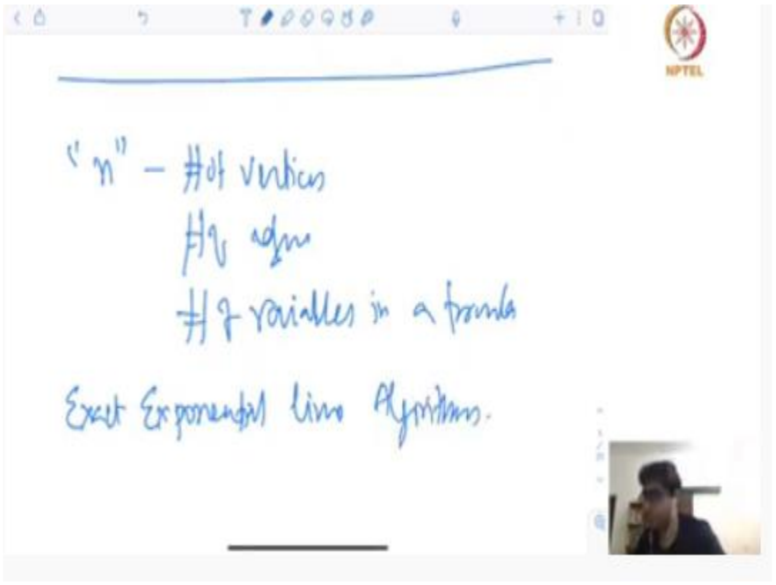
Parameterized Algorithms

Prof. Neeldhara Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Gandhinagar
Prof. Saket Saurabh
Department of Theoretical Computer Science
Institute of Mathematical Science, Chennai

Lecture - 41
Algebraic Algorithms Matrix Multiplications

And we will talk about algebraic algorithms up until now, these how to use inclusion exclusion. Today I will give you another algorithm for matrix multiplications. And our parameters still will be slightly nonconventional.

(Refer Slide Time: 00:35)

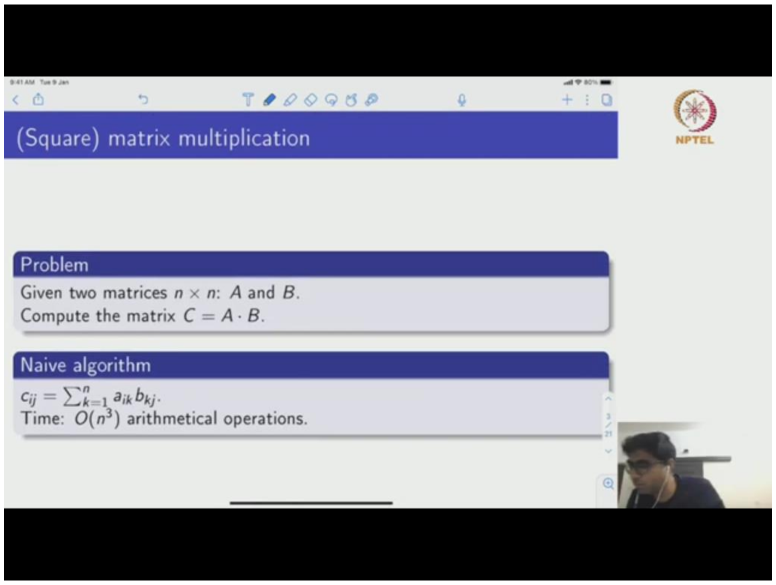


"n" – # of vertices
# of edges
# of variables in a formula

Exact Exponential time Algorithms.

This is also known this whenever use this n kind of parameter like n being number of vertices or number of edges or number of variables in a formula. Then all these things are slightly nonconventional. And this is also termed as exact exponential time algorithms. And these have been around for more time like from 70s and other. So, today we are going to talk about an algorithm based on matrix multiplication and this slide is from quite early slide by Lukasz but I will use that.

(Refer Slide Time: 01:21)



(Square) matrix multiplication

Problem

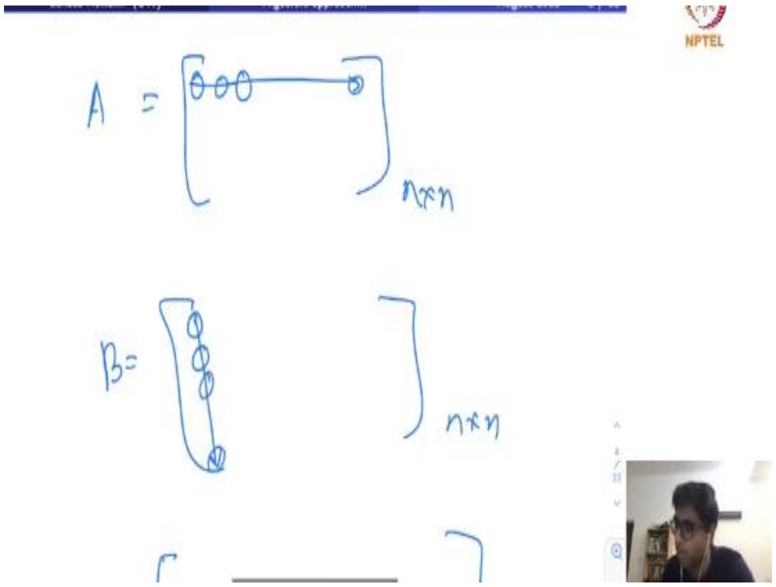
Given two matrices $n \times n$: A and B.
Compute the matrix $C = A \cdot B$.

Naive algorithm

$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$.
Time: $O(n^3)$ arithmetical operations.

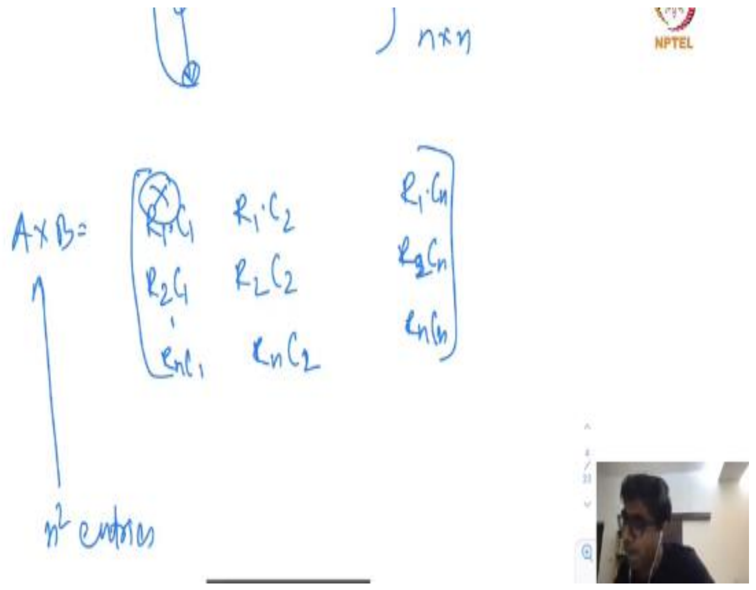
So, we are going to talk about square matrix multiplications, you are given 2 matrices n cross n, A and B. And you want to compute the matrix C = A cross B, so that is very simple. The classical algorithm takes n cube time because all you have to do is that you are given A and B.

(Refer Slide Time: 01:48)



So, you are given A and n times n matrix and B another n times n matrix and all you are interested is in A times B. So, what do you do? So, what is the first entry basically first entry, you look at the first row of A, multiply it with the first column, so this entry will get multiplied with this, this entry gets multiplied with this, this entry gets multiplied by a third entry, so on and so forth.

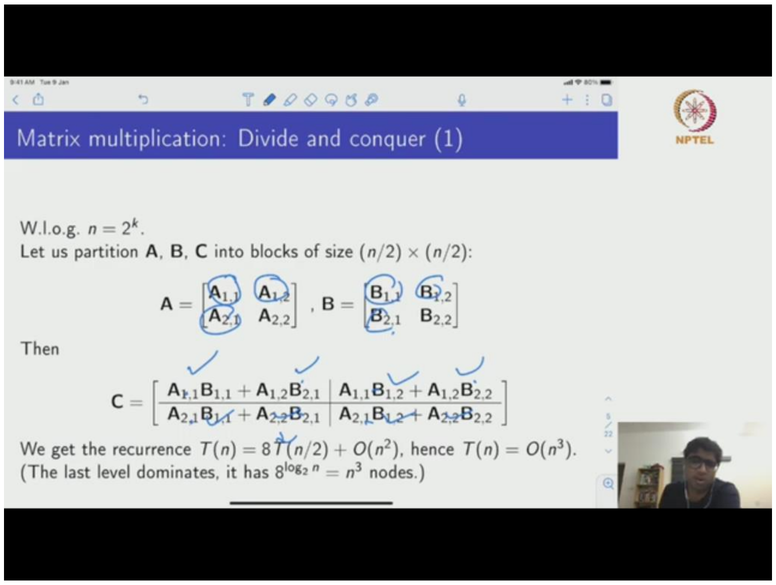
(Refer Slide Time: 02:17)



So, to get 1 entry here, you need basically it is a dot dot product of, if you think of a dot dot product of, so if this is say, let us call this row 1 to row n for this one. And let us call this column 1 to column n, so what is the first 1 is basically a dot product of of R 1 C 1 and what your second entry is basically dot product of of R 1 C 2 dot product of R 1 dot C n and second is R 2 C 1 R 2 C 2 R 2 C n dot dot dot R n C 1 R n C 2 R n C n. Now so A cross B also has n square entries. And each dot product is nothing but n multiplication.

