Efficient List Decoding of Explicit Codes with Optimal Redundancy

Atri Rudra

Department of Computer Science and Engineering
University at Buffalo, State University of New York
Buffalo, 14260, USA.
atri@cse.buffalo.edu
http://www.cse.buffalo.edu/~atri

Abstract. Under the notion of list decoding, the decoder is allowed to output a small list of codeword such that the transmitted codeword is present in the list. Even though combinatorial limitations on list decoding had been known since the 1970's, there was essentially no algorithmic progress till the breakthrough works of Sudan [14] and Guruswami-Sudan [11] in the mid to late 1990's. There was again a lull in algorithmic progress till a couple of recent papers [12,8] closed the gap in our knowledge about combinatorial and algorithmic limitations of list decoding (for codes over large alphabets). This article surveys these latter algorithmic progress.

1 Introduction

Under the list decoding problem (introduced in [1,16]), given a code $C \subseteq \Sigma^n$, an error parameter $0 \leqslant \rho \leqslant 1$ and a received word $\mathbf{y} \in \Sigma^n$; the decoder should output all codewords in C that are within Hamming distance ρn of $\mathbf{y}$. Suppressing the motivation for considering such an error recovery model for the time being, let us consider the following natural trade-off: Given that one wants to correct ρ fraction of errors via list decoding, what is the maximum rate R that a code can have?

Before we address this question, let us formally define the notion of list decoding we will consider in this survey. For a real $0 \leqslant \rho \leqslant 1$ and an integer $L \geqslant 1$, we will call a code $C \subseteq \Sigma^n$ to be (ρ, L)-list decodable if for every received word $\mathbf{y} \in \Sigma^n$, $|\{c \in C | \Delta(c, \mathbf{y}) \leqslant \rho n\}| \leqslant L$ where $\Delta(c, \mathbf{y})$ denotes the Hamming distance between the vectors c and $\mathbf{y}$. Note that the problem is interesting only when L is small: in this survey L is considered to be small if it is polynomially bounded in n.

Using a standard random coding argument it can be show that there exists $(\rho, O(1/\varepsilon))$ list decodable codes over alphabets of size q with rate $R \geqslant 1 - H_q(\rho) - o(1)$ where $H_q(x) = -x \log_q \left(\frac{x}{q-1}\right) - (1-x)\log_q(1-x)$ is the q-ary entropy function (cf. [17,2]). Further, a simple counting argument shows that R must be at most $1 - H_q(\rho)$ (for $R > 1 - H_q(\rho)$ the list size L needs to be super-polynomial in n). In other words, the maximum fraction of errors that can be

corrected (via list decoding) using a rate R code (or the *list decoding capacity*), is given by the trade-off $H_q^{-1}(1-R)$. For $q = 2^{\Omega(1/\varepsilon)}$, $H_q^{-1}(1-R) \geqslant 1-R-\varepsilon$ (cf. [13]). In other words, for large enough alphabets, the list decoding capacity is $\rho_{\mathrm{cap}}(R) = 1-R$.

Now is a good time to compare the list decoding capacity with what can be achieved with "usual" notion of decoding for the worst-case noise model (called *unique decoding*), where the decoder has to always output the transmitted word. Note that list decoding is a relaxation where the decoder is allowed to output a list of codewords (with the guarantee that the transmitted codeword is in the list). It is well known that unique decoding can only correct up to half the minimum distance of the code, which along with the Singleton bound implies the following limit on the fraction of errors that can be corrected: $\rho_U(R) = (1-R)/2$. In other words, list decoding has the potential to correct *twice* as many errors than unique decoding.

