Notes for Lecture 11

In this lecture, we continue our study of list decodable codes that can recover from $(1-\epsilon)$ fraction errors. We saw expander-based construction of such list decodable codes with rate $\Omega(\epsilon^2)$ in the last two lectures. We improve the rate to optimal $\Omega(\epsilon)$ in today's lecture, but with an increase in decoding time from polynomial time to sub-exponential time ([1], pp 192–198,207–208). Note that this result is better than a purely existential result, which promises only exponential decoding time.

Recall the setting. Using an $(1/2, 1-\epsilon)$-expander as a tool, we can construct a $(1-\epsilon, L)$-list decodable code from a $(1/2, l, L)$-list recoverable code, for $l = O(1/\epsilon)$ and some $L > l$. In the process, we incur a loss of ϵ factor in rate. So, if the list recoverable code has rate $\Omega(\epsilon)$ (as seen in the last lecture for $L = O(1/\epsilon)$), the final list decodable code has rate $\Omega(\epsilon^2)$.

A quick note before proceeding: remember that α in (α, l, L)-list recovery or (α, L)-list decoding indicates the permissible *error* level (i.e., atmost αn of the n lists or positions are "corrupted").

11.1 Code Construction with rate $\Omega(\epsilon)$

To get a $(1-\epsilon, L)$-list decodable code with rate $\Omega(\epsilon)$, it is enough to get a $(1/2, l, L)$-list-recoverable code with rate $\Omega(1)$ (a constant independent of ϵ) by the above discussion. To get the desired list-recoverable code, we use a concatenation scheme similar to the one in the last lecture, with the only difference being the alphabet size used in concatenation. This lets us achieve a construction with constant rate but sub-exponential decoding time. In what follows, N refers to the block length of the concatenated code, $l = O(1/\epsilon)$, $L = 2^{O_\epsilon(\sqrt{N})}$, and the other L (as in L-wise independent family) is $O(l)$. The Θ_ϵ, O_ϵ or Ω_ϵ notation hides dependence on ϵ.

Concatenated code: The outer code is a Reed-Solomon code with alphabet size $2^{O(\sqrt{N})}$, and block and information length (n and k) $\Theta(\sqrt{N})$. The inner code is from a L-wise independent family of codes with alphabet size $O(1/\epsilon^2)$, and block and information length $\Theta_\epsilon(\sqrt{N})$. The dependence on ϵ here is required to match the inner and outer code's alphabet size. The Reed-Solomon outer code is $(3/4, O(l), L)$-list recoverable in $2^{O_\epsilon(\sqrt{N})}$ time by a brute force approach outlined in Section 11.1.2. The inner code, whose construction is explained in Section 11.1.1, is $(3/4, l, O(l))$-list recoverable in $2^{O_\epsilon(\sqrt{N})}$ time by a simple brute forcing over all codewords. Further, both the outer and inner code have rate $\Omega(1)$. Therefore, using observations from previous lectures, we see that the concatenated code is $(1/2, l, L)$-list recoverable with rate $\Omega(1)$ as desired and sub-exponential decoding time $2^{O_\epsilon(\sqrt{N})}$. The decoding time can be reduced to $2^{O_\epsilon(N^\gamma)}$ for any arbitrary power $\gamma < 1$ with a degradation in rate from $\Omega(1)$ to $\frac{1}{2^{O(1/\gamma^2)}}$ via a multi-level concatenation scheme [2] that extends the above basic scheme.

The next subsections provides some details of the inner and outer code to complete the concatenated code construction. At the end, the reader should convince himself that the inner, outer and hence concatenated codes have explicit specifications (using parity matrices or graphs for instance), claimed list-recoverability, polynomial-time encoding and sub-exponential time decoding procedures.

11.1.1 Construction of the inner code from a L-wise independent family

Let the inner code be represented as the $[n, k]_q$ code C, for appropriate n, k and prime power q used in the concatenation scheme. We now describe a construction of C that ensures C to be $(3/4, l, O(l))$-list recoverable as desired. We saw a probabilistic construction of such a code C from a L-wise independent family of codes in the last lecture. Each code in this family is specified by a $n \times O(k\mathrm{L})$ matrix A over the alphabet set $[q]$ and maps a message x as $C(x) = A.H_x$, where H_x is the column indexed by x of the parity check matrix H of a BCH code. A code picked at random from this L-wise independent family has good $(3/4, l, \mathrm{L}-1)$-list recoverability, and is constructed simply by picking a random matrix A.

We construct C by derandomizing the picking of the $n \times O(k\mathrm{L})$ matrix A. A naive derandomization of going over all such matrices would take $q^{O(nk\mathrm{L})}$ time, which is not sub-exponential for our choice of parameters (i.e., $k, n = \Theta_\epsilon(\sqrt{N})$ and $q, \mathrm{L} = \Omega_\epsilon(1)$). We use the method of conditional expectations to achieve the desired $q^{O(n\mathrm{L}+k\mathrm{L})}$ time. Proceed by picking the rows of A one after the other as follows. Consider the $(i+1)$st step when the first i rows are already fixed, the $(i+1)$st row is about to be picked and the remaining rows are unfixed. Out of all possible choices for the $(i+1)$st row, pick the one that minimizes a certain expectation made precise in the following definitions.