So, overall in cube time, you should be able to do this. So, this is basically what is C ij? It is like a ik b kj. This is basically so, your total number of n cube arithmetic operations so, you can do square matrix multiplication n cube arithmetic operation. So, this is a classical algorithm.

(Refer Slide Time: 03:28)



Matrix multiplication: Divide and conquer (1)

W.l.o.g. $n = 2^k$.
Let us partition **A**, **B**, **C** into blocks of size $(n/2) \times (n/2)$:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}$$

Then

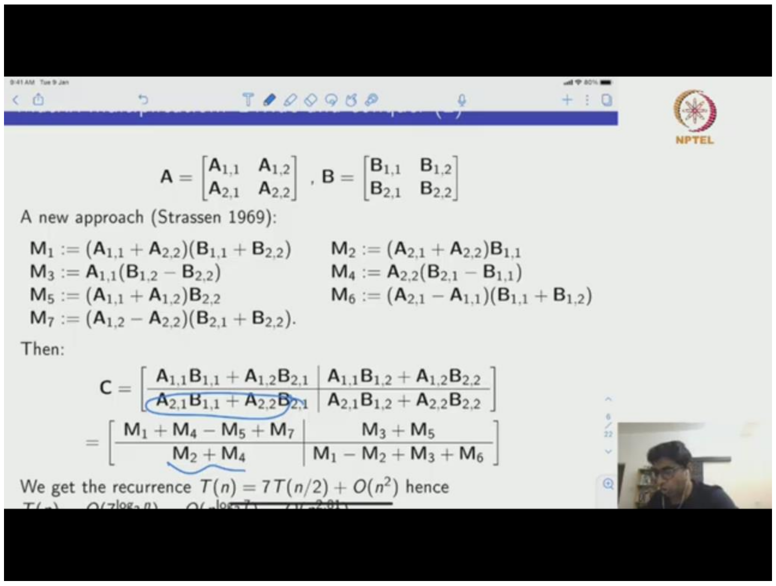
$$\mathbf{C} = \left[\begin{array}{c|c} \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} & \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \hline \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} & \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{array}\right]$$

We get the recurrence $T(n) = 8T(n/2) + O(n^2)$, hence $T(n) = O(n^3)$.
(The last level dominates, it has $8^{\log_2 n} = n^3$ nodes.)

And, but there is also so, there is a very simple divide and conquer algorithm. So, suppose you like you take n = 2 power k without loss of generality so, you divide A, B, C into blocks of size n / 2 cross n / 2 cross this is n / 2 times n / 2 so on and so forth. So, what is C 1? So, you take this multiply with this plus you take this and multiply this and so on and so forth. So, we get the recurrence that if you are trying to solve multiply n cross n matrix then this is 812348 T n of T plus order n squared.

And hence T of n is order n cubed because the last level dominates it has the roughly n cubed nodes. So, how many multiplications we have? 12345678 and that is what corresponds to this 8. So, like 8 big multiplication of objects you will get and from there you have just had to sum them up, so once you have got this multiplication, then it is nothing but just normal summation, which is like order n squared time. So, but what is the time consuming is that this 8 big matrix multiplication of matrices of size n / 2 times n / 2.

(Refer Slide Time: 04:51)



$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, \quad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

A new approach (Strassen 1969):

$M_1 := (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$ $\quad M_2 := (A_{2,1} + A_{2,2})B_{1,1}$

$M_3 := A_{1,1}(B_{1,2} - B_{2,2})$ $\quad M_4 := A_{2,2}(B_{2,1} - B_{1,1})$

$M_5 := (A_{1,1} + A_{1,2})B_{2,2}$ $\quad M_6 := (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$

$M_7 := (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$.

Then:

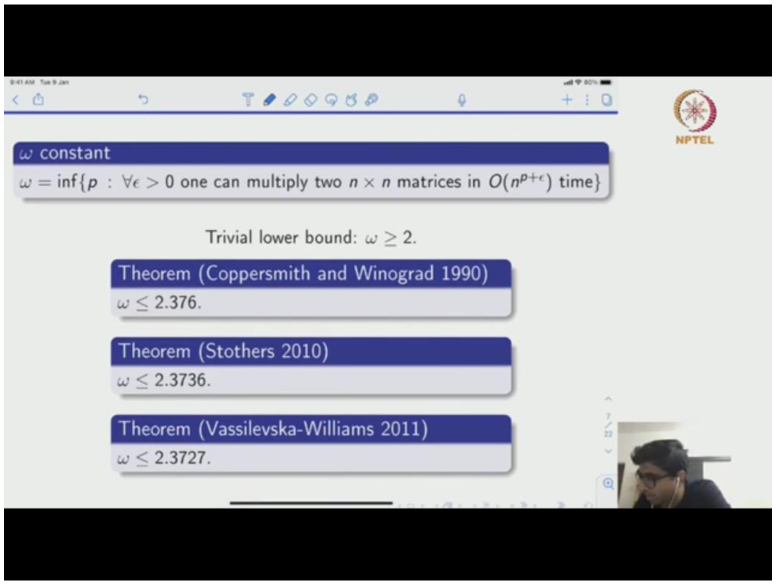
$$C = \left[\begin{array}{c|c} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ \hline A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{array}\right] = \left[\begin{array}{c|c} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ \hline M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{array}\right]$$

We get the recurrence $T(n) = 7T(n/2) + O(n^2)$ hence

But if you I am sure you have seen this in your classical algorithm course also this approached by Strassen. So, what is approached by Strassen is hey look, if you really want to get this multiply this matrix A, B, then we do not need to compute 8 matrix multiplication, but we could compute some 7 matrix multiplication and then the product matrix C can be written as sum or addition of this dot product of matrix.

So, we came of this very nice 7 matrix multiplication of size n / 2 times n / 2 times n / 2 times of the matrices of order n / 2 times n / 2 and then he was able to write look at for example, what is this term here? It is nothing but some of M 2 + M 4 so on and so forth. But now he has reduced the matrix multiplication do not from 8 to 7 matrix multiplication of matrices of order n / 2 each. So, that implies that T of n is 7 T of n / 2 plus order n square and so the running time became n to the power 2.81.

(Refer Slide Time: 06:07)



ω constant

$\omega = \inf\{p : \forall \epsilon > 0$ one can multiply two $n \times n$ matrices in $O(n^{p+\epsilon})$ time$\}$

Trivial lower bound: $\omega \geq 2$.

Theorem (Coppersmith and Winograd 1990)

$\omega \leq 2.376$.

Theorem (Stothers 2010)

$\omega \leq 2.3736$.

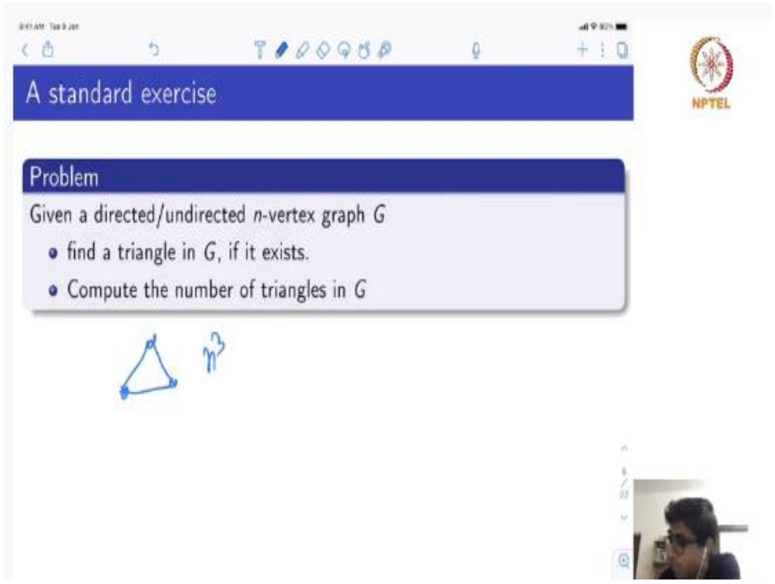
Theorem (Vassilevska-Williams 2011)

$\omega \leq 2.3727$.

