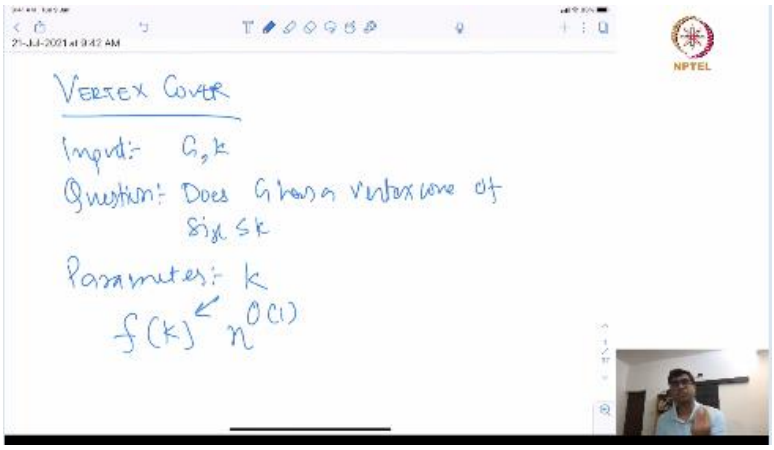
Parameterized Algorithms
Neeldhara Misra and Saket Saurabh
The Institute of Mathematical Sciences
Indian Institute of Technology – Ghandhinagar

Lecture - 03
Kernelization: High Degree Rule

(Refer Slide Time: 00:16)

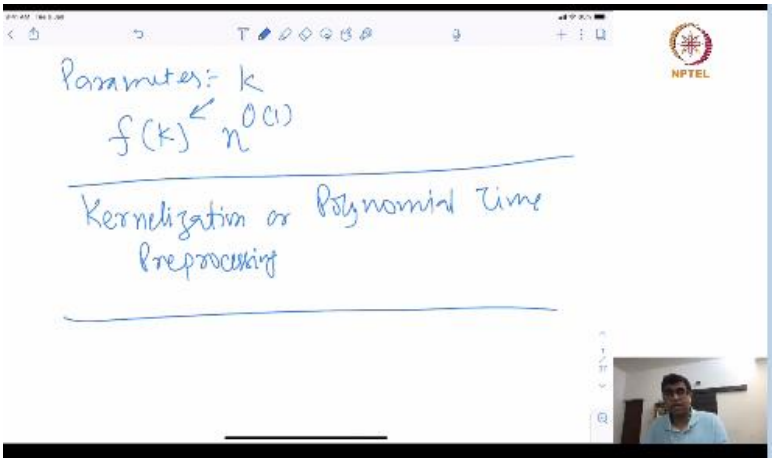


So, in the first couple of lectures of this course, we introduced this area of parameterized complexity. And so, basically the parameterized complexity if you recall the kinds of problems for example, in the vertex cover problem. We had input G, k and then we had a question. And the question was: does G has a vertex cover of size at most k and our parameter was.

So, basically all we are trying to do is to design an algorithm where the running time dependence is only on the parameter. And the other dependence on the input size is just polynomial that is all that we should care about from now onwards. So, your problem will have 3 pieces of information, an input, a question and the parameter. And the parameters should tell you that you need to design an algorithm such that the running time is the f of k and polynomial in the input size.

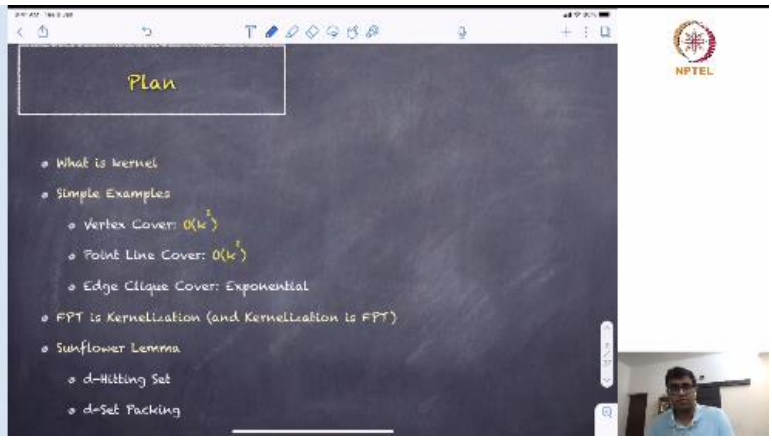
And this f of k could be any function which depends on k only. So, that is, with that in mind, let us try to learn our first set of tools; these set of tools are very basic and these are called kernelization or polynomial time pre processing.

(Refer Slide Time: 01:48)



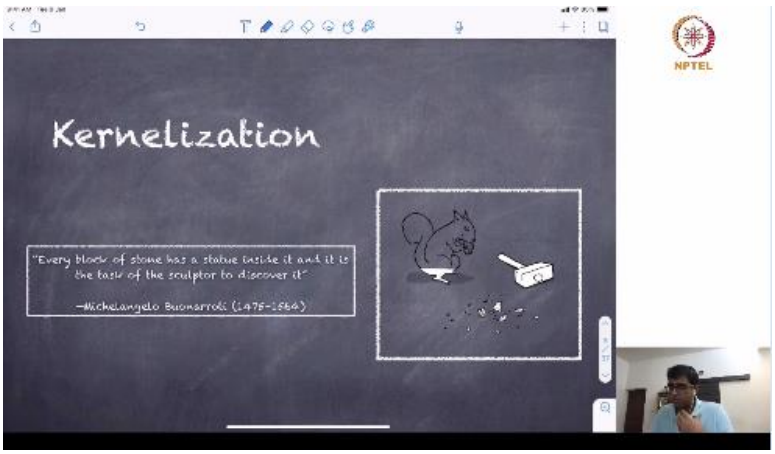
So, the first technique that we are going to learn is called kernelization and polynomial time pre processing or rather, this is also called, or polynomial time pre processing. So, it is a standalone technique and it has its own merits and we will see how to.

(Refer Slide Time: 02:21)



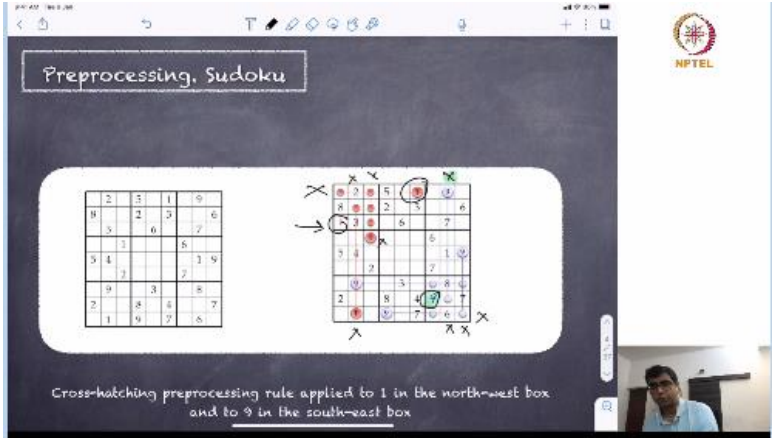
So, the plan of the lecture is very simple. The plan is that to define what kernel is, give you some simple examples and tell you how FPT and current fixed parameter tractability and kernelization are related and give you a couple of more examples in the next 2 lectures.

