List-Decoding of Reed-Solomon Codes

Huan Yao
September 18, 2002

Reed-Solomon (RS) codes are error-correction codes that have been widely used since the 60's. Efficient decoding algorithms that decodes up to $e < d/2$, i.e., the number of errors less than half of the minimum distance of the code, have been known for over four decades. In the past few years, new list-decoding algorithms have been discovered to allow decoding beyond that bound, up to $e < n - \sqrt{n(n-d)}$, where n is the block length of the code. Variations of the algorithm can also be applied to other algebraic codes, such as the BCH code, algebraic-geometric codes, etc.

1 Introduction

Let us use $(n, k+1, d)_q$ [1] to denote an error-correction code over an alphabet $\mathbb{F}_q$ of size q which takes a word from $\mathbb{F}_q^{k+1}$ (the message) and outputs a codeword in $\mathbb{F}_q^n$ such that the minimum Hamming distance between any pair of codewords is at least d.

Let us define a Hamming ball around a point $\mathbf{y} \in \mathbb{F}_q^n$ with radius e as a set of points that is within Hamming distance e of $\mathbf{y}$. The number of points in a Hamming ball is $\sum_{i=0}^{e} \binom{n}{i} (q-1)^i$.

The list-decoding problem is that of finding all codewords within the Hamming ball around the received word $\mathbf{y}$ with radius e, which is referred to as *decoding radius*. Zero, one, or more codewords might be listed. An error happens when more than e symbols are incorrectly received and the transmitted codeword is not among the codewords listed.

For an error-correction code with minimum distance d, the largest decoding radius we can choose so that the Hamming balls around all codewords are non-overlapping is $\lfloor (d-1)/2 \rfloor$. Each received word will be in the Hamming ball of at most one codeword. This corresponds to the *unique* decoding situation that we are familiar with, where at most one codeword is returned.

In most cases, these Hamming balls will not cover all points in $\mathbb{F}_q^n$, meaning, for received words that are not inside any of the Hamming balls, nothing will be outputted, i.e, undecodable. The percentage of coverage could be very very small in high dimensions. When the non-overlapping Hamming

[1] The reason for using "$k + 1$" will become apparent later.

balls do cover the entire $\mathbb{F}_q^n$, the code is known as a *perfect code*. The only perfect codes are the Hamming codes (e.g., the (7,4,3) code) and two specific codes proposed by Golay [16].

We know that RS codes are optimal in the sense that for given n and k, it has the maximum d. This is because RS codes meet the *Singleton bound* of $d \leq n-k$. Codes that meet the Singleton bound are called *maximum distance separable* (MDS) codes. For most values n and k, the Singleton bound limits the maximum value of d, and prohibits the code from being perfect. If we can only decode when $e \leq \lfloor (d-1)/2 \rfloor$, then there will be a lot of received words that are undecodable.

When doing list-decoding with $e > \lfloor (d-1)/2 \rfloor$, the Hamming balls around the codewords will be overlapping. The decoding is not always unique, but it allows more received words to be decodable. If the resulting list is small, we can then find the best one among them as the final unique output.

To see the impact of increasing decoding radius e, let us look at some numerical examples. For a $(64, 48, 17)_{64}$ Reed-Solomon code, the traditional decoding radius is $e = \lfloor (d-1)/2 \rfloor = 8$ while the list-decoding can achieve up to $e = \lfloor n - \sqrt{n(n-d)} \rfloor = 9$. The volume of the Hamming balls are 1.1×10^{24} and 4.3×10^{26}, respectively. In terms of coverage of the whole space $\mathbb{F}_q^n$, they achieve 0.0014% and 0.54% (ignore overlapping), respectively. We can see that list-decoding is much better. Although, not all points are equally important to cover. The ones closer to codewords are more important.

Let us look at a performance measure very dear to us, the block error rate. Assume the (IID) symbol error rate is, say, 0.03. Correcting up to 8 errors out of 64 symbols will lead to block error rate 1.2×10^{-4}. Correcting one more error gets us 2.0×10^{-5}, a factor of 6 improvement.

