Coding Theory Nov 2nd

List Decoding

Instructor: Venkatesan Guruswami *Scribe: Anish Sevekari, Yifan Song*

1 Introduction of List Decoding

In this lecture, we go back to the *worst-case errors* model. Our goal is to correct more errors, approaching what is possible with random errors.

However, the main issue is *half-the-distance* barrier: the maximum number of errors which are correctable is less than $d/2$, where the d is the distance of a code.

Concretely, say the distance between two codewords $c_1$and c_2 is exactly d. Then the "middle" of two codewords $y = c_1 + e_1 = c_2 + e_2$, where $\mathtt{wt}(e_1) = \mathtt{wt}(e_2) = d/2$, is not correctable. Combining with the known Singleton Bound $d \leq n - k + 1$, the maximum number of errors which are correctable is bounded by $(n-k)/2$, or equivalently, the fraction of errors is bounded by $(1-R)/2$ ($\leq 1/2$). Here we use n for the length of the codeword, k for the size of the codeword and $R = k/n$.

An important observation is that, for most of error vector $\vec{e}$ with $\mathtt{wt}(\vec{e}) = e \gg d/2$, there is still only one codeword within distance e. So, such an $\vec{e}$ ought to be corrected.

To correcting more errors, there are two ways to overcome the barrier:

- Make some assumptions in the distance of $\vec{e}$ and succeeds with high probability. E.g., Shannon Model. The solutions like polar code depend highly on the property of the channel.
- Stick to the *worst-case errors* model but relax the decoding requirement. Allow the decoder to output a *small* list of closely codewords.

In the following, we focus on the second way and it is what we call List Decoding.

2 Motivations

Before we introduce the definition of List Decoding, a very natural question is: *Does List Decoding make sense?*

Here are several motivations to answer the above question:

- Small list is surely better than giving up.
- Context Specific information may help disambiguate the list.
- Can even treat the case where the list size is larger than 1 as a decoding failure. It expands the unique decoding radius on many error patterns.

- Connection to complexity, cryptography and pseudorandomness.

  ⋆ Hard-core predicate from hard (one-way) functions. Basically, one way functions are easily computed but hard to invert. A hard-core predicate is a function which "extracts" the hardness from a one-way function into one bit. Specifically, say f is a one-way function. We pick $h_r(x) = \langle x, r\rangle$ as the hard-core predicate. Then, given $f(x)$ and a random r, predicting $h_r(x)$ is hard, cannot do better than 50/50.

  ⋆ Amplify the hardness of functions. Boost worst-case hardness to average-case hardness.

3 Definition and Existential results of List Decoding

In this section, we first give a formal definition of List Decoding, then give several existential results to see how good the List Decoding can get.

Definition 1. *A code $C \subset \Sigma^n$ is $(\tau, L)-$list decoding if for all $y \in \Sigma^n$,*

$$|B(y, \tau n) \cap C| \leq L,$$

where Σ is an alphabet set and $B(y, \tau n)$ is the hamming ball of radius τn with center y.

In the definition, τ is the error fraction that the code C can handle and L is the list size. Note that a $(\tau, 1)-$list decodable code is equivalent to say the code has distance ratio 2τ.

The following lemma gives us a lower bound on the existence of List Decodable Code.

Lemma 1. *There exists $(\tau, L)-$list decodable $q-$ary codes of rate at least $1 - h_q(\tau) - 1/L$.*

Remark 1. *Here are some interpretations of Lemma 1:*

1. *When $q = 2$, it corresponds to the case for binary code. The rate is at least $1 - h(\tau) - 1/L$. As L grows, the rate can approach $1 - h(\tau)$, which is the capacity of BSC_τ. It means that, List Decoding can approach Shannon Capacity even in the* worst-case error *model, "a bridge between Shannon to Hamming".*

2. *When q is large, $h_q(\tau) \approx \tau + 1/\log q$. The rate is at least $1 - \tau - 1/\log q - 1/L$. If we pick $L = \Omega(1/\epsilon), q \geq \exp(\Omega(1/\epsilon))$, the rate is approaching $1 - \tau - \epsilon$. It means that τ can be $1 - R - \epsilon$ which matches the Singleton Bound! Moreover,*

   (a) It is two times of the $(1 - R)/2$ limit for unique decoding.

