors-and-erasures decoding will recover that codeword.

The Reddy-Robinson algorithm is then just a way of satisfying the conditions of that theorem of Forney's. That is, they assign weights in such a way that they are guaranteed to find a codeword in the second step. Suppose the decoder for column i encounters e_i errors. For example, the Berlekamp-Massey can correct up to t errors for a $(n, k, 2t + 1)$ Reed-Solomon code. Provided the number of errors is no more than t, we can get an accurate estimate of e_i. If $t + 1$ or more errors occurred we can detect the error but not correct it unless the error pushes the transmitted codeword into the decoding region for another codeword. The weight assignment for e_i is then:

$$w_i = \begin{cases} \frac{d_2 - e_i}{d_2} & e_i \leq t_2 \\ \frac{1}{d_2} & \text{otherwise} \end{cases} \tag{14}$$

These weights must satisfy a theorem of Forney that guarantees the row decoding will be successful as long as the number of errors that occurs is not too large.

3.2 Guruswami-Sudan algorithm for Reed-Solomon Codes

The Guruswami-Sudan algorithm differs from previous decoding algorithms in that it is not a bounded distance decoding algorithm. Previous algorithms would search within a sphere of radius t around the received vector, generating an error if no codewords were found. In many RS codes it may be possible to receive a vector that is not within the conventional decoding radius of any

codeword but is closest to a single codeword. In particular, it may be possible to decode nearly all error patterns of a weight higher than half the minimum distance of the code [10].

In order to achieve this improved performance, the Guruswami-Sudan algorithm outputs a short list of candidate codewords within a given radius of the received vector. Suppose the received vector is $(\beta_1, \beta_2, \ldots, \beta_n)$. The algorithm proceeds in two steps. First, given a multiplicity factor m the decoder constructs (interpolates) a bivariate polynomial $Q(x, y)$ in $F[x, y]$:

$$Q(x, y) = \sum_{i,j} q_{ij} x^i y^j \tag{15}$$

such that $Q(x, y)$ has a zero of multiplicity m at (α_i, β_i) for $i = 1, \ldots M$ and for which the $(1, k-1)$ weighted degree is minimal. The factors of this polynomial of the form $(y - f_k(x))$ gives the list of output codewords. The second step of the Guruswami-Sudan algorithm is to factor $Q(x, y)$ into these y factors, which we call y-roots.

3.3 Kötter interpolation of bivariate polynomials

Kötter developed a polynomial-time algorithm to accomplish the bivariate polynomial interpolation required by the Guruswami-Sudan algorithm, given a target list size L, interpolation points, and their multiplicities. His approach is general enough to deal with a list of (x, y) pairs each with their own multiplicity. However, the running time increases quadratically with the number of constraints. Since the number of constraints imposed by a zero of multiplicity m is $O(m^2)$ and the running time of the algorithm is $O(C^2)$ where C is the number of constraints, the running time is roughly $O(n^2 m^4)$ for the standard Guruswami-Sudan algorithm.

Recall that the problem we are trying to solve is to impose a zeros of order m_{ij} on the points (α_j, β_i). The constraints are given by the first fact about Hasse derivatives in the previous section: $Q(x, y)$ has a zero of multiplicity m at (α, β) if and only if $D_H^{(u,v)}[Q(\alpha, \beta)]$ for all (u, v) such that $0 \leq u + v < m$. These constraints are all linear, so we can impose them one-by-one. However, for technical reasons the order in which we impose them is important – the end result is that we should impose the conditions by fixing u and incrementing v so that the constraints are imposed in the order $(0, 0)(0, 1) \cdots (0, m-1)(1, 0) \cdots (m-10)$. Let (u_c, v_c) be the derivative orders for the c-th constraint, and (α_c, β_c) the associated field elements.

The algorithm creates a list of $L+1$ polynomials and then modifies them to meet the constraints one by one. Note that L is the desired target list size. Let $Q_{c,l}(x, y)$ be the l-th polynomial in the list after applying c constraints. The output after applying all the constraints is the minimum $(1, k-1)$-weighted degree polynomial in the list. Assume that we use this weighted degree for all polynomial degree calculations.