And let omega be the constant to multiply the best matrix multiplication. So, there is a trivial lower bound of omega at least 2 because you have to look at each entry of this matrix is and that is like n squared entry. So, Coppersmith and Winograd obtained that you can do it using some nontrivial application, omega is 2.376 then Stothers obtained slightly improved one and in 2011 Williams got 2.3727 this is even more smaller currently, this is not current.

So, this number is slightly better than what you see in the slides. But the moral of the story is that you can do matrix multiplication much better like much closer to than n cube. And now we will exploit this algorithmic development about matrix multiplication to design an algorithm.

(Refer Slide Time: 07:02)



A standard exercise

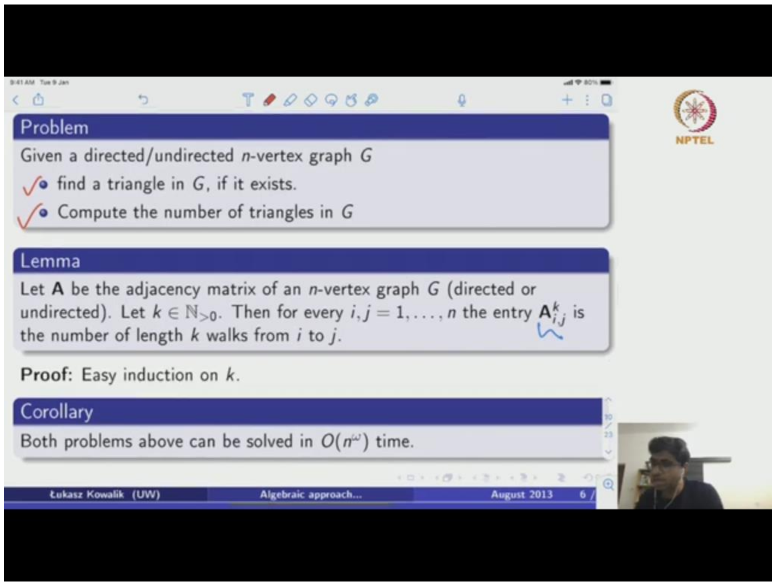
Problem

Given a directed/undirected n-vertex graph G

- find a triangle in G, if it exists.
- Compute the number of triangles in G

So, this is the standard exercise that you are given a directed or undirected in vertex graph G you want to find a triangle in G if it exists compute the number of triangles in G, all this other very standard exercise. One thing is trivial like you can n cube algorithm just try all possible triplets and do n cube.

(Refer Slide Time: 07:27)



Problem

Given a directed/undirected n-vertex graph G

- find a triangle in G, if it exists.
- Compute the number of triangles in G

Lemma

Let $\mathbf{A}$ be the adjacency matrix of an n-vertex graph G (directed or undirected). Let $k \in \mathbb{N}_{>0}$. Then for every $i, j = 1, \ldots, n$ the entry $\mathbf{A}^k_{i,j}$ is the number of length k walks from i to j.

Proof: Easy induction on k.

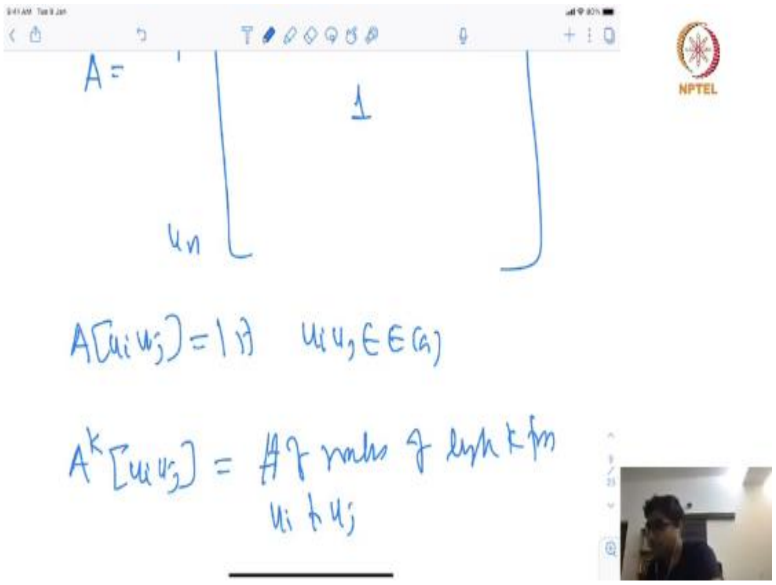
Corollary

Both problems above can be solved in $O(n^\omega)$ time.

Łukasz Kowalik (UW) Algebraic approach... August 2013 6

But there are slightly better algorithm let be the adjacency matrix of an N vertex graph G directed or undirected. And you fix any k then for every ij 1 to n, if you look at the kth power and look at so, basically what it says you? So, what it says you that look at the adjacency matrix of a graph G.

(Refer Slide Time: 08:00)



$A =$ u_n 1

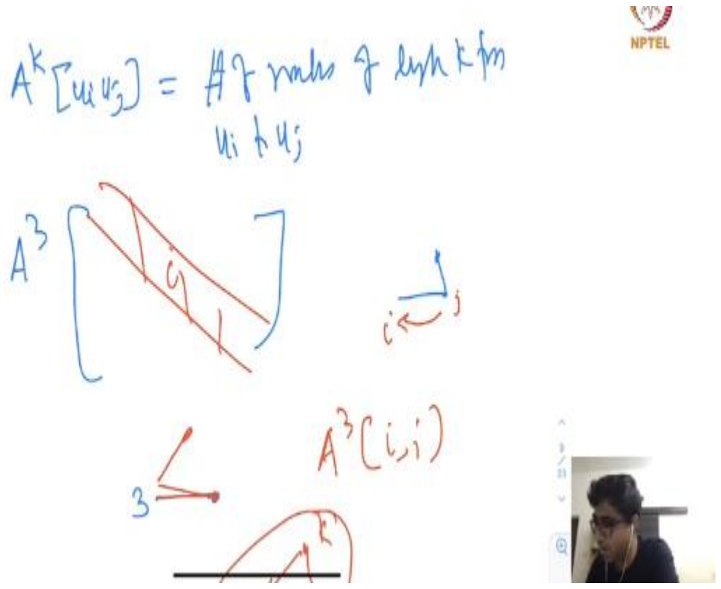
$A[u_i u_j] = 1$ if $u_i u_j \in E(G)$

$A^k[u_i u_j] = \#$ of walks of length k from u_i to u_j

So, what is an adjacency matrix of graph G? A so you have and what is A ui uj is 1, if ui uj is an edge. Then you put 1 here 0 otherwise, so, this is the adjacency matrix and if you look at the power A to the power k, kth power of A and you look at this ui uj or maybe or i comma j then this is nothing but it counts number of walks of length k from ui to uj. Or you can also do dynamic programming for all that, but like this is what it can do.

And you can prove this using induction because so this is what it means is easy induction on k, but what is the corollary? Can we find your triangle in G if it exists?

(Refer Slide Time: 09:25)



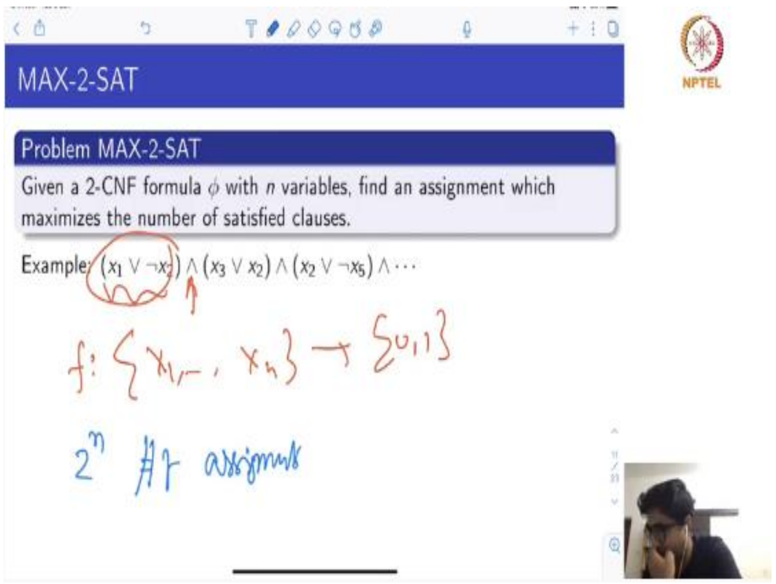
Well, if you go back here and look at A 3 and look at diagonal entries. So, diagonal entries are what those was works with support, this is like A 3, it starts at 3, it goes somewhere and it has to come back and it is length 3 walk. So, if it is a length 3 walk, suppose I start at vertex 3, I go somewhere. So, what will it can consist of I can go here but notice if I come back to 3, then it cannot be a walk. And then if I look at so, all the closed walk of length 3 which let us I think this is not a good. So, look at a vertex i and look at a walk of length 3.

