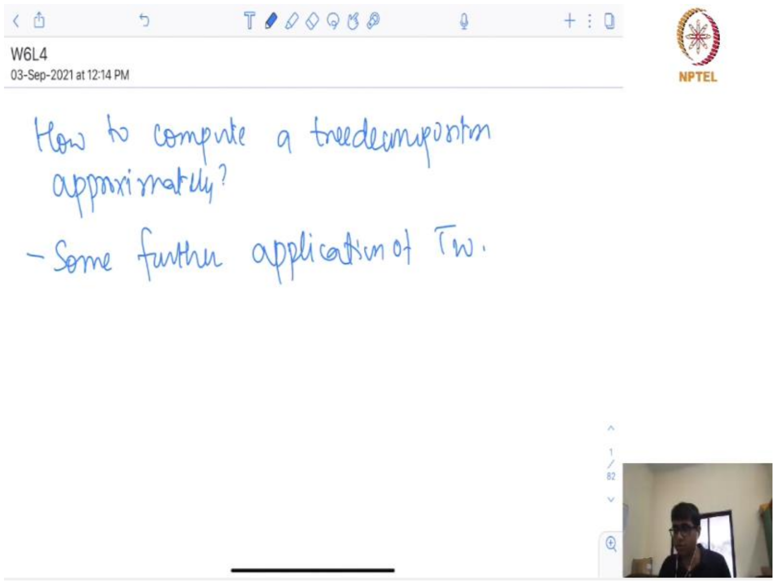
Parameterized Algorithms

Prof. Neeldhara Misra
Prof. Saket Saurabh
The Institute of Mathematical Science
Indian Institute of Technology, Gandhinagar

Lecture - 29
FPT Approximation Algorithm for Computing Tree Decomposition and Applications-Part 01

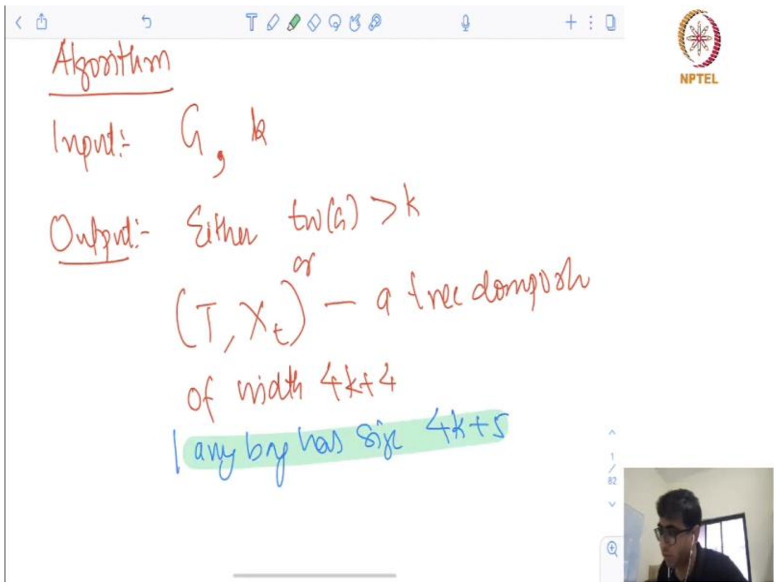
(Refer Slide Time: 00:16)


W6L4
03-Sep-2021 at 12:14 PM

How to compute a treedecomposition approximately?
- Some further application of Tw.

Last lecture we saw how to compute tree decomposition. Today we will how to compute tree decomposition approximately. In today's lecture we will see some further applications of treewidth. But before we do that let us try recall the argument for computing tree decomposition approximately because we did it slightly in the fast in the lecture.

(Refer Slide Time: 01:00)



Algorithm

Input:- G, k

Output:- Either $tw(G) > k$

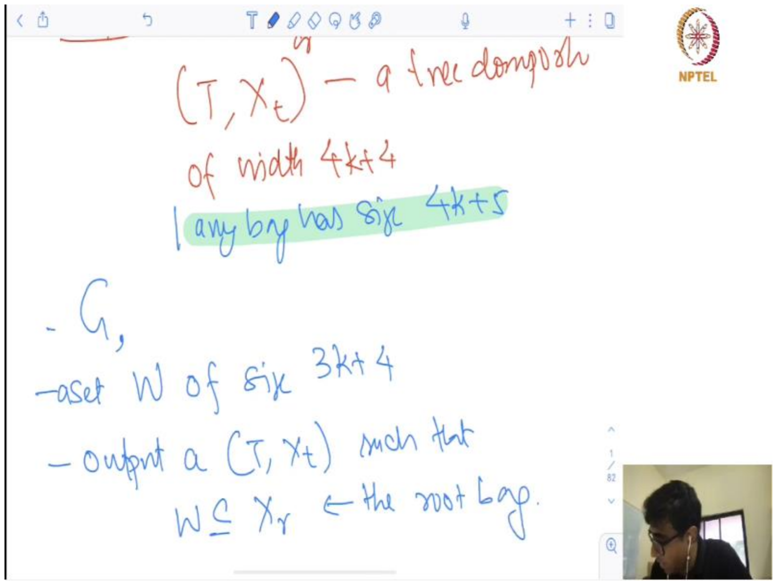
or

(T, X_t) — a tree decomposn of width $4k+4$

| any bag has size $4k+5$

So, what did we do? So, our algorithm had the following property, what the property of our algorithm? So, our algorithm took as input a graph G a number k and what does it output? It outputs two things either treewidth of G is more than k or T X t a tree decomposition of width 4k + 4 or in other words outputs such that any bag has size 4k + 5. So, this is what our algorithm did. Either it outputs G, or so how did our algorithm?

(Refer Slide Time: 02:15)



or

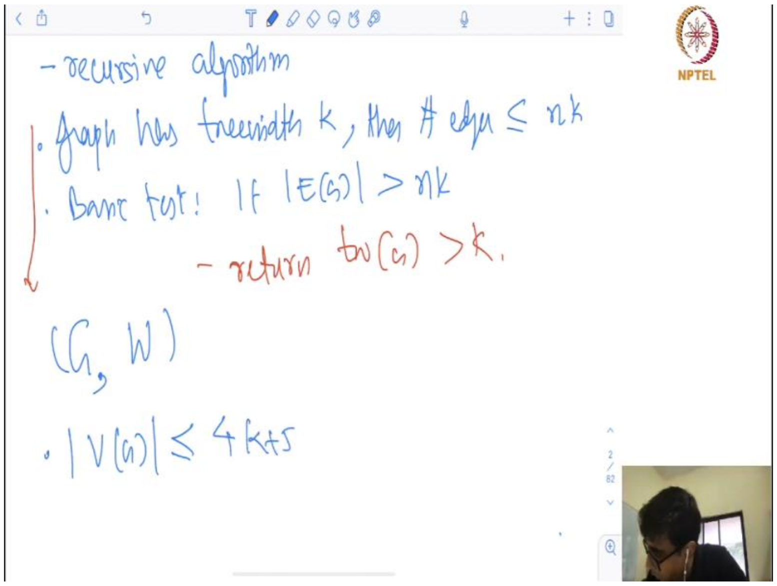
(T, X_t) — a tree decomposn of width $4k+4$

| any bag has size $4k+5$

- G,
- a set W of size $3k+4$
- output a (T, X_t) such that $W \subseteq X_r$ ← the root bag.

