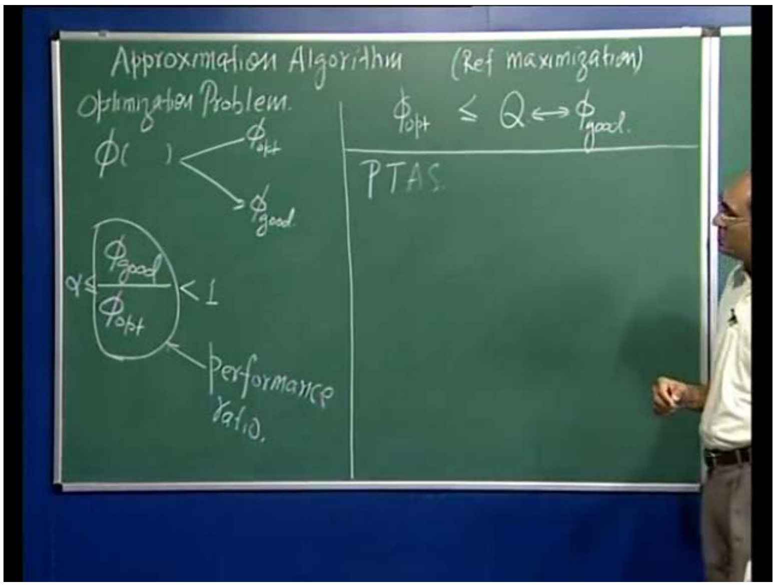
Computer Algorithms - II

Prof. Dr. Shashank K. Mehta
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture -32
Approximation Algo I

Hello, so in the last phase of these lectures we are going to discuss some approximation algorithms, so let me start with some basics. You must be familiar with some problems are known to be n p hard in another set of lectures you may have seen n p completeness and examples of those. So, for we do not know whether the class P and n p are equal, but if assume that they are not equal then we do not hope to be able to solve those problems in a polynomial time. In that case what we would like to do is a try to approximately solve those problems, so first of all let us try to precisely describe what we mean by approximate solutions.

(Refer Slide Time: 01:22)


Approximation Algorithm (Ref maximization)
Optimization Problem.
$\phi(\ )$
ϕ_{opt}
ϕ_{good}
$\alpha \leq \frac{\phi_{good}}{\phi_{opt}} < 1$
Performance ratio.
$\phi_{opt} \leq Q \leftrightarrow \phi_{good}$
PTAS

So, in this context the approximation algorithms refer to a class of problems namely optimization problems note all problems are in this category. But, often what we tend to do is we transform any problem into an optimization problem and then try to solve those approximately. So, let us suppose we have an optimization problem in which we are given some background and we are given some optimization function, some function of few parameters and our objective is to minimize or maximize this.

To find an assignment to various parameters which optimizes this function, we know that to compute the exact the most optimum value of that function is hard. In that case what we would like to do is if we cannot determine the optimum value, let us find some good value. We say how good is it for that we can actually take the ratio of that rule and say suppose this was a maximization problem we do not reach as for as this. So, this is actually less than 1, we know that it is less than 1, but can we say that it is definitely not smaller than, so value alpha.

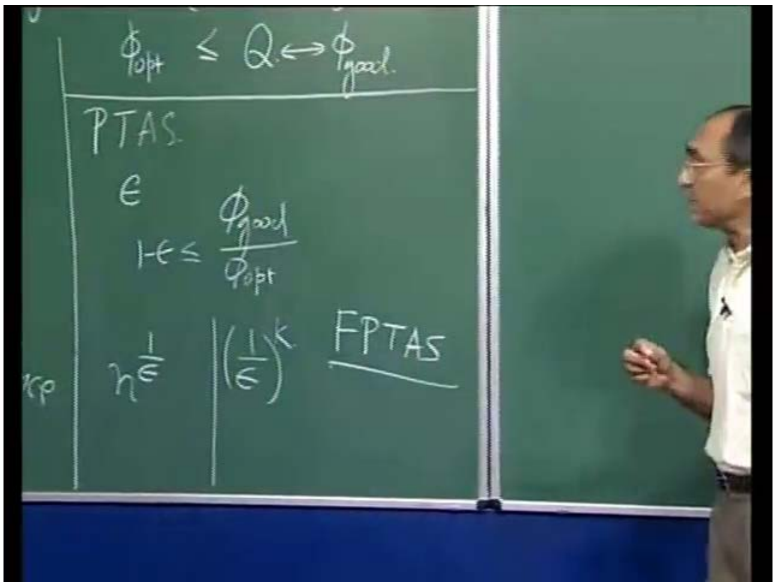
If we can bound this ratio we guaranty that no matter which instance of the problem you give me I will give you a solution. Where the object function will not be too far from the optimal value, that is to say the ratio of the 2 will be bounded by alpha in case of minimization. Similarly, we will have a bound on the other side, now this alpha could be a constant or even a function of the parameter, some cases we cannot due to while then we have some function of parameters. For example, if we can give a 7 half three-fourth, one-third that kind of bound that is found, but if we turn there may be a, we can say 1 over log n or n to the power minus 2, minus 0.2.

