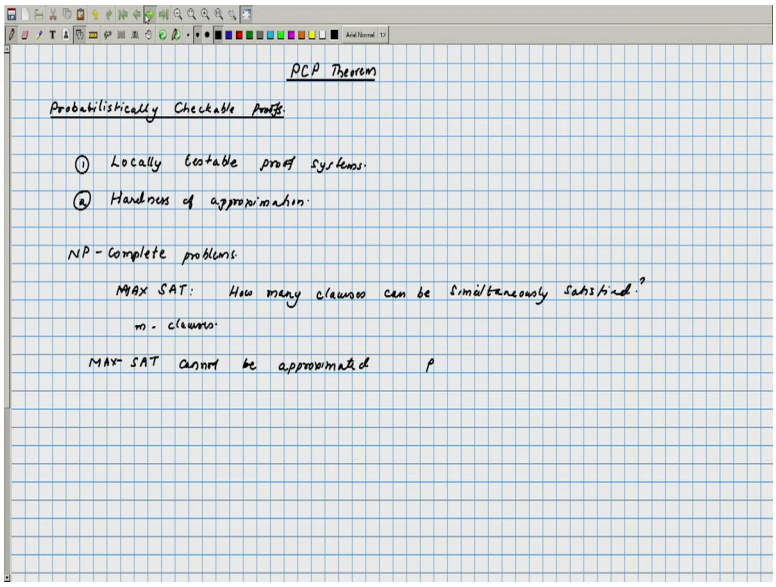
Randomized Algorithms
Prof. Benny George Kenkireth
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture – 36
Probabilistically checkable proofs - I

In this lecture, we will learn about PCP Theorem. So, PCP refers to Probabilistically Checkable Proofs.

(Refer Slide Time: 00:31)



There are two ways in which we can view PCP theorem. The first view is the probabilistically checkable viewpoint that is in some sense, locally testable property of proof systems. In one sentence it essentially means that proofs can be understood without going through the entire proof. You can just look at parts of the proofs and understand the proof. And the second view is the hardness of approximation view.

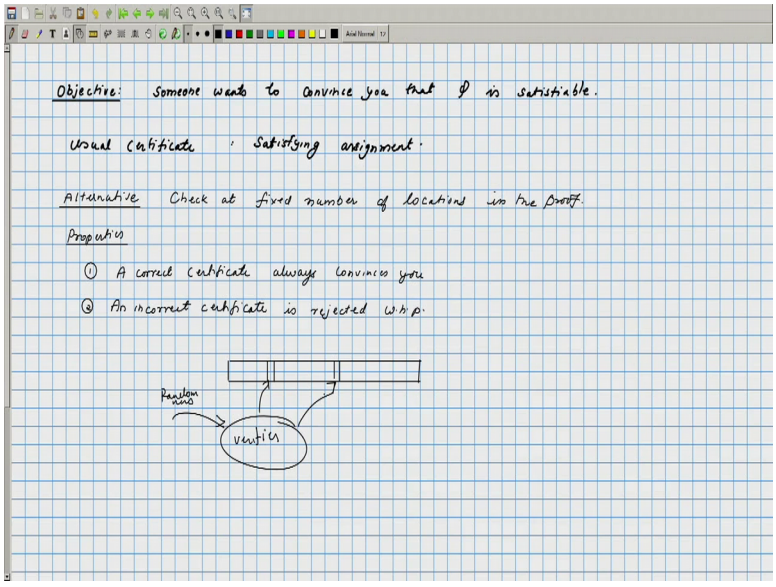
So, we were looking at let us say NP hard problems or NP complete problems. For example, let us say MAX SAT. So, here when you are looking at MAX SAT problem the question is how many classes can be simultaneously satisfied, ok. So, you are given a Boolean formula containing different classes, let us say m classes, now you want to know what is the maximum number of classes that can be satisfied. Now, if the answer to this question is m; that means, all the classes can be simultaneously satisfied; that means

the formula itself is satisfiable. In case of unsatisfiable formula, we want to know what is the maximum number that can be simultaneously satisfied.

Now, PCP theorem essentially tells that if you could approximate this above some particular threshold then that itself is good enough to solve the particular problem. So, PCP theorem essentially says unless P naught equal to NP MAX SAT cannot be approximated to an arbitrary high accuracy, ok. So, there is a threshold, let us call that as rho. So, if you approximate MAX SAT to above a certain number rho then you can actually solve the exact instance and therefore since the exact instance we assumed to be a difficult problem. In the sense, it cannot be solved in polynomial time we will PCP theorem, by PCP theorem you can say that MAX SAT cannot be approximated, ok.