So, if you recall we had this nice construct nice existential algorithm which basically because for the recursive nature of the algorithm. Actually, what our algorithm will do take our algorithm will take a graph G a set W of size 3k + 4 and output or tree decomposition X t such that W is contained inside X r the root value. So, this is how we designed our algorithm.

(Refer Slide Time: 02:59)


- recursive algorithm

· Graph has treewidth k, then # edges $\leq nk$

· Basic test: If $|E(G)| > nk$

- return $tw(G) > k$.

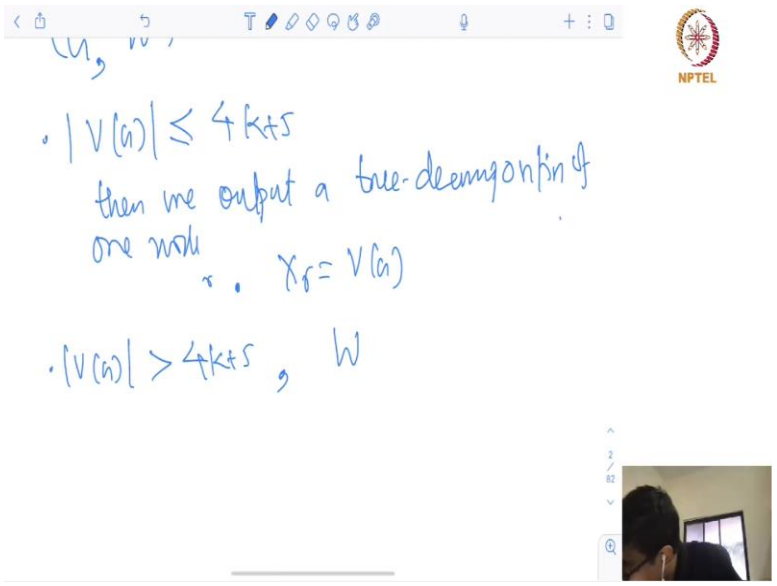
(G, W)

· $|V(G)| \leq 4k+5$

So, let us just try to so this is what we will try to do. So, what are the steps of our algorithm? So, the algorithm, it is a recursive algorithm, and it works as follows it just check so, first of all you know that if the graph has graph has treewidth k, then the number of edges in this graph is upper bounded by n times k. So, you could have a basic test basic test if number of edges in the graph is more than n k return treewidth of G is more than k.

So, they are like some of the basic tests that we could perform at any point of time or just before even algorithm starts. And now what this algorithm does so algorithm will take a particular a graph G it takes a set W and it will construct the required so first thing we check. What is the number of vertices in my graph? If the number of vertices in my graph is upper bounded by if the number of vertices in this graph is upper bounded by 4k + 5.

(Refer Slide Time: 04:59)



• $|V(G)| \leq 4k+5$

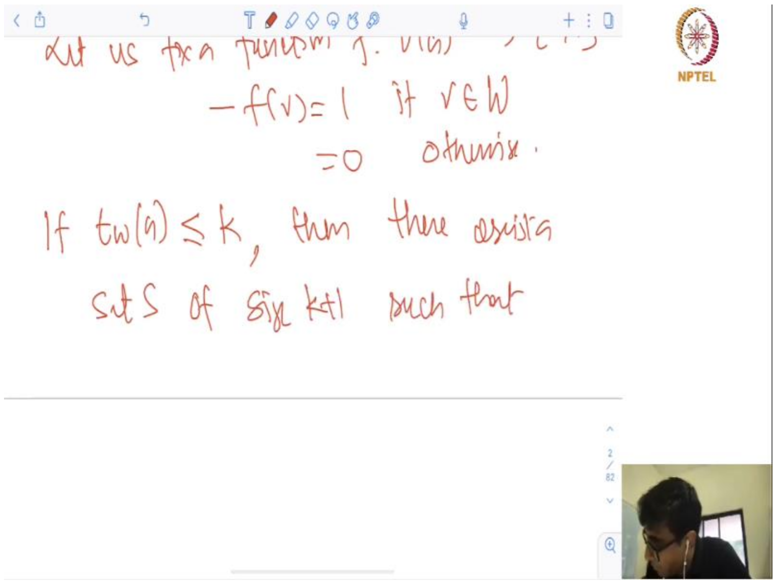
then we output a tree-decomposition of one node

r. $X_r = V(G)$

• $|V(G)| > 4k+5$, W

Then the output or tree decomposition of one node. What is that? So, this is it and the X r because r is basically vertex set of G and it automatically satisfies all the property. Otherwise now we are in the case when vertex set of G is more than strictly more than 4k + 5 and we are given a set W.

(Refer Slide Time: 05:32)



Let us fix a function $f : V(G) \to \{0,1\}$

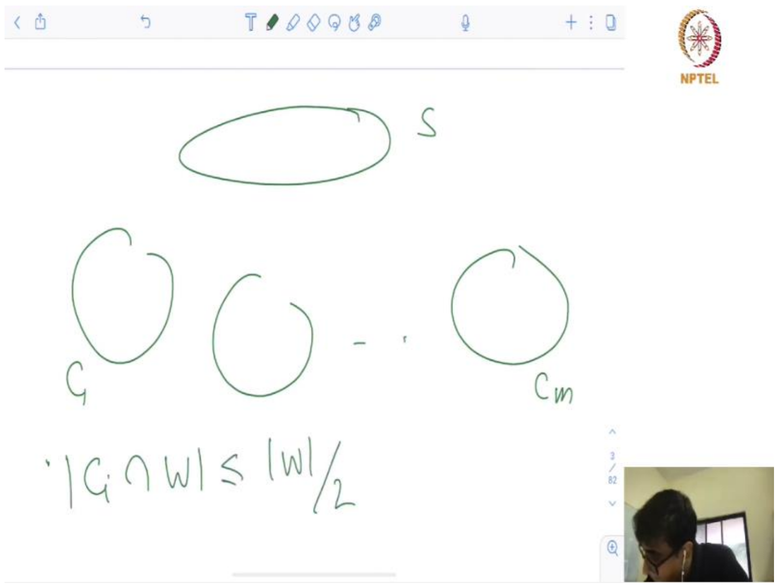
$- f(v) = 1$ if $v \in W$

$= 0$ otherwise.