That kind of bound would also define, depends how we will can approximate that problem because sometimes even approximating in a sense. In this sense with binding the ratio by good number is also very hard in many problems, so this parameter this parameter is what we are after and this is known as performance ratio. In our case, we will keep this sometimes people normalize this for maximum maximization and minimization, but in our case we will just stay with this interpretation of performance ratio.

So, if this is the goal to push this to one of the key element in the designs of the algorithms is to be able to analysis it, although we do not know this value. We need to show that this is very close to this without actually having this with us, so one of the ways to do that is to find some quantity which is not necessarily quiet relevant to this problem. To determine some value, for which we can say let say that value Q and we can say that value is greater than equal to Q opt, so I am, I am talking about my reference is maximization. So, obviously this is not the value of phi, for any instance of the problem that is not possible, so it is probably some other quantity.

Then to relate this Q with Q phi, the value that we compute through the approximation algorithm and that way we can actually find a bound. Today, I am going discuss 2 examples of two very simple examples that will give you some flavor of the wave and analyzers. Usually, these algorithms are not that difficult while at least these elementary cases that we will discuss the algorithms will be simple, it will be the analysis which will be somewhat less simple. Now, let me, now describe something called P T A S, unlike optimization algorithm this is known as a polynomial time approximation scheme.

(Refer Slide Time: 08:39)

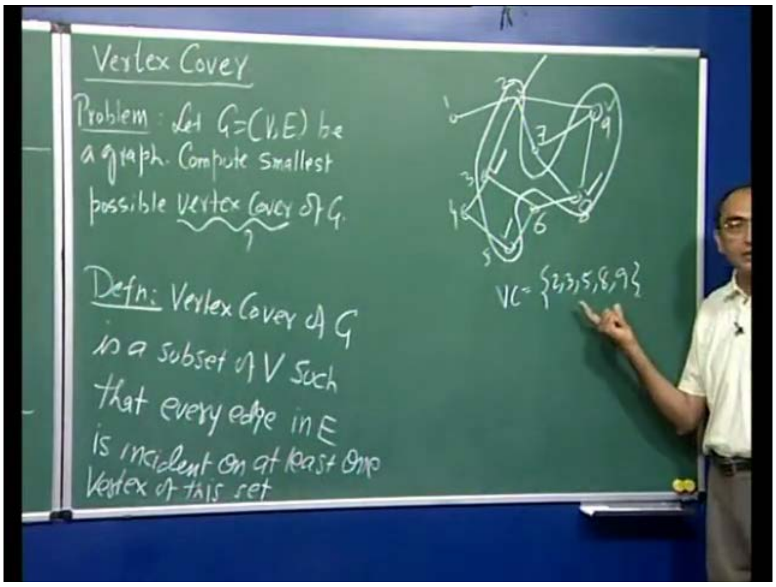

$\phi_{opt} \leq Q \leftrightarrow \phi_{good}$
PTAS
ϵ
$1-\epsilon \leq \frac{\phi_{good}}{\phi_{opt}}$
$n^{\frac{1}{\epsilon}}$
$\left(\frac{1}{\epsilon}\right)^k$
FPTAS

Here, what we are trying to do is given an epsilon once again let us assume we are talking about maximization given an epsilon. If we can provide within this scheme a method such that my phi over phi optimal is greater than equal to 1 minus epsilon, in other words I can bring phi good arbitrary close to phi optimum. But, then you might wonder well how can that be, we have to always provided a polynomial times solution always. There is no discussion of none because there is no point in approximating as well as spending time huge amount of time in solving it.

So, there is a polynomial time algorithm and we are bringing it arbitrary close to the optimal, but there must be some catch, so the answer that is we bring epsilon close to 0 the cost grows. So, it may happen that the time complexity might be, so function of 1 over epsilon, so the time complexity of this may be quite reasonable when epsilon is large. But, as we bring it closer to 0 this grows up, so this can be very bad, now in general it could even be like say an exponent on some parameter that may well happen that the complexity is a function of this form.

In case, in case the time complexity is not of this form although it is a function of one over epsilon, but may be some polynomial of this may be something power K occurs in the time complexity which is not so bad as this one. If we can currently say that my time complexity for this is a partly, a polynomial of this then we call it a fully P T A S, fully polynomial time approximations came or F P T A S. We will have one example of this as well as in the following lectures, so let us start with a first example.

