6.440 Essential Coding Theory April 7, 2008

Lecture 16

Lecturer: Madhu Sudan *Scribe: Brendan Juba*

In this lecture we will examine the performance of Parvaresh-Vardy codes, and we will find that although their list-decoding algorithm yields an improvement over our list-decoding algorithm for Reed-Solomon codes (as the rate approaches zero), they are lacking in certain respects. We will then see how Guruswami and Rudra managed to add a touch of "magic" algebra to recover from these deficiencies to obtain (essentially) rate-optimal efficiently list-decodable codes over a large alphabet.

1 Review: Parvaresh-Vardy codes

We start with a review of the construction of Parvaresh-Vardy codes and their efficient list-decoding algorithm, with a brief summary of its analysis.

1.1 P-V codes

Code parameters. We fix $\overbrace{\mathbb{F}_q, k, \alpha_1, \ldots, \alpha_n}^{\text{standard R}-\text{S}}, \overbrace{h(x), D}^{\text{new}}$ where $\alpha_1, \ldots, \alpha_n$ are distinct elements over $\mathbb{F}_q$, $h(x)$ is an irreducible monic polynomial of degree k, and $D > \left(\frac{n}{k}\right)^{1/3}$. Our alphabet is $\Sigma = \mathbb{F}_q^2$.

Encoding. Recall that our message is given by a polynomial $p_1 \in \mathbb{F}_q[x]$ of degree less than k. We put $p_2 = p_1^D \pmod{h(x)}$, and our encoding of p_1 is given by $\{(p_1(\alpha_i), p_2(\alpha_i))\}_{i=1}^n$

Decoding problem. Given fixed $\mathbb{F}_q, k, D, h(x)$, and the triples $\{(\alpha_i, \beta_i, \gamma_i)\}_{i=1}^n$ (where $\{(\beta_i, \gamma_i)\}_{i=1}^n$ is from the received word), we wish to find all polynomials p_1 of degree less than k such that if $p_2 = p_1^D \pmod{h(x)}$, then $|\{i : \beta_i = p_1(\alpha_i), \gamma_i = p_2(\alpha_i)\}| \geq t$.

1.2 P-V list-decoding algorithm

Algorithm

Step 1: Find $Q(x, y, z) \not\equiv 0$ such that

1. $Q(\alpha_i, \beta_i, \gamma_i) = 0$ for all $i = 1, \ldots, n$.
2. $\deg_x Q \leq k^{2/3} n^{1/3}$, $\deg_y Q, \deg_z Q \leq \left(\frac{n}{k}\right)^{1/3}$.

Step 1.5: While $h(x) | Q(x, y, z)$, put $Q \leftarrow Q/h$.
(Note: still $Q(\alpha_i, \beta_i, \gamma_i) = 0$, degree conditions hold)

Step 2: Put $Q_x(y, z) = Q(x, y, z) \pmod{h(x)}$, put $p_x(y) = Q_x(y, y^D)$, and output all roots of $p_x \in E = \mathbb{F}_q[x]/h(x)$.
Note, $Q_x \in E[y, z]$ and if $Q(x, y, z) \not\equiv 0$ and $h \not| Q$ then $Q_x(y, z) \not\equiv 0$.

Note that the algorithm efficiently solves two large search problems: the first is in steps 1 and 1.5 where, out of the space of all low-degree multivariate polynomials over $\mathbb{F}_q$, we find one satisfying our constraints. The second (which we did not discuss in detail) is in step 2 where we find the roots of p_x over the large field E of low degree polynomials.

Analysis We will not review the full analysis of this algorithm, but recall that we needed essentially the two following claims:

1. If p_1 is a polynomial such that for $p_2 = p_1^D \pmod{h(x)}$, $|\{i : p_1(\alpha_i) = \beta_i, p_2(\alpha_1) = \gamma_i\}| > 3k^{2/3}n^{1/3}$, then p_1 is a root of p_x.
2. The degree of p_x is not too large and $p_x \not\equiv 0$.