For $c = 0$ the polynomials are $Q_{0,l}(x, y) = y^l$ for $0 \leq l \leq L$. To apply the $(c+1)$-th constraint we first calculate the **discrepancies**

$$\delta_{c,l} = D_H^{(u_{c+1}, v_{c+1})}[Q(\alpha_c, \beta_c)] \tag{16}$$

Let J_1 be the set of l's which have nonzero discrepancy – the corresponding Q's are the ones we have to fix. For all $l \notin J_1$ we set

$$Q_{c+1,l}(x, y) = Q_{c,l}(x, y) \ . \tag{17}$$

For $l \in J_1$ we calculate the minimal degree polynomial (using the weighted degree above) and its discrepancy.

$$\begin{aligned} \bar{l} &= \arg\min\{Q_{c,l}(x,y) : l \in J_1\} && (18)\\ \bar{\delta} &= \delta_{c,\bar{l}} && (19)\\ \bar{Q}(x,y) &= Q_{c,\bar{l}}(x,y) && (20) \end{aligned}$$

For $l \neq \bar{l}$ we have to modify the function to satisfy the constraint:

$$Q_{c+1,l}(x,y) = \bar{\delta} Q_{c,1}(x,y) + \delta_{c,l}\bar{Q}(x,y) \ . \tag{21}$$

For $l = \bar{l}$ we have to increase its degree:

$$Q_{c+1,\bar{l}}(x,y) = \bar{\delta} x\bar{Q}(x,y) - D_H^{(u_{c+1},v_{c+1})}[x\bar{Q}(x,y)]\Big|_{(\alpha_{c+1},\beta_{c+1})} \bar{Q}(x,y) \tag{22}$$

By choosing the smallest degree polynomial to increase its degree we are being greedy in allocating degree. At the end of imposing all the constraints we output

$$Q(x,y) = \min\{Q_{C,l}(x,y) : 0 \leq l \leq L\} \tag{23}$$

To make this concrete, consider the following short example. Suppose we want to send the polynomial $X^2 + X + \alpha^2$ using the $(7,3)$ Reed-Solomon code over $GF(2^3)$. After encoding we get a codeword that looks like:

$$\alpha_3 \quad \alpha_0 \quad \alpha_1 \quad \alpha_6 \quad \alpha_7 \quad \alpha_4 \quad \alpha_5 \tag{24}$$

After noise we get

$$\alpha_2 \quad \alpha_1 \quad \alpha_1 \quad \alpha_6 \quad \alpha_7 \quad \alpha_4 \quad \alpha_5 \tag{25}$$

The interpolated polynomial is:

$$Q(x,y) = \alpha_4 + \alpha_7 x + \alpha_7 x^2 + \alpha_7 y + \alpha_7 y^2 \ . \tag{26}$$

Indeed, to exploit the richness of information provided by the soft information, the total number of interpolation conditions s needs to be somewhat large, which in turn increases the computation time. In particular, the running time scales poorly in the interpolation multiplicity.

Example. For a larger example, let us consider the following message polynomial for a $(15,5)$ RS code over $GF(2^4)$ with primitive polynomial $1 + X + X^4$:

$$f(X) = 1 + \alpha X + \alpha^2 X^2 + \alpha^3 X^3 + \alpha^4 \ .$$

Evaluating this function at $\{\alpha^0, \alpha^2, \ldots, \alpha^{14}\}$ gives

$$c = [\alpha^{11}\ \alpha^{14}\ \alpha^9\ \alpha^{10}\ \alpha^4\ \alpha^6\ \alpha^8\ \alpha^0\ \alpha^3\ \alpha^{13}\ \alpha^7\ \alpha^0\ 0\ \alpha^0\ \alpha^{11}] \ .$$

Suppose we have an error pattern

$$e = [1\ 1\ \alpha^2\ \alpha^8\ \alpha^4\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0] \ ;$$

Then the received vector is

$$r = c + e = [\alpha^{12}\ \alpha^{3}\ \alpha^{11}\ \alpha 0\ \alpha^{6}\ \alpha^{8}\ \alpha^{0}\ \alpha^{3}\ \alpha^{13}\ \alpha^{7}\ \alpha^{0}\ 0\ \alpha^{0}\ \alpha^{11}]\ .$$