So, I started i go at j some other vertex because in the first because there are no self loose, but now where can I go? I can either come back to i but then if I go out that is not a walk which starts at i and n. So, what are we looking for? We are looking for A 3, i like those walks which starts at i ends at i. So, basically what happens at A 3 i, i is that it is basically I started i I visit some other vertex j I have to visit some other vertex k and then I come back.

But then they see a triangle. So, then this is a triangle. So, basically a graph is a triangle if and only if in A to the power 3 like third power of A has non zero entry. And in fact, what it is so, basically if you wanted to count the number of triangles and you can count all that you can sum the trace. Then you can sum the trace and then you know that like and then you have to divide by some number like 3 factorial because you will be counting the same cycles of several times.

So, you just do you sum this up and divide appropriately. So, in n the power omega time we can find a triangle in G if we take this, we can compute the number of triangles, in G effects like so, we can do this task in n to the power omega time.

(Refer Slide Time: 11:56)



MAX-2-SAT

Problem MAX-2-SAT

Given a 2-CNF formula ϕ with n variables, find an assignment which maximizes the number of satisfied clauses.

Example: $(x_1 \vee \neg x_2) \wedge (x_3 \vee x_2) \wedge (x_2 \vee \neg x_5) \wedge \cdots$

$f : \{x_1, \ldots, x_n\} \to \{0,1\}$

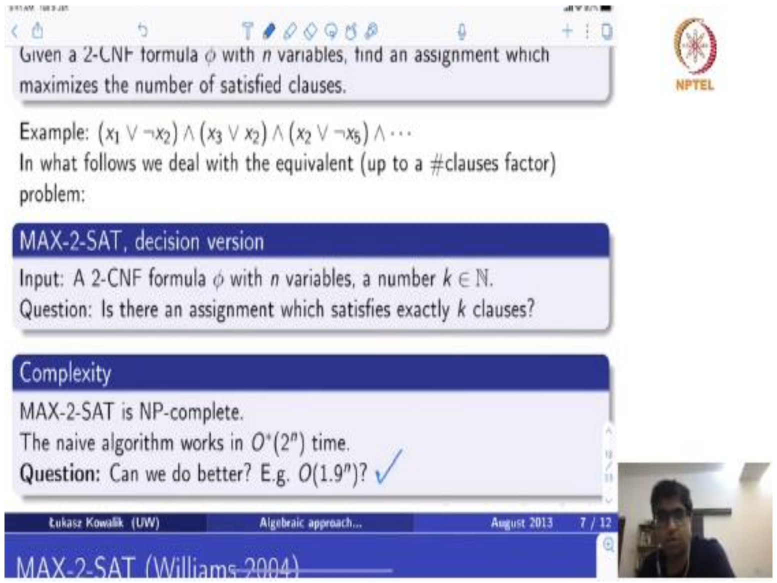
2^n # assignments

So, the problem we are interested in is the following. So, you are given a 2 CNF formula. So, with n variables, and we want to find an assignment which maximizes the number of satisfied clauses. So, remember, it is a 2 set formula in the sense that you have n variables and each clause consists of just 2 variables x 1 or x 2, x 3 or x 2, x 2 or x naught x 5 so on and so forth. So, what is CNF? It is hands of odds and what is the type, what is a satisfying assignment?

So, you want to find a function from say V 1 to V n, if you have n variables, that is. Say x 1 to x n to 0 1 and when a clause is satisfied? A clause is satisfied if so, for example x 1 naught x 2, so x 1 if assigned 1 then this clause is satisfied. So, if any one of the literals is set to 1 then the clause is satisfied. Are you have if you set a set x = 0, then naught of x 2 is 1 so this clause is satisfied. So, you want to find an assignment which satisfies the maximum number of clauses.

And for 2 set formulas you can check whether there is an assignment that satisfies all the clauses in polynomial time, but finding an assignment that satisfies maximum number of clauses is not easy.

(Refer Slide Time: 13:30)



Given a 2-CNF formula ϕ with n variables, find an assignment which maximizes the number of satisfied clauses.

Example: $(x_1 \vee \neg x_2) \wedge (x_3 \vee x_2) \wedge (x_2 \vee \neg x_5) \wedge \cdots$
In what follows we deal with the equivalent (up to a #clauses factor) problem:

MAX-2-SAT, decision version

Input: A 2-CNF formula ϕ with n variables, a number $k \in \mathbb{N}$.
Question: Is there an assignment which satisfies exactly k clauses?

Complexity

MAX-2-SAT is NP-complete.
The naive algorithm works in $O^*(2^n)$ time.
Question: Can we do better? E.g. $O(1.9^n)$? ✓

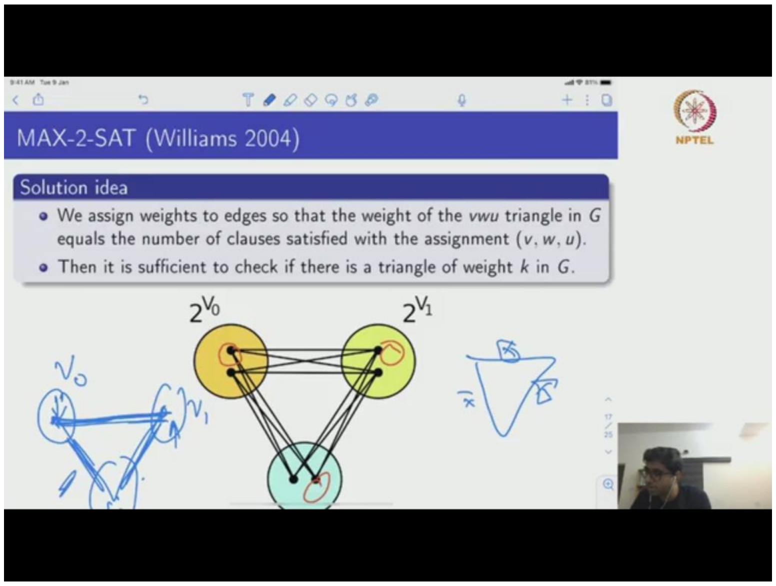
Łukasz Kowalik (UW) Algebraic approach... August 2013 7 / 12

MAX-2-SAT (Williams 2004)

So, what are we interested in the decision version the problem is that phi with n variables and a number k and we are interested in is there an assignment which satisfies exactly k clauses. And because, if you wanted to find the maximum number of satisfying assignment then you can start from 1, 2, 3, 4 and you can go up to the maximum since the number of clauses are like n square. What do you mean up to a clause factor in the sense that no clauses are repeated for example.

So, is there an assignment to satisfy the exactly k clauses, of course MAX 2 SAT is NP complete. So, naive algorithm will definitely work in order 2 power n time, what you try all possible assignments of x 1. So anyway, how many such numbers of functions are there? So, such numbers of functions are actually at most 2 to the power n number of assignments are there each x i could take 0 and 1. So, the total number of assignments is to power n. So, the very naive question was that is it possible to design an algorithm which is better than 2 power n, can we come up with an algorithm running time say 1.9 to the power n.

(Refer Slide Time: 14:57)

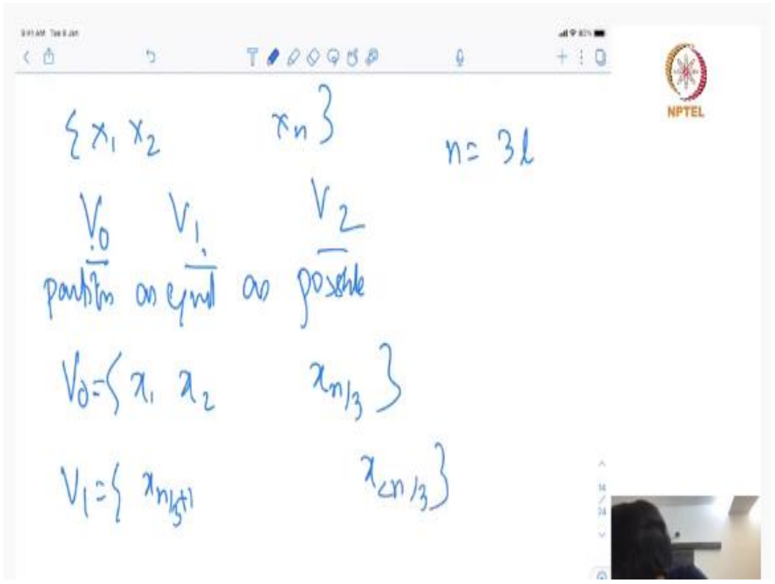

MAX-2-SAT (Williams 2004)

Solution idea

- We assign weights to edges so that the weight of the *vwu* triangle in *G* equals the number of clauses satisfied with the assignment (v, w, u).
- Then it is sufficient to check if there is a triangle of weight *k* in *G*.

