In the first couple of lectures we looked at error-correcting codes for various settings of the rate and minimum distance parameters. For the application we had in mind – constructing two-source hitters – the ability to find such a code efficiently was also important. In this lecture we will touch upon another algorithmic aspect of codes: The efficiency of the decoding procedure. As a motivating application for efficient decoding, we introduce the problem of resilient secret sharing.

We also ask what happens when there are too many corruptions in a codeword so that unique recovery of the message becomes impossible. One sensible answer is to return not one but a list of several possible messages. This motivates the notion of list decoding whose basic properties we analyze.

1 Resilient secret sharing

In a typical secret sharing scenario, n parties want to share some secret so that no party knows what the secret is, but when the parties get together they can recover the secret. This comes up in some cryptographic applications, for example electronic voting.

In a real-world election, the physical ballot boxes are usually locked at the start of the day and the key is given under trust to the electoral commission, which has representatives from every party. At the end of the day, the representatives unlock the box and count the votes. This setup ensures both integrity (the votes are counted properly) and anonymity (it cannot be determined who voted for whom).

There are different proposals how similar guarantees can be achieved in the digital setting. Here is one such idea. When a person wants to vote, they publish an *encryption* of their vote using a public key (known to the whole community). At the end of the day, the commission applies some (publicly known) function to these encrypted votes, which gives an encryption of the tally. To determine the outcome, the commission decrypts the encrypted tally.

There are various security and implementation issues that this oversimplified description does not address, but I want to ignore these and focus on one point: Who should have access to the decryption key? If any central authority knows the whole decryption key, then it can decrypt individual votes compromising anonymity. A natural solution is to share the decryption key among the parties. Then no single party can decrypt votes, but at the end of the day all the parties can submit their shares of the key in order to reveal the tally.

But as we know all too well from real life, in every election some of the parties are bound to lose, and they have an incentive to spoil the election. Such a party may submit a bogus share of the secret key and affect the integrity of the election. How can we prevent this from happening? We want to provide some redundancy in the shares so that even if some of them are incorrect, the secret can still be recovered. This leads us to the following definition.

Definition 1. An n-party, r-secret, t-resilient secret sharing scheme over message space Σ is a pair of functions (S, R), where $S\colon \Sigma \to \Sigma^n$ is randomized and $R\colon \Sigma^n \to \Sigma$ is deterministic such that

- **Secrecy:** For every r coordinates $i_1, \ldots, i_r \in [n]$, the distribution $(S(m)_{i_1}, \ldots, S(m)_{i_r})$ is independent of the message m.
- **Resilience:** For every m and every y that differs from $S(m)$ in at most t positions, $R(y) = m$ (with probability one over the randomness of S).

The resilience condition is very reminiscent of codes, which suggests that codes can be helpful in constructing such schemes. When Σ is a finite field $\mathbb{F}$, secret sharing can be obtained from linear codes with some additional properties. Take an $[n, k, d]_{\mathbb{F}}$ linear code $C(x) = Mx$ and let $s \in \mathbb{F}^k$ be a special vector whose properties will become apparent shortly. To share the secret $m \in \mathbb{F}$, we choose a random $x \in \mathbb{F}^k$ conditioned on $\langle s, x \rangle = m$ and output the codeword $C(x)$.

As long as $d \geq 2t + 1$, the resilience property is immediate: On input y, R determines the closest codeword $C(x)$ to y and outputs $\langle s, x \rangle$. Since C is uniquely decodable up to radius $(d-1)/2 \geq t$, we have $R(y) = R(S(m)) = m$.

