Lecturer: Venkatesan Guruswami Scribe: Sean Guan

1 List Decoding

Last time we introduced the idea of list decoding and proved the list decoding capacity. We summarize the results as follows:

Definition 16.1. A code $C \subseteq \Sigma^n$ is (p, L)-list decodable if for all $y \in \Sigma^n$, $|B(y, pn) \cap C| \leq L$.

Note that $L = 1$ gives us unique decoding. More precisely, a code is $(p, 1)$-list decodable if and only if has relative distance $\delta > 2p$.

For a fixed alphabet size $|\Sigma| = q$, we are interested in the best rate of any q-ary family of (p, L)-list decodable codes. Let's define this quantity as $R_L^{(q)}(p)$.

Definition 16.2. $R_L^{(q)}(p)$ is the best rate of any q-ary family of (p, L)-list decodable codes.

And we define the list decoding capacity as:

Definition 16.3. The list decoding capacity for q-ary codes of list decoding radius p is $R^{list,q}(p)$, where

$$R^{list,q}(p) = \limsup_{L \to \infty} R_L^{(q)}(p)$$

Note that we have the following bounds on $R_L^{(q)}(p)$:

Theorem 16.4. *For all $q \geq 2$:*

1. $R_L^{(q)}(p) \geq 1 - h_q(p) - 1/L$
2. $R_L^{(q)}(p) < 1 - h_q(p)$

Here, $h_q(x)$ is the q-ary entropy function. We showed last time that random codes satisfy *1*, and *2* was an exercise from homework, where we got an upper bound of $1 - h_q(p) - \frac{(1-2p)}{8} \cdot p^L$. It follows from these bounds that we have the following list decoding capacity for q-ary codes of list decoding radius p:

Theorem 16.5 (List Decoding Capacity)**.** $R^{list,q}(p) = 1 - h_q(p)$

Now for a fixed list decoding radius p, our goal is to achieve the list decoding capacity explicitly. This is still an open question for a fixed alphabet size q, but for large q it has been resolved. Note that for large q, the capacity is approximately:

$$\begin{aligned} 1 - h_q(p) &= 1 - \left(p \log_q(q-1) + p \log_q \left(\frac{1}{p} \right) + (1-p) \log_q \left(\frac{1}{1-p} \right) \right) \\ &= 1 - \left(p \log_q(q-1) + \frac{h(p)}{\log_2 q} \right) \\ &\approx 1 - p - \frac{1}{\log_2 q} \end{aligned}$$

where $h(x)$ is the binary entropy function, and we used the fact that for large q, $\log_q(q-1) \approx 1$, and $0 < h(p) < 1$ is small in comparison with $\log_2 q$. So to get within ϵ of the capacity, it is necessary that $q \geq \exp(1/\epsilon)$. For such large q there are explicit constructions of list decodable codes of rate R that achieve:

$$R = 1 - p - \epsilon \iff p = 1 - R - \epsilon$$

with polynomial time encoding and decoding algorithms and alphabet size $q = \exp(\text{poly}(1/\epsilon))$. The construction we will discuss in a later lecture is the Folded Reed-Solomon codes, which requires $q = n^{1/\epsilon^2}$, where n is the size of the finite field on which we define the Reed-Solomon code.

It is incredible that list decoding can deal with a fraction of error $p = 1 - R - \epsilon$ (for large q and at capacity). Imagine a very benigh erasure model where the adversary always erases the last p fraction of codewords. Even in this case, we have a limit of $p \leq 1 - R$ if we want to recover a unique codeword from the erasure. List decoding gets to this limit.

If we insist on unique decoding, even for codes achieving the singleton bound, the fraction of errors we can tolerate is limited to $p < \delta/2 = \frac{1-R}{2}$, which is strictly less that $1/2$. List decoding can deal with almost two times as many errors.

Recall that the Johnson radius gives us the following baseline for list decoding any codes:

Theorem 16.6. *Let C be a q-ary code of block length n and relative distance δ, then C is $(J_q(\delta), \text{poly}(n))$-list decodable, where $J_q(\delta)$ is the Johnson radius:*

$$J_q(\delta) = \left(1 - \frac{1}{q}\right)\left(1 - \sqrt{1 - \frac{q\delta}{q-1}}\right)$$

For all $q \geq 2$, note that $J_q(\delta) \geq 1 - \sqrt{1-\delta} = 1 - \sqrt{R}$ for codes on the singleton bound. So we can ask if there is an efficient algorithm to list-decode C up to error fraction $p = 1 - \sqrt{R}$. This is less than the list decoding capacity $1 - R$ but certainly greater than $\frac{1-R}{2}$ for unique decoding.