(Refer Slide Time: 02:21)



So, let us just get started. So, what is kernelization? So, basically, it is like in philosophically, you can think of this that every block of stone has a statue inside it and it is the task of the sculpture to discover it. So, there is a big problem, but not every aspect of the problem is hard. It is just our job to get rid of all the easy things and get to the hard, not of the difficult problems. So, with that philosophical quote, let us just get started.

(Refer Slide Time: 03:06)



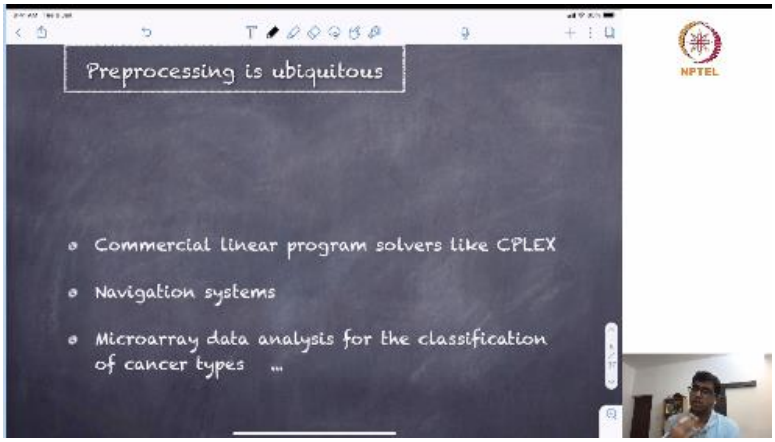
Here is a simple pre processing that you see quite naturally in day to day life, is that if you play Sudoku, where in Sudoku, if you recall that it is like a 3 cross 3 block Sudoku. So, in every 3 cross 3 boxes, you need to have numbers from 1 to 9, every row has to have a numbers from 1 to 9, every column has to have a numbers from 1 to 9 and suppose this is a partially filled Sudoku table.

Now, notice that here by looking at this, we can fill 1 by what is called cross hatching pre processing. Why? Because let us look at 1, so can, because 1 is present; because 1 is being present; 1 is present here that implies that 1 cannot be in this row. Now, notice that 1 is present here, it implies that 1 cannot be in this column. Now 1 is also present here. So, it implies 1 cannot be in this column. So, but we know that in this box, we need to fill 1.

So, by applying some simple rule, we are able to determine that actually, 1 needs to be here. Similarly, we can say about, let us say 9 here. Similarly, we can say about 9 here, if you notice, 9. Why? Because look at this box, presence of 9 here, presence of a 9 here, rules out 9 in this column. Presence of 9 here, rules out 9 in this row and presence of 9 in the last column rules out.

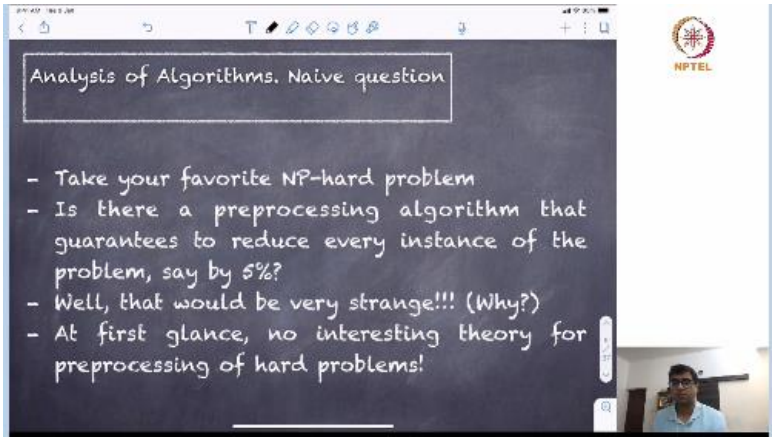
So, then if I look at this, then the only entry which is left to be filled with 9 is here. So, these are called cross processing rules. So, notice given an instance, you are able to determine some reduction rule, which tells you where to look for the new value. So, this kind of pre processing is quite; quite a big was in actually in computer science.

(Refer Slide Time: 05:21)



So, lots of you know, if you look at linear programming solvers like CPLEX and everything, they rely on such kind of simple rules to reduce your data ; navigation systems, microarray data analysis for classification of cancer types and so on and so forth. So, this kind of polynomial time pre processing to reduce input is quite well present in every aspect of computer science and science in general.

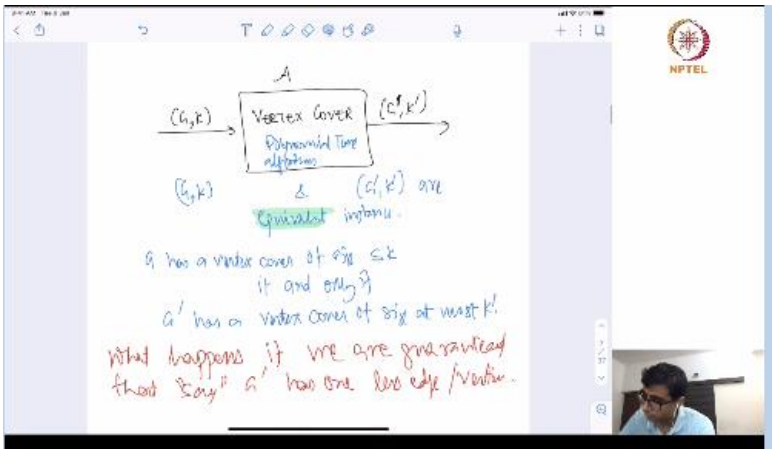
(Refer Slide Time: 05:51)



But what we are interested in is like in terms of analysis of algorithms. It is like, some nice question, so, now suppose I take my favourite NP hard problem. And I asked, is there a pre processing algorithm, which is a polynomial time algorithm that guarantees to reduce every instance of problem say by 5%? So, let us see what happens if we had such an algorithm.

So, what I am saying so, for example, look at example of our vertex cover. let us look at the example of a vertex cover problem.

(Refer Slide Time: 06:20)



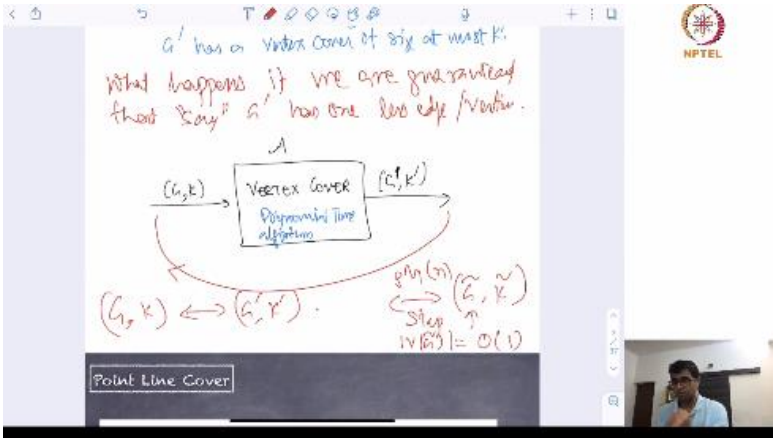
So, suppose I had an algorithm; here is an algorithm for say the black box, which is a vertex cover. For vertex cover, it takes an instance G, k. It outputs an instance G prime, k prime. And what is more a specification of this? It is a polynomial time algorithm. And what more G, k and G prime k prime or equivalent instance? So, let us, what is the meaning of this?