So, that is the hardness of approximation view, but in this lecture what we will be looking at is the locally testable proof systems, ok. So, that view of PCP theorem is what we will try to understand. So, what does this mean?

(Refer Slide Time: 04:09)



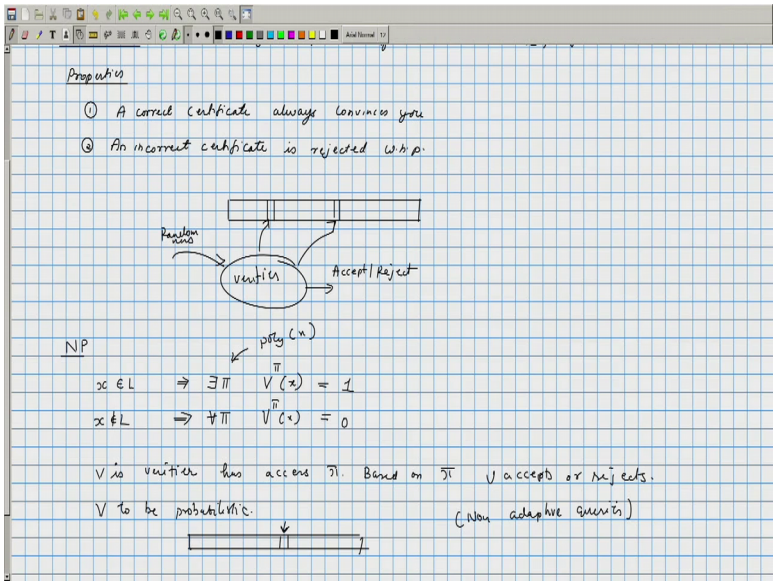
So, let us look at our objective. So, let us say someone wants to convince you that a particular Boolean formula is satisfiable. So, let us say phi is a Boolean formula and someone wants to convince you that phi is satisfiable. The usual approach would have been to give the assignment which satisfies phi that is called as a certificate. Usual certificate is a satisfying assignment.

The problem with this is you have to go through the entire assignment and based on what you identify as the value of each particular variable you can evaluate the expression and say that it is satisfiable or unsatisfiable. The question that we are looking at is can we look at a part of the certificate and still be convinced that the formula is satisfiable, ok. So, the alternative that we are interested in is check at fixed number of locations in the proof and based on what you see in these locations you will make a choice as to whether you will make a decision as to whether the formula is satisfiable or unsatisfiable, ok.

So, the properties that you would require are the following. If you have a correct certificate then that will never fail to convince you that the formula is satisfiable and an incorrect certificate is rejected with very high probability. So, if the formula is unsatisfiable then all certificates that going to be incorrect certificates. So, for the unsatisfiable formula with very high probability all the certificates will be rejected, ok. So, this is intuitively the requirement, this is what we want to have, ok. So, diagrammatically you can think of this.

A large certificate is given, ok. Now, since we are examining only few bits from the entire proof the length actually is not so much of, the length of the proof is not or the length of the certificate is not so much of an issue, ok. So, once this proof was given, the verifier based on some random choices examines certain locations, ok. So, these locations are queried and based on what it finds there based on that the verifier decides whether accept or reject, ok.

(Refer Slide Time: 08:03)



In order to understand this, let us look at this notion of NP a bit more carefully. The PCPs are essentially a relaxation of NP in a certain way. So, when you are looking at the class NP we will say that the language belongs to NP if the following conditions were true. x belongs to L implies there exists a proof or a certificate pi such that verifier with access to pi on input x will always return 1, and if x does not belong to L that would imply that for any pi V pi of x will be equal to 0. The way this has to be understood is V is a verifier which is a deterministic program. Now, that deterministic program has access to the values pi or has access to the certificate pi based on pi V accepts or rejects, ok.