2 Goldreich-Levin Theorem and List Decoding the Hadamard Code

The Goldreich-Levin Theorem proves the existence of probabilistic learning algorithms for linear functions. The linear function $l_a : \mathbb{F}_2^k \to \mathbb{F}_2$ is defined as inner product with a over $\mathbb{F}_2$:

$$l_a(x) = a \cdot x = \sum_{i=1}^{k} a_i x_i \mod 2$$

Now give an arbitrary function $f : \mathbb{F}_2^k \to \mathbb{F}_2$ and oracle access to f, we want to find all $a \in \mathbb{F}_2^k$ such that f agrees with l_a on at least $(\frac{1}{2} + \epsilon)$ fraction of inputs. The Goldreich-Levin theorem states that this is possible in the following sense:

Theorem 16.7 (Goldreich-Levin)**.** *There is a probabilistic algorithm such that given $f : \mathbb{F}_2^k \to \mathbb{F}_2$ and $\epsilon > 0$, it runs in time $\text{poly}(n, 1/\epsilon)$ and outputs a list L so that for every $a \in \mathbb{F}_2^k$:*

$$\Pr_{x \sim \mathbb{F}_2^k}[a \cdot x = f(x)] \geq \frac{1}{2} + \epsilon \implies \Pr[a \in L] \geq \frac{1}{2}$$

An application of the Goldreich-Levin theorem is that given a one-way function, we can always construct a one-way function with a hard-core predicate. More precisely, given a one-way permutation $g : \mathbb{F}_2^k \to \mathbb{F}_2$, we can produce another one-way permutation $f : \mathbb{F}_2^{2k} \to \mathbb{F}_2$ defined as:

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

where x and y stand for the first and last k bits of the input to f. It can be shown using the Goldreich-Levin theorem that the predicate $B(x, y) = x \cdot y$ is hard-core for f. [GL89]

Note that Theorem 16.7 can be rephrased in terms of list decoding Hadamard code. The function f is just a vector in $\mathbb{F}_2^{2^k}$, and can be viewed as the received word of the $[2^k, k, 2^{k-1}]$ Hadamard code. Our goal is to find all messages $a \in \mathbb{F}_2^k$ such that $\Delta(Had(a), f) \leq (\frac{1}{2} - \epsilon)2^k$ efficiently in the sense of Theorem 16.7. So essentially we want to list decode the Hadamard code up to a fraction of errors of $\frac{1}{2} - \epsilon$ (note that $\frac{1}{2}$ is also the minimum (relative) distance of the Hadamard code).

2.1 Unique Decoding of the Hadamard Code

Let's first consider the easier problem of decoding the Hadamard code up to a fraction of errors of $\frac{1}{4} - \epsilon$. Since the minimum relative distance of Hadamard code is $\frac{1}{2}$, this is the case of unique decoding. Here the decoding algorithm is a 2-query local decoder. The main idea is that to decode the ith bit a_i of a, we can query $f(x)$ and $f(x + e_i)$ and compute $f(x + e_i) + f(x)$ from random positions $x \in \mathbb{F}_2^k$. Since $f(x) = a \cdot x$ with high probability, $f(x + e_i) + f(x) = a \cdot (x + e_i) + a \cdot x = a \cdot e_i = a_i$ with high probability (e_i is the vector with 1 at the ith position and 0 elsewhere).

Algorithm 1 Hadamard Unique Decoder

```
for i = 1 to k do
    for j = 1 to s do
        Pick x ∈ F_2^k uniformly at random
        b_ij ← f(x + e_i) + f(x)
    end for
    b_i ← Maj(b_i1, ··· , b_is)
end for
return (b_1, ··· , b_k)
```

We made 2 queries in each iteration of the inner loop. For each query, $f(-)$ disagrees with $a \cdot -$ with a probability of $\frac{1}{4} - \epsilon$. So by union bound, $b_{ij} = a_i$ with probability at least $\frac{1}{2} + 2\epsilon$. Then by Chernoff bound, $Maj(b_{i1}, \cdots, b_{is}) \neq a_i$ with probability at most $e^{-\Omega(\epsilon^2 s)}$. It follows that:

$$\mathbf{Pr}[(b_1, \cdots, b_k) = (a_1, \cdots, a_k)] \geq (1 - e^{-\Omega(\epsilon^2 s)})^k$$

By picking $s = O(1/\epsilon^2 \cdot \log(k))$ with an appropriate constant, we see that the decoder succeeds with high probability, in particular satisfying Theorem 16.7. The runtime is $O(sk) = O(1/\epsilon^2 \cdot k \log(k))$ which is polynomial in k and $1/\epsilon$.