If $tw(G) \leq k$, then there exists a set S of size $k+1$ such that

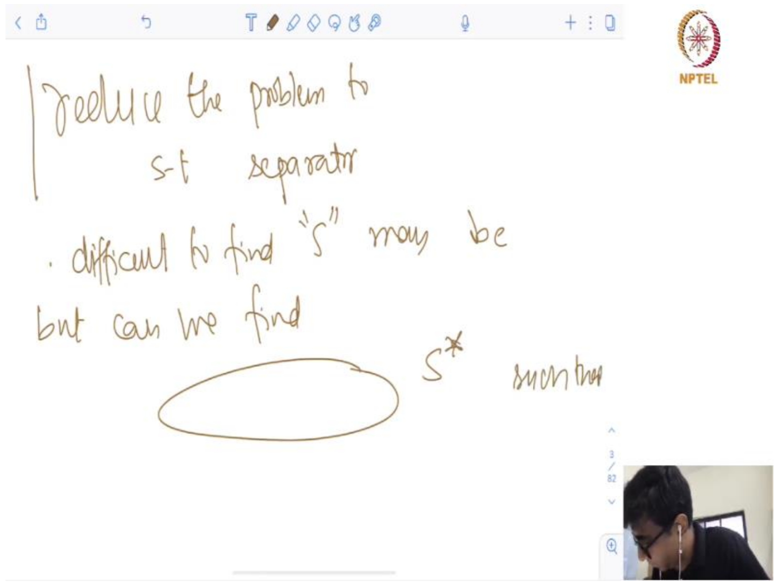
So, if you recall correctly what how did our algorithm proceed at this point of time we said. Let us fix a function f from V G to 0, 1 and what is the property of this function is that f of v is = 1 if v belongs to capital w = 0 otherwise. And we know that if, what we know? If treewidth of G is at most k then there exist a set f of size k + 1 such that what happens is that?

(Refer Slide Time: 06:31)



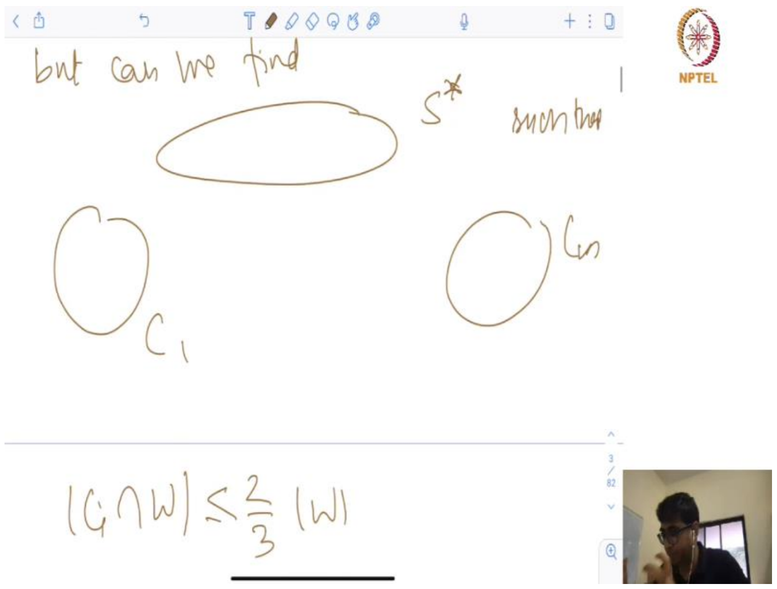
Here is set S and look at any connected component C 1 to C m C i intersection W is upper bounded by w by 2. So, if a graph G did have a treewidth at most k, then there is a property that you can find a set f of size at most k + 1. Such that if I delete this then every connected component of G - S contains at most half the vertices of W. Now we did not know how to prove this algorithmically, but we observed something interesting. So, what was that? This is the place where say let us try to reduce.

(Refer Slide Time: 07:19)



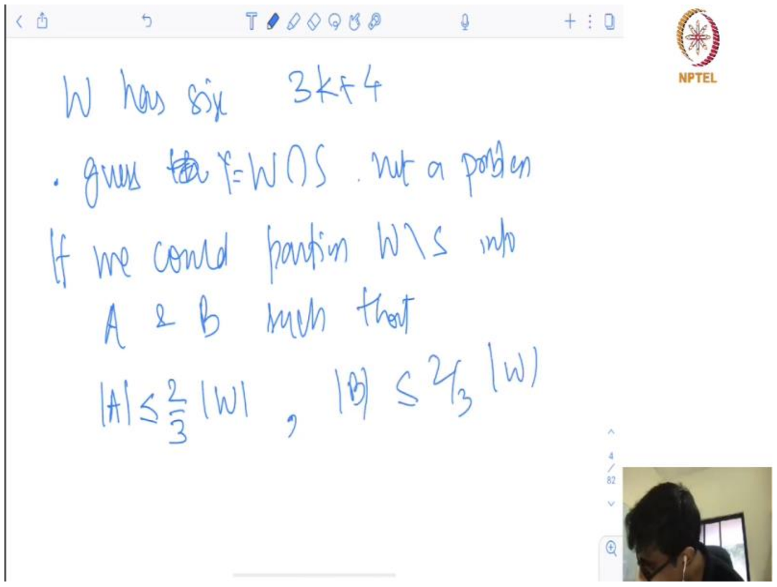
Reduce the problem to S t separator. So, rather than whatever we are looking? We say look it is difficult to find S maybe but can we find say let us call it S star such that.

(Refer Slide Time: 08:05)



If you look at components now so you want to see a property? That C i intersection W say at most two-thirds of W. So, I am saying that look I cannot guarantee that I can find such a S efficiently of course we could have found like in n to the power big of k time we can find you try all possible case i subset k + 1 size subset and you have found it. But we do not want to do this. So, what we said so what could be.

(Refer Slide Time: 08:35)


W has size $3k+4$

· guess ~~that~~ $Y = W \cap S$. not a problem

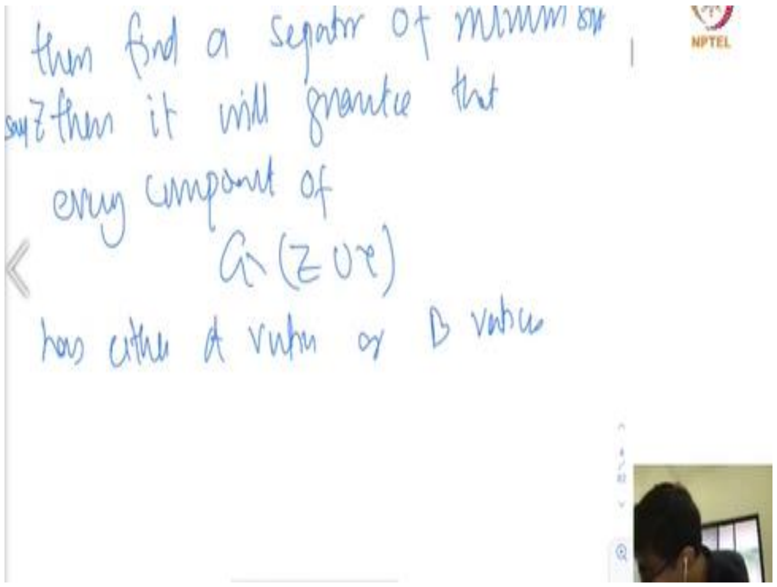
If we could partition $W \setminus S$ into

A & B such that

$|A| \leq \frac{2}{3}|W|$, $|B| \leq 2/3\,|W|$

So, basically the idea is that look here is your S and here is your; so let us. So, S here these are the vertices of W now our idea is that is it possible that look the w is small so we can guess which vertices, W has size 3k + 4. So, what we could do? So, guess W intersection is not a problem we can guess because this only 3k + 4. Now if we could partition W into W - S let us call this Y W-S into A and B such that mod A is at most say two third of W mod B is at most two third of W.

