VLSI Physical Design

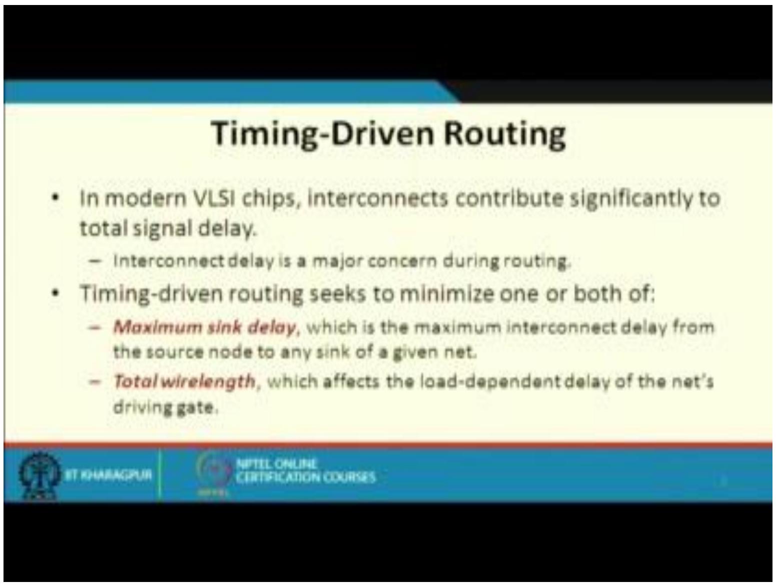
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 38
Timing-Driven Routing

So, if you recall in our last lecture we were talking about the timing driven placement issues some algorithms and some techniques that are used there. So we continue with our discussion today. So today we start with some discussion on timing driven routing.

So, the basic idea is as follows, see means when we talked about timing driven placement our objective was how to place the blocks such that some timing constraints were made with respect to criticality slacks and so on. And now we are talking about routing. We are talking about the individual nets typically there will be multi pin nets. Net will be connecting more than 2 points in general. So, how to layout those nets on the metal wires so that some delay and some other requirements or constraints are met or satisfied. So let us look into this.

(Refer Slide Time: 01:27)



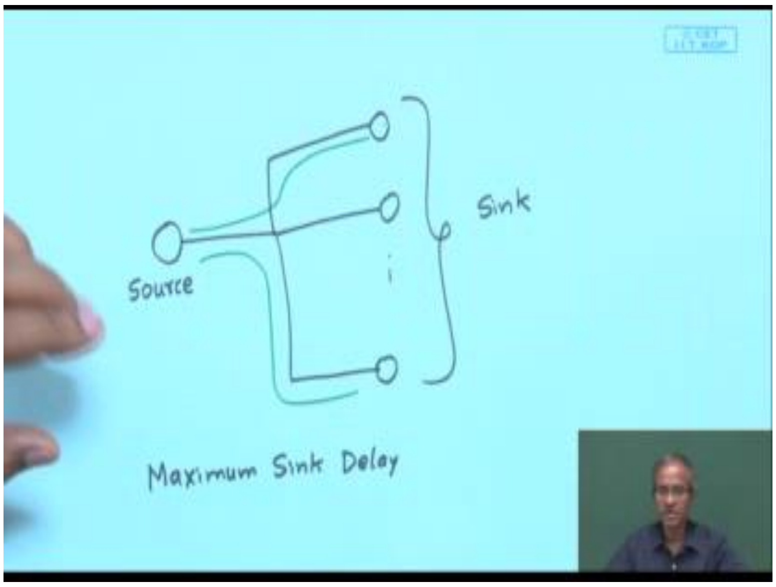
Timing-Driven Routing

- In modern VLSI chips, interconnects contribute significantly to total signal delay.
  - Interconnect delay is a major concern during routing.
- Timing-driven routing seeks to minimize one or both of:
  - *Maximum sink delay*, which is the maximum interconnect delay from the source node to any sink of a given net.
  - *Total wirelength*, which affects the load-dependent delay of the net's driving gate.

So, here the motivation I have already talked about earlier also, so we mention that in today's VLSI chips interconnects contribute significantly to the total signal delay, because of 2 things you see, because of the scaling down of feature sizes the transistors are becoming smaller gates are also becoming smaller there are switching speeds are increasing, so gates are becoming faster, but if you think of the interconnecting wires they are biking they are becoming thinner. Thinner means their resistance per unit length is increasing and also pairs of wires which are running in parallel, they are now running much closer together. So there are more capacitive effects between them. There are some cross talk and other issues. So these issues are coming into the picture because of which interconnection today is playing a more significant role means for deciding the overall delay as compared to the delay of the individual gates, fine.

So, here when you talk about timing driven routing, so we are talking about possibly 2 objective functions or means objective criteria. So we may seek to minimize either one of them or both. Maximum sink delay, well sink what is a sink? Sink is something like this. Let us say I have a net.

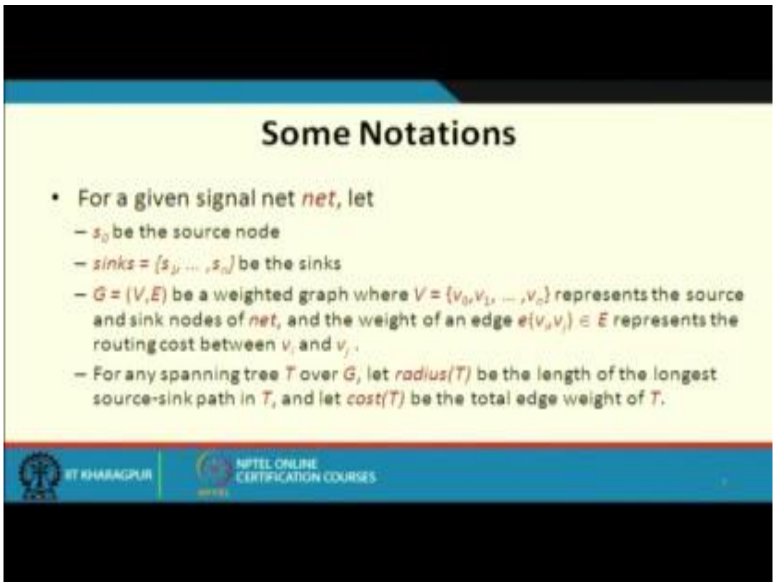
(Refer Slide Time: 03:10)



Let us consider a net like this. There is a point which needs to be connected to several other points. So there are so many ways to connect them. I am showing one possible connections. So a one of these points we refer to as the source and the other points to which the source needs to be connected is called the sink. Now when we talk about the maximum sink delay, so what we are talking about is this. From this source down to each of the sinks what are the delays. So the delays in general be different because the paths are different they will be varying in length and other parasitic effects will also be different. So we want to minimize the maximum sink delay; that means, this internetwork tree what we are drawing this has to be balanced in some way. So that the distance or the total delay from the source to the sinks are not that much different they should be balanced and the maximum sink delay should be minimized as far as possible.