The secrecy property requires that for every $r \times n$ submatrix M' of M, the distribution $M'x$ conditioned on $\langle s, x \rangle = m$ should be independent of the message m. For this, it is sufficient (in fact equivalent) that $M'x$ is statistically independent from $\langle s, x \rangle$, which holds if (and only if) s is linearly independent from the row space of M'. We summarize these observations in the following lemma:

Lemma 2. *Let $C(x) = Mx$ be an $[n, k, d]_{\mathbb{F}}$ linear code and let $s \in \mathbb{F}^k$ be a vector that is linearly independent from every collection of r rows of M. Then the scheme (S, R) described above is an n-party, r-secret, $\lfloor (d-1)/2 \rfloor$-resilient secret sharing scheme.*

Here is an implementation of an n-party, $(k-1)$-secret, $\lfloor (n-k)/2 \rfloor$-resilient secret sharing scheme based on the $[n, k, n-k+1]_{\mathbb{F}}$ Reed-Solomon code. Recall that each message $x \in \mathbb{F}^k$ is associated with a degree $k-1$ polynomial p_x and the corresponding codeword is obtained by evaluating p_x over all points of some evaluation set $S \subseteq \mathbb{F}$. We will assume that $0 \notin S$ (actually any element will do). Let M be the encoding matrix of this code and $\langle s, x \rangle = p_x(0)$. To show secrecy, we need to argue that s is linearly independent from every set of $k-1$ rows of M. If not, then there must be a linear dependency of the form

$$s = b_1 M_1 + \cdots + b_{k-1} M_{k-1}$$

where $M_1, \ldots, M_{k-1}$ are some rows of M. Recalling that $\langle s, x \rangle = p_x(0)$ and $\langle M_i, x \rangle = p_x(a_i)$ for some nonzero $a_i \in \mathbb{F}$, we obtain that

$$p_x(0) = b_1 p_x(a_1) + \cdots + b_{k-1} p_x(a_{k-1})$$

for every polynomial p_x of degree at most $k-1$. However, the polynomial $p(z) = \prod_{i=1}^{k-1} (z - a_i)$ is a polynomial of degree $k-1$ that vanishes at all a_i but not at zero, contradicting our assumption of linear dependence.

There is one unsatisfying aspect of the secret sharing scheme we just described. In our description of the recovery algorithm R we asked that R finds the closest codeword $C(x)$ to y. How should R go about finding this closest codeword? The brute-force approach for finding the closest codeword takes time around $\binom{n}{t}(|\mathbb{F}| - 1)^t$, which is quite large even for small values of t. Can we do better?

2 Decoding Reed-Solomon codes

The algorithmic aspects of decoding are quite intricate; it is unreasonable to expect an efficient, optimal decoder that works for all codes as we know examples of codes for which optimal decoding is computationally hard. Even for a randomly chosen linear code, only small improvements in the effiiciency of decoding over brute-force search are known. For this reason it is more common to study decoding algorithms for specific codes like the Reed-Solomon code, and sometimes to build additional features into the code that will make decoding more tractable.

Fortunately, for the Reed-Solomon code, the algorithmic aspects of (unique) decoding are quite well understood. Here is an elegant algorithm of Berlekamp and Welch for this task.

Since the codewords of the $[n, k, n-k+1]_{\mathbb{F}}$ Reed-Solomon codes are degree $k-1$ polynomials evaluated at some set $S \subseteq \mathbb{F}$, $S = \{a_1, \ldots, a_n\}$ the problem of Reed-Solomon decoding can be viewed as polynomial reconstruction. Let's start with the case when there are no errors. To decode the codeword $(y_1, \ldots, y_n)$, we must find a polynomial p of degree $k-1$ such that $p(a_1) = y_1$ and $p(a_2) = y_2$ ... and ... $p(a_n) = y_n$. In fact, any k of the pairs (a_i, y_i) uniquely determine the polynomial p, and its coefficients can be found by solving $p(a_i) = y_i$ as a system of linear equations in the coefficients of p.

Here is an equivalent way of describing the same thing. Notice the bivariate polynomial $p(x) - y$ vanishes on all the pairs (a_i, y_i). Suppose we found a nonzero polynomial $q(x, y)$ of the form $q(x, y) = p'(x) - y$, where p' has degree $k-1$ and $q(a_i, y_i) = 0$ for k pairs (a_i, y_i). Then p' must equal p: Saying that $q(a_i, y_i) = 0$ is the same as saying that $p'(a_i) = p(a_i)$, and if this happens at k points then $p' - p$ is a polynomial of degree $k-1$ with k zeros, so $p' = p$.