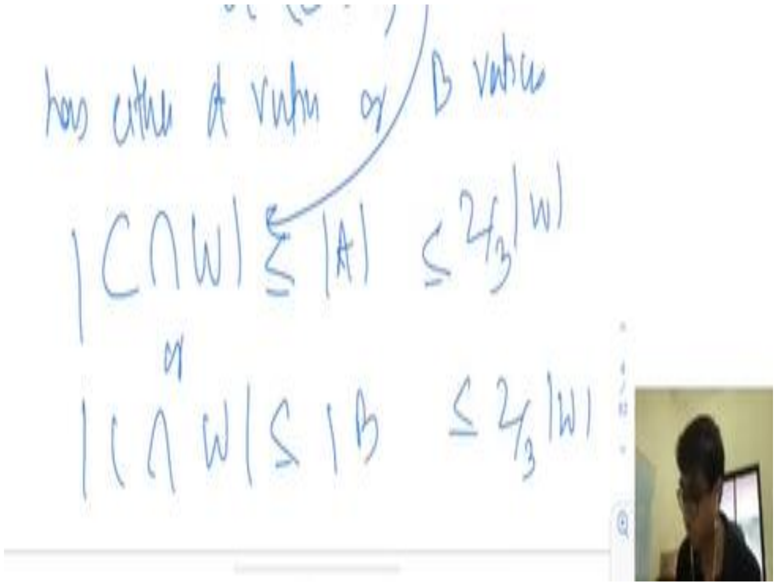
(Refer Slide Time: 10:06)



then find a separator of minimum size Z then it will guarantee that every component of $G \setminus (Z \cup Y)$ has either A vertices or B vertices

And then find a separator of minimum size such that this and this then find a then find a separator of minimum size. Then find a separator of minimum size then, it will guarantee that every component say minimum size say v that every component of G - Z union Y has either A vertices or B vertices.

(Refer Slide Time: 11:05)



has either A vertices or B vertices

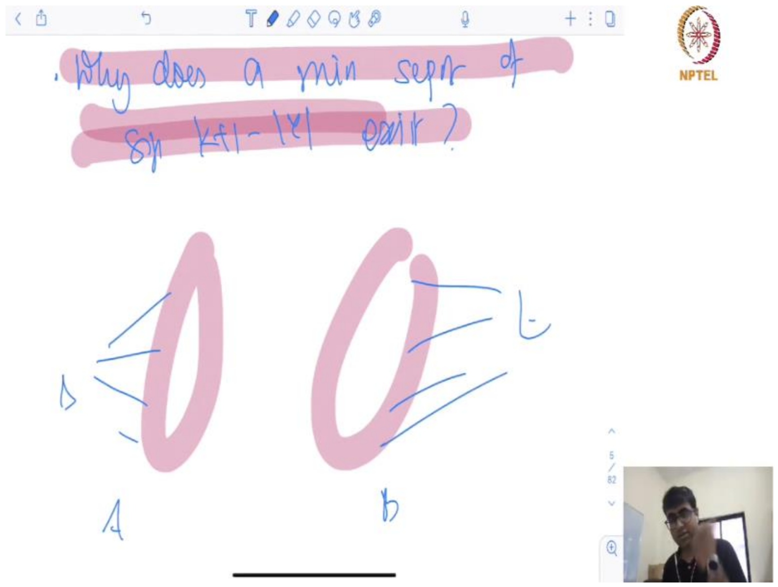
$$|C \cap W| \leq |A| \leq 2/3 |W|$$

or

$$|C \cap W| \leq |B| \leq 2/3 |W|$$

And what is the property? So, that implies that that C like any component here C intersection W is upper bounded by either A or C intersection W is upper bounded by B which is two third of W and it is at most two third of W.

(Refer Slide Time: 11:28)

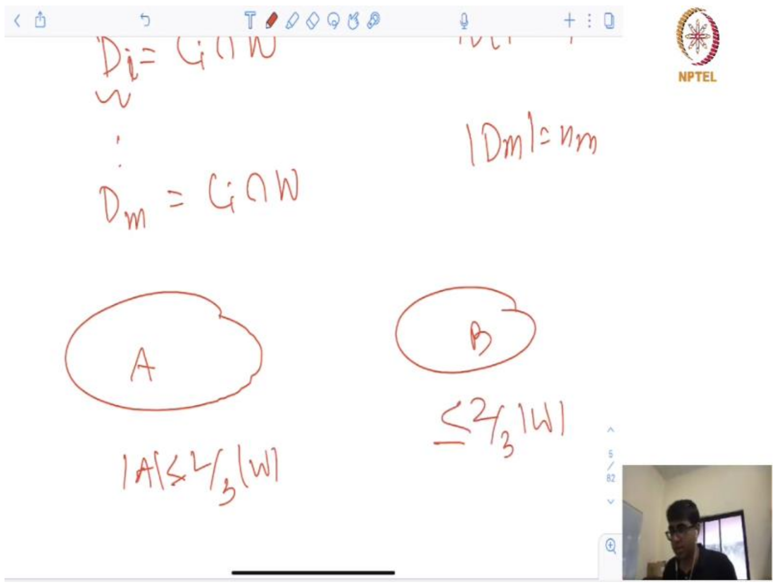


But the question arises. That is fine but why does a minimum separator of size k + 1 - Y exist not clear. This is true that a minimum separator will have the property that if I find an A and B grouping of W - S into A and B such that A is at most two third of W B is at most two third of W and then we find a minimum separator say Z. Them of course if I delete Z union Y then I know that no connected component will contain both vertices from A and B.

Because we have find a separator, we have been able to separate this and this is very easy to make such a S t separator problem because this is like your A this is like your B this is like A, B you just make S make adjacent to is and then like then you can make a S t separate. But more importantly why does the minimum separator of size k + 1 - Y exist. So, to do that what we actually can show is look at this S and look at this one.

Like we; will be able to group some of these patches into A and some of these patches into B. So, now if I can group some of these patches into A and some of these intersection into of B some of these intersection into B, then what we know? Then definitely the S which was the which was a half W by 2 separator it is also a separator of the kind which will separate A from B and that immediately implies that they just separate the mean separator has to have a size k + 1 – Y.

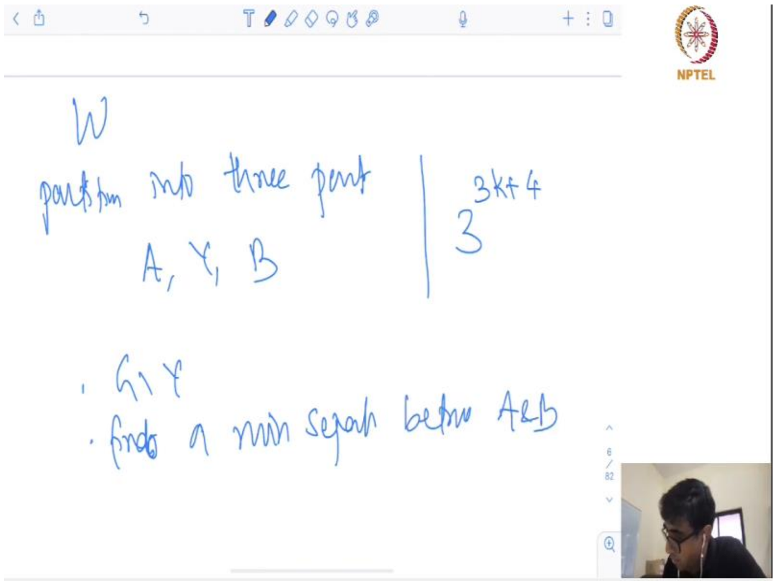
(Refer Slide Time: 13:33)



And last time we saw that look at these patches what we recalled it. So, we call these patches as let us call it x r x y is overly used so let us call these patches this is also overly use notation let us call it i. Let us call it D i is C i intersection W and if you recall correctly, we call this and suppose we had the size D 1 dot dot D m C i intersection. And suppose D i has side n 1 and D m has side n m and then we were able to show that.