(Refer Slide Time: 12:01)



Vertex Cover

Problem: Let G=(V,E) be a graph. Compute smallest possible vertex cover of G.

Defn: Vertex Cover of G is a subset of V such that every edge in E is incident on at least one vertex of this set

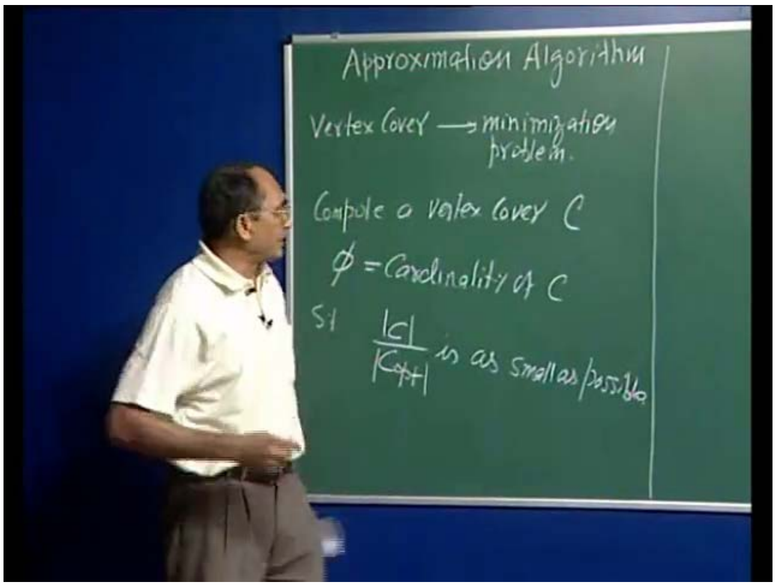
VC = {2,3,5,8,9}

That is something called vertex cover lets describe the problem first, so problem is that let G be a graph and we want to compute smallest possible vertex cover of G. So, what is vertex cover, so vertex cover, vertex cover of G is a subset of V, it is a subset of vertices such that every edge in E is incident on at least one vertex of this set. So, let suppose we have a graph, then perhaps if I try to cover these three that will take care of it does not matter not terribly I can avoid this. So, I am taking these vertices in the set, now what we note is that every edge there are 2 vertices incident on an edge at least one of those two vertices of every edge in this set.

So, this edge has this vertex in the set similarly these 2 edges also have some vertex in this and soon these are obviously both the end vertices of this edges are inside the set and so on. So, if I label them 1, 2, 3, 4, 5, 6, 7, 8 and 9 then one possible vertex cover is 2, 3 a vertex cover is 2, 3, 5, 8 and 9. So, this is a possible vertex cover, the size of this vertex cover is 5, the 5 vertex is in A, so the challenges to compute this smallest possible vertex cover that is say vertex cover with the minimum cardinality.

That is what we would like to compute this is known to be an n p hard problem, so the optimum vertex cover this is naturally A, an optimization problem there are problems which are not naturally optimization problems. One sometimes artificially converts them into an optimization problem, but this one is A, an optimization problem, so the smallest vertex cover is not always possible to compute in polynomial time.

(Refer Slide Time: 17:00)



Approximation Algorithm

Vertex Cover $\rightarrow$ minimization problem.

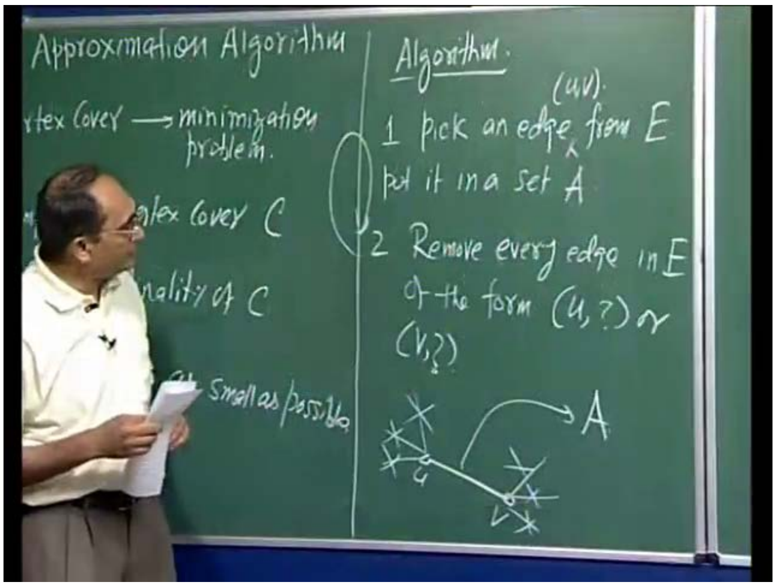
Compute a Vertex Cover C

ϕ = Cardinality of C

S.t $\frac{|C|}{|C_{opt}|}$ is as small as possible

So, our problem is actually a minimization problem our goal is to compute a vertex cover our object function is the cardinality vertex cover C cardinality of C, such that C divided by C opt is as small as possible, we would like to bound this value above. So, usually the, sorry I would say the cardinality of this, so we would like usually this will be bigger than this we would like to bounded by upwards, so here is a very simple algorithm.

