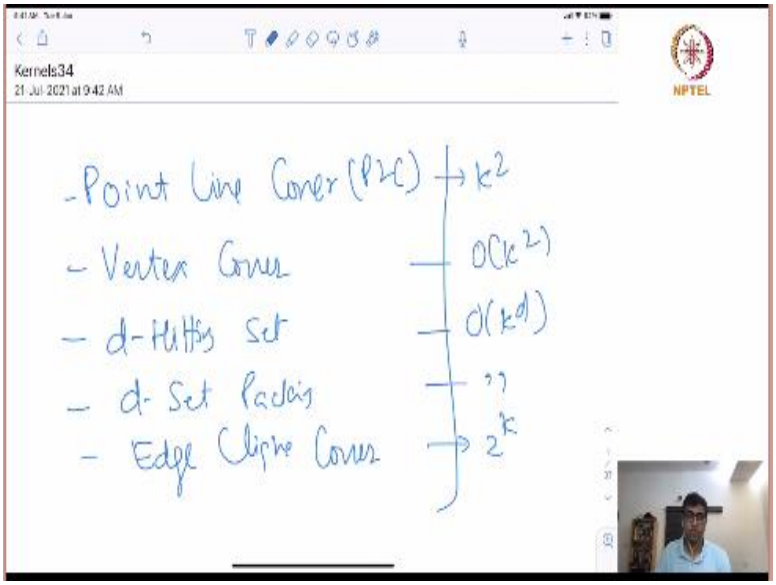
Parameterized Algorithms
Neeldhara Misra – Saket Saurabh
Department of Electrical Engineering
Indian Institute of Technology – Gandhinagar

Lecture - 05
Kernalization: Crown Reduciton

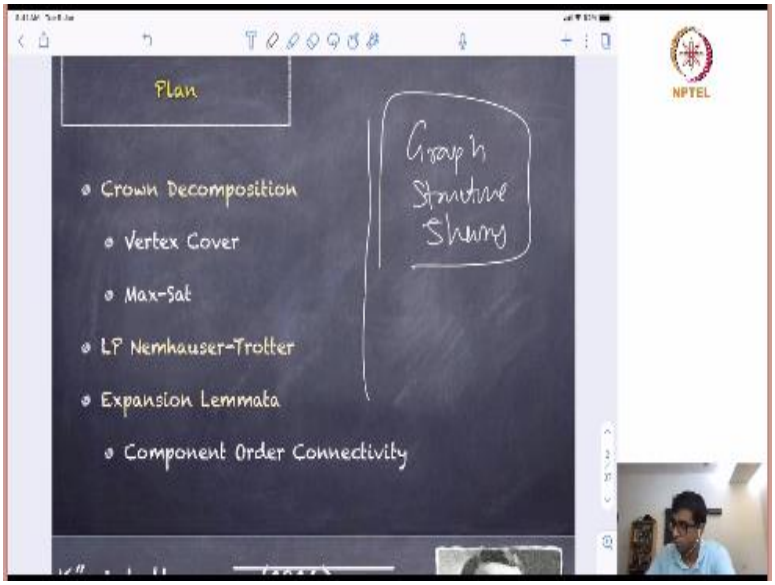
(Refer Slide Time: 00:15)



So, welcome back to the new lectures. So, in the previous lecture we saw Kernel for several problems like point line cover. It had k squared points for vertex cover we saw Kernel with order k squared vertices and edges for d fitting set we saw Kernel with k power d sets and elements. Similarly, for set packing and for the edge click cover we saw kernel 2 power k.

So, today we will see some more techniques largely rooted in graph structure theory and how that is employed to design kernels for so, how that is employed to design kernels for the problems of our interest.

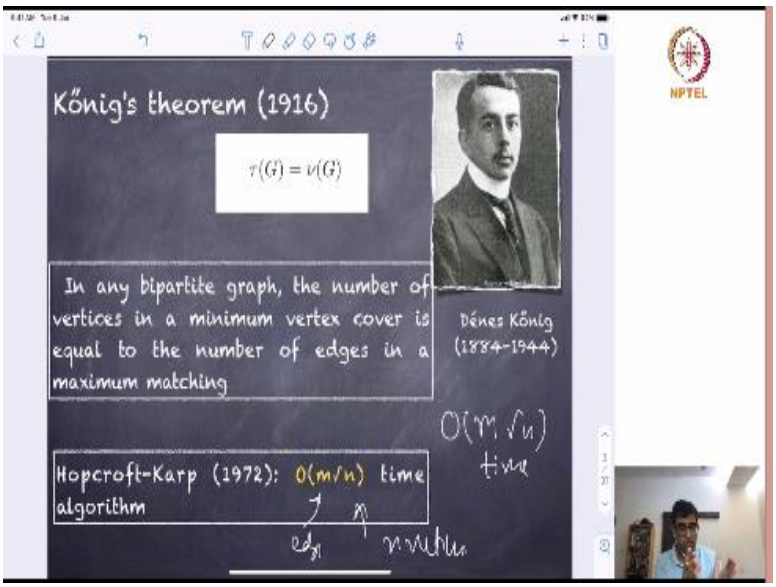
(Refer Slide Time: 00:58)



So, now the plan is to cover what is called, I will teach you what is called crown decomposition, which is literally rooted in matching theory and give you another application in Max SAT and then Nemhauser Trotter and Expansion Lemma. So, this part of the lecture should be considered as kernalization, which comes from structure theory graph structure theory.

Okay. You will see what I mean by that, and this will be covered in 2 lectures of 35 40 minutes each, hopefully, or maybe 1, we will see as we go forward. Okay.

(Refer Slide Time: 01:40)



So, this is a classical theorem from 1916 by Konig, who said that, in a bipartite graph, the number of vertices in a minimum vertex cover is equal to the number of edges in a maximum matching. And using an algorithm for maximum matching and goes back to Hopcroft and

Karp from 1972 order m route n time algorithm, okay. So, basically, if you are given a bipartite graph with m edges and n vertices.

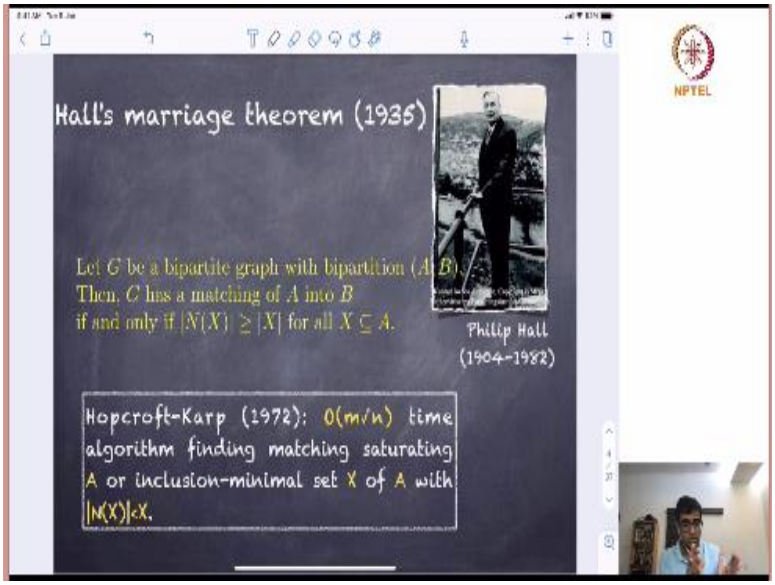
In order m route n time, we can check whether there exists a vertex, we can find a minimum vertex covered of a bipartite graph in polynomial time. And another property of this vertex cover is that, in fact, notice that up until now, we have been using the side of the maximum matching as a lower bound for the vertex covered. In bipartite graphs actually, it happens with equality.

So, in fact, the size of the minimum vertex cover is equal to the maximum matching. So, in some sense packing is equal to covering. So, vertex cover should could be thought of as a covering problem means you want to cover it all the like, we want to cover all the edges with the vertices. And packing is like packing of disjoint edges, pair wise disjoint edges. And in fact, this property holds not only for a bipartite graph.

