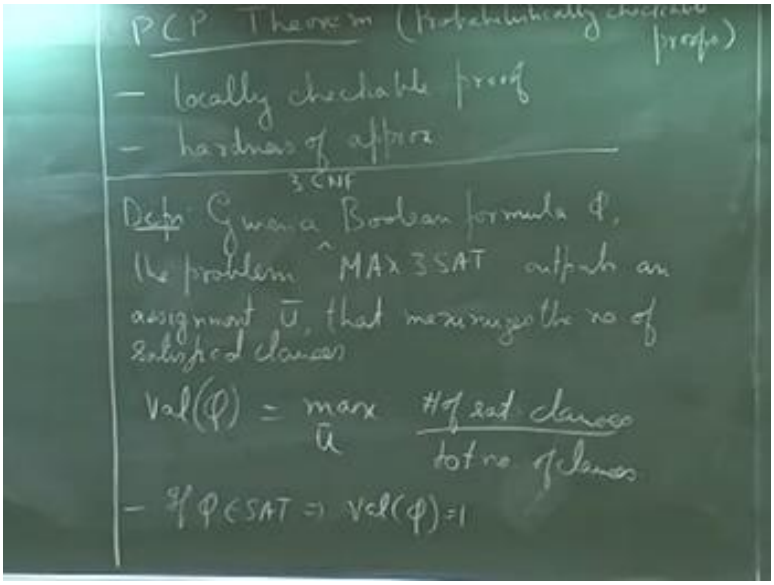
Computational Complexity Theory
Prof. Raghunath Tewari
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture -40
Introduction

(Refer Slide Time: 00:13)



So, PCP stands for probabilistically checkable proofs. So, before I go head, let me just give a brief history. So, what happened in this early 90s, I mean after the proof of Shamir showing that IP is equal to PSPACE. So, people started looking that other any other such interactive model where you have a all powerful prover and a verifier which will characterize lower classes. So, IP characterized PSPACE but what about NP?

Because NP is sort of more of more interest to complexity theories. So, they were trying to see that how can we characterize NP. So, one strategy which was thought of this what if we bound the space that the verifier can use. So, earlier the bound on the verifier was that it is a probabilistic polynomial time machine but in addition to that what if you also bound the space. So, that had a line of work I mean, although it did not turn out to be very fruitful.

But was an important line of work nonetheless. Alternatively, there is other group of researchers

who are trying to see that, so the prover produces a proof or a certificate. Now, what if the verifier only has restricted access to that certificate? In the sense that verifier is not able to read the entire certificate may be let say just a few bits of that certificate. Suppose, if we consider a model where the given an input, so let us say we are looking at the SAT problem.

So, given a Boolean formula phi prover will come up with an assignment but now the verifier does not look at the entire assignment. Of course if the verify looks at the entire assignment, he can plug it into the formula and check if it is satisfiable but suppose if he has access to only a few bits of that certificate what can be done. But, we allow the verifier to be not only polynomial but also probabilistic.

So, this was the sort of the starting point and after a few couple of papers by several people, in fact, I think there are some eight or nine people who are credited for this proof they were able to show that a certain model does actually characterize NP, exactly and again, this is another Godel Prize winning work all these eight or nine guys, they got the Godel Prize in 2001 for this one. So, this is a very important.

So, we will not look at the proof because the proof is quite complicated it needs a lot of time but at least the goal today is to look at the definitions, what is this system, what are we trying to prove here and maybe try to motivate why it is important? So, that is what will try to see and there is also another way of looking at this PCP theorem that is in terms of the hardness of approximation.

So, a very important way of looking at NP hard problems is to ask the question that can we approximate. I mean, if I mean if not get an actual answer, can we get an approximate answer. Again with respect to the SAT problem, suppose if I ask the question that given Boolean formula phi can you at least give me a assignment which satisfies at least half the number of clauses or maybe five sixth of the number of clauses or some constant.

That is not getting an exact answer but at least some approximate answer. So, this is the problem of approximating NP hard problem and so again say approximation became an interest of study

soon after NP completeness was studied that is in the mid 70's and people were trying to see that can we get an approximate solution to an NP problem NP complete problem, which is as close as possible to 1.

Suppose, for any epsilon greater than 0 if we can get an approximate solution, that is 1 minus epsilon good. So, we will see what that means then for most practical purposes NP complete problems are not much of a hurdle because we are able to achieve quite a good amount of approximation but that also phase tape, road block and research as soon realize that probably even approximating an NP hard problem is a difficult job.

But they are not able to formalize this, how do you formalize, what does it mean say that approximating is also hard. So, another way of looking at PCP is via this approximation method and what they achieved or was to show that even approximating NP hard problem up to some constants is a hard thing in the sense that if that is achieved it will show that is equal to NP. So, that is basically what our agenda is going to be today.

So, look at these two different notions and see at least at least get the idea why they are equivalent. As I said that, we have these two models. So, one is a locally checkable proof notion and the second notion is that of hardness of approximation. So, let us just keep this at this point for the time being let us look at what it means to say that we can approximate a problem. So, let us diverge a little bit. So, firstly let me define an optimization version of the three SAT problem.

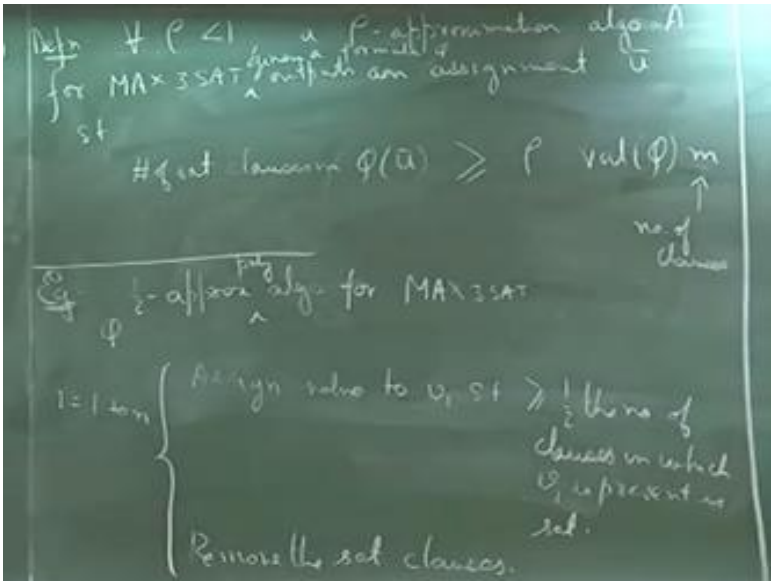
So, given a Boolean formula will actually always deal with three SAT Boolean formulas but let me just still write it for the sake of completeness, given a three SAT Boolean formula phi the problem MAX 3 SAT outputs an assignment. So, let us call it u. So, basically u is an assignment to the variables of phi that maximizes the number of satisfied clauses I am sorry 3 CNF that is all these clauses contain three literals.