Definition 1 *Let C be a* supposed *$(3/4, l, L-1)$ list recoverable $[n, k]_q$ code, and let a sequence T of L messages, S of n lists, and A of L "correct positions" be chosen as:*

$$\begin{aligned} T = &\ (x_1, \ldots, x_L), \quad x_i \in [q]^k \\ S = &\ (S_1, \ldots, S_n), \quad S_i \subseteq [q],\ |S_i| \leq l \\ A = &\ (A_1, \ldots, A_L), \quad A_i \subseteq \{1, \ldots, n\},\ |A_i| = (1 - 3/4)n. \end{aligned}$$

Then, T, S, A is a counter-example to C if $\forall i = 1, \ldots, L, \forall j \in A_i, C(x_i)_j \in S_j$.

In the next definition, we view C as a random code with first few rows of A fixed and rest unfixed and equally likely to be one of the possible choices.

Definition 2 *For sequences T,S,A as in Definition 1, the indicator random variable $I_{T,S,A}$ and random variable X are defined as:*

$$\begin{aligned} I_{T,S,A} &= \begin{cases} 1, & \textit{if the particular choice of } T, S, A \textit{ is a counter-example to } C \\ 0, & \textit{otherwise} \end{cases} \\ X &= \sum_{T,S,A} I_{T,S,A} \end{aligned}$$

Initially when all rows of A are unfixed, the expected number $E[X]$ of counter-examples to C is less than 1, using arguments about L-wise independence from the last lecture. The $(i+1)$st row is fixed by picking the choice that minimizes $E[X \mid \text{first } (i+1) \text{ rows' values}]$, so the expectation of X (conditioned on the already picked rows) stays less than 1. Finally when A is completely fixed and C is no longer random, $X = 0$ as its expectation conditioned on the choices made is still less than 1. Thus, there are no counter-examples to the constructed $(3/4, l, \mathrm{L}-1)$-list recoverable code C.

We have one detail left to complete the procedure. The expectation of X conditioned on the already picked rows have to be computed *without* going over all codes in the family. If the first $(i+1)$ rows of A are fixed, the first $(i+1)$ positions of all codewords are fixed too, but the family of codes defined by the remaining choices in A is still L-wise independent. Hence we can compute exactly $E[I_{T,S,A} \mid \text{first } (i+1) \text{ rows' values}]$ for a particular T, S, A, much in the same way we derived the success probability of the probabilistic construction of C in the last lecture (see also [1], pp 197). We then compute exactly $E[X \mid \text{first } (i+1) \text{ rows' values}]$ by going over all possible choices of T, S, A.

The running time of the derandomization procedure is simply:

$$(nq^{O(k\mathrm{L})} \text{ choices of rows}) \times (q^{O(k\mathrm{L})}q^{O(nl)}2^{O(n\mathrm{L})} \text{ choices of } T, S, A) = q^{O(n\mathrm{L}+k\mathrm{L})} \text{ time.}$$

11.1.2 List recovering the Reed-Solomon outer code

We now give a sub-exponential time list recovery procedure for the Reed-Solomon outer code $[n_{RS}, k_{RS}]_{q_{RS}}$. Remember from the concatenation scheme that we need to decode a $(3/4, O(l), L)$-list recoverable code with $l = O(1/\epsilon)$ and $L = 2^{O_\epsilon(\sqrt{N})}$.

Given a sequence of n_{RS} lists, each of them a subset of $[q_{RS}]$ and size atmost $O(l)$, we decode it by a pure brute-force approach. The decoding procedure first guesses the "correct" subset of the n_{RS} lists, and then guesses the "correct" element from each of these lists to recover the messages. A recovered message corresponds to a *close* codeword, which is any codeword that "agrees" with atleast 1/4th of the n_{RS} lists. Here is the pseudocode for list recovering the RS code.

input: Lists $S_1, S_2, \ldots, S_{n_{RS}}$, $S_i \subseteq [q_{RS}]$, $|S| \leq O(l)$
output: Messages that correspond to all close codewords

for every $A \subseteq \{1, \ldots, n_{RS}\}$, $|A| = \frac{n_{RS}}{4}$ **do**
  for every choice of $C_i \in S_i$, $\forall i \in A$ **do**
    Interpolate to find $p(.)$ s.t.
    - $p(x_i) = C_i$, $\forall i \in A$,
    - $p(.)$ is of degree $k_{RS} - 1$.

The running time, determined by the possible choices of the two loops, is $2^{O(n_{RS})} l^{O(\frac{n_{RS}}{4})}$, which is sub-exponential $2^{O_\epsilon(\sqrt{N})}$ for our choice of parameters $n_{RS}, k_{RS} = \Theta(\sqrt{N})$ used in the concatenation scheme.