The meaning of this, if you started with G, k, which is s instance, then G prime k prime is also a s instance that is G has a vertex cover of size k imply G prime k parameter vertex cover of size. If you started with a no instance of vertex cover, G does not have a vertex cover of size k, then the reduced instance G prime k prime also does not have a vertex cover of size.

So, in other words, in mathematically you can say, equivalent can be well defined as follows G has a vertex cover of size at most k if and only if G prime has a vertex cover of size at most k. So, G has a vertex cover of size at most k if and only if G prime had a vertex cover of size at most sorry k prime. Now, what is the meaning of this? So, this is an amazing algorithm that polynomial time algorithm that takes an instance of a vertex cover outputs an instance of vertex cover without changing the answer, which is great.

Now, what happens if this algorithm reduces suppose, by, even by 1 bit, so, let us ask ourselves a simple question; let me make a small so that you can see everything in one go. What happens if we are guaranteed that say G prime has 1 less edge say, or vertices whatever. Suppose, if this is a guarantee this algorithm could do, then what we could do is the following.

(Refer Slide Time: 09:42)



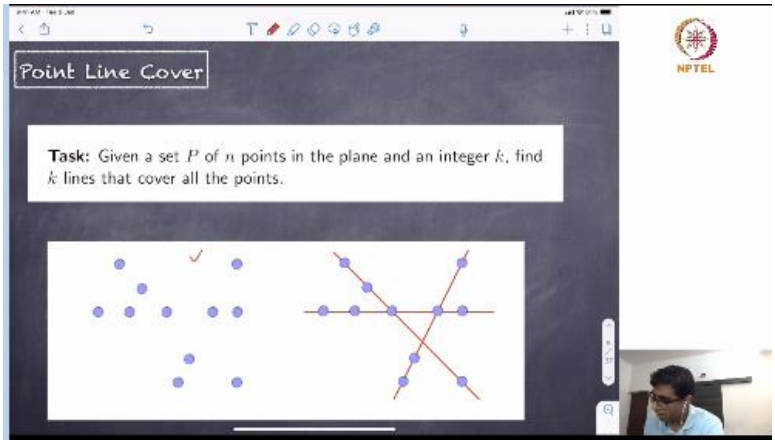
Then, we can feed this algorithm again and again, again and again and from G prime, G k; I will get an equivalent instance G prime k prime dot dot dot but after polly-steps, polly in say, n-steps, what will I get? I will get some G tilde k tilde. So, that the size of, is like constant. Now, once this is done, once I have applied this reduction rule or polynomial time algorithm, then if I apply polynomial time algorithm polynomially many time, I still a polynomial time algorithm.

But we started with a large instance but after polynomially many in what you call, application this algorithm, we are able to reduce the input size to constant and now, we can solve the problem in polynomial time that would tell us that will be a contradiction to our assumption that there is no poly time algorithm for NP hard problems. And that is a reason why, why it is, there is no algorithm that can guarantee to reduce every instance of the problem say by 5%.

In fact, even by 1 bit, it is not possible because you start with an instance will have polynomially many bits and if you are able to reduce even 1 bit in 1 invocation of your algorithm, then after a polynomially many application, this algorithm will have reduced to a constant size and then you can solve the problem optimally. Well, yeah, so that is what I meant by strange here.

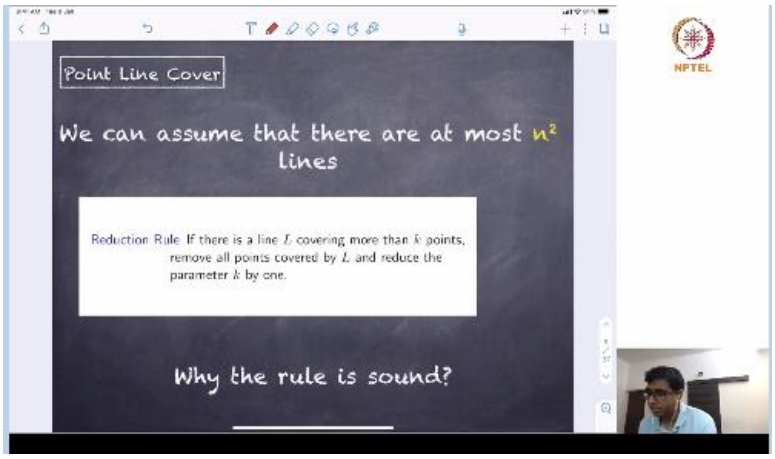
So, at first glance, there is no interesting theory for pre processing of hard problems. And we will see how the kind of algorithms, we would like to design this theory naturally comes into play.

(Refer Slide Time: 11:41)



So, let us take a new problem today. So, we have a new problem is what is called point line cover. So, you are given a set of set P of n points in the plane and integer k and I would like to know whether just k lines that covers all the points. So, look at the set of points given here, look at the set of points given here. And if you notice that I can take this 3 lines and that will cover all the points in the plane.

(Refer Slide Time: 12:13)

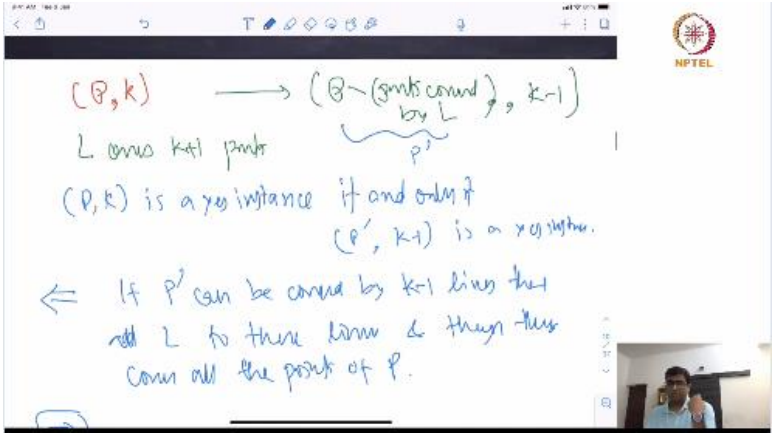


Now, so, we can also assume that there are at most n square lines, we can also assume that there are at most n square lines. Why? Because, notice that I mean, why do you care about a line just containing 1 point because, I mean and why do you care about lines, which does not touch anything. So, it is best that line should contain touch or intersect at least 1 point.

And then if there are n points, you know that like there is no point taking a line which contains 1 point because you could very well take a line which contains at least 2 points and between any 2 points you can draw a unique line. So, we can assume that the space of lines which we should consider is n square lines, they were given by lines passing through 2 points given into play.