And, we say that the value of phi value of phi basically will be that what fractions of the clauses get satisfied? So, the value of phi is the maximum over all assignments u that will maximize the following quantity number of satisfied clauses divided by the total number of clauses. So, given

any Boolean formula phi you always have a value Phi attached to it. So, this is define this way. So, if phi is satisfiable what is value of phi? It is 1.

So, that is a simple observation that phi belongs to SAT then value of phi because there is some assignment which will satisfy all the clauses. So, the next is an approximation algorithm.

(Refer Slide Time: 10:42)



For all rho less than 1 a rho approximation algorithm A for MAX 3 SAT, outputs an assignment U so what do you expect it to be? Of course given a formula 5, so given a formula of phi it will output an assignment U such that the number of satisfied clauses in phi of u is at least rho times value of phi times m, where m is the number of clauses. So, basically this quantity is the total number of possible clauses that can be satisfied by any assignment.

So, we say that this algorithm will approximate it up to a factor of rho, if it outputs an assignment u bar that will satisfy at least rho times that many clauses. So, let us look at an example a half approximation algorithm for MAX 3-SAT and not only is it a half approximation it say half approximation polynomial time, polynomial time algorithm for MAX 3-SAT. So, this is quite simple.

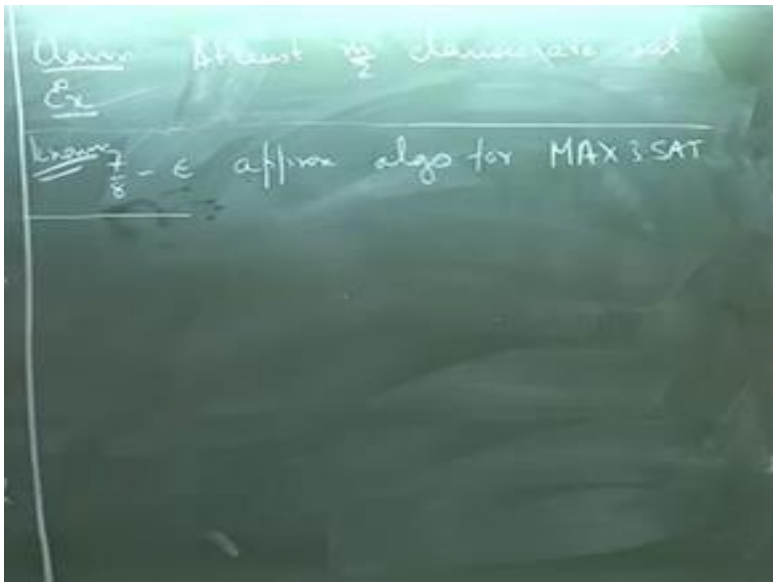
So, given let us say a Boolean formula phi what the algorithm will do is, it will do the following for let us say some i going from 1 to n, so it will pick the variable V i, so initial to pick variable

V 1 or in this case so let us pick some variable V 1 in the formula and it will try to assign a value to that variable that would satisfy at least half the number of clauses in which that variable is present. So, the variable can be present as in the positive form or in it is negative form.

So, I can always assign a value 0 or 1 to it. So, that at least half the number of clauses in which it is present gets satisfied. So, assign value to V i such that greater than or equal to half the number of clauses in which V i is present is satisfied. This can easily be done in polynomial time I will just run through the entire formula and then what we do is that after we do this assignment, we remove all those clauses, remove the satisfied clauses.

So, those clauses that got satisfied as a result of an assignment of value to V i, just remove them remove the satisfied clauses and you do this for every variable in an iterative fashion. So, the claim is that at the end the values that get assigned to all the variables will satisfy at least half the number of clauses that can very easily be seen from this I mean, basically the way this algorithm is designed.

(Refer Slide Time: 16:38)



I am just leave this as a small exercise; if it as m clauses at least m by 2 clauses are satisfied. So, the idea way to argue this is basically you each time you look at the number of clauses that are not satisfied and finally you show that that can be at most m by 2. So, I just prove this; so this is a very easy half approximation algorithm for MAX 3 SAT and in fact what can quite easily not

very easily but what can be achieved is.

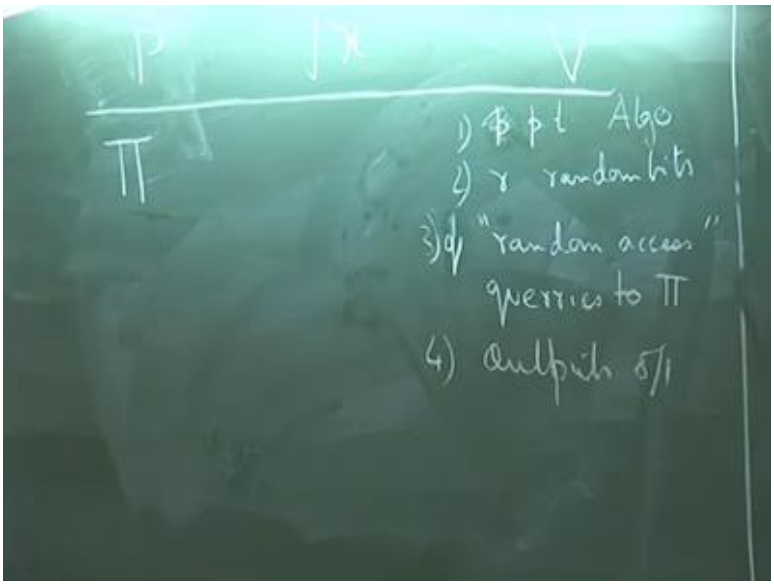
You can get something like a 7 by 8 minus epsilon approximation algorithm for this problem. For any epsilon greater than 0, there was a proof and this was not very, I mean, this was quite recently actually using semi definite programming some people showed that you can come up with a 7 by 8 minus epsilon approximation algorithm for this problem and suppose I mean, suppose if you look at the special case where each clause has distinct literals that is literals, do not reappear.

In that case, suppose, if I look at a random assignment of values to all my variables and let us look at one clause what is the probability that that clause will get satisfied so suppose you have just one problem think of a Boolean formula which has just one clause it has 3. It will be 7 by 8 because only one assignment will make it unsatisfiable and all the other seven assignments to those three literals will make it satisfiable.

So, suppose if you have a Boolean formula where all the literals are distinct and if you pick a random assignment that will get you a approximation not quite I mean because it still not deterministic polynomial always, any random assignment will always satisfy, actually, so the expected number of thing will be but again this can easily be made deterministic also this approach.

So, that is the easy case but generally we will have Boolean formulas where clauses will repeat literals, that is why this proof is not quite trivial but this is what is known. So, let us come back to PCP is now, so what is the PCP so first let me give a informal idea and will then formalize it.

(Refer Slide Time: 20:17)



So, we have a prover and we have a verifier. So, given an input x, both the prover and the verifier what the prover does it? In generate say certificate pi and if that x belongs to the language then there always exists a proper certificate and if the input does not belong to the language the goal of the prover is to some over cheat the verifier. Basically, try to convince them falsely that x does belong to the language.