Note, that list-decoding doesn't always do better. For example, for a $(255, 239, 17)_{256}$ code (which is used in practice in satellite communications), both methods can correct up to 8 errors. It turns out that the gain of list-decoding is more significant at low code rate codes than high code rate. We can see this by looking at a comparison of e/n vs. rate $R = (k+1)/n \approx k/n$ for the two methods in Fig. 1.

2 A Brief History

- In 1957, Elias [3] first introduced list-decoding.
- In 1960, Peterson [12] developed an algorithm that can be used for decoding RS codes in polynomial time, as long as $e \leq \lfloor (d-1)/2 \rfloor$.

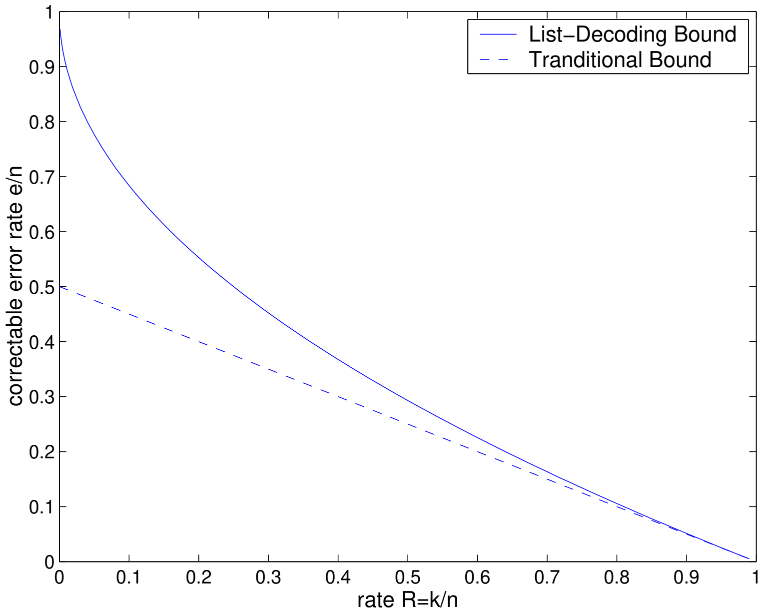

Figure 1: Comparison of the error correcting capability as a function of code rate between the list-decoding method, which can correct up to $e/n < 1 - \sqrt{k/n}$, and the traditional (unique decoding) method, which can correct up to $e/n < (1 - k/n)/2$ errors.

- In 1966, Forney [4] developed generalized minimum distance decoding, which was a form of soft list-decoding.
- By 1968, Berlekamp and Massey [1] had developed the now well-known classical algorithm for RS decoding, improving the running time to $O(n^2)$ [2], but they did not improve the decoding radius e.
- Almost two decades later, Welch and Berlekamp [17] improved the decoding radius to $\lfloor (d-1)/2 \rfloor + 1$ using univariate rational functions.
- A few years later, Sidelnikov [13] and Dumer [2] constructed algorithms which correct up to $\lfloor (d-1)/2 \rfloor + c \log n$ for some constant c.
- In 1997, Sudan [14], using the ideas of polynomial reconstruction developed a list-decoding algorithm for RS decoding that further beats

[2] $O(\cdot)$ stands for "order of" $f = O(x)$ means $|f| < ax$ for some a as $x \to \infty$.

the half distance bound. The exact expression of the number of errors correctable is complicated, but a lower bound is $n - \sqrt{2n(n-d)}$.

- In 1999, Guruswami and Sudan [8] improved upon the previous list-decoding algorithm to achieve a decoding radius of $n - \sqrt{n(n-d)}$, which is the maximum possible using a list-decoding type algorithm.
- In 2000, Koetter and Vardy [11] extend the above algorithm to soft decoding.