11.2 Linear time list decoding warmup

As a preparation for next lecture on expander-based codes with linear-time list decoding procedures, we see a slightly different scheme of using an expander as a tool to construct list decodable codes from list recoverable ones [3]. The linear-time list decodable code to be seen next time has a purely combinatorial construction, thereby making it very unlike the other list decodable codes that we have seen which involve some form of Reed-Solomon codes. But this code has rate worse than that of Reed-Solomon codes. It appears that linear-time list decoding is impossible if Reed-Solomon codes are involved, as such a decoding is equivalent to polynomial or long integers multiplication, a problem for which linear-time procedures are believed to be very unlikely.

We already know a *previous scheme* that uses an $(1/2, 1-\epsilon)$-expander to construct a $(1-\epsilon, L)$-list decodable code from a $(1/2, l, L)$-list recoverable code for $\epsilon = O(1/l)$. The **specific goal** now is to extend this result to reduce (δ, l, L)-list recoverability to $(1-\epsilon, L)$-list decoding for an arbitrarily small δ. We first discuss why the previous scheme won't work for small δ and then provide a different scheme that works. Let $G = (A, B, E)$ be a D-regular $(\delta, 1-\epsilon)$-expander graph with $|A| = |B| = n$. Such an expander is possible for $D = O(\frac{1}{\delta\epsilon})$. If A is interpreted as left vertex set and B as right, the expansion property ensures that every δn vertices on the left has an edge into every ϵn vertices on the right.

Previous scheme: Every left vertex $v \in A$ gets D "predictions" from all right vertices $w \in B$ to which it is connected, then all predictions are assembled into a list after removing duplicates and passed on to the decoding algorithm of the list-recoverable code. The size l of the list could be as high as D, so this scheme lets us reduce (δ, l, L)-list recoverability to $(1-\epsilon, L)$-list decoding only for $\epsilon = O(\frac{1}{\delta l})$. The constraint on ϵ, which follows from that on D, makes the scheme fail for small $\delta < 1/l$ (it works for $\delta = 1/2$ though).

New scheme: In the new decoding scheme, we don't let l grow with D. We fix l at $O(1/\epsilon)$ and construct the list of every left vertex $v \in A$ by taking only the l most frequently occuring elements in the predictions of v. Now, we prove by contradiction that this scheme indeed reduces (δ, l, L)-list recoverability to $(1-\epsilon, L)$-list decoding for an arbitrarily small δ, provided D is high enough.

Suppose that the new scheme doesn't work, which is to say that there exists a codeword with the permissible $(1-\epsilon)n$ errors but with the following property. Under the new scheme, the errors in the codeword corrupt the constructed lists of more than δn left vertices. Call some δn of these left vertices with *corrupted* lists as S, and call the right vertices corresponding to the *correct* elements of the codeword as T (so $|S| = \delta n, |T| = \epsilon n$). Now it follows that every bad vertex $v \in S$ has atmost D/l neighbours in the good set T, since the l most occuring elements in the D predictions of v are *not* from T. Therefore, a upper-bound on the number of edges between S and T is:

$$\#\,\text{edges(S,T)} \quad \leq \quad |S|\frac{D}{l} = \frac{\delta\epsilon n}{2}D \text{ (letting } l = 2/\epsilon). \tag{1}$$

And a lower-bound given by the expansion property of G is:

$$\left|\#\,\text{edges}(S,T) - |S|D\left(\frac{|T|}{n}\right)\right| \quad \leq \quad \lambda\sqrt{|S||T|}, \text{ where } \lambda \text{ is } G\text{'s second eigen value.} \tag{2}$$

$$\therefore\ \#\,\text{edges}(S,T) \quad \geq \quad \delta\epsilon n D - \lambda\sqrt{\delta\epsilon}n \tag{3}$$

We get a contradiction between (1) and (3) if $\frac{\delta\epsilon n}{2}D > \lambda\sqrt{\delta\epsilon}n$, or $\frac{D}{\lambda} > \frac{2}{\sqrt{\delta\epsilon}}$. There are expanders with $\frac{D}{\lambda} = \Theta(\sqrt{D})$, so the new scheme constructs corrupted lists for *atmost* δn left vertices for high enough $D = \Theta(\frac{1}{\delta\epsilon})$. The above argument also works for other $l = O(1/\epsilon)$

for a suitably high D. Note that if the (δ, l, L)-list recoverable code has rate r, then a $(1-2/l, L)$-list decodable code from this new scheme has rate $\frac{r}{D} = \Omega(r\delta\frac{1}{l})$. The rate thus degrades with smaller δ, but can be made constant for an arbitrarily small constant δ.

References

[1] Venkatesan Guruswami. *List decoding of error-correcting codes.* PhD thesis, MIT, 2001.

[2] Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. In *IEEE Symposium on Foundations of Computer Science*, pages 658–667, 2001.

[3] Venkatesan Guruswami and Piotr Indyk. Linear time encodable and list decodable codes. In *Proceedings of the 35th ACM symposium on Theory of computing*, pages 126–135, 2003.