Error-correction up to the information-theoretic limit*

Venkatesan Guruswami†
Computer Science and Engineering
University of Washington
Seattle, WA 98105
venkat@cs.washington.edu

Atri Rudra‡
Computer Science and Engineering
University at Buffalo, SUNY
Buffalo, NY 14260
atri@cse.buffalo.edu

ABSTRACT

Ever since the birth of coding theory almost 60 years ago, researchers have been pursuing the elusive goal of constructing the "best codes," whose encoding introduces the minimum possible redundancy for the level of noise they can correct. In this article, we survey recent progress in *list decoding* that has led to *efficient* error-correction schemes with an *optimal* amount of redundancy, even against *worst-case errors* caused by a potentially malicious channel. To correct a proportion p (say 20%) of worst-case errors, these codes only need close to a proportion p of redundant symbols. The redundancy cannot possibly be any lower information-theoretically. This new method holds the promise of correcting a factor of two more errors compared to the conventional algorithms currently in use in diverse everyday applications.

1. INTRODUCTION

Coping with corrupt data is an essential part of our modern day lives, even if most of the time we are blissfully unaware of it. When we watch television, the TV has to deal with signals that get distorted during transmission. While talking on our cellphones, the phone has to work with audio signals that get corrupted during transmission through the atmosphere though we definitely are aware of it when the connection is poor! When we surf the Internet, the TCP/IP protocol has to account for packets that get lost or garbled while being routed. When we watch movies on DVDs, the player has to overcome loss of data due to scratches. Even when we do our groceries, the scanner has to deal with distorted barcodes on packages.

The key ingredient in coping with errors in these and many other applications is an *error-correcting code* or just *code* for brevity. The idea behind codes is conceptually simple: add redundancy to the information so that even if the resulting data gets corrupted, e.g. packets get corrupted during routing or the DVD gets some scratches, the original information can still be recovered.

Ideally, we would like to add a small amount of redundancy and at the same time be able to correct many errors. As one might expect, these are conflicting goals and striking the right balance is where things get interesting. For example, consider the code where every information bit is repeated say a 100 times (this is known as the repetition code). Intuitively, this code should do well. In particular, the following is a natural error-recovery procedure or a *decoding algorithm*: for every consecutive 100 bits of the data, identify whether the majority of the bits is 0 or 1, and output the corresponding bit. Unless we happen to be unlucky, this decoding algorithm can recover from quite a few errors. The downside is that every hundred bits of data contain only *one* bit of information– imagine how large a DVD would need to be in order to store a movie with such redundancy! On the other extreme is the parity code, which appends the parity of the information bits at the end of the message. This code uses the minimum amount of redundancy possible but has poor error recovery capabilities. Indeed, even if just two bits get flipped, it can go undetected. For example, 0001 gets encoded as 00011 under the parity code. If the first two bits get corrupted and we receive 11011, we would misinterpret the original message to be 1101. Imagine Clark Gable saying at the end of your small parity encoded DVD for *Gone with the Wind*, "Frankly, my dear, I don't give a ham !"

To capture this inherent tension between the redundancy and the error tolerance of codes, let us define codes and some key parameters formally. A code C is given by an *encoding* map of the form $C : \Sigma^k \to \Sigma^n$ (for integers $k < n$) which encodes a sequence of k symbols from Σ into a larger sequence of n symbols. Given a *message* $m \in \Sigma^k$, $C(m)$ is known as the corresponding *codeword*. The parameters k, n and Σ are called the *dimension*, *block length* and *alphabet* of C respectively. We will often use the ratio $R = k/n$, which is called the *rate* of C. Note that R exactly captures the amount of information contained per bit of a codeword. The *Hamming distance* between two strings in Σ^n is the number of coordinates where they differ. The *minimum distance* of a code is defined to be the smallest Hamming distance between two distinct codewords.

Thus, our question of interest can be now re-stated as follows: given a code C of rate R, what is the maximum fraction of errors (which we will henceforth refer to as ρ) that can be tolerated by C? Now as every codeword has k symbols of information, it is intuitive that in the worst case at least k symbols of a codeword should be uncorrupted to have any hope of recovering the original information. In

*The final version of this paper appears in the IEEE Transactions on Information Theory as [9].

†Currently visiting the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

‡This work was done when the author was at U. Washington.

other words, we can only have $\rho \leqslant (n-k)/k = 1-R$, irrespective of the computational power of the decoder.

The main focus of this article is the following question:

> Can we construct a code C of rate R that can be efficiently decoded from close to a $1-R$ fraction of errors?

Quite surprisingly, we will show the answer to be yes. Thus, the above simple information-theoretic limit can in fact be approached! In particular, for small rates, we can recover from situations where almost all of the data can be corrupted. For example, we will be able to recover even if Clark Gable were to spout "alhfksa, hy meap xH don'z hive b hayn!" There are in fact two parts to the above question. First, the code should be such that the identity of the message is uniquely (or near uniquely) pinned down based on the noisy version of its encoding. Second, we need an *efficient* procedure to recover the message based on the corrupted codeword, with runtime bounded by a hopefully small polynomial in the block length n. Brute-force approaches such as searching through all codewords require time exponential in n, and are prohibitive computationally. The main message of this article is that a variant of the widely deployed Reed-Solomon codes can in fact be error-corrected *efficiently*, even as the fraction of errors approaches the absolute $1-R$ limit.

We stress that in the above question, the noise model is a *worst-case* one, where the channel is modeled as an adversary/jammer that can corrupt the codeword in an arbitrary manner, subject only to a limit on the total number of errors. No assumptions are made on how the errors are distributed. The worst-case model was put forth in Hamming's influential paper [12]. An alternate approach, pioneered by Shannon in his seminal work [16], is to model the noisy channel as a stochastic process whose behavior is governed by a precisely known probability law. A simple example is the binary symmetric channel (BSC_ρ) where each bit transmitted gets flipped with a probability ρ, independent of all other bits. For every such channel, Shannon exactly characterized the largest rate at which reliable communication is possible.