RS codes used in practice :

Some RS codes used in practice are shown here for the readers' information [3]. In the digital video broadcasting (DVB) standard, RS $(255, 239, 17)_{256}$ code is used as an outer code to an inner convolutional code. For deep-space communication, a typical configuration would use a $(255, 233, 23)_{256}$ RS code as outer code, also to an inner convolutional code. CD-ROM uses a concatenation scheme, known as cross-interleaved Reed Solomon code, which uses a $(32, 28, 5)_{256}$ RS code as inner code and a $(28, 24, 5)_{256}$ code as outer code. Note, that they are mostly high rate codes.

3 Johnson Bound

One question you might have is why can't we increase the decoding radius arbitrarily so we can list more and more codewords and get better and better performance. If we take this argument to the limit and set $e = n$, then we would be listing all q^{k+1} codewords, and this would be point-less. So the key is to choose a decoding radius e as large as possible, such that there are only a small number of codewords in the Hamming ball. In particular, *small* is defined as polynomial in block length n as $n \to \infty$ (fix $R \approx k/n$), instead of exponential, like q^{k+1}.

This problem is addressed by the (extended) Johnson bound [9], which states that

Theorem 1 *Let q,n,d,e be arbitrary positive integers with $d < (1 - 1/q)n$. Let $A'_q(n, d, e)$ denote the maximum number of points that may be place in a Hamming ball with radius e in the space $\mathbb{F}_q^n$ such that all pairwise distances between the points are at least d. If e satisfies the condition*

$$e < \left(1 - \frac{1}{q}\right)\left(1 - \sqrt{1 - \frac{q}{q-1} \cdot \frac{d}{n}}\right) n, \qquad (1)$$

[3]See http://www.ittc.ukans.edu/ paden/reference/guides/ECC/application

then we have

$$A'_q(n,d,e) \le \min\left\{ n(q-1), \frac{nd}{nd-2e(n-\frac{qe}{2(q-1)})} \right\}. \qquad (2)$$

Furthermore, if e equals R.H.S. of eq.(1), then $A'_q(n,d,e) \le 2n(q-1)-1$.

The second upper bound on $A'_q(n,d,e)$ in eq.(2) is the "classical" version of Johnson bound for the q-ary case. There is a geometric proof for the above bound. We will not cover it here. Interested readers are referred to [9].

To simplify the above bound, let us assume $q \approx q-1$, then the bound becomes.

$$\textit{If } e < n - \sqrt{n(n-d)}, \textit{ then } A'_q(n,d,e) \le \min\left\{nq, \tfrac{d}{d-(2e-e^2/n)}\right\}.$$

Only looking at the first term of the upper bound, nq, we see that this is polynomial in n as long as q does not grow exponentially with n, which is true for most codes.

It turns out that if e is above Johnson bound, then a random coding argument shows that there must be exponential number of codewords in the Hamming ball. The argument is as follows [15, 5] :

Let $\mathbf{y}$ be the all zeros vector. Pick $c_1, \cdots, c_m$ (the codewords to be inside the Hamming ball) independently as follows : Each coordinate of c_i is chosen IID to be 0 with probability $1-e/n$ and to be a random non-zero element of $\mathbb{F}_q$ otherwise. On average, each codeword has Hamming weight e. So a significant fraction of the codewords we randomly pick would be inside the Hamming ball. We now just have to make sure they are at least distance d away from each other. If q is large, then the probability of two randomly chosen coordinates being equal is approximately $(1-e/n)^2$, when both are 0's. The expected distance between two codewords is thus $n(1-(1-e/n)^2) = 2e - e^2/n$. This quantity equals d when $e = \sqrt{n(n-d)}$. However, if the required minimum distance is less (even if slightly, percentage-wise), i.e., $d < (1-\epsilon)(2e-e^2/n)$, $\epsilon > 0$, then as $n \to \infty$ (fix e/n and d/n), by the law of large numbers, the probability that a pair of codewords are within distance d is exponentially small. Thus it is possible to pick exponentially many codewords at least distance d apart and have Hamming weight no more than e.