2.2 Decoding Hadamard Code up to $\frac{1}{2} - \epsilon$

For the algorithm above to work, it is necessary that the error fraction is less than $\frac{1}{4}$. This is no longer the case when we only have the upper bound of $\frac{1}{2} - \epsilon$. We can construct a new function from f which satisfy the requirement and use it in place of f in Algorithm 1:

1. Pick $x_1, \cdots, x_t \in \mathbb{F}_2^k$ independently at random.
2. *Assume* that we know $a \cdot x_1, \cdots, a \cdot x_t$.
3. Define the function $A_{x_1,\cdots,x_t}$ as:

$$A_{x_1,\cdots,x_t}(z) := Maj\{f(z + x_i) + a \cdot x_i\}$$

Now we claim that with high probabilty over the choices of x_i, we obtain:

$$\mathop{\mathbf{Pr}}_{z}[a \cdot z \neq A_{x_1,\cdots,x_t}(z)] < \frac{1}{4}$$

To see this, we define variables $E_1, \cdots, E_t$ where $E_j = 1$ when $f(z + x_j) = a \cdot (z + x_j)$ and 0 otherwise. By hypothesis on f, we know $\mathbb{E}[E_j] \geq \frac{1}{2} + \epsilon$. Now:

$$\mathop{\mathbf{Pr}}_{x_1,\cdots,x_t,z}[a \cdot z \neq A_{x_1,\cdots,x_t}(z)] = \mathbf{Pr}\left[\sum_j E_j < t/2\right] \tag{16.1}$$

$$\leq \mathbf{Pr}\left[\left|\sum_j E_j - \mathbb{E}\left[\sum_j E_j\right]\right| > \epsilon t\right] \tag{16.2}$$

$$\leq \frac{\mathbf{Var}[\sum_j E_j]}{\epsilon^2 t^2} \tag{16.3}$$

$$= \frac{\sum_j \mathbf{Var}[E_j]}{\epsilon^2 t^2} \tag{16.4}$$

$$\leq \frac{1}{4\epsilon^2 t} - \frac{1}{4t} \tag{16.5}$$

where 16.1 follows by definition of $A_{x_1,\cdots,x_t}$. 16.2 follows from linearity of expectation. 16.3 is Chebyshev's bound. 16.4 follows from the fact that the x_j's are pairwise independent. 16.5 uses the fact that E_j is a Bernoulli(p) variable with $p \geq \frac{1}{2} + \epsilon$ and the variance of a Bernoulli(p) random variable is $p(1-p)$.

Taking $t = 2/\epsilon^2$ gives:

$$\mathop{\mathbf{Pr}}_{x_1,\cdots,x_t,z}[a \cdot z \neq A_{x_1,\cdots,x_t}(z)] \leq \frac{1}{8} - \frac{\epsilon^2}{8}$$

and this implies that

$$\mathop{\mathbf{Pr}}_{x_1,\cdots,x_t}\left[\mathop{\mathbf{Pr}}_{z}[a \cdot z \neq A_{x_1,\cdots,x_t}(z)] \leq \frac{1}{4} - \frac{\epsilon^2}{4}\right] \geq \frac{1}{2}$$

using Markov's inequality.

This shows that we can call $A_{x_1,\cdots,x_t}$ instead of querying f in Algorithm 1 to recover a. However, we assumed that we know $a \cdot x_1, \cdots, a \cdot x_t$ in advance, which is not true. A naive solution is to simply try all guesses for $a \cdot x_1, \cdots, a \cdot x_t$ and run Algorithm 1 once for each set of guesses. This allows us to recover all possible a such that $\Delta(Had(a), f) \leq (\frac{1}{2} - \epsilon)2^k$ with high probability, but there are 2^t set of guesses in total, which makes the runtime exponential in $1/\epsilon$.