   (b) It can correct as many errors (respect to List Decoding) as erasures (respect to Unique Decoding).

   (c) It matches the best possible τ.

Now we give a proof for Lemma 1.

Proof. We pick C at random (each codeword is chosen independently and randomly). We will show that, with high probability, for all y,

$$|B_q(y, \tau n) \cap C| \leq L.$$

Let $C = \{c_1, c_2, ..., c_{|L|}\}$. For a fix y, by union bound,

$$\begin{aligned}
\Pr_C[|B_q(y, \tau n) \cap C| \geq L+1] &\leq \sum_{(i_0, i_1, ..., i_L)} \Pr_C[c_{i_0}, c_{i_1}, ..., c_{i_L} \in B_q(y, \tau n)] \\
&= \binom{|C|}{L+1} \left(\frac{\mathtt{Vol}(B_q(y, \tau n))}{q^n}\right)^{L+1} \\
&\leq q^{Rn(L+1)} (q^{h_q(\tau)n - n})^{L+1}.
\end{aligned}$$

Still by union bound,

$$\begin{aligned}
\Pr_C[\exists y, |B_q(y, \tau n) \cap C| \geq L+1] &\leq \sum_y \Pr_C[|B_q(y, \tau n) \cap C| \geq L+1] \\
&\leq q^n q^{Rn(L+1)} (q^{h_q(\tau)n - n})^{L+1} \\
&= q^{(\frac{1}{L+1} - \frac{1}{L})n(L+1)} \\
&= q^{-\frac{n}{L}}.
\end{aligned}$$

Thus, with probability at least $1 - q^{-n/L}$, a random C is $(\tau, L)-$list decoding. □

Also we have the converse of Lemma 1:

Lemma 2. *A code $C \in [q]^n$ of rate $1 - h_q(\tau) + \gamma$ is not $(\tau, q^{\Omega_\gamma(n)})-$list decoding, where $\gamma > 0$ is a constant.*

Proof. Pick $y \in [q]^n$ at random. Then we have

$$\begin{aligned}
\mathbb{E}[B(y, \tau n) \cap C] &= \mathbb{E}[\sum_{c \in C} \mathbb{1}(c \in B(y, \tau n))] \\
&= \mathbb{E}[\sum_{c \in C} \mathbb{1}(y \in B(c, \tau n))] \\
&= \sum_{c \in C} \mathbb{E}[\mathbb{1}(y \in B(c, \tau n))] \\
&= |C| \frac{\mathtt{Vol}(B_q(c, \tau n))}{q^n} \\
&\geq q^{(1 - h_q(\tau) + \gamma)n + (h_q(\tau)n - o(n)) - n} \\
&\geq q^{\gamma n/2}
\end{aligned}$$

Therefore, there exists $y^\star$ such that $|B(y^\star, \tau n) \cap C| \geq q^{\gamma n/2}$. □

Remark 2. *Note that Lemma 1 and Lemma 2 pin down "List Decoding Capacity" to $1 - h_q(\tau)$.*

4 Johnson Bound

Johnson Bound gives us an upper bound on the list size of a code based on its distance and value of τ. It assures that there are values of τ for which there is a poly(n) size list decoding. This assures that the size of the list doesn't suddenly grow from constant to exponential as we increase τ above $\delta/2$ where δ is the relative distance. This is the analogue of *Elias Bassalygo bound* in the list decoding setting.

Lemma 3 (Johnson Bound)**.** *Any code of distance δn is $(\tau, poly(n))-$List Decoding when*

$$\tau < 1 - \sqrt{1-\delta}.$$

(And $\tau < (1-\frac{1}{q})(1-\sqrt{1-\frac{q\delta}{(q-1)}})$ for $q-$ary code)

Proof. For a fixed $y = (y_1, y_2, ..., y_n) \in \Sigma^n$, let $S = \{c_1, c_2, ..., c_{|S|}\} \subset C$ be the set of codewords which are in $B(y, \tau n)$. Consider the bipartite graph G:

Each codeword $c_i \in S$ corresponds to a node in G, each entry y_j corresponds to a node in G. There is an edge between c_i and y_j if and only if $c_{ij} = y_j$.