These elements are the β_j's in the algorithm. Interpolating at the points (α^{j-1}, β_j) with multiplicity 2 and $L = 4$ gives:

$$\begin{aligned}
Q(x,y) = (&\alpha^{12} + \alpha^{10}x + \alpha^{1}x^2 + \alpha^{14}x^3 + \alpha^{10}x^5 + \alpha^{5}x^6 + \alpha^{2}x^7 + \alpha^{2}x^8 + \alpha^{14}x^9 + \alpha^{1}x^{10} \\
&+ \alpha^{2}x^{11} + \alpha^{14}x^{13} + \alpha^{4}x^{14} + \alpha^{12}x^{15}) \\
&+ (\alpha^{11} + \alpha^{0}x^2 + \alpha^{9}x^3 + \alpha^{7}x^4 + \alpha^{3}x^5 + \alpha^{12}x^6 + \alpha^{7}x^7 + \alpha^{5}x^8 + \alpha^{1}x^9 + \alpha^{13}x^{10} \\
&+ \alpha^{4}x^{11})y \\
&+ (\alpha^{2} + \alpha^{0}x^2 + \alpha^{5}x^3 + \alpha^{13}x^4 + \alpha^{3}x^5 + \alpha^{5}x^6 + \alpha^{12}x^8)y^2 \\
&+ (\alpha^{6} + \alpha^{4}x + \alpha^{9}x^2 + \alpha^{14}x^3 + \alpha^{5}x^4)y^3 \\
&+ \alpha^{14}y^4
\end{aligned}$$

Computing this polynomial in MATLAB took a few minutes, as opposed to a smaller polynomial for $m = 1$ and $L = 1$ which took a few seconds. The effect of asking for a larger list size and/or higher multiplicities has such an effect on the running time that it became infeasible to run large simulations[2].

3.4 Roth-Ruckenstein factorization for bivariate polynomials

The Roth-Ruckenstein algorithm [13] is a fast way of finding factors of the form $(y - f(x))$ in a bivariate polynomial $Q(x, y)$ such that $\deg f(x) \leq k - 1$. This algorithm accomplishes the factorization step in the Guruswami-Sudan algorithm – the output is a list of candidate message polynomials. The implementation used in this report is the depth-first-search version due to McEliece.

The algorithm works by reducing the degree of $Q(x, y)$ and calling itself recursively on the reduction. We can think of this a depth first search on a tree. Associated with each vertex v of the tree is a bivariate polynomial $P_v(x, y)$. The root vertex has polynomial $Q(x, y)$. The depth in the tree is the degree of the root of the polynomial. Thus if we are looking for polynomials of degree less than k we need only search up to depth $k + 1$ in this tree (the root is defined to have depth -1 for the purposes of this algorithm). Associated to each edge in the tree is a coefficient that will be used to construct the y-root. The y roots will then be paths to the root in the tree with coefficients given by the edges.

Let $\deg(v)$ be the degree (depth) of vertex v. Suppose that

$$P_v(x, 0) \neq 0 \tag{27}$$

and $\deg(v) < k$. Let A_v be the roots of $P_v(0, x)$. For each $r \in A_v$ we create a vertex u_r that is a child of v and label the edge with r. We associate to u_r the polynomial

$$P_{u_r} = \Gamma\left(P_v(x, xy + r)\right)\ , \tag{28}$$

where $\Gamma(\cdot)$ is a function that maps a polynomial $P(x, y)$ to $P(x, y)/x^j$ for the largest j such that $x^j | P(x, y)$. The algorithm then operates on each of the u_r. If $\deg(v) \geq k$ then we have hit a dead-end and we return nothing.

[2]Unfortunately we did not have the time to build a faster (and more difficult to prototype) simulator.

If $P_v(x, 0) = 0$ then we have found a y-root. To construct it, we trace a path back to the root of the tree. Suppose the vertices in this path are $\{v_i\}$, with $v_1 = v$. If r_i is the label associated with the edge from v_i to its parent then the output y-root is given by

$$f_v(x) = \sum_i r_i x^{\deg v_i} \ . \tag{29}$$

This algorithm is a fast way of finding the roots of a bivariate polynomial and represents a significant improvement over Gaussian elimination.

Example. For the polynomial calculated above, the errors are within the decoding capabilities of the code. The factorization algorithm outputs a single factor:

$$f(X) = 1 + \alpha X + \alpha^2 X^2 + \alpha^3 X^3 + \alpha^4 \ .$$

Which is the original message polynomial.