2 Performance of Parvaresh-Vardy codes

We now examine the performance of Parvaresh-Vardy codes. Recall that we chose an alphabet of pairs of elements of $\mathbb{F}_q$, where our message corresponded to an element of $\mathbb{F}_q^k = \left(\mathbb{F}_q^2\right)^{k/2}$, i.e., $k/2$ symbols from our alphabet. Thus, our rate is $R = \frac{k}{2n}$. Recall that we could list-decode from $3k^{2/3}n^{1/3}$ agreement, and hence we could recover from

$$\frac{n - 3k^{2/3}n^{1/3}}{n} = 1 - 3(2R)^{2/3} = 1 - O(R^{2/3})$$

fraction of errors. Compared with the rate $1-\sqrt{R}$ we achieved with Reed-Solomon codes, this is substantially better as $R \to 0$ but still short of the optimal $1 - R$ fraction we'd like.

2.1 Two improvements

There are two tricks we can use to squeeze additional performance out of the Parvaresh-Vardy codes, at the cost of an algorithm that is substantially more complex both computationally and conceptually. We will discuss them briefly.

1. **Multiplicities:** As we did while discussing Reed-Solomon codes, we can insist that Q have multiple roots at $(\alpha_i, \beta_i, \gamma_i)$. This would eliminate the leading constant factor of 3 in $3k^{2/3}n^{1/3}$, and would improve our rate to $1 - (2R)^{2/3}$.
2. m**-Correlated polynomials:** We can use additional correlated polynomials: letting p_1 be our message, for $j = 2, \ldots, m$, we put $p_j = p_{j-1}^D \pmod{h(x)}$, and our encoding becomes

   $$p_1 \mapsto \{(p_1(\alpha_i), \ldots, p_m(\alpha_i))\}_{i=1}^n$$

   We will pay an exponential price in m in the running time when decoding, but for any fixed m it is still a polynomial time algorithm, and yields recovery from $1-(mR)^{\frac{m}{m+1}}$ fraction of errors. Asymptotically, for large m, this approaches (letting $R \to 0$ now) $1 - O(R \log \frac{1}{R})$ but doesn't really do much better for any fixed R.

   We're also hiding another drawback here: since our alphabet becomes m-tuples of $\mathbb{F}_q$, our rate can't possibly exceed $1/m$.

2.2 A concrete examination

The final observation in the second improvement demonstrates an inherent weakness in these codes. Suppose we expect 5% of our packets to be corrupted (i.e., 5% errors in our received word) in some application. Petersen's algorithm for decoding Reed-Solomon codes would yield $R \geq 90\%$, while our $\sqrt{kn}$-agreement list-decoding algorithm would yield $R \geq 90.1\%$. Parvaresh-Vardy codes, by contrast, always have $R \leq 50\%$!

3 Guruswami-Rudra

The insight of Guruswami and Rudra is essentially how some "algebraic magic" will eliminate the constant factor overhead of m in using m correlated polynomials. We start with a motivating "wishful thinking" approach that will inspire an approach that succeeds.

3.1 A wishful approach

Fix q odd (so that $x \neq -x$) and suppose that we could arrange that for *all* p_1 of degree less than k, $p_1(x)^D = p_1(-x) \pmod{h(x)}$. We could then recover the factor of two loss in P-V codes: fix a choice of $\{\alpha_i\}$ so that for $j = 1, \ldots, n/2$, $\alpha_j = -\alpha_{j+n/2}$. Then

$$p_2(\alpha_j) = p_1(\alpha_j)^D = p_1(-\alpha_j) = p_1(\alpha_{j+n/2})$$

and likewise, $p_1(\alpha_j) = p_2(\alpha_{j+n/2})$, i.e., the encoding is

$$\begin{array}{cccc|cccc} p_1(\alpha_1) & p_1(\alpha_2) & \cdots & p_1(\alpha_{n/2}) & p_1(-\alpha_1) & p_1(-\alpha_2) & \cdots & p_1(-\alpha_{n/2}) \\ p_1(-\alpha_1) & p_1(-\alpha_2) & \cdots & p_1(-\alpha_{n/2}) & p_1(\alpha_1) & p_1(\alpha_2) & \cdots & p_1(\alpha_{n/2}) \end{array}$$

so the first and second halves contain *exactly* the same pairs (permuted) and hence we don't need to send the second half of the encoding. This approach, if successful, would therefore recover our factor of two loss in the rate.