4 List-Decoding of Reed-Solomon Code

In this section, we will cover the details of list-decoding of Reed-Solomon codes. We will first review the encoding of RS codes, and rephrase the RS decoding problem as a polynomial reconstruction ("curve fitting") problem. Next, we will go through the details of the algorithm, describe it, justify it, and show that it achieves the Johnson bound. Finally, we will also present some complexity results.

Almost all material in this section are from the Guruswami and Sudan paper [8]. I will not explicitly reference again.

4.1 Reed-Solomon Encoding

The Reed-Solomon encoder which maps a message from $\mathbb{F}^{k+1}$ to a codeword in $\mathbb{F}^n$ is define as follows. For each message $(m_0, m_1, m_2, \cdots, m_k)$, a polynomial $p(x)$ is defined as

$$p(x) \stackrel{\text{def}}{=} m_0 + m_1 x + m_2 x^2 + \cdots + m_k x^k. \tag{3}$$

Then, we take a vector $\mathbf{x}$ of *distinct* elements $x_1, x_2, \cdots, x_n \in \mathbb{F}_q$ (hence we need $n \leq q$), and evaluate $p(x)$ at these values. The codeword outputted is $(p(x_1), p(x_2), \cdots, p(x_n))$.

There are other ways of defining the RS codes. (I suppose they are all equivalent.) But this one is the most compatible with the decoding algorithm.

4.2 The Polynomial Reconstruction Problem

The list decoding problem is that of finding all codewords within the Hamming ball around the received word $\mathbf{y}$ with radius e. Using the definition of RS encoding and Hamming ball, this means finding all polynomials $p(x)$ such that $(p(x_1), p(x_2), \cdots, p(x_n))$ differs from $\mathbf{y}$ at no more than e positions, equivalently, agrees with $\mathbf{y}$ at no less than $t = n - e$ positions.

Formally, the RS decoding problem reduces to a *polynomial reconstruction* (curve fitting) problem defined as follows :

INPUT : *Integers k, t and n points $\{(x_i, y_i)\}_{i=1}^n$ where $x_i, y_i \in \mathbb{F}_q$.*

OUTPUT : *All polynomials $p(x)$ of degree at most k such that $y_i = p(x_i)$ for at least t values of $i \in \{1, 2, \cdots, n\}$.*

Next, we will solve the polynomial reconstruction problem.

4.3 Description of Algorithm

Let us first look at how the algorithm works before looking at why it works. (If all this seem a little magical at the beginning, the justification is only one page away.)

The algorithm to be introduced has two main phases. In the first phase, we solve for a bivariate polynomial $Q(x, y)$ which "fits" all the points $\{(x_i, y_i)\}_{i=1}^{n}$, where fitting implies $Q(x_i, y_i) = 0$. We also require that Q has singularity of at least r at those points, which we will define later. In the second phase, we factorize $Q(x, y)$. In particular, we find all factors of $Q(x, y)$ that are in the form of $(y - p(x))$. These polynomials $p(x)$ will be the *potential* outputs.

It is interesting that in order to solve for univariate polynomials $p(x)$, we actually start with solving for a bivariate polynomial $Q(x, y)$.

Before the formal description of the algorithm, we need a couple of definitions.

Definition 2 - Singularity: *The meaning of $Q(x, y)$ having singularity r at (x_i, y_i) is the shift of $Q(x, y)$ to (x_i, y_i), i.e, $Q^{(i)}(x, y) = Q(x + x_i, y + y_i)$ has no monomial terms of degree less than r. (For example, if $r = 3$, we don't want terms like $1, x, y, x^2, xy, y^2$.)*

Definition 3 - Weighted Degree: *For non-negative weights w_1 and w_2, the (w_1, w_2)-weighted degree of the monomial $x^i y^j$ is defined as $iw_1 + jw_2$.*

We now describe the polynomial reconstruct algorithm :
Input : *Integers k, t and n points $\{(x_i, y_i)\}_{i=1}^{n}$ where $x_i, y_i \in \mathbb{F}_q$.*
Step 0 : Compute parameters r, l such that

$$rt > l \text{ and } n \begin{pmatrix} r+1 \\ 2 \end{pmatrix} < \frac{l(l+2)}{2k}. \tag{4}$$

