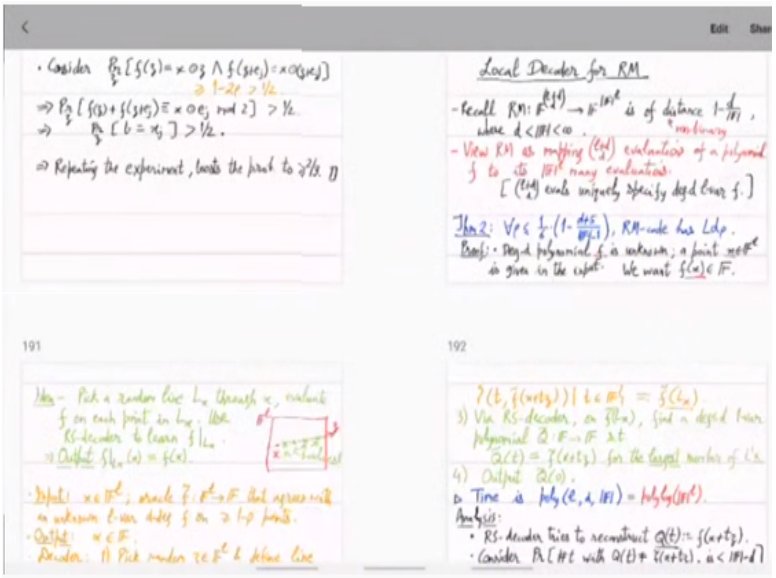
Randomized Methods in Complexity
Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology – Kanpur

Lecture – 25
List Decoding

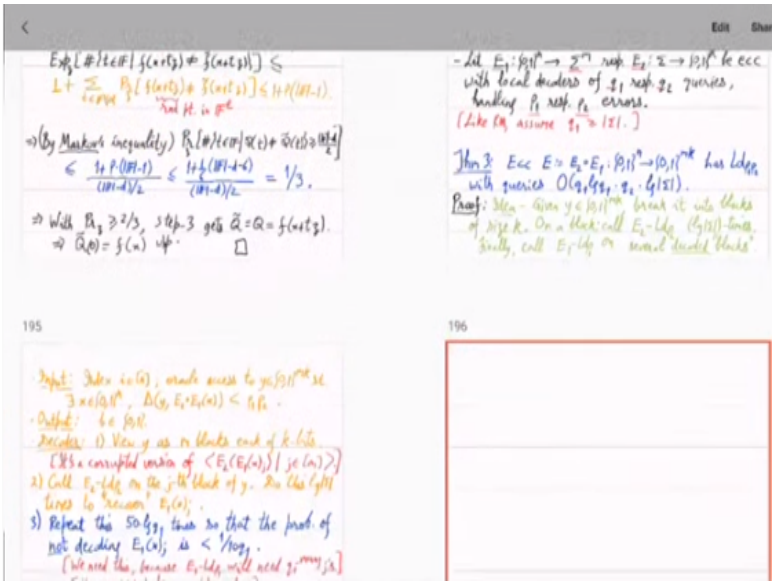
Local decoding and we did it for Walsh Hadamard by just doing 2 queries we can recover a bit of x.

(Refer Slide Time: 00:29)



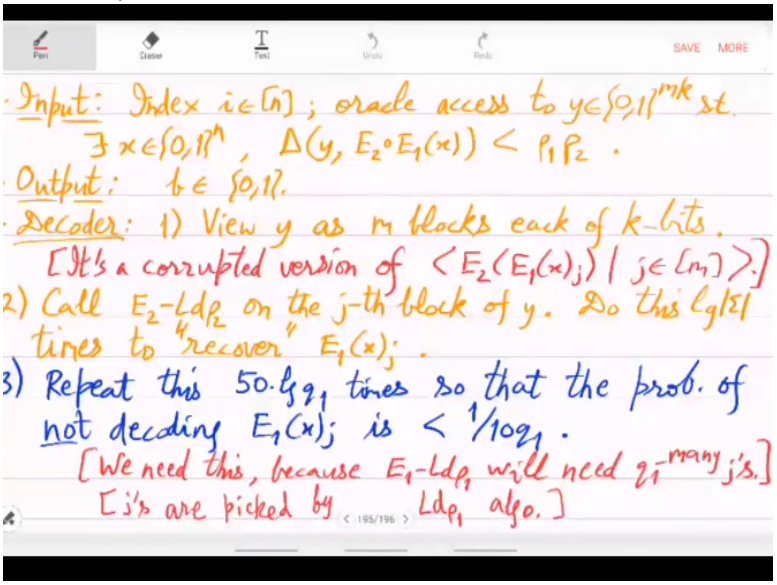
Then we did local decoding for Reed Muller, which will also imply local decoding for Reed Solomon. So, here this was more complicated because it is a multivariate polynomial over finite fields. To compute its value at a given point, we actually draw a random line through this point x and then on the line we do Reed Solomon decoding number of queries needed was the size of the field.