However, in order to harness the real potential of list decoding, we need explicit codes along with efficient list decoding algorithms that can achieve the list decoding capacity. For this survey, a list decoding algorithm with a polynomial running time is considered to be efficient. (Note that this puts an *a priori* requirement that the worst case list size needs to be bounded by a polynomial in the block length of the code.) Even though the notion of list decoding was defined in the late 1950's, there was essentially no algorithmic progress in list decoding till the breakthrough works of Sudan [14] and Guruswami-Sudan [11] which can list decode Reed-Solomon codes up to the trade-off $\rho_{GS}(R) = 1-\sqrt{R}$. One can check that $\rho_{GS}(R) > \rho_U(R)$ for every rate R (with the gains being more pronounced for smaller rates). This fact lead to a spurt of research activity in list decoding including some surprising applications outside the traditional coding domain: see for example [15], [4, Chap. 12]. However, this result failed to achieve the list decoding capacity for *any* rate (with the gap being especially pronounced for larger rates).

The bound of ρ_{GS} resisted improvements for about seven years till in a recent breakthrough paper [12], Parvaresh and Vardy presented codes that are list-decodable beyond the $1-\sqrt{R}$ radius for low rates R. For any $m \geqslant 1$, they achieve the list-decoding radius $\rho_{\mathrm{PV}}^{(m)}(R) = 1 - \sqrt[m+1]{m^m R^m}$. For rates $R \to 0$, choosing m large enough, they can list decode up to radius $1 - O(R\log(1/R))$, which approaches the capacity $1-R$. However, for $R \geqslant 1/16$, the best choice of m is in fact $m = 1$, which reverts back to RS codes and the list-decoding radius $1-\sqrt{R}$. Building on works of Parvaresh and Vardy [12], Guruswami and Rudra [8] present codes that get arbitrarily close to the list decoding capacity $\rho_{\mathrm{cap}}(R)$ for every rate. In particular, for every $1 > R > 0$ and every $\varepsilon > 0$, they give *explicit* codes of rate R together with polynomial time list decoding algorithm that can correct up to a fraction $1-R-\varepsilon$ of errors. These are the first explicit codes (with efficient list decoding algorithms) that get arbitrarily close to the list decoding capacity for *any* rate. This article surveys the results of [12,8] and some of their implications for list decoding of explicit codes over small alphabets.

2 Folded Reed-Solomon Codes and the Main Results

The codes used in [8] are simple to state. They are obtained from the Reed-Solomon code by careful bundling together of codeword symbols (and hence, are called *folded Reed-Solomon codes*). We remark that the folded RS codes are a *special* case of the codes studied by [12]. However, for the ease of presentation, we will present all the results in terms of folded Reed-Solomon codes: this would be sufficient to highlight the algorithmic techniques used in [12]. See the survey [5] in these proceedings for a more detailed description of the Parvaresh-Vardy codes.

Consider a Reed-Solomon (RS) code $C = \mathsf{RS}_{\mathbb{F},\mathbb{F}^*}[n,k]$ consisting of evaluations of degree k polynomials over some finite field $\mathbb{F}$ at the set $\mathbb{F}^*$ of nonzero elements of $\mathbb{F}$. Let $q = |\mathbb{F}| = n+1$. Let γ be a generator of the multiplicative group $\mathbb{F}^*$, and let the evaluation points be ordered as $1, \gamma, \gamma^2, \ldots, \gamma^{n-1}$. Using all nonzero field elements as evaluation points is one of the most commonly used instantiations of Reed-Solomon codes.

Let $m \geqslant 1$ be an integer parameter called the *folding parameter*. For ease of presentation, it will assumed that m divides $n = q - 1$.

Definition 1 (Folded Reed-Solomon Code). *The m-folded version of the RS code C, denoted $\mathrm{FRS}_{\mathbb{F},\gamma,m,k}$, is a code of block length $N = n/m$ over $\mathbb{F}^m$. The encoding of a message $f(X)$, a polynomial over $\mathbb{F}$ of degree at most k, has as its j'th symbol, for $0 \leqslant j < n/m$, the m-tuple $(f(\gamma^{jm}), f(\gamma^{jm+1}), \cdots, f(\gamma^{jm+m-1}))$. In other words, the codewords of $C' = \mathrm{FRS}_{\mathbb{F},\gamma,m,k}$ are in one-one correspondence with those of the RS code C and are obtained by bundling together consecutive m-tuple of symbols in codewords of C.*

The following is the main result of Guruswami and Rudra.