So, the question is, can we arrange that $p_1(x)^D = p_1(-x) \pmod{h(x)}$ for every message p_1? Recall that our paramteters are $\mathbb{F}_q, k, \alpha_1, \ldots, \alpha_n, D$, and $h(x)$. We don't really have much choice in $\mathbb{F}_q$, and we need a family of $k \to \infty$, so our main parameters to work with are D and $h(x)$.

Our first magic trick is to choose $D = q$. We needed $D > \left(\frac{n}{k}\right)^{1/3}$, but we're free to choose a larger value if we wish, so this is a valid choice. Notice now that we have the following identity over $\mathbb{F}_q$:

$$p_1(x)^q = \left(\sum_i c_i x^i\right)^q = \sum_i c_i^q x^{iq} = \sum_i c_i x^{qi} = p_1(x^q)$$

So, if we could arrange it so that $x^q = -x \pmod{h(x)}$, we would find that for all p_1, $p_1(x)^q = p_1(-x) \pmod{h(x)}$.

Since we have fixed D, the remaining parameter in our control is $h(x)$, and thus the question is, are there choices of $h(x)$ (irreducible, monic polynomials with degrees $k \to \infty$) so that $x^q + x = 0 \pmod{h(x)}$? Unfortunately, the answer is a clear "no."

To see this, recall that $x^{\frac{q-1}{2}} + 1$ splits completely, and has all of the quadratic nonresidues of $\mathbb{F}_q$ as roots, so we know we can write $x^{\frac{q-1}{2}} + 1 = \prod_{\text{nonresidue } \beta}(x - \beta)$. Using this observation, we now write

$$x^q + x = x \cdot ((x^2)^{\frac{q-1}{2}} + 1) = x \cdot \prod_{\text{nonresidue } \beta} (x^2 - \beta)$$

So if $h(x)$ is irreducible and divides $x^q + x$, it has degree at most two (and thus $k \not\to \infty$).

3.2 The successful approach

To recover from this unfortunate situtation, we try to obtain $x^q = \eta x \pmod{h(x)}$ for some η other than -1. Since we had problems due to -1 having low order, suppose we pick an element η with high order instead; let $\alpha \in \mathbb{F}_q^*$ be a primitive element. (Recall: this means that $\alpha, \alpha^2, \ldots, \alpha^{q-1}$ are distinct.)

We can now find $h(x)$ so that $x^q = \alpha x \pmod{h(x)}$: $h(x) = x^{q-1} - \alpha$ works (and turns out to be the only choice). We might fear now that since $h(x)$ has high degree, and we are raising p_1 to some large power, we might obtain a correlated polynomial of very high degree (which would hurt us later) but it turns out that we get very lucky indeed: our choices have arranged that

$$p_1(x)^q = p_1(\alpha x) \pmod{h(x)}$$

so in particular, $p_1(x)^q$ is also a polynomial of degree less than k.

We aren't quite done yet, however, as we no longer find that $(p_1(x), p_2(x)) = (p_2(\alpha x), p_1(\alpha x))$, so we no longer have half of our pairs carrying redundant information, as they did in our wishful motivating approach, and it seems like we need to send all of the pairs. Fortunately, this is not the case, and there is something else we can do, as we illustrate in a small example next.