What the verifier does is, the verifier is firstly a probabilistic polynomial time algorithm it picks some r random bits, so see what we want our r to be and it makes random access queries makes let us say some q many random access queries to the proof pi and then finally it outputs 0 or 1 depending on whether it thinks that x does not belong to the language or it belongs to the language.

So, these parts are, we are familiar with these parts but what does it mean to say that it makes random access queries. So, basically what the verifier does is that it does not of course it does not go through the entire proof it just writes an index of this proof. So, let say there is some special tip, there is some special address tip where the verifier writes i then in the next step the verifier will basically get what is the bit which is contained in the ith position of pi.

Basically, what is the value of pi of i? So, that is basically one query, so this query is something which is very valuable and the verifier does not have too many queries at his disposal so he very

carefully selects what is queries are going to be. So, let us say he says 10 may be he writes binary of 10 and in the next step he gets what is the bit at the 10th position of pi then again he may be using some random coin tosses and he says that may be 99th.

So, he will get what is the bit at the 99th position. So, in this manner he is allowed to make some q queries. So, the reason why this model is useful is that now note that the length of pi can actually be exponential in the address that the verifier writes. Suppose, if the verifier wants to come see what is the 99th element in pi you needs to writes the binary of 99 and the binary of 99 is just some log 99 bit number.

So, if the verifier has let us say the polynomial time machine and he writes some polynomial address then I mean, ideally the proof pi can be exponentially long. By writing a polynomial and address you can refer to a proof, which is 2 to the power, that polynomial in length that is what is crucially. So, this clear to everybody we are saying here. So, pi is a certificate which the prover generates.

So, again prover is somebody who is all powerful I mean, there is no computational bound on the prover. Prover can generate any certificate. So, his goal is to always convince the verifier that the input belongs to the language. For example, let us look at that case again when we are looking at the language SAT. So, then given let us say a Boolean formula phi the verifier what we know I mean the standard certificate based definition the way I mean, the proverb basically generates an assignment to than Boolean formulas.

So, such that if the Boolean formula is satisfiable then plugging in those values will get it satisfied and if it is not satisfiable then no matter what assignment he produces, he can never satisfy the formula phi. So, now how many so suppose you have this model how many bits does the verifier need to query an assignment. So, what is the length of an assignment? It is n bits, suppose if you have a formula on n variables it is an n bit string.

So, you only need a log n bit, you only need log n bits to make your queries to denote the index of that that is what I am saying that the random access model basically enables to vary something

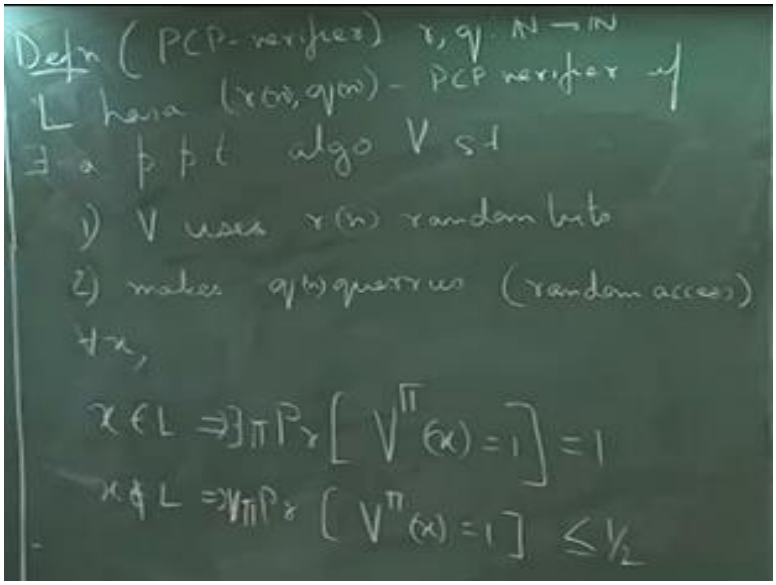
which is by using space that is much smaller than the actual length of the proof. Here the, how long the query can be is not bounded but what is bounded is that it is a probabilistic polynomial time algorithm, more importantly it is a polynomial time algorithm.

So, what can actually happen is that you can have proofs in this model which are exponentially long. So, we will see so we will actually look at an example of a problem where there is an exponentially long proof with a PCP verifier, but let us look at the formal definition. See again, let me go back what is crucial, I mean the resource that is very important in this case is how many queries you are making.

So, that is the resource that is very crucial to the verifier it is does not want to waste that resource. So, here very carefully select is a queries and also of course this is also another resource how many random bits is using. So, one notation that we follow is we will use the random variable V pi X. So, V pi X is a random variable that denotes that takes values either 0 or 1. So, V pi X is 1; if the verifier accepts and or if the verifier outputs 1 and it is 0 otherwise.

And, it is a random variable because the value of V pi X will depend on the random string of the verifier, so basically what random bits the verifier chooses will determine what this value is going to be.

(Refer Slide Time: 28:56)



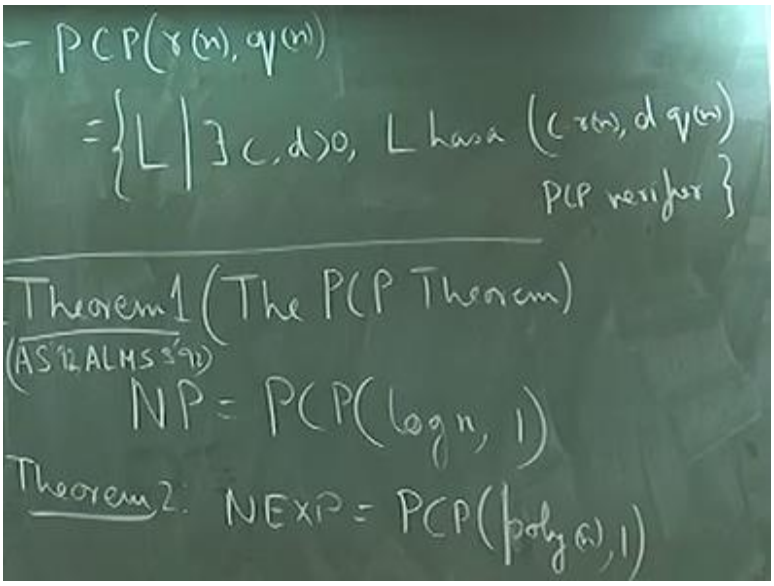
So, let us look at the definition of what PCP verifier is? So, let r and q be functions from natural numbers to natural numbers and so we say that the language L has a rn, qn PCP verifier expression if there exists a probabilistic polynomial time algorithm V such that so V have the following properties V uses rn random bits and makes qn many queries. Such that for all x there exist a proof pi.