Example. Suppose we transmit the same message polynomial as before but now there is one additional error, so that we are now beyond the design distance of the code:

$$e = [1\ 1\ \alpha^2\ \alpha^8\ \alpha^4\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0] \ ; \tag{30}$$

Interpolating with $L = 1$ and $m = 1$ we obtain:

$$\begin{aligned} Q(x, y) = (&\alpha^8 + \alpha^{13} x + \alpha^3 x^2 + \alpha^{11} x^3 + \alpha^5 x^4 + \alpha^8 x^5 + \alpha^5 x^6 + \alpha^{10} x^7 + \alpha^{13} x^8) \\ &(\alpha^8 + \alpha^{10} x + \alpha^{13} x^2 + \alpha^4 x^3 + \alpha^{13} x^4) y \end{aligned}$$

Using the factorization algorithm above we again recover

$$f(X) = 1 + \alpha X + \alpha^2 X^2 + \alpha^3 X^3 + \alpha^4 \ .$$

So we can see that the Guruswami-Sudan algorithm can correct errors beyond the design distance in this example.

3.5 Kötter-Vardy algorithm for soft-decision decoding

Kötter and Vardy [6] modified the Guruswami-Sudan algorithm to use soft information from the communication channel to help improve decoding performance. Their algorithm translates the *a posteriori* symbol probabilities into a choice of interpolation points and multiplicities. More specifically, given a $q \times n$ **reliability matrix** Π with entries

$$\pi_{ij} = \Pr(f(\alpha_j) = \gamma_i \mid \beta_j) \ , \tag{31}$$

provide a construction for a $q \times n$ matrix of nonnegative integer entries M, called a **multiplicity matrix**. They then modify the interpolation step in the Guruswami-Sudan algorithm to construct $Q(x, y)$ to have zeros of multiplicity m_{ij} at (α_j, γ_i). The factorization step is identical.

As we have seen, the interpolation and factorization can be accomplished using the Kötter and Roth-Ruckenstein algorithms. Kötter and Vardy's contribution was a method of constructing M from Π using a fixed total interpolation order s. As s gets large, intuitively we should see the

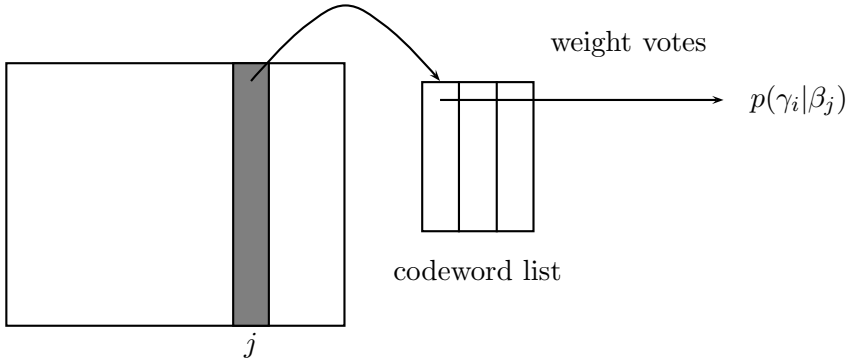

Figure 3: Soft information generation in pictures.

histogram of each column of M approach the distribution $p(\gamma_i|\beta_j)$. Their algorithm is a way of approximating this distribution for finite s. One interesting observation made in [6] is that this approximation problem can be viewed as a gambling problem.

The algorithm works by constructing M one point at a time and modifying Π until s points in total are added. We start by setting $\Pi^* = \Pi$. At each step step we calculate

$$\pi_{ij}^{ast} \rightarrow \frac{\pi^*}{m_{ij}+2} \tag{32}$$

$$m_{ij} \rightarrow m_{ij}+1 \tag{33}$$

$$s \rightarrow s-1 \tag{34}$$

We continue until $s = 0$.

Note that if $\pi_{ij} = 1$ only for the point (α_i, β_j) and $s = n-1$, then the Kötter-Vardy algorithm is exactly the same as the Guruswami-Sudan algorithm with $m = 1$.

In order to fully take advantage of the soft information it is clear that adding more points will be necessary. However, large s implies more constraints and increased running time. In order to achieve the improved error-correction capability of this algorithm we must both be clever in the design of our soft information in order to limit the extra computational burden.

4 Error-correction for product codes

The initial goal of this project was to investigate the suitability the new decoding techniques for RS codes in the decoding of product RS codes. That is, can we somehow generate soft information from column decoders (like in the Reddy-Robinson algorithm) that can be used to do soft-decision decoding of the rows in the vein of Kötter and Vardy? In particular, can we obtain improved performance beyond the minimum distance for the product code?