Theorem 1 ([8]). *For every $\varepsilon > 0$ and $0 < R < 1$, there is a family of folded Reed-Solomon codes that have rate at least R and which can be list decoded up to a fraction $1 - R - \varepsilon$ of errors in time (and outputs a list of size at most) $(N/\varepsilon^2)^{O(\varepsilon^{-1}\log(1/R))}$ where N is the block length of the code. The alphabet size of the code as a function of the block length N is $(N/\varepsilon^2)^{O(1/\varepsilon^2)}$.*

The result of [8] also works in a more general setting called *list recovery*, which is defined next.

Definition 2 (List Recovery). *A code $C \subseteq \Sigma^n$ is said to be (ζ, l, L)-list recoverable if for every sequence of sets $S_1, \ldots, S_n$ where each $S_i \subseteq \Sigma$ has at most l elements, the number of codewords $c \in C$ for which $c_i \in S_i$ for at least ζn positions $i \in \{1, 2, \ldots, n\}$ is at most L.*

A code $C \subseteq \Sigma^n$ is said to (ζ, l)-list recoverable in polynomial time if it is $(\zeta, l, L(n))$-list recoverable for some polynomially bounded function $L(\cdot)$, and moreover there is a polynomial time algorithm to find the at most $L(n)$ codewords that are solutions to any $(\zeta, l, L(n))$-list recovery instance.

Note that when $l = 1$, $(\zeta, 1, \cdot)$-list recovery is the same as list decoding up to a $(1 - \zeta)$ fraction of errors. Guruswami and Rudra have the following result for list recovery.

Theorem 2 ([8]). *For every integer $l \geqslant 1$, for all R, $0 < R < 1$ and $\varepsilon > 0$, and for every prime p, there is an* explicit *family of folded Reed-Solomon codes over fields of characteristic p that have rate at least R and which can be $(R+\varepsilon, l)$-list recovered in polynomial time. The alphabet size of a code of block length N in the family is $(N/\varepsilon^2)^{O(\varepsilon^{-2}\log l/(1-R))}$.*

Theorem 2 will be put to good use in Section 4.

3 Informal Description of the Algorithms

In this section, we will give an overview of the list decoding algorithms that are needed to prove Theorem 1. Along the way we will encounter the main algorithmic techniques used in [14,11,12]. We start by stating more precisely the problem that needs to be solved for Theorem 1. We need list-decoding algorithms for the folded Reed-Solomon code $\mathrm{FRS}_{\mathbb{F}_q,\gamma,m,k}$ of rate R. More precisely, for every $1 \leqslant s \leqslant m$ and $\delta > 0$, given a received word $\mathbf{y} = \langle (y_0, \ldots, y_{m-1}), \ldots, (y_{n-m}, \ldots, y_{n-1}) \rangle$ (where recall $n = q-1$), we want to output all codewords in $\mathrm{FRS}_{\mathbb{F}_q,\gamma,m,k}$ that disagree with $\mathbf{y}$ in at most $1 - (1+\delta)\left(\frac{mR}{m-s+1}\right)^{s/(s+1)}$ fraction of positions in polynomial time. In other words, we need to output all degree k polynomials $f(X)$ such that for at least $(1+\delta)\left(\frac{mR}{m-s+1}\right)^{s/(s+1)}$ fraction of $0 \leqslant i \leqslant n/m - 1$, $f(\gamma^{im+j}) = y_{im+j}$ (for every $0 \leqslant j \leqslant m-1$). By picking the parameters m, s and δ carefully, we will get folded Reed-Solomon codes of rate R that can be list decoded up to a $1-R-\varepsilon$ fraction of errors (for any $\varepsilon > 0$). We will now present the main ideas needed to design the required list-decoding algorithm.