Then we can find a partition into A and B such that mod A is at most two third of W and B is at most two third of W. So, we knew how to do this we saw it. So, this is exactly what we do so look I cannot so which implies that that we have such a good S if we have a such a good S then actually there exists a S star with the property that each of this component has at most two thirds and in fact this can be done by partition of W S into A and B of size at most two third of W and finding a main separator.

(Refer Slide Time: 15:11)


W

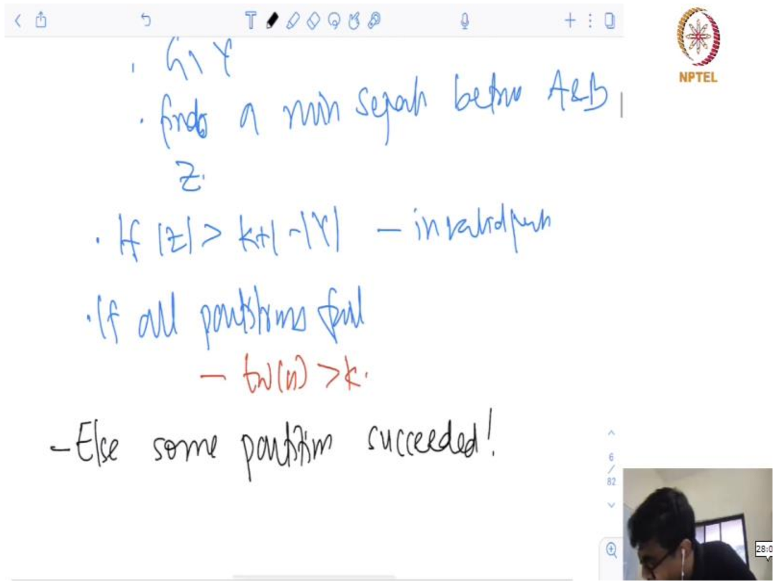
partition into three part A, Y, B | 3^{3k+4}

· $G \setminus Y$

· finds a min separator betw A&B

So, the algorithm is very simple so whatever algorithm does at any point of time, it takes W partitions into three parts A, Y and B and how many such partitions are there 3 to the power 3k + 4 and in G-Y, what it does? In G - Y finds a min separator between A and B and what did we call that we call that Z.

(Refer Slide Time: 16:03)


· $G \setminus Y$

· finds a min separator betw A&B

Z.

· If $|Z| > k+1 - |Y|$ — invalid partn

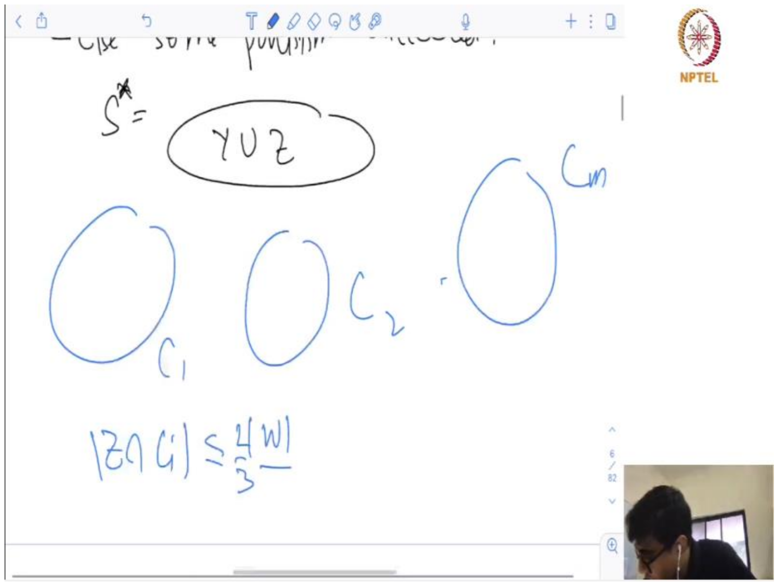
· If all partitions fail

— $tw(G) > k$.

– Else some partition succeeded!

Now if cardinality of Z is greater than k + 1 - Y invalid partition and if all partitions fail what we know about it if all partitions fail what do we know. Like we know that if the treewidth has at most k then there exists a good partition. So, if all partition fails, we can immediately conclude that treewidth of G is more than k. Else, some partition succeeded and even if one partition succeeded you can make a progress.

(Refer Slide Time: 16:51)

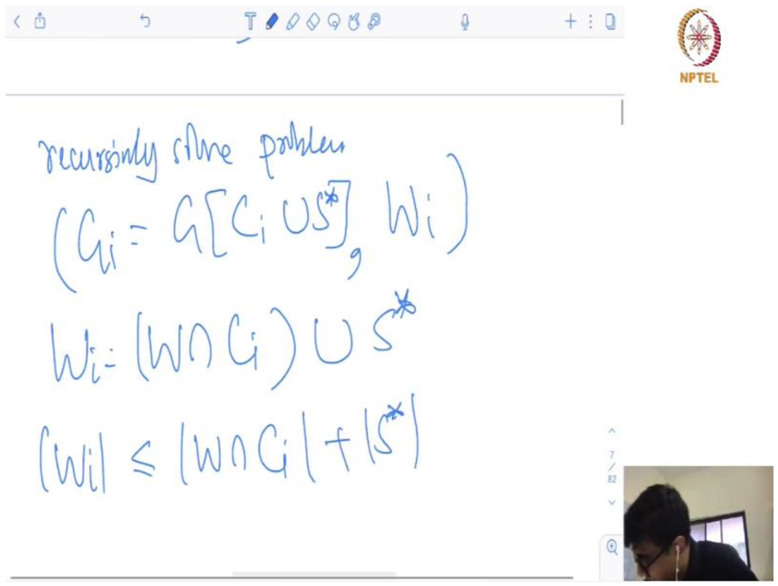


$S^* = \boxed{Y \cup Z}$

$|Z \cap C_i| \le \frac{2}{3}|W|$

So, how will we do this then you will look at Y union Z and this is your potential this is your S star and you look at your components C 1 C 2 C m and whatever guaranteed that Z intersection this C i is at most two third of W.