So, we can assume that the so, my reduction rule is very simple, if they are line L, which covered more than k points, removing all points covered by L and reduce the parameter k by 1. So, let us try to understand what is this rule saying why the rule is sound So, the reduction rule is very simple. So, the reduction rule is this. If there is a line covering more than k points, which means so, we have to show so suppose our, my instance was sorry.

(Refer Slide Time: 13:46)

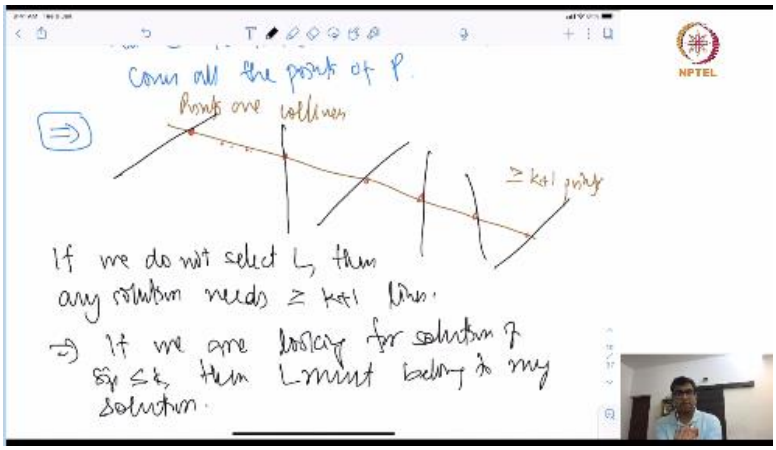


Suppose my instance was the set of points and line k. Now, what is my instance? So, I, what I will, suppose if there is a line L covers which is more than k points, I suppose, L covers k plus 1 points, I reduced instances, let us P – points covered by L and k. Let us try to show why this is sound. So, what is the meaning of sound? The sound as we have told before, it means say P, k is a yes instance if and only if, say this is let us call this the set of points covered is P prime, k is a yes instance. It is proof.

Forward direction is trivial or which direction is trivial? I think the reverse direction is much more trivial. So, why reverse direction is trivial is, because well if P prime can be covered by k – 1 lines, then add if P prime can be covered by k – 1, then add L to these lines and then they covered all the points of P. Why? Because the only points that P prime does not have belongs to this line L.

So, if you could cover all the points of P prime with k – 1 lines, then you add L right; then you add L to the set of the lines which covers the points of P prime. Now, the budget is k – 1; now, we have added 1 more line, so, budget has become k. So, you are only using k lines and able to cover all the points. Now, let us okay but what is the forward direction? This is an interesting direction. Why? What have we done? So, we are essentially say, so, look at this. So, to draw this, this draw some pictures.

(Refer Slide Time: 16:35)

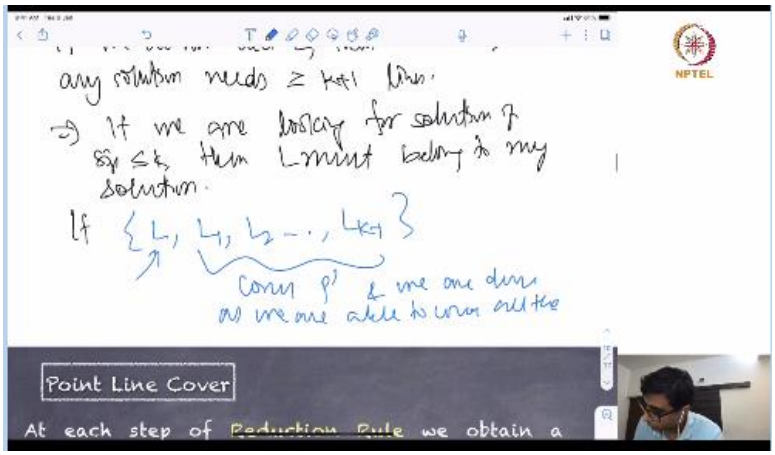


So, suppose these are my lines. So, these are my, some say dot dot dot and suppose this is my line which covers this, say greater equal to k plus 1 points. Now, ask ourselves if I do not pick this k plus 1 line there, these line these points are collinear. Now, you take any line, any other line, how many points can it cover? If I do not pick this line, any other line can cover at most 1 point

So, what does it implies? Implies if we do not select L, then any solution needs at least k plus 1 lines. So, if we are looking for which implies, if we are looking for a solution of size at most k, then L must belong to my solution. Then we are done. Because now, what happens? Then for the forward direction, we know that there is an optimum solution which contains the line L.

So, now, which implies that if you take out all the points that L covers, there are k – 1 lines that covered all the points which L does not cover.

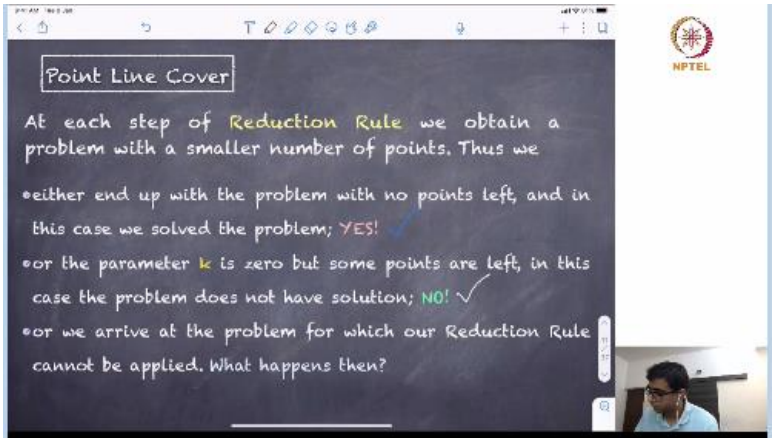
(Refer Slide Time: 18:30)



And that is precisely the points in P prime which implies that if which implies, then implies suppose if you had a solution, then L, L l, L 2 dot L k – 1, where L is the our; then this covers P prime and we are done as we are able to cover all the points of P prime with at most k – 1 lines So, we have shown that well, if you have an instance of this, then we can get an yes instance.

Then P prime is also an yes; P prime, k prime k – 1, it also yes instance and other way around So, what happens?

(Refer Slide Time: 19:20)

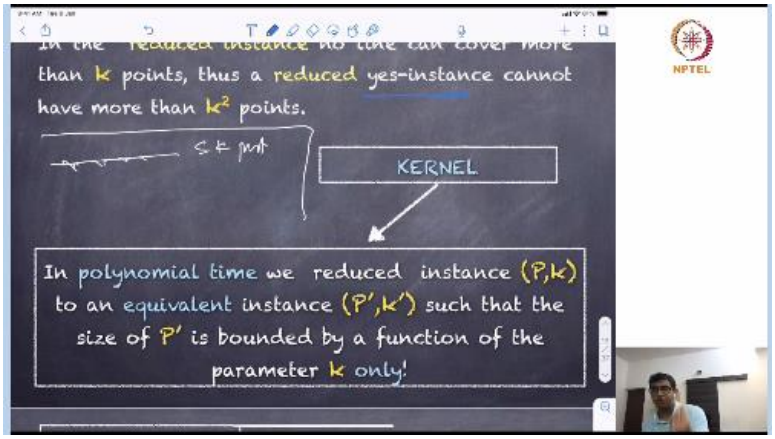


So, at each of the steps of the reduction tool, we obtain a problem with a smaller number of points. Thus, we either end up with a problem with no point left. So, if we apply this reduction tool exhaustively, what could happen? That every step we are taking, you are taking, we are reducing some number of points and parameter decreasing. So, we either end up with a problem with no points left and in this case, we have actually solved the problem and then we can say yes, this is a yes instance

Or the parameter cage you know, but still some points are left. So, like we are applying this reduction rules every time that look there is a line which covers so much points you; if you do not pick him, then your budget will be large. So, you apply this rule, but you have applied this rule exhaustively. And still there is some points left, then you know that there is no way that we can cover all these points with just k lines or we arrive at the problem for which our reduction rule cannot be applied.