Now let us bring in the errors: Suppose that $p(a_i) = y_i$ for all but at most $t \leq (n-k)/2$ of the pairs (a_i, y_i). Let E denote the indices of these erroneous pairs. Then the polynomial

$$(p(x) - y) \prod_{i \in E} (x - a_i) = p(x) \prod_{i \in E} (x - a_i) - y \prod_{i \in E} (x - a_i)$$

still vanishes on all the pairs (a_i, y_i). This polynomial has terms x^i with $i \leq k + t - 1$ and terms yx^i with $i \leq t$. So let's see what happens if we manage to find a nonzero polynomial $q(x, y) = n'(x) - ye'(x)$, where n' and e' have degree at most $k + t - 1$ and t, respectively. This polynomial q has $k + 2t + 1$ coefficients. If $k + 2t = n$, we can certainly find such a q by solving $q(a_i, y_i) = 0, 1 \leq i \leq n$ as a system of linear constraints in the coefficients of q. (If n is larger we can ignore some of the data.) By analogy with the errorless case, we would like to say that $e'(x)$ has to equal $\prod_{i \in E}(x - a_i)$ and $n'(x)$ must equal $p(x) \prod_{i \in E}(x - a_i)$. This is not quite true but very close:

Lemma 3. *Let $(a_1, y_1), \ldots, (a_n, y_n)$ be distinct pairs, p a degree $k-1$ polynomial, and suppose that $p(a_i) = y_i$ for all i except possibly those $i \in E$. If the polynomial $q(x, y) = n'(x) - ye'(x)$ (where n' and e' have degree at most $k + t - 1$ and t, respectively) vanishes on all the pairs (a_i, y_i), then $n'(x) = p(x)e'(x)$, provided $k + 2t \leq n$.*

Proof. Since $q(a_i, y_i) = 0$ for all i outside E, and $y_i = p(a_i)$ for all i outside E, it follows that $n'(a_i) = p(a_i)e'(a_i)$ for all $n - t$ values of i outside E. Since $n - t \geq k + t$, the polynomials n' and pe' agree at $k + t$ points. They both have degree at most $k + t - 1$, so they must be identical. □

This shows the correctness of the following algorithm for decoding Reed-Solomon codes up to $\lfloor (n-k)/2 \rfloor$ errors. On input $(a_1, y_1), \ldots, (a_n, y_n)$ all distinct, find a polynomial $q(x, y) = n'(x) - ye'(x)$ that vanishes on all (a_i, y_i), where n' has degree at most $k + t - 1$ and e' has degree at most t, and output the polynomial $n'(x)/e'(x)$.

3 List decoding

Recall that we obtained a pretty good upper bound on the the message length of codes at small distance via the volume bound: In an $[n, k, d]$ code, all balls of radius $\lfloor (d-1)/2 \rfloor$ centered at codewords must be disjoint, so $2^k V(\lfloor (d-1)/2 \rfloor) \leq 2^n$ or $k \leq n - \log V_n(\lfloor d-1 \rfloor/2)$, where $V_n(r)$ is the volume of a hamming ball of radius r in $\{0,1\}^n$. At large distances $d = (1-\varepsilon)/2$, we have $V_n(\lfloor d-1 \rfloor/2) \leq V_n(n/4) \leq 2^{nH(1/4)}$, so this argument cannot give a better bound than $k \leq (1 - H(1/4))n \approx 0.18n$.

One intuition why the volume bound doesn't do so well at large distances is that when we pack large hamming balls into $\{0,1\}^n$, it is unreasonable to expect that every point of $\{0,1\}^n$ will be covered, and the above calculation suggests that an overwhelming fraction of the space won't be covered at all. A natural thing to try is to use larger balls, but then we have to account for the possibility that these balls will intersect and some of the points in $\{0,1\}^n$ will be counted multiple times. How can we manage these overcounts?

Let's look at a code of distance $(1-\varepsilon)n/2$ where ε is small. We center a hamming ball of radius ρn around every codeword. When $\rho \leq (1-\varepsilon)/4$, every point in $\{0,1\}^n$ is covered at most once, so let's make ρ a little bit larger, say $\rho = 1/4$. What is the largest number of such balls that can cover a point $v \in \{0,1\}^n$? An equivalent way to ask this question is: How many codewords can a ball of radius $n/4$ centered at v contain?