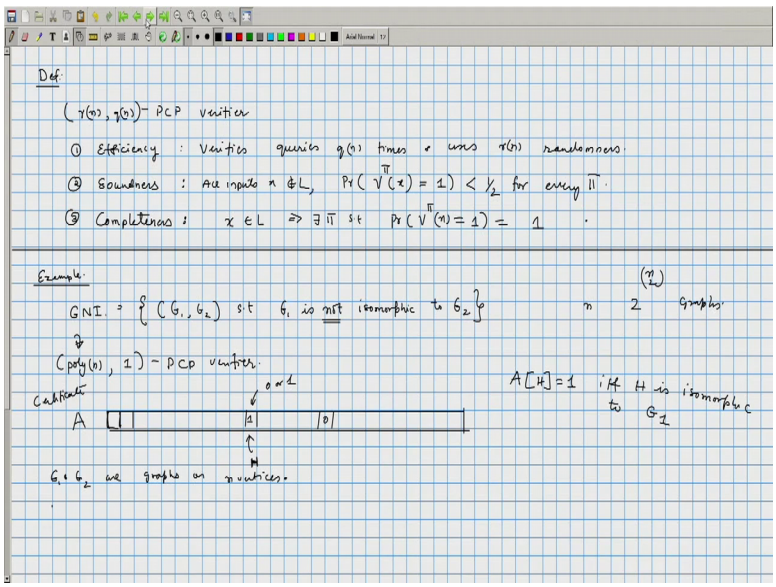
Once the input is fixed and the certificate is fixed, then the verifier is a deterministic program which takes a decision as to whether the input is there in L or input is not there in L. Language belongs to a NP if there is a verifier such that for all strings in L the verifier will accept and for all strings not in L the verifier will reject no matter what was the certificate, ok.

Now, when we look at PCP we are essentially relaxing certain conditions, ok. We no longer insist that V is deterministic and we allow V to be probabilistic. So, the certificate pi here has to be bounded in it should be some poly log, poly n. Means its length should be bounded by a polynomial in n. When we look at PCP that requirement is really not there, but instead we will assume that there is an proof if you think of it as a long bit

vector we have access to any position. So, pi is some array with random access, ok. So, any position of pi can be found quickly, ok.

So, we will formally define what is a PCP verifier. The intuition as this that verifier is a randomized verifier and it has access to and it has random access to the certificate. So, let us formally define what is a PCP verifier. So, there are two parameters one is the amount of randomness that is there and the second is the number of queries, number of locations that is being accessed in the proof. We will assume that the accesses are non-adaptive. Once the proof is given the verifier is querying individual locations inside the proof which location it will query will not depend upon the answers to the previous queries, ok. Such kind of queries are called as non-adaptive queries. Of course, adaptive queries is more powerful than non-adaptive queries, but we will restrict ourselves to non-adaptive queries, ok.

(Refer Slide Time: 12:17)



So, formal definition of PCP verifier; so, we will call this as an r n, q n PCP verifier, ok. So, we want to say when there is a language L have an r n, q n PCP verifier. So, this is the case when there are so 3 conditions has to be satisfied, one corresponding to efficiency, the second condition corresponding to the soundness, and the third corresponding to completeness.

So, efficiency condition says that the verifier q n times and uses r n amount of randomness. So, n is here denoting the input size. So, the number of queries and the

amount of randomness used can depend on the size of the input. So, input is of size n then the number of random bits used is bounded by r n and the number of queries is bounded by q n. So, that is efficiency criterion for the PCP verifier with parameters r n and q n.

Soundness condition says that for all inputs that is not there in the language which does not belong to L the probability that the verifier accepts. So, x is the input the verifier when we say V pi x; that means, the verifier with access to the proof pi on input x, so this will say 1 and that probability is going to be less than rho for every or let us say less than half for every pi. And the completeness says that x belongs to L implies there exists pi such that probability that V pi x is equal to 1 is equal to 1 with probability 1 the proof will be accepted, ok. So, this is the definition of the PCP verifier.

