Lecture 9: List decoding Reed-Solomon and Folded Reed-Solomon codes

Error-Correcting Codes (Spring 2016)
Rutgers University
Swastik Kopparty
Scribes: John Kim and Pat Devlin

1 List decoding review

Definition 1. *$C \subseteq \Sigma^n$ is called* (ρ, L) list decodable *if* $\forall x \in \Sigma^n$, $|B(x, \rho) \cap C| \leq L$.

Remark 2. *ρ is the list decoding radius and L is an upper bound on the list size. $L = 1$ corresponds to standard decoding.*

There exist codes $C \subseteq \{0,1\}^n$ with rate R that are (ρ, L) list decodable with $R > 1 - H(\rho) - \epsilon_L$, where $\epsilon_L \to 0$ as $L \to \infty$. Furthermore, we cannot improve the tradeoff between ρ and R with $L = O(1)$ or even $L = \texttt{poly}(n)$.

Over large alphabets Σ, there exists codes $C \subseteq \Sigma^n$ with $R > 1 - \rho - \epsilon_L$. With the same rate, we can correct about twice as many errors in L-list decoding (for large L) than with unique decoding.

Theorem 3 (Johnson bound)**.** *Codes with minimum relative distance δ are $(J(\delta), \texttt{poly}(n))$ list decodable, where*

$$J(\delta) = \begin{cases} 1 - \sqrt{1-\delta} & \text{for large } \Sigma \\ \frac{1-\sqrt{1-2\delta}}{2} & \text{for } \Sigma = \{0,1\} \end{cases}.$$

We used the Johnson bound to improve the R, δ tradeoff for standard codes.

2 List decoding of Reed-Solomon codes

We saw a $\texttt{poly}(n)$ time algorithm to list decode Reed-Solomon codes of degree d from $n - \sqrt{nd}$ errors with a $\texttt{poly}(n)$ size list.

List decoding problem: Given $r : S \to \mathbb{F}_q$, find all polynomials $P(X)$ of degree at most d such that $\Delta(r, P) < n - \sqrt{nd}$.

Two steps:

1. **Interpolation**

   Find a low-degree bivariate polynomial $Q(X, Y)$ such that

   $$Q(a, r(a)) = 0, \forall a \in S.$$

2. **Factoring/Root-finding**

   Factor $Q(X, Y)$. If $P(X)$ is close to $r(X)$, then $Y - P(X)$ shows up as a factor of $Q(X, Y)$.

How should we choose $Q(X, Y)$ so that the factoring step finds all the polynomials we want? To this end, pick $Q(X, Y)$ with $(1, d)$ weighted degree $\leq D$ that vanishes at each $(a, r(a))$ with multiplicity $\geq M$. Note that we are free to tune our parameters D and M to capture all polynomials with at least $\sqrt{nd}$ agreements with r.

Total number of monomials of $(1, d)$ weighted degree $\leq D$ is at least:

$$\frac{D^2}{2d}.$$

Total number of constraints imposed by vanishing conditions is at most:

$$n \cdot \frac{M^2}{2}.$$

This is because vanishing at $(a, r(a))$ with multiplicity M means that the partial derivatives $Q^{(i,j)}(a, r(a)) = 0$ for $i + j < M$.

As long as the number of variables exceeds the number of constraints, then we can interpolate such a polynomial $Q(X, Y)$:

$$\begin{aligned} \frac{D^2}{2d} &> n \cdot \frac{M^2}{2} \\ \frac{D}{M} &> \sqrt{nd}. \end{aligned}$$

So we choose D, M so that $\frac{D}{M} = \sqrt{nd} + \epsilon$, and interpolate $Q(X, Y)$ with $(1, d)$ weighted degree $\leq D$ that vanishes at each $(a, r(a))$ with multiplicity $\geq M$.

Question: How close does P have to be to r for this algorithm to capture P?

Suppose $\Delta(P, r) < U$. Then the polynomial $h(X) = Q(X, P(X))$ vanishes with multiplicity M at each $a \in S$ for which $P(a) = r(a)$. So h has a multiplicity M root at $n - U$ distinct points. Since $\deg(h) \leq D$, if $(n - U)M > D$, then $h(X) \equiv 0$. This means that $Y - P(X) | Q(X, Y)$, and we will find $P(X)$ through the factorization of $Q(X, Y)$.

So to find $P(X)$, we need $U < n - \frac{D}{M} = n - \sqrt{nd} - \epsilon$.