This is of course; one of the criteria and the second criteria as you can see is the total wire length which means my interconnection should be such that the total length of the wire should be minimum. Now you see earlier when we talked about the traditional routing there our main criteria were to minimize the wire length. So whatever we are talking about there we were mainly talking about how much was the total Manhattan distance between the 2 points Steiner tree cost and so on and so forth. But here in addition to the total wire length we are also talking about the worst case delay from a source down to some of the sinks what is the maximum delay of this paths fine.

(Refer Slide Time: 05:29)

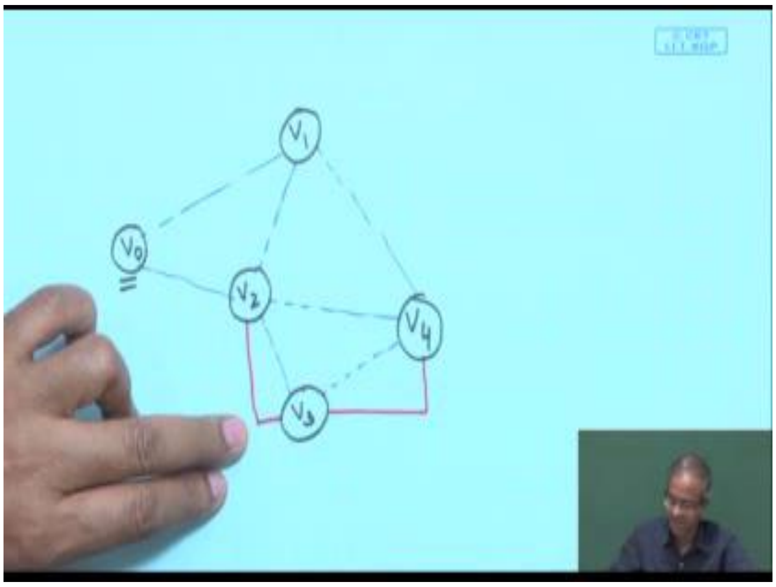


Some Notations

- For a given signal net *net*, let
  - s_0 be the source node
  - $sinks = \{s_1, \dots, s_n\}$ be the sinks
  - $G = (V,E)$ be a weighted graph where $V = \{v_0, v_1, \dots, v_n\}$ represents the source and sink nodes of *net*, and the weight of an edge $e(v_i, v_j) \in E$ represents the routing cost between v_i and v_j.
  - For any spanning tree T over G, let $radius(T)$ be the length of the longest source-sink path in T, and let $cost(T)$ be the total edge weight of T.

So, let us introduce some notations. So we consider a particular net we call it as net. The source node we designate it as s 0. Let us assume that there are n number of sink nodes s 1 to s n. So we define a graph, where the vertices of a graph of this graph are the points that need to be connected. So one is this source and other is the sink. So let us say our graph may look like this.

(Refer Slide Time: 06:08)

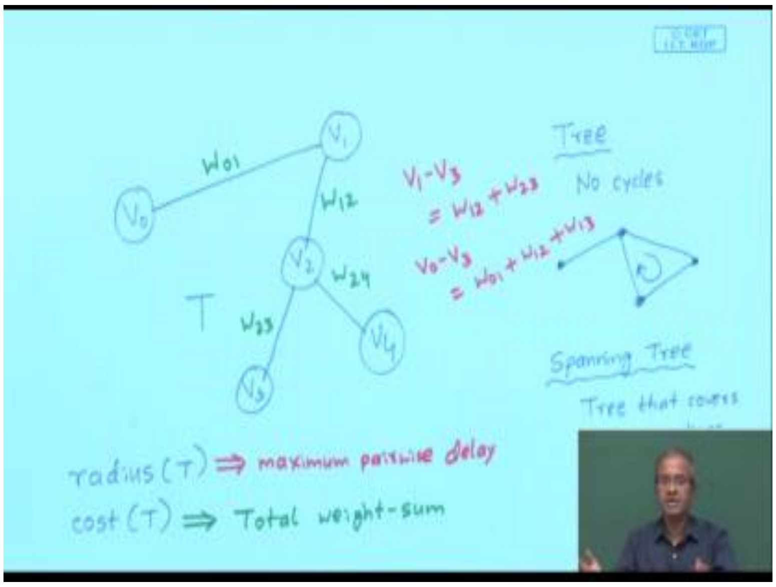


This is v 0 this is say v 1 this is say v 2. Let us say here is v 3. Let us say here is v 4. So this is the source and these are the 4 sinks. Now we have to connect then in some way this is our requirement. So addition we are also defining some weights of an edge between a pair of vertices v i and v j, there is an edge so this e v i v j denotes the edge, between v i and v j. And we can define a weight of the edge the weight of that edge will indicate the routing cost between v i and v j.

For example, let us say here between v 3 and v 4 this can be one possible connection. So the weight between them will indicate the cost of this. Between v 2 and v 3 the connection may be like this. So when we talk about a graph means we will assume that there will be edge between v 2 and v 3, there is an edge between v 3 and v 4 may be there can be other edges. Also so potentially there can be edge between any pair of vertices. So I am showing some of the edges so the weight of this each of this edges will indicate that what will be the cost or the delay of the path connecting these 2 pair of vertices. This of course, will determine or be determined by the wire length and also on the resistance and capacitive resistive and capacitive effects that the RC delays in the other parasitic effects, fine.

