Efficient decoding of error-correcting codes

Satyadev Nandakumar

September 11, 2026

Contents

1 Motivation

In order to use error-correcting codes practically, we need *efficient decoders*. Further, for our purpose of derandomizing BPP, we will need decoders that can recover from a large fraction of errors. So we will start with efficient decoders, and then proceed to more powerful decoders like local decoders, list decoders and local list decoders.

2 Efficient (non-local) decoder for Reed-Solomon code

Any polynomial of degree d can be uniquely reconstructed from any set of $d+1$ values using a procedure such as *Lagrange Interpolation*. What we have to do here is a *robust* version of this procedure which can uniquely recover the polynomial in presence of errors.

Theorem 2.1. *There is a polynomial-time algorithm that does the following: Given a list* (a_1, b_1), $\ldots$, (a_m, b_m) *of pairs of elements of a finite field* $\mathbb{F}$ *such that there is a degree* d *polynomial* $G : \mathbb{F} \to \mathbb{F}$ *satisfying* $G(a_i) = b_i$ *for at least* t *pairs, if* $t > \frac{m}{2} + \frac{d}{2}$*, the algorithm determines* G *uniquely.*

The recovery procedure is the famous *Berlekamp-Welch algorithm*, which may appear a bit mysterious at first. We transmit m pairs corresponding to the polynomial. At most $m - t = \frac{m-d}{2}$ pairs are wrong. If we knew the exact positions at which the polynomial is wrong, then we can use the other locations to determine the polynomial uniquely.

Thus, if we have an *error-locator polynomial* E whose roots (zero values) are exactly the erroneous values, we can do the following. Take a higher degree polynomial C, and write $C(x) = P(x)E(x)$.

For every pair that has been correctly received, *i.e.* $P(a_i) = b_i$, we have $C(a_i) = P(a_i)E(a_i) = b_iE(a_i)$. For every error pair, we have $P(a_i) \neq b_i$, but $E(a_i) = 0$. So in either case, the equation $C(x) = P(x)E(x)$ holds.

Proof. The proof uses the *Berlekamp-Welch procedure*. The following proof works for any values satisfying $t > \frac{m+d}{2}$. However, we will assume, for simplicity of notation, $m = 4d$ and $t = 3d$ - that is, the length of the message is 4 times the degree of the univariate polynomial, and the number of correctly received values is at least 3 times the degree of the polynomial, ensuring that at least 3/4 fraction of the values are correct.

1. Find a degree-$2d$ polynomial $C(x)$ and a degree-d polynomial $E(x)$ such that for all $i \in \{1, 2, \ldots, m\}$

$$C(a_i) = b_iE(a_i). \tag{1}$$

   There are $4d$ linear equations, and the number of coefficients of C are $2d+1$ and the number of coefficients of E to be determined are $d+1$. Since $4d > 3d+2$, we can determine the coefficients by solving the system of linear equations.

2. Divide C by E to get a polynomial P such that $C(x) = P(x)E(x)$ for all x.

Correctness. We know that $C(x) = G(x)E(x)$ for at least $3d$ values, where G is the original polynomial that was sent. Thus, $C(x) - G(x)E(x)$ is a degree $2d$ polynomial that has at least $3d$ roots. We know that a non-zero degree $2d$ polynomial has at most $2d$ roots. This means that $C(x) - G(x)E(x)$ is identically zero. That is, $G(x)$ indeed equals C/E.

Time Efficiency The main step in the above algorithm is solving the system of linear equations, which can be done using Gaussian elimination. Thus the above algorithm is polynomial time in d. □

3 Local decoding

Let $f : \{0,1\}^n \to \{0,1\}$ be a boolean function. This can be represented by its truth table, which is a bitstring of length $N = 2^n$.

For hardness amplification, we need to show that, given a circuit which can solve a large fraction of inputs to f (say, 0.9), we can produce another slightly larger circuit which can solve *all* the inputs to f. For this, we need a *local decoder*.

A *local decoder* is a decoding algorithm that, given ranom access to a corrupted code y' which is close to an actual codeword $E(x)$, can recover any arbitrary bit of x.

Since we want to efficiently compute f, the decoding must take only poly(n) time. (This is a very tough requirement - the input to the decoder has length $N = 2^n$, so the local decoder must run in time polynomial in the logarithm of its input length.)

Definition 3.1. Let $E : \{0,1\}^n \to \{0,1\}^m$ be an error-correcting code. Let $\rho \in [0,1]$ and $q \in \mathbb{N}$ be numbers. A *local decoder* for E *handling* ρ *errors* is an algorithm D such that given random access to a string y such that $\Delta(y, E(x)) < \rho$ for some x, and an index $j \in \mathbb{N}$, runs for polylog(m) time and outputs $x[j]$ with probability at least $\frac{2}{3}$.

3.1 Local decoder for Walsh-Hadamard codes

The following local decoder for Walsh-Hadamard codes works when the fraction of errors, ρ is less than $1/4$.