(Refer Slide Time: 17:20)



recursively solve problem

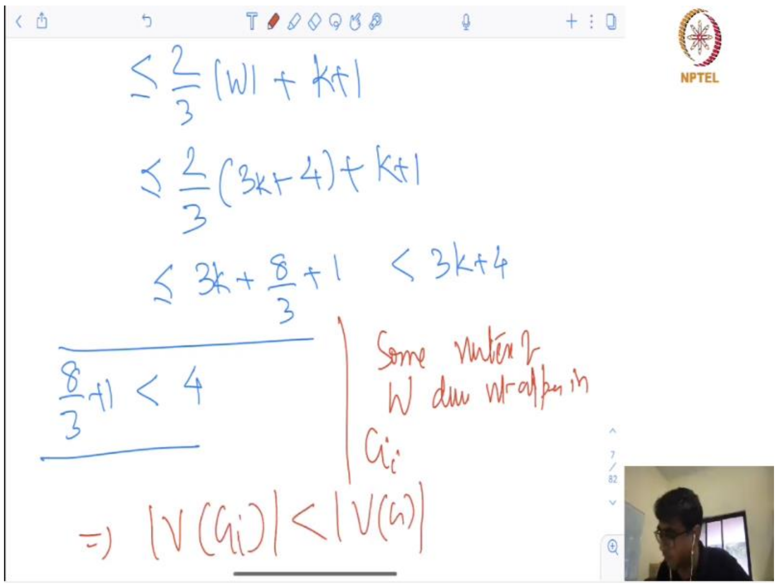
$(G_i = G[C_i \cup S^*], W_i)$

$W_i = (W \cap C_i) \cup S^*$

$|W_i| \le |W \cap C_i| + |S^*|$

So, what you do? You recursively solve problem on G i which is graph induced on C i union S star and you assign W i. So, what is W i? W i is W intersection C i union is star. Now what is the size of w i? Well of course the size of W i is plus S star.

(Refer Slide Time: 17:56)



$$\leq \frac{2}{3}|W| + k+1$$

$$\leq \frac{2}{3}(3k+4) + k+1$$

$$\leq 3k + \frac{8}{3} + 1 < 3k+4$$

$$\frac{8}{3}+1 < 4$$

Some vertex of W does not appear in G_i

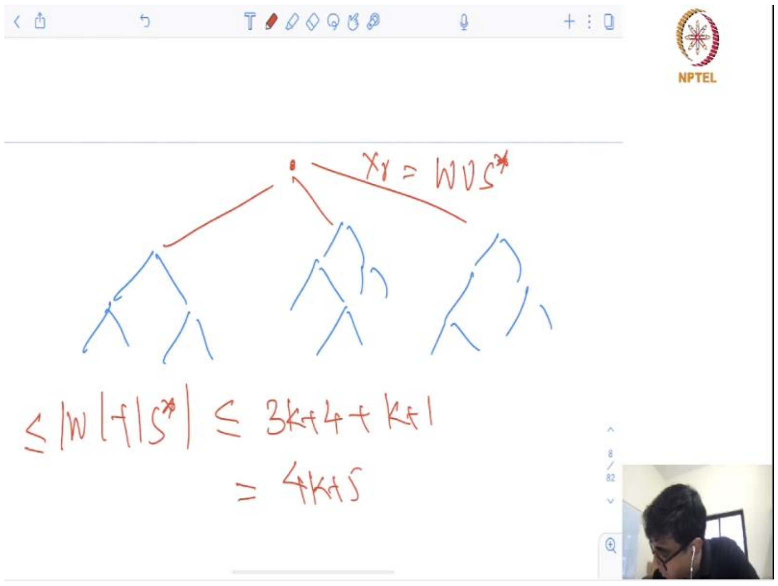
$$\Rightarrow |V(G_i)| < |V(G)|$$

This is at most two third of W size of S star is at most k + 1. Well so what is this? Two third of W which is like 3k + 4 + k + 1 which is 3k plus so what is this going to be? This is going to be 8 by 3 + 1 now 8 by 3 + 1, I claim is strictly less than 4 and this is easy to take. Because it is 8 by 3 is less than 9 so this is strictly less than 4 which implies this is strictly less than 3k + 4 now. So, this definitely implies that W i satisfy this inductive like induct recursive like recursive definition of the algorithm that the cardinality of W i has to be at most 4k + 4.

One thing which I missed out when trying to explain to you is that, this W at any stage if W size is less than 3k + 4 the arbitrary add vertices from outside to make it size 3k + 4 exactly. Now what are this implies? Two thing that look there it is some vertex of W some vertex this also implies that. Some vertex of W some vertex of W does not appear in G i which implies the cardinality of V G i is strictly less than V G.

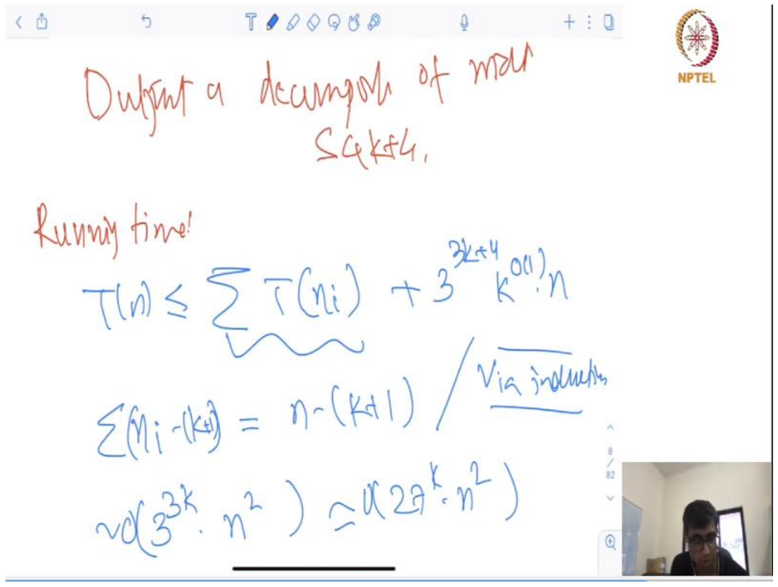
And hence we can we make some sort of progress. So, you recursively solve the problem on this smaller instance.

(Refer Slide Time: 20:50)



And you recall correctly once we got this small tree decomposition for all these components, we designed a tree decomposition of the whole graph by adding a root and making it adjacent to root of each of these guys and the what did we assign to X of r. So, we assigned X of r is nothing but W union star and this is where this is why the tree decomposition of this algorithm is upper bounded by cardinality W cardinality S star upper bounded by this which is 3k + 4 + k + 1 which is 3k + 4k + 5.

(Refer Slide Time: 21:39)


Output a decomposition of width $\leq 4k+4$.
Running time:
$T(n) \leq \sum T(n_i) + 3^{3k+4} k^{O(1)} \cdot n$
$\sum (n_i - (k+1)) = n-(k+1)$ / Via induction
$\sim O(3^{3k} \cdot n^2) \simeq O(27^k \cdot n^2)$

4k + 5 and hence we have been able to output decomposition of width at most 4k + 4 and then of course you could ask what about running time. So, well the running time of this algorithm is essentially if you notice is like T of n is like upper bounded by summation some T of n i like where that release plus some 3 to the power I would say 3k + 4 times some k to the power big of 1 times n and this is like you have to be a bit careful.