So, I mean effectively when you talk about connecting a set of points in terms of a graph we would want that what we need is a tree is a spanning tree. Now let us again look take an example to illustrate this.

(Refer Slide Time: 08:24)

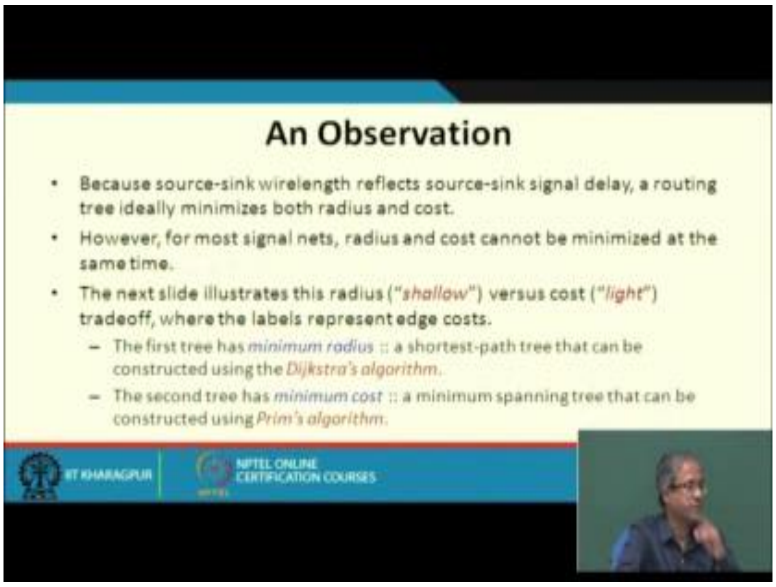


Let us say I have again a set of vertices. It is I am just showing a few. So we define a tree see a tree is what a graph. Graph is what a set of vertices connected by some edges. So a tree is a graph where there are no cycles. Because you see when I am connecting a set of points, there is no point in making a connection like this where already I have connected the 4 points in addition I am making another connection like this. So I have made a cycle here so there is no point in have a cycle in such an interconnection tree because ultimately we are talking about interconnecting these points. So what do we need is a tree and when you talk about a spanning tree. So what is a spanning tree? A spanning tree is a tree that covers all the vertices. So for this graph there can be so many possible spanning trees. One possible spanning tree can be this. Spanning tree is a way of connecting all the vertices such that there is no cycle anywhere because cycles for a interconnection network does not make any sense.

Now, with respect to this spanning tree so in this diagram I am showing a spanning tree with respect to the spanning tree let us call this spanning tree as T. We define 2 terms, one is the radius of the spanning tree and the other is the cost of the spanning tree. Now you see these points will be located on the 2 dimensional grid in some exact co ordinates. So we can also define some weights. Let us say between v 0, v 1 this is a weight W 01, let us say this is a weight W 12, this is a weight W 23 and W 24. So these weights actually indicate that what will be the delay or estimated delay along these interconnecting segments of wires. Now when I talk about cost, cost means the total weight sum you can say, the sum of the weights of this spanning tree. So suppose this is the spanning. This is a spanning tree with respect to this I am trying to connect. So if I take this sum of all the weights that will define my cost. This is one factor cost, and the second one which we call as radius. What is radius is the worst case distance between any pair of vertices. Let us say between v 1 and v 3, between vertices v 1 and v 3, what will be the distance W 12 plus W 23. This will be the delay estimate 1 2 plus W 23.

Say between v 0 and v 3 what will be the delay, 0 1 1 2 2 3. So like this you can estimate the distance or the delay between every pair of vertices. Now the maximum of this that is defined as the radius; maximum you can say pair wise delay. This will define as a radius. You see both of these are important. Cost will tell me that what will be my total connecting wire length; that means, my cost in terms of the interconnection area and this radius tells me that what will be the maximum pair wise delay; that means, the longest path in this spanning tree that will tell that if there is a signal transition at v 0 so how much delay or how much time it will take for that signal transition to reach some other node. So the maximum of that will determine the maximum delay of my net. So, here we are defining radius and cost. Radius is the length of the longest pair wise sourcing path and cost is the total edge weight, some of the edges in this spanning tree.

(Refer Slide Time: 13:39)



An Observation

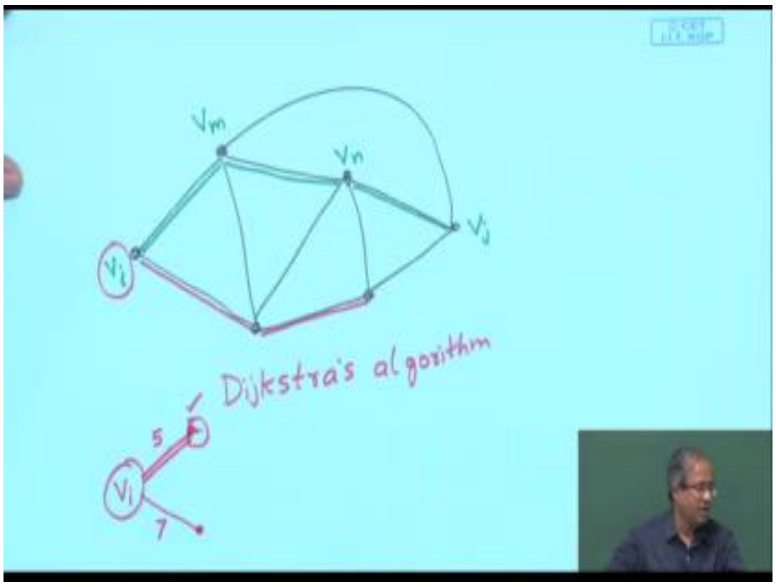
- Because source-sink wirelength reflects source-sink signal delay, a routing tree ideally minimizes both radius and cost.
- However, for most signal nets, radius and cost cannot be minimized at the same time.
- The next slide illustrates this radius (*"shallow"*) versus cost (*"light"*) tradeoff, where the labels represent edge costs.
  - The first tree has *minimum radius* :: a shortest-path tree that can be constructed using the *Dijkstra's algorithm*.
  - The second tree has *minimum cost* :: a minimum spanning tree that can be constructed using *Prim's algorithm*.