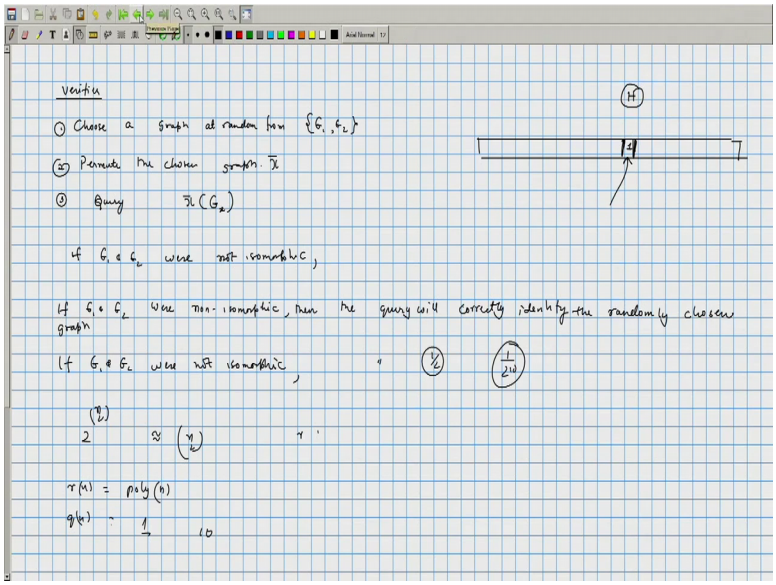
So, let us see an example of a PCP verifier. So, we will look at the problem of graph non-isomorphism, ok. So, this is the language consisting of graphs G 1 comma G 2 such that G 1 is not isomorphic to G 2. So, we will have a poly n comma 1 PCP verifier, ok. So, here the amount of randomness that is made is proportional to a polynomial in n. The number of queries made is going to be at most when; so, order of 1 constant number of queries. So, the input will consist of two graphs G 1 and G 2, and we want an algorithm which will tell that these graphs are not isomorphic when they are not isomorphic and wanted to say that they are isomorphic if they were isomorphic, ok.

But when we think of when we want to design a PCP verifier for all s instances the probability of the algorithm returning the correct answer should be equal to 1 and for all the no instances the probability of saying s should be bounded by half, ok. So, basically the proof or the certificate is going to be a long bit string where each position corresponds to a particular graph, ok. So, each position is indexed by a graph, so look at all possible graphs on n vertices.

So, we may assume that G 1 and G 2 are graphs on n vertices because if they were not on n vertices you can readily throw them out because they are not going to you can easily identify that they are not isomorphic. And each position in this long array will correspond to one particular graph. So, look at all graphs on n vertices and each position is indexed by that, ok. So, on n vertices are 2 raise to n choose 2 graphs. So, this is a bit string with those many entries, ok. And this entry here is either 0 or 1, ok. So, if it is 0

you can encode in such a way that this position was 0, so let us say it is called as array as A. So, A H is equal to 1 if and only if H is isomorphic to G 1. So, this is the proof or the certificate, at each position you can go to that position and check whether it is 0 or 1. If it is 1 that would mean that the graph H is isomorphic to G 1, ok. So, once this certificate is constructed the verifier can do the following, ok.

(Refer Slide Time: 18:53)



The verifier just chooses a random graph from G 1. So, verifiers algorithm as follows. Choose a graph at random from G 1 G 2, ok. The adversary or the prover does not really know which of the graphs he had chosen. So, once this graph is chosen the second step is to choose a random permutation. So, permute the chosen graph, ok. So, just remap the vertices using some particular permutation pi, ok. And we will query the location pi G x, ok. So, suppose x was the graph that was obtained and pi G x is what we query. If G 1 and G 2 were non-isomorphic then we know that when you query at pi G x the value that you have to get is the random graph that you have picked, ok.

So, this was the proof or the certificate, now we are querying at some random location now, that random location is some particular graph H. This H is isomorphic to precisely one of G 1 and G 2. And the value here will basically indicate that, ok. Clearly, as this location is isomorphic to only one of them, we had, if this was 1 that would mean that the graph that we had initially chosen should have been G 1 and if it was 0 it would mean that it had to be G 2. So, by querying this location if the proof correctly identifies the

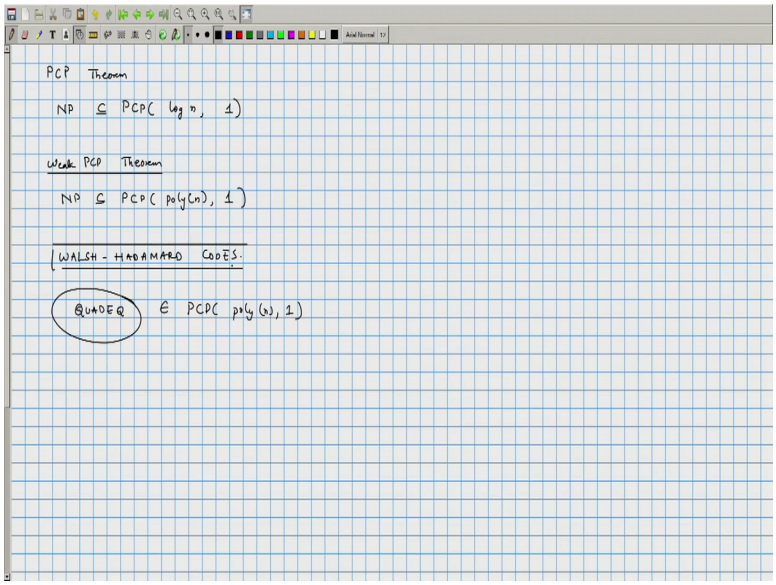
graph from which we had randomly obtained H then we can accept otherwise we can reject, ok.