That is x belongs to L then the probability that V with pi as it is certificate of x and output 1 is equal to 1. And, if x does not belong to L then the probability that leads pi of x of outputs 1 is less than or equal to half. So, basically a language has said to have a rn, qn, PCP verifier if there is some proof there is some prover, verifier where given an x the prover produces pi and the verifier firstly is a probabilistic polynomial time algorithm.

It uses at most r n random bit and it makes q n queries. Again, as I said that these queries are done by this random access such that if x belongs to the language then the probability that the verifier over his choice of random strings will accept x is equal to 1 and if x does not belong to the language then no matter what proof the prover produces, so just sorry for this but let me just change this a little bit because this is not exactly same what I wanted to say.

So, for all x if x belongs to the language then there exists a pi such that the probability of this happening is 1 and if x does not belong to the language then for all pi no matter what proof the prover produces this probability is at most half.

(Refer Slide Time: 33:47)



And, now so we define this set PCP of rn, qn as the set of all languages such that there exists some constants there exists some c, d greater than 0 such that L has a c dot rn d dot qn PCP verifier. So, the c and d are just to replacing the big O notation. So, PCP r n, q n is basically the set of all languages for which there exists some c rn, d qn PCP verifier, where this is what a PCP verifier is.

So, this is the definition of this class and the famous PCP theorem; you call this theorem 1, is also known as the PCP theorem because so basically after the PCP theorem was discovered I mean many variants of this theorem came along, there are many other PCP theorem but since this was the first one and this was in some sense the most important one this is known as the PCP theorem. So, what this says is NP is equal to PCP log n, 1.

So, in other words you are using only log n many random bits order log n many and making just some constant queries to your proof, this was very surprising and not only that it is an equivalent condition I mean it exactly characterizes NP. So, this direction is easy to show I mean why PCP log n, 1 content in NP is very easy to show we will see that but the it is the other direction which is difficult and even showing one part of the other direction is quite easy.

So, you always suppose I take a language in NP and so let I take SAT for example, then I can always say that my prover will produce as it is certificate a satisfying assignment, so no matter

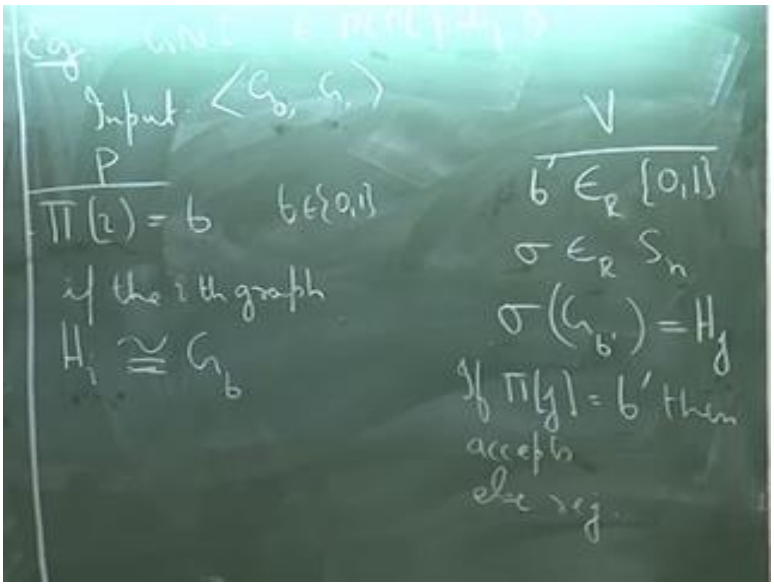
which constant bits of that satisfying assignment you query, the verifier will always accept. So, does not matter which bits it queries but it is this thing that is very difficult that if you have an instance which does not belong to the language.

It will reject with the probability at least half for any certificate. So, maybe I should also say the author so there was an earlier paper prior to some basically this is credited to these two authors so Arora and Safra so Arora is the same guy who wrote text book that you are following and there is another subsequent paper by a Arora, Lund, Motwani, Madhu Sudan and Szegedy, that was also 92 and Motwani is actually an alumni of this department. So, that is a good thing.

So, basically, so these two guys meet some progress in this theorem I mean, they showed some intermediate result and it was in the subsequent paper where this theorem actually appeared. So, there is also another variant of this theorem which appeared soon after that I just called that m 2 that is the class NEXP is equal to PCP of using polynomially many random bits and constant number of queries.

This was not very easy I mean, this was not just an immediate extension of this but the proof was along similar lines, some modification is you so any questions about the model?

(Refer Slide Time: 39:45)



So, let us look at an example so what we will show is that a graph known isomorphism problem

has a PCP verifier that uses polynomially many random bits and one query. So, in the second half of this course this graph known isomorphism problem is become quite popular this occurring in several contexts but anyway, so, what do we know about the graph known isomorphism problem? It is in am and it is not minutes, it is not known to be in NP.

It is of course in co NP because if it were to be in NP then by the PCP theorem it would have implied that it has a log n, one PCP verifier but that is not the case. So, let us see why this is the case; so suppose you are given as input two graphs G 0 and G 1 this is the input, what is the prover does this? So, let us again assume that both these graphs are on n vertices because of the number of vertices are not the same then one can immediately accept they cannot be isomorphic.

So, let us assume there over the same they have the same number of vertices so what the prover does is, he constructs a certificate pi that has length equal to the number of graphs that is possible on n vertices. So, one way to look at that is so suppose if you have a graph on n vertices, what is the size of the adjacency matrix? It is an n by n matrix, so it has n square bits. So, what you can do is that you can refer the ith bit of the proof to the graph, which is represented by the ith adjacency matrix.

I mean, I can always look at a lexicographical ordering of all n cross n matrices over 0 ones and once I have a lexicographical ordering I can talk about the ith matrix and that matrix corresponds to some graph pi i is equal to b if the, ith graph, let us call it h of i is isomorphic to G b. So, if the ith graph is isomorphic to 0 then I said pi of i to be 0 and otherwise I said pi of i to be 1. So, I am defining what my proof is going to be, so I am defining what I mean, what is the optimal choice of the prover, so I want to define a PCP I mean I want to give a PCP verifier.

In other words, it should have the property that for every x in L there will exist a pi such that the verifier always accepts and if I have an instance which is not in the language then no matter what the proof is the verifier will always reject. So, I am just looking at the case when let us say I have an instance which is in the language that is these are the two graphs are not isomorphic. What is an optimal choice of pi when any other choice of pi I am claiming will be a bad choice but that will come to later on.

So, the ith bit of pi, so pi of i denotes the ith bit of pi, so this bit is equal to b, so b belongs to the set 0, 1. If you look at the ith graph, as I said that you can always consider the ith graph. If the ith graph, h of i is isomorphic to G of b absolutely it can be that ith graph is not isomorphic to either of them then you set it to be any value. So, let us say without loss of generality, you always set it to 0.