So just one observation you can have, that when we talk about the wire length between any pair of vertices between a source and a sink. This wire length is also some kind of a measure of delay because longer the wire typically delay will be longer, but there are other issues as you know that there can be the parasitic resistance capacitances so along the wire because of that not only the length of the wires some other neighboring affects also can come in, but roughly speaking length of the wire is a measure of the delay.

So, because traditional routing algorithm as we have seen earlier, it aims to minimize wire length. So we can say that well in intuitively should minimize both radius and cost, but will see I will take an example, a little later will see that you cannot really optimize radius and cost simultaneously. They are mutually conflicting in some sense. So minimizing radius, we can say it is a shallow kind of a tree; that means, the tree is not spreading out the tree is quite compact the radius is minimum. I mean such versus you can say minimum cost which means I have a tree whose total edge weights are very light. The total cost of the edges is less. So we are defining a tradeoff. Tradeoff means we can have in the extreme case 2 different kinds of trees. So the first kind of tree is one which has minimum radius; that means, it is not spreading out it is short in terms of the span. The radius the distance between any pair of vertices in the tree is shortest.

So, there is a classical algorithm called Dijkstras algorithm. This algorithm can be used to find the minimum radius spanning tree. Because you see what Dijkstras algorithm does given a graph it finds out the minimum distance path between any pair of vertices, like let us say I have a graph like this there are some vertices.

(Refer Slide Time: 16:19)

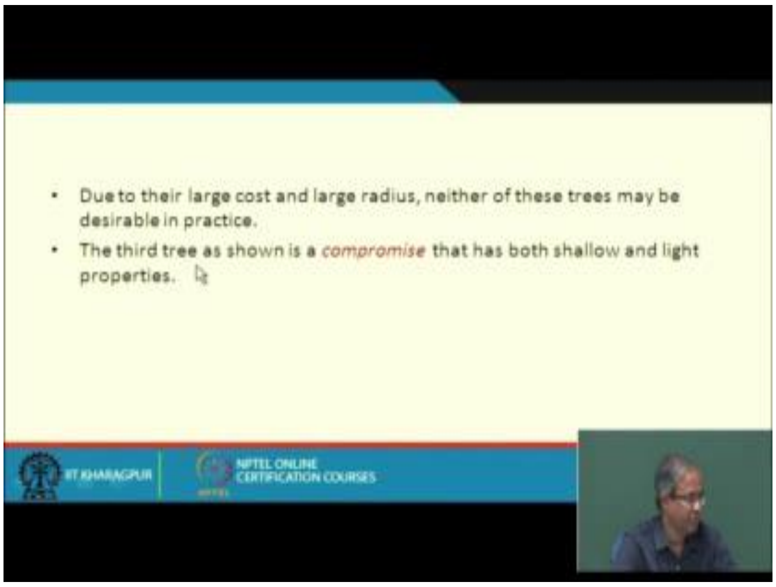


Suppose I have a graph. So each edge will be having some weight. Each edge will be having some weight. It is a measure of the cost. Now what Dijkstras algorithm says that between any pair of vertices let us call it v i let us call this vertices v j. It will find out the shortest path. Let us say the shortest path is this. So if the shortest path is this let us give these to vertices some name also let us call it v m and v n, if the shortest path between v i and v j is this then it can be easily proved through means arguments that the shortest path between v i and v m should be this v i and v n should be this only. It cannot be some other path because if there is a shorter path up to v n. Then you could have reached v j via that path.

So, whenever I have a shortest path all the vertices along the shortest path also will be included in the shortest path delay. Similarly let us say when you consider the shortest path between v i and some other vertices let us say this; you may find that will my shortest path is this. So if you include all the shortest path like this already we have found out a spanning tree. See the green and the red edges the this form a spanning tree so from a particular source, this spanning tree will tell you the shortest distance path will give you the shortest distance path up to each of these vertices. Right and this is what Dijkstas algorithm does. This is a very standard algorithm so it basically starts with one vertex and iteratively adds one of the edges depending on some cost criteria. So it finds out which one is the lowest cost like for example, it is starts with a initial node v i. It checks that there are 2 vertices connected to v i, which one has the shortest weight. Suppose this has 5 this has 7 so it will pick up this. And it will visit this node and declare that this is the shortest path from v i to here, in this way you proceed at every step you add one more vertex to this graph and in the process you construct the tree, fine.

In the other extreme you can have a tree that has a minimum cost, and again there is well known algorithm called Prims algorithm. Now Prim’s algorithm what it does? It actually constructs a minimum spanning tree. Minimum spanning tree is a tree spanning tree which who’s some of the edge weights is minimum. Now this is what we are wanting in the minimum cost tree. So we have 2 alternate ways of constructing the tree Dijkstra’s or Prim’s. One will give you minimum radius other will give you minimum cost.

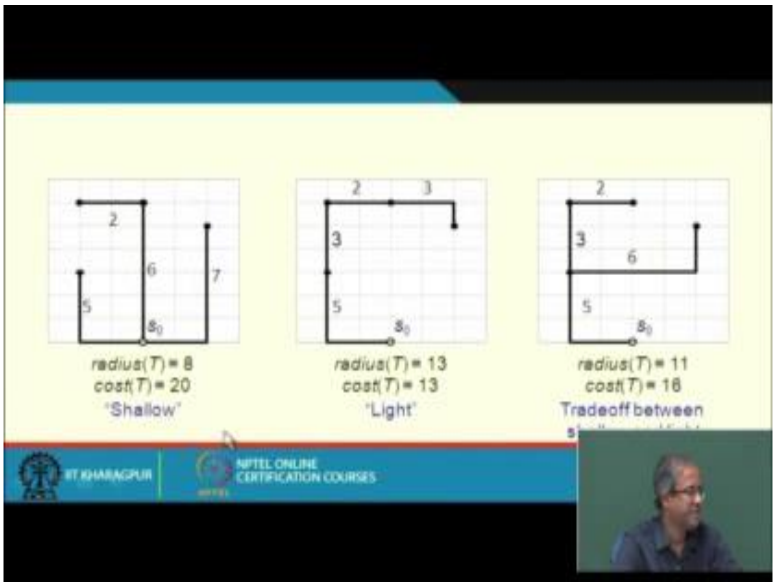
(Refer Slide Time: 19:50)



- Due to their large cost and large radius, neither of these trees may be desirable in practice.
- The third tree as shown is a *compromise* that has both shallow and light properties.