There is another graph called Konig graph. Konig Egervary graph, which basically the definition is a family of graphs where the size of the maximum matching is equal to minimum vertex cover. And, yeah, and even those, and in these family of graph, you can find minimum vertex cover in polynomial time. But I am not going to prove Konig's theorem, It is a very classical theorem, you can find in any graph theory book of your interest.

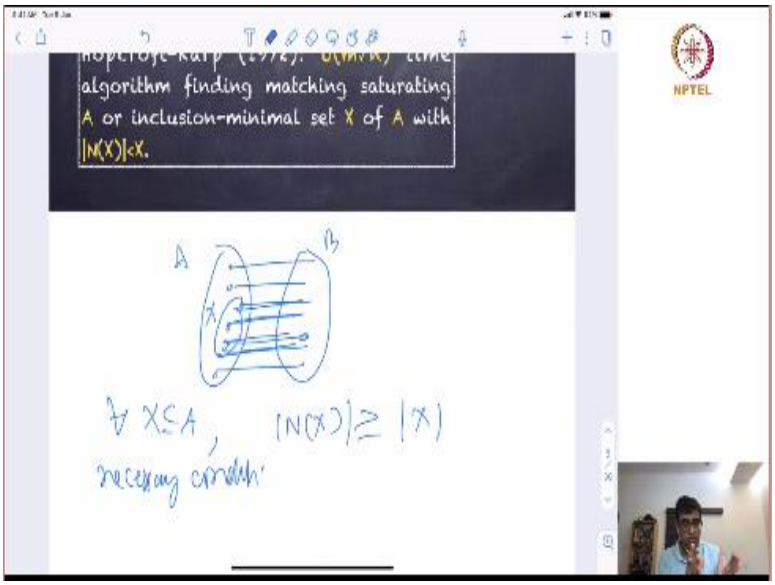
In particular distance graph theory book you can find in Douglas West graph theory book you can find. But for us, what is important is that in a bipartite graph, the number of vertices in a minimum vertex cover is equal to the number of edges in a maximum matching.

(Refer Slide Time: 03:38)



Okay. And there is a Hall's marriage theorem. This is also very classical theorem. What does Hall's theorem says: let g be a bipartite graph. So just to help you a little bit with let us add a piece.

(Refer Slide Time: 04:01)



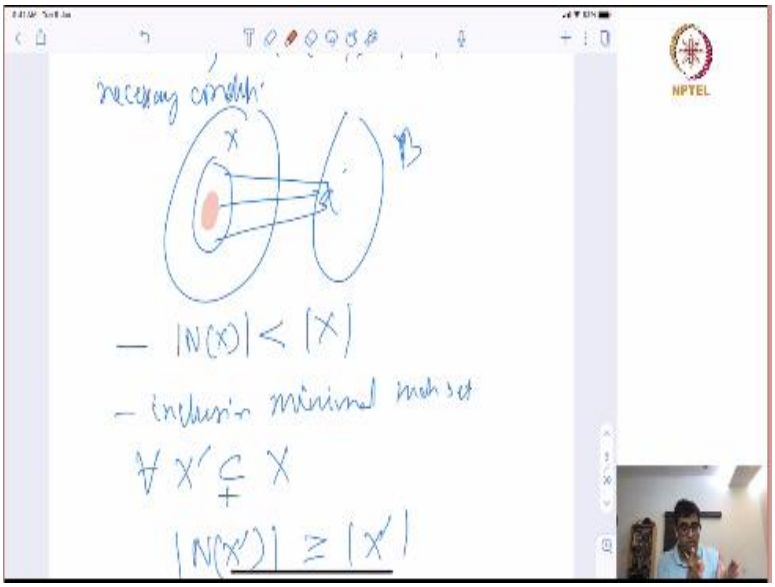
Basically it says that if you look at a bipartite graph with bipartition A and B, and then there is a matching that saturates A, meaning there is a matching that selects. There is no vertex at A that is not part of this matching if and only if for every X subset of A for every X subset of A look at the look at this condition. This is a necessary condition that if there is a matching that saturates every vertex of A.

Then if you look at some set X look at the number of neighbours in the file B, that number of neighbours must be at least side effects because otherwise how will you match the vertices of

X. So, this is the necessary condition and all said that this is also a sufficient condition. So, this is what he proved. G has a matching of A into B if and only if, for all sets X of A number of neighbours that it has in side of B that is N of X is at least as big as side of A.

And again using the matching algorithm of Hopcroft and Karp we can do the following. Either we can find a matching that saturates A or we can find an inclusion minimal set X, so what we can find? An inclusion member.

(Refer Slide Time: 05:21)



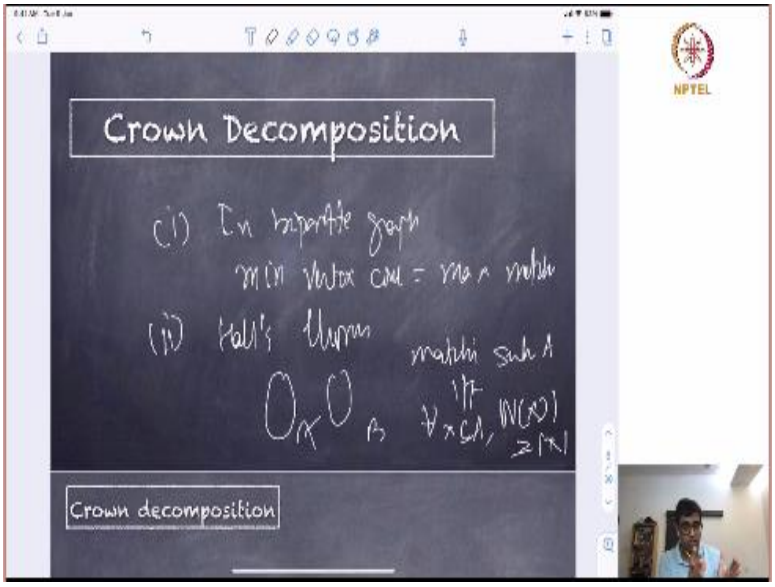
So, it will say hey you are matching the saturate set X or it will give me a set X with a property that the number of neighbours in Y in B like such that the neighbourhood of X is strictly less than X. And what is another property? This is inclusion minimal such set. Meaning for all X in X or rather inclusion minimal sets, I think the best definition should be for all X prime which is a proper subset of X, neighbourhood of X prime is greater than or equal to X prime.

So, if I look at any other small subset a small subset here and look at their neighbourhood in B okay then that is as greater than equal to its side. So, this is a minimal set in the sense that if I look at any proper subset it satisfies the Hall's condition, but if I take the whole set it does not satisfy Hall's condition. So, this is called inclusion minimal witness that it is it does not have a matching. So, first of all okay. So, what does it tell us?

This algorithm tells us that by using this algorithm of Hopcroft Karp in polynomial time, we can either say that look at the matching the saturates a or we can find a subset that violates

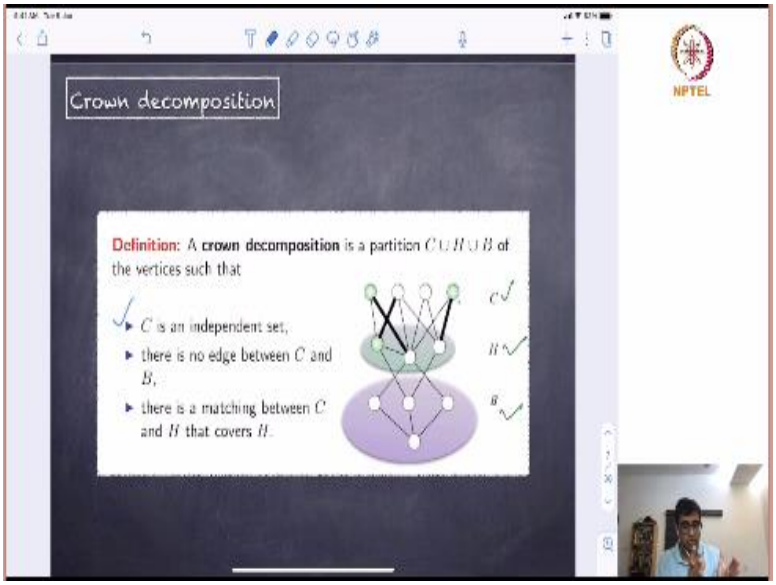
the Hall's condition. That is, hey look at this set. It has a strictly smaller neighbourhood and B and so, you should not be able to find a matching the saturates A. So, this is what you can do using a polynomial time.