Theorem 3.2. *For every $\rho < 1/4$ the Walsh-Hadamard code has a local decoder handling ρ errors.*

Proof. Consider the following algorithm.

- Assumption: $x \in \{0,1\}^n$ is the original string. The sent codeword was WH(x). The received codeword is g.
- Input: $j \in \{0, \ldots, n-1\}$, random access to a function $f : \{0,1\}^n \to \{0,1\}$ such that
$$Pr_y[g(y) \neq x \odot y] \leq \rho.$$
- Output a candidate for $x[j]$.
- Let e^j be the unit bitvector of length n with 1 at the j^{th} position, and 0 elsewhere.
- Choose a random string $y \in \{0,1\}^n$.
- If $f(y) == f(y + e^j)$, then output 0, else output 1.

Analysis. With probability at most ρ, we have that $f(y) \neq x \odot y$ and with probability at most ρ, $f(y + e^j) \neq x \odot (y + e^j)$. Hence, with probability at least $1 - 2\rho$, we have $f(y) = x \odot y$ and $f(y + e^j) = x \odot (y + e^j)$.

Note also that if $x[j] = 0$ if and only if $x \odot y == x \odot (y + e^j)$. Thus the algorithm correctly outputs the bit $x[j]$ as 0 if $f(y) == f(y + e^j)$ and 1 if $f(y) == f(y + e^j)$, with probability at least $1 - 2\rho$. □

The above algorithm is a local decoder, since we only look at two randomly chosen locations in the received codeword to decide our output. We do not have to look at the entire received codeword.

3.2 Local decoder for Reed-Muller code

Theorem 3.3. *For every field $\mathbb{F}$, ℓ-variable polynomial P of total degree d, there is a poly$(|\mathbb{F}|, \ell, d)$-time local decoder for the Reed-Muller code, handling $(1 - \frac{d}{6|\mathbb{F}|})$ fraction errors.*

Note that a large field and a polynomial of small (total) degree, we can handle a greater fraction of errors.

We need to error-correct and decode the multivariate polynomial P at x. The overall idea is to reduce the Reed-Muller local decoding problem to the Reed-Solomon decoding problem by taking a "random slice" of the multivariate polynomial passing through x.

Proof. We represent the polynomial not as a list of its coefficients, but as a list of received values. This is what makes sense in our setting, because the receiver receives not a list of coefficients, but a list of values of the polynomial. Further, as stated, assume that at most $\frac{d}{6|\mathbb{F}|}$ fraction of the received values is wrong.

- **Input:** $x \in \mathbb{F}^\ell$, random access to a function f such that

  $$Pr_{x \sim \mathbb{F}^\ell}\left[f(x) \neq P(x)\right] < \rho,$$

  where P is the (actual) ℓ-variate polynomial of total degree d.
- **Output:** a candidate for $P(x)$.
- Choose a random $z \in \mathbb{F}^\ell$.
- Let $L_x = \{x + tz \mid t \in \mathbb{F}\}$ be a random line passing through x, determined by z. (Note that t is a field element, not an element in $\mathbb{F}^\ell$)
- Obtain the set $\{(t, f(x + tz)) \mid t \in \mathbb{F}\}$ by querying f.
- Run the Reed-Solomon decoding algorithm to obtain the univariate polynomial $Q(t)$
- Output $Q(0)$. (When $t = 0$, we have $x + tz = x$, so the corresponding point in the set is $(0, f(x))$.)

Analyis: Recall from the discussion on the Reed-Solomon code that the fractional Hamming distance of the Reed-Solomon code is $\left(1 - \frac{d}{m}\right)$, where m is the number of values transmitted. Since $m \leq |\mathbb{F}|$, an *upper bound* on the fractional Hamming distance of the code is $\left(1 - \frac{d}{|\mathbb{F}|}\right)$.

Let ρ be the probability with which a value in the Reed-Muller (multivariate code) gets erroneous. For a random $z \neq 0 \in \mathbb{F}$, for any $x \in \mathcal{F}$, the values

$$S_{x,z} = \{x + tz | t \in \mathcal{F}\}$$

are independent. Hence the expected number of errors in this set of values is $\rho|\mathbb{F}|$.

By Markov inequality, the probability that the number of erroneous valus in $S_{x,z}$ is more than $3\rho|\mathbb{F}|$ is at most $1/3$.

Thus we get

$$3\rho|\mathbb{F}| \leq \frac{\left(1 - \frac{d}{|\mathbb{F}|}\right)}{2}|\mathbb{F}|,$$

yielding that $\rho < \left(1 - \frac{d}{|\mathbb{F}|}\right)/6$, which is the condition under which error-correction is assured.

Thus, if $\rho < \left(1 - \frac{d}{|\mathbb{F}|}\right)/6$, then for every $x \in \mathbb{F}$, the above procedure will get the correct value with probability at least $2/3$. $\square$