What happened? Well, the point is that we are applying this reduction rule. It is possible this reduction who does not apply and if this reduction rule does not apply, then let us see what happens.

(Refer Slide Time: 20:40)



If the reduce instance no line can cover more than k points, then a reduced yes instance cannot have more than k square points. Correct. So, which implies, what does this imply? This implies very interesting things. So, when you know that like if every line covers only at most k points, then k line together can cover at most k square points. So, at this point of time, we can say, our reduction rule is if number of points.

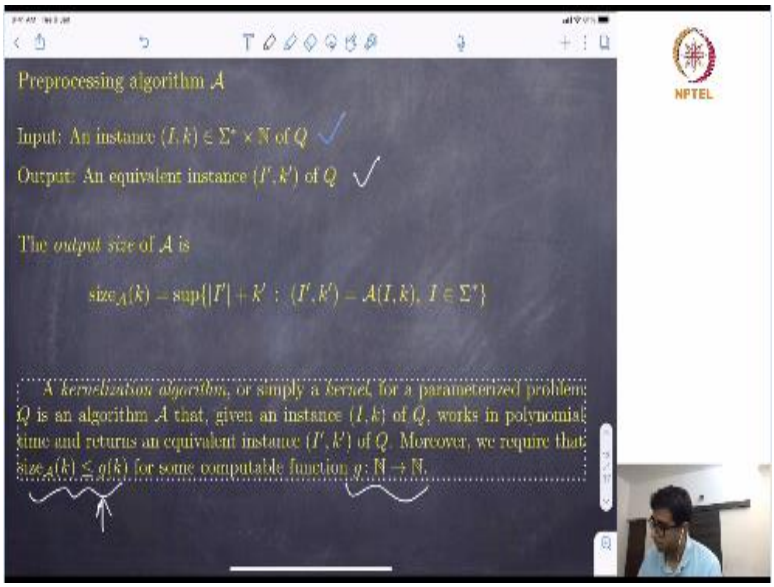
So, this is why we say look, yes instance, we are able to bound the size of the yes instance. If it is a yes instance meaning there are k lines that can cover all the points. So, in those instances, you know that each line covers at most k points. So, the total number of points that could be present is at most k square. So, here is the simple thing. What we will do?

So, if this is the case and if the number of points and more than k square, then it will say no, you say no instance. So, in polynomial time, we reduce instance P, k to an equivalent instance P prime on k prime such that the size of P prime is bounded by function of the parameter k only. So, this is the difference between parameterize complexity and classical complexity that in the classical complexity, we cannot say that every invocation of the polynomial time algorithm will reduce the instance size.

But here, we can say that look, if we cannot reduce the instance size, then I can at least guarantee you that the size of the instance is bounded by a function of k 1, your function of

the parameter k 1. So, parameters complexity or this field allows you to say a third thing, yes, no, size is bounded.

(Refer Slide Time: 22:49)

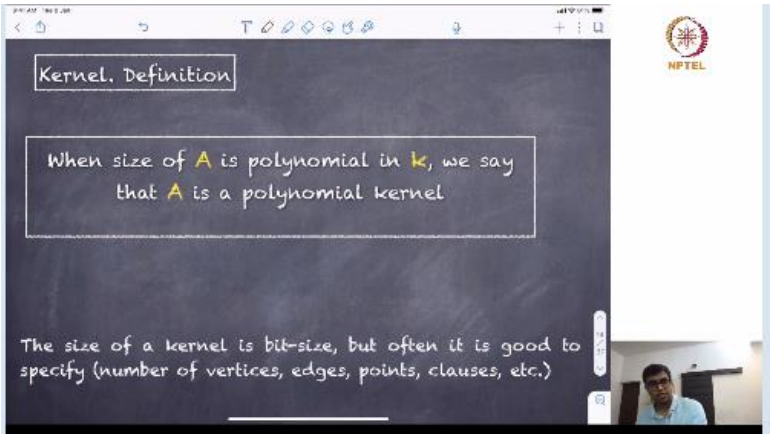


So, now let us define this notion of kernel formally. So, it is a pre processing algorithm. It takes an instance and instance i, k of a problem of Q and returns an equivalent instance of the problem and the output size of A is like, this is size of A is like maximum size you could have in terms of case. So, this is why it is defining formally in terms of supremum, but let us not care about them, because we will see we will be able to analyze the size of this by hand.

So, a kernelization algorithm are simply a kernel for a parametrized problem, Q is an algorithm A that given an instance I, k of Q works in polynomial time and returns an equivalent instance I prime k prime of Q. And we require that the size is bounded by some computable function G. And we will see why we need this.

But like, again computable function just means that there is not a tuning machine that given an integer n, it will work in polynomial time, it will work in time; at the end of this, it will write down G of n in the output that is it But we will be our functions will be like 2 power k poly k something of this nature. So, I will not worry about it. This is once you have understood the concept well enough, you can go and start putting these formality, this formal notions to make the theory, go through the mathematical rigor.

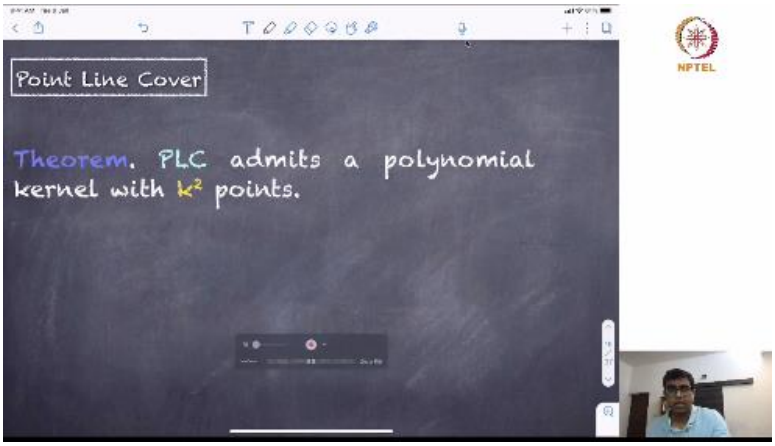
(Refer Slide Time: 24:38)



So, now, when the size of A polynomial in k, we say that A is a polynomial corner. So, for example, what was the size in point line cover? Well, we were able to say that the size of the kernel is like roughly k square points. So, we say that a admits a polynomial kernel. The size of kernel is generally the way the size of the kernel should mean is bit size. What is the number of bits?