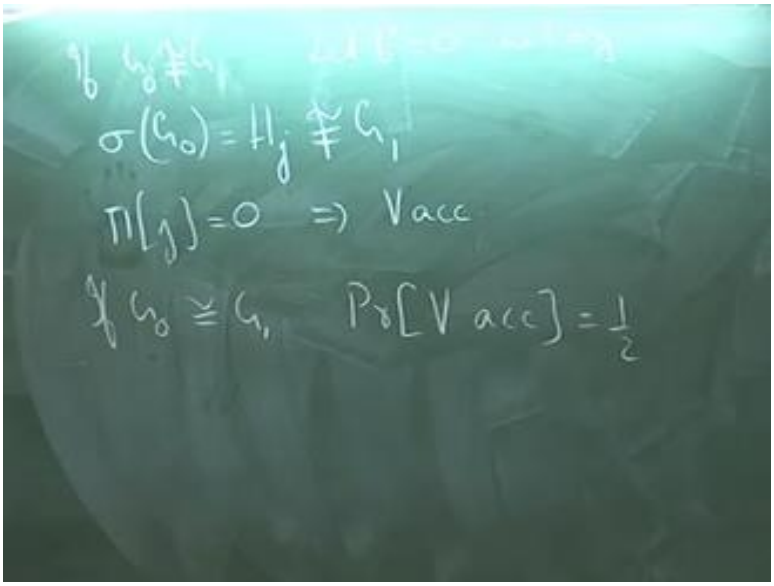
So, what the verifier will do is the following; so given this input G 0 g one first it picks, let me not use the same b let us say b prime takes a bit b prime randomly from 0, 1; so note that it can use polynomially bits. So, first it takes a bit and then it picks a random permutation, so it picks a permutation sigma randomly from the set of all permutation of 1 to n. So, again we call that from what we had done earlier in the case of I P.

S n is the set of all permutations of 1 to n takes some random permutation for this it will basically need a many, many bits and then what it does is, it applies the permutation on this graph G b. Looks at what sigma G of b prime is and it outputs so if there is the proof pi using j. So, basically try it writes down this j on it is address tape and it looks at the jth bit of pi. So, if pi j is equal to b prime then it accepts else rejects.

So, let us again understand carefully what is happening here? It picks a bit b prime at random, so basically b prime corresponds to which graph it will pick. So, let us say b prime without loss of generality is zero, so it looks at the graph G 0, it again picks a random permutation sigma and it permutes the vertices of G 0 using sigma. So, that will give it some adjacency matrix. So, it queries what is the bit corresponding to that adjacency matrix?

It queries that and then depending on what that bit is, is that bit is equal to b prime then it accepts otherwise rejects. So, let us see why this correct this is the algorithm clear?

(Refer Slide Time: 47:44)



Suppose, if G 1 is not isomorphic to G 2 and what happens is that, let us say that b prime is G 0, or sorry b prime is 0 without loss of generality 0 and 1. So, let b prime be 0 without loss of generality, so then what is sigma? G 0. So, sigma G 0 is sum H j and what do we know about H j, H j is of course isomorphic to G 0 because it is a permutation I am just renaming the vertices but what is the relation between H j and G 1? They are non isomorphic. H j is not isomorphic to G 1.

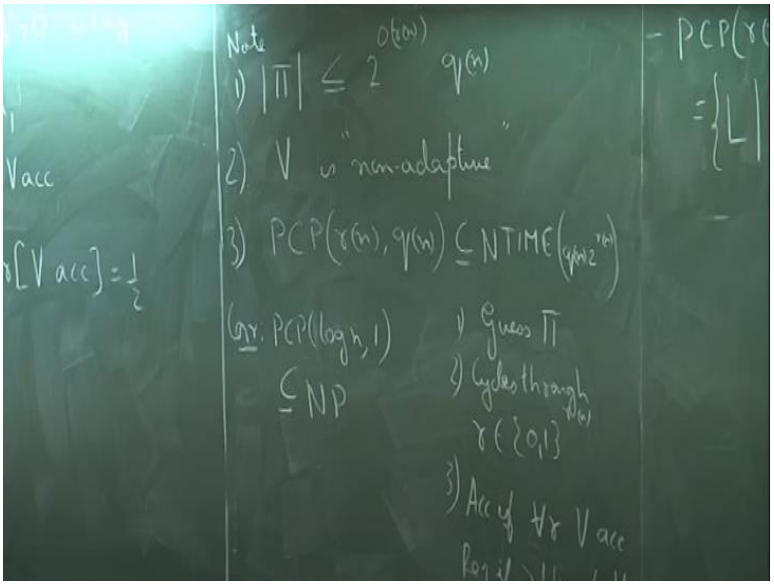
But now, what it does is it looks at the jth bit of pi? So, what will the jth bit of pi b? The jth bit of pi in this case will have to be 0 because the jth bit corresponds to this graph, H j and H j is isomorphic to G 0 but it is not isomorphic to G 1, so the jth bit does not have any other choice but the construction it has to be 0, so it is 0 and therefore this is 0 and therefore V accept it will always accept.

And, if these two graphs are isomorphic then what happens? Again there I mean, they are basically the prover stuck I mean the prover in it is jth bit can put a 0 or 1, so I mean, it does not matter what he puts but the thing is that with probability at most half because initially the verifier is doing a coin toss to decide which of the two graphs it will pick there is a probability exactly half that it will pick the graph, which is not the same as that bit.

So, these two graphs are isomorphic the probability that V accepts basically exactly half, because

H of j in that case will be isomorphic to both the graphs, I mean does not matter what permutation if they picks always picks random permutation, but since the bit b prime since, the bit b prime was picked random depending on what that b prime is we will either accept or reject the probability exactly half, this is the correct protocol. So, is this clear to everybody? A few observations about this PCP verifier;

(Refer Slide Time: 51:12)



So, firstly so suppose if you have a verifier with which uses r n random bits and q queries, what should be the maximum length of pi? So, ideally pi can be anything but I mean how much of that pi can the verifier actually access? So, let us let us just break that problem talk, suppose the verifier suppose qn is equal to 1, in a sense that suppose the verifier can make only one query. So, he has access to rn many random bits what is the length of the proof?

That is actually of any use to the verifier with only one query. Suppose qn is 1, in the sense that I mean 1 in the sense not 1 according to this definition but just 1 query. So, basically if you look at I mean, how many random strings can you have of length rn. You will have 2 to the power rn random strings and because of this because we have this d sitting here that is just assume it is 2 raise to order of rn or some d rn.

And for each of those different random strings, it can query a different bit of pi potentially. If the prover has only one, I mean if the verifier has only one query allotted to him I mean, the length

of proof that is of any used to the verifier is 2 raise to order of rn. So, now suppose if he has qn many queries that he can make it will be qn times 2 raise to the power rn, correct there is no size of the query.

But a size can be basically so that is what I said the size of the query can be anything I mean as long as it is accessible by a polynomial time machine. So, ideally the size of the query can be polynomial. Now, it will not be affected because we suppose, so I see what you are thinking a think of it this way the prover always knows what is the verifier strategy so the prover has access to the algorithm V.