(Refer Slide Time: 07:10)



So, based on these 2 theorems, the crown decomposition will be made. So, what is a crown decomposition? So, I hope, so let us just let us just let us just write. So, first what did we saw? We saw that in bipartite graph minimum vertex cover is equal to max matching. Second, Hall's theorem. What is Hall's theorem? It says that if you have a bipartition A and B matching saturating A if and only if, for all X subset of F neighbourhood of X should be at least greater than equal to X.

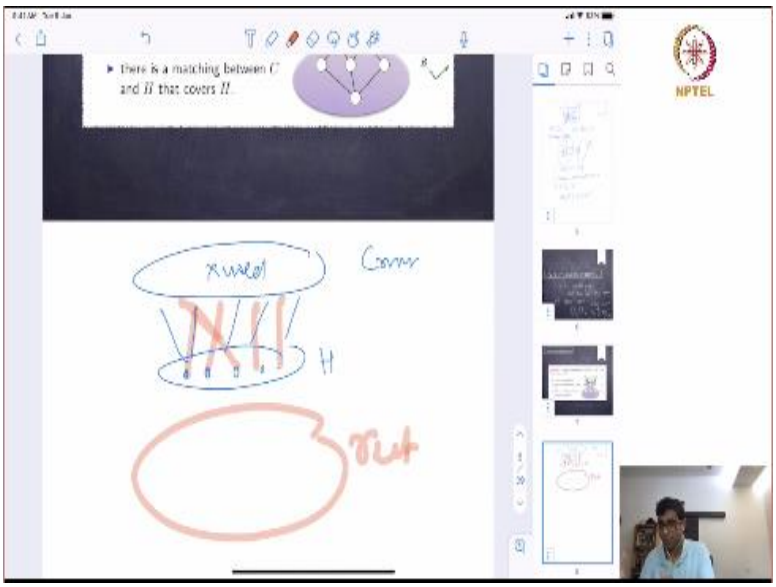
(Refer Slide Time: 08:01)



So, these are the 2 classical theorem that we are going to use and going to show what is called crown decomposition. The crown decomposition is an interesting object. So, what is the crown decomposition? So, crown decomposition is a partition of vertex set of graph into 3 parts, C crown, H head and B the remaining body. So, you can think of this as a crown being placed on the Royal King.

What is the property of this C in an independent set? What is the property? The C is an independent set. So, it is a partition of vertex set into 3 parts, C, H and B. C forms an independent set. There is no edge between C and B. So, you can think of this, if I delete H, there is no edge between C and B. So, H acts as a separator. It separates C from B and more importantly also there is a matching that saturates H into C.

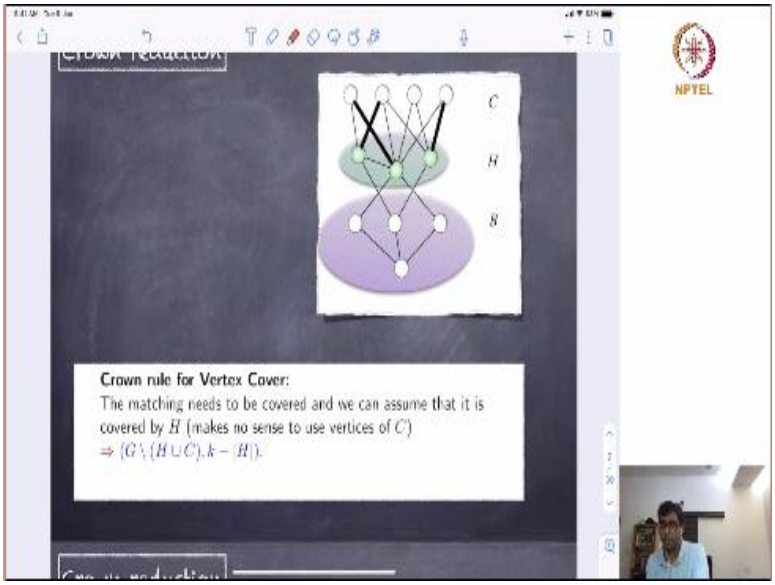
(Refer Slide Time: 08:55)



So, what does it says? I have a head, I have a crown, no edge hair. In fact, there is a matching that saturates head and the rest of the body which is separated So, look at the how about, like just look at our body. This is a head, this is our head, this is our neck, the head, crown and this is the rest of the body. So, if I delete this head then crown and the rest of the body can be separated.

So think of that. Let this head acts as a separator between crown and the rest of the body. And given that crown decomposition exists, what can we say about as a reduction rule.

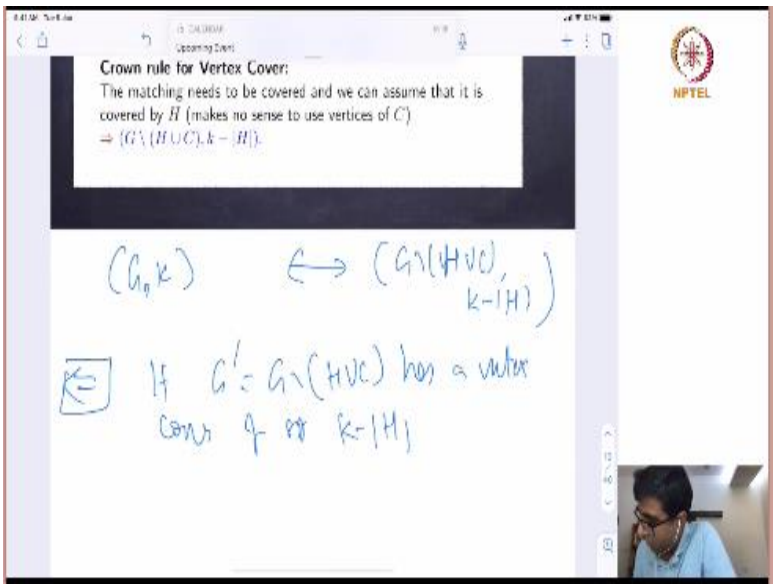
(Refer Slide Time: 09:46)



So it provides a very beautiful reduction rule for vertex cover. So the reduction rule is that if we can find such a separation, then you delete H from the graph. So if you delete H from the graph, then what happens? C becomes isolated vertex. They become degree 0 vertex because there is no edge among them. And so they are all the edges incident to C is goes. So if I put H, I decided to put H in the vertex cover, then I can, C becomes isolated.

So I could delete H union C and reduce my problem to this. Okay, so let us see why this reduction rule is correct. I mean, this is not very hard to show.