So, this process will have the following property. If G 1 and G 2 were non-isomorphic then the query will correctly identify the graph and if G 1 and G 2 are not isomorphic, the query will be correct with probability half; so, can be little more careful while we are designing the proof. If the particular graph is isomorphic to H we put it as I mean to G 1 you put it as one if it is not isomorphic you put I mean if it is isomorphic to G 2 we will put 0 on all other positions you can randomly choose between 0 and 1, ok. So, if you do it this way, if the proof is constructed this way then the verifier will be convinced that G 1 and G 2 are non-isomorphic whenever they were actually non-isomorphic. And if and if they were not isomorphic then the probability of the verifier getting convinced that they were isomorphic is going to be less than half.

So, this is PCP proof or a probability PCP verifier and the parameters here. So, each of them is a graph and therefore, the amount of randomness required to choose one number between 1 and 2 raised to n choose 2 is approximately n choose 2. So, you need to have n choose two random bits in order to choose a graph at random. So, r n is equal to poly n and the number of queries we just made one query.

If you wanted to have let us say more guarantees in terms of the correctness, so here the correctness is to the extent of half the sense the incorrect input could be accepted with probability half. If we wanted to reduce the error probability you can do let us say 10 queries and bring down the error probability to 1 by 2 to the power 10. So, repeated queries can improve the performance, ok. But anyway, r n is going to be poly n and queries will be constant number of queries, ok. So, this is an example of a PCP system. Now, we are in a position to state the PCP theorem, ok. We have understood what is a PCP system or a PCP verifier.

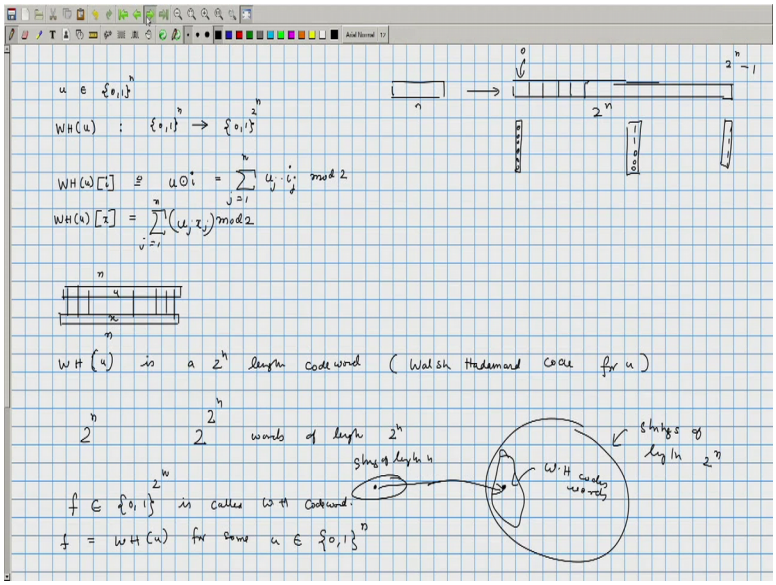
(Refer Slide Time: 24:15)



Now, PCP theorem tells something about the relationship between NP languages and the PCP languages. So, this says that NP is a subset of PCP log n comma 1. In other words what it means is every language in NP can be accepted by a PCP system or a PCP verifier and the randomness that is required is bounded by log n, the number of queries is constant, ok.

What we will prove in these lectures is a weaker form. We will show that NP is contained inside PCP poly n comma 1, and in order to use this we will have some tools. So, we will learn about Walsh Hadamard at code words, ok. So, the first thing that we will do now is to learn about what are Walsh Hadamard codes and then look at one particular problem. We will look at an NP complete problem called as quadratic equations or QUADEQ which is known to be an NP complete problem and we will show that PCP poly n comma 1 can, we will show that QUADEQ belongs to PCP poly n comma 1. So, this is what we will prove. In order to do this, we will look at what are known as Walsh Hadamard codes, ok.