And William came up with some very beautiful algorithm. And he used in this like he constructed an auxiliary graph and for that he used matrix multiplication. So, what it did is the following. So, before we let me give you an overview of his algorithm and then we will see. So, what he did is the following?

(Refer Slide Time: 15:24)


$\{x_1\ x_2 \quad x_n\}$ $\quad n = 3l$

$V_0 \quad V_1 \quad V_2$

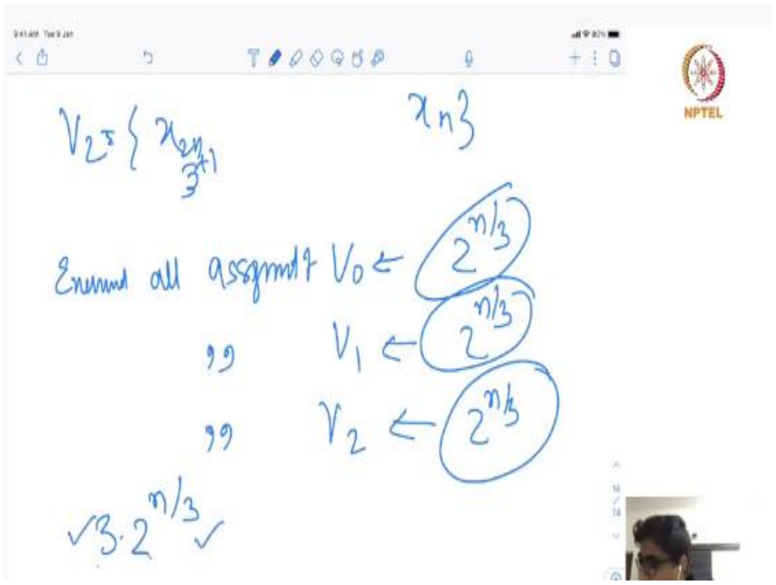
partition as equal as possible

$V_0 = \{x_1\ x_2 \quad x_{n/3}\}$

$V_1 = \{x_{n/3+1} \quad x_{2n/3}\}$

So, suppose you had this x 1 x 2 x n. So, he first partitions this into V 0 V 1 V 2 partition as equal as possible. So, suppose if n would have been say some 3k then V 0 V 1 V 2 are not tricky, let us not use because that is like number of 3l then V 0 V 1 and V 2 will be of equal size and otherwise you just you can always come up with a sets which are like roughly like 1 or 2 elements here and there. So, you have come up with this partition. Now, what is his idea? The idea is that look at this V 0, V 1 and V 2, V 0 is like say suppose V 0 consists of x 1 x 2 x n / 3 variables, V 1 consist of x n / 3 + 1, x to n / 3 variables.

(Refer Slide Time: 16:30)



$V_2 = \{ x_{\frac{2n}{3}+1}$ $x_n \}$

Enumd all assgnmt $V_0 \leftarrow 2^{n/3}$

" $V_1 \leftarrow 2^{n/3}$

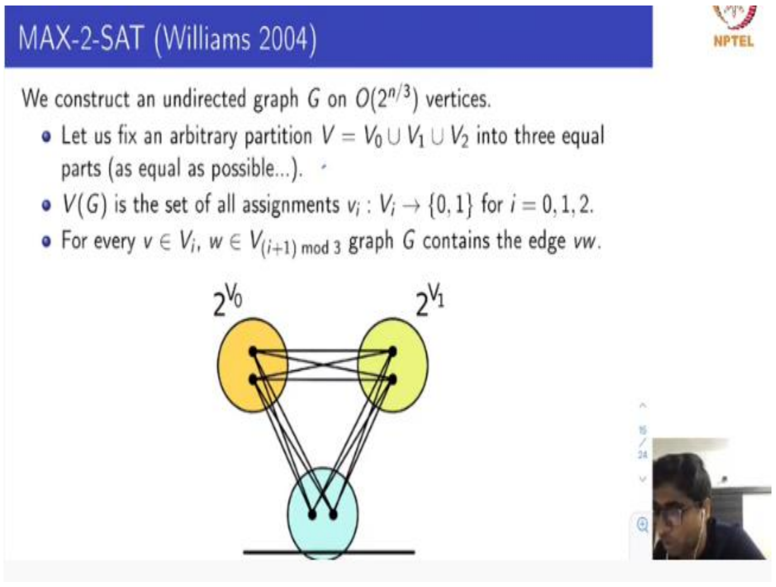
" $V_2 \leftarrow 2^{n/3}$

$\checkmark 3 \cdot 2^{n/3} \checkmark$

And V 2 consists of x 2 n / 3 + 1, x can vary. So, he says that, let us enumerate all the assignments of V 0. So, what are the possible assignments are V 0 x 1 can take 0 1, x 2 could take 0 1, x n 3 could take 0 1. So, enumerate all assignment of V 0 how many of them will be there 2 to the power n / 3 enumerate all assignments of you and how many of them are there 2 to the power n / 3 enumerate all assignments of V 2 how many of them are 2 to the power n / 2. So, in total we have enumerated how many assignments 3 to the power 2 to the power n over 3.

And now, each of these assignments are going to form like each assignment will correspond to a vertex in our graph. So, total number of vertices in our graph will be 3 into 2 to the power n / 3 and this is what is happening.

(Refer Slide Time: 17:35)



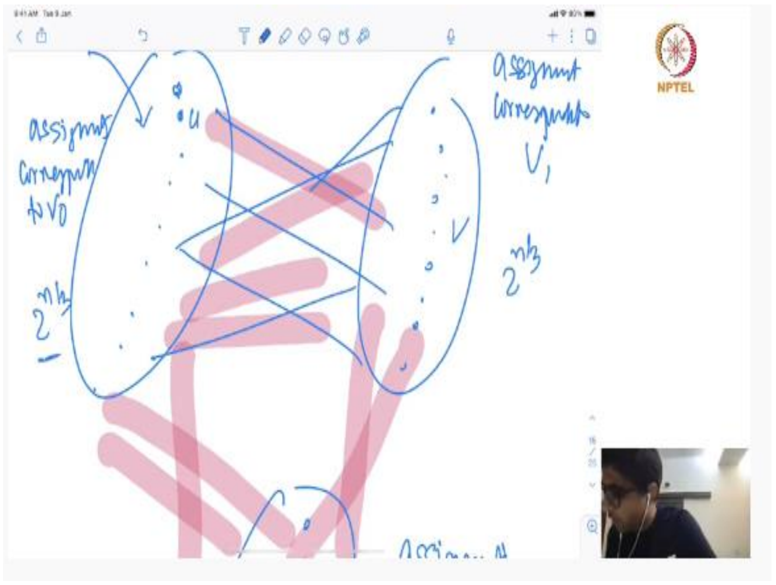
MAX-2-SAT (Williams 2004)

We construct an undirected graph G on $O(2^{n/3})$ vertices.

- Let us fix an arbitrary partition $V = V_0 \cup V_1 \cup V_2$ into three equal parts (as equal as possible...).
- $V(G)$ is the set of all assignments $v_i : V_i \to \{0, 1\}$ for $i = 0, 1, 2$.
- For every $v \in V_i$, $w \in V_{(i+1) \bmod 3}$ graph G contains the edge vw.

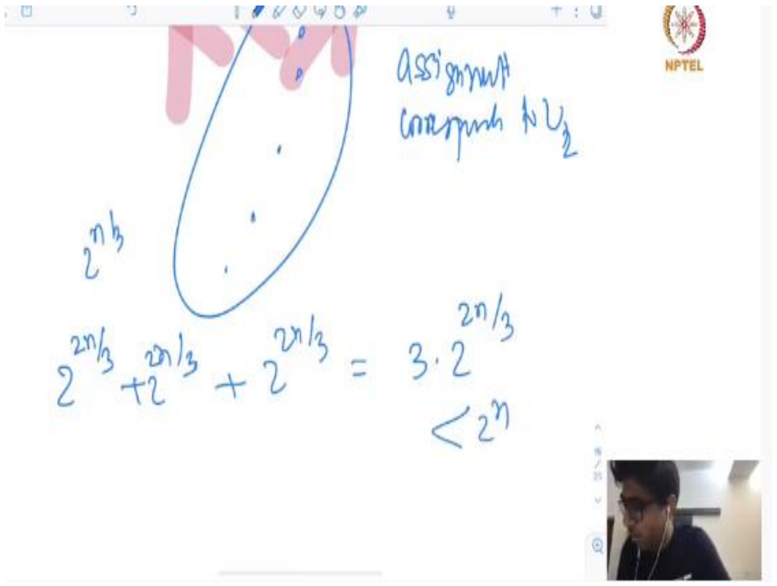
So, you fix an arbitrary partition V 0, V 1 into 3 equal parts and what is the vertex the set of all possible assignments V i from V i to 0 1 for i = 0 1. Now how do you give an edge? So, giving an edge is very simple. So, you put an edge between every assignment; so now if you look at this.