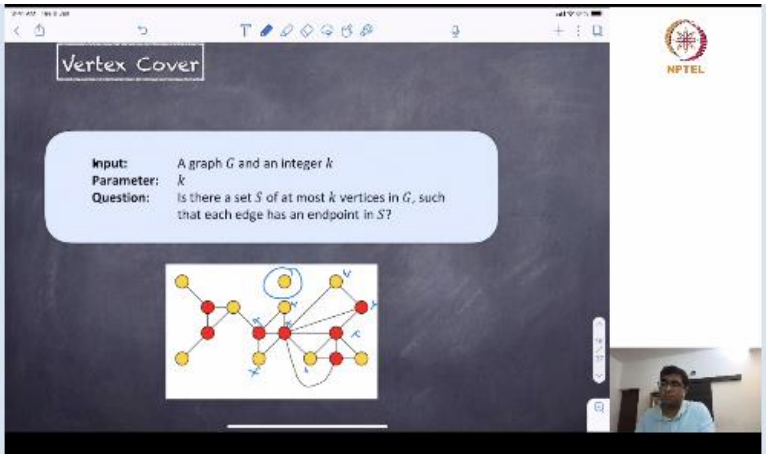
You need to represent, then output is 10 but often it is good to specify number of vertices edges, points, clauses etcetera that you are able to reduce in the smaller instance..

(Refer Slide Time: 25:24)



So, in that language, what did we prove? PLC admits a polynomial kernel with k square points. I am not saying k squared bits because each point could require a certain amount of representation based on how precisely we are represented. So, let us not get into those all we have been able to do is that point line cover admits a polynomial kernel with k square points. Just a minute okay. So, let us just continue

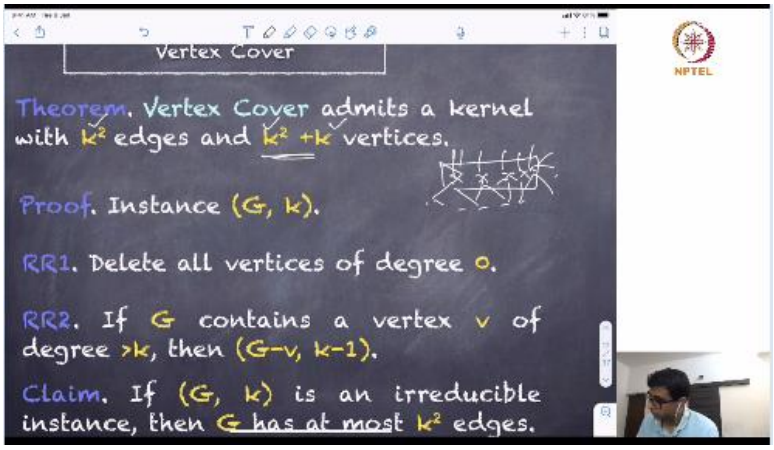
(Refer Slide Time: 26:06)



So, let me give you another example. This is our favourite problem a graph G and integer k. Is it a set S of at most k vertices in G so that each edge has an endpoint in S? Now, there are some simple reduction rules for this. Like for example, if you had this vertex, isolated vertex, you never need this isolated vertex because it is not covering any edge. Similarly, if a vertex covers k plus 1 edges, then if you do not pick him, so, for example, if I do not pick him this red vertex and how many vertex do I need in vertex cover?

1, 2, 3, 4, 5, 6, 7. All its neighbours must go into my solution. So, if I do not pick a vertex and there are like 7 neighbours, like all its neighbour must go to the vertex cover.

(Refer Slide Time: 27:01)



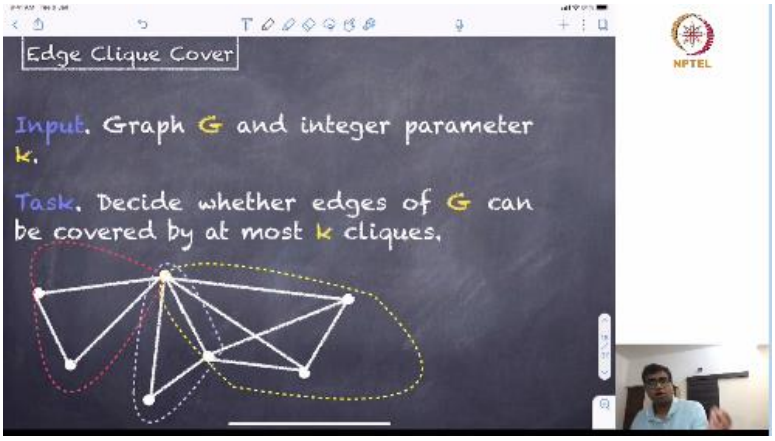
Because we will cover these edges which implies that if there is a vertex of degree is strictly greater than k, then this vertex must go into the solution. So, again the ended reduce instances you delete vertex V from the graph and you decrease the parameter by 1. You can prove that G, like this reduction also sound the way we proved for point line cover because you replace line with a vertex and the argument would go through

So and why if G, K irreducible instances, the G has at most k square edges, because look, if a vertex can cover only k edges, then k vertex together can only cover k square edges. So, if every vertex has degree at most k, you are able to get to that stage and if the number of edges are more than k square, you will immediately say no. So, what we are able to show because of that is that vertex cover admits a kernel with k square edges and k square plus k vertices.

Why k square plus k vertices? Because look, let me just say that we just some draw some picture for you. So, we have reached the stage that we have if there is a vertex cover of size k, they have degree at most k. So, the total number of vertices total number of edges they cover is k square that is correct, but total number of vertices could still v k square plus k because this could be stars on k guys. So, like this.

So, they cover k square edges. So, these guys so k square vertices from there and k for themselves. So, it can show just like point line cover that vertex cover admits kernel with k plus kernel with k square edges and k square plus k vertices.

(Refer Slide Time: 29:06)



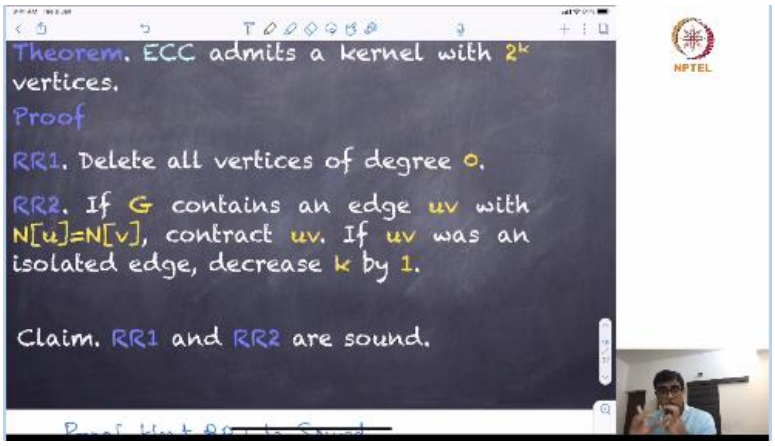
So, we will do a last example for this lecture, which is about edge clique cover. So, what is edge clique cover? So, input is a graph G integer parameter k and we have to decide whether edge of G can be covered by at most k cliques. So, I want to find k cliques. Look at any edges, it participates into one of the cliques. So, look at this example here, we have this blue clique, red clique and yellow clique and look at any edge, it must belong to one of the clique.