So, but because of either large cost or large radius, as I shall see an example a little later neither of tree in it is purest sense is desirable. So what we need is some kind of a compromise. This is shown in these example as we see now.

(Refer Slide Time: 20:12)



radius(*T*) = 8
cost(*T*) = 20
'Shallow'

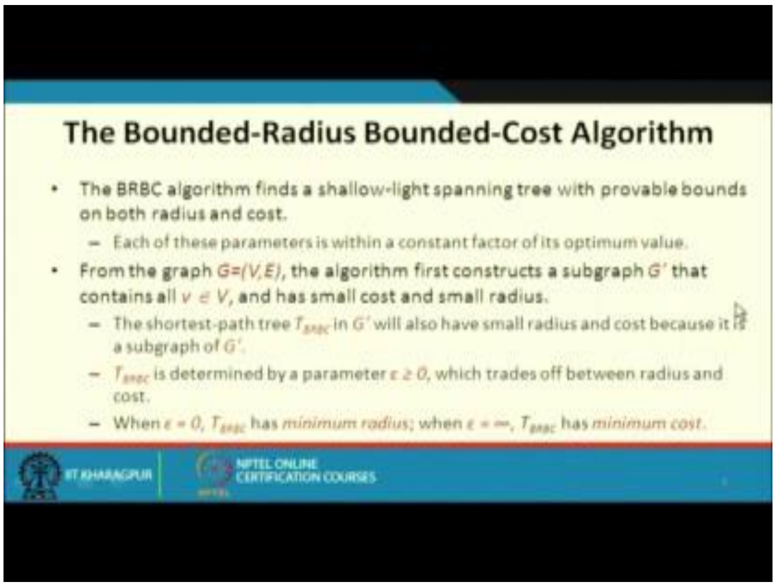
radius(*T*) = 13
cost(*T*) = 13
'Light'

radius(*T*) = 11
cost(*T*) = 16
Tradeoff between

Let us look at this example where there are some points. This S0 is the source and there are 1, 2, 3, 4 other points. These are the sinks one, 2, 3, 4. So the first 2 diagrams they show the shallow tree and the light tree constructed using Dijkstras and Prims algorithm respectively. Now this numbers alongside the edges they give the cost of the respective edges. In this case these are the Manhattan distance as you can see 5 means 1, 2, 3, 4, 5, 7 means 1, 2, 3, 4, 5, 6, 7 in this way you can compute the cost. And this is the tree as you can see if you compute the sum of all the edges 5 plus 2 plus 6 plus 7 this will be more than 3 plus 2 plus 3 plus 5. This is a lower cost the total cost is 13 as compared to 20 here, but the problem here is that the length of the pair wise path the worst case is the longest from S 0 to this. This is 13. This is the radius so as compared to it here from S 0 to this node there is a longest path which is 8 only.

So, I am just showing here another graph which is not minimum radius nor it is minimum cost, but it is an intermediate one. Where you see radius is between 8 and 13 means 11 and cost is between 20 and 13 it is 16. So what we are looking for is a tree which is something like this. This is a tradeoff between the shallow tree and the light tree fine.

(Refer Slide Time: 22:05)



The Bounded-Radius Bounded-Cost Algorithm

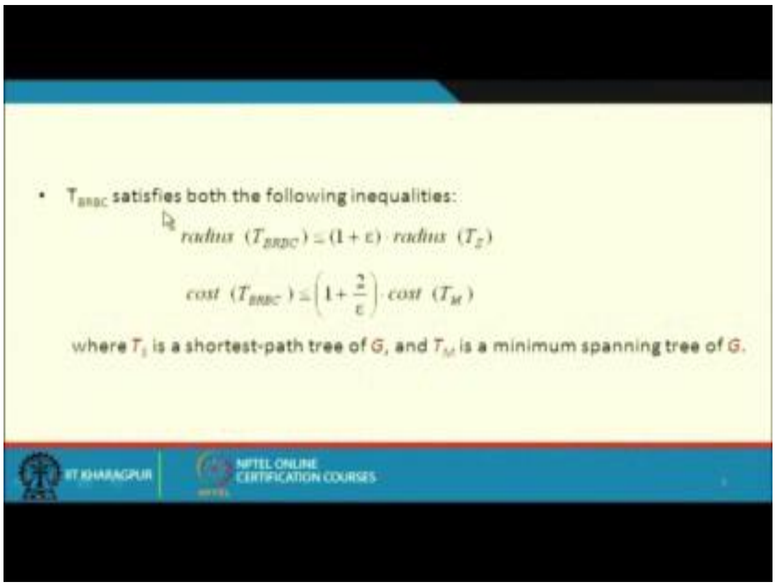
- The BRBC algorithm finds a shallow-light spanning tree with provable bounds on both radius and cost.
  - Each of these parameters is within a constant factor of its optimum value.
- From the graph $G=(V,E)$, the algorithm first constructs a subgraph G' that contains all $v \in V$, and has small cost and small radius.
  - The shortest-path tree T_{BRBC} in G' will also have small radius and cost because it is a subgraph of G'.
  - T_{BRBC} is determined by a parameter $\varepsilon \geq 0$, which trades off between radius and cost.
  - When $\varepsilon = 0$, T_{BRBC} has *minimum radius*; when $\varepsilon = \infty$, T_{BRBC} has *minimum cost*.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, there are a number of algorithms which have been proposed and these are available in the literature. So one is called the bounded radius bounded cost algorithm. So this algorithm actually gives you some bounds on radius and cost and it guarantees a spanning tree which will not cross that bound.