Step 1 : Find a non-trivial polynomial $Q(x, y)$ with $(1, k)$-weighted degree at most l that has singularity r at all points $\{(x_i, y_i)\}_{i=1}^{n}$. More specifically, we need to find all the coefficients $q_{j_1 j_2}$ such that :

$$\forall i \in 1, 2, \cdots n, \forall j_1, j_2 \geq 0, s.t., j_1 + j_2 < r,$$
$$q_{j_1 j_2}^{(i)} \stackrel{\text{def}}{=} \sum_{j_1' \geq j_1} \sum_{j_2' \geq j_2} \begin{pmatrix} j_1' \\ j_1 \end{pmatrix} \begin{pmatrix} j_2' \\ j_2 \end{pmatrix} q_{j_1' j_2'} x_i^{j_1' - j_1} y_i^{j_2' - j_2} = 0. \tag{5}$$

Step 2 : Find all polynomial $p(x)$ with coefficients in $\mathbb{F}_q$ of degree at most k such that $y - p(x)$ is a factor of $Q(x, y)$. For each such polynomial $p(x)$, check if $p(x_i) = y_i$ for at least t values of $i \in 1, 2, \cdots n$, and if so, include $p(x)$ in output list.

4.4 Correctness of the Algorithm

In this section, we will justify the above algorithm and discuss why it works.

Our final goal is to find polynomials $p(x)$ of degree at most k such that $y_i = p(x_i)$ for at least t values of $i \in \{1, 2, \cdots, n\}$, in other words, find bivariate polynomials $y - p(x)$ that fits at least t out of the n pairs of (x_i, y_i).

One major problem is that we do not know which t out of the n points to fit. Trying all possibilities has exponential complexity. So the idea is to find one bivariate polynomial $Q(x, y)$ that fits all of them, and somehow get $y - p(x)$, which has a subset of the roots, to be a factor of $Q(x, y)$. (The backward statement, if $y - p(x)$ is a factor of $Q(x, y)$, then every root of $y - p(x) = 0$ is a root of $Q(x, y) = 0$, is clearly true.) Lemma 4 and 5 below show how the algorithm gets $y - p(x)$ to be factors of $Q(x, y)$.

Lemma 4 *If (x_i, y_i) is an input point and $p(x)$ is any polynomial such that $y_i = p(x_i)$, then $(x - x_i)^r$ divides $g(x) \stackrel{\text{def}}{=} Q(x, p(x))$.*

The proof mainly uses the property of Q having singularity of degree r at (x_i, y_i), which is a requirement in Step 1. See [8] for the full proof.

Lemma 4 tells what happens when $y - p(x)$ fits one pair of (x_i, y_i). Lemma 5 states that when $y - p(x)$ fits at least t points, it will be a factor of $Q(x, y)$, which is what we want.

Lemma 5 *If $p(x)$ is a polynomial of degree at most k, such that $y_i = p(x_i)$ for at least t values of $i \in \{1, 2, \cdots, n\}$ and $rt > l$, then $y - p(x)$ divides Q.*

The main idea of the proof is that suppose $p(x)$ is such that $y_i = p(x_i)$ for $i = 1, 2, \cdots, t$. Then $(x - x_1)^r (x - x_2)^r \cdots (x - x_t)^r$, which is a degree rt polynomial divides $Q(x, p(x))$ which has degree at most $l < rt$. The only possibility for this to happen is $Q(x, p(x)) \equiv 0$, which means $y - p(x)$ divides $Q(x, y)$. (We can view $Q(x, y)$ as a polynomial in y with coefficients being polynomials in x and $y = p(x)$ is one of the roots.) See [8] for the full proof.

Note, the reason we want to consider $(1, k)$-weighted degree of $Q(x, y)$ is that it is an upper bound on the degree of $Q(x, p(x))$ when $p(x)$ is at most degree k.