An "angle" of G is a triple (c_{i_1}, y_j, c_{i_2}) where $i_1 < i_2$ and there is an edge between (c_{i_1}, y_j) and between (c_{i_2}, y_j). Let d_j be the degree of node y_j. Since all codewords in S is in $B(y, \tau n)$, we have $\sum_{j=1}^{n} d_j \geq (1-\tau)n|S|$. Therefore, the number of "angles" in G is

$$\sum_{j=1}^{n} \binom{d_j}{2} \geq n\binom{(1-\tau)|S|}{2}.$$

Note that, for a fixed c_{i_1}, c_{i_2}, they can only agree on at most $(1-\delta)n$ entries. Thus, the number of "angles" in G is bounded by

$$\binom{|S|}{2}(1-\delta)n.$$

Combining the above two inequalities, we have

$$n\binom{(1-\tau)|S|}{2} < \binom{|S|}{2}(1-\delta)n,$$

which implies

$$|S| \leq \frac{\delta - \tau}{(1-\tau)^2 - (1-\delta)}$$

Therefore, when $(1-\tau)^2 > (1-\delta)$, which is just $\tau < 1 - \sqrt{1-\delta}$, the size of $|S|$ is bounded. Note that $|S|$ is the size of the list. □

5 List Decoding Reed Solomon

List decoding is harder than unique decoding since unique decoding is the special case where list size is 1. Decoding is hard, since we cannot even verify that the given code has large distance easily.

Our goal is to give a list decoding algorithm for $[n, k+1, n-k]_q$ RS code which can list decode up to Johnson Bound, that is, up to $n - \sqrt{kn} = n(1 - \sqrt{R})$ errors. Assume that $k \leq \frac{n}{4}$. Suppose that the RS encoding is given by $f(x) \mapsto (f(a_1), \ldots, f(a_n))$. Given a codeword w, we want to list the polynomials f which agree with w on at least $t = \sqrt{kn}$ positions.

5.1 Solving an easy subcase

First, we consider the simple case where the input is in fact a mixture of outputs of two polynomials. More precisely, we want to solve the following problem:

Input: $y \in \mathbb{F}_q^n$ st $\exists A, B \subset [n]$, $A \cap B = \emptyset$ $|A| = |B| = \frac{n}{2}$, st,
for all $i \in A$, $y_i = f(a_i)$
for all $i \in B$, $y_i = g(a_i)$
for some polynomials f and g.
Output: Polynomials f and g.

First we have the following lemma:

Lemma 4. *Suppose $\exists h \in \mathbb{F}_q^n[x]$ such that $\deg(h) \leq k$ and $h(a_i) = y_i$ for at least $\frac{n}{2}$ values of i. Then $h = f$ or $h = g$.*

If $h(a_i) = y_i$ for at least $\frac{n}{2}$ values, then h agrees with at least $\frac{n}{4} \geq k$ positions with either f or g. Since all the polynomials have degree less than k, this actually implies equality.

Now we are in position to describe the algorithm to find f and g. Note that for all i, $(y_i - f(a_i))(y_i - g(a_i)) = 0$. Define

$$Q_0(x, y) = (y - f(x))(y - g(x))$$

Then $Q_0(a_i, y_i) = 0$ for all i. Further, $Q_0(x, y) = y^2 - S(x)y + P(x)$ with $\deg(S) \leq k \leq \frac{n}{4}$ and $\deg(P) \leq 2k \leq \frac{n}{2}$. Now, we can find f, g using the following algorithm:

Algorithm 1 EasyDecode

1: Find $Q(x, y) = y^2 - S(x)y + P(x)$ such that $Q(a_i, y_i) = 0$ for all i.
2: Factorize $Q(x, y)$ into $(y - \tilde{f}(x))(y - \tilde{g}(x))$.
3: **return** $\tilde{f}, \tilde{g}$.

For the step 1, $\deg(S) \leq k$ and $\deg(P) \leq 2k$. Now, we can express the polynomial condition in terms of a linear program with coefficients of P and S as variables. Note that this linear system has a solution, namely Q_0 as defined above. Therefore, we can use linear programming to get a solution to this in polytime.

We can just use the quadratic formula to factorize the quadratic, which does work over $\mathbb{F}_q^n[x]$, if there is actually a factorization of $Q(x, y)$. Note that quadratic formula does involve taking a square root in the ring $\mathbb{F}_q^n[x]$, which we can do assuming we can take square roots over $\mathbb{F}_q^n$ if the discriminant is actually a square in $\mathbb{F}_q^n[x]$. Note that the discriminant is a square if $Q(x, y)$ has factors linear in y. We will show below that such a factorization always exists.