(Refer Slide Time: 00:58)



Now, once we have local decoding for Walsh Hadamard, Read Muller we can also do it for concatenation by an idea that you have seen before. We just wanted to analyse this carefully, so ρ_1 is the local decoding parameter for E_1 and ρ_2 for E_2. And what the code is doing is it first applies E_1 to get sigma alphabet and then every letter in the alphabet it will convert using E_2 to binary, so there are x into m alphabets. You can think of the long string y which is accessible through an apple you can think of it as E_1 x j^{th} element.

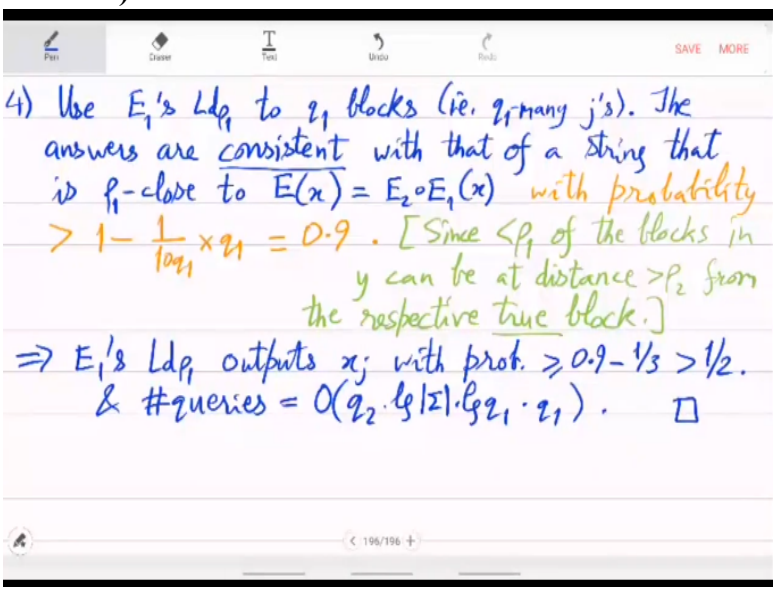
(Refer Slide Time: 01:42)



j^{th} co-coordinator of $E_1(x)$ on which E_2 was applied. Based on the j's that E_1 local decoder will require, you go to those blocks $E_1(x_j)$ with E_2 applied you go to those j^{th} blocks and first you learn E_1 (x_j)/ E_2 local decoders. That is what we did last time. Step 2 is E_2 local decoder with parameter ρ_2 this was applied on the j^{th} block and this was applied again and again.

It was applied actually log sigma many times to recover $E_1(x_j)$ the whole symbol. So, this will work if in this block the error is less than ρ_1 fraction and since E_1 will require q_1 many j's. You have to access those many blocks, so the error probability also should be made very small. We actually repeat this process around log q_1 many times and that will give us $E_1(x_j)$ with a very good probability. Let us do that part, so step 4 will be about E_1 local decoder now.

(Refer Slide Time: 03:02)



Use E_1's local decoder to query q_1 blocks that is q_1 many j's and in every j we have used E_2's $Ld\rho_2$ as before in step 2 and step 3. After applying E_2's local decoder on those blocks, we are now moving to E_1. The answers are consistent with that of a string that is ρ_1 close to E of x which is remember E_2 E_1 x. This is with probability greater than $1 - (1/10q_1) \times q_1$.

For a single block the error is 1 over $10q_1$; for q_1 blocks it can be at most 1 / 10, so the probability is 90% or more. With this much probability E_1 will be able to I mean the answers will then be correct for how many of these blocks? So, that we can write here since, less than ρ_1 of the blocks in y can be a distance greater than ρ_2 from the respective true block remember that the overall cumulative error is ρ_1, ρ_2 or less.

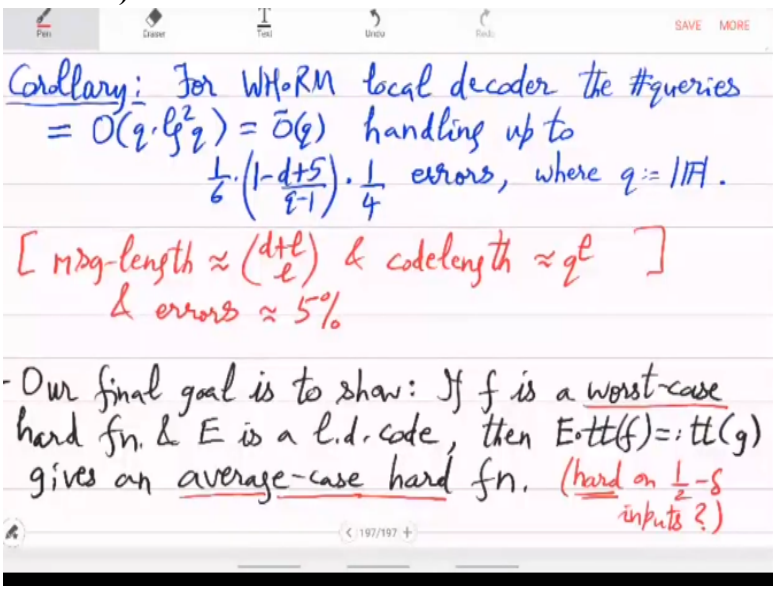
So, which means that it is few blocks only which will be badly corrupted. By bad I mean more than ρ_2 corrupted so, only less than ρ_1 of the blocks are corrupted to an extent more than ρ_2 in each such block. In other words if you look at the level of blocks, the error is only