Lemma 6 explains why we need the second condition in Step 0. (We saw the use of the first condition $rt > l$ in Lemma 5).

Lemma 6 *If* $n\binom{r+1}{2} < \frac{l(l+2)}{2k}$*, then a polynomial* $Q(x, y)$ *as sought in Step 1 does exist (and can be found in polynomial time by solving a linear system.)*

In Step 1, we have to solve for polynomial $Q(x, y)$ with $(1, k)$-weighted degree at most l that has singularity r at all points $\{(x_i, y_i)\}_{i=1}^{n}$. This is equivalent to solving a set of linear equations, eq.(5), which is possible if we have at least as many variables as equations. For each of the constraint of Q having singularity r at (x_i, y_i), there are $r(r+1)/2$ equations. (The monomials of degree less than r are $1, (x, y), (x^2, xy, y^2), \cdots, (x^{r-1}y^0, x^{r-2}y, \cdots, x^0y^{r-1})$.) The number of variables is the number of monomials of Q, which is $\sum_{j=0}^{\lfloor \frac{l}{k} \rfloor}(l + 1 - kj) \leq \frac{l(l+2)}{2k}$. We then have the inequality in Lemma 6.

Lemma 7 *If* n, k, t *satisfies* $t^2 > kn$*, then both conditions in Step 0 hold.*

Having $t^2 > kn$ is equivalent to $e < n - \sqrt{n(n-d)}$, which is Johnson bound. The proof is by solving for r and l explicitly as a function of n, k, and t.

It is worth noting that in the original version of the algorithm proposed by Sudan in 97 [14], the parameter r was absent, i.e., it simply required $Q(x_i, y_i) = 0$, so $r = 1$. It turned out they couldn't reach Johnson bound. The introduction of the parameter r closed the gap.

4.5 Complexity

One component of the complexity analysis is to upper bound the number of solutions listed. Johnson bounds already gave us one set of upper bound, here is another.

Since $y - p(x)$ are factors of $Q(x, y)$, the number of such factors, M, which is the maximum number of solutions possibly produced by the list-decoder, is at most the degree of the variable y in $Q(x, y)$. Since the $(1, k)$-weighted degree of Q is at most l, $M \leq \lfloor \frac{l}{k} \rfloor$.

For $q = 64$, $n = 64$, $k = 48$, and $d = 17$, we have $t = \lfloor \sqrt{kn} \rfloor + 1 = 55$. The smallest set of r and l that satisfies Step 0 is $r = 25$ and $l = 1374$. With those values, we have 20800 equations and 20805 variables. (Looks large, but hopefully, our computers are fast enough.) The upper bound on the number of solutions is $\lfloor \frac{l}{k} \rfloor = 29$, (which is not that large).

Another component of the complexity analysis is run time. The linear system in Step 1 has $O(l^2/k)$ variables, which takes $O((l^2/k)^3) = O(l^6/k^3)$

operations in field $\mathbb{F}_q$ to solve. It turns out that factoring of $Q(x, y)$ in Step 2 takes $O(l^5/k^3)$ operations, see [8] for detail. So overall, the complexity is $O(l^6/k^3)$. Since

$$l = O(\max\left\{\frac{knt}{t^2 - kn}, t\right\}),$$

the algorithm can be implemented to run using $O(\max\{\frac{k^3n^6t^6}{(t^2-kn)^6}, \frac{t^6}{k^3}\})$ field operations.

5 Conclusions

In this report, we have summarized the list-decoding algorithm for Reed-Solomon codes introduced by Guruswami and Sudan, and discussed some reasons and intuitions behind it. This new algorithm increases the decoding radius of Reed-Solomon codes beyond half of the minimum distance, which is a bound that we have lived with for four decades. It also reaches Johnson bound for the theoretical maximum decoding radius of list-decoding.

RS decoding was done using a polynomial reconstruction ("curve fitting") algorithm. It first solves for a bivariate polynomial $Q(x, y)$ which has singularity of degree r at all points $\{(x_i, y_i)\}_{i=1}^n$, and then factors it to find the desired solutions. Both the number of solutions listed and the run time of the algorithm are bounded by polynomials in block length n, so the algorithm is efficient.