(Refer Slide Time: 18:22)


Approximation Algorithm

rtex Cover → minimization problem.

lex Cover C

nality of C

small as possible

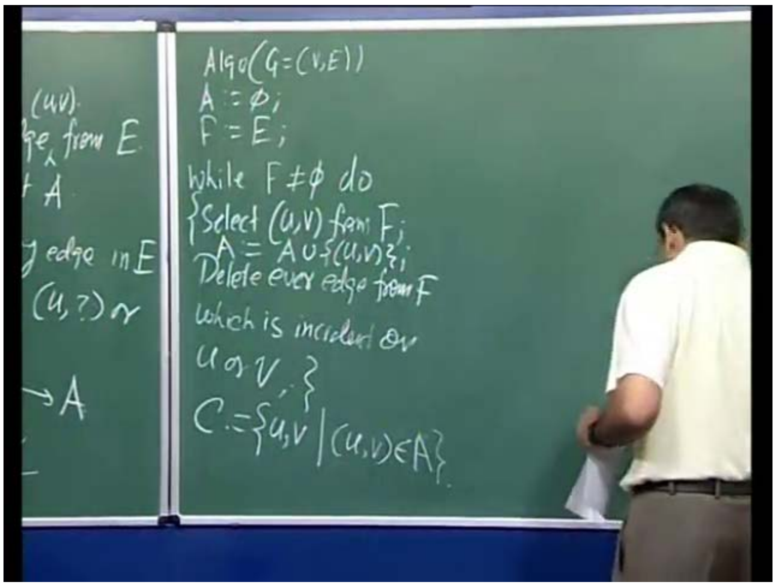
Algorithm.

1 pick an edge (u,v) from E put it in a set A

2 Remove every edge in E of the form $(u,?)$ or $(v,?)$

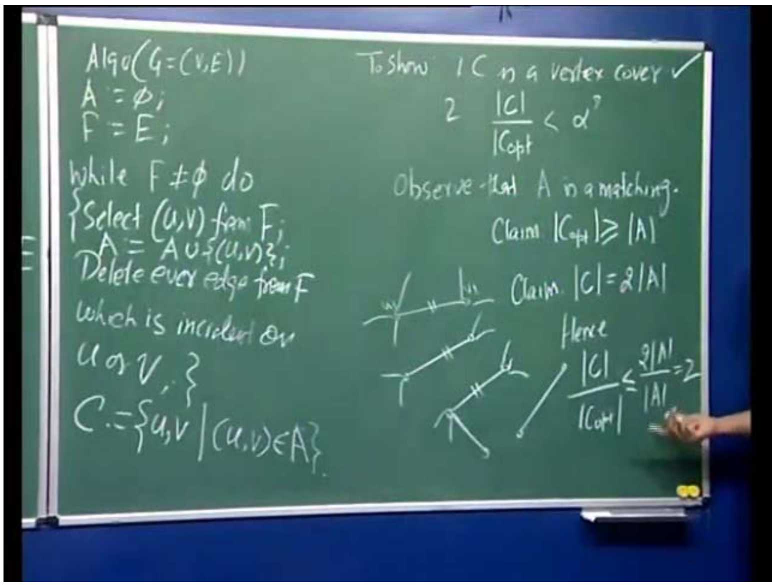
First pick an edge from E, say edges say u, v put it in a set A, so I am just picking orbiter edge and putting it into a set it, in step 2 remove every edge in E of the form u something or v something. So, I take this edge put in a set an every edge, so here is the picture I had pick W V and if there are edges here. So, I am going to remove them put this into set A and then iterate, and then keep do this until edge set is empty, so to be more accurate let us just write down.