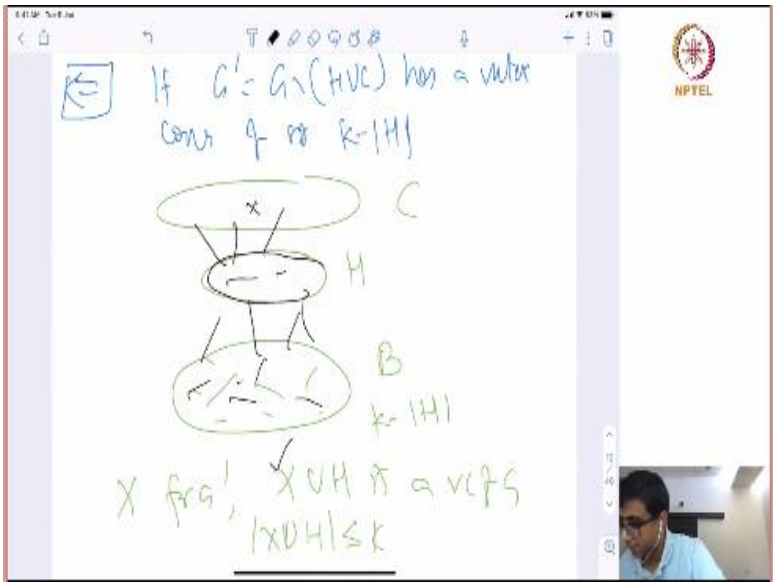
(Refer Slide Time: 10:38)



So this is our ground rule for vertex cover. So, what did we do? We have G, K. And I want to show that G – H union C, K – H is an instance in A. So let us try to understand. So, one direction is obvious, right? Why? So this direction is obvious. If G prime G – H union C had

a vertex cover of size K – mod H, then what will I do? Okay, fine. So, if you recall correctly, how was this? How was it?

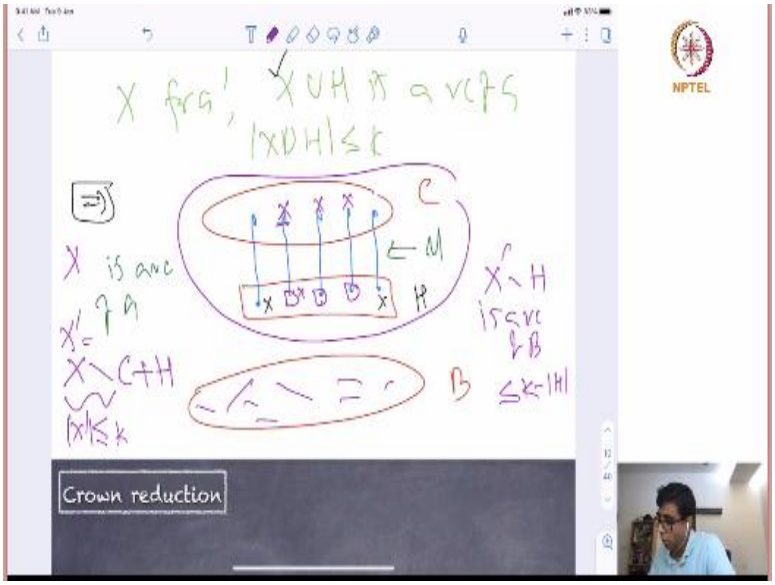
(Refer Slide Time: 11:15)



So, here is our head, here is our crown, and here is our B. Now, you gave me a vertex cover of B of size K – mod H. What I am going to say? Add h, Add h. So, support. So, if you gave me a vertex cover say, X, if you gave me a vertex cover X for G prime, I claim that X union H is a vertex cover of G and of course, the side of X union H is at most K because I just added H vertices. Why? Let us see. All the edges which are here is taken care by x.

Now, look at every other edge every there is no edge here. So every other edge could be here, could be here or could be present here. Now, so in fact, for the remaining edges, which are not present in B, it forms a vertex cover. So if you have taken care of all the edges in B by X, then by adding H, you can take care of all the edges in the graph itself. Okay.

(Refer Slide Time: 12:24)



What about the forward direction this is more interesting direction, okay. But this is also not very, this is where we will use the fact that, here is our C, here is our H and here is our B. So let us suppose there is a matching the saturates this, we know this. Now notice that any vertex covered, any C is a vertex cover of G, what is the property of C? Like, this is a matching.

It means every vertex cover must pick at least 1 vertex of the matching edges. So if it picked up all these edges, all these vertices, then we are very happy. Because then I know there is a vertex cover that contains whole of head. So that is great. It is done. But what could happen is that it could be that your vertex cover picks up some. It is like this, then my transformation is going to be X. This is my vertex cover X. So I am going to transform like this, I am going to say look, from X, delete the vertices of S, add all the vertices of H.

Look what I did. From X, so notice that X definitely contains, now for every vertex here, I have added this vertex, that is all that I have done. So what I've done, I have made a new X prime, which is from X, I've deleted all the vertices in C. And I have added all the vertex in H. Now, what is the size of this? So every time you delete something, you delete something from C you have added exactly 1 vertex side.

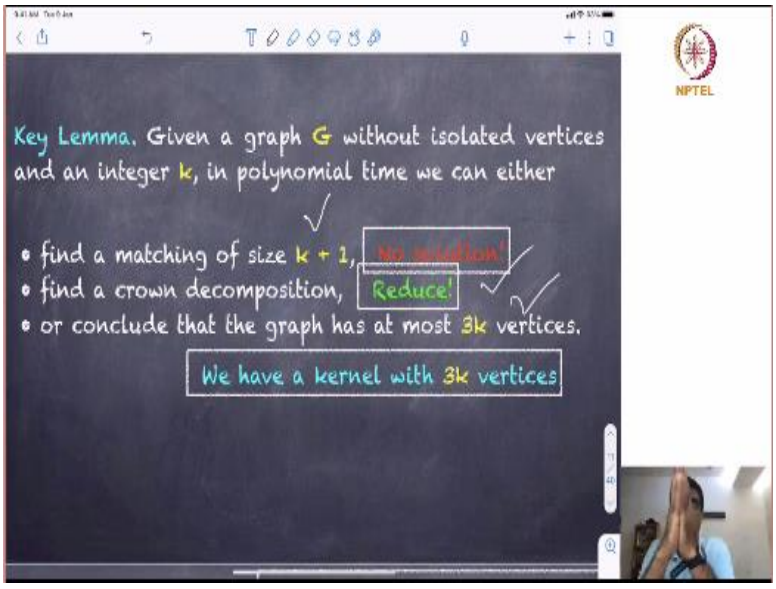
You might say that, well, maybe this vertex is already there, but then you are adding nothing to it, right? So you can show that even the sine of X prime is less than equal to K. But what is a property of this? It is a vertex cover that contains all of head. So because of the property that you have a matching saturating edge, you are assured that luckily every vertex cover must pick at least 1 vertex of this matching edges.

You use this fact to move the vertices from C to vertices in H to get another vertex cover of size K, X prime that contains all of head. Now, once you have this you can say that look clearly X prime – H is a vertex cover of B, because who will cover these edges? This cannot be covered by head, it can only be covered by this. And hence, the side is at most K – H. You might say that oh maybe, but look big.

This is where we also use the fact that there is a matching saturating edge which implies that any vertex cover X must contain H vertices Cardinal TH vertices from C union H. So, actually when you do delete the vertices from C union H, you are decreasing the parameter by Cardinal TH. So, if you work around, tickle around with it, you will be able to get it. So, I hope.

So, the reduction rule is clear the matching needs to be covered and we can assume that it is covered by H because makes no sense to use your dcfc. And this is your formal proof that I tried to give you.

(Refer Slide Time: 16:18)



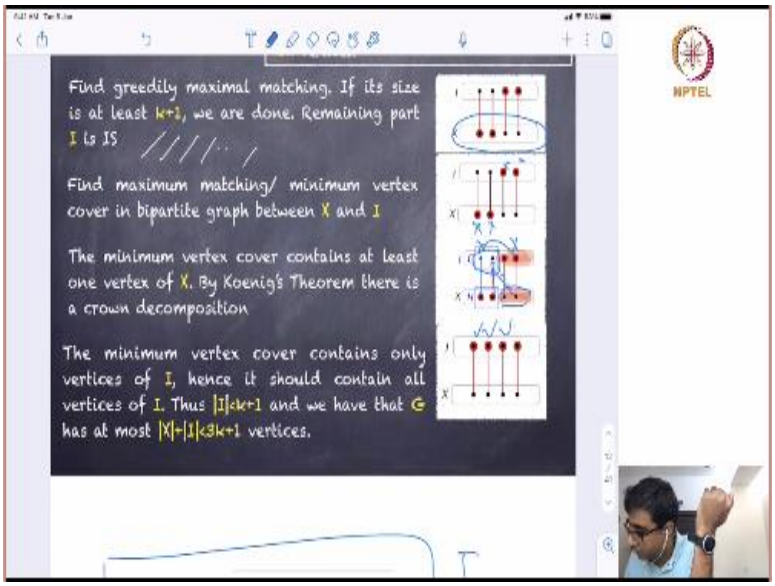
Now, so what is my crown reduction rule? So if I can find a crown, then I have a reduction rule, but how do I find a crown? So in the next step, I am going to give you a lemma which I am going to show how to find. So my key lemma is going to do the following. I am going to give you, given a graph G without isolated vertex and integer k in polynomial time, we can either find a matching of size k + 1. Okay.