For the ease of presentation we will start with the case when $s = m$. As a warm up, let us consider the case when $s = m = 1$. Note that for $m = 1$, we are interested in list decoding Reed-Solomon codes. More precisely, given the received word $\mathbf{y} = \langle y_0, \ldots, y_{n-1} \rangle$, we are interested in all degree k polynomials $f(X)$ such that for at least $(1+\delta)\sqrt{R}$ fraction of positions $0 \leqslant i \leqslant n-1$, $f(\gamma^i) = y_i$. We now sketch the main ideas of the algorithms in [14,11]. The algorithms have two main steps: the first is an *interpolation* step and the second one is a *root finding* step. In the interpolation step, the list-decoding algorithm finds a bivariate polynomial $Q(X,Y)$ that *fits* the input. That is,

> for every position i, $Q(\gamma^i, y_i) = 0$.

Such a polynomial $Q(\cdot,\cdot)$ can be found in polynomial time if we search for one with large enough total degree (this amounts to solving a system of linear equations). After the interpolation step, the root finding step finds all factors of $Q(X,Y)$ of the form $Y - f(X)$. The crux of the analysis is to show that

> for every degree k polynomial $f(X)$ that satisfies $f(\gamma^i) = y_i$ for at least $(1+\delta)\sqrt{R}$ fraction of positions i, $Y - f(X)$ is indeed a factor of $Q(X,Y)$.

However, the above is not true for every bivariate polynomial $Q(X,Y)$ that satisfies $Q(\gamma^i, y_i) = 0$ for all positions i. The main ideas in [14,11] were to introduce more constraints on $Q(X,Y)$. In particular, the work of Sudan [14] added

the constraint that a certain weighted degree of $Q(X,Y)$ is below a fixed upper bound. Specifically, $Q(X,Y)$ was restricted to have a non-trivially bounded $(1,k)$-weighted degree. The $(1,k)$-weighted degree of a monomial X^iY^j is $i+jk$ and the $(1,k)$-weighted degree of a bivariate polynomial $Q(X,Y)$ is the maximum $(1,k)$-weighted degree among its monomials. The intuition behind defining such a weighted degree is that given $Q(X,Y)$ with weighted $(1,k)$ degree of D, the *univariate* polynomial $Q(X,f(X))$, where $f(X)$ is some degree k polynomial, has total degree at most D. The upper bound D is chosen carefully such that if $f(X)$ is a codeword that needs to be output, then $Q(X,f(X))$ has more than D zeroes and thus $Q(X,f(X)) \equiv 0$, which in turn implies that $Y - f(X)$ divides $Q(X,Y)$. To get to the bound of $1-(1+\delta)\sqrt{R}$, Guruswami and Sudan in [11], added a further constraint on $Q(X,Y)$ that requires it to have r roots at (γ^i, y_i), where r is some parameter (in [14] $r=1$ while in [11], r is roughly $1/\delta$).