So, for example, this is your graph. So, you can just check that every edge belongs to some of the cliques. So, the question is: can we partition my vertex set into, not parties that is wrong word, can be fine k cliques in my graph such that every age participates into one of these

cliques. And we are not partition our vertex because look at the blue clique and yellow clique, they intersect, red clique and blue clique and yellow clique, they all intersect.

So, we are just, we just want to find k cliques. So that look at any edge, you can at least find one clique where this edge belongs to.

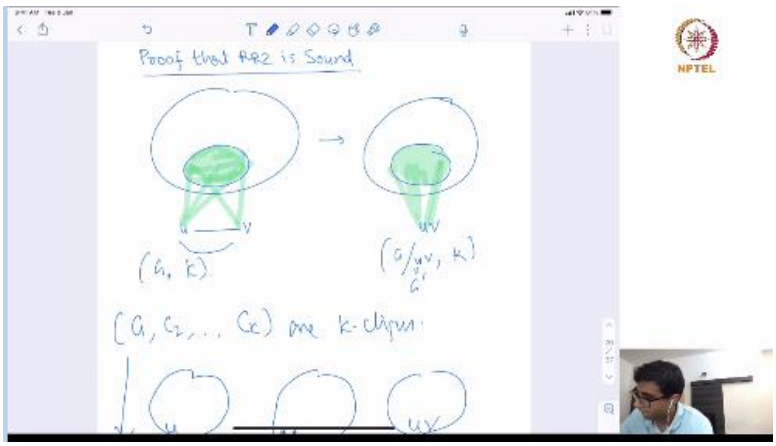
(Refer Slide Time: 30:26)



So, we will show that it is clique cover admits a kernel with 2 power k vertices. So here are reduction rule. Delete all vertices of degree 0. So, let us delete all vertices of degree 0 because they are not adjacent to any edges anyway. So, first of all, they cannot participate into any clique and even if they participate, so they cannot participate into any clique anyway or except the clique of 1 vertex, but clique of 1 vertex covers no edges. So that is redundant, so you can remove them.

But look at reduction rule 2. If G contains an edge uv with a close neighbourhood to fail, then we would like to contract u v. If uv was an isolated edge decrease k by 1. So, what is the meaning of this? Let us look at this.

(Refer Slide Time: 31:09)



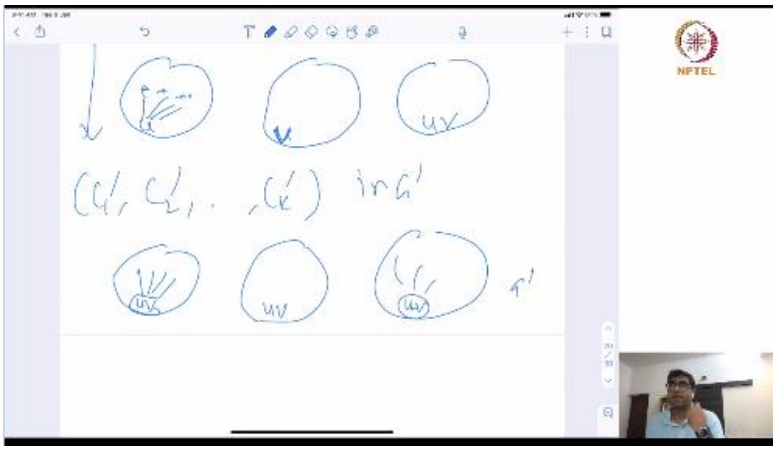
First, what it says is that it says that look at suppose, this is my graph. And here is an edge v u, who has an edge among themselves. And they had the same neighbourhood. u had the same neighbourhood, v had the same, which is this patch. Then I am saying that apply the following reduction rule. What are the following reduction rule? Let us make the same vertex uv and make them adjacent to like you just contract uv into a single vertex. And here it is.

So, our instance become G, k and say, hey, look at G u, v, k, which is contracted, let us call this. This is what. So, contraction means you identify u and v and make u make all the neighbours, like if a vertex was neighbour to u, u make a neighbour to uv that is it. And so like anyway, they have a common neighbourhood, so like does not matter. So, now, I want to say that well, this reduction rule is safe. What is the meaning of reduction rule is safe?

If so, look at the forward direction, suppose C 1, C 2, C k are k cliques. Now, what could happen? This clique could contain u, but maybe contain or this clique could contain v or this cliques could contain u v. What I am going to do? I am going to get a clique. From here, I am going to get a clique C 1 prime, C 2 prime, C k prime in G prime. How do I do this? For this clique u, I take okay, I am not going to do anything but I am just going to replace your name with u v.

I am not going to do anything with u, I am going to replace u like this with v. I am going to replace this with uv just naming and if there is an edge u v, you take contract this edge and make this. Now, what happens then? Why is it true?

(Refer Slide Time: 34:17)

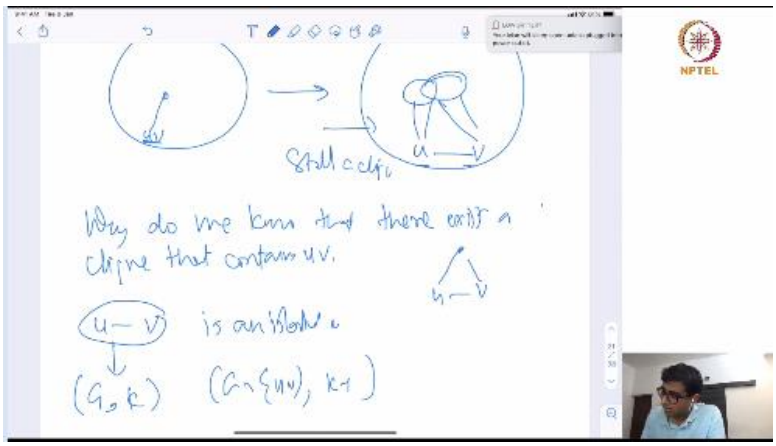


So, notice one thing right, look at all the edges, which you are adjacent to u. You know that v is also adjacent to same thing So, when I replace by u the vertex u v, it does not matter Now,

look at any edge, which is the only edge which is not present not the look at the edges which are, look at this, look at the graph G u v. Every edge which is present in graph G is also present in the graph G prime in the following sense.

Look, basically, if you had a copy of edge here, you know there are 2 copies of edge here. This is just keeping only 1 copy. So, whatever u covers, uv covers, so now replace u v, you can check for yourself that every edge is covered. Every edge of G prime is covered by this new k cliques where I look at the old k cliques and replace every vertex u with the new vertex uv and for the reverse direction, what will I do?

