1 Local list decoding

We extend the local (unique) decoding definition to the list-decoding setting, but we need to be careful. Consider the following “bad” definition:

Definition 1. Fix an \([n, k]_q\) code \(C\). An algorithm \(R(\tau, L)\) locally list-decodes \(C\), if given \(w \in \{0, 1\}^n\) and \(i \in [k]\) the algorithm \(R\) outputs a list of values \(S\) of cardinality \(L\), such that if \(\text{agr}(w, C(x)) \geq \tau n\) then \(x_i \in S\).

This definition is useless. For example, if the code is binary the algorithm can output \(\{0, 1\}\) and always be right. The problem is that we lost the consistency between the solutions. The correct definition is:

Definition 2. Fix an \([n, k]_q\) code \(C\). An algorithm \(R(\tau, L)\) locally list-decodes \(C\), if for every \(w \in \{0, 1\}^n\) and every codeword \(C(x)\) that has \(\tau\) agreement with \(w\), there exists a \(j \in [L]\) such that for every \(i \in [k]\) the algorithm’s value is correct, namely, \(R(w, j, i) = x_i\).

We are interested in situations where there are few codewords (at most \(L\)) that are \(\delta\)-close to \(w\). We require that every such codeword has a corresponding index \(j \in [L]\) such that \(R(w, j, \cdot)\) locally decodes it (i.e., given \(i\), it outputs the \(i\)-th coordinate of the solution, and it runs in poly-logarithmic time). Notice also that we allow \(R\) to include garbage solutions among the \(L\) solutions – we only insist that all the good solutions appear in the list.

Again, the algorithm \(R\) may be randomized, and then we require that for every input \(w\) and codeword \(C(x)\) with \(\text{agr}(w, C(x)) \geq \tau n\), there exists \(j \in [L]\) such that for all \(i \in [k]\), \(R(w, j, i) = x_i\) with high probability over the internal random coins of \(R\).

2 Local list decoding Reed-Muller codes – the STV construction

We fix a field \(\mathbb{F}_q\), a subset \(H \subseteq \mathbb{F}_q\) and an integer \(m\). Let \(n = q^m\) and \(k = |H|^m\) and identify \(H^m\) with \([k]\) and \(\mathbb{F}_q^m\) with \([n]\). Recall that to encode a string \(x \in \mathbb{F}_q^k\) to a codeword \(p: \mathbb{F}_q^m \rightarrow \mathbb{F}_q\) we first put \(x\) on \(H^m\), that is \(p(i) = x_i\) for every \(i \in [k]\), and then extend \(p\) to be an \(m\)-variate polynomial of degree at most \(|H| - 1\). We let \(\text{deg} = m(|H| - 1)\) be the largest possible total degree of \(p\).

We want to show local list-decoding for the Reed-Muller code. We are given as input:

- Parameters \(q, H \subseteq \mathbb{F}_q, m, \tau, L = \frac{\tau}{\tau}, j \in [L]\) and \(a \in [k] = H^m\).
- A noisy word \(f : \mathbb{F}_q^m \rightarrow \mathbb{F}_q\) (or equivalently, \(f : [n] \rightarrow \mathbb{F}_q\)).
The parameters are set such that for every \( f \in \mathbb{F}_q^m \) there are at most \( L \) codewords having \( \tau n \) agreement with \( f \). Our goal is to design a (probabilistic) reconstruction algorithm \( R \) such that for every \( f \) (a possibly corrupted codeword) and every codeword \( c \) with \( \tau \) (relative) agreement with \( f \), there exists a \( j \in [L] \) such that \( R(f, j, \cdot) \) locally decodes \( c \).

### 2.1 A naive attempt

We first try the following:

1. Let \( \ell \) be a random line that passes through \( a \in [k] = H^m \). To pick \( \ell \) you can, e.g., pick another random point \( z \in \mathbb{F}_q^m \) and pass the line connecting \( a \) and \( z \).

2. Compute the restriction \( f \circ \ell \) of \( f \) on the line \( \ell \). For this we need to query \( f \) on each of the \( q \) points that lie on the line \( \ell \). Recall that if \( c \) is a true codeword (viewed as a low-degree multivariate polynomial \( c : \mathbb{F}_q^m \rightarrow \mathbb{F}_q \)) then \( c \circ \ell \) is a univariate polynomial of degree \( \deg(c) \leq \deg \).

3. Find all degree-\( \deg \) univariate polynomials \( h_1, \ldots, h_L \) that agree with \( f \circ \ell \) for at least \( \tau/2 \) fraction of the points, using the algorithm for list-decoding Reed-Solomon codes.