(Refer Slide Time: 18:03)



assignments corresponding to V_0
u
$2^{n/3}$
assignment corresponding V_1
v
$2^{n/3}$

(Refer Slide Time: 18:30)


assignment correspondnt to V_2
$2^{n/3}$
$2^{2n/3}+2^{2n/3}+2^{2n/3} = 3\cdot 2^{2n/3}$
$< 2^n$
NPTEL

So, this is let us say assignments corresponding to V 0 this is assignment corresponding to V 1 and this is assignment corresponding to the V 2. So, you have vertices and basically these are like vertices these are vertices. So, there are 2 power n over 3 of them, 2 power n over 3 of them, 2 power n over 3 of them and now what I do? You make a complete bipartite graph. So, for every vertex u and v here you put an edge. So, this is a complete bipartite graph. So, you give all the edges from here to here, here to here and here to here.

So, you ask yourself how many edges are here. So, there are 2 power n / 3 vertices there are 2 power n / 3 vertices, all possible edges. So, this is like 2 power 2 n / 3 fine. How many edges are between V 1 and V 2 another 2 power 2 n / 3 very good between V 2 and V 0 how many, another 2 power n / 3. So, you have roughly 3 times 2 but still this is strictly less than 2 n numbers of vertices number of edges is strictly less than 2 power n numbers of edges in our clause.

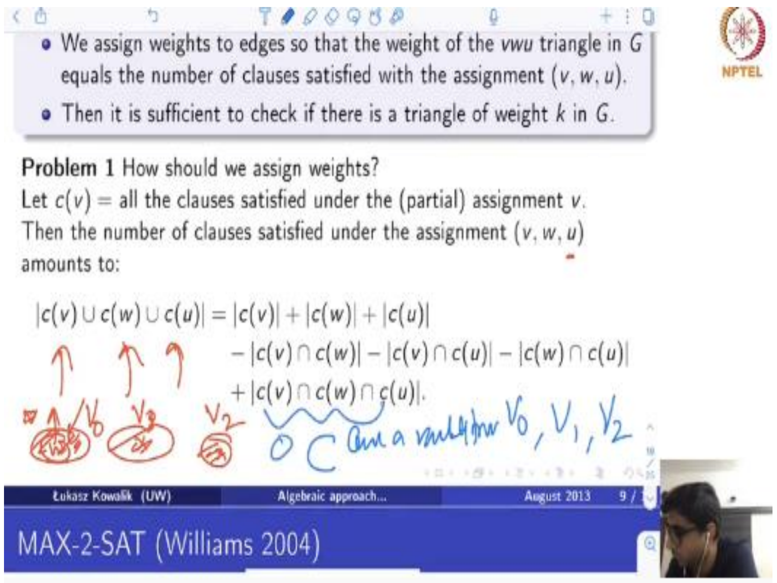
So, this is how you make your graph. So, this is this is how you make so you have all the assignment corresponding to V 0, all the assignment responding to V 1, all the assignment responding to V 2 and you make a complete bipartite graph between V 0 V 1, V 1 V 2, V 2 V 3. Now, in this graph we would like to compute a triangle and say we are great. But to do that, our goal was to check if there is a triangle.

So, we want, so now the idea of the solution is that we assign weights to edges, so the weight of the vwu triangle in G equals the number of clauses satisfy the assignment v w u what I mean by this? Now, if you pick up a vertex here, let us pick up a vertex here. You let us pick up a vertex here, let us pick up a vertex here. So, if I take a partial assignment here, a partial assignment here and the partial assignment here, then that corresponds to an assignment of all the variables.

Now, that corresponds to assignment of all the variables and I would like to assign edges to this particular triangle, because you know you take up this triangle that it is saying as a number of clauses satisfied by these 3 variables. This concatenation of assignments, so this is what I would like to do. And then it is sufficient to check if there is a triangle of weight k in G or not, so if you did like to do these 2 things.

So, remember, if I pick up an vertex here, pick up a vertex here and I pick up a vertex here then take this, this and this then tells me number of assignments in G, that tells that fix in an assignment of my variables and that immediately tells me if once I fixed number of clauses that they could satisfy.

(Refer Slide Time: 22:16)



- We assign weights to edges so that the weight of the *vwu* triangle in *G* equals the number of clauses satisfied with the assignment (v, w, u).
- Then it is sufficient to check if there is a triangle of weight *k* in *G*.

Problem 1 How should we assign weights?
Let $c(v)$ = all the clauses satisfied under the (partial) assignment v.
Then the number of clauses satisfied under the assignment (v, w, u) amounts to:

$$|c(v) \cup c(w) \cup c(u)| = |c(v)| + |c(w)| + |c(u)| - |c(v) \cap c(w)| - |c(v) \cap c(u)| - |c(w) \cap c(u)| + |c(v) \cap c(w) \cap c(u)|.$$

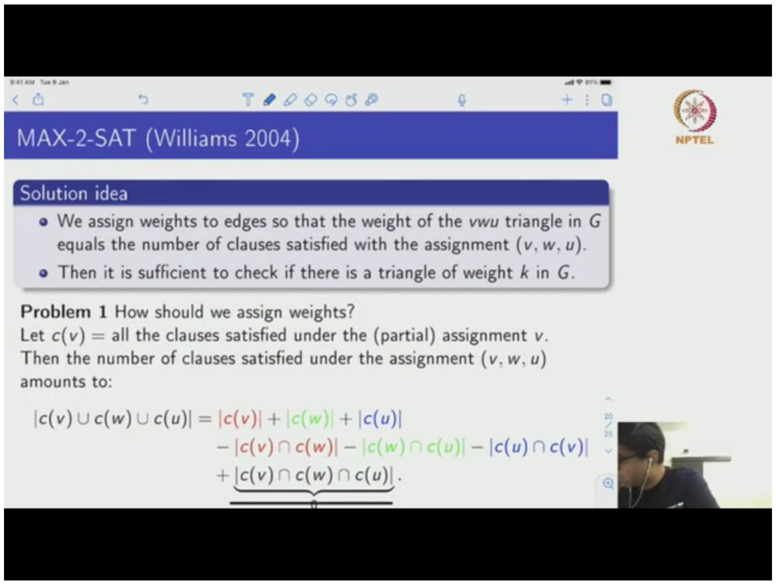
So far so good, so, first let us solve the first one, how should we assign weights? Let us see all the clauses satisfied under the possible assignment v then a number of satisfies. So, these are like these are the some sets the number of clauses satisfied under the assignment v w u amounts to we have just seen it with an exclusion. So, like cv is all the clauses. So, set cv we are interested in this, but now maybe some clauses are satisfied both by v and w. So, cv cw cu you now subtract those clauses.

And then you know because of inclusion exclusion, you have to also add cv intersection cw union cu. But imagine yourself that these are disjoint. So, it is so, look at a clause which is also in the set cv it means it is satisfied by some variable v like some variable it contains some variable w which is contained inside V 0 and I look at some set w like some assignment w it means this clause is satisfied by some variable which is in V 1 and then it also belongs to see you it means it is like, so this is another set of variables.

So, basically what I mean to this to say this look at v correspond to a partial assignments of V 0 w corresponds to partial assignment of V 1, u correspond to partial assignment on V 2. And now fix a partial assignment and is a look at these are the clauses which are these are the clauses which are satisfied a partial assignment v. Now let us fix a clause C that belongs to cv cw and cu union.

It means the clause c must contain a variable from the C contains a variable from V 0, C contains a variable from V 1, C contains a variable from V 2 but that cannot happen because it has to say so this contributes to 0, so this contributes to 0.

(Refer Slide Time: 24:49)



MAX-2-SAT (Williams 2004)

Solution idea

- We assign weights to edges so that the weight of the vwu triangle in G equals the number of clauses satisfied with the assignment (v, w, u).
- Then it is sufficient to check if there is a triangle of weight k in G.