Now the idea is very I mean intuitively simple given a graph of the vertices which we want to connect, you first construct a sub graph a sub set of that graph which contains all the vertices and some of the edges; that means, the tree. And which has small cost and small radius I am using a qualitative term small. So what it says is that so in this G dash need not be a tree G dash can be a general graph. Also, but it has a small cost and small radius it is the sub set of the original graph. So if I construct a tree in this graph G dash because it has a small cost and radius this tree is also expected to have a small radius and a small cost. Now the way this tree is constructed is based on parameter epsilon which is a positive constant which determines some kind of a tradeoff between radius and cost.

(Refer Slide Time: 23:58)



- T_{BRBC} satisfies both the following inequalities:

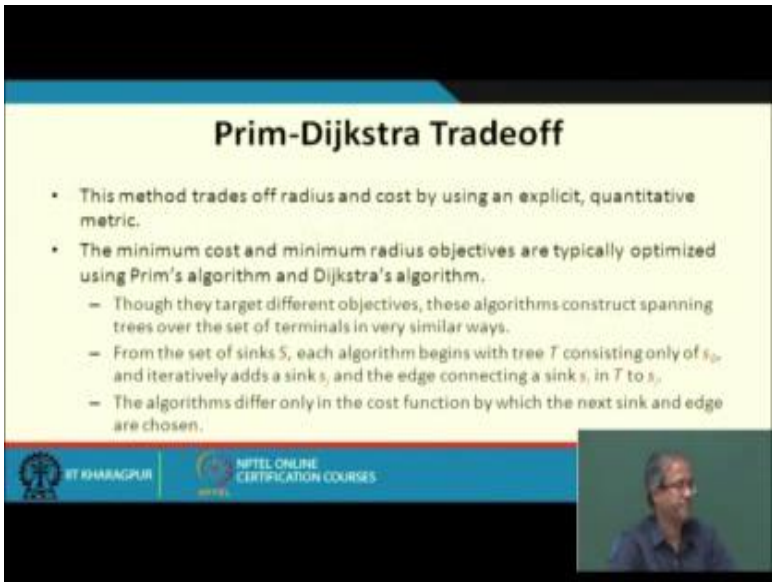
$$radius\ (T_{BRBC}) \le (1+\varepsilon) \cdot radius\ (T_S)$$

$$cost\ (T_{BRBC}) \le \left(1+\frac{2}{\varepsilon}\right) \cdot cost\ (T_M)$$

where T_S is a shortest-path tree of G, and T_M is a minimum spanning tree of G.

So, the idea is that when epsilon is equal to 0, this tree will be the minimum radius tree and when it is very large it will tend to become the minimum cost tree. The way you define the radius and cost is like this. So in the method that I am talking about radius is upper bounded by 1 plus epsilon into the radius of the shortest path tree; that means, whatever is the shortest path tree of the original path the radius of this BRBC tree that you are getting it will be upper bounded by 1 plus epsilon times this. And cost will be upper bounded by a factual 1 plus 2 by epsilon. So when epsilon tends to infinity this term will be 0 it will be just cost and an epsilon equals to 0 this will be only radius this will be only 1 only. This so I am not going with the details of the algorithm, but the idea is like this there is a parameter that creates a tradeoff between these 2.

(Refer Slide Time: 24:47)



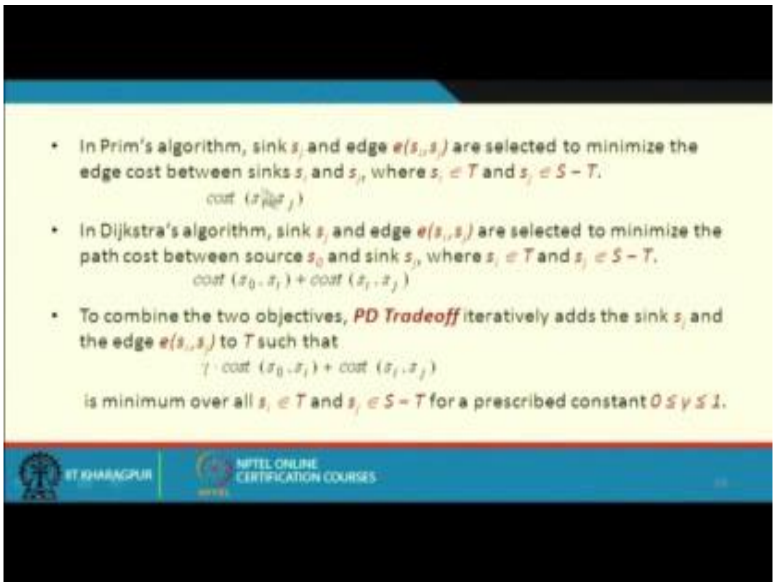
Prim-Dijkstra Tradeoff

- This method trades off radius and cost by using an explicit, quantitative metric.
- The minimum cost and minimum radius objectives are typically optimized using Prim's algorithm and Dijkstra's algorithm.
  - Though they target different objectives, these algorithms construct spanning trees over the set of terminals in very similar ways.
  - From the set of sinks S, each algorithm begins with tree T consisting only of s_0, and iteratively adds a sink s_j and the edge connecting a sink s_i in T to s_j.
  - The algorithms differ only in the cost function by which the next sink and edge are chosen.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, there is another interesting kind of a tradeoff which you can also think off. See this Prims algorithm and the Dijkstras algorithms from the 2 extremes. One gives us minimum cost and the other gives us minimum radius. So why cannot we have a direct compromise between or a tradeoff between Prims and Dijkstra. So this method talks about that. You see the observation that is made here is that this Prims and Dijkstra although they are different algorithms the way they work is quite similar. How? Both the algorithms progressively try to construct the tree from the set of sinks, it starts with a tree consisting only of the source node at the beginning and at every step it adds one node from S and the tree is progressively constructed. The only difference is that the cost which node to add that is based on some cost function and this cost value will tell you that which is the next node which is that candidate for we included in the spanning tree as you are adding the nodes one by one.