3 Polynomial reconstruction

In the above algorithm for list decoding Reed-Solomon codes, we only used the fact that the points determined by r were distinct. This means that the same algorithm should work for the following more general noisy polynomial interpolation problem:

Polynomial reconstruction: Given distinct points $(a_i, b_i) \in \mathbb{F}_q^2, i = 1, \ldots, n$, find all polynomials $P(X)$ of degree $\leq d$ such that $Y = P(X)$ passes through at least A of the points.

The Reed-Solomon list decoding algorithm solves this problem with $A = \sqrt{nd}$.

In the algorithm, we interpolated the polynomial $Q(X, Y)$ to vanish with the same multiplicity at all the points. However, we have freedom to vanish with different multiplicities at different points. This gives us the ability to give greater weight to some points over others. Using this idea, we get an algorithm for the following weighted polynomial reconstruction problem:

Weighted polynomial reconstruction: Given distinct points $(a_i, b_i) \in \mathbb{F}_q^2, i = 1, \ldots, n$, and weights $w_i \in [0, 1]$, find all polynomials $P(X)$ of degree $\leq d$ such that $Y = P(X)$ passes through points with total weight $\geq A$.

We can solve this problem with $A = \sqrt{\sum w_i^2 d}$. The algorithm also works if we replace the finite field $\mathbb{F}_q$ with $\mathbb{R}$. We can also find irreducible polynomials $T(X, Y)$ passing through many points, and shapes in sets of points even with outliers.

4 Constructing binary list decodable codes

We will construct binary list decodable codes from nearly $1/2$ fraction errors. Our plan is to take a Reed-Solomon code of rate $\hat{\epsilon}$ and distance $1-\hat{\epsilon}$, and concatenate it with a suitable inner binary code C_{in} with message length $\log q$ (so we can brute force decode on the inner code in time $\texttt{poly}(q)$) and distance $1/2 - \epsilon$. From the previous lecture, we know the inner code has rate ϵ^3. By list decoding each block of the inner code, we will get a list of candidates for the corresponding symbol of the Reed-Solomon outer code.

To decode this concatenated code from $1/2 - \epsilon'$ fraction errors, we need to be able to decode each block from $1/2 - \epsilon'$ fraction errors. This means that we must do list decoding of C_{in} as unique decoding only allows us to correct up to $1/4$ fraction errors (half the minimum distance).

We can list decode C_{in} from $1/2 - \sqrt{\epsilon}$ fraction errors by brute force in $2^{O(\log q)} = \texttt{poly}(q)$ time with list size $\leq O(1/\epsilon)$.

The constant upper bound on the list size follows from a version of the Johnson bound stating that codes of distance δ are $(J(\delta) - \epsilon, O(1/\epsilon^2))$ list decodable. Recall from the proof of the Johnson bound that we need to show that a collection of vectors with pairwise inner product at most $-\epsilon$ has size at most $O(1/\epsilon)$. This is left as an exercise. (Hint: If the vectors are $v_1, \ldots, v_k$, then $\langle \sum v_i, \sum v_i \rangle \geq 0$).

If there are $1/2 - \epsilon'$ fraction errors in the received string, then in many blocks, few ($\leq 1/2 - \sqrt{\epsilon}$) errors occur. This follows from Markov's inequality:

$$\begin{aligned} \Pr\left[\#\text{ errors in block } i > \frac{1}{2} - \sqrt{\epsilon}\right] &\leq \frac{1/2 - \epsilon'}{1/2 - \sqrt{\epsilon}} \\ &= 1 - 2(\epsilon' - \sqrt{\epsilon}). \end{aligned}$$

In other words:

$$\Pr\left[\#\text{ errors in block } i < \frac{1}{2} - \sqrt{\epsilon}\right] \geq 2(\epsilon' - \sqrt{\epsilon}).$$

So choose $\epsilon' = 2\sqrt{\epsilon}$. Then at least $2\sqrt{\epsilon}$ fraction of the blocks have the right answer in their list decodings.

At this point, we have list-decoded all the inner codewords [the blocks]. For each such block, we have a list of size at most $O(1/\epsilon)$, and although perhaps some of these blocks were list-decoded incorrectly (because too many errors happened to occur on those blocks), we know that at least $2\sqrt{\epsilon}$ fraction of the blocks have been list-decoded correctly. So to finish list decoding the entire message, we need only (efficiently) solve the following problem:

> **Subproblem:** For each coordinate $a \in \mathbb{F}_q$, we have a list $L_a \subseteq \mathbb{F}_q$ of size $|L_a| \leq O(1/\epsilon)$. We want to find all $P(X)$ in the Reed-Solomon code with $|\{a \in \mathbb{F}_q : P(a) \in L_a\}| \geq 2\sqrt{\epsilon}q$.