Theorem 5 (Correctness of EasyDecode)**.** *Algorithm above correctly gives you the polynomial f and g such that $y_i = f(a_i)$ for half the indices i and $y_i = g(a_i)$ for other half of the indices, assuming that such polynomials exist.*

Proof.

(i) Linear system for step 1 of the algorithm always has a solution, since Q_0 is a solution. Therefore, we can solve the system and find some nonzero solution using gaussian elimination. Let this solution be $\tilde{Q}(x, y)$.

(ii) $\tilde{Q}(a_i, y_i) = 0$ by definition. We claim that $y - f(x) \mid \tilde{Q}(x, y)$. By long division, we can write $\tilde{Q}(x, y) = (y - f(x))S(x, y) + T(x)$. Thus, it suffices to show that $T(x) = \tilde{Q}(x, f(x)) = 0$. Note that $\deg(B) < \frac{n}{2}$, but B vanishes on the set A, and hence has $\frac{n}{2}$ roots. Therefore, $B = 0$.

Therefore, $y - f(x) \mid \tilde{Q}(x, y)$ and similarly, $y - g(x) \mid \tilde{Q}(x, y)$. This implies that $Q_0(x, y) \mid \tilde{Q}(x, y)$ and since $Q_0(x, y)$ and $\tilde{Q}(x, y)$ agree on degree and the leading coefficient, they must be the same. In particular, $\tilde{Q}(x, y)$ can be factorized into factors linear in y.

(iii) Given any factorization $(y - \tilde{f}(x))(y - \tilde{g}(x)) = \tilde{Q}(x, y)$, at least one of $\tilde{f}$ and $\tilde{g}$ satisfy the hypothesis of lemma 4, and hence equals either f or g. Wlog, let $\tilde{f} = f$. This forces $\tilde{g} = g$ from part 2 and completes the proof.

Also, since $\mathbb{F}_q^n[x, y]$ is a UFD, factorization of $\tilde{Q}(x, y) = Q_0(x, y)$ into linear (in y) factors is unique, and hence any factorization must in fact give us f and g.

□

5.2 General list decoding algorithm

Now we proceed to the General List decoding, where the input y might not be a combination of two polynomials f and g. Let $\mathbb{F} = \mathbb{F}_q$.

Input:	n distinct points $(a_i, y_i) \in \mathbb{F}^2$ agreement parameter t degree k
Output:	All polynomials $f \in \mathbb{F}[x]$ st $\deg(f) < k$ and $\lvert\{i \mid f(a_i) = y_i\}\rvert \geq t$

We will first give an algorithm which solves this problem for all $t > 2\sqrt{kn}$, then improve it to work for all $t > \sqrt{2kn}$ and then further improve it to work for all $t > \sqrt{kn}$. The improvement from first to second algorithm follows by tweaking parameters cleverly, but the improvement from second to third requires another fundamental idea, which will be discussed in next lecture.

Algorithm 2 ListDecode

1: $l \leftarrow \sqrt{n/k}$
2: Find $Q(x, y) \neq 0$ such that
$Q(x, y) = A_l(x)y^l + \cdots + A_0(x)$ where $\deg(A_j) \leq d_x$
and $Q(a_i, y_i) = 0$ for all i.
3: Find all linear (in y) factors $y - f(x)$ of Q.
Output f if f agrees on $\geq t$ values and has degree less than k.

Theorem 6 (Correctness of ListDecode)**.** *The algorithm ListDecode outputs the list of all polynomials f such that $f(a_i) = y_i$ for at least t values of i. Further, the list contains at most l elements.*

Proof. The conditions in step 1 can be expressed as a set of homogeneous linear constraints in coefficients of A_i. Since each A_j has degree d_x, there are $(l + 1)(d_x + 1)$ total variables. And n total constraints, one per point (a_i, y_i). Therefore, if $n < (l + 1)(d_x + 1)$ then this system is under determined, and must have a non-zero solution. Hence, if we choose $d_x = \frac{n}{l}$, then the system has a non-zero solution.