(Refer Slide Time: 20:14)


```
Algo(G = (V,E))
A := $\phi$;
F = E;
while $F \neq \phi$ do
{Select (u,v) from F;
 A = A $\cup$ {(u,v)};
 Delete every edge from F
 which is incident on
 u or v. }
C := {u,v | (u,v) $\in$ A}.
```

We have let say F E, we initialize a set A to be empty and then while F is not empty repeat this steps select u, v from F, F is initially the whole of E we select this, delete every edge from F which is incident on u or v. So, this is where the while loop ends, now I have to produce a vertex cover, so my covers C would be all the vertices such that this edge is in F in A. So, you just pick all the end vertices of edges in, yes I have select A and input in A u, v sorry, so I am going to take this edge put in A. Remove everything else of this form, these edges after your through with this process you pick up the end vertices of the edges stored in A, so there are 2 things I have to show.

(Refer Slide Time: 22:36)



```
Algo(G = (V, E))
A = $\phi$;
F = E;
While F $\neq \phi$ do
{Select (u,v) from F;
  A = A $\cup$ {(u,v)};
  Delete every edge from F
  which is incident on
  u or v. }
C := {u,v | (u,v) $\in$ A}
```

To Show 1 C is a vertex cover ✓

2 $\frac{|C|}{|C_{opt}|} < \alpha$?

Observe that A is a matching.

Claim $|C_{opt}| \geq |A|$

Claim $|C| = 2|A|$

Hence $\frac{|C|}{|C_{opt}|} \leq \frac{2|A|}{|A|} = 2$

First one that C is a vertex cover and 2 that is bounded by some alpha which we have to decide whatever this alpha edge, so let us look at the first objective what we have computed. So, observe that a is a matching, now just to remind you matching is a subset of edges, remember A is just collection of edges here, so matching is a collection of edges such that no two edges share any vertex. What we are doing is we pick this u 1, v 1 and we removed all the other edges, after that in the second round when I am going to pick an edge I am going to pick an edge between completely different pair of vertices and so on.

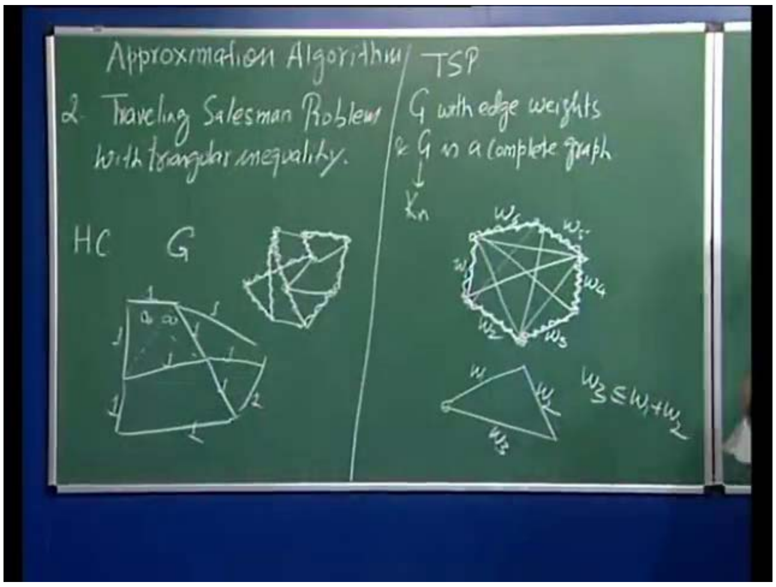
So, every edge that I pick, so there are other edges and all that everything that I pick will be between completely fresh pair of vertices such a thing is a matching. Well, in that case my claim is that C opt cardinality cannot be less than cardinality of A, we cannot have any vertex cover with cardinality less than the cardinality of A the region is simple. Any vertex cover must pick at least one of these two vertices, it cannot avoid both otherwise this edge never be covered similar it must pick one of these two and one of these 2.

So, from every one of these matching edge one of the two end vertices will be put in any vertex cover in particular C opt, hence the size of A is a lower bound for the optimum vertex cover. Now, why does this pair what we have put in C the cover that we have computed contains both of this vertices what does, it form a vertex cover and the answer is simple. Suppose there is an edge which is not a incident upon any of these vertices such an edge will remain in F, it will not be deleted because all you delete is only the edges incident upon the end vertices of picked edge F will not go empty.

So, if this was eliminated if this was not covered, here either it this is covered as a member of A, no problem then else this fellow must have been eliminated. If this was eliminated then certainly this might be an edge of this kind somewhere, hence we know that this vertex is in C. So, that takes care of the fact that C is indeed vertex cover and, now what about putting a bound on the ratio is computing the performance of this approximation. The answer is an obviously claim is that the size of C is equal to 2 times the size of A because all we do for every edge in A, we put this two vertices is in C.