Note that we can frame the above in the language of the polynomial reconstruction problem. Namely, if we consider the points $\{(a, y) \ : \ a \in \mathbb{F}_q, y \in L_a\}$, then we have $N := O(1/\epsilon)q$ points and $A = 2\sqrt{\epsilon}q$. Therefore, this can be solved efficiently if $A > \sqrt{Nd}$, where d is the degree of the polynomials in the outer Reed-Solomon code.

This means we need $\sqrt{O(1/\epsilon)qd} < 2\sqrt{\epsilon}q$, which is implied by $d \leq C\epsilon^2 q$, for some fixed constant C. Therefore, if we take our Reed-Solomon code to have rate $\hat{\epsilon} = C\epsilon^2$, then we can efficiently solve the above problem.

With this choice, we obtain that our concatenated code has rate $O(\hat{\epsilon}\epsilon^3) = O(\epsilon^5) = O((\epsilon')^{10})$, and we can list-decode from $1/2 - \epsilon'$ errors efficiently.

5 List decoding over large alphabets with optimal rate/radius tradeoff

We will now see (although the proof will have to be completed next time) a construction that will allow us to get an optimal tradeoff between rate and radius (when the alphabet size is large). This is by a gadget known as a *folded Reed-Solomon code* (which is a special case of what's known as a Parvaresh-Vardy code).

Definition 4. *Let $\gamma \in \mathbb{F}_q^*$ be a generator of the multiplicative group of $\mathbb{F}_q$, and let $s \geq 1$ be an integer parameter where s divides $q-1$. Then a folded Reed-Solomon code (FRS) is a code, C, with alphabet $\Sigma = \mathbb{F}_q^s$, and $C \subseteq \Sigma^{(q-1)/s}$ is given by*

$$C := \{(f(\gamma^{is+0}), f(\gamma^{is+1}), \ldots, f(\gamma^{is+s-1}))_{i=0}^{(q-1)/s-1} : f \in \mathbb{F}_q[X],\ \deg(f) \leq d\}.$$

Remark 5. *Intuitively, the FRS code can be thought of as taking a Reed-Solomon code and grouping some of the symbols together (viewing each such grouping of s original symbols as one collective symbol in a larger alphabet). Note that $s = 1$ corresponds to the usual Reed-Solomon code.*

Remark 6. *Note that this code can in fact be constructed efficiently. An initial concern may be that finding a multiplicative generator of $\mathbb{F}_q^*$ might be prohibitively time-consuming. But although it remains an interesting question if such a task can be done in $\texttt{log}(q)$ time, it is trivial to see that it can be done in $\texttt{poly}(q)$ time simply by checking each element. (Recall that a generator of the multiplicative group of $\mathbb{F}_q^*$ is an element such that $\gamma, \gamma^2, \ldots, \gamma^{q-1}$ are all distinct*[1]*.)*

Remark 7. *A good question is "why would such a regrouping of symbols help us?" The idea behind this is that in the original setting (before grouping these symbols together) correcting from "ϵ-fraction of error" could have meant that the adversary corrupted an ϵ-fraction of each of our now grouped together symbols. But if we instead view the grouped together symbols each as one character of a larger alphabet, then if we are correcting "ϵ-fraction of error," we know that many of these large blocks are still completely intact, which is what will help us. In essence, by folding, we restrict what types of corruption are allowed.*

To quote Professor Kopparty, "the best motivation for the code is to see the analysis of the decoding algorithm."

List decoding algorithm for FRS codes (outline): Suppose we are given a message r. Although r consists of a $(q-1)/s$-tuple of elements of $\mathbb{F}_q^s$, for convenience we will view it instead as a function from $\{\gamma^0, \gamma^s, \ldots, \gamma^{q-1-s}\}$ to $\mathbb{F}_q^s$ (where $r(\gamma^{si})$ is simply the index i coordinate of the received tuple).

The FRS algorithm is extremely similar to that for Reed-Solomon codes. It consists of the following:

- **Step 1:** Find a polynomial $Q(X, Y_0, Y_1, \ldots, Y_{s-1})$ with $Q(\gamma^{is}, r(\gamma^{is})) = 0$ for all $i < (q-1)/s$.