The algorithm we propose is a slight modification of the Reddy-Robinson approach. We use the original Guruswami-Sudan algorithm on the columns and then construct symbol probabilities for the decoded columns. The approach we take is to let the decoded codeword candidates "vote" on the symbol probabilities. These votes are normalized into a distribution on the symbols and passed to the row decoder in the reliability matrix. The proposed model is shown in Figure 3

Suppose we are decoding column j – call this w_j. The column decoder uses the Guruswami-Sudan algorithm to output a list of possible messages, $\{f_1(x), f_2(x), \ldots f_L(x)\}$, each of which induces a different codeword $\{c_1, c_2, \ldots c_L\}$. Note that the number of message candidates may be less than L. We measure the Hamming distance $\epsilon_l = d_H(c_l, w_j)$ for all l. This is an estimate of the number of errors for this candidate. We then construct a weighted histogram for each position in the decoded vector.

Let the weight associated to message candidate f_l be

$$\eta_l = \begin{cases} \frac{d_2 - \epsilon_l}{d_2} & \epsilon_l \leq t_2 \\ \frac{1}{d_2} & \text{otherwise} \end{cases} \tag{35}$$

This is the same weighting scheme as the Reddy-Robinson approach. However, in this case we construct a histogram using the weights and the output message list. Let $\bar{\pi}_{ij}^{(u)}$ be

$$\bar{\pi}_{ij}^{(u)} = \sum_{l=1}^{L} \eta_l \mathbf{1}(f_{l,u} = \alpha_i) \ . \tag{36}$$

where $\mathbf{1}$ is the indicator function. This is a weighted sum of the occurrences of α_i in row u of the decoded message across the candidate codewords l. The matrix $\bar{Pi}^{(u)}$ is an unnormalized reliability matrix for row (u). To construct the true reliability matrix we simply normalize each column:

$$\pi_{ij}^{(u)} = \frac{\bar{\pi}_{ij}^{(u)}}{\sum_i \bar{\pi}_{ij}^{(u)}} \ . \tag{37}$$

One question we must address is what to do if the decoder suffers an error – that is, what happens if there are too many errors to decode correctly but not enough to force a decision to a different codeword? In this case the Reddy-Robinson algorithm just uses the original data bits as the decoder estimate. This is only be valid for encodings in systematic form – if we use Fourier encoding, then truncating to the first k_2 bits will not have very much relation to the original data sequence. The diagram in Figure 2 is very important for the decoding of product codes. We therefore assume that the encodings have all been systematic, and we will re-interpret out decoded messages by translating the DFT message to the BCH message.

As in the Reddy-Robinson result, the choice of the soft information is crucial to the providing performance guarantees. That is, we must verify that the weighting above will produce success in decoding for the Kötter-Vardy algorithm. Unlike with GMD, there are no tight conditions to guarantee correct decoding, since we are trying to decode outside the error-correction radius and thus we cannot easily provide general guarantees without taking the particular structure of the code into account. However, one relevant result we can examine is Theorem 17 in [6]:

Theorem 1 *The Kötter-Vardy algorithm with list size limited to L produces a list that contains a codeword $\mathbf{c} \in \mathcal{C}$ if*

$$\frac{\langle \Pi, c \rangle}{\sqrt{\langle \Pi, \Pi \rangle}} \geq \frac{\sqrt{k-1}}{1 - \frac{1}{L}\left(\frac{1}{R^*} + \frac{\sqrt{q}}{2\sqrt{R^*}}\right)} \ , \tag{38}$$

where $R^* = (k-1)/n$.

The inner products are

$$\langle \Pi, c \rangle = \sum_{ij} \pi_{ij} \mathbf{1}(c_j = \alpha_i) \ . \tag{39}$$

$$\langle \Pi, \Pi \rangle = \sum_{ij} \pi_{ij}^2 \ . \tag{40}$$

For the examples we have done, we can write an expression for L that will guarantee a nonempty decoding list for the $(15, 5)$ code over $GF(16)$ that has served as our example. Let γ denote the ratio on the left. Then

$$\gamma \geq \frac{2}{1 - \frac{1}{L}(15/4 + \sqrt{15})} \tag{41}$$

or

$$L \geq \frac{15/4 + \sqrt{15}}{1 - 2/\gamma} \tag{42}$$

As we can see, to guarantee finding a codeword, we must have $L \approx 8$. In practice, however, much smaller L are sufficient.