So, in other words, suppose if the verifier writes down a very large address, although that address refers to just one bit, the prover is part enough to just keep that one particular bit. So, let us say the verifier writes down the address 1298. The prover will basically I mean, he knows the difference. Suppose the verifier is a deterministic machine who makes just one query and he is saying that I will query the cell of the verifier that has this particular address.

So, what the prover will do is that he will just keep that one particular bit in his proof. It does not need to keep the other bits. Every time both the prover and the verifier gets an input that is the only bit that will be of any use. Since it is a randomized machine then since the verifier is a probabilistic machine based on different choices of random strings the verifier can make different queries.

That is why he needs the proof to have length at least 2 to the power r n and since the number of queries allowed is q that has to be q times 2 to the power rn. So, actually his point is a very good point I mean, it is a good question. Because I mean ideally the verifier has polynomial time. So, nobody is stopping the verifier from writing down an address which has polynomial length and if the verifier is writing down an address, which is polynomial length that can basically, query a proof that has length exponential.

But the point is that in that case the prover will only keep that one particular bit in its proof. The rest of the proof is of no use. No, I mean in which case here constantly many times because of

this c here, 1. So, in each query it gets 1 bit. In queries for some address and whatever is the bit present at that address of the proof it gets that bit. So, the first point is that effective length of the proof.

So, by effective what I mean, whatever is actually necessary for the verifiers algorithm to proceed is bounded by two to the power order of r n times q of n. Although the proof as I said is ideally unbounded but this is what is of any importance. The second point is that the verifiers algorithm that is this algorithm V is non-adaptive. So, what do we mean by non-adaptive? Basically earlier queries precisely.

So, basically when the verifier goes on making queries it does not depend on what the earlier queries were. So, the only variables on which a query will depend is what the input is and what the random string is. So, this is basically what I mean, although adaptive verifiers are considered in the literature, but we just assume that our verifier is non adaptive. And there was another point.

So, what can we say about the PCP of rn, qn in terms of a non-deterministic time bounded machine? I want to simulate a PCP verifier using a non-deterministic machine. How much time do you think it will take? So, let me give the answer maybe then we can see why that is true. So, the amount of time that basically non-deterministic machines will take is basically the length of pi.

In other words, that is q n times 2 to the power let me just skip the whole. I will just write it as r n. So, why is this that case? Any ideas? Everything is not done. So, what do you do? So, I want to construct a non-deterministic machine that is allowed to use so much time. But it has to simulate a language in this class. What you said was correct partly. I mean initially what it does is it will guess the proof. It will guess this certificate pi. Then what?

If it guesses the random bits, suppose it guesses the random bits then it can deterministically go to the random position and then based on. So, guessing the random bits actually will not work. Because there can be one random string. Let us look at this strategy. So, the strategy is basically

so what it does is it first guesses, the string pi. And then what it does is that it cycles through all strings are in 01 to the power rn. Whatever is the length of the random string that the PCP verifier was using its cycles through all these random strings.

And for each random string it can then deterministically check whether the verifier is accepting or not. For each random string it will just look at that particular bit of pi and then it will see that whatever is the answer of the verifier. So, it keeps a track and then finally when does it accept? In other words for all random strings the verifier should accept. So, this machine accepts if for all r the verifier is accepting and it rejects if for at least half the fraction of r is rejecting.

I mean since we are looking at a language in this class we know that by definition it will be one of these two quantities. Either for all random strings the verifier will go ahead and accept or there will be at least half many random strings for which no matter what proof is being guessed a verifier will end up rejecting. So, this is a very simple way of simulating a PCP verifier using a non-deterministic machine.

Is this clear to everybody what we are doing? But the other thing that you are saying the reason that might not work is suppose if you guess a pi and then you guess a random string what happens? It is only looking at basically one case. There can be two to the power r n random strings. So, depending on if you take your decision based on just one string that might be wrong because that string can accept.

But maybe there are other half fraction of random strings which will reject which will lead to a wrong answer. What does this give us? What does this containment give us immediately? Let us see if we plug in those parameters suppose if you choose r n to be log n and q n to be constant what we get is an immediate corollary is PCP rn, qn is contained in n time of some polynomial. That is why I said that one direction of this proof is quite easy.

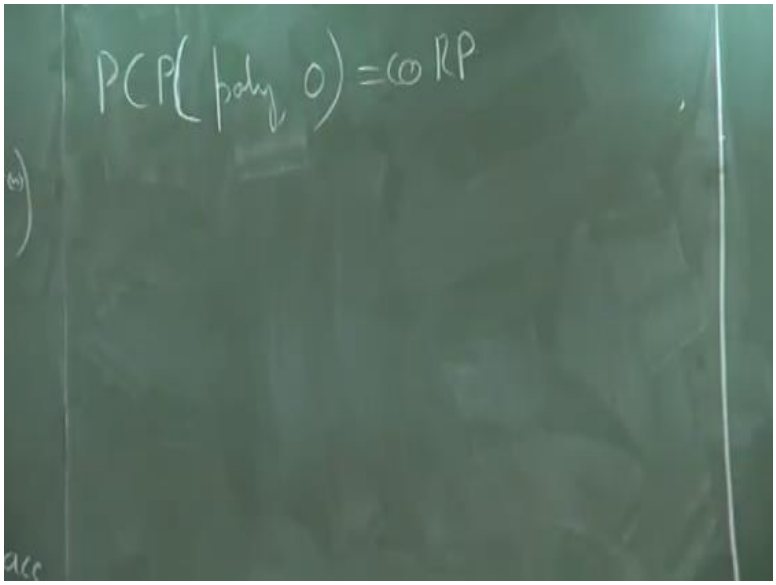
It is the other direction which is difficult. NP is in PCP log n, 1. It is not known for general qn and rn. Maybe so you are saying that for certain functions qn and rn the reverse containment is not true. Because again there are certain rn and qn for which this is true. Basically when rn is

equal to log n and it is equal to 1. So, I mean what would be a candidate choice for r n and qn in that case?

I do not know. I mean, maybe it is known but what my guess is that probably it is an open question. I mean showing something is not contained in something is a difficult problem anyway. So, I do not think that is. Because essentially what you have to show is that even if you pick let us say two good candidate functions for qn and r n. What do you have to show is that there is no PCP verifier possible with rn and qn as its parameters that will contain every language in this class.

So, what were the other things? The other thing is that this constant half is not against something which is very crucial.