(Refer Slide Time: 26:41)



So, these codes are basically linear functions. So, let us say that u is a string of length n, and the Walsh Hadamard code word corresponding to u which we will denote by WH u is a vector, ok. So, this is a function from 0, 1 power n to 0, 1 to the power 2 to the n. So, you take a string of length let us say n and map it to a string of length 2 to the n. This function will be I mean, so the Walsh Hadamard code word is one such function and its defined at the following way.

So, when we are looking at mapping an n bit string into the 2 to the power n size string, we can look at each position and say what is going to be the value at that position. So, you can think of the code word corresponding to you as a word of length 2 to the power n and we will think of this entire code word as an array indexed by 0 to 2 to the power n minus 1, sorry 2 to the power n minus 1, ok. So, since there are 2 to the power n locations instead of thinking them as numbers, we can also think of them as bit vectors, ok. So, this is the all 0 vector and the last position is the all 1 vectors and maybe in between some where is the 1 1 0 0. So, look at any particular code word.

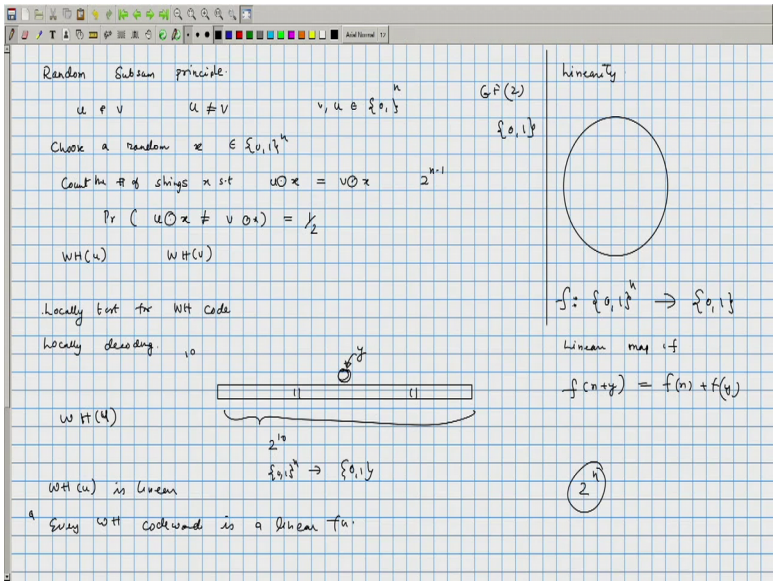
Corresponding to that code word; corresponding to that position there is a number 0 or 1, and instead of thinking of the indices as ranging from 0 to 2 to the power n minus 1 we will think of the index set being bit strings of length n. So, when we say WH u is an array we can look at the ith location; so, WH u and ith location. So, i now we are going to think of it as a bit vector of length n; so, WH u of i is defined as u dot i, ok. So, this is

nothing but summation j going from 1 to n, u j times i j mod 2, ok. Instead of i if u think of them as x at the x position. So, x is now you can think of x as a vector this is equal to summation u j, x j this summed up and evaluated mod 2, j going from 1 to n. So, u is now a vector of length n and x is a number which we are thinking as a bit vector of length n, this is x, this is u take the dot product I mean and then compute it mod 2, and that is WH u of x.

So, now, so this is called as, so WH u is a 2 raise to n length code word and we will view it as the Walsh Hadamard code for u. Now, if you now look at all the strings of length 2 to the power n, there are 2 to the 2 to the n such words. Out of them very few2 of them it is precisely 2 to the n of them are going to be Walsh Hadamard code words, ok. So, if you look at this collection of all words of length 2 to the power n in that there is a small set. So, these are all strings of length 2 to the n, ok. In that there is a small subset which are Walsh Hadamard code words, and each word here is the Walsh Hadamard code word for some particular string of exactly n, ok.

So, these are strings of length n, each string of length n is mapped to some Walsh Hadamard code word. And we will say that let us take one particular f belonging to 0, 1 to the power 2 to the n. We will say that it is a Walsh Hadamard code word if f is equal to WH u for some u belonging to 0, 1 to the power n. This is the case then f is called a Walsh Hadamard code word, ok. Now, there are few very important properties of these code words which makes it useful in designing probabilistically checkable proofs. We will see those properties.