(Refer Slide Time: 26:15)



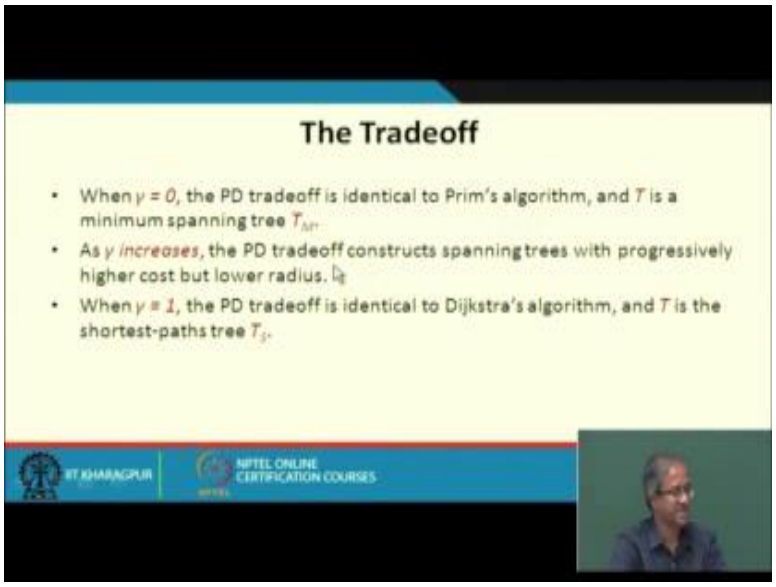
- In Prim's algorithm, sink s_j and edge $e(s_i, s_j)$ are selected to minimize the edge cost between sinks s_i and s_j, where $s_i \in T$ and $s_j \in S - T$.
  $$cost(s_i, s_j)$$
- In Dijkstra's algorithm, sink s_j and edge $e(s_i, s_j)$ are selected to minimize the path cost between source s_0 and sink s_j, where $s_i \in T$ and $s_j \in S - T$.
  $$cost(s_0, s_i) + cost(s_i, s_j)$$
- To combine the two objectives, ***PD Tradeoff*** iteratively adds the sink s_j and the edge $e(s_i, s_j)$ to T such that
  $$\gamma \cdot cost(s_0, s_i) + cost(s_i, s_j)$$
  is minimum over all $s_i \in T$ and $s_j \in S - T$ for a prescribed constant $0 \le \gamma \le 1$.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, just if you look at it in Prim's algorithm the weight works is as follows. That at every step you just add a vertex s j like say j is the sink s j and s i is another node you just add a vertex such that cost of s i and s j is minimum. This is my cost function. So given a graph I am trying to add a vertex such that it is distance from any of the other vertices is minimum not necessarily from the source vertices, that is Prim's algorithm. So this s i can by any one of the vertices in the existing partial tree and I am adding another node s j. So I am trying to minimize. I am to select that s i for which the s i, s j cost is minimum and Dijkstras algorithm here we are trying to minimize the cost of this s j with respect to the original source S 0. So whenever we add an edge s i, s j cost will be actually the cost from S 0 to si plus cost of s i to s j this is Dijkstra.

So, just looking at these 2 the combination or the tradeoff says that you modify the cost function by adding a factor gamma, just write gamma into cost S 0 s i plus cost of s i s j. So you can see if your parameter gamma is equal to 0, the cost function becomes same as the Prims algorithm. And if gamma equals to 1 it becomes same as Dijkstras algorithm. And any value between 0 and 1 it will actually represent a tradeoff. So if it is closer to 0 closer to 0 means it will be closer to Prims if it is closer to 1 means it will be closer to Dijkstras.

(Refer Slide Time: 28:25)

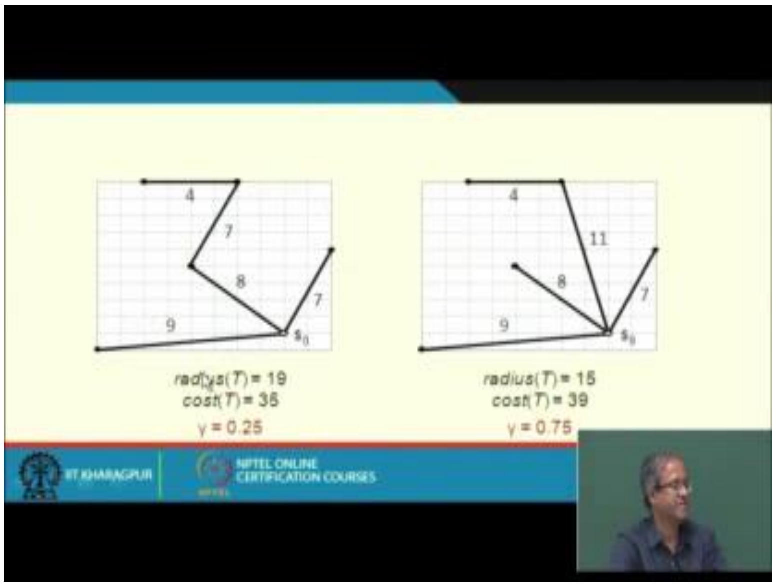


The Tradeoff

- When $\gamma = 0$, the PD tradeoff is identical to Prim's algorithm, and T is a minimum spanning tree T_M.
- As γ *increases*, the PD tradeoff constructs spanning trees with progressively higher cost but lower radius.
- When $\gamma = 1$, the PD tradeoff is identical to Dijkstra's algorithm, and T is the shortest-paths tree T_S.

So, you can have a tradeoff like this. This is what I mentioned for gamma equals to 0 is identical to Prims algorithm gamma equal to 1 it becomes Dijkstras algorithm, but in between it is a tradeoff.