We note that obtaining *algorithmic* results is more difficult in worst-case noise model, and traditionally it was believed that codes designed for worst-case noise faced a limit of $(1-R)/2$ fraction of errors, which is factor 2 off from the information-theoretic limit of a fraction $(1-R)$ of errors. In attempting to correct $e > (n-k)/2$ errors, we face a problem: there may be more than one codeword within Hamming distance e from the received word. In any code mapping k symbols to n symbols, there must be two codewords at distance $(n-k+1)$ or less. If one of these codewords is transmitted and gets distorted by errors to half-way between the two codewords, unmbiguous recovery of the original message becomes infeasible. This suggests that beyond a fraction $(1-R)/2$ of worst-case errors, the original message is unrecoverable. This was indeed the conventional wisdom in coding theory till the 1990's.

A natural strategy, in the event of multiple close-by codewords, would be to allow the decoder to output a list of codewords. This model is called *list decoding*. It was introduced in the late 1950s by Elias [2] and Wozencraft [19], and revived with an algorithmic focus for a cryptographic application by Goldreich and Levin [4]. Further, it turns out that the bad case above is rather pathological — for typical patterns of e errors, for e much bigger than $(n-k)/2$, the original codeword will be the only codeword within distance e. Thus, list decoding in addition to handling the bad error patterns for unique decoding, also allows us to to uniquely recover the transmitted codeword for *most* error patterns.

It is interesting to note that even though the list decoding problem has a long history in the coding theory world[1], a large share of the algorithmic progress in list decoding has happened in the theoretical computer science community. One of the reasons for this happy coincidence is that list decoding has found numerous applications in cryptography and computational complexity theory (e.g. see the discussion on randomness extractors in Section 5).

In particular, in the last decade, the subject of list decoding has witnessed a lot of activity, culminating in algorithms that correct close to the information-theoretically optimal $1-R$ fraction of errors with rate R. The purpose of this article is to discuss this recent body of results which deliver the full promise of codes against worst-case errors in list decoding. We begin in Section 2 by describing a popular family of codes and few decoding algorithms for it.

2. REED-SOLOMON CODES

In 1960, Irving Reed and Gustave Solomon described a construction of error-correcting codes, which are called Reed-Solomon codes after them, based on polynomials over finite fields[2]. Almost 50 years after their invention, Reed-Solomon codes (henceforth, RS codes) remain ubiquitous today in diverse applications ranging from magnetic recording to UPS bar codes to satellite communications. To describe the simple and elegant idea behind RS codes, imagine Alice wishes to communicate a pair of numbers (a,b) to Bob. We can think of (a,b) as specifying a line in the plane (with X,Y axes) with equation $Y = aX+b$. Clearly, to specify the line, it suffices to communicate just two points on the line. To guard against errors, Alice can oversample this line and send more points to Bob, so that even if a few points are distorted by errors, the collection of points resembles the original line more closely than any other line (Figure 1). Thus, Bob can hope to recover the correct line, and in particular (a,b).

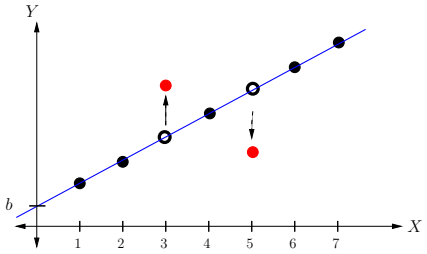

Figure 1: Oversampling from a line $Y = aX + b$ to tolerate errors, which occur above at $X = 3$ and 5.

To encode longer messages consisting of k symbols via an RS code, one thinks of these as the coefficients of a polynomial $f(X)$ of degree $k-1$ in a natural way, and encodes the message as $n > k$ points from the curve $Y =$

[1]The problem was introduced about 50 years ago and the main combinatorial limitations of list decoding were established in the 1970s and 1980s.

[2]For this article, readers not conversant with fields can think of a finite field as $\{0,1,\ldots,p-1\}$ for a prime p with addition and multiplication operations defined modulo p.

$f(X) = 0$. Equivalently, the encoding consists of the values of the polynomial $f(X)$ at n different points. Formally, if $\mathbb{F}$ is a finite field with at least n elements, and $S = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ is a sequence of n *distinct* elements, the RS encoding $\mathsf{RS}_{\mathbb{F},S,n,k}(m)$ of a message $m = (m_0, m_1, \ldots, m_{k-1})$ is given by

$$\mathsf{RS}_{\mathbb{F},S,n,k}(m) = (f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n))$$

where $f(X) = m_0 + m_1 X + \cdots + m_{k-1} X^{k-1}$. We stress here that the choice of S is up to the code designer — we will exploit this feature in Section 3.2.

The following basic algebraic fact will be crucial:

> A non-zero polynomial $p(X)$ of degree D over a field $\mathbb{F}$ can have at most D distinct roots, i.e., at most D elements $\alpha \in \mathbb{F}$ can satisfy $p(\alpha) = 0$.

This fact implies that the encodings of two distinct messages m and m' by $\mathsf{RS}_{\mathbb{F},S,n,k}$ must differ in more than $n - k$ locations.[3] The minimum distance of the RS code is thus at least $n - k + 1$. It is in fact equal to $n - k + 1$: e.g., consider encodings of the messages corresponding to the zero polynomial and the polynomial $\prod_{i=1}^{k-1}(X - \alpha_i)$. A minimum distance of $(n - k + 1)$ is the best possible for a code of dimension k, making RS codes optimal in this regard.

2.1 Correcting errors in RS codewords

Why is the above large distance useful? If at most $(n - k)/2$ errors corrupt the evaluations of a polynomial $f(X)$, then the encoding of $f(X)$ remains the best fit of the corrupted data in terms of agreements than the encoding of any other polynomial $g(X)$ of degree less than k. Thus, one can hope to recover $f(X)$ and the correct message even in the presence of $(n - k)/2$ errors. However, it is not immediate how to isolate the errors and recover $f(X)$ *efficiently*. We do not know the locations of the errors, and trying all possibilities will need exponential time.

Back in 1960, even before polynomial running time was formalized as the notion underlying efficient algorithms, Peterson [15] described a polynomial time algorithm to solve the above problem. We now describe the high level idea behind a different algorithm, due to Welch and Berlekamp [18], following the elegant description in [3].