ρ_1 or less. Whatever he has given in these q_1 block out of them the error is only ρ_1 or less. So, hence when you will apply E_1's local decoder $Ld\rho_1$ it will work fine and give you a bit of x.

This means that E_1's local decoder outputs x_j with probability already it was 0.9 came from what E_2 local decoder did and after that E_1's local decoder is doing there may be error of 1 / 3 that is by $Ld\rho_1$ property local decoders property which is more than half. With probability more than half you will get x_j and number of queries so, we in the initial steps it required log q_1 times q_1 times log sigma and there is also q_2.

It is required q_2 log sigma and this was repeated log q_1 time just for probability boosting and for E_1 sake number of blocks will be q_1, so that is another q_1. This is the number of queries and the success probability for x_j good. This finishes the local decoder for concatenated codes. Once you have local decoding for the individual components E_1, E_2 and this was actually quite straightforward but when you look at the parameters they look complicated.

(Refer Slide Time: 09:46)



And what this means for us, for Walsh Hadamard, Reed Muller local decoder, so the number of queries will be sigma is the size of the field and which is also equal to the queries you make for Reed Muller, so which is q_1, so you get q_1 times log q_1 whole square, q_2 is for Walsh Hadamard which was only 2 so, that is just constant. You get q $\log^2$ q which is basically linear in q just slightly more than linear in q that many queries handling up to how many errors what is the ρ parameter, so that is $\rho_1 \times \rho_2$.

Fr Reed Muller, you will get $1/6(1 - (d + 5)/(q - 1)) \times 1/4$ which is around 4% errors where q is the size of the field and also note that the message length it is around what is the stretch of Reed Muller. So, that is d degree l variate polynomials, so you get $\frac{d+1}{l}$ and code length is regular see x as a polynomial l variate d degree.

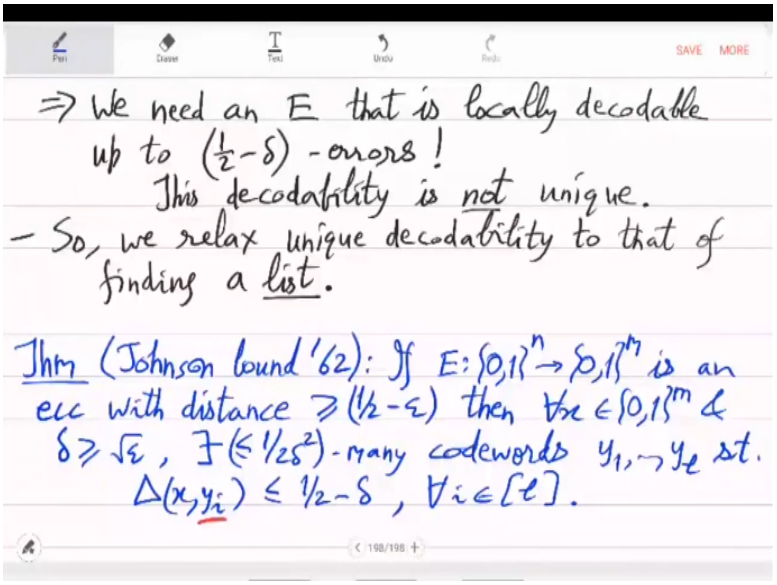
So, that $\frac{d+1}{l}$ many coefficients or many values that defines a polynomial multivariate polynomial and then Reed Muller will evaluate it on every point in the field. So, that is q^l that is the code length. Now observe in terms of the code length in terms of code length q^l, that is the length of wide it is very long and this algorithm local decoder is making only q queries which is much less than q^l.

And it will give your favourite value amongst these $\frac{d+1}{l}$ values with good probability and even errors can be quite a lot it is 4% of why we corrupted and errors around 5%. This is a very important interesting algorithm. But this would not be enough remember our final goal is hardness amplification from worst case to average case. For that 5% error is not enough, we actually should be able to handle much more, closer to 50% actually something like 49% errors we should be able to handle.

So that is what we have to do now, our final goal is to show if f is a worst case hard function and E is a locally decodable code. Then, on the truth table of f I can look at that long string 2^n, n bit input the values that f takes on all possible and bit inputs is 2^n value. So that is called the truth table of f on the truth table you apply the locally decodable code E and you will get a slightly bigger string and that defines a function f' truth table of g is should gives a average case hard function that is what we want to show.