(Refer Slide Time: 28:39)

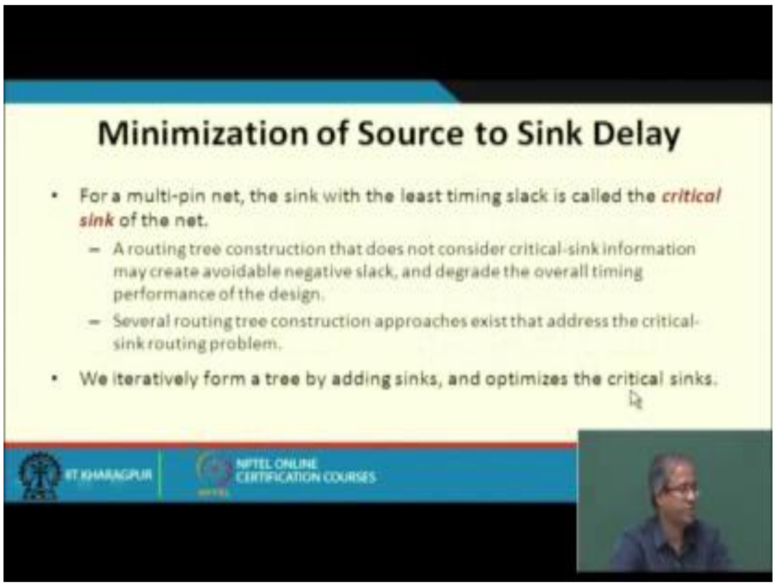


$radius(T) = 19$
$cost(T) = 35$
$\gamma = 0.25$

$radius(T) = 15$
$cost(T) = 39$
$\gamma = 0.75$

So, just an example is shown here. For gamma equal to 0.25 for a certain example you get a tree like this, but if you take a larger value of gamma equal to 0.75 you get a tree like this. So lower value of gamma it becomes more like a minimum spanning tree, but higher value of gamma it becomes more like a minimum radius tree. So you see this tree looks like be less spread as compared to this.

(Refer Slide Time: 29:13)

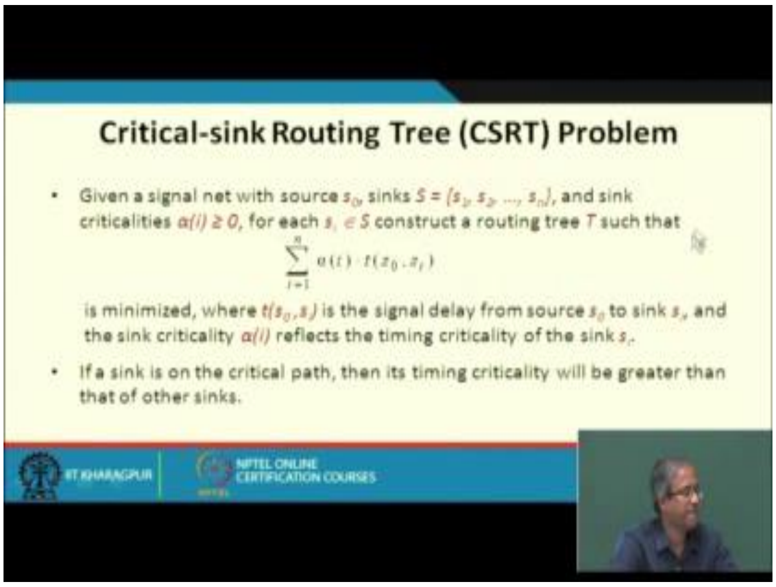


Minimization of Source to Sink Delay

- For a multi-pin net, the sink with the least timing slack is called the ***critical sink*** of the net.
  - A routing tree construction that does not consider critical-sink information may create avoidable negative slack, and degrade the overall timing performance of the design.
  - Several routing tree construction approaches exist that address the critical-sink routing problem.
- We iteratively form a tree by adding sinks, and optimizes the critical sinks.

So, there are a few other parameters also which can be used. Like for a multi pin net whether a multi pins in general most of the pins are multi net, so this sink which has the least timing slack, we identify that sink as the critical sink. So here what you are saying is that, we are considering a routing tree that does not consider critical sink information. If we do this then we may be constructing a tree where some critical parts still remain some slack may become negative, which may result in the overall degradation in the timing performance. So here minimization of source of sink delay means we are keeping track of the critical sinks. And we are iteratively adding nodes to the tree keeping in view that which of the sinks are critical.

(Refer Slide Time: 30:16)



Critical-sink Routing Tree (CSRT) Problem

- Given a signal net with source s_0, sinks $S = \{s_1, s_2, ..., s_n\}$, and sink criticalities $\alpha(i) \geq 0$, for each $s_i \in S$ construct a routing tree T such that

$$\sum_{i=1}^{n} \alpha(i) \cdot t(s_0, s_i)$$

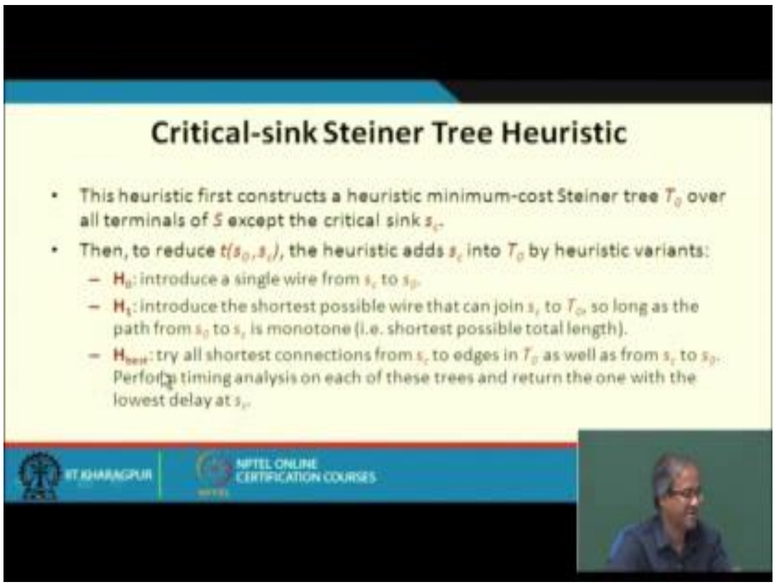
is minimized, where $t(s_0, s_i)$ is the signal delay from source s_0 to sink s_i, and the sink criticality $\alpha(i)$ reflects the timing criticality of the sink s_i.

- If a sink is on the critical path, then its timing criticality will be greater than that of other sinks.

Like here we are defining a criticality factor, say alpha i, for every sink s i so the sinks which are critical they will be having a higher value of alpha i and the total cost of the tree we are computing as not only just sum of the h costs, but alpha i multiplied by the h cost. So the edges which are critical they will be having a higher contribution to this total summation. So this is the basic idea.

So, if you try to minimize this criterion so automatically you are the sinks which are critical they will automatically be try to connect to some other node with a lower edge cost such that this overall sum is minimized.

(Refer Slide Time: 31:16)