Assume that the encoding $(f(\alpha_1), \ldots, f(\alpha_n))$ of a polynomial $f(X)$ was transmitted, and we receive a corrupted version $(y_1, y_2, \ldots, y_n)$, where the set $E = \{i : y_i \neq f(\alpha_i)\}$ of error locations has size at most $(n - k)/2$. Suppose we miraculously *knew* the set E. Then we could simply discard the y_i's corresponding to these locations, and interpolate $f(X)$ through the rest of the correct data points. We will have at least $(n + k)/2 \geqslant k$ locations, so interpolation will uniquely identify $f(X)$.

Error-location via bivariate interpolation. The key is thus a clever method to locate the set E of error locations quickly. To motivate this, let us cast the problem geometrically as an equivalent *noisy curve fitting* problem. We are given n points (α_i, y_i), $i = 1, 2, \ldots, n$, in the "plane" $\mathbb{F} \times \mathbb{F}$. The goal is to find the (unique) curve with equation $Y - f(X) = 0$ where degree$(f) < k$ that passes through all but $e \leqslant (n - k)/2$ locations i (namely those in E). If there was no noise, fitting a curve through all n points is easy — it is just polynomial interpolation. We know $Y - f(X)$ passes through $n - e$ points, so we can get a curve that passes through all the points by fitting vertical lines through the error points along with the curve $Y - f(X) = 0$; see Figure 2. Algebraically, if we define

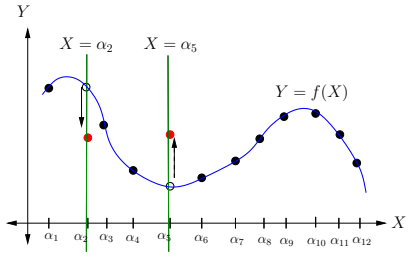

Figure 2: A RS codeword (polynomial $f(X)$ evaluated on points $\alpha_1, \alpha_2, \ldots, \alpha_{11}$); its corruption by two errors (at locations α_2 and α_5); and illustration of the curve fitting through the noisy data using correct curve and "error-locator lines."

$$P(X, Y) = (Y - f(X)) \prod_{i \in E} (X - \alpha_i) \ , \qquad (1)$$

then the curve $P(X, Y) = 0$ passes through all the points, i.e., $P(\alpha_i, y_i) = 0$ for every i. The second factor in the expression (1) is called the error-locator polynomial, which has the error locations as its roots.

Given $P(X, Y)$, one can find its factors (which can be done efficiently) and thus read off the message polynomial $f(X)$ from the $Y - f(X)$ factor. But how do we find $P(X, Y)$? Finding $P(X, Y)$ in its factored form (1) is begging the question, but note that we can also write $P(X, Y)$ in the form $P(X, Y) = D_1(X)Y - N_1(X)$ where degree$(D_1) \leqslant e \leqslant (n - k)/2$ and degree$(N_1) < e + k \leqslant (n + k)/2$.

Knowing such a polynomial exists, we can try to find a nonzero bivariate polynomial $Q(X, Y) = D_2(X)Y - N_2(X)$ satisfying:

1. degree$(D_2) \leqslant (n - k)/2$ and degree$(N_2) < (n + k)/2$.
2. $Q(\alpha_i, y_i) = 0$ for $i = 1, 2, \ldots, n$.

This can be done by setting up a system of linear equations over $\mathbb{F}$ with unknowns being the coefficients of $D_2(X)$ and $N_2(X)$, and n homogeneous constraints $Q(\alpha_i, y_i) = 0$. We have called the polynomial $Q(X, Y)$ since we can't assume that the solution will in fact equal $P(X, Y)$ (there may be multiple nonzero solutions to the above system). Solving this linear system can certainly be done in polynomial time, and also admits fast, practical methods.

One can prove, using elementary algebra, that when the number of errors $e \leqslant (n - k)/2$, *any* interpolated $Q(X, Y)$ satisfying the above two conditions *must* have $P(X, Y)$ as a factor, and be of the form $P(X, Y)A(X)$ for some nonzero (possibly constant) polynomial $A(X)$. The intuitive reason is that since there are so few errors in the data compared to the curve $Y - f(X) = 0$ that $P(X, Y) = 0$ is the most economical way to fit all the data points. The formal proof proceeds by considering the polynomial $R(X) \stackrel{\text{def}}{=} Q(X, f(X))$, and arguing it must be identically zero since (i) it has at least $(n + k)/2$ roots (namely all α_i's for which $f(\alpha_i) = y_i$) and

[3] If not, $\mathsf{RS}_{\mathbb{F},S,n,k}(m) - \mathsf{RS}_{\mathbb{F},S,n,k}(m')$, which corresponds to the evaluation of the polynomial $\sum_{i=0}^{k-1}(m_i - m'_i)X^i$ of degree at most $k - 1$, has at least k zeroes — a contradiction.

(ii) it has degree less than $(n+k)/2$ (by design of $Q(X,Y)$). Thus, $Q(X,Y)$ also has $Y - f(X)$ as a factor, and we can recover $f(X)$ by factoring $Q(X,Y)$. (The actual task here is easier than general bivariate polynomial factorization, and admits near-linear time algorithms.)

2.2 List Decoding Reed-Solomon codes

We now turn to list decoding of Reed-Solomon codes. The setup is as before: given n points $(\alpha_i, y_i) \in \mathbb{F}^2$, find polynomials $f(X)$ of degree less than k such that $f(\alpha_i) \neq y_i$ for at most e locations i. The difference is that now $e \gg (n-k)/2$, and so there may be more than one such polynomial $f(X)$ that the decoder needs to output.

Before delving into the algorithms, we pause to consider how large a number of errors e one can target to correct. Clearly, we need the guarantee there will be only be a few (say, at most polynomially many in n) solution polynomials $f(X)$ (or else there is no hope for the decoder to output all of them in polynomial time). Using the fact that encodings of any two polynomials differ in more than $(n-k)$ locations, it can be shown (via the so-called "Johnson bound") that for $e \leqslant n - \sqrt{kn}$, the number of solutions (called the *list size*) is guaranteed to be polynomially small. Whether one can prove a polynomial list size bound for certain RS codes for even larger e remains a key open question.