So, we have this, hence C over C opt the cardinality of this 2, so this is 2 A and this is greater than A, so this is greater than equal to 2 mode of A divide, this is 2. Since, this was actually smaller, so here it is greater, so we have an upper bound for this ratio as 2, so we can never do what is, then twice it does not matter how we select the edges from F. So, this is a very simple example how easily I can perform this task of computing a cover which is no more than twice the optimum, while optimum takes enormous amount of time this is a very simple polynomial time algorithm.

(Refer Slide Time: 28:31)



So, let us look at second example the second one is about traveling means salesman problem with, so let us first of all describe the problem you must be familiar with A, well know in pick hard problem known as Hamilton’s cycle problem. So, in Hamilton’s cycle problem you have given a graph G and you would like to know if there exists a cycle there are certain edges we would like to know if there exists a cycle, a cycle is closed walk in which we do not repeat any vertex. So, probably, now write, so may be very hard to see, but the objective is to find a cycle which passes through all the vertices.

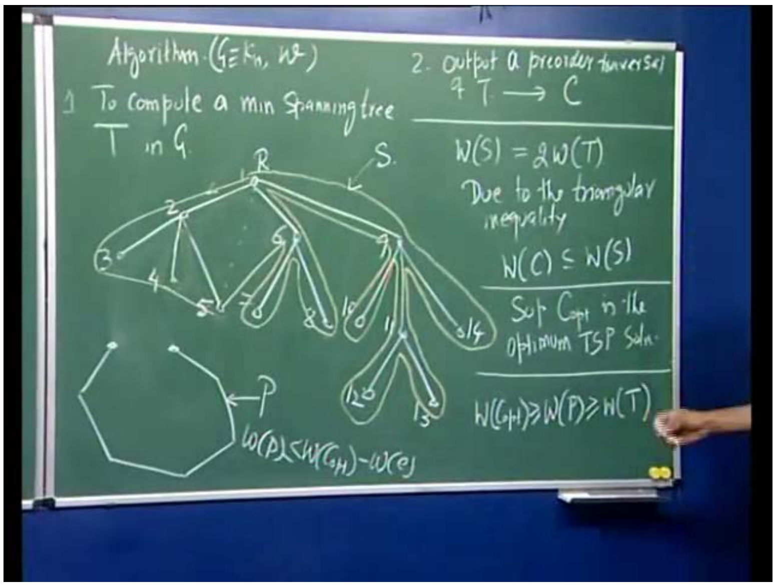
This is also well known hard problem no polynomial time algorithm is known a variant of this problem which is an optimization this is not an optimization problem. Either you have a cycle which passes through all vertices or you do not, but traveling salesman problem is a variant of this problem, which is of the optimization flavor and in T S P which stands for traveling salesman problem. We have the following statement of the problem, so we are given G with edge weights and G is a complete graph. A complete graph is one in which there is an edge between every pair of vertices it is also known as K n, G is K n complete graph on n vertices edges have weight.

Now, in a complete graph to find a cycle passing through every vertex is a trivial task because every permutation of vertices is a cycle in a complete graph because all pairs are connected. So, the hard part is not to find a cycle, but to find a cycle whose weight is minimum, so the cycle weight in this case is the rates of the edges of that cycle. If I have this cycle let said this are the only vertices of the graph, these there are another edges of course we are not concerned all there are edges between every pair of course. But, the weights that we are interested in, so this is the cycle W 1, W 2, W 3, we would like to minimize the sum of these weights, these are the weights of the edges on the cycle.

Now, one can show that this has to be n P r the region is we can solve the Hamilton cycle problem, if we can optimize solve the traveling salesman problem. The region being you give me arbitrary graph G, so I have this some graph which is not complete of course I can put weight of 1 unit on this and then I add edges which are not present, I put all the edges and put infinite weight on those infinite weight. Now, this is an instance of T S P, find the optimum solution of this T S P if there is Hamilton cycle, we know that their exist a cycle with weight n these are n vertices on it and edges on cycle each as unit weight.

So, there will be a solution with weight n, otherwise it will be infinite, so immediately it can tell whether this is whether this has got Hamilton cycle or not. So, indeed exact solution is hard, but we are going to make one more assumption about this, what we assume is that the weights are such that they satisfy a triangular inequality. What it says is that suppose you have in this graph G which is K n whenever you have a triangle, if the weight of these edges W 1, this is W 2 and this is W 3 in that cases W 3 is always less than equal to W 1 plus W 2. Well, by the same token W 1 less than equal to W 2 plus W 3 or W 2 is less than equal to W 1 plus W 3, when this condition is satisfies we can give a decent approximation algorithm, so let us now consider the following algorithm.