If I find a matching of size k + 1 then there is no solution. There is no solution of size at most k, done or I can find a crown decomposition in which case you can reduce the size or conclude the graph at at most 3k vertices. So, either I will say look, this has no vertex cover of size at most k or I will find a place to apply your reduction rule which is a crown decomposition. Delete head, get a smaller graph, reduce the parameter by some amount.

If I cannot do if I cannot assert it has no solution, I cannot apply reduction rule, then I will say that then the graph is very small number of vertices, mostly k vertices and then that is your kernel. So, if you have applied the sequence of operation and you have ended up into a graph with at most 3k vertices then this is 1 angle card.

Notice what is the difference between this kernel and the kernel for vertex cover that we have seen before. Before the size of the vertex cover kernel was like k square. Now, what we are able to get is a kernel with order k vertices, just 3k vertices. That is the basic difference.

(Refer Slide Time: 17:47)

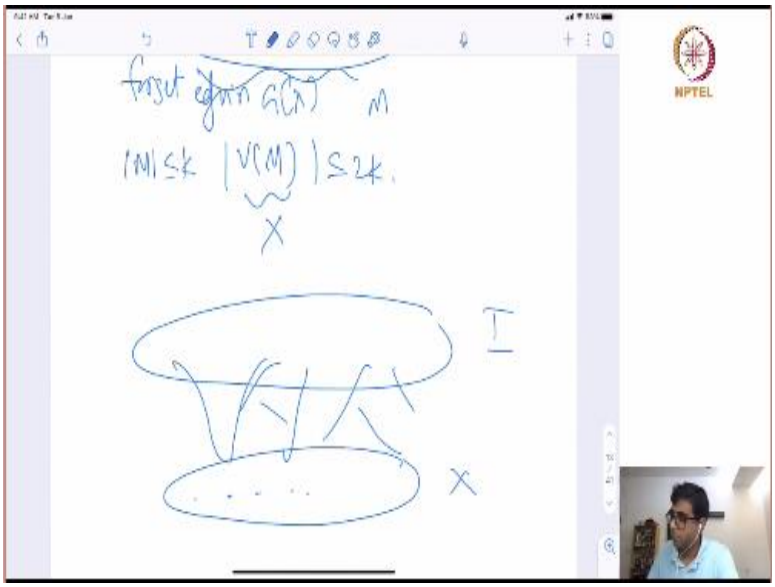


The proof is very simple. So, this is a proof of key lemma. So, the proof of key lemma is that we find a greedily maximum matching. So, we find a greedy maximal matching in graph G. So, we find matching. If it has size at least k + 1 what happens? Then you say no. So, that is your find a matching of size k + 1 alright, if we are not able to do it, we will find.

So, there is no maximum matching of size k + 1 which implies that so, I am find a greedy maximum matching. If its size is at least k + 1, we are done. So, what is the outside of it. So, we know that if I do not include any like, if I look at the vertices of maximal matching and

look at outside, what is it that forms an independent set? So, let me draw a picture for you. So what has happened?

(Refer Slide Time: 18:59)



So, my first step was that, let us find this. Let us find this maximum matching. And this is my I, this is my I why? Because what you are side of this M, right? We know that the maximum matching size is at most k otherwise we would have said no. Then the total number of vertices involved in this matching is bounded by 2K. Okay. So now we will look at. So let us, we have called it this vertex set at X.

So let us call that X. So this is X, and this is an independent set. So this is our first step. So now let us just focus on x, which is the endpoints of my matching edges. So this is endpoints of my maximal edges and the vertices which is outside this. Now, what I do is, that now I think of this graph, I forget, so this is my X, forget edge n in graph induced on X. So let us for now, I said okay, this is my X, I do not have any edges here, I have an I here.

So what is this graph induced on X union I now? It is a bipartite graph, why? Because there is no edges in X, there is no edges in I. So every edge is going between a vertex in X and vertex in I. So now what I am going to do? Between X and I, I am going to find a matching maximum matching slash minimum vertex cover in bipartite graph between X and Y. So now I am going to find a maximum matching between X and I.

Notice that if this also maximum matching has size k + 1. Then what can you say if this maximum matching also assigns at least k + 1, then you can say hey, look, there is a small

portion of my graph there I can find a maximum matching of size k + 1 or more, then this is not a solution. So, let us assume that the maximum matching had size upper bounded by k. Now, what do you know about this?

So, this is a matching and you found minimum vertex cover. Now, you know that every minimum vertex cover must pick 1 end point of every edge. So, in particular of this matchingedge but by Konig's theorem, I know that in bipartite graph minimum vertex cover is equal to maximum matching. So, the property is that minimum vertex cover contains exactly 1 vertex from this matching edge that we have found.

So, suppose this is the vertex cover. So, this is my match. This red coloured edge of my matching edges and this bulbs are my bulbs are my vertex of X. So, if minimum vertex cover contains at least 1 vertex of X, then I will find your crown decompose. What is my crown decomposition? You look at the intersection of X intersection of vertex minimum vertex cover into X.

Call that a head, look at their matching endpoints called just those 2 guys an endpoint and remaining as the rest of the body. So, now, let us understand. So, look by definition this is my head and there is an edge and you know that the endpoints of edge are into I. So, definitely the there is no edge. There is no edge among the vertex edge and C. And but the only thing we have to make sure that there is no edge between C and the remaining.

But look at, suppose the edge between C and a vertex in X because this is my whole vertex set. My whole vertex set is like some My body is going to be some part of C and some part of edge. So, definitely a vertex in the crown does not have an edge here, because this is whole independent set. But what about an edge here?

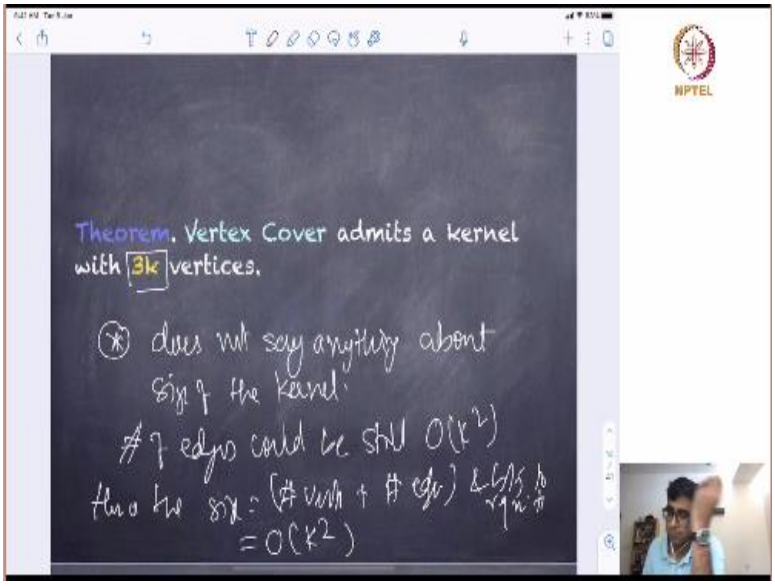
But if there is an edge here, who is going to cover this vertex? Who is going to cover this edge? In the bipartite graph, the only edges which can be like the red colour guys form a vertex cover. So, now, these are the vertices which are not selected in the vertex cover. This is a vertex. The vertices which are not selected in the vertex cover, which implies that there is no edge between this and this vertex, which implies that we can indeed get a decomposition into crown, head and the rest of the body.