Average case hard means that it should be half minus smaller than half many inputs. So, hard on close to $(1/2 - \Delta)$ inputs. This actually if you think about this carefully this will require a local decoder for E which can handle errors close to 50% which is asking for too much.

(Refer Slide Time: 16:46)

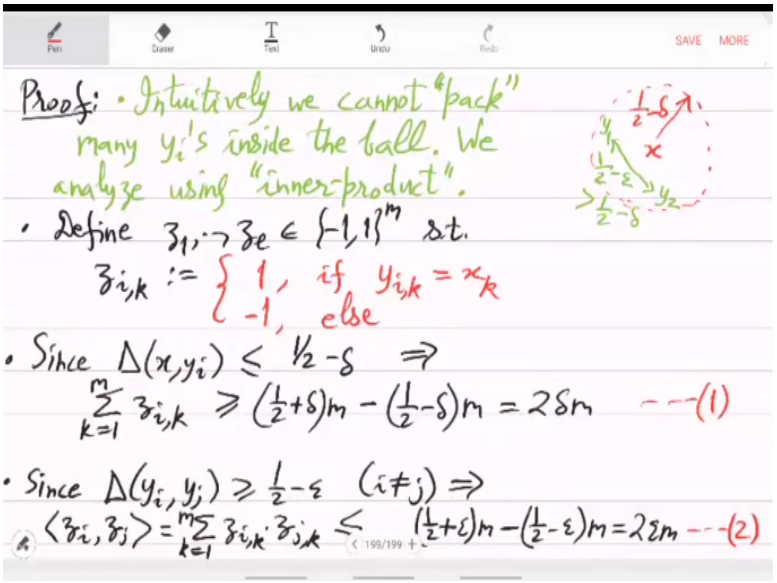


Implies that we need and E that is locally decodable up to $(1/2 - \Delta)$ errors, so this type of decode ability cannot be unique meaning that there will be many x's for a y so corrupted the x's options for messages will be many it will not just be one but they will be super constant many x's. Now, the question is how many and the next question is whether you can computationally list them. We relax unique decodability to data finding a list, so this is called local list decoding.

But whether it is possible or not that will depend on how long the list is, so let us first show that and that is called Johnson's bound. If you have a map that is an error correcting code with distance at least $(1/2 - \varepsilon)$ then for x's and $\delta \geq \sqrt{\varepsilon}$. There exists at most $1/2\delta^2$ many code words $y_1,...,y_l$ such that they are close to x. The numbers of y's that are close to x that is what we are interested in, how many y's are close to x?

So, think of x as a corrupted code word and the question is how many code words are close to x that number is in terms of δ it is inverse for polynomial. So, it is $1/2\delta^2$ in terms of ε you can think of this as $1/\varepsilon$. For distance $(1/2 - \varepsilon)$ the list is $1/\varepsilon$ and that is a very good bound, so for all i 1 to l. So, you can think of this as the following picture.

(Refer Slide Time: 21:45)



You look at the ball around x $(1/2 - \delta)$ look at this radius ball around x which means collect all the strings which are fractional hamming distance less than half exactly $(1/2 - \delta)$ or less. And in this world how many code words can you trap. So, say y_1 is a codeword here, y_2 is a codeword here. Now, remember that code words themselves are far apart in the space because of the distance of E, so, the distance between them is already $(1/2 - \varepsilon)$.

So, since y_1 and y_2 are far apart, the intuition says that you cannot pack many y_i's inside this ball because this $(1/2 - \varepsilon)$ is more than $(1/2 - \delta)$, it is more than the radius. So, you cannot have too many of them, but how many what is the bound? How do we upper bound is that is the thing, so let us look at the proof. So, intuitively we cannot pack many y_i's inside the ball and we will analyze this in terms of distances and inner products. We analyze using inner product.

Let us define inner product so, that it captures this hamming distance well, so define these strings z_1 to z_l in $\{-1,1\}^m$. Let us define these strings z_i's such that what is the k^{th} location of the z_i this is 1 if will associating the z_i with y_i. So, 1 if $y_{i,k} = x_k$, so, if the k^{th} bit of y_i and x is the same that is indicated by $z_{i\,k}$ otherwise negative and now since distance of x and y_i is smaller than $(1/2 - \delta)$. So, what you get is over all the case all the locations of x and y_i.