Without loss of generality, we may assume $v = 0^n$ (we can shift each codeword by v to make this happen; this doesn't change the distance). Then every codeword inside the ball has Hamming weight at most $n/4$. If there are at most four codewords inside the ball, then all these codewords could have disjoint support (i.e. all the ones occur in different positions) so the distance between any pair could be as large as $n/2$. But as soon as there are five codewords inside the ball, it is not difficult to see that there must be a pair of codewords whose supports intersect on at least $n/60$ positions, and so the distance between these codewords is at most $n/4 + n/4 - n/60 = 29n/60$, so ε cannot be too small.

So we conclude that for ε sufficiently small, any code of distance $(1-\varepsilon)n/2$ cannot have more than four codewords in any ball of radius $n/4$. In other words, suppose you receive a corrupted codeword with $n/4$ errors. Although unique decoding is impossible at this error rate, we can still come up with a list of four candidate codewords one of which is guaranteed to be the correct one.

The argument we gave is reminiscent of our intuition that led to the Plotkin bound. Recall that the actual Plotkin bound was derived by adopting a geometic viewpoint. This line of thinking will be helpful here too. It leads to the following theorem.

Theorem 4 (Johnson bound)**.** *Let $c_1, \ldots, c_\ell \in \{0,1\}^n$ be such that each c_i has Hamming weight at most $(1-\alpha)n/2$ and such that the Hamming distance between every pair (c_i, c_j) is at least $(1-\varepsilon)n/2$. Then $\ell \leq 1/(\alpha^2/4 - \varepsilon)$.*

For $\alpha = \sqrt{8\varepsilon}$, we get that in a code of distance $(1-\varepsilon)n/2$, a ball of radius $(1-\sqrt{8\varepsilon})n/2$ can contain at most $1/\varepsilon$ codewords. Since the distance is an integer, $1/\varepsilon \le n$. (It is possible to improve some constants here.)

Proof. As in the proof of the Plotkin bound, we embed $c_1, \ldots, c_\ell$ in $\{-1,1\}^n$ by replacing 0 with 1 and 1 with -1. Then the assumptions translate into the geometric conditions

$$\langle c_i, \mathbf{1}\rangle \ge \alpha n \text{ for all } i \quad \text{and} \quad \langle c_i, c_j\rangle \le \varepsilon n \text{ for all } i \ne j$$

where $\mathbf{1}$ is the all ones vector. (In geometric terms, this says that the vectors c_i must be almost perpendicular to one another, but they must also lie inside a "cone" of angle α around the vector $\mathbf{1}$.) Then for every parameter t,

$$\begin{aligned} 0 \le \Big\|\sum_{i=1}^{\ell} c_i - t\mathbf{1}\Big\|^2 &\le \sum_{i=1}^{\ell} \|c_i\|^2 + t^2\|\mathbf{1}\|^2 + \sum_{i\ne j}\langle c_i, c_j\rangle - \sum_{i=1}^{\ell} t\langle c_i, \mathbf{1}\rangle \\ &\le \ell n + t^2 n + \ell^2 \varepsilon n - t\ell\alpha n \\ &= \big(t^2 - \alpha\ell\cdot t + (\ell + \ell^2\varepsilon)\big)n. \end{aligned}$$

The right hand side is minimized at $t = \alpha\ell/2$, where it takes value $(-(\alpha\ell/2)^2 + (\ell+\ell^2\varepsilon))n$. So we must have $(\alpha\ell/2)^2 \le \ell + \ell^2\varepsilon$, from where $\ell \le 1/(\alpha^2/4 - \varepsilon)$. □