We now consider the next non-trivial case of $m = s = 2$ (the ideas for this case can be easily generalized for the general $m = s$ case). Note that now given the received word $\langle (y_0,y_1),(y_2,y_3),\ldots,(y_{n-2},y_{n-1})\rangle$ we want to find all degree k polynomials $f(X)$ such that for at least $(1+\delta)\sqrt[3]{2R^2}$ fraction of positions $0 \leqslant i \leqslant n/2-1$, $f(\gamma^{2i}) = y_{2i}$ and $f(\gamma^{2i+1}) = y_{2i+1}$. As in the previous case, we will have an interpolation and a root finding step. The interpolation step is a straightforward generalization of $m=1$ case: we find a trivariate polynomial $Q(X,Y,Z)$ that fits the received word, that is, for every $0 \leqslant i \leqslant n/2-1$, $Q(\gamma^{2i}, y_{2i}, y_{2i+1}) = 0$. Further, $Q(X,Y,Z)$ has an upper bound on its $(1,k,k)$-weighted degree (which is a straightforward generalization of the $(1,k)$-weighted degree for the bivariate case) and has a multiplicity of r at every point. For the root finding step, it suffices to show that for every degree k polynomial $f(X)$ that needs to be output $Q(X,f(X),f(\gamma X)) \equiv 0$. This, however does not follow from weighted degree and multiple root properties of $Q(X,Y,Z)$. Here we will need two new ideas, the first of which is to show that for some irreducible polynomial $E(X)$ of degree $q-1$, $f(X)^q \equiv f(\gamma X) \mod (E(X))$ [8]. The second idea, due to Parvaresh and Vardy [12], is the following. We first obtain the bivariate polynomial (over an appropriate extension field) $T(Y,Z) \equiv Q(X,Y,Z) \mod (E(X))$. Note that by the first idea, we are looking for solutions on the curve $Z = Y^q$ (Y corresponds to $f(X)$ and Z corresponds to $f(\gamma X)$ in the extension field). The crux of the argument is to show that all the polynomials $f(X)$ that need to be output correspond to (in the extension field) some root of the equation $T(Y,Y^q) = 0$.

As was mentioned earlier, the extension of the $m = s = 2$ case to the general $m = s > 2$ case is fairly straightforward. To go from $s = m$ to any $s \leqslant m$ requires another simple idea from [8]: We will reduce the problem of list decoding folded Reed-Solomon code with folding parameter m to the problem of list decoding folded Reed-Solomon code with folding parameter s. We then use the algorithm outlined in the previous paragraph for the folded Reed-Solomon code with folding parameter s. A careful tracking of the agreement parameter in the reduction, brings down the final agreement fraction (that is required for the original folded Reed-Solomon code with folding parameter m) from $(1+\delta)\sqrt[m+1]{mR^m}$ (which

can be obtained without the reduction and is the bound achieved by [12]) to $(1+\delta)\sqrt[s+1]{\left(\frac{m}{m-s+1}\right)R^s}$.

4 Codes Over Small Alphabets

To get within ε of capacity, the codes in Theorem 1 have alphabet size $N^{\Omega(1/\varepsilon^2)}$ where N is the block length. This leads to the following natural questions:

1. Can we achieve the list decoding capacity for smaller alphabets, say for $2^{\Omega(1/\varepsilon)}$ (for which the list decoding capacity as we saw in the introduction is $1-R$)?
2. Can we achieve list decoding capacity for codes over fixed alphabet sizes, for example, binary codes?

The best known answers to both of the questions above use the notion of *code concatenation* and Theorem 2. We now digress for a bit to talk about concatenated codes (and along the way motivate why list recovery is an important algorithmic task).

Concatenated codes were defined in the seminal thesis of Forney [3]. Concatenated codes are constructed from two different codes that are defined over alphabets of different sizes. Say we are interested in a code over $[q] \stackrel{def}{=} \{0,1,\ldots,q-1\}$ (in this section, we will think of $q \geqslant 2$ as being a fixed constant). Then the *outer code* C_{out} is defined over $[Q]$, where $Q = q^k$ for some positive integer k. The second code, called the *inner code* is defined over $[q]$ and is of dimension k (Note that the message space of C_{in} and the alphabet of C_{out} have the same size). The concatenated code, denoted by $C = C_{out} \circ C_{in}$, is defined as follows. Let the rate of C_{out} be R and let the block lengths of C_{out} and C_{in} be N and n respectively. Define $K = RN$ and $r = k/n$. The input to C is a vector $\mathbf{m} = \langle m_1,\ldots,m_K\rangle \in ([q]^k)^K$. Let $C_{out}(\mathbf{m}) = \langle x_1,\ldots,x_N\rangle$. The codeword in C corresponding to $\mathbf{m}$ is defined as follows

$$C(\mathbf{m}) = \langle C_{in}(x_1), C_{in}(x_2), \ldots, C_{in}(x_N)\rangle.$$

It is easy to check that C has rate rR, dimension kK and block length nN.