This algorithm does not only apply to Reed-Solomon codes, variations of the algorithm can also be applied to other codes. It can be applied to alternant codes that includes a wide family of codes such as BCH codes, Goppa codes, etc. It can also be applied to algebraic-geometric codes, which are generalizations of RS codes, but do not have the constraint of $n \leq q$, so that they can be implemented over small alphabets. Algebraic-geometric code decoding is presented in the second half of the paper [8], but are not be covered here. Other codes that can be decoded include Reed-Muller codes [15] and Chinese remained codes [7].

If we also allow the degree of singularity to be different at different points, we can use that to indicate a measure of confidence. This algorithm can then be modified to work as a soft decoding algorithm [11] to work with channels like the erasure channel or the AWGN channel. It can also be applied to concatenated codes [10] where the inner code provides some confidence measure to the outer code where the list-decoding takes place.

Readers who want to learn more about list-decoding are refered to [6].

References

[1] E. R. Berlekamp, *Algebraic Coding Theory.* McGraw Hill, New York, 1968.

[2] I. I. Dumer, *Two algorithms for the decoding of linear codes.* Problems of Information Transmission, Vol. 25(1), pp. 24-32, 1989.

[3] P. Elias, *List-decoding for noisy channels.* 1957-IRE WESCON Convention Record, Pt. 2, pp. 94-104, 1957.

[4] G. D. Forney, *Generalized minimum distance decoding.* IEEE Transactions on Information Theory, Vol. 12(2), pp. 125-131, Apr. 1966.

[5] O. Goldreich, R Rubinfeld, and M. Sudan, *Learning polynomials with queries: The highly noisy case.* Proc. of the 36th Annual IEEE Symposium on Foundations of Computer Science, pp. 294-303, 1995.

[6] V. Guruswami, *List decoding of error-correction codes.* Ph.D. thesis, MIT, Aug. 2001.

[7] V. Guruswami, A. Sahai, and M. Sudan, *"Soft-decision" decoding of Chinese remainder codes.* Proceedings of the 41st Annual Symposium on Foundations of Computer Science, pp: 159-168, 2000.

[8] V. Guruswami and M. Sudan, *Improved decoding of Reed-Solomon and algebraic-geometry codes.* IEEE Transactions on Information Theory, Vol. 45(6), pp. 1757 -1767, Sept. 1999,

[9] V. Guruswami and M. Sudan, *Extensions to the Johnson bound.* manuscript, Feb. 2001. (http://theory.lcs.mit.edu/ madhu/papers.html)

[10] V. Guruswami and M. Sudan, *Decoding concatenated codes using soft information.* Proc. 17th IEEE Annual Conference on Computational Complexity, pp, 148-157, 2002.

[11] R. Kotter and A. Vardy, *Algebraic soft-decision decoding of Reed-Solomon codes.* Proc. IEEE International Symposium on Information Theory, pp. 61, 2000.

[12] W. W. Peterson, *Encoding and error-correction procedures for Bose-Chaudhuri codes.* IRE Transactions on Information Theory, Vol. 6, pp. 459-470, 1960.

[13] V. M. Sidelnikov, *Decoding Reed-Solomon codes beyond* $(d-1)/2$ *and zeros of multivariate polynomials.* Problems of Information Transmission, Vol. 30(1), pp. 44-49, 1994.

[14] M. Sudan, *Decoding of Reed-Solomon codes beyond the error-correction bound.* Journal of Complexity, Vol. 13(1), pp. 180-193, 1997.

[15] M. Sudan, *List decoding: algorithms and applications.* unpublished manuscript, May 2000. (http://theory.lcs.mit.edu/ madhu/papers.html)

[16] M. Sudan, coding theory course notes.

[17] L. Welch and E. R. Berlekamp, *Error correction of algebraic block codes.* US Patent Number 4,633,470 issued December 1986.