Problem 1 How should we assign weights?
Let $c(v)$ = all the clauses satisfied under the (partial) assignment v.
Then the number of clauses satisfied under the assignment (v, w, u) amounts to:

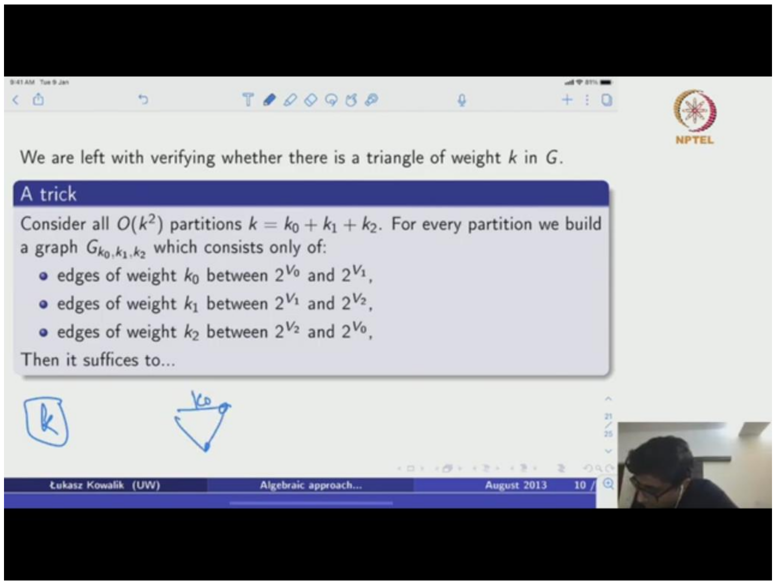
$$\begin{aligned}|c(v)\cup c(w)\cup c(u)| = {} & |c(v)| + |c(w)| + |c(u)| \\ & - |c(v)\cap c(w)| - |c(w)\cap c(u)| - |c(u)\cap c(v)| \\ & + \underbrace{|c(v)\cap c(w)\cap c(u)|}_{0}.\end{aligned}$$

So, this is not at all an important factor for us. So, for us cv union cw union cu, it just this one then this is easy. Then there is very easy assignment u. So, cv so you asked and there are 3 edges. So, for edge v w u assign like you fix v. So, for you fix v w and you assigned edge cv - cv intersection cw. So, basically whichever is the largest index you assign the value of that and you subtract that is it.

So, this is how you can so, you put the weight x y is cx - cx intersection cy. This is how you put it. So, we can think of this as a right. So, now look at this triangle. So, for an edge here, you look for an edge which starts at here. So, think of this way; that you will always give importance to the V 0 if you look at V 1, you will look at an edge between V 1 and V 2. So, for edge between V 0 and V 1 between you will assign the intersection weight as well as the weight of this for edges here we will assign the intersection weight as well weight.

And the edges here assign the intersection weight of this vertex and edge. So, this is the convention you make and assign weight this. Now you know that if so, what would you have been able to achieve? So, we are able to achieve that now if you look at any triangle in this huge graph, look at any triangle and look at the sum of the weights of the edges, then that is exactly equal to the number of clauses satisfied by this. Now we do some preprocessing, what kind of preprocessing we do is as follows.

(Refer Slide Time: 27:08)


We are left with verifying whether there is a triangle of weight k in G.

A trick

Consider all $O(k^2)$ partitions $k = k_0 + k_1 + k_2$. For every partition we build a graph G_{k_0,k_1,k_2} which consists only of:

- edges of weight k_0 between 2^{V_0} and 2^{V_1},
- edges of weight k_1 between 2^{V_1} and 2^{V_2},
- edges of weight k_2 between 2^{V_2} and 2^{V_0},

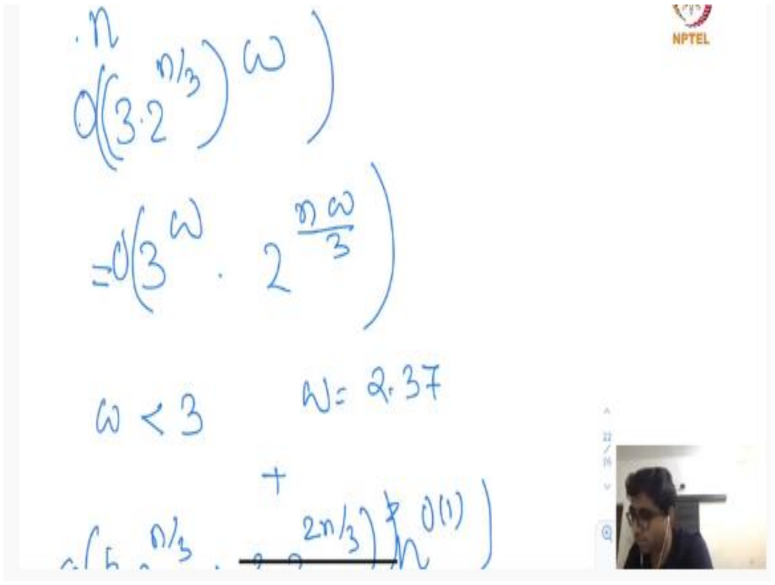
Then it suffices to...

We say, fine. Look, at if I am looking for weight exactly k then let us look at this triangle may be I want the triangle should look like weight k 0 here, k 1 here, k 2 here. So that this is equal to k, how many such partitions are there at most order k squared, such partitions are there because once you fix k 0 and k 1 k 2 gets fixed, because k is fixed. So, for every partition, you build the graph G k 0 k 1 k 2 which consists only of what property edge of weight k 0 between this.

So, now I say look, I am only interested in finding a triangle contains edge weight k 0 between V 0 and V 1, weight of edge between V 1 and V 2 is k 1 and weight of an edge between a vertex V 1, V 2 and V 0 is k. So, what do you do to look at your big graph? You look at your big graph and from here, you delete all the edges whose value is not equal to k 0, you delete all the edges from here whose value is not equal to k 1, you delete all the edges from this.

So, now what you know, now the property is that in it, then it suffices to check whether there is a triangle at all. Because if there is a triangle in our graph, then the wave we have any triangle must contain an edge, because it is a bipartite graph any triangle must contain a vertex from V 0, V 1 and V 2 and then it must contain an edge of weight k 0, must contain an edge of weigh k 1, must contain an age of weigh k 2. So, if you sum them up, then this is going to be equal to k. So, now, what is our algorithm? So, in this graph G k 0 k 1 k 2 graph, we need to check whether there is a triangle or not. So, we have an algorithm.

(Refer Slide Time: 29:08)



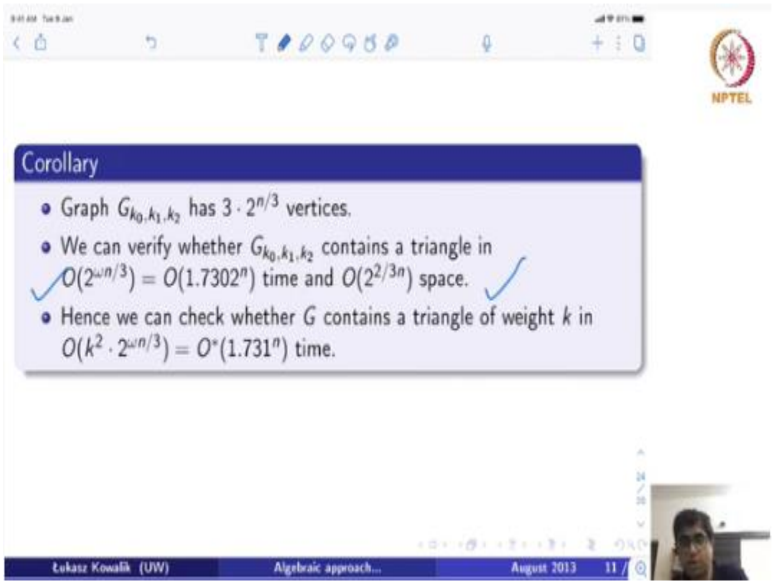
$O((3\cdot 2^{n/3})^{\omega})$
$= O(3^{\omega} \cdot 2^{\frac{n\omega}{3}})$
$\omega < 3$
$\omega = 2.37$
+

We saw an algorithm to test whether we have a triangle or not in time n to the power omega. Where n is the number of vertices. So, in our graph, how many vertices are there? So, the number of vertices are there is. So, the running time of our algorithm will be this. So, this is equal to if you notice, of course it is some O here, 2 to the power n omega divided by 3. Now we know that omega is less than, strictly less than 3. In fact, omega is like roughly like 2.3 or something 2.37.

So, now till omega is a constant in this is less than strictly less than n. So, yeah so the running time of this but we also had to construct our graph so, to construct our graph how much running time it takes it takes running time we had 3 times 2 n over 3 vertices and then you have to give him edges. So, the total number of edges are 3 times 2 to n / 3. So, this is an some O of this is an extra cost of constructing this graph of course, you also will have some polynomial factors to take care of k.

But the whole total running time actually boils down to constructing this graph and then running this polynomial matrix multiplication algorithm on this. And this can be done very efficiently as we have seen. So, this would have been a better than 2 power algorithm for if we had a matrix multiplication algorithm as close to anything which is better than 37. So, if we can get an algorithm with running time say for example, for matrix multiplication which is of n square time, then this algorithm will run in time 2 to the power 2 n / 3. But currently, it is a matrix multiplication running time which dominates this algorithm.