Notice that to construct a q-ary code C we use another q-ary code C_{in}. However, the nice thing about C_{in} is that it has small block length. In particular, since R and r are constants (and typically Q and N are polynomially related), $n = O(\log N)$. This implies that we can use up exponential time (in n) to search for a "good" inner code. Further, one can use the brute force algorithm to (list) decode C_{in}.

Finally, we motivate why we are interested in list recovery. Consider the following natural decoding algorithm for the concatenated code $C_{out} \circ C_{in}$. Given a received word in $([q]^n)^N$, we divide it into N blocks from $[q]^n$. Then we use a decoding algorithm for C_{in} to get an intermediate received word to feed into a decoding algorithm for C_{out}. Now one can use unique decoding for C_{in} and list

decoding for C_{out}. However, this loses information in the first step. Instead, one can use the brute force list-decoding algorithm for C_{in} to get a sequence of lists (each of which is a subset of $[Q]$). Now we use a list-recovery algorithm for C_{out} to get the final list of codewords.

By concatenating folded RS codes of rate close to 1 (that are list recoverable by Theorem 2) with suitable inner codes followed by redistribution of symbols using an expander graph (similar to a construction for linear-time unique decodable codes in [6]), one can get within ε of capacity with codes over an alphabet of size $2^{O(\varepsilon^{-4}\log(1/\varepsilon))}$ [8].

For binary codes, recall that the list decoding capacity is known to be $\rho_{\text{bin}}(R) = H_2^{-1}(1-R)$. No explicit constructions of binary codes that approach this capacity are known. However, concatenating the Folded RS codes with suitably chosen inner codes, one can obtain polynomial time constructable binary codes that can be list decoded up to the so called "Zyablov bound" [8]. Using a generalization of code concatenation to *multilevel code concatenation*, one can achieve codes that can be list decoded up to the so called "Blokh-Zyablov" bound [9]. See Figure 1 for a pictorial comparison of the different bounds.

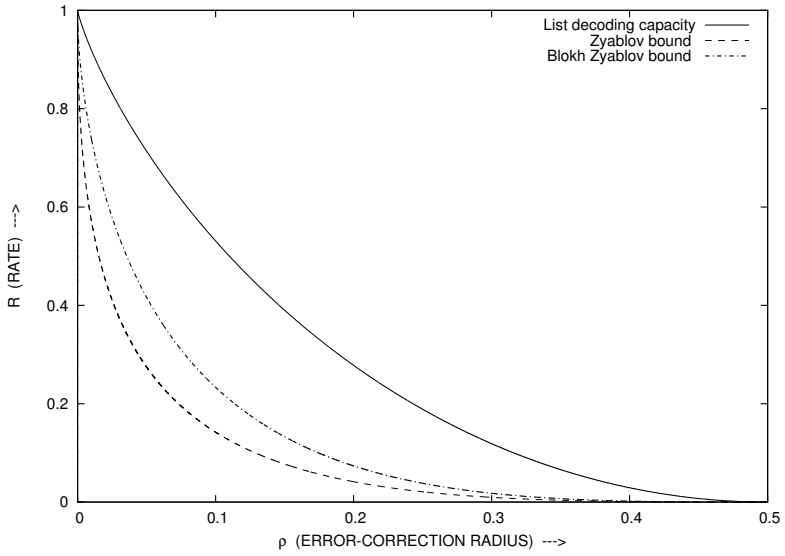

Fig. 1. Rate R of binary codes from [8,9] plotted against the list-decoding radius ρ of their respective algorithms. The best possible trade-off, i.e., list-decoding capacity, $\rho = H_2^{-1}(1-R)$ is also plotted.