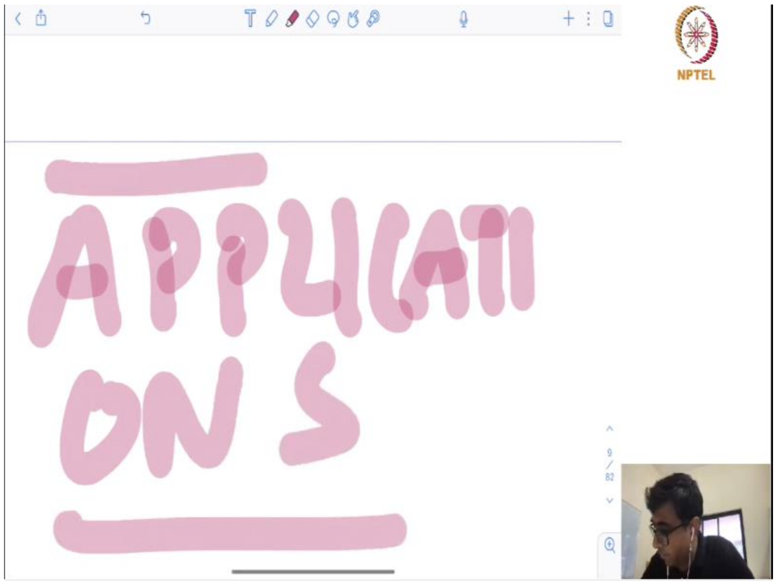
Because these are like, what is this a property? So, the property is that if you sum n i - k then it is equal to n - k + 1 actually n because in each of this component there is a k + 1 size which is common. So, this so you have to take max over all possible partition of this and by a simple induction you can show that this is actually an algorithm with running time 3 to the power 3k times n square. So, this will come to you 27 k big o of 27 k times n square.

So, but I will leave this this is via induction. So, I hope this algorithm is little bit more clear to you now that we have readed it. So, basically this is an algorithm which we look at this we looked at this existential algorithm or the existential proof that gave us this upper bounded by tree k but notice that when we went to the constructive, we lost another factor by k + 1 because existential we could guarantee that the W gets split very balanced at most half.

But algorithmically we could only get two third and there is no magic about two third, if you could only guarantee say even just slightly less than one. So, maybe say 4 over 5 then the factor of approximation will change but you still be able to get some C power k approximation algorithm. So, it will make progress even if the W is at most a constant fraction of what we started with in each of these things.

But then you have to make whatever the initial bag that you started with big enough. So, that this math holds. So, I hope all this is good.

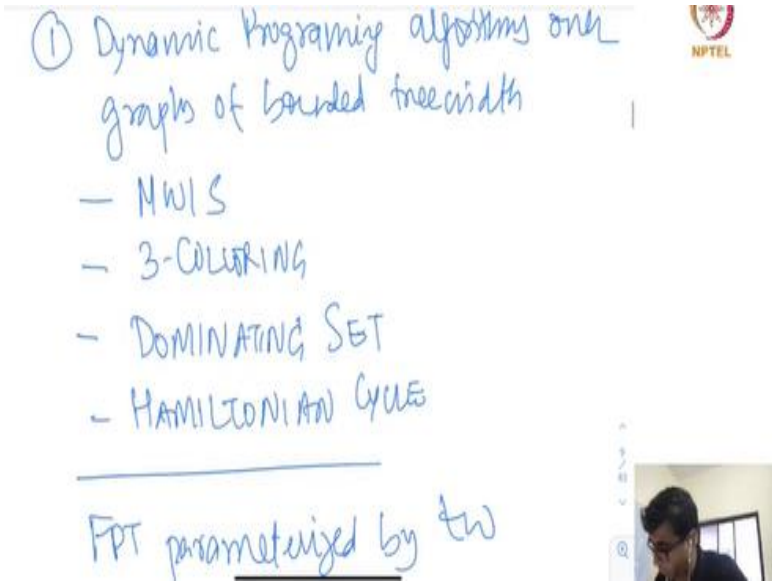
(Refer Slide Time: 24:37)



APPLICATIONS

And so now what I want to tell is some applications. So, now we will move to applications.

(Refer Slide Time: 24:58)

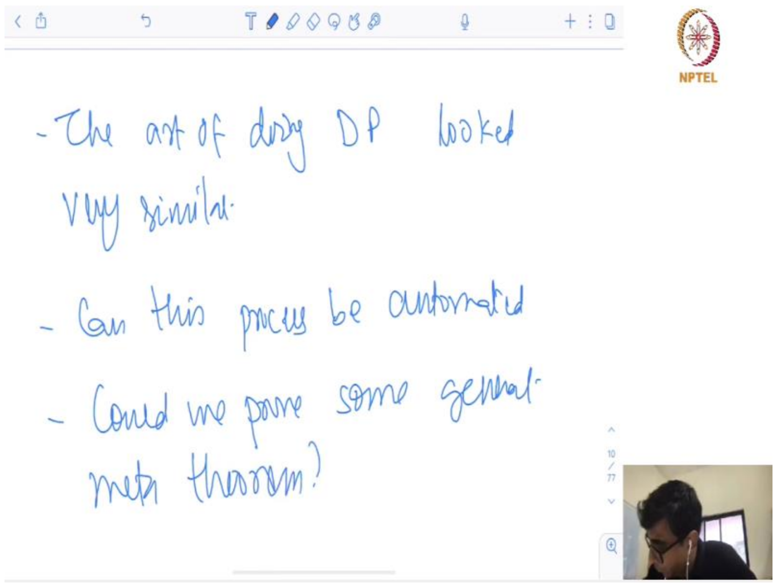


1. Dynamic Programming algorithms over graphs of bounded treewidth
   - MWIS
   - 3-COLOURING
   - DOMINATING SET
   - HAMILTONIAN CYCLE

FPT parameterized by tw

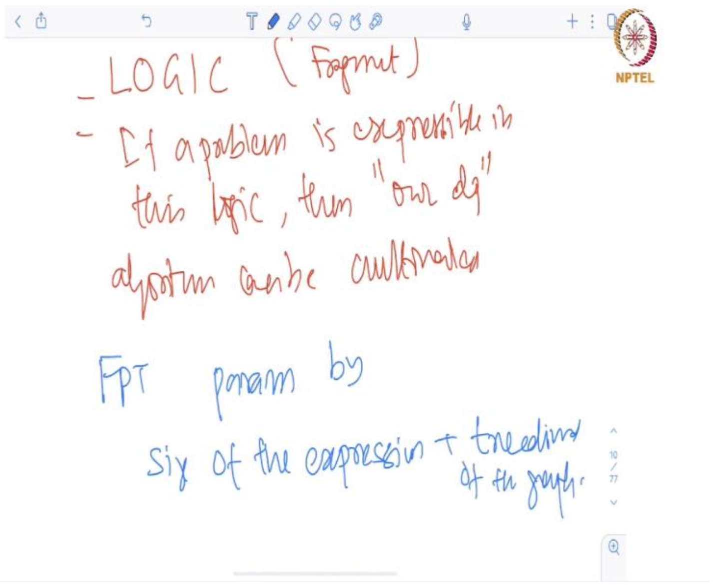
So, some applications we have already see, dynamic programming algorithms over graphs of bounded treewidth, so we saw that we saw an algorithm for max weight independent set 3 colouring, then we also saw an algorithm for dominating set. We saw an algorithm for Hamiltonian cycle and we showed that all these problems are FPT parameterized by treewidth. So, treewidth is an interesting object.