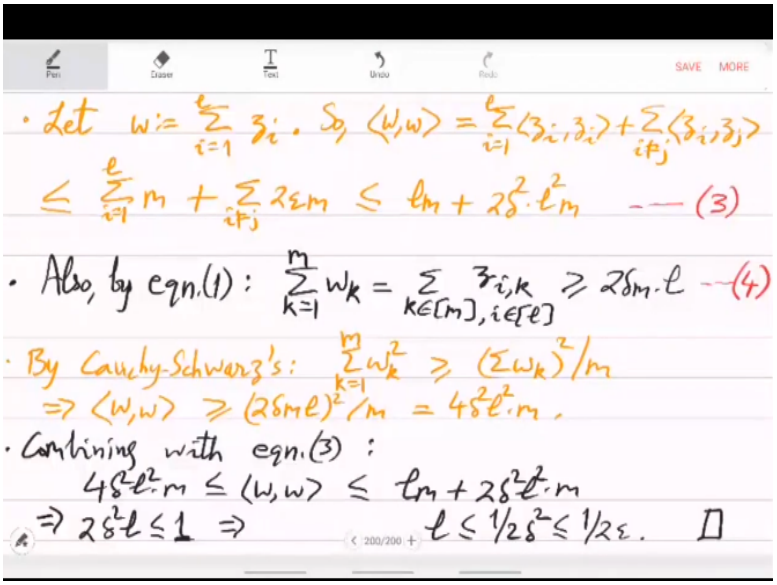
That we see distance being small means that x and y_i, overlap in many locations they are similar, so which means that $z_{i\,k}$ will be a lot more positive than negative. How many times positive $(1/2 + \delta)m$ times positive otherwise negative which is $(1/2 - \delta)m$. That is $2\delta m$

that is 1 equation on the other hand, the opposite picture is that y_i and y_j have to be far apart by the notion of distance of E, so that gives you something else. So, since y_i y_j are away what you will get is the inner product of z_i, z_j.

Now, since y_i y_j or differ a lot so, this overlap in fewer places so, which means the inner product is expected to be small, how small? Let us go over the coordinates and $z_{i\,k}$ $z_{j\,k}$ they are of the same sign if the match x_k is their opposite. So, in how many places will the match x_k well that will be fewer than $(1/2 + \varepsilon)$ by the above inequality and the negatives will be more than $(1/2 - \varepsilon)$t which is $2\varepsilon\, m$.

Let us number these 2 conditions that we have as 1 and 2. The first condition is kind of saying that z_i is a long vector, second condition is saying that z_i z_j are the inner product is small which means they are almost orthogonal. So, these are long vectors and they are almost orthogonal these are the 2 conditions geometrically that we have written. And how do we combine them to get something useful.

(Refer Slide Time: 29:20)



Let us define w sum of all the z_i, and how long is w, length of the w, it will involve both z_i inner product with itself and z_i inner product with others z_j's. Now z_i with itself fill coordinate by coordinate if you see that will be all plus 1 values, so that is m and z_i z_j inner product for different i j that will come out to be $2\varepsilon\, m$, which is less than equal to, you get $lm + \varepsilon$ is smaller than δ^2.

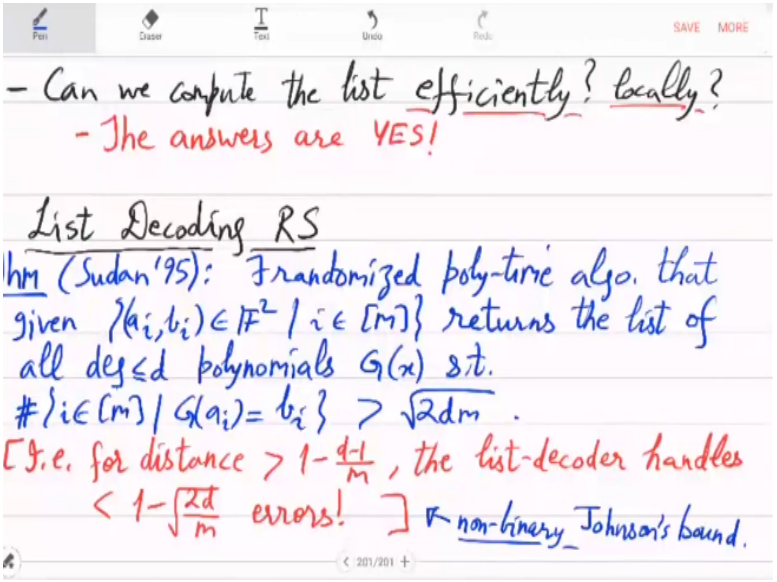
You get $2\delta^2$ and number of i j appears is definitely less than this l^2, so you get this that is your equation 3. Also by equation 1, you can get more information about w inner product with itself because equation 1 is saying that this $z_{i\,k}$ they are kind of large, so that should mean that inner product of w with itself should also be large. Let us think of it that way. So, you will get Σw_k which is all k and all l i's, so $z_{i\,k}$ sum over all the $z_{i\,k}$'s that will be by equation 1 at least $2\delta m \cdot l$.

Think of it as equation 4 and now equation 3 and 4 are somehow opposites, so we will combine them and get the answer. By Cauchy Schwarz you can see that essentially average of squares is at least squares of the average that inequality, so which means that inner product of w with itself is at least Σw_k which is by equation 4 $(2\delta ml)^2/m = 4\delta^2 l^2 m$. So, inner product of w itself is large this is what you have deduced and previously deduced that is small, so that will give you the result.