(Refer Slide Time: 36:00)


Algorithm (G ≡ K_n, W)
1 To compute a min spanning tree T in G
2. output a preorder traversal of T → C
R
S.
1
2
3
4
5
6
7
8
9
10
11
12
13
14
P
W(P) < W(C_H) − W(e)
W(S) = 2W(T)
Due to the triangular inequality
W(C) ≤ W(S)
Suppose C_opt is the optimum TSP soln.
W(C_H) ≥ W(P) ≥ W(T)

Notice that this is again a problem of minimization you want to compute a cycle Hamilton cycle in this graph with minimum weight. So, the step 1 instead of formally writing it I will actually describe by diagram, first one is to compute a minimum spanning tree T. So, here we have graph G which is of course K n and a weight function W in G, so we being with a spanning tree the computation of the minimum weight. Spanning tree G notes that there exists an algorithm to compute minimum spanning tree in polynomial time, so let us suppose we have some tree, but I am drawing this tree with a particular objective and picking any one vertex and calling that is a root.

So, we actually draw the tree in that fashion where the root comes in then it just falls down, now we are going to compute a preorder traversal of this. So, preorder traversal is in the following passion we start with the root then we do the preorder traversal this sub tree then preorder traversal of this tree, sub tree and then of this. So, preorder traversal is defined recursively lets actually go through that we come, here then we come here then we go to this type.

Then this, then this because these are 3 sub trees then we come to this right to this and to this then to this and this, so let us, now label them the order in which we visited first. Then second, third, fourth, fifth, sixth, maybe I will just for the sake of an example seventh, eighth, ninth, tenth, eleventh, twelve, thirteen and fourteen, this is the order in which we can will traversal this particular order is my output so output. So, step 2 is output a preorder traversal of T we know that any permutation of the vertices forms a Hamilton cycle in a complete graph K n.

So, this indeed is a valid because there are edges between every pair, so how good are we doing in terms of the performance compare to the optimum. So, let us do the following first thing we are going to do is generate this traversal what we are doing is we are writing a closed walk 1 to 2 to 3 back to 2 to 4 back 2 to 5 back to 2 back to 1 then 6, 7, 6, 8, 6, 1 and so on. So, you visualize this picture, let us call this closed walk I am going to call this close walk or label it by S and I am trying to determine the determine how good is this output.

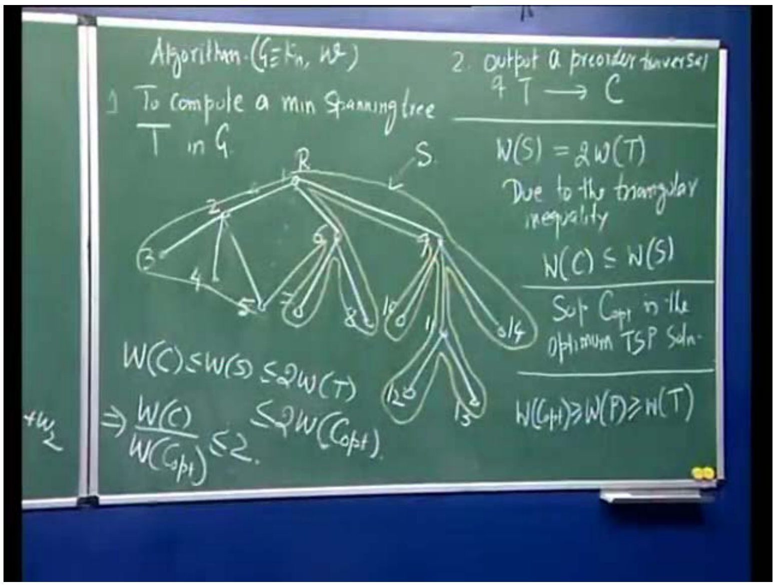
So, what we have done is we have basically covered this tree by a closed walk and we calling it S, then what is the weight of S. Note that weight of S is the sum total of the weights of its edges and what we are doing here is we are you look at any edge we are once going in one direction. Second time we are going in the other direction in each every single one of this, so the weight of this is precisely two times the weight of the tree which is trivial. Now, let us neither of this 2 or any interest to us this is not output and this is not even a cycle not a cycle because we are revisiting certain vertices, now we will do is the following lets looking at this 3, 2 and back to 4 what we will say.

Let us cut this half and connect this directly, so we have take this of 3, 2 to 4 and replace that by 3, 4 triangular in equality says that the weight cannot increase, once we do this the weight cannot increase. But, same token we will directly go from 4 to 5, we will directly go from 5 to 6, now in this case let say C carefully, actually I have replaced 5 to 2, 1 by phi 1 first. Then phi 1, 1 6 by directly 5, 6 in other word in every step, I have use the triangular inequality to argue that the resulting cycle as we go on cannot way more than the weight of S. At the end of the day, the cycle that we will be building is nothing but the cycle we have, so let us call this cycle C, so the triangle or any quality ensures that due to the triangular in equality the weight of c is less than equal to the weight of S.