To fix this issue, observe that we only required $x_1, \cdots, x_t$ to be pairwise independent (16.4). This

suggests that we only need to pick $y_1, \cdots, y_{\log_2(t)}$ points in $\mathbb{F}_2^k$ independently at random, then take their span to be $x_1, \cdots, x_t$. The algorithm still works the same, but note that in order to try all guesses for $a \cdot x_1, \cdots, a \cdot x_t$, now we only need to try all guesses for $a \cdot y_1, \cdots, a \cdot y_{\log_2(t)}$, because the y_i's span all the x_i's and inner product is a linear function. This eliminates the $\exp(1/\epsilon)$ factor in the runtime and we obtained an algorithm satifying Theorem 16.7. For more details, see [Tul05]

Remark 16.8. The idea of local list decoding has other applications in complexity theory. For example, the Reed-Muller codes are useful in hardness amplification.

3 List Decoding Reed-Solomon Codes up to the Johnson Radius

Recall the Reed-Solomon codes $RS_{\mathbb{F}_q}[S, k+1]$, where $S = \{a_1, \cdots, a_n\} \subset \mathbb{F}_q$ is the set of evaluation points and the rate is $R = \frac{k+1}{n}$:

$$f \in \mathbb{F}_q[x]_{\deg \leq k} \xrightarrow{Enc} (f(a_1), \cdots, f(a_n)) \xrightarrow{Noise} (y_1, \cdots, y_n) \xrightarrow{Dec} f$$

Suppose the noise introduced e errors. Let $t = n - e$ be the number of correct positions, i.e. $|\{i | y_i = f(a_i)\}| = t$.

We have seen that when $t > \frac{n+k}{2}$, we can uniquely decode the Reed-Solomon code efficiently using the Welsh-Berlekamp algorithm. Now we introduce an algorithm to list-decode the Reed-Solomon codes with a smaller t. More precisely, our goal is:

Given $(a_i, y_i) \in \mathbb{F}_q^2$, find all polynomials $f \in \mathbb{F}_q[x]$ with degree at most k such that $f(a_i) = y_i$ for at least t values of $i \in [n]$.

To motivate the algorithm, first recall the Welsh-Berlekamp algorithm for unique decoding. The first step of the algorithm is to find a nonzero polynomial $Q(x, y) \in \mathbb{F}_q[x, y]$ such that:

$$Q(x, y) = A_0(x) + A_1(x)y$$

$$\deg(A_0) \leq D$$

$$\deg(A_1) \leq D - k$$

$$Q(a_i, y_i) = 0 \ \ \forall i$$

Now define $R(x) := Q(x, f(x))$, by the properties of Q, we see that:

$$\deg(R) \leq D$$

$$f(a_i) = y_i \implies R(a_i) = 0$$

Since there are t positions where $f(a_i) = y_i$, $R(x)$ has at least t roots. If $t > D$, we know $R(x) = Q(x, f(x))$ must be the zero polynomial, and it follows that $y - f(x)$ divides $Q(x, y)$.

The first step can be done efficiently by solving a linear (homogeneous) system where the coefficients of A_0, A_1 are the unknowns, and the constraints are $Q(a_i, y_i) = 0$. For a nonzero Q to exist, the number of unknowns needs to exceed the number of constraints, namely:

$$(D + 1) + (D - k + 1) > n \iff D > \frac{n + k}{2} - 1$$

So it is necessary that $t > \frac{n+k}{2}$.

The list decoding algorithm we are about to present [Sud96] generalizes the approach of the Welsh-Berlekamp algorithm. Our goal is to find a polynomial $Q(x, y)$ such that for all $f(x)$ we need to output, $y - f(x)$ is a factor of $Q(x, y)$ (or equivalently $Q(x, f(x)) = 0$). To achieve a lower t, we will interpolate a polynomial with higher degree in y. In particular, we propose:

$$Q(x, y) = A_0(x) + A_1(x)y + \cdots + A_L(x)y^L$$

where L will turn out to be the target list size. Again we want $Q(x, f(x))$ to have degree at most D (parameter to be set later), so we require $\deg(A_j(x)) \leq D - jk$.

Now we describe the algorithm more formally.

3.1 Algorithm

Step 1 Find a nonzero polynomial:

$$Q(x, y) = A_0(x) + A_1(x)y + \cdots + A_L(x)y^L \in \mathbb{F}_q[x, y]$$

such that $Q(a_i, y_i) = 0$ for all $i \in [n]$ and $\deg(A_j(x)) \leq D - jk$, where D, L are parameters to be set later.

Step 2 Find all $f(x) \in \mathbb{F}_q[x]$ such that $y - f(x)|Q(x, y)$, and output those which satsify $\deg(f) \leq k$ and $f(a_i) = y_i$ for at least t values of $i \in [n]$.