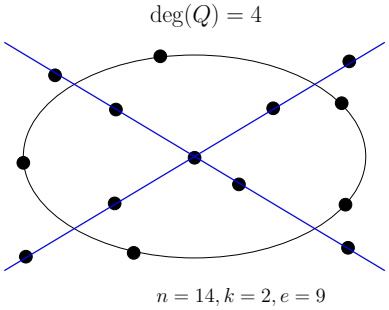

Figure 3: Illustration of list decoding of RS code evaluating lines over the points $-7, -5, -4, \ldots, 4, 5, 6, 7$. The two lines are recovered as factors of a degree 4 interpolated curve through all the points.

We now describe the idea underlying Sudan's breakthrough algorithm for list decoding RS codes [17]. Recall that we want to solve a noisy curve fitting problem, and output all curves $Y - f(X) = 0$ with $\deg(f) < k$ that pass through $n-e$ or more of the n points (α_i, y_i). For $e \leqslant (n-k)/2$, the Berlekamp-Welch algorithm interpolated a bivariate polynomial $Q(X,Y)$ of a very specific format through all the n points. Sudan's idea for $e > (n-k)/2$ was to interpolate/fit a *general* nonzero curve $Q(X,Y) = 0$ of just high enough "degree" (so that its existence is guaranteed) through *all* the n points. Fitting such a curve can be done efficiently by solving a system of homogeneous linear equations to determine the coefficients of $Q(X,Y)$.

For the Berlekamp-Welch algorithm, arguing that $Y - f(X)$ was a factor of $Q(X,Y)$ followed from the very special structure of $Q(X,Y)$. In the list decoding case, Sudan exploited special properties of intersections of curves of the form $Y - f(X)$ with any interpolated bivariate polynomial $Q(X,Y)$ with appropriate degree constraints. Informally, Sudan's idea is that given the strong degree constraint on $Q(X,Y)$, every curve $Y - f(X) = 0$ with $\deg(f) < k$ that picks up at least $n-e$ of points must be "used" by the interpolated curve in meeting the requirement to pass through all n points. As an example, in Figure 3, the goal is to find all lines (i.e. we have $k = 2$) that pass through all but $e = 9$ of the $n = 14$ input points (there are two such lines, marked in the figure as $L_1(X,Y)$ and $L_2(X,Y)$). There are enough degrees of freedom in the equation of a degree 4 curve so that one can fit a degree 4 curve through any set of 14 points. The figure illustrates one such curve, which is the product of the two lines with an "ellipse" $E(X,Y)$. (Note that the total degree of $Q(X,Y)$ is 4.) Further, we see that the two relevant lines pop out as factors. This is not a coincidence, and every degree 4 curve passing through the 14 points must have these two lines as factors. The reason: if a line is not a factor, then it can intersect a degree 4 curve in at most 4 points. Since each of these lines intersects any interpolated curve in at least 5 points, it must be a factor.

More formally, the "degree" measure of the interpolated polynomial $Q(X,Y)$ will be the $(1, k-1)$-degree, which is defined as the maximum of $i + (k-1)j$ over all monomials X^iY^j that occur with a nonzero coefficient in $Q(X,Y)$. Let D denote the $(1, k-1)$ degree of $Q(X,Y)$. Generalizing the above argument for lines, if a curve $Y - f(X) = 0$ with $\deg(f) < k$ passes through more than D points, then $Y - f(X)$ must be a factor of $Q(X,Y)$. With a counting argument, one can show that a $(1, k-1)$-degree D of $\sqrt{2kn}$ suffices to fit a nonzero curve. Together, this leads to an algorithm that can correct $n - \sqrt{2kn}$ errors, or a fraction $1 - \sqrt{2R}$ of errors as a function of the rate R.

For low rates, this algorithm enables recovery even in settings when noise overwhelms correct data, and close to 100% of the symbols may be in error! This feature set the stage for several powerful applications in cryptography and complexity theory. However, the algorithm does not give any improvement over the $(1-R)/2$ error-fraction corrected by traditional algorithms for rates $> 1/3$, and also falls short of the $1 - \sqrt{R}$ radius suggested by the combinatorial bounds.

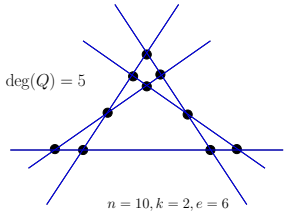

Figure 4: Illustration of Guruswami-Sudan algorithm for list decoding RS codes. The lines are recovered as factors of a degree 5 curve that passes through each point twice.

We now turn to the improved algorithm correcting a fraction $1 - \sqrt{R}$ of errors due to Guruswami and Sudan [10]. The key new idea is to insist that the interpolated polynomial $Q(X,Y)$ have *multiple* zeroes at each of the n points. To explain this, we attempt a high level geometric description. Consider the example in Figure 4 with $n = 10$ points, the goal being to output all lines that pass through at least $n-e = 4$ points. This example cannot be solved by Sudan's algorithm. Indeed, since there are five solution lines, if they

are all factors of some interpolated curve, the curve must have degree at least 5. However, there is no guarantee that an arbitrary degree 5 curve through the points must have every line passing through 4 of the points as a factor (the line has to pass through 6 points to guarantee this). Let C^* be the degree 5 surve that is the product of the five solution lines. As mentioned above, if we interpolate a degree 5 curve through the 10 points, we will in general *not* get C^* as the solution. However, notice a special property of C^* — it passes through each point *twice*; a priori there is no reason to expect that an interpolated curve will have this property. The Guruswami-Sudan idea is to *insist* that the interpolation stage produce a degree 5 curve with zero of multiplicity at least 2 at each point (i.e., the curve intersects each point twice). One can then argue that each of the five lines must be a factor of the curve. In fact, this will be the case for degree up to 7. This is because the intersection of each of these lines with the curve, counting multiplicities, is at least $4 \times 2 = 8$ which is greater than the degree of the curve. Finally, one can always fit a degree 7 curve passing through any 10 points twice (again by a counting argument). So by insisting on multiplicities in the interpolation step, one can solve this example.