So this is great, okay. So, the last case I found. So, I found this maximum matching and I find this minimum vertex cover but there is no vertex inside X. If there is no vertex inside X, then every minimum vertex cover is here whatever assumption. Our assumption is graph does not contain any isolated vertex. There is no degree to your vertex which implies that all these vertices are like look at any vertex in independent set, they are incident to some edge going to X, which implies that my minimum vertex cover must contain every vertex in I.

But the maximum matching was size k. So, you have picked up some k vertex from I. So, now, what is your size? Now, minimum vertex cover contains only vertices of I. Hence it contains all vertices of I. So, the side of is bounded by k + 1 at strictly less than k + 1 and we have that. So, what is the size of my whole graph? Side of x which is bounded by 2k and side of I which is now we have bounded by I, k.

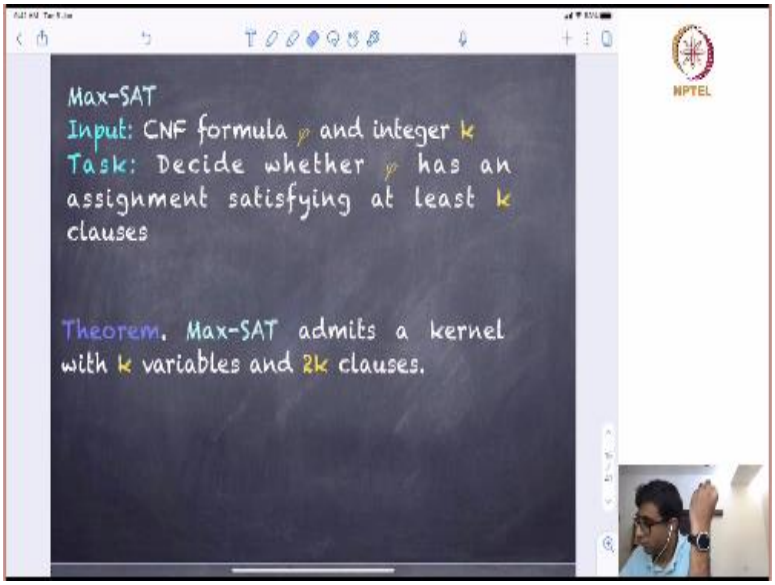
So, total number of vertex in this graph is upper bounded by 3k. I hope that kinds of clears it. So, we have done with the crown reduction. So, we proved with the crown reduction.

(Refer Slide Time: 24:57)



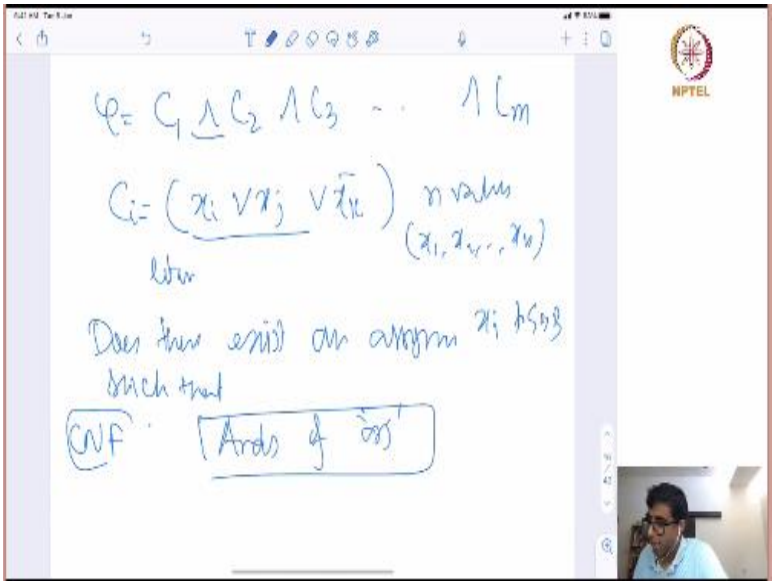
So what we have shown is a vertex cover admits a kernel with 3k vertices. By the way, this does not say anything about side of the kernel. Number of edges could be still order k square, right. So hence the size, which is number of vertices + number of edges and bits to represent them could still be order k squared. All we are able to do with this new kernel is to reduce the size of my graph to order k squared up. So order k vertices in particular 3k vertices.

(Refer Slide Time: 26:04)



So let us see on the application of this nice kernel. So the next problem, which is nice crown reduction rule. So I am going to talk about, slightly up until now, we have been talking about graph problem for now, I will talk about a satisfiability problem a slightly different problem. So, we are given a max SAT which you see in a formula of I and integer k. So what is CNF? formula? I am not sure if you are aware, maybe you are aware.

(Refer Slide Time: 26:21)

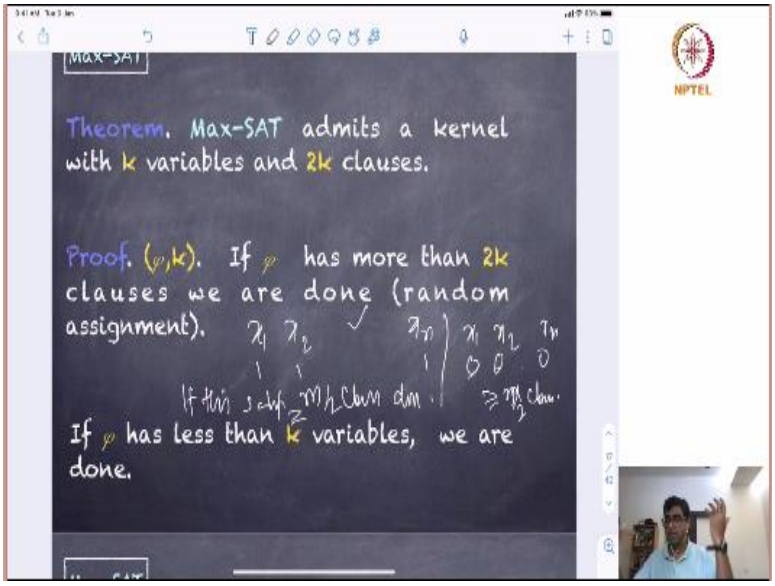


But like, let me just show. What is CNF formula? CNF formula phi is like m clauses, with each C i is like some x i or x j or x k bar. So It is like some literals. So we have n variables x 1,x 2,x n. And what are we looking for? Does that exist an assignment x i to 01 such that each of the clauses satisfy. Each clause because this is and. And this is or. CNF means and of or s. And when a clause is satisfiable?

So, if you set x equal to 0 and 1, if a clause evaluates to 1 then it is 1. So for example, if I put x k equal to 0 then x k bar is 1. So, it does not matter what other variables are satisfied because or even 1 of these variable gets 1 this clause gets satisfied. And we are looking for a finding an assignment so that each clause is satisfiable. Such that each clause is satisfiable. But we are not like but not all formula may be satisfiable.

So, we are asking is there a clause is there an assignment which satisfies at least k clauses, okay. So, our problem is CNF formula phi and integer k can we find an assignment satisfying at least k clauses. And what we will show that max SAT admits a kernel with k variables and 2k clauses. Okay.