We tested the product-code decoding algorithm described in this report using the $(15, 5)$ Reed-Solomon code discussed in the examples in the previous section for both row and column encoding. The transmitted message matrix is given by the following matrix over $GF(2^4)$:

$$w = \begin{bmatrix} 0 & 1 & \alpha & \alpha^2 & \alpha^3 \\ 1 & \alpha & \alpha^2 & \alpha^3 & 0 \\ \alpha & \alpha^2 & \alpha^3 & 0 & 1 \\ \alpha^2 & \alpha^3 & 0 & 1 & \alpha \\ \alpha^3 & 0 & 1 & \alpha & \alpha^2 \end{bmatrix} \ . \tag{43}$$

The matrix is encoded and errors are added in the form of a banded matrix. For a band of width 3 we have:

$$\begin{bmatrix} e & e & & & & & & & & & & & & & \\ e & e & e & & & & & & & & & & & & \\ & e & e & e & & & & & & & & & & & \\ & & e & e & e & & & & & & & & & & \\ & & & e & e & e & & & & & & & & & \\ & & & & e & e & e & & & & & & & & \\ & & & & & e & e & e & & & & & & & \\ & & & & & & e & e & e & & & & & & \\ & & & & & & & e & e & e & & & & & \\ & & & & & & & & e & e & e & & & & \\ & & & & & & & & & e & e & e & & & \\ & & & & & & & & & & e & e & e & & \\ & & & & & & & & & & & e & e & e & \\ & & & & & & & & & & & & e & e & \end{bmatrix} \ . \tag{44}$$

The algorithm can correct all banded errors of width up to 5, which is also achievable with Reddy-Robinson and GMD. Unfortunately, we did not have sufficient time to investigate experimentally its performance on other error patterns, but we suspect it will work as well as the Reddy-Robinson method and in some cases surpass it.

5 Conclusion

Recent improvements in the decoding capabilities of Reed-Solomon codes using soft decision decoding makes it appealing to examine decoding algorithms for Reed-Solomon product codes. Using the standard Reddy-Robinson framework, it may be possible to use list decoding to provide soft information to the Kötter-Vardy decoder. In this project we described the relevant algorithms and their implementation along with some example computations. We then described a way of using list decoding to generate soft information for the Kötter-Vardy decoder and showed its performance for a sample error pattern.

References

[1] S. B. Wicker and V. K. Bhargava, eds., *Reed-Solomon Codes and their Applications*. New York: IEEE Press, 1994.

[2] E. Martinian, G. W. Wornell, and R. Zamir, "Source Coding With Encoder Side Information," *IEEE Transactions on Information Theory*, vol. submitted, 2004. ArXiV cs.IT/0412112.

[3] A. Dimakis, "Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes," in *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks*, April 2005.

[4] S. Lin and D. J. Costello, *Error Control Coding (2nd edition)*. Upper Saddle River, NJ: Prentice-Hall, 2004.

[5] M. Sudan, "Decoding of Reed Solomon Codes beyond the Error-Correction Bound," *Journal of Complexity*, vol. 13, pp. 180–193, 1997.

[6] R. Koetter and A. Vardy, "Algebraic Soft-Decision Decoding of Reed-Solomon Codes," *IEEE Transactions on Information Theory*, vol. 49, no. 11, pp. 2809–2825, 2003.

[7] S. R. Reddy and J. P. Robinson, "Random Error and Burst Correction by Iterated Codes," *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 182–185, 1972.

[8] G. D. Forney, "Generalized minimum distance decoding," *IEEE Transactions on Information Theory*, vol. 12, no. 2, pp. 125–131, 1966.

[9] V. Guruswami and M. Sudan, "Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes," *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 1757–1767, 1999.

[10] R. J. McEliece, "The Guruswami-Sudan Decoding Algorithm for Reed-Solomon Codes," Tech. Rep. 42-153, JPL Interplanetary Network Progress Report, 2003.

[11] R. E. Blahut, *Algebraic Codes for Data Transmission*. Cambridge: Cambridge University Press, 2002.

[12] N. Kamiya, "On Acceptance Criterion for Efficient Successive Errors-and-Erasures Decoding of Reed-Solomon and BCH Codes,"

[13] R. Roth and G. Ruckenstein, "Efficient Decoding of Reed-Solomon Codes beyond Half the Minimum Distance," *IEEE Transactions on Information Theory*, vol. 46, no. 1, pp. 246–257, 2000.