5 Concluding Remarks

The results in [8] could be improved with respect to some parameters. The size of the list needed to perform list decoding to a radius that is within ε of capacity grows as $N^{O(\varepsilon^{-1}\log(1/R))}$ where N and R are the block length and the rate of the code respectively. It remains an open question to bring this list size down to a constant independent of N (recall that the existential random coding arguments work with a list size of $O(1/\varepsilon)$). The alphabet size needed to approach capacity was shown to be a constant independent of N. However, this involved a brute-force search for a rather large (inner) code, which translates to a construction time of about $N^{O(\varepsilon^{-2}\log(1/\varepsilon))}$ (instead of the ideal construction time where the exponent of N does not depend on ε). Obtaining a "direct" algebraic construction over a constant-sized alphabet, such as the generalization of the Parvaresh-Vardy framework to algebraic-geometric codes in [7], might help in addressing these two issues.

Finally, constructing binary codes (or q-ary codes for some fixed, small value of q) that approach the respective list decoding capacity remains a challenging open problem. In recent work [10], it has been shown that there *exist* q-ary linear concatenated codes that achieve list decoding capacity (in the sense that every Hamming ball of radius $H_q^{-1}(1-R-\varepsilon)$ has polynomially many codewords, where R is the rate). In particular, this results holds when the outer code is a folded RS code. This is somewhat encouraging news since concatenation has been the preeminent method to construct good list-decodable codes over small alphabets. But realizing the full potential of concatenated codes and achieving capacity (or even substantially improving upon the Blokh-Zyablov bound) with explicit codes and polynomial time decoding remains a huge challenge.

References

1. P. Elias. List decoding for noisy channels. *Technical Report 335, Research Laboratory of Electronics, MIT*, 1957.
2. P. Elias. Error-correcting codes for list decoding. *IEEE Transactions on Information Theory*, 37:5–12, 1991.
3. G. D. Forney. *Concatenated Codes*. MIT Press, Cambridge, MA, 1966.
4. V. Guruswami. *List decoding of error-correcting codes*. Number 3282 in Lecture Notes in Computer Science. Springer, 2004.
5. V. Guruswami. List decoding and pseudorandom constructions. In *Proceedings of the 17th Symposium on Applied algebra, Algebraic algorithms, and Error Correcting Codes (AAECC)*, 2007.
6. V. Guruswami and P. Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, October 2005.
7. V. Guruswami and A. Patthak. Correlated Algebraic-Geometric codes: Improved list decoding over bounded alphabets. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 227–236, October 2006. Accepted to *Mathematics of Computation*.

8. V. Guruswami and A. Rudra. Explicit capacity-achieving list-decodable codes. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 1–10, May 2006.
9. V. Guruswami and A. Rudra. Better binary list-decodable codes via multilevel concatenation. In *Proceedings of 11th International Workshop on Randomization and Computation*, pages 554–568, August 2007.
10. V. Guruswami and A. Rudra. Concatenated codes can achieve list decoding capacity. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, January 2008. To appear.
11. V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45:1757–1767, 1999.
12. F. Parvaresh and A. Vardy. Correcting errors beyond the Guruswami-Sudan radius in polynomial time. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 285–294, 2005.
13. A. Rudra. *List Decoding and Property Testing of Error Correcting Codes*. PhD thesis, University of Washington, 2007.
14. M. Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.
15. M. Sudan. List decoding: Algorithms and applications. *SIGACT News*, 31:16–27, 2000.
16. J. M. Wozencraft. List Decoding. *Quarterly Progress Report, Research Laboratory of Electronics, MIT*, 48:90–95, 1958.
17. V. V. Zyablov and M. S. Pinsker. List cascade decoding. *Problems of Information Transmission*, 17(4):29–34, 1981 (in Russian); pp. 236-240 (in English), 1982.