(Refer Slide Time: 28:36)

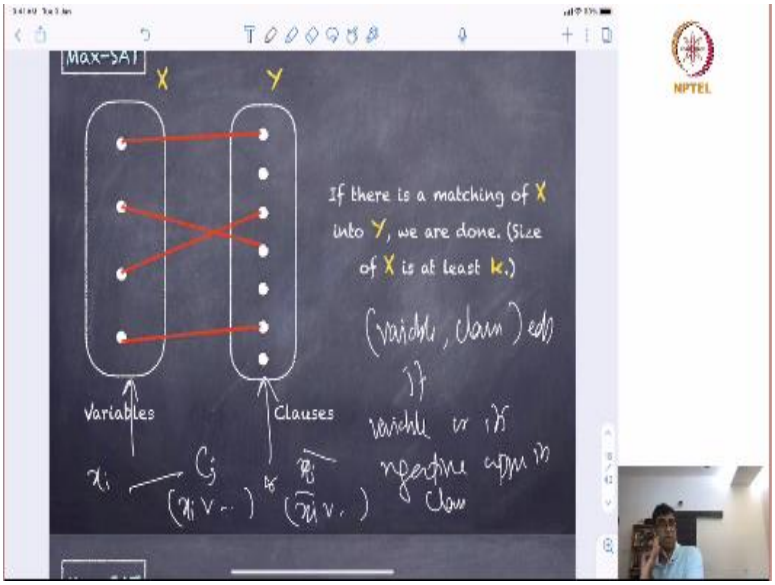


So, 2k clauses is very easy. If I had more than 2k clauses we are done. Why? You start with an arbitrary assignment. Start with an arbitrary assignment, any assignment. So you let us say, that is what I mean by are arbitrary segments. I do x 1, x 2, x n. I substitute everybody 1.What happens? What happens is that, if this satisfies at least m by 2 clauses done.

Otherwise what I will do, let us complement this. It means x 1,x 2, 00 everyone is 0 right. So, every clause which was unsatisfied before because right now will be satisfied, because we have complemented this. So, either this has at least m by 2 clauses or at least this satisfies m by 2 clauses, right. So if the number of if the if you have more than 2 clauses, then you know that in polynomial time that hey, I mean, there is an assignment that will satisfy them they can find this assignment in polynomial time.

So this is an yes instance. So for now, let us assume that the number of clauses is upper bounded by 2k. And if phi also is atmost k variables then we are done. So let us assume that we are in the case when we have at most 2k clauses, and the number of variables are more than k. So let us see what else.

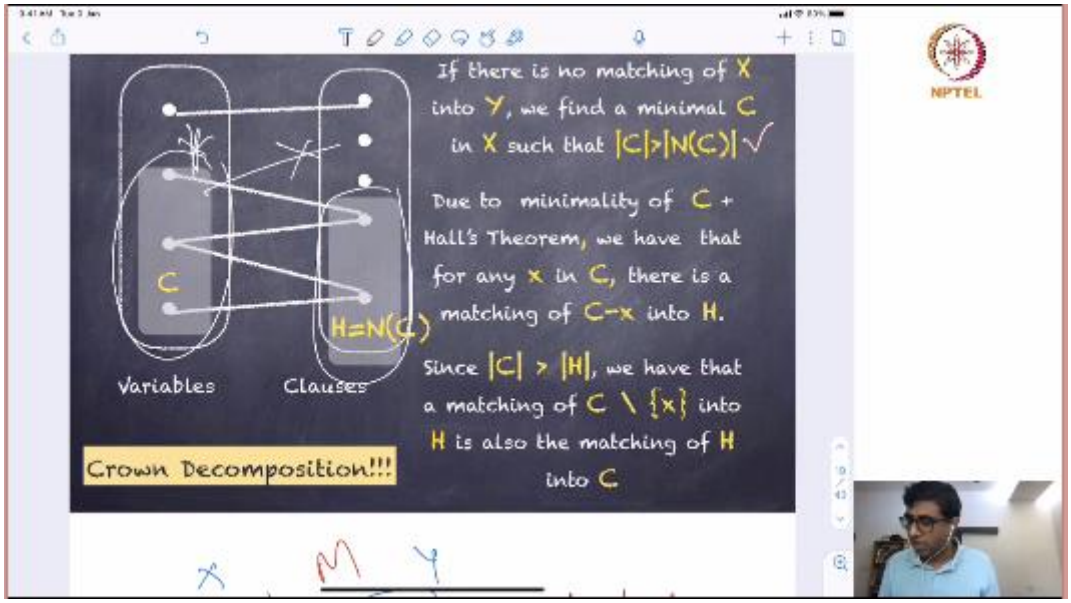
(Refer Slide Time: 30:07)



Now in this case, we are going to make a following graph, bipartite graph. So 1 side of my bipartite graph are variables other side is a vertex for so there is a vertex for each variable. In Y there is a vertex for each clauses and I put an edge between a variable clause variable clause edge if variable or its negation appear in clause. So suppose I have a variable x i and there is a clause C j.

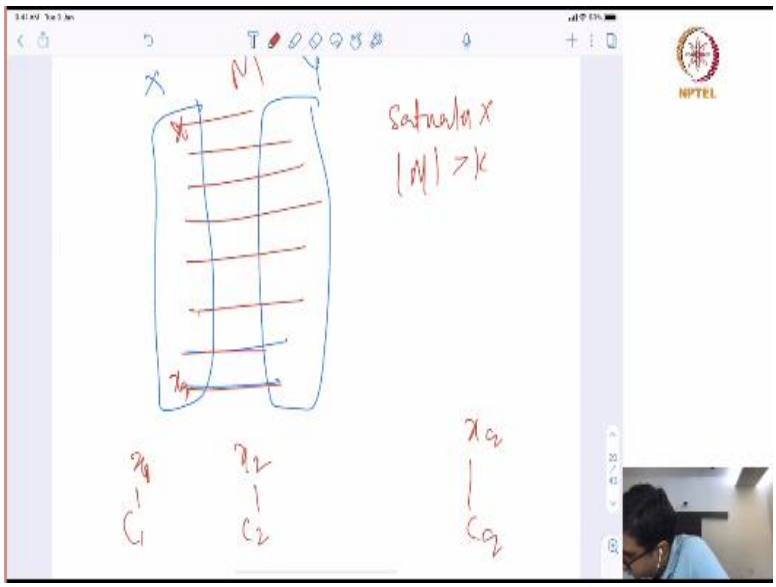
So, if it appears x i, then I put an edge or it appears x i bar something even then I put an edge. So this is what might happen. Now, I would like to apply.

(Refer Slide Time: 31:04)



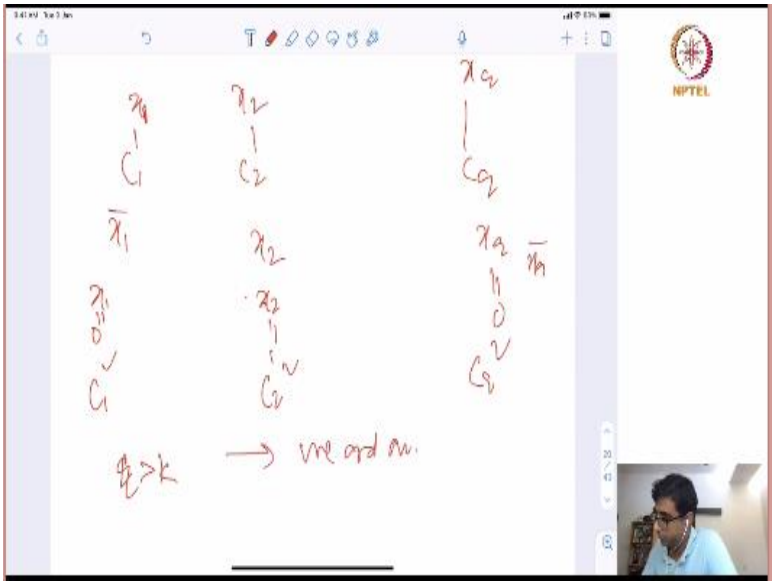
So now I look at X and look at Y. So if there is a matching then what happens? If there is a matching that saturates X, let us see. That is a okay. So I look at this X and Yand I ask myself, what happens if there is a okay. Let us ask ourself what happens?

(Refer Slide Time: 31:41)



This is a X and this is Y. Okay. And then the matching that saturates X. Let us use some other colour. There is a matching m, that saturates X. What is the size of matching? It is strictly greater than k. Otherwise, we are done. Now let us look at these clauses. Okay, for each variable, x 1, let us suppose this is x q. Each x i, x 1, let us call the clauses with C 1,x 2 is associated to C 2, x q associated to C q. Now I asked myself. Look at x 1.