In general, the interpolation stage of the Guruswami-Sudan list decoder finds a polynomial $Q(X, Y)$ that has a zero of multiplicity w for some suitable integer w at each (α_i, y_i).[4] Of course this can always be accomplished with a $(1, k-1)$-degree that is a factor w larger (by simply raising the earlier interpolation polynomial to the w'th power). The key gain is that the required multiplicity can be achieved with a degree only about a factor $w/\sqrt{2}$ larger. The second step remains the same, and here each correct data point counts for w zeroes. This $\sqrt{2}$ factor savings translates into the improvement of ρ from $1 - \sqrt{2R}$ to $1 - \sqrt{R}$. See Figure 5 for a plot of this trade-off between rate and fraction of errors, as well as the $(1 - R)/2$ trade-off of traditional unique decoding algorithms. Note that we now get an improvement for *every* rate. Also plotted are the information-theoretic limit $1 - R$, and the Parvaresh-Vardy improvement for low rates that we will discuss shortly.

Soft decoding. We now comment on a further crucial benefit of the multiplicities idea which is relevant to potential practical applications of list decoding. The multiplicities can be used to encode the relative importance of different codeword positions, using a higher multiplicity for symbols whose value we are more confident about, and a lower multiplicity for the less reliable symbols that have lower confidence estimates. In practice, such confidence estimates (called "soft inputs") are available in abundance at the input to the decoder (e.g. from demodulation of the analog signal). This has led to a promising *soft-decision* decoder for RS codes with good coding gains in practice [13], which was adopted in the Moonbounce program to improve communication between Ham radio operators who bounce radio signals off the moon to make long distance contacts.

3. FOLDED REED-SOLOMON CODES

We now discuss a variant of RS codes called *folded Reed-Solomon codes* (henceforth folded RS codes), which will let us approach the optimal error-correction radius of a fraction $1 - R$ of errors. The codewords in the folded RS code will be in one-to-one correspondence with RS codewords. We begin with an informal description. Consider the RS codeword corresponding to the polynomial $f(X)$ that is evaluated at the points $x_0, x_1, \ldots, x_{n-1}$ from $\mathbb{F}$, as depicted by the codeword on top in Figure 6. The corresponding codeword in the folded RS code (with *folding parameter* of $m = 4$) is obtained by juxtaposing together 4 consecutive symbols on the RS codeword as shown at the bottom of Figure 6. In other words, we think of the RS code as a code over a larger alphabet (of 4 times the "packet size") and of block length 4 times smaller. This repackaging reduces the number of error-patterns one has to handle. For example, if we are targeting correcting errors in up to a 1/4 fraction of the new larger symbols, then we are no longer required to correct the error pattern corresponding to the (pink) shaded columns in Figure 6 (whereas the same error pattern over the original symbols needs to be taken care of in the Reed-Solomon case).

We would like to stress on a subtle point here: in the worst-case error model, the "atomic" unit of error is an alphabet character. This was used crucially in the example above to rule out an error pattern that was admissible for the smaller alphabet. For the reader who might we worried that this constitutes "cheating," e.g. what if one collapses the entire RS codeword into one large symbol, we offer two counter-points. First, since we will only use a *constant* folding parameter, the increase in alphabet size from that of RS codes is modest. Second, in Section 4, we will see how to convert folded RS codes into codes over alphabets whose

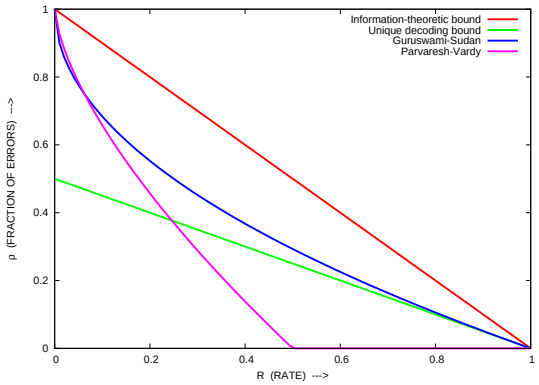

Figure 5: Rate vs. error-correction radius for RS codes. The optimal trade-off is also plotted, as is the Parvaresh-Vardy's improvement over RS codes.

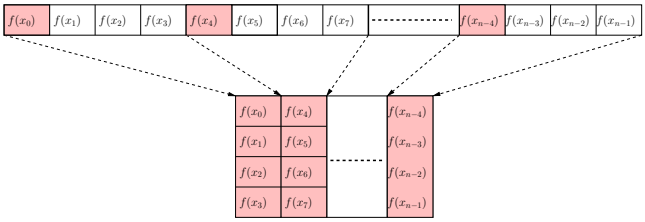

Figure 6: Folding of the Reed-Solomon code with parameter $m = 4$. Each column represents an alphabet character.

[4]We skip formalizing the notion of multiple zeroes in this description, but this follows along standard lines and we refer the interested reader to [10] for the details.

size *does not depend at all* on the block length, while still maintaining similar error correction properties!

We now formally define the folded RS code. Let the non-zero elements of the field $\mathbb{F}$ be generated by γ, i.e. every non-zero element is γ^i for some $0 \leqslant i \leqslant |\mathbb{F}| - 2$. (Such a γ always exists for any finite field $\mathbb{F}$.) Let $m \geqslant 1$ be the *folding parameter* and let n be an integer that is divisible by m and $n \leqslant |\mathbb{F}| - 1$. The folded RS encoding (with folding parameter m) of the message polynomial $f(X)$ has as its j'th symbol for $0 \leqslant j < n/m$, the m-tuple $(f(\gamma^j m), f(\gamma^{jm+1}), \ldots, f(\gamma^{jm+m-1}))$.

The block length of these codes is $N = n/m$. The rate of the code remains k/n, since the folding operation does not introduce any further redundancy.

The folding operation does restrict the error patterns that one needs to worry about. But how can one actually exploit this in a decoding algorithm and manage to correct a larger fraction of errors compared to the unfolded RS codes? We turn to this question next.