(Refer Slide Time: 01:07:13)



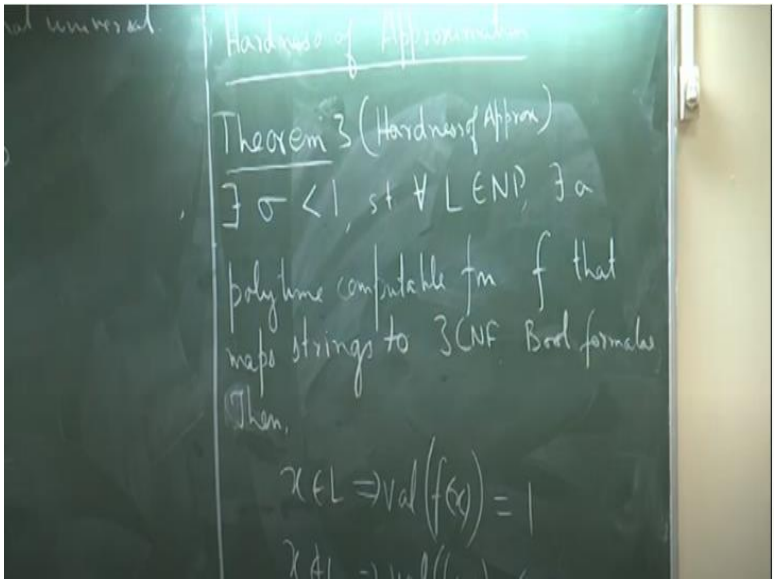
This is also known as the soundness constant. Half is let me call it universal. I mean the sense that you can always make suppose if I want to reduce the error if I want to bring it down to maybe one fourth or one tenth or something I can always repeat this experiment many times this algorithm and by the usual strategy bring that down. That is not very crucial. So, that at least ends one way of looking at the PCP theorem.

The other way as I said is the hardness of approximation. Can anyone say? Forget about the poly

part for now. What does that 0 mean? So, the certificate is redundant in some sense. So, BPP, RPP, ZPP. So, what is happening? So, it is a probabilistic algorithm polynomial time is more than P and what kind of error does it have? It has a one sided error. It is not BPP but is it RP? Think about what kind of error it is having. Yes, it is co-RP actually.

Because for x belonging to the language you are accepting with full probability, but for x not belonging to the language you are rejecting with probability at least half. So, it is basically co-RP. So, just think before you leave. So, what does the hardness of the approximation theorem say?

(Refer Slide Time: 01:09:48)



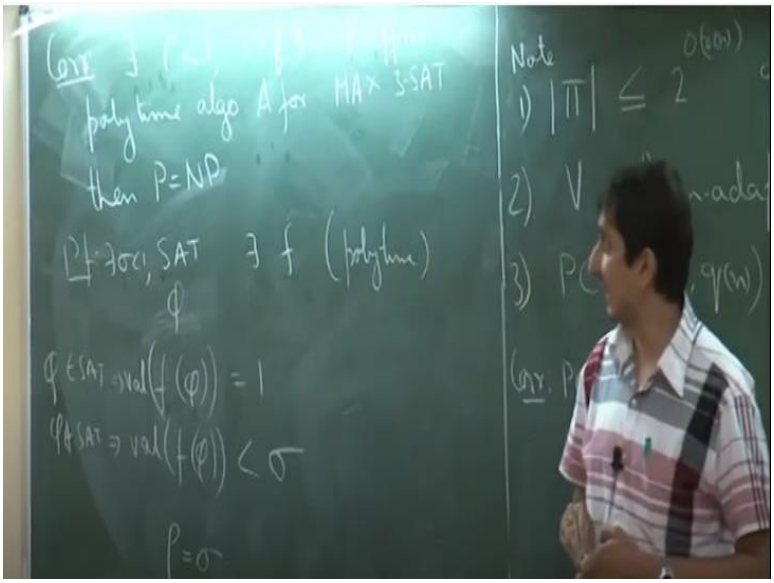
So, this is the third important theorem that we will see today. This is generally known as the hardness of approximation result. So, what this says is there exists a constant let me call it sigma. So, there exists some sigma less than 1 such that for all languages L in NP there exists a poly time computable function f that maps strings to let us say 3 CNF boolean formulae. Then for those x's which belong to the language;

If I look at f of x so by definition f of x is a Boolean formula. I can therefore look at val of f of x. So, recall what val was? What was val? Fraction of assignments which are satisfied. So, val of f of x is equal to one and for those x's that do not belong to the language val of f of x less than sigma. So, there exists some fixed constant such that any polynomial time function I mean, if

you take any mapping from else basically this f kind of corresponds to reducing it to the MAX 3SAT problem.

So, you take any language in NP and you reduce it to an instance of MAX 3SAT. What it is saying is that if x belongs to the language then this is always satisfiable. But if x does not belong to the language then you cannot get any more than a sigma fraction of satisfying assignments. So, this is the small signal. Sorry, satisfying clauses not assignments.

(Refer Slide Time: 01:13:12)



This immediately gives the following corollary that there exists a rho less than 1 such that there exists a rho approximation poly time algorithm A for MAX 3 SAT. I am sorry if there is such that if there exist such an algorithm then P is equal to NP so there exists some constant maybe rho is half or something like that of course. It has to be more than 7 by 8. So, there is some constant rho such that if there is a rho approximation polynomial time algorithm for the MAX 3 SAT problem.

Then P is equal to NP. So, in other words what this says is that you cannot get any approximation algorithm you do not expect to get any approximation algorithm that is better than rho. That is equal to rho. The moment you get an approximation algorithm that approximates it up to a factor of rho its equivalent to showing P is equal to NP. So, not only is deterministically solving an NP hard problem difficult, even approximating an NP hard problem is difficult upto

any constant.

So, let us see why this corollary follows from the theorem. How do you prove this? So, let us take a language in NP. In fact, let us take the SAT. So, I think it is enough if we show that SAT as a polynomial time algorithm. So, let us pick SAT. We are looking at 3 SAT and let us take an instance phi. So, how do we prove this? What does this theorem tell us? So, what the theorem says I mean, if I just imitate that theorem what it says is that for this problem SAT.

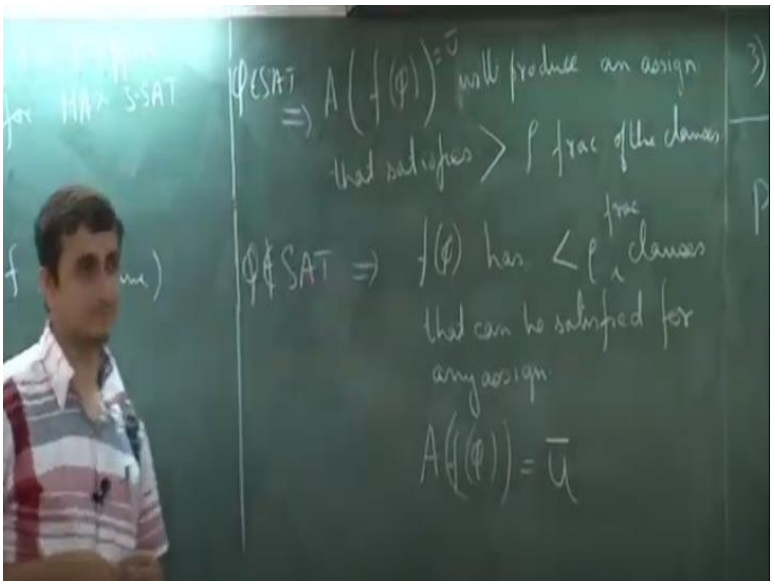
I mean there exists some constants sigma less than one then such that if I pick the problem SAT and I pick some instance of SAT there exist a function f such that if I look at f of phi. So, f of phi has either all its clauses satisfied or f of phi has less than sigma fraction getting satisfied depending on whether phi belongs to SAT or not. Sorry for moving so far left. So, basically if I pick my NP language to be SAT.