(Refer Slide Time: 32:04)



How did x 1 appears in C 1? Suppose x 1 appears at x 1 bar, then I am going to be ok, fine. Then let us assign x 1 equal to 0. I look at x 2 in C 2. x 2 appears in C 2 as x 2 Oh, then let us assign x 2 equal to 1, so on and so forth, x q had been assigned 0 because maybe it appears at x q bar. Now notice what happened. Is clause 1 satisfied? Yes. Now I have given x 1 to x q. These I have come up with an assignment. And let us look at C 1.

Is C 1 satisfied? Yes. Because x 1 bar x 1 is 0. So x 1 bar is 1 and I do not rest of the variables. Whatever assignment you give does not matter. So C 1 is satisfied. Similarly c 2 is satisfied. See so we can satisfy all q clauses at least right? I do not know what that could be the number of clauses is around 2k. But I know that I have found an assignment which is able to satisfy q clauses, but q is more than k. So we are done. So this is how you.

So, basically, what happens is that for every clause you have assigned a variable. Say look, your job is to satisfy this. Do not care about other clauses. Other there is like you do not care about. So having able to find for each variable, 1 clause each like that they all take care of themselves. And rest, there may be more clauses which can be satisfied, but that is perfectly fine.

So, if we have a matching that saturates my X, then I do have a, I do have an assignment that satisfies at least k clauses. And now what I will show to you is that if this is not a case that we can apply reduction rule. If there is no matching of x into y, we find a minimal C into X says that cardinality of C is like this is a violating set. We are at the end of C this cardinality of C is larger than the neighbourhood.

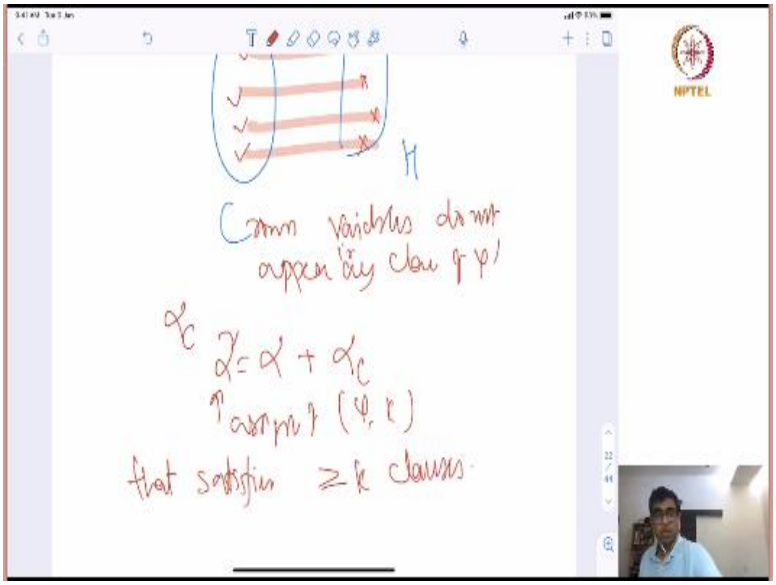
Remember, we talked about that. In a bipartite graph there is a matching otherwise there is a set a minimal set which violates the Hall's condition. Now, what is the definition of C? You know that for every subset of C, there is a matching that saturates? There is a match for every subset of C, the neighbourhood of that C prime is greater or equal to C, which implies for any X and C there is a matching of C – X into H. So you delete a vertex set.

Now for every subset you know their neighbourhood. So, there is a data set which saturates C. But because of the size constraint, because the size constraint that matching must be saturating edge. So, since C, cardinality of C is greater than equal to H we have that a matching of C – x into H is also the is also the matching of H into C. So, this automatically gives us a kind of a crown reduction. Why? Look at the variables here.

This is my crown. This is my rest of the this is my head. And by definition the neighbourhood of C is exactly equal to this. So, there is no edge here, there is no edge here. So, we automatically get a crown decomposition with C H and the rest here. So what is our reduction rule? Our reduction rule is very simple. Remove H from remove ah the head means the clauses which appears here from psi and decrease k by cardinality of H. That is you have decrease psi – H and k – H is a new instance.

Now, the question is why is this correct? So, I will show you why this is correct. As before this is not something amazingly tread terribly difficult, but like let us see why it is easy. So, why it is easy. So, what we have done.

(Refer Slide Time: 36:27)



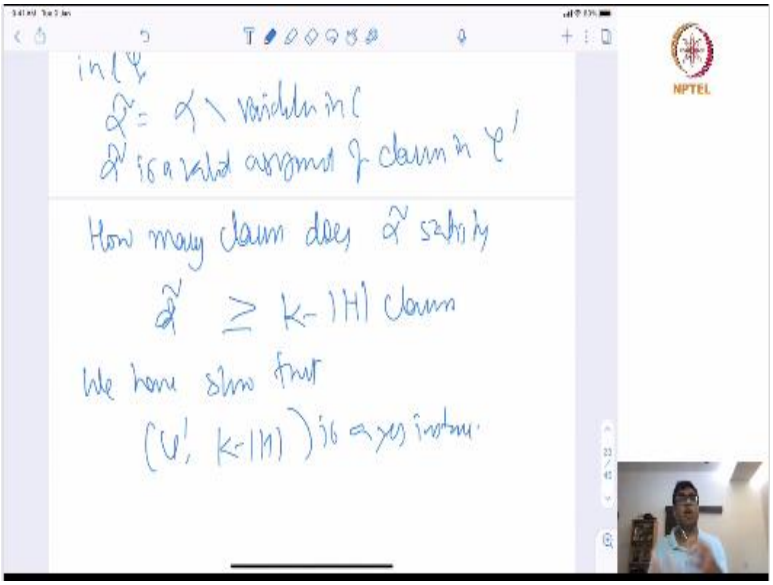
So, again the reverse direction is much more easier. So, what happens? If notice that these variables only appears in these clauses, right. So, these variables are local to these clauses, because they do not see anybody else that is a property. So, these are like this head separates these variables from other clauses. So suppose there is a phi prime which is like phi – H.

Suppose there is an assignment alpha satisfying K – H clauses in phi prime. Now, what I will do? I will say okay fine. Let us come to my crown. Let us come to my head. I know there exists a matching that saturates H. And I know that this clause is this like, this crown. This crown variables do not appear anywhere, in do not appear in any clause of phi prime. And now, we have this matching I said okay.

For this clause, take the variable and assign it accordingly. Take this clause, there is a local variable, a private variable. Assign it accordingly, the way we did when we had a matching. And by doing this local assignment of every vertex, so now you fix this assignment. Let us call it alpha C. So now alpha + alpha C gives you a some alpha tilde on assignment of phi, k that satisfies at least k clauses that is great.

The reverse direction is even more easier right? Reverse direction is that okay fine. Look at an assignment that satisfies at least k clauses.

(Refer Slide Time: 38:54)



And you know, that look at an assignment that satisfies at least k clauses in psi. Look at alpha minus variables in C. Okay, let us call it alpha tilde. So alpha tilde is a valid assignment of clauses in phi prime. Okay, alpha tilde is a valid assignment of clauses in the phi prime. So now ask yourself how many clauses does alpha tilde satisfy? Alpha tilde definitely satisfies at least K – mod H clauses.

Because the only clauses which alpha tilde will not satisfy are the ones which appear here. Because rest of the clauses do not have these variables and those must be satisfied with the variables from outside which it is fine. So, we have shown that phi prime, K – H is a yes instance. So, you see using combinatories and using arguments to matching theory, we are able to come up with a kernel for the problem, which is much more complex and complicated or interesting.

And the property which we use is that I have been able to find a local solution which is opt in the sense and we can modify that solution so that on local solution can be extended to a optimum solution for the whole problem. And that is what we have exploited in all this reduction. Oh, look at a solution of optimum, look at how the locally looks like. Oh, if they look like locally I can transform into another solution that locally look like the way we wish to look like. And then we argued.

So, in some sense, there is a good local solution which can be extended to an optimum solution. And we will take this idea forward in the next lecture. And that will be the last lecture on the topic of kernalization. Thank you.