3.1 Multivariate Decoding

Recall the two step Guruswami-Sudan (GS) algorithm. First, we interpolate a bivariate polynomial $Q(X, Y)$ through the points $(\alpha_i, y_i) \in \mathbb{F}^2$. Then in the second step, we factorize the bivariate polynomial and retain factors of the form $Y - f(X)$, where $f(X)$ is a polynomial of degree less than k (there might be factors that are not linear in Y: we ignore them). Let us recast the second step in an equivalent description, which we be useful later. In particular, consider the *univariate* polynomial $R_X(Y)$ equivalent to $Q(X, Y)$, where the coefficients themselves are *polynomials* in indeterminate X with their own coefficients from $\mathbb{F}$: given the polynomial $Q(X, Y)$ one can compute $R_X(Y)$ by collecting all the coefficients of the same power of Y together. (For example, if $Q(X, Y) = (Y - (X - 1))(Y^2 + X^3)$ then $R_X(Y) = a_3 \cdot Y^3 + a_2 \cdot Y^2 + a_1 \cdot Y + a_0$, where $a_3 = 1, a_2 = -X + 1, a_1 = X^3$ and $a_0 - X^4 + X^3$.) Now note that $Y - f(X)$ is a factor of $Q(X, Y)$ if and only if $f(X)$ is a *root* of the univariate polynomial $R_X(Y)$, that is, the polynomial $R_Y(f(X))$ is the same as the zero polynomial. (In the example, $Y - (X - 1)$ divides $Q(X, Y)$ and $R_X(X - 1) \equiv 0$.)

Let us now return to problem of list decoding folded RS code with $m = 4$. Given the received word whose i'th symbol (for $0 \leqslant i < N$) is $(y_{i,0}, y_{i,1}, y_{i,2}, y_{i,3})$, we need to output all the close-by folded RS codewords. To motivate the idea behind the algorithm, for the time being assume that the transmitted codeword was from the so-called *interleaved* RS code of *order* 4. Any codeword in such a code will have as its i'th symbol $(0 \leqslant i \leqslant N - 1)$ the 4-tuple $(f(\gamma^{4i}), f_1(\gamma^{4i}), f_2(\gamma^{4i}), f_3(\gamma^{4i}))$, where $f(X), f_1(X), f_2(X)$ and $f_3(X)$ are some polynomials of degree at most $k - 1$. We remark that the folded RS code is a subcode[5] of the interleaved RS code where $f_j(X) = f(\gamma^j X)$ for $1 \leqslant j \leqslant 3$.

Given the setup above, the first thing to explore is whether one can generalize the GS algorithm to the setting of interleaved RS codes. To see one such generalization, note that RS codes are interleaved RS codes of order 1. The GS algorithm interpolated a non-zero bivariate polynomial $Q(X, Y)$ in thus case. Thus, for an interleaved RS code of order 4, a natural attempt would be to compute a non-zero 5-variate polynomial $Q(X, Y, Z, U, W)$, where (as before) Y is a placeholder for $f(X)$ and (this part is the generalization) Z, U and W are placeholders for $f_1(X), f_2(X)$ and $f_3(X)$ respectively. For the next step of root finding, we compute the 4-variate polynomial $R_X(Y, Z, U, W)$ that is equivalent to $Q(X, Y, Z, U, W)$. Now the hope would be to find out all the tuples (Y, Z, U, W) that make $R_X(Y, Z, U, W)$ vanish and that the required tuple $(f(X), f_1(X), f_2(X), f_3(X))$ is one of them. The latter condition can in fact be satisfied, but the trouble is that the number of tuples that make R_X zero could be very large (growing exponentially in n). To see intuitively what goes wrong, recall that in the Guruswami-Sudan setting, we had one unknown Y and one constraint $R_X(Y) = 0$. However, in the interleaved RS setting, we have *four* unknowns Y, Z, U, W but *only one* constraint $R_X(Y, Z, U, W) = 0$. This essentially means that three of the four unknowns are unconstrained and can thus be almost any polynomial of degree less than k.

The generalization above (and similar ideas) were tried out in a few works, but could not decode beyond the $1 - \sqrt{R}$ radius. Finally, seven years after the GS algorithm was published, Parvaresh and Vardy [14] had an ingenious idea: force the polynomials $f_1(X), f_2(X)$ and $f_3(X)$ to be related to $f(X)$! In particular, they only look at the subcode of interleaved RS code where $f_j(X) = (f_{j-1}(X))^d \mod (E(X))$ for $1 \leqslant j \leqslant 3$ (we set $f_0(X) = f(X)$), for some positive integer parameter d and an irreducible polynomial $E(X)$. The reason we compute the modulus using an irreducible polynomial is that the relationships between these polynomials translate to the following relationships between their corresponding placeholders: $Z = Y^d$, $U = Y^{d^2}$ and $W = Y^{d^3}$. In other words, we gain *three new* constraints on the four variables Y, Z, U, W. Together with the interpolation constraint $R_X(Y, Z, U, W) = 0$, this restores equality in the number of unknowns and the number of constraints. This in turn implies that the number of possible solutions is polynomially bounded. (There are some steps involved to obtain this conclusion but they are mostly all "low-level details.") Further, this method not only establishes a bound on the number of solutions, but also gives a polynomial time algorithm to find these solutions. To see this, note that given the three new constraints, we are looking for roots of the *univariate* polynomial $R_X(Y, Y^d, Y^{d^2}, Y^{d^3})$, which can be accomplished by well known polynomial time algorithms [1].

Finally, let us return to the problem of list decoding folded RS codes with $m = 4$. In folded RS codes also, we have the property that $f_j(X)$ is related to $f(X)$ for $1 \leqslant j \leqslant 3$. In fact, combining a couple of well known results in finite fields, in [9] we show that $f(\gamma X) = (f(X))^{|\mathbb{F}|-1} \mod (E'(X))$, where $E'(X) = X^{|\mathbb{F}|-1} - \gamma$ is an irreducible polynomial. We remark that the irreducible polynomial $E(X)$ in the Parvaresh-Vardy (henceforth, PV) codes only has some degree requirements. In particular, our restriction on $E'(X)$ is stricter and thus, folded RS codes are a special case of PV codes. However, we are not done yet. Until now all we can claim is that folded RS code of rate R with folding parameter $m = 4$ can be list decoded from the same fraction of errors as the corresponding PV codes, which happens to be $1 - \sqrt[5]{(4R)^4}$. We have $4R$ appearing instead of R because the rate of the PV code is 1/4'th the rate of the original RS code, since the encoding of the message $f(X)$ now consists of the evaluations of *four* polynomials instead of just those of $f(X)$. Next we expand on our other main idea which

[5]This is the same as looking at an appropriate subset of messages.