4. Output \( h_j(a) \).

The intuition behind the algorithm is that on average the codeword \( c \) that we seek has non-negligible \( \tau \) agreement with \( f \). Therefore, it should also have about the same \( \tau \) agreement with a random line passing through \( a \) (but, is it true?). If so, \( c \) will appear as one of the solutions in the list decoding of \( f \) restricted to \( \ell \) as we wish. There are two problems with the naive approach:

1. The list-decoding algorithm should be worst-case with respect to \( a \), and in such a situation we cannot prove that indeed w.h.p. over the lines passing through \( a \), the restriction to the line has \( \tau/2 \) agreement with \( f \).

2. We do not keep consistency. I.e., it is possible that \( c \) appears as the first solution in the list for \( a \), but as the second solution in the list of some \( a' \neq a \). In fact, the index in the set of solutions may also depend on the internal randomness (that determines the line).

### 2.2 A second attempt

To address the second point above, we need a consistent way to separate different solutions. We do that by choosing a “filtering” point \( z \in \mathbb{F}_q^m \). For every \( y \in \mathbb{F}_q \), we will have one solution, that will consist of those polynomials whose evaluation on \( z \) is \( y \) (and so we assume \( q \gg L \)). We will see that a random \( z \) is a good “splitting” filter.

We describe the algorithm \( A_{z,y} \). On input \( f : \mathbb{F}_q^m \rightarrow \mathbb{F}_q \), \( a \in H^m \), \( \deg \) and \( \tau \):

1. Let \( b = z - a \) and let \( \ell \) be the line \( a + tb \) for \( t \in \mathbb{F}_q \). Note that \( \ell(0) = a \) and \( \ell(1) = z \).

2. Find all degree-\( \deg \) univariate polynomials \( h_1, \ldots, h_L \) that agree with \( f \circ \ell \) on at least \( \tau/2 \) fraction of the points, using the list-decoding algorithm for Reed-Solomon codes \( L \leq 4 \frac{\tau}{\tau} \).

3. If there exists a unique \( i \in [L] \) for which \( h_i(z) = y \), output \( h_i(0) \).
We now want to show that a random $z$ splits well.

**Lemma 3.** Fix $\varepsilon > 0$ and assume $q \geq \frac{16(|H| m + 1)}{\tau^2 \varepsilon}$. Fix $f : \mathbb{F}_q^m \rightarrow \mathbb{F}_q$. The probability over a random $z, a \in \mathbb{F}_q^m$ that $A_{z, p(z)}(a)$ outputs $p(a)$ is at least $1 - \varepsilon$. The algorithm runs in time $\text{poly}(m, q)$.

**Proof.** The proof is similar (at least in spirit) to Theorem 5 from Lecture 1, where the main difference is that there we had little noise and we were in the unique decoding setting so we used unique decoding of Reed-Solomon, and now we have high noise, so we are in the list-decoding setting and we use list-decoding of Reed-Solomon codes.

We define two bad events. Let $B_1$ be the event that $p$ and $f$ have less than $\frac{\tau}{2}$ agreement on $\ell$ and let $B_2$ be the event that there exists a pair $(i, j)$ such that $h_i(z) = h_j(z)$. If neither $B_1$ nor $B_2$ occur, then $A_{z, p(z)}$ outputs $p(a)$ on input $a$ as the parameters were set to match the Reed-Solomon list-decoding algorithm. We will see this afterwards.

We shall now bound the probability for $B_1$ and the probability for $B_2$.

**Claim 4.** $\Pr_{z, a}[B_1] \leq \frac{4}{\tau q}$.

**Proof.** $\ell$ is uniquely determined by $a$ and $z$, so it is a random line. The points on $\ell$ are thus uniformly distributed and pairwise independent (why?). On any one point, the probability that $f$ agrees with $p$ is $\tau$. The expected number of agreements between $f$ and $p$ on $\ell$ is then $\tau q$. By Chebyshev, the probability that we deviate by half is at most $\frac{4}{\tau q}$ (check!).

**Claim 5.** $\Pr_{z, a}[B_2] \leq \frac{8 \deg}{\tau q}$.

**Proof.** For every $(i, j) \in [L]^2$, let $BAD_{i, j}$ be the event that $h_i(z) = h_j(z)$ (over a random $a$ and $z$). First, suppose $z$ is a random point on the line $\ell$ after it is fixed. Thus, the probability for $BAD_{i, j}$ is the probability over $z$ that $(h_i - h_j)(z) = 0$, which is at most $\frac{\deg}{q}$.

In fact, we can suppose this is the case. Instead of choosing $z$ and $a$ at random and then determining $\ell$ according to them, we could pick $\ell$ at random first and then choose $a$ and $z$ at random from $\ell$.

So indeed $\Pr[BAD_{i, j}] \leq \frac{\deg}{q}$. By the union-bound, $\Pr[B_2] \leq \frac{L^2 \deg}{2q \tau \varepsilon}$. By the list-decoding algorithm for Reed-Solomon we saw in the previous lecture, $L \leq \frac{4}{\tau}$ so $\Pr[B_2] \leq \frac{8 \deg}{\tau q}$.