3.3 G-R codes: a small example

We will illustrate G-R codes with $c = 3$. Our alphabet is now $\Sigma = \mathbb{F}_q^3$, and a message p_1 is encoded as

$$p_1 \mapsto [p_1(\alpha)\ p_1(\alpha^2)\ p_1(\alpha^3)] \cdots [p_1(\alpha^{q-3})\ p_1(\alpha^{q-2})\ p_2(\alpha^{q-1})]$$

Our savings are obtained in the following way: we can obtain two pairs of symbols from each block of three elements of $\mathbb{F}_q$. For example, in the first block, we have the pair $(p_1(\alpha), p_2(\alpha)) = (p_1(\alpha), p_1(\alpha^2))$, as well as the pair $(p_1(\alpha^2), p_2(\alpha^2)) = (p_1(\alpha^2), p_1(\alpha^3))$ (i.e., $p_1(\alpha^2)$ is used twice), where we know that if the block is correct, both pairs of symbols are. We can now use the P-V list-decoding algorithm on these pairs.

Our performance is as follows: our rate is $R = \frac{k}{q-1}$, and we have $n' = \frac{2(q-1)}{3}$ evaluations. Recall that P-V decoding (using multiple roots) can recover from $1 - \left(\frac{k}{n'}\right)^{2/3}$ fraction errors. In our case, this means we recover from $1 - \left(\frac{3}{2}R\right)^{2/3}$ fraction errors, which is already an improvement. In general, we get better performance still by using more correlated polynomials and a larger value of c.

3.4 G-R codes: the general construction

We will use m correlated polynomials for $m = 1/\epsilon$, and $c = m/\epsilon$. Now, our encoding of p_1 is

$$p_1 \mapsto [p_1(\alpha)\ p_1(\alpha^2)\ \cdots\ p_1(\alpha^c)] \cdots [p_1(\alpha^{q-c})\ \cdots\ p_2(\alpha^{q-1})]$$

similar to before, we can get $c - m + 1$ m-tuples out of the each symbol containing m elements. The first symbol, for example, contains

$$\begin{bmatrix} p_1(\alpha) & p_1(\alpha^2) & \cdots & p_1(\alpha^{c-m+1}) \\ p_1(\alpha^2) & p_1(\alpha^3) & & \vdots \\ \vdots & \vdots & \ddots & \\ p_1(\alpha^m) & p_1(\alpha^{m+1}) & \cdots & p_1(\alpha^c) \end{bmatrix} = \begin{bmatrix} p_1(\alpha) & p_1(\alpha^2) & \cdots & p_1(\alpha^{c-m+1}) \\ p_2(\alpha) & p_2(\alpha^2) & & \vdots \\ \vdots & \vdots & \ddots & \\ p_m(\alpha) & p_m(\alpha^2) & \cdots & p_m(\alpha^{c-m+1}) \end{bmatrix}$$

i.e., each column is the evaluation of m correlated polynomials at some α^j. So, we use the P-V algorithm for m correlated polynomials to decode. By a similar calculation to the previous case, we find that we can recover from $1 - \left(\frac{c}{c-m+1}R\right)^{\frac{m}{m+1}}$ fraction errors. In particular, for our choice of m and c, it turns out that we can recover from

$$1 - (1+\epsilon)R^{1-\epsilon} \longrightarrow 1 - R - f(\epsilon)$$

so this code is essentially optimal.

4 Summary

Previously, we saw that we could achieve *some* improvement in our ability to efficiently recover from errors by list-decoding. Now, we see that for large alphabets (and large polynomial running times) we can essentially achieve capacity. Of course, we can reduce the alphabet size by concatenation, from $n^{\text{poly}(1/\epsilon)}$ symbols to $\exp(1/\epsilon)$, and still decode from $1 - R - \epsilon$ errors. The running time is still a problem that we don't know how to address, though. It is currently $\text{poly}(n^{\text{poly}(1/\epsilon)})$. Can we reduce it to $f(1/\epsilon) \cdot \text{poly}(n)$? Or, even to $\text{poly}(n, 1/\epsilon)$? The latter would be surprising, since we don't even know how to achieve such performance under a random error model.

This concludes our treatment of algebraic codes. Next time we will see efficient codes based on a different kind of structure: graph-theoretic codes from sparse graphs.