(Refer Slide Time: 26:04)


- The art of doing DP looked very similar.
- Can this process be automated
- Could we prove some general meta theorem?

But one natural question arises. Like the art of doing Dp looked very similar. So, a natural question was, can this process be automated? Can could be proved some general meta theorem?

(Refer Slide Time: 26:49)


- LOGIC (Fragment)
- If a problem is expressible in this logic, then "our dp" algorithm can be automated

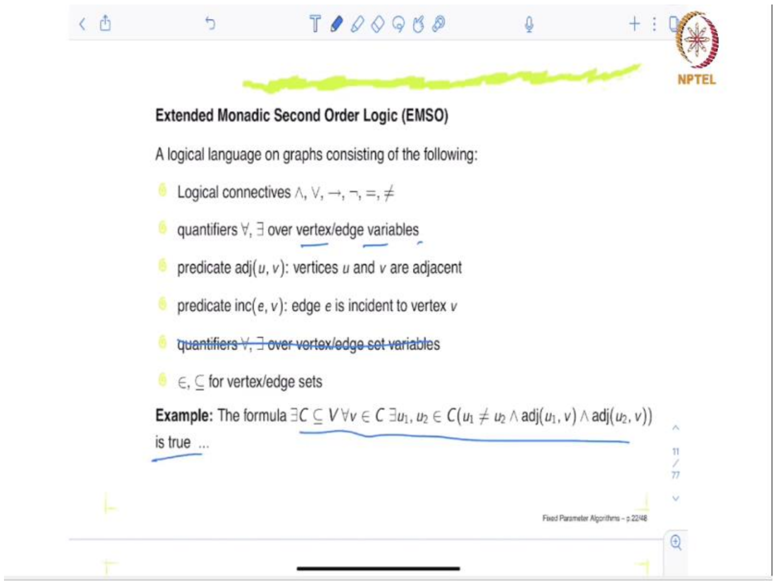
FPT param by

Size of the expression + treewidth of the graph.

And what I mean by could prove some general meta theorem? So, in these things what happens is that you say look generally these meta theorems are like so if this part of the like this next 10 minutes of lecture even if you do not follow completely, it is perfectly fine. So, we say could we find a fragment of logic fragment. Such that we can say if a problem is expressible in this logic, then our Dp algorithm can be automated.

Or rather if a problem is expressible in this logic, then I would like to say then the problem is FPT parameterized by what by size of the expression plus the treewidth of the graph.

(Refer Slide Time: 28:03)



Extended Monadic Second Order Logic (EMSO)

A logical language on graphs consisting of the following:

- Logical connectives $\wedge, \vee, \rightarrow, \neg, =, \neq$
- quantifiers $\forall, \exists$ over vertex/edge variables
- predicate $\text{adj}(u, v)$: vertices u and v are adjacent
- predicate $\text{inc}(e, v)$: edge e is incident to vertex v
- ~~quantifiers $\forall, \exists$ over vertex/edge set variables~~
- $\in, \subseteq$ for vertex/edge sets

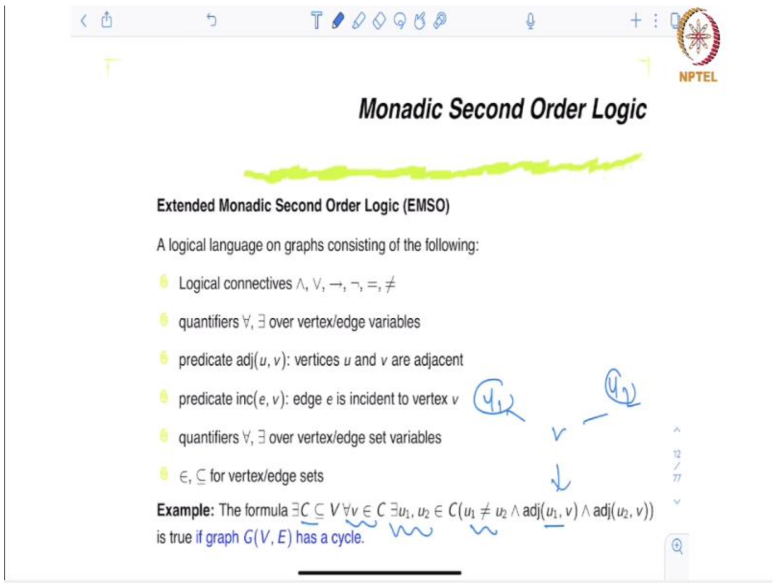
Example: The formula $\exists C \subseteq V \forall v \in C \exists u_1, u_2 \in C (u_1 \neq u_2 \wedge \text{adj}(u_1, v) \wedge \text{adj}(u_2, v))$ is true ...

Fixed Parameter Algorithms – p.22/48

And this was proved by Coureselle’s and this is also called CMSO 2 like or extended monadic secondary logic. So, this is a logical language and graph and which has following properties. So, it has some logical connective like and or implication not equal not equal. There are quantifiers over like for all that exists over vertex and edge variables. There is a predicate to check whether 2 u and v are adjacent they predicate to check whether some vertex is incident to some edge or not.

And there is also a quantifier for all the exist over vertex or x set variables I think this is just repeated. So, this is and we can also check something is an element or subset of some vertex at subsets. So, for example look at the formula that if you C subset of V for all v in C there exists u 1, u 2 in C such that u 1 is not equal to u 2. Adjacent of u 1, v and adjacent of v 2 is true what is the meaning of like what does this formula if a graph satisfies this formula what does this mean.

(Refer Slide Time: 29:14)



Monadic Second Order Logic

Extended Monadic Second Order Logic (EMSO)

A logical language on graphs consisting of the following:

- Logical connectives $\wedge, \vee, \rightarrow, \neg, =, \neq$
- quantifiers $\forall, \exists$ over vertex/edge variables
- predicate $\mathrm{adj}(u, v)$: vertices u and v are adjacent
- predicate $\mathrm{inc}(e, v)$: edge e is incident to vertex v
- quantifiers $\forall, \exists$ over vertex/edge set variables
- $\in, \subseteq$ for vertex/edge sets

Example: The formula $\exists C \subseteq V \, \forall v \in C \, \exists u_1, u_2 \in C (u_1 \neq u_2 \wedge \mathrm{adj}(u_1, v) \wedge \mathrm{adj}(u_2, v))$ is true if graph $G(V, E)$ has a cycle.

It will mean that if a graph G V, E is true only if graph has a cycle. Why? Because what are we saying C there exists C subset like who will satisfy this formula. So, only those graphs will satisfy this formula if it has a cycle. So, what are we seeing look C is a subset of V for all v in cycle for all there is just two other vertices in C which are not equal to 2 not equal and u and v are v adjacent to u 1.

And so basically, I looked at V I say look for every vertex I can find two other vertices to which it is neighbour 2. So, that will only happen if a graph G has a cycle so for example this.