Combining with equation 3 gives you that $4\delta^2 l^2 m$ is less than inner product w itself which is less than equal to $lm + 2\delta^2 l^2 \cdot m$ which implies after some simplification you will get $2\delta^2 l \leq 1$. δ is taken to be positive as well. So, because of the nonzero δ you can divide. And since δ was more than $\sqrt{\varepsilon\delta}$ you will get $1/2\varepsilon$ also.

So, basically for error correcting codes of distance $1/2 - \varepsilon$ the list is no longer than $1/2\varepsilon$. That is a beautiful result, list is small basically. Next question would be whether even we can find small list and the error is very close to half.

(Refer Slide Time: 37:14)



Can we compute the list efficiently and locally? Locally means that by minimizing the queries to the corrupted code word, so we will actually do both. Let us first decode Reed Solomon that is a fundamental decoder, this is a famous theorem by Madhu Sudan. He showed that there exists a randomized poly time algorithm that given a corrupted Reed Solomon code word, which means that you are given evaluations some of them are wrong evaluations of a polynomial.

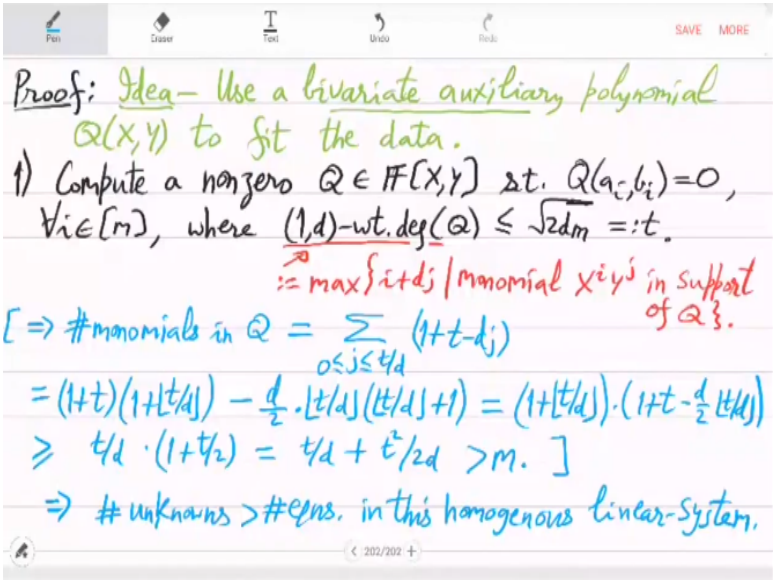
That given $\{a_i , b_i\}$ pairs where a finite field let us say m many pairs returns the list of all degree d. Suppose the polynomial whose corruputed evaluations you are given is of degree d or less and b_i's let us say close to half of them are corrupt. There will be many polynomials actually which are consistent with this data. The algorithm will output all of them returns the list of all degree d polynomials G such that the number of i's on which G matches this is at least $\sqrt{2dm}$.

This can be significantly smaller than m so, that is for distance greater than 1 - d - 1 / m this is the distance of Reed Solomon code the list decoder handles 1 – 2d / m, for length m over the field alphabet for length m the correct values are $\sqrt{2d/m}$ and the rest is all erroneous so, these many errors are being handled. So, this is also what you saw in Johnson's bound that is the theory you had a half here, it is 1 but this d/m versus $\sqrt{d/m}$.

This seems similar to Johnson bond but remember that this is not a binary code, it is non binary alphabet. Since in Johnson bound you are talking about $1/2 - \delta$ and $1/2 - \varepsilon$. This

is the non-binary version of that you can say, but it is very good. This is able to handle a lot of errors, it seems that the errors can be close to100% still it is able to give you the list it is so good.

(Refer Slide Time: 43:14)



How do you do this, what is the idea? The idea just like Reed Solomon decoder is to introduce an auxiliary polynomial. Use a bivariate auxiliary polynomial Q(x,y) to fit the data. And then once you have this Q the auxiliary polynomial, you factorize it and the factors will magically give you the list. Compute a nonzero Q bivariate over the finite field F such that $Q(a_i, b_i)$ is 0 for all the input data points where the weighted degree of Q is this bound $\sqrt{2dm} =: t$. What is weighted degree?