To conclude, pick $q$ large enough so both terms will be at most $\frac{\varepsilon}{2}$, and this yields the lower bound on $q$.

The solution for the list-decoding of Reed-Solomon codes is given provided $\frac{\tau}{2} \geq \sqrt{\frac{2 \deg}{q}}$. We choose the parameters such that this requirement is met, as $q \geq \frac{8 \deg}{\tau^2 \varepsilon}$.

The running time of the algorithm is $\text{poly}(m, q)$.

As we said the running time of the algorithm is $\text{poly}(m, q)$. A possible choice of parameters is that given $k, \varepsilon$ and $\tau$ we set $|H| = \log k / \varepsilon\tau$, $m = \log k / \log |H|$ and $q = \frac{16(|H| m + 1)}{\tau^2 \varepsilon}$. Thus, $q = \text{poly}(|H|)$ (so the encoding length is polynomial in the input size) and $\text{poly}(m, q)$ is $\text{poly}(\log k, 1/\varepsilon, 1/\tau)$. 

3
2.3 The local list decoding algorithm

In the last subsection, we showed that for a random pair \((z, a)\), if \(p\) and \(f\) have \(\tau\)-agreement then the algorithm will output \(p(a)\) with high probability. How do we obtain our list-decoding algorithm?

We set \(\epsilon = \frac{1}{16}\) in Lemma 3. We can therefore deduce that for every \(f \in \mathbb{F}_q^m\) that has \(\tau\)-agreement with some codeword \(p : \mathbb{F}_q^m \to \mathbb{F}_q\), there exists a good splitting point \(z_0 \in \mathbb{F}_q^m\) such that

\[
\Pr_{a \in \mathbb{F}_q^m} [A_{z_0,p(z_0)}^f(a) = p(a)] \geq 1 - \epsilon = 1 - \frac{1}{16}.
\]

We let \(L = [n] \times \mathbb{F}_q\). Given \(j \in L\) we interpret it as \((z_0, y_0) \in L = [n] \times \mathbb{F}_q\). We then let \(R(f, j, i)\) for \(j = (z, y) \in [L]\) and \(i \in \mathcal{L}\) to run the local unique-decoding algorithm of Theorem 5 in Lecture 1 on the function \(A_{j=(z,y)}^f\). Verify that indeed \(R\) is local and that it list-decodes \(f\) (i.e., for any \(p\) with \(\tau\)-agreement with \(f\) there exists a \(j\) such that \(R(f, j, \cdot)\) decodes \(p\)). Overall, we obtained:

**Theorem 6.** For every \(k, \tau\) and \(q = \Omega\left(\frac{\log^2 k}{\tau^2}\right)\), the multivariate reconstruction problem above can be locally solved in time \(\text{poly}(q, 1/\tau)\).

Let us now unfold the algorithm. We have black-box access to a noisy \(f : \mathbb{F}_q^m \to \mathbb{F}_q\) that has \(\tau\) relative agreement with \(p : \mathbb{F}_q^m \to \mathbb{F}_q\) of degree \(\deg\). We want to find the value of \(p\) at \(a \in \mathbb{F}_q^m\).

So fix \((z_0, y_0) \in \mathbb{F}_q^m \times \mathbb{F}_q\). Here is what we do:

- We pass a random degree 3 curve through the given \(a\).
- For each of the \(q\) points \(x_1, \ldots, x_q\) on the curve, we run \(A_{z_0,y_0}^f\) and get a value. I.e., we pass the line connecting \(z_0\) and \(x_i\), we query all the points on the line, we run the list decoding algorithm with \(\tau/2\) agreement and we filter by \((z_0, y_0)\). We output a value if and only if we get a unique answer.
- We run the unique-decoding algorithm on the the \(q\) points and the answers (for uni-variate polynomials of degree \(2\deg, \deg = m(|H| - 1)\)). We get a unique polynomial, and we output its value on the point \(a\) on the line.

3 Local List decoding concatenated codes

To transform the above code to a binary one, we concatenate it with the Hadamard code. Recall that for a string \(z \in \{0, 1\}^k\), the \(w\)-th coordinate of \(\text{Had}(z) \in \{0, 1\}^{2k}\) is \(\langle z, w \rangle_2\). We saw the Hadamard code has good list-decoding properties:

**Lemma 7.** For every \(k\), \(\text{Had} : \{0, 1\}^k \to \{0, 1\}^{2k}\) is \((\frac{1}{2} + \epsilon, \frac{4}{2})\)-list-decodable.