So, we now have at least an upper bound for the weight of C, but still we have to somehow bind this above by the weight of the optimum traveling salesman problem solution. But, that is also very easy let suppose C opt is the optimum cycle optimum T S P solution, now right take this. So, this is some cycle which I cannot draw this, I am just for the sake of visualizing I am just thinking, let say this is that C opt passing through very vertex pick edge and just delete it. So, what you get is that the weight of this is A, this path P, so the weight of this P is actually equal to the weight of C opt minus weight of E, now temporarily let us assume weights of all the edges are non negative.

We will take care of that general case in a minute assume this in that this is actually a tree path is a tree. So, we know that the weight of P has to be greater than equal to weight of T because this is the, this is the spanning tree because it passes through every vertex and T is one of the optimum spanning trees. So, this argument simply shows that the weight of C opt is greater than equal to weight of P, which is greater than equal to weight of T. The optimum spanning tree we are done, all we have to do is put the whole thing together, so let us, let us do that.

(Refer Slide Time: 47:54)


Algorithm $(G \equiv K_n, w)$

1 To compute a min Spanning tree T in G

2. output a preorder traversal of $T \longrightarrow C$

$W(S) = 2W(T)$

Due to the triangular inequality

$W(C) \leq W(S)$

Set C_{opt} is the optimum TSP Soln.

$W(C_{opt}) \geq W(P) \geq W(T)$

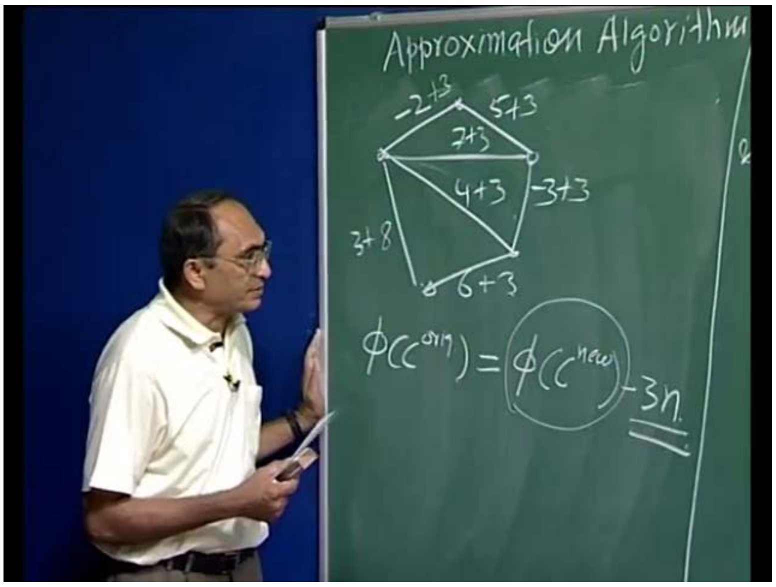
$W(C) \leq W(S) \leq 2W(T)$

$\leq 2W(C_{opt})$

$\Rightarrow \frac{W(C)}{W(C_{opt})} \leq 2.$

The whole thing, now comes out to be W C the weight of the cycle is computed less than equal to W S which is less than equal to 2 of W T which is less than equal to W of, so which is 2 W C opt. Hence, W C divided by W C opt is less than equal to 2, once again we have a performance ratio of 2, now to wrap this finally up one more question, I have to solve what happens, what happens when there are negative weights some edges are negative weights.

(Refer Slide Time: 49:02)



Well, so suppose I have my graph let say the weight is, so this is minus 2 5, 7 minus 3, 4, 8, 6 well there is smallest number is minus 3, we will add 3 to everybody. Let us put plus 3, well indeed this is a new set of weight and there are no negative weights in this, but how is it related to the old problem. So, the answer is very straight forward everybody has added 3 weights and in any Hamilton's cycle, any cycle passing through all the vertices is the exactly n, n being the number of vertices there are energies. So, if the weight of a original cycle, let us say C original is always C new minus 3 n the 3 new in the new we are added units per edge uniformly.

So, this is a constant, so whatever be this value n is a constant this is the weight of the smallest cycle the smallest weight edge, hence to solve the problem on the original graph or on this graph it is same it is just in A. You have to minimize this, are you minimize this are you just minimize this is the same thing because this is just a constant. Hence, it is sufficient to assume the all the weights are non negative because the moment you have negative weights simply upgrade the weights by a constant so that there is no negative weight. So, here is we have we have given to two examples of approximation, in incidentally both of them had constant performance ratio will see some other example in the following 2 lectures.