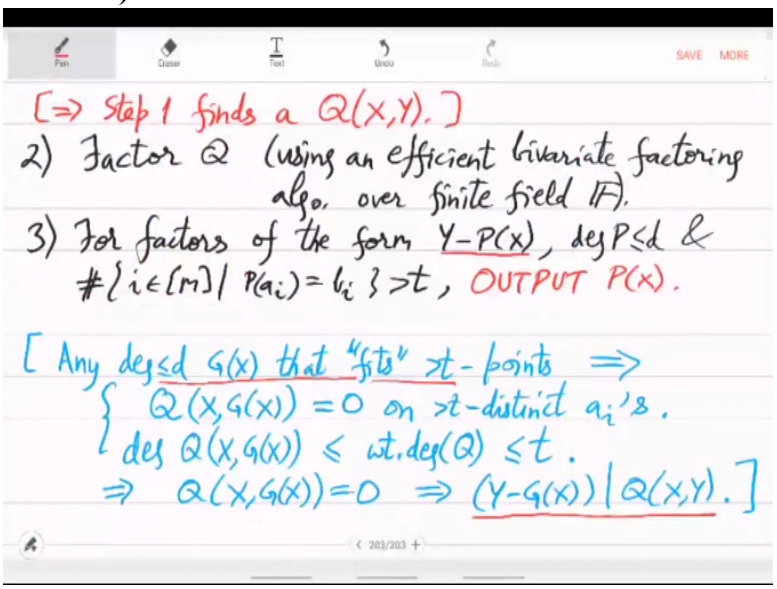
This is the max of i + dj for all monomials X^iY^j in the support of Q. Weighted degrees just like degree, but now X and Y have been given different weights X has been given weight 1, Y has been given weight d because you should think of Y has being this unknown message polynomial which had degree d. So, Y is given degree d and then any monomial $X^i Y^j$ has degree i + d j and you maximize over that.

So, number of monomials that Q can have this is how much, so you want i + d j maximum to be t which means that j can be at most t / d. So, j goes up to t / d and starts from 0 and for j how many i's, so the i's will be t - d j - t. This is the number is at most t - d j, so you get 1 plus that or you can also be 0. Over all these j's that is the number of monomials that Q can have for this weighted degree t which is what which is simple calculation.

So, it is $(1 + t)(1 + t/d)$ for the 1 + t contribution and for dj contribution you will get $- d/2(t/d)(t/d + 1)$. So, we can take out 1 + t / d and what remains is 1 + t - d / 2 around d / 2. In fact this is exactly equal $(1 + t/d)(1 + t - d/2(t/d))$. Now you can approximate this, so this is at least $t/d(1 + t/2)$, so that is $t/d + t^2/2d$ and you can see that this is m straight from here.

So, you get more than m, number of monomials in Q, this unknown Q. Our coefficients are the unknowns and the number of unknown coefficients comes out to be more than m and from the data, which was m pairs, you will get m constraints, so, the constraints are fewer than the unknowns. Number of unknowns is greater than number of equations in this homogeneous linear system, that means step 1 will find the Q.

(Refer Slide Time: 51:25)



This is the step and also the analysis. It can actually be computed and since, it is as simple as solving a linear system, you can do this in polynomial time. It will always exist, atleast 1 option will exist. So, you have a Q that is fitting the data and it is 1, d weighted degrees at most t. Once you have this Q what you do is factorize factor Q. That is step 2. Factor Q using an efficient bivariate factoring algo over finite field F. These exist such algorithms, you can look them up in the literature.

Uou will get irreducible factors of Q and the final step is for factors of the form Y - P(X) where degree of P is less than or equal to d and number of i's on which it fits a_i, b_i greater than t output P(X). This is the full algorithm fit the data with degree bounded Q, factorize Q and output the factors that are linear in Y and which satisfy this degree condition of the

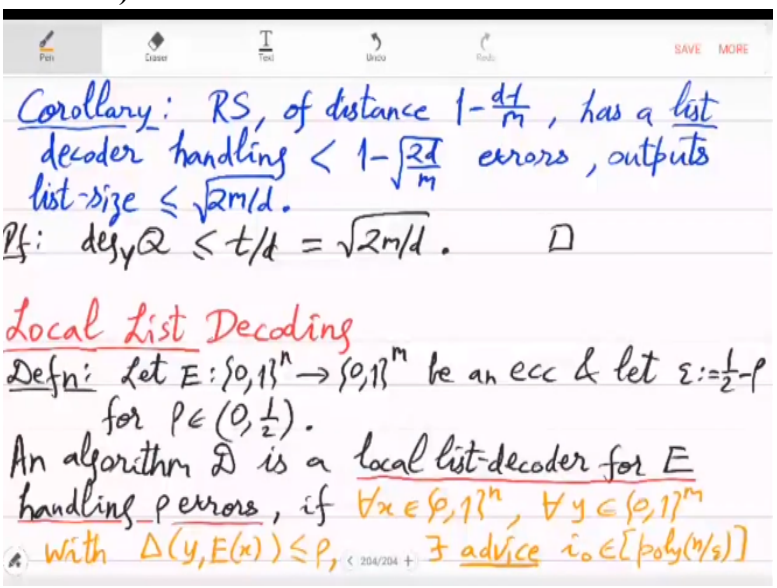
message polynomial that you were looking. The degree should be less than or equal to d and it should fit the points on more than t locations.

So, why will it work? The reason is any G(X) that fits greater than t points, it will satisfy the following property: it will actually be when you substitute in Q(X, G(X)) that will be 0 on greater than t distinct a_i's that is one thing Q will vanish at G(X) because simply because when you look at a_i, G_i, $G_i = b_i$ on more than t many a_i's, and Q fits that. So, Q is not completely vanishing, but you have more than t many values of a_i's.