There is an efficient list-decoding algorithm for the Hadamard code, although for our purpose, exhaustive search in time \(\text{poly}(2^k)\) will do.
Encoding  To encode a string $x \in \mathbb{F}_2^k$, first compute $y = C(x) \in \mathbb{F}_q^{n_0}$. Then, encode each coordinate of $y$ as a $q$-bits string using $\text{Had} : \{0, 1\}^{\log q} \to \{0, 1\}^q$. That is,

$$C'(x) = \text{Had}(y_1) \circ \ldots \circ \text{Had}(y_{n_0}).$$

The encoding is of length $n = n_0 \cdot q$.

Decoding  We first list-decode each symbol of the inner code and then list-decode the outer code. Given oracle access to a word $f \in \mathbb{F}_2^n$ viewed as a function $f : [n_0] \times [q] \to \{0, 1\}$, assume $f$ has $\tau \geq \frac{1}{2} + \varepsilon$ agreement with some codeword. Set $\delta = \frac{c^3}{32}$ and let $C$ be the $[n_0, k]_q$ code that is $(\delta, L_0)$-list-decodable. The decoding is as follows:

1. For every $i \in [n_0]$, let $f_i : [q] \to \{0, 1\}$ be the restriction $f_i(j) = f(i, j)$. Viewed as a word $f_i \in \{0, 1\}^q$, apply list-decoding of $\text{Had}$ for codewords of distance at most $\frac{1}{2} - \frac{\varepsilon}{2}$ from $f_i$ and obtain a list of solutions $H_i \subseteq \mathbb{F}_2^{\log q}$. By Lemma 7, $|H_i| \leq \frac{16}{c^2}$.

2. For every $m \in \left[\frac{16}{c^2}\right]$, let $h_m = (H_1(m), \ldots, H_{n_0}(m)) \in \{0, 1\}^{n_0}$ where $H_i(m)$ is the $m$-th element of $H_i$ in some fixed order. Apply the list-decoding procedure of $C$ on $h_m$ for codewords with agreement at least $\delta$ with $h_m$ and obtain a list $L_m$ of cardinality at most $L_0$.

3. Output $L = \bigcup_{m=1}^{16/e^2} L_m$.

We first prove the list-decoding correctness. Consider a message $x \in \mathbb{F}_2^k$ and $f \in \mathbb{F}_2^n$ such that $d(C'(x), f) \leq \frac{1}{2} - \varepsilon$ and denote $y = C(x) \in \mathbb{F}_q^{n_0}$. By definition, we have that $\text{Pr}_{i,j}[f_i(j) = \text{Had}(y_i)_j] \geq \frac{1}{2} + \varepsilon$, so by an averaging argument there exists a set $I \subseteq [n_0]$ of cardinality at least $\frac{\varepsilon}{2} n_0$ such that for every $i \in I$, $\text{Pr}_{j}[f_i(j) = \text{Had}(y_i)_j] \geq \frac{1}{2} + \frac{\varepsilon}{2}$.

Therefore, for every $i \in I$ there exists $m \in \left[\frac{16}{c^2}\right]$ such that $H_i(m) = y_i$. Specifically, $\text{Pr}_{i,m}[H_i(m) = y_i] \geq \frac{\varepsilon}{2} \cdot \frac{\varepsilon^2}{16} = \frac{c^3}{32}$, so there exists $m_0 \in \left[\frac{16}{c^2}\right]$ for which $\text{Pr}_{i,m}[H_i(m_0) = y_i] \geq \frac{c^3}{32} = \delta$. Put differently, we have that $\text{agr}(h_{m_0}, C(x)) \geq \delta$ so the list $L_{m_0}$ includes $x$ and we are done. The list size is at most $L_0 \cdot \frac{16}{c^2} = \text{poly}(L_0, \log(\frac{1}{\varepsilon}))$.

Another alternative is to take the whole list of $O(\frac{1}{\varepsilon^2})$ points output at each point, and input them to the RS list-decoding algorithm (that is willing to accept several choices for each point). We recommend the reader to check the details.

We now show that the running time is $\text{poly}(\log k, 1/\varepsilon)$. To see that, observe that the time-consuming step is $16/e^2$ calls to list-decoding of $q$-ary Reed-Muller codes, as the list-decoding of the Hadamard code can be done in $\text{poly}(q)$ time. By the previous section, the list-decoding of Reed-Muller takes $\text{poly}(q, 1/\tau)$ time.

We summarize the result of this lecture in the following theorem:

**Theorem 8.** There exists an explicit $[n, k]_2$ code that is $(\frac{1}{2} + \varepsilon, L)$ locally list-decodable where $n = \text{poly}(k, 1/\varepsilon)$ and $L = \text{poly}(n/\varepsilon)$. The (local) list-decoding procedure runs in time $\text{poly}(\log k, 1/\varepsilon)$.

The reason the list size is multiplied by $n$ is because we go over all possible splitting points. If we are given a good splitting point (or if we choose it at random) the list size is reduced to $\text{poly}(\frac{1}{\varepsilon})$. 

5