(Refer Slide Time: 32:51)



So, the first property that we will understand of these code words is the random sub sum principle, ok. So, take two words u and v which are not equal, ok. So, this is some arbitrary. So, imagine that u v both belong to 0, 1 to the power n. So, choose a random x belonging to 0, 1 to the power n. So, u and v are both belong are n bit are n bit vectors and choose a random vector, ok.

And now, you can look at the number of strings x such that u dot product x is equal to v dot product y. So, these are dot product computed mod 2. So, all the computations here unless mentioned will be over GF 2, it means you are looking at 0 and 1 and the addition and multiplications are carried out mod 2 in this particular field, ok.

So, you want to count the number of x as that u x is equal to v x and their number will be exactly equal to, I mean they will be half of the total number of numbers. So, 2 raised to n minus 1 of them would make it equal and 2 raise to n minus 1 would make them unequal. So, in other words probability that u x is equal to v x is going to be equal to or not equal to v x will be equal to half, ok. If you take two things which were unequal and multiply it with x the probability that they are still remain unequal is going to be half.

This would essentially mean that if you look at the Walsh Hadamard code word for u and the Walsh Hadamard code word for v they will differ at half of their positions, ok. In other words, we will say that the distance of Walsh Hadamard code words is half. Means, half of the words I mean if you look at a random position in the Walsh Hadamard code

word corresponding to u and v which are unequal then the probability that they are still unequal is half, ok.

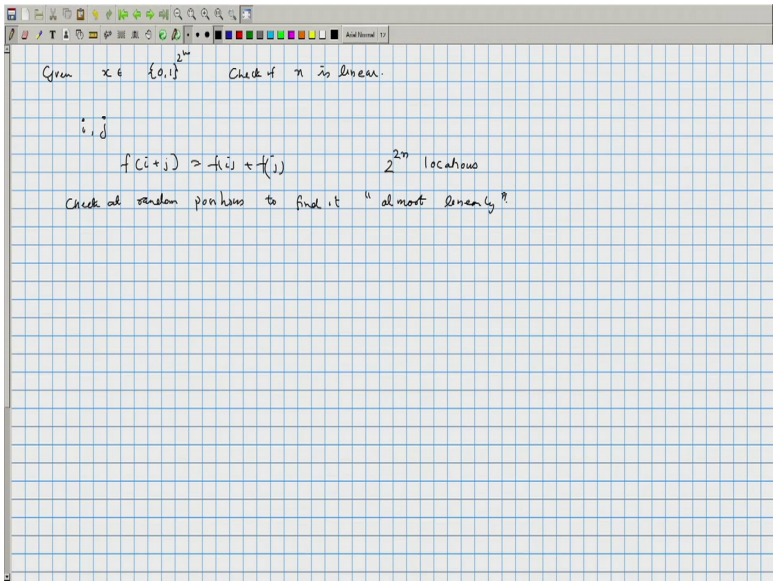
And because of this key property called as random sub some principle we can do the following things on the Walsh Hadamard code words. We can locally test for Walsh Hadamard code words and we can do local decoding, ok. What does this mean? So, Walsh Hadamard code has an important property called as linearity, ok. So, we were looking at all the Walsh Hadamard code words, we can also look at functions which takes strings of length n to strings of length 2 to the power n.

We will call one such function as a linear map if f of x plus y the addition carried out mod n is equal to f x plus f y, the multiplying by constants does not really matter much here because there are only two numbers, 0 and 1, and those conditions are trivially verified. So, here linear essentially means we have to verify this condition f x plus y equals f x plus f y.

Note that Walsh Hadamard code words are linear, ok. So, we were viewing the code words as functions, ok. So, some of those functions are going to be I mean, so if you look at arbitrary functions from 0, 1 to the n, to 0, 1 to the 2 n, WH of any u is basically a linear function, ok. For each particular u you will get a different code word, and each of those code words we could view them as functions, ok. So, this is a code word, let us say our strings where of length 10 and this is an array of size 2 to the power 10. We could also view this as functions from 0, 1 to the n to 0, 1, ok. So, this is, ok. Look at functions from 0, 1 to the power n to 0, 1 it is called linear if f of x plus y is equal to f x plus f y, ok.

Walsh Hadamard code words are linear functions and every Walsh Hadamard code word is a linear function, ok. So, they are essentially the same set, you can either view them as linear functions or you can view them as WH code words. So, in order to check if something is a WH code, so, you are, so here now you think of the problem where you are given a string of length 2 power n and you want to identify whether it is a Walsh Hadamard code word. In order to do that it is sufficient if you check whether it is a linear function, ok. In order to check it at a linear function we have to essentially look at all possible x and y.