And second thing is that if you look at the degree of Q(X, G(X)) this is less than or equal to t like this follows from the weighted degree of Q which we have set to be less than equal to t. So, Q(X, G(X)) is a degree t polynomial which is vanishing on more than t values, which could only mean that it is 0 which means that Y - G(X) divides Q(X,Y). So, any degree d, G(X) that fits a lot of points actually appears as a fact making hence factorizing Q made sense in step 2 and our output is correct.

This algorithm with the ideas and the descriptions given analysis given finishes, Sudan's theorem, so, for t $\sqrt{2dm}$ more than these many matches, f(G) matches more than these many places the data then you will find G and the list will be small. List will be small because this whole this algorithm is a polynomial time algorithm, so automatically the list is small. That finishes the proof.

(Refer Slide Time: 58:21)



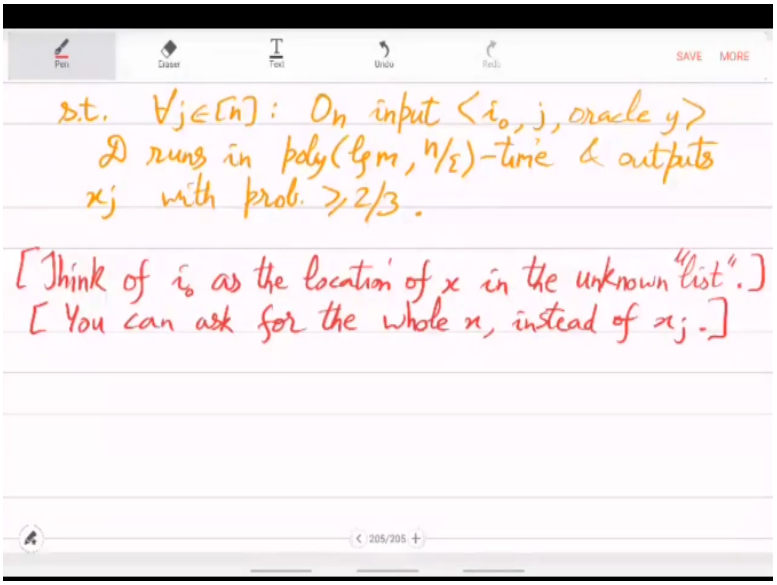
And we actually also get this corollary for the length of the list, so Reed Solomon of distance 1 - (d - 1) / m for d degree polynomials has a list decoder handling $1 - \sqrt{2d/m}$ errors and the list sizes that you can deduce by looking at the degree of Q with respect to Y and it comes out to be handling these many errors and outputs list size at most $\sqrt{2m/d}$. This will follow from degree of Q is t, degree of Q is weighted degrees at most t and you divided by d.

That gives you the degree in Y that is the simple proof. The proof is that degree of Q with respect to Y is at most t / d which comes out to be $\sqrt{2m/d}$. And since you output the factors Y – P(X), P(X) cannot exceed more than $\sqrt{k/d}$. That is the complete description of Reed Solomon list decoding. Now, we move to the final concept which is local list decoding and the goal here will be to find the list by making as few queries as possible to a long corrupted code word the focus will also be on the queries made.

Remember that this Reed Solomon that we did was not local because we actually fitted the whole data to find Q. We were actually querying everything this is what we want to improve upon. So, let E be a binary code and let ε be $1/2 - \rho$ for $\rho \in (0, 1/2)$. ρ will be again this local list decoding parameter - those many errors. There is no meaning to ε except how close is root to half, at half for binary code you will not be able to do anything but just below half what can you achieve?

An algorithms D is a local list decoder for E handling ρ errors. If for every message x, for every corrupted string y that is close to x there will exist, let us say an index for x. This kind of error is very high that is ρ is close to half, there will be many x's, so once you are given the index of x that you want so that we will call i_0 there exists an advice i_0 which is a number between 1 and polynomial in n / ε.

(Refer Slide Time 01:05:34)



Such that for all locations when you are given input this index or advice that is which x you want implicitly, this is what we are capturing int the jth bit of x that location and then oracle for y. D will run in time polynomial, again very few queries, so log m for m length corrupted code word log m time and also depends on n / ε if you are very close to half, ε is 0 then this is not possible, but just below half it is possible this much time.

So, D runs in this much time and outputs x_j with probability two-thirds, so that is the definition of a local list decoder handling ρ errors for any message x and corrupted code word y given as an oracle once somebody gives you an advice then any j^{th} location of x can be found with high probability and very efficiently. You could think of i_0 as the location of x in the list which is of course unknown and you can ask for the whole x in place of x_j.

Because if you cannot put x_j with high probability then by repetition you can also put all because the same i_0 will work for all the j's. So, it is as good as you can also output the full string x. That is the concept and will now do the final steps in this course. We will do local list decoding for Walsh Hadamard and Reed Muller and that will finish the course.