(Refer Slide Time: 34:17)



I said okay, fine. Suppose C 1 prime, C 2 prime, C k prime covers k edges, covers edges in G prime. So, now look at C 1 prime. If there is an edge u v, you uncontracted yourself, you now make u, v, put an edge and just leave it. Now what do you know? Whatever this u are adjacent to same v adjacent to. So, does not matter. This is still a clique. And you know that every edge here will be covered by either u or v.

Every edge which uv edge adjacent to, you know both u and v adjacent and so if I replace v u with this, then this is good. Now, this happens for every edge. So that is perfectly fine. Now, but the problem is: why do you know? Why do we know uv will belong together? Because if uv does not belong together, it means the only reason u will not belong together in some clique because or the, I mean that is not the right way of saying.

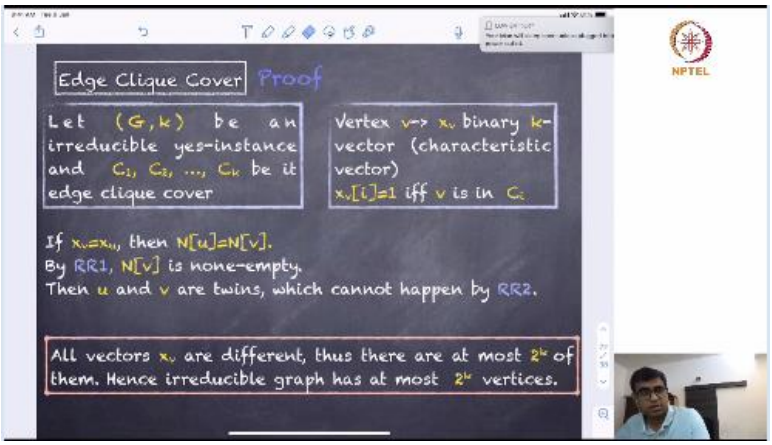
Why do we know that the clique that there exists a clique that contains uv right? And this is why we have a, because it could be that uv becomes an isolated vertex. This is why we have the following rule that if uv is an isolated edge, then our reduction rule is very simple. What

we did? Because we know that if uv is an isolated, they are not adjacent to any other edges, then there is no; the only way you can cover this clique is by taking this edge, then you just decrease delete this edge and you decrease the parameter by 1.

So, this is why we have a rule if such an edge is there, then you take G – uv edge and k – 1. This is why if you notice we have this rule. If uv is an isolated edge decrease k by 1 and so, if uv such an isolated do not exist, it means what is the meaning of this? uv also have some other neighbour, common neighbour. In that case uv also in G prime, uv has an adjacent to itself.

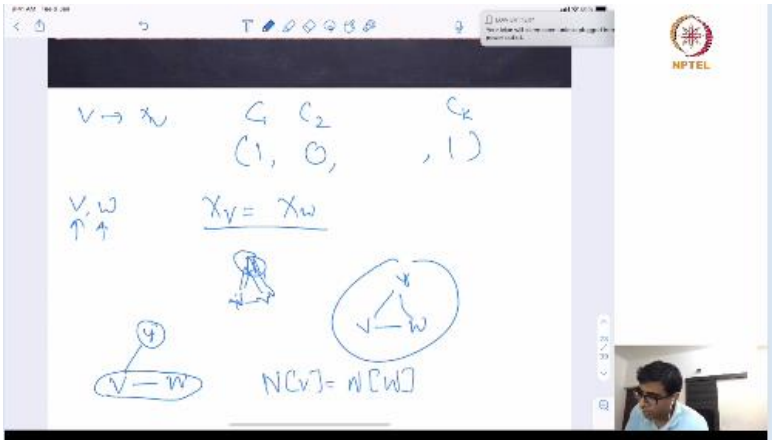
It means there will be a clique where uv will be present and then if uv will be present, then the edge uv will be covered also in G prime G. So, this nice contraction of placement preserved. So, we like reduction rules. We look at the graph, we analyze its structure and we are able to say and the proof is very simple.

(Refer Slide Time: 38:50)



So, that is what deduction rule. What happens? So, let G, k be an irreducible yes instance meaning you cannot apply any of this reduction rules and C 1 to C k be it yes clique cover And so what we do to apply to prove to prove that the reduction rules is sound or deduction applies is very simple. So, what we say that look for every vertex v, let us associate a binary vector x v. So, let us fix this cliques C 1, C 2, C k, let us fix this.

(Refer Slide Time: 39:28)

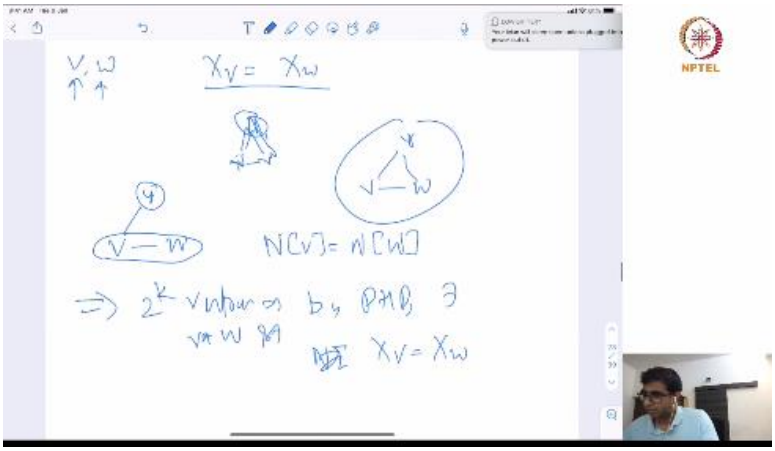


And I am going to write what is a binary vector for x v. Well, is v present in C 1, then I put 1; is V present in C 2, answer is no, then I put 0; is v present in C k, yes then, so then you get a k length vector which tells v is present in which are cliques. Now, let us ask yourself if for 2 vector v and w, 2 vertices v and w, xv is equal to x w, it means notice that v and w, they are present in the same cliques.

So, they see the same neighbours everywhere. So, you can show that if this happens xv equal to x w, then because they are presenting the same cliques everywhere, so, look at a neighbour which is a neighbour of, we will show, so, look at a neighbour of y. So, first of all v and w are present together in some clique, so, the edge there is a neighbour between v and w; they are neighboured.

Now, look at any edge y which any neighbour of y So, this edge v y will be covered somewhere in some clique. So, look at this edge But I know that w is also present there, which implies that w also neighbour to y which implies that neighbourhood of v is equal to neighbourhood of w.

(Refer Slide Time: 40:59)



So, if there are more than 2 power k vertices that implies by Pigeonhole principle that exist v and w such that xv equal to x w and if there are 2 vertices which is close, they will do the same, then our reduction rule 2 would apply if G contains an edge uv with N u equal to N v contract uv. So, we found 2 vertices uv whose is close neighbourhood the same which means not only they are open; not they have a neighbour among themselves.

And all their outside neighbours are also the same which implies that, then this kind of vertices are also called twins. So, all vectors xv are different, thus their utmost 2 power k of them. Hence, irreducible graph as at most 2 power k vertices. So, what is our kernel? So, we apply this kernel addition reduction rule. If the number of vertices is more than 2 power k, you immediately say no or you obtain a kernel. I think with this, we will stop this lecture. And we will come back again for the next lecture.