What it says is that there exists a constant such that there is a polynomial time computable function f. f is poly time computable, this is important such that if I construct this formula f of phi f of phi either has all its clauses that gets satisfied or at most sigma many. So, what does this tell us? What should the rho be for us? So, now what happens if I set my rho to be equal to sigma?

So, what do I get? So, what I get is if I set rho to be equal to sigma what I have is an sigma approximation algorithm for this problem. In other words what does the sigma approximation algorithm mean? It means that if an instance belongs to the language. No, I am sorry. So, what is sigma approximation algorithm for MAX 3 SAT means is that at least sigma fraction of the clauses are always satisfied.

Suppose if I start with phi which is in SAT what would f of phi be? Let me just replace this a little bit. Maybe I am just misleading here. So, we have f of phi which is an instance of MAX 3 SAT.

(Refer Slide Time: 01:19:25)



So, what is A? What can we say about the A f of phi? What is if I feed this input f of i to this algorithm A? Suppose if phi is satisfiable what can we say about A f of phi? So, it will come out with a satisfying assignment such that so I am setting rho to be equal to sigma. So, it will come out with an assignment that satisfies at least rho many clauses in f of phi. So, let us I mean once again so we have phi.

So, phi gives us f of phi and we know that in f of phi all the clauses are satisfied by some assignment. Since A is a polynomial time approximation algorithm by definition given a formula phi it always produces an assignment which satisfies at least rho many clauses. I mean rho fraction of the clauses. So, basically A f of phi will produce an assignment that satisfies more than rho fraction of the clauses. So, this is the case when phi belongs to SAT.

And both these things are computable polynomial time. So, f is also a polynomial time computable function so A is also a polynomial time computable function. So, basically whatever this assignment that I will get that is obtained in polynomial time. Now what happens in the other case when phi does not belong to SAT? What do we have from here? So, sigma and rho are the same. So, just think of them as the same.

So, what that means is that in f of phi how many clauses are satisfiable? So, by this result in f of phi at most rho many clauses are satisfied. So, f of i has less than rho clauses that can be satisfied

for any assignment. Not for some particular, for any assignment. So, now if you apply A to f of i, of course, it cannot give you better than that. It will always be less than that. Even if I assume that it gives a perfect approximation when applied to f of i it will still be less than rho fraction.

It will be some same rho, but I wrote sigma here. So, if I apply A f of i here the number of clauses the fraction of clauses that will be satisfied will be less than rho and greater than rho square because it is an approximation algorithm. So, then what we can do is then finally we can just count how many clauses are getting satisfied? So, if the count is greater than rho times m, we accept and if the count is less than rho times m, we reject.

So, just think and let me know if something is not clear. Actually nothing complicated is happening here. It is just plugging in the definitions. I mean, this is to recall the definitions of approximation algorithm and what is going on in this theorem that is all that is being used. There exists so this function is basically producing another Boolean formula which has this property. So, basically we know from this theorem that there exists a polynomial time computable function that produces 3CNF Boolean formulas where either all the clauses are satisfiable by some assignment or at most rho fraction of the clauses are satisfiable.

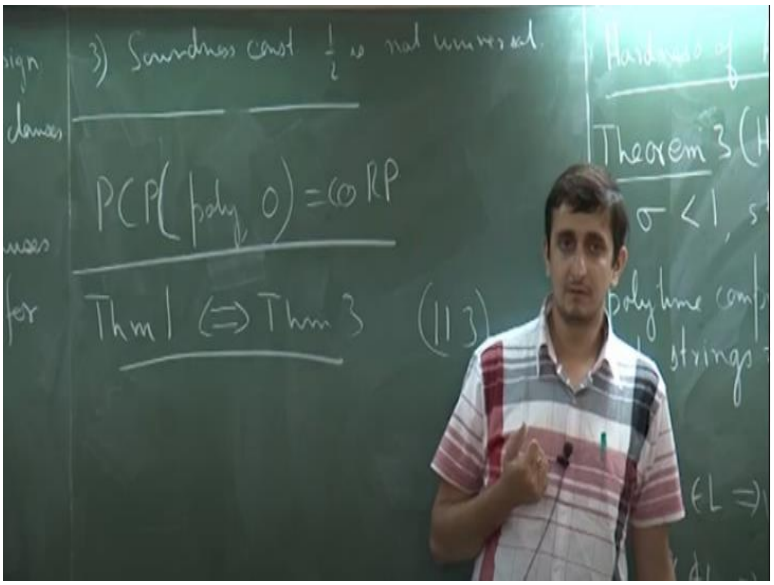
So, again forget about sigma. Let us say we have just one constant rho or at most rho fraction of clauses that are satisfiable. So, either all or rho fractions. So, there is a gap between them. Now that is the gap which we are exploiting. So, now if phi belongs to SAT, by that theorem, we know that f of phi in f of phi all the clauses get satisfied by some assignment. So, an approximation algorithm has to have the property that it satisfies at least rho fraction of them.

In fact strictly greater than rho fraction of them. On the other hand if it does not belong to SAT then this theorem has the property that f of phi has less than rho fraction of clauses that are satisfiable. So, even if I apply the best possible approximation algorithm that will still be less than rho. So, it basically lies between rho and rho square which is smaller than rho. So, ultimately what do we do after this is we just count how many clauses are getting satisfied and that is the algorithm.

So, that will show that SAT is in P. So, this corollary is what? No, it will not because I just do this is not quite correct. So, what I do is that then I look at A f of phi as in the earlier case and this produces some assignment u right as here. So, I just plug in this u in f of phi and then I count. I am not counting over all possible strings. So, look over it. I mean, it is very interesting. So, although this is the main hardness of approximation theorem this is what is mostly used in practice.

I mean again here we can change a MAX 3 SAT with any other problem in NP. Because from any problem in NP there will always be a reduction to SAT. So, that is always there.

(Refer Slide Time: 01:27:39)



So, I intended to show that theorem 1 and 3 are equivalent. So, in fact, this is I think section 11.3 in your textbook and this is also a very nice proof. I mean it is easy. It is not at all difficult. But it has many notations. So, just have to carefully work through the notations. But the nice thing about this proof is that it is connecting to very seemingly different areas. So, theorem 1 if you recall was basically that NP is equal to PCP log n, 1 and theorem 3 is talking about hardness of approximating NP hard problems.

So, these seemingly different areas or the seemingly different results are equivalent to one another. So, I think I will go and it is better not to do it now. But I do advise that you go and read this section 11.3. It is a nice book. So, what we will do tomorrow is we will just complete our

discussion on communication complexity and we will do a review of what we have seen so far in this course. I mean I plan to do a review. I do not know how much successful I will be. Thank you.