For a nonzero Q to exists, the number of unknowns needs to exceed the number of constraints. So in step 1 we require:

$$\sum_{j=0}^{L}(D - jk + 1) > n$$

Rearranging, this becomes:

$$D \geq \frac{n}{L+1} + \frac{kL}{2}$$

3.2 Analysis

Similar to the first step in Welch-Berlekamp algorithm, step 1 can be done efficiently since it is a linear system. Now for the correctness of step 2:

Claim 16.9. *Given that $t > D$, step 2 outputs all $f(x) \in \mathbb{F}_q[x]$ such that $\deg(f) \leq k$ and $f(a_i) = y_i$ for at least t values of $i \in [n]$.*

Proof. It's enough to show that if $\deg(f) \leq k$ and $f(a_i) = y_i$ for at least t values of $i \in [n]$, then $y - f(x)$ is a factor of $Q(x, y)$. Define $R(x) := Q(x, f(x))$. By construction of Q, we know $\deg(R) \leq D$. Since $f(a_i) = y_i \implies R(a_i) = Q(a_i, f(a_i)) = Q(a_i, y_i) = 0$, we know R has at least t roots. Since $t > D$, $R(x) = Q(x, f(x))$ must be the zero polynomial.

It remains to show that $Q(x, f(x)) = 0$ implies $y - f(x)|Q(x, y)$. For this we use the fact that $y - f(x)$ is monic in y. In particular, we can divide $Q(x, y)$ by $y - f(x)$ using the remainder algorithm to get:

$$Q(x, y) = (y - f(x))P(x, y) + g(x)$$

Now since $Q(x, f(x)) = 0$, we see that $g(x)$ must be the zero polynomial, so $y - f(x)$ divides $Q(x, y)$. □

Lastly, for the list size and efficiency of step 2:

Claim 16.10. *Given that $t > D$, step 2 outputs at most L polynomials efficiently.*

Proof. We use the fact that $\mathbb{F}_q[x, y] = (\mathbb{F}_q[x])[y]$ and that $\mathbb{F}_q[x]$ is an integral domain. In general, for an integral domain R, a degree L polynomial in $R[y]$ has at most L roots. Since $Q(x, y)$ has degree L in y, it has at most L linear factors of the form $y - f(x)$.

For efficiency, it suffices to give an efficient algorithm to factor of $Q(x, y)$ into linear factors in y. For this, pick an irreducible polynomial $E(x) \in \mathbb{F}_q[x]$ such that $\deg(E)$ is greater than the degree of $Q(x, y)$ in x. Now $F = \mathbb{F}_q[x]/(E(x))$ is a field, and the quotient map induces a map on the polynomial rings $(\mathbb{F}_q[x])[y] \to F[y]$. Let Q' be the image of Q in $F[y]$. Since $\deg(E)$ is greater than the degree of Q in x, each linear factor of Q corresponds to a unique linear factor of Q', so it suffices to factor Q' in $F[y]$, for example, using Berlekamp's algorithm. [Ber71] □

Remark 16.11. For $L = 1$, the algorithm reduces to the Welch-Berlekamp algorithm.

As mentioned in the beginning, our goal is to minimize t. Since the only constraint on t is $t > D$, we need to minimize D. For example, taking

$$D = \frac{n}{L} + \frac{kL}{2}$$

We have $D \geq 2\sqrt{\frac{nk}{2}} = \sqrt{2nk}$ at $L = \left\lceil \sqrt{\frac{2n}{k}} \right\rceil$. In this case we can take $t \approx \sqrt{2nk} \approx \sqrt{2R}n$, which gives an error fraction is $1 - \sqrt{2R}$. Note that this does not achieve the baseline we proposed earlier. In the next lecture we will modify our algorithm to achieve the baseline $e = 1 - \sqrt{R}$.

References

[Ber71] Elwyn R. Berlekamp. Factoring polynomials over large finite fields. *SYMSAC '71*, 1971. 3.2

[GL89] Oded Goldreich and Leonid A Levin. A hard-core predicate for all one-way functions. *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32, 1989. 2

[Sud96] Madhu Sudan. Decoding of reed solomon codes beyond the error-correction bound. 1996. 3

[Tul05] Madhur Tulsiani. Notes for lecture 8. https://cs.stanford.edu/people/trevisan/pacc/lecture08.pdf, 2005. CS 294, UC Berkeley, Professor Luca Trevisan. 2.2