Now let's go back to the covering argument. If we center a ball of radius $\rho n = (1-\alpha)n/2$ around every codeword, each point in 2^n is covered at most ℓ times, so $2^k V_n((1-\alpha)n/2) \le \ell 2^n$. By the Johnson bound, when $\alpha = \sqrt{8\varepsilon}$, $\ell \le 1/\varepsilon \le n/2$. So we must have

$$2^{k-n}\cdot V_n((1-\alpha)n/2) \le n$$

Using Stirling's apporximation of the factorial, we have that $V_n(\rho n) \ge \binom{n}{\rho n} = 2^{nH(\rho)-o(n)}$. Using the estimate

$$H(\rho) = H((1-\alpha)/2) = 1 - O(\alpha^2) = 1 - O(\varepsilon)$$

we conclude that

$$2^{k-n}\cdot 2^{n(1-O(\varepsilon))-o(n)} \le n$$

from where it follows that in an $[n, k, (1-\varepsilon)n/2]$ code we must have $k = O(\varepsilon n)$. (A sharper version of this argument is known as the Elias-Bassalygo bound.)

4 Sudan's list-decoding algorithm (optional)

Notice how crucial was the choice of degree of the polynomials n' and e' in the Berlekamp-Welch algorithm: The degrees had to be large enough so there is enough degrees of freedom in the coefficients of q to satisfy all the constraints $q(a_i, y_i) = 0$, but small enough to make sure that n' and e' do not have too many roots (and so they must be the same).

Since the algorithm we described works all the way up to the unique decoding radius, this is the best we can do. To go beyond it, we must somehow make the degrees of n' and e' even smaller while still preserving the number of coefficients of q. One thing that comes to mind is to allow for larger degrees of y, so we take $q(x,y) = q_0(x) + q_1(x)y + \cdots + q_{d-1}(x)y^{d-1}$. In the unique decoding setting,

we were able to read off the codeword from q_0 and q_1; since we are now in the list-decoding regime, maybe we can hope that the other polynomials q_i correspond in some way to various codewords in the list. This sounds somewhat crazy and is not quite right, but it essentially works!

As before, we are given a list of pairs $(a_1, y_1), \ldots, (a_n, y_n)$ so that $p(a_i) = y_i$ for some polynomial p of degree at most $k-1$ for all but at most t of these pairs and we want to reconstruct the polynomial p. The only difference is that t is larger than before so there could be several polynomials that agree with $n-t$ of the pairs. We find a nonzero polynomial q of degree $d_x - 1$ in x and $d_y - 1$ in y that vanishes on all these points; this is possible as long as $d_x d_y > n$. Can we somehow "read off" p by looking at q?

We know only one thing, namely that $q(x, p(x))$ takes value zero at the $n - t$ points (a_i, y_i) where $p(a_i) = y_i$. But now $q(x, p(x))$ is a *univariate* polynomial of degree at most $(d_y - 1) + (k-1)(d_x - 1)$; if this is is smaller than $n - t$, then $q(x, p(x))$ must be identically zero.

Claim 5. *If $q(x, p(x))$ is identically zero, then $y - p(x)$ divides $q(x, y)$.*

Proof. Let $q(x, y) = \sum q_i(x) y^i$. Then

$$q(x,y) = q(x,y) - q(x,p(x)) = \sum q_i(x)(y^i - p(x)^i) = (y - p(x)) \sum q_i(x) \sum_{j=0}^{i} y^j p(x)^{i-j}. \qquad \square$$

On the other hand, multivariate polynomials have a unique factorization which can be found efficiently, so p can be read off from the factorization of q. Also, since each candidate codeword p gives rise to the term $y - p(x)$ in the factorization of q, there can be no more than $d_y - 1$ such terms.

It remains to calculate the parameters. We are free to choose d_x and d_y as long as $d_x d_y > n$ and $(d_y - 1) + (k-1)(d_x - 1) < n - t$. By optimizing roughly we find out that both constraints can be satisfied as long as $t < n - 2\sqrt{(k-1)n}$ (by setting $d_x = \sqrt{n/(k-1)} + 1$ and $d_y = \sqrt{(k-1)n} + 1$). When k is small compared to n, the decoding radius almost doubles, although we have to allow a list of size $d_y \approx \sqrt{kn}$.

This algorithm was discovered by Madhu Sudan. Later Guruswami and Sudan managed to remove the factor of 2 in front of the square root, which makes a difference at small distances.