"compresses" the PV encoding to avoid this rate loss, and enables correcting close to a fraction $1 - R$ of errors.

3.2 The Final Piece

The improvement over Parvaresh-Vardy comes from comparing apples to oranges. In particular, till now we have seen that the folded RS code with folding parameter 4 is a special case of the PV code of order 4. Instead let us compare the folded RS code with a PV code of *smaller* order, say 2. It

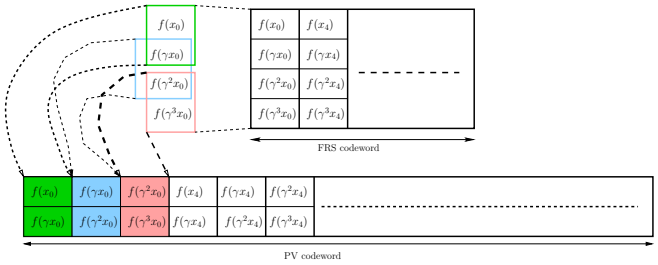

Figure 7: The correspondence between a folded RS code (with $m = 4$ and $x_i = \gamma^i$) and the PV code (of order $s = 2$) evaluated over $\{1, \gamma, \gamma^2, \gamma^4, \ldots, \gamma^{n-4}, \ldots, \gamma^{n-2}\}$. The correspondence for the first block in the folded RS codeword and the first three blocks in the PV codeword is shown explicitly in the left corner of the figure.

turns out that the folded RS code with folding parameter 4 are compressed forms of certain specific PV codes of order 2 while containing the same amount of information! In particular, as in Figure 7 compare the folded RS codeword with the PV code of order 2 (where the polynomial $f(X)$ is evaluated at the points $\{1, \gamma, \ldots, \gamma^{n-1}\} \setminus \{\gamma^3, \gamma^7, \ldots, \gamma^{n-1}\}$). We find that in the PV encoding of f, for every $0 \leqslant i \leqslant n/m - 1$ and every $0 < j < m - 1$, $f(\gamma^{mi+j})$ appears exactly twice (once as $f(\gamma^{mi+j})$ and another time as $f_1(\gamma^{-1}\gamma^{mi+j})$), whereas it appears only once in the folded RS encoding. In other words, the information contained in one symbol in the folded RS codeword (which is worth *four* elements from $\mathbb{F}$) is repeated over three symbols in the PV codeword (which is worth *six* elements from $\mathbb{F}$). This implies that even though both the folded RS codeword and the PV codeword has exactly the same information, the folded RS codeword is compressed by a factor of 3/2. This in turn bumps up the rate of the folded RS code by the same factor. Hence, we can list decode folded RS codes with folding parameter 4 and rate R from a fraction $1 - \sqrt[5]{(8R/3)^4}$ of errors.

Thus, our list decoding algorithm for folded RS with folding parameter m can be modularly defined as follows: unfold the received word for the appropriate PV code of order $s \leqslant m$ and then run the Parvaresh-Vardy list decoder on this unfolded received word. It turns out that this list decoder can correct such a folded RS code of R from up to $1 - \sqrt[s+1]{\left(\frac{mR}{m-s+1}\right)^s}$ fraction of errors. By picking m to be (somewhat) larger than s and picking s to be sufficiently large (in terms of $1/\varepsilon$), leads to the following result.

THEOREM 1. *For every $\varepsilon > 0$ and $0 < R < 1$, there is a family of folded RS codes that have rate at least R and which can be list decoded up to a fraction $1 - R - \varepsilon$ of errors in time $(N/\varepsilon^2)^{O(\varepsilon^{-1}\log(1/R))}$ where N is the block length of the code. The alphabet size of the code is $(N/\varepsilon^2)^{O(1/\varepsilon^2)}$.*

We note that the time complexity has an undesirable dependence on ε, with $1/\varepsilon$ in the exponent. Improving this bound remains a challenging open question.

4. DECODING OVER SMALL ALPHABETS

So far we have discussed codes over large alphabets. For example, folded RS codes of rates R that can be list decoded from $1-R-\varepsilon$ fraction of errors needs alphabet size of roughly $n^{O(1/\varepsilon^2)}$, where n is the block length of the code. This large alphabet size can be a shortcoming. Next, we discuss know techniques that help us "reduce" the alphabet size.

We start with perhaps the most natural small alphabet: $\{0, 1\}$. For codes defined over this alphabet (also called *binary* codes), it turns out that to list decode from ρ fraction of errors the best possible rate is $1 - H(\rho)$, where $H(x) = -x \log_2 x - (1 - x) \log_2(1 - x)$ is the entropy function. Two remarks are in order. First, the rate $1 - H(\rho)$ is much smaller than the rate of $1-\rho$ that folded RS codes can achieve. (It turns out that to attain a rate of $1 - \rho - \varepsilon$, the alphabet size needs to be at least $2^{1/\varepsilon}$; more on this later in the section.) Second, as shown in Shannon's seminal paper [16], the quantity $1 - H(\rho)$ is *exactly* the same as the best possible rate (aka "capacity") that can be achieved in the binary symmetric channel BSC_ρ. Thus, list decoding can bridge the traditionally perceived gap between the Shannon stochastic model and the Hamming worst-case model.