  An important gain compared to earlier is that in this setting $\deg(Q) \leq O(n^{1/(s+1)})$.

- **Step 2:** If $P(X) \in \mathbb{F}_q[X]$ is such that its codeword is close to r, then we consider $h(X) = Q(X, P(X), P(\gamma X), P(\gamma^2 X), \ldots, P(\gamma^{s-1} X))$.

  By construction of Q, if $r(\gamma^{is}) = (P(\gamma^{is}), P(\gamma^{is+1}), \ldots, P(\gamma^{is+s-1}))$, then $h(\gamma^{is}) = 0$. Therefore, if r is close to the codeword of P, then h has many roots, which will imply $h(X) \equiv 0$ (since h will have low degree).

- **Step 3:** We therefore will obtain that $0 \equiv Q(X, P(X), P(\gamma X), \ldots, P(\gamma^{s-1} X))$ whenever P is close to r. So we simply need to efficiently list all such low degree polynomials for Q as given.

  Having done this, we will be done.

Remark 8. *In this lecture, we will expound on steps 1 and 2, but we leave step 3 for next time. Step 3 can in fact be done efficiently, and this is where all the "magic" of the folding comes in (and where the fact that γ is a generator comes in).*

[1] One can see that such an element must exist by considering the roots of the polynomials of the form $p_d(X) = X^d - 1$. Namely, p_d has at most d roots (because $\mathbb{F}_q$ is a field), so this means that $\mathbb{F}_q^*$ has at most d elements whose order divides d. And since $\mathbb{F}_q^*$ is a finite abelian group, it can be seen (through a number of easy arguments) that this condition implies it is cyclic.

Remark 9. *Note that in its full generality, the problem described in step 3 cannot be solved efficiently. For example, if we took $Q(X,Y,Z) = Y + Z$ and $\gamma = -1$, then the problem would be to list all polynomials with $P(X) + P(-X) = 0$. But there are very many such polynomials (since this just implies that P is odd), so they could not possibly be listed efficiently (being too numerous).*

List decoding algorithm for FRS codes (with details and analysis): We let r be as in the outline. As in the Reed-Solomon case, let D and M be parameters to be chosen.

[*Step 1*] We want to find $Q(X, Y_0, Y_1, \ldots, Y_{s-1})$ to have $(1, d, d, \ldots, d)$ weighted degree at most D, and we want Q to vanish with multiplicity M at each of the $(q-1)/s$ points $(\gamma^{is}, r(\gamma^{is}))$. The total number of monomials having that weighted degree at most D is

$$(D \cdot D/d \cdot D/d \cdots D/d)/(s+1)! = \frac{D^{s+1}}{d^s(s+1)!}.$$

On the other hand, the number of linear constraints we impose regarding the vanishing of Q is

$$\frac{q-1}{s}\binom{M+s}{s+1} \leq \frac{q-1}{s} \cdot \frac{M^{s+1}}{(s+1)!} \exp\left[\frac{s(s+1)}{2M}\right].$$

To ensure a non-zero solution to the corresponding homogeneous linear system, we therefore insist

$$\frac{q-1}{s}\frac{M^{s+1}}{(s+1)!} \exp\left[\frac{s(s+1)}{2M}\right] < \frac{D^{s+1}}{d^s(s+1)!},$$

or equivalently $D/M > e^{s/2M}(d^s(q-1)/s)^{1/(s+1)} = e^{s/2M}(d^s n)^{1/(s+1)}$. If R is the rate of our code, then $d = R(q-1) = Rsn$. Therefore, the polynomial Q can be found [accomplishing step 1] provided that

$$D/M > e^{s/2M} n (Rs)^{s/(s+1)}. \tag{1}$$

[*Step 2*] Now let $P(X)$ and $h(X)$ be as in the outline, and suppose P agrees with r on A points. Then h is a polynomial of degree at most D and h vanishes with multiplicity M at every point of agreement between P and r (by the construction of Q and h). We would like to conclude $h \equiv 0$, and we are in fact able to do so provided $A > D/M$.

Thus, combining with (1), we have accomplished steps 1 and 2 for all P agreeing with r on at least $e^{s/2M} n (Rs)^{s/(s+1)}$ points. [In fact, we will assume that M is much larger than s, and thus replace this lower bound with simply $(1 + o(1)) n (Rs)^{s/(s+1)}$.]

Next time, we will show step 3, and we'll also improve the argument to eliminate the $s^{s/(s+1)}$ term.