Now we claim that for any polynomial f with $\deg(f) \leq k$ such that $f(a_i) = y_i$ for at least t values of i, $y - f(x) \mid Q(x, y)$ if $l = \sqrt{n/k}$ and $t > 2\sqrt{kn}$. Therefore, all such polynomials are included in linear factors of $Q(x, y)$. To prove the claim, it suffices to show that $Q(x, f(x)) = 0$ (see proof of theorem 5). Let $R(x) = Q(x, f(x))$. For any i such that $f(a_i) = y_i$, $R(a_i) = Q(a_i, y_i) = 0$. Therefore, $R(x)$ has at least t distinct roots. But, if $l = \sqrt{n/k}$ then

$$\deg(R) < d_x + kl < \frac{n}{l} + kl = 2\sqrt{kn} < t$$

And hence, $R(x) \equiv 0$. This proves the claim.

Only remaining part is to factorize $Q(x, y)$ into linear factors efficiently. To do this, we pick an irreducible polynomial $E(x)$ such that $\deg(E) > \deg_x(Q) = d_x$. Then we can look at $Q(x, y)$ as a polynomial in $\mathbb{F}'[y]$ where $\mathbb{F}' = \frac{\mathbb{F}[x]}{E(x)}$. This reduces the problem to finding roots of Q over $\mathbb{F}'$ and then we can pull them back to $\mathbb{F}[x]$ since degree of any factor is less than E. We can use Berlekamp's algorithm to find the roots of Q over $\mathbb{F}'$. [1] □

Note that we are really being wasteful with degree of R. Only term which contributes the largest is $A_l(x)(f(x))^l$, which has degree $d_x + kl$. We can balance these degrees so that all the terms contribute roughly the same number. Since $f(x)^j$ is going to contribute significantly less than $f(x)^l$ in degree of $R(x)$, we can balance that out by increasing degree of A_j.

Let $\deg(A_j) = d_j$. Then we want $d_i + ki = D$ for all i, where $D = \lfloor\sqrt{2kn}\rfloor$. Then if the linear system in step 1 of ListDecode has non-trivial solutions, rest of the proof goes through for all $t > D$. Therefore, we just need to assure that the linear system above has a non-zero solution. Note that

number of variables in this system is

$$\begin{aligned}\sum_{i=0}^{\lfloor \frac{D}{k} \rfloor}(D-ki+1) &= (D+1)\left(\left\lfloor \frac{D}{k} \right\rfloor + 1\right) - \frac{k}{2}\left\lfloor \frac{D}{k} \right\rfloor \left(\left\lfloor \frac{D}{k} \right\rfloor + 1\right) \\ &\geq \left(\left\lfloor \frac{D}{k} \right\rfloor + 1\right)\left(D+1-\frac{D}{2}\right) \\ &\geq \left(\left\lfloor \frac{D}{k} \right\rfloor + 1\right)\left(\frac{D+2}{2}\right) \\ &\geq \frac{(D+1)(D+2)}{2k}\end{aligned}$$

The last inequality follows since $\left\lfloor \frac{D}{k} \right\rfloor \geq \frac{D-k+1}{k}$. Therefore, system is underdetermined when $\frac{(D+1)(D+2)}{2k} > n$ which is satisfied for $D = \lfloor \sqrt{2kn} \rfloor$. This proves that we can always find a Q as required, and then rest of the algorithm works for all $t > \sqrt{2kn}$. Thus, we have following theorem:

Theorem 7. *If we choose* $Q(x,y) = A_l(x)y^l + \cdots + A_0(x)$ *where* $\deg(A_j) = D - kj$*, for* $D = \lfloor \sqrt{2kn} \rfloor$*, then ListDecode correctly outputs list of all polynomials* f *with* $\deg(f) < k$ *and* $f(a_i) = y_i$ *for at least* t *values of* i*.*

The quantity $a + kb$ is called as $(1,k)$-weighted degree of monomial $x^a y^b$, and $(1,k)$-weighted degree of a polynomial is just the maximum $(1,k)$-weighted degree of its monomials. The arguement above shows that $(1,k)$-weighted degree of $Q(x,y)$ is actually a better quantity to optimize over rather than the degree.

References

[1] Elwyn R. Berlekamp. Factoring polynomials over large finite fields*. In *SYMSAC '71*, 1971.