We "transfer" our result for folded RS codes to a result for binary codes via a natural method for composing together codes called *code concatenation*, proposed by Forney over 40 years ago. Thanks to the powerful algorithms for decoding folded RS codes, we can use this approach to achieve a certain trade-off called the *Zyablov bound* between rate and fraction of errors corrected [9]. In a subsequent work, using another generalization of code concatenation, we improved the tradeoff to the *Blokh-Zyablov bound* [8]. Figure 8 plots these two tradeoffs along with the best possible tradeoff (list-decoding capacity). There is a large gap between

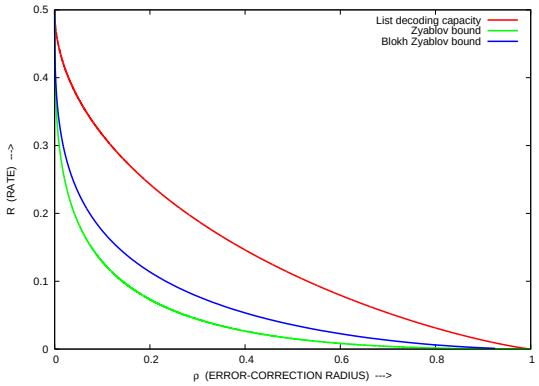

Figure 8: Error-correction radius ρ of our algorithms for binary codes plotted against the rate R. The best possible trade-off, i.e., list decoding capacity, is $\rho = H^{-1}(1 - R)$, and is also plotted.

the list-decoding capacity and the best bound known to be achievable with efficient algorithms. Closing this gap remains a central and extremely challenging open question.

We now briefly mention how we resolve the large alphabet issue that was raised in Section 3. When the folding parameter of the folded RS code is a constant (as in in Theorem 1), the number of bits needed to represent a symbol from the alphabet is no larger than roughly the logarithm of the block length of the folded RS code. This is small enough to use the idea of code concatenation mentioned above to reduce the alphabet. In order to maintain the optimal trade-off between rate and fraction of errors list decoded, we need to combine concatenation with an approach to redistribute symbols using *expander graphs* [6]. This leads to codes of rate R that can be list decoded from a fraction $1 - R - \varepsilon$ of errors over an alphabet of size $2^{1/\varepsilon^4}$, which is close to the lower bound of $2^{1/\varepsilon}$ mentioned earlier.

5. CONCLUDING REMARKS

First, we mention some related work that appeared subsequent to the initial publication of our result. Further work on extending the results in this article to to the framework of *algebraic-geometric codes* has been done in [7, 5]. A surprising application of the ideas in the Parvaresh-Vardy list decoder is the construction of *randomness extractors* by Guruswami, Umans and Vadhan [11]. Randomness extractors convert input from a weakly random source into an almost perfectly random string, and have been intensively studied in theoretical computer science for over 15 years. This recent extractor is almost optimal in all parameters, while having a simple, self-contained description and proof.

Even though the work presented in this article makes good progress in our theoretical understanding of list decoding, applying these ideas into practice requires further innovation. We conclude by posing two practical challenges.

The first challenge is specific to making the list decoding algorithms for folded RS codes more practical. Recall that the algorithm involved an interpolation step and a "root-finding" step. There are fast heuristic approaches for the latter step that could be used in practice. The interpolation step, however, seems too inefficient for practical purposes due to the large size of the linear systems that need to be solved. It would be very useful to have more efficient algorithms for this step. We note that such improvements for the Guruswami-Sudan algorithm have been obtained.

The second challenge is more general. Codes have found numerous practical applications in domains such as communication and data storage. Despite its promise and the recent advances, list decoding has not yet found widespread use in practical systems (though as mentioned earlier, the Moonbounce program does use the multiplicities based list decoder). One possible reason could be that the previous list decoding algorithms do not provide much gain for the high rate regime over traditional unique decoding algorithms. However, this is no longer a concern — we now have algorithms that obtain much better theoretical bounds in this regime. Further, folded RS codes are very similar to RS codes that are ubiquitous in practice. Hopefully in the near future, list decoding will be used more widely in practical systems.

Acknowledgments. The research described here was supported in part by NSF award CCF-0343672 and fellowships from the Sloan and Packard Foundations. We thank Ronitt Rubinfeld for several valuable comments on an earlier draft of this paper.

6. REFERENCES

[1] E. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24:713–735, 1970.

[2] P. Elias. List decoding for noisy channels. *Technical Report 335, MIT Research Lab of Electronics*, 1957.

[3] P. Gemmell and M. Sudan. Highly resilient correctors for multivariate polynomials. *Information Processing Letters*, 43(4):169–174, 1992.

[4] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proc. of the 21st ACM Symp. on Theory of Computing*, pages 25–32, 1989.

[5] V. Guruswami. Artin automorphisms, cyclotomic function fields, and folded list-decodable codes, 2008. Manuscript.

[6] V. Guruswami and P. Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Trans. on Info. Theory*, 51(10):3393–3400, 2005.

[7] V. Guruswami and A. Patthak. Correlated Algebraic-Geometric codes: Improved list decoding over bounded alphabets. *Mathematics of Computation*, 77(261):447–473, January 2008.

[8] V. Guruswami and A. Rudra. Better binary list-decodable codes via multilevel concatenation. In *Proc. of the 11th Intl. Workshop on Randomization and Computation*, pages 554–568, 2007.

[9] V. Guruswami and A. Rudra. Explicit codes achieving list decoding capacity: Error-correction up to the Singleton bound. *IEEE Trans. on Info. Theory*, 54(1):135–150, 2008. Preliminary version in *STOC'06*.

[10] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Trans. on Info. Theory*, 45:1757–1767, 1999.

[11] V. Guruswami, C. Umans, and S. P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. In *22nd IEEE Conf. on Computational Complexity*, pages 96–108, 2007.

[12] R. W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29:147–160, April 1950.

[13] R. Koetter and A. Vardy. Algebraic soft-decision decoding of Reed-Solomon codes. *IEEE Transactions on Information Theory*, 49(11):2809–2825, 2003.

[14] F. Parvaresh and A. Vardy. Correcting errors beyond the Guruswami-Sudan radius in polynomial time. In *Proc. of the 46th IEEE Symposium on Foundations of Computer Science*, pages 285–294, 2005.

[15] W. W. Peterson. Encoding and error-correction procedures for Bose-Chaudhuri codes. *IEEE Transactions on Information Theory*, 6:459–470, 1960.

[16] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

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

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

[19] J. M. Wozencraft. List Decoding. *Quarterly Progress Report, Research Laboratory of Electronics, MIT*, 48:90–95, 1958.