(Refer Slide Time: 31:32)



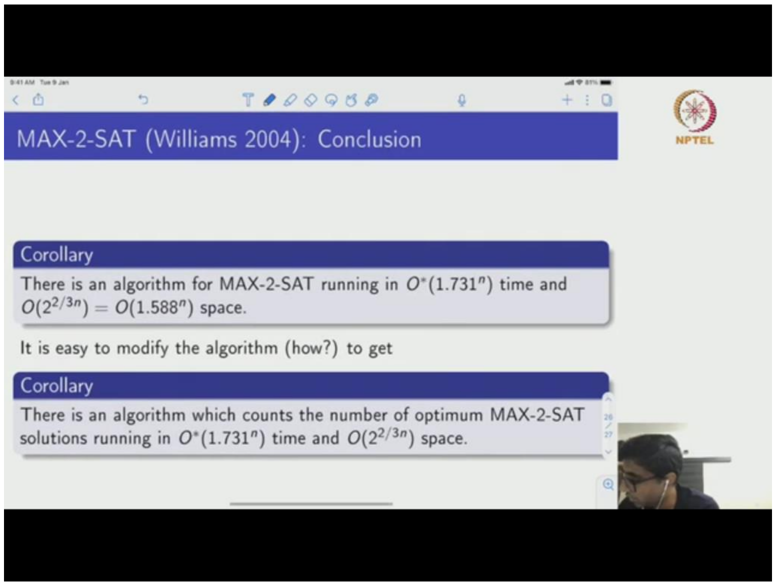
Corollary

- Graph G_{k_0,k_1,k_2} has $3 \cdot 2^{n/3}$ vertices.
- We can verify whether G_{k_0,k_1,k_2} contains a triangle in $O(2^{\omega n/3}) = O(1.7302^n)$ time and $O(2^{2/3n})$ space.
- Hence we can check whether G contains a triangle of weight k in $O(k^2 \cdot 2^{\omega n/3}) = O^*(1.731^n)$ time.

Łukasz Kowalik (UW) Algebraic approach... August 2013 11

So, given all this had this many, but he said we can verify contains a triangle in this on time or not in this much running time. And this much space because you also have to take a big graph and hence we can check with a G contains a triangle of weight k in time k square 2 to the power 2 omega n / 3 because you have to take about all possible values of k. So, in this much running time, we can actually check whether like find the maximum number of satisfying assignment.

(Refer Slide Time: 32:13)



MAX-2-SAT (Williams 2004): Conclusion

Corollary

There is an algorithm for MAX-2-SAT running in $O^*(1.731^n)$ time and $O(2^{2/3n}) = O(1.588^n)$ space.

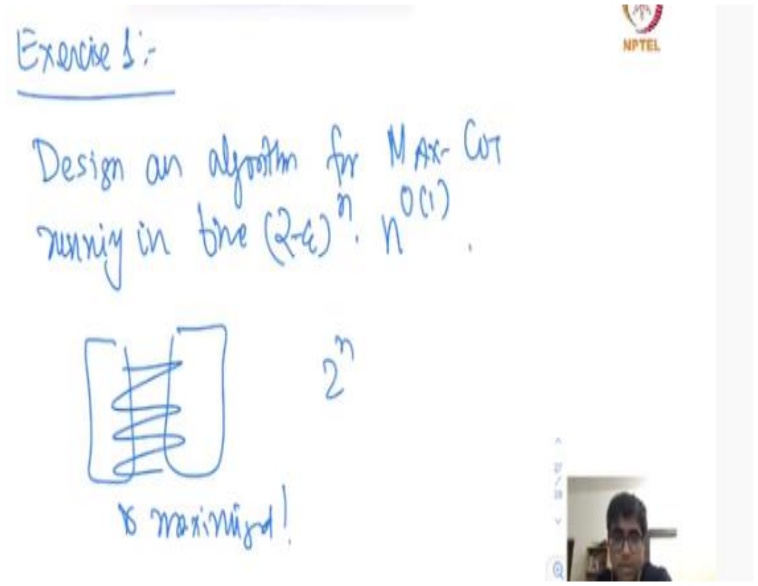
It is easy to modify the algorithm (how?) to get

Corollary

There is an algorithm which counts the number of optimum MAX-2-SAT solutions running in $O^*(1.731^n)$ time and $O(2^{2/3n})$ space.

So, what we learned that there is an algorithm for MAX-2-SAT running in this much time and this much space, in fact you can also count the number of this and here is an exercise which I would like to gift to you. Starting from here.

(Refer Slide Time: 32:30)



Exercise 1:-

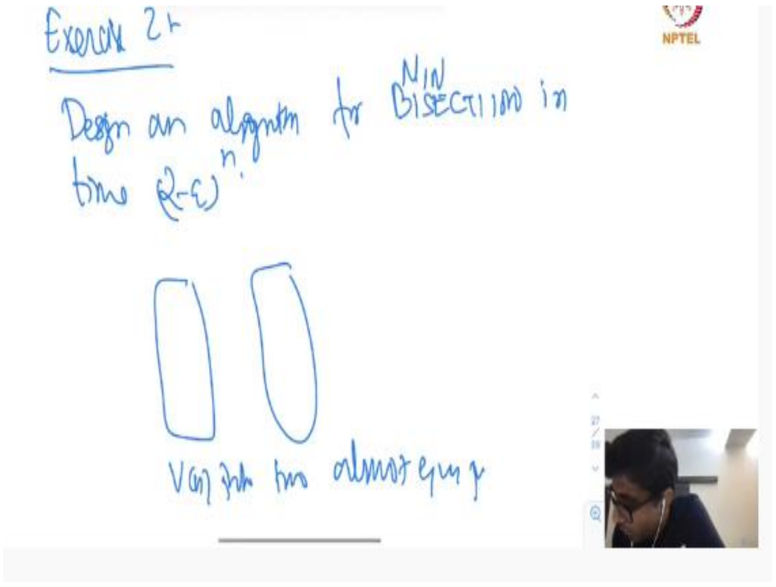
Design an algorithm for MAX-CUT running in time $(2-\epsilon)^n \cdot n^{O(1)}$.

2^n

is maximized!

There are some exercises that you could try design an algorithm for MAX-CUT running in time 2 - epsilon to the power n, n to the power O of 1, for that is the first algorithm, exercise. So, you have to design such an algorithm for MAX-CUT. So, what is the MAX-CUT? You want to find the partition of vertex set of graph? So, the number of edges going across is maximized it is maximized. So, definitely there are 2 power n algorithm, you try all possible partition and do this, this is one exercise.

(Refer Slide Time: 33:23)

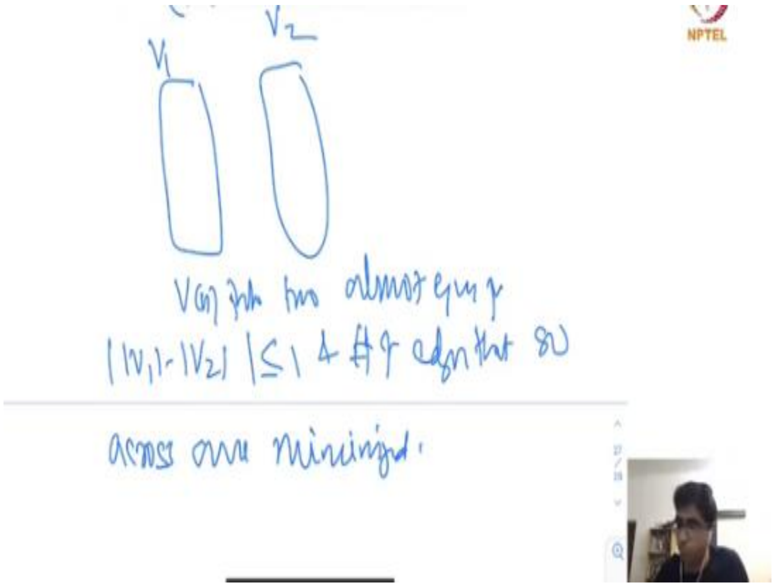

Exercise 2:

Design an algorithm for MIN BISECTION in time $(2-\epsilon)^n$.

V(G) into two almost equal p

Exercise 2, design an algorithm for N IN BISECTION in time, so what is the N IN BISECTION? So, you want to find a partition of vertex set again, VG into 2 almost equal parts.

(Refer Slide Time: 34:00)


V_1 V_2

V(G) into two almost equal p

$||V_1|-|V_2|| \leq 1$ & # of edges that go

across are minimized.

So, they partitions say suppose this is V 1 and V 2. Cardinality of V 1 V 2 should be at most, so they should be differ by at most one. And numbers of edges that go across are minimized. There is several other application of matrix multiplication. But these were the simpler application that I would like wanted to say at with explain to you and so that will be it for matrix multiplication. Thank you.