(Refer Slide Time: 39:05)



So, given x belonging to 0, 1 to the power 2 n check if x is linear, ok. So, you can look at all positions i comma j and check if f i plus j is equal to f i plus f j, ok. But i can take 2 to the power n different values, j can take 2 to the power n different values. So, this basically requires you to check at 2 to the, 2 to the n locations, ok. So, that is not quick enough, but we can locally decode.

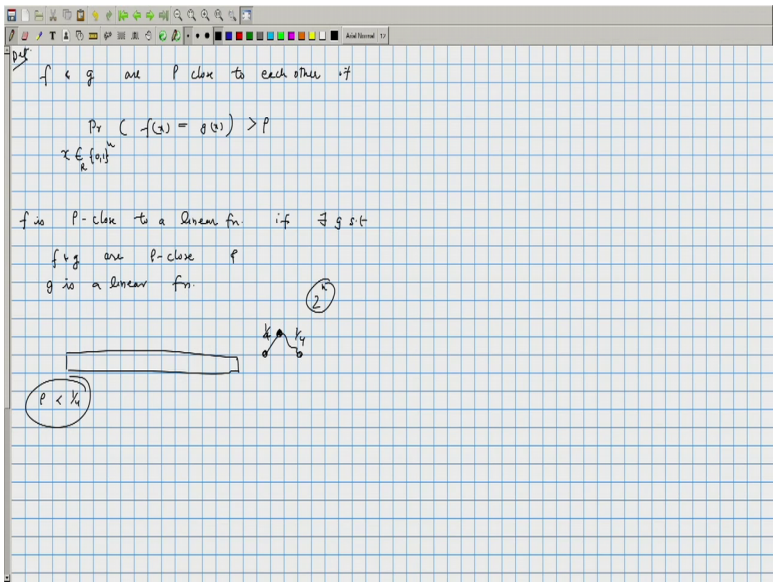
We will just randomly look at one particular position, ok. So, check at random positions, and based on the values at the random position we can decide whether it is linear, ok. We cannot actually check if it is linear, but we will check for almost linear, ok. So, in nutshell what this means is if you are given a string of length 2 to the power n you can locally check at few positions, and after checking these positions you can say that this particular bit string of length 2 to the power n is very close to a linear function, ok.

So, that is because of certain properties of Walsh Hadamard code words, so we will describe those properties. And further, if you had Walsh Hadamard code words, if you want to find the value at some particular position you can figure out the value at that position by checking couple of locations. Even if the adversary had modified the value at that exact position we can do a decoding by accessing very few locations inside the code word. The other positions essentially helps us figure out the value at position.

So, let us say we were interested in finding the value at y you do not necessarily have to query at y, we could look at other positions inside the code word and based on the value

we found there we can figure out the value at y. These properties are called as locally testing for WH code and locally decoding the WH code, ok. We will need couple of definitions before we proceed.

(Refer Slide Time: 41:45)



The first is our relaxation from linear to almost linear. So, we will say that f and g are row close to each other, row some parameter between 0 and 1. So, they are close if the probability of f x equal to g x; when x is chosen randomly from 0, 1 to the power n uniformly at random from this. So, this probability if this is greater than rho then we will say that they are rho close to each other, and we can say that f is rho close to a linear function if there exists a g such that f and g are rho close and g is a linear function, ok.

So, if you have these two definitions in place then we can say, whenever we have codeword the amount by which it is close is defined by rho. So, we can say whether they are close to a Walsh Hadamard code word or a linear function, ok. Now, the important point is if your rho is let us say less than one-fourth, for any particular bit string of length let us say 2 to the power n, there can be at most one Walsh Hadamard codeword to which it is say close.

And if you try to find the nearest Walsh Hadamard code word corresponding to this particular bit string there is a unique one, because if there are two of them which were close and the distance is bounded by half by one-fourth then between those two strings they are going to be closed by length half. I mean they will agree on at least half of their

bit string which we know is not the case because Walsh Hadamard code words are reasonably separated. They differ in two different Walsh Hadamard code words differ at half of the bit positions. So, for any code word within one-fourth, I mean with if you say that they are one-fourth close to a linear function and that is precisely one particular code word.

So, we will see how the linearity of a given a code word, how can we find the code word which is reasonably close to it and how do we decode individual locations inside the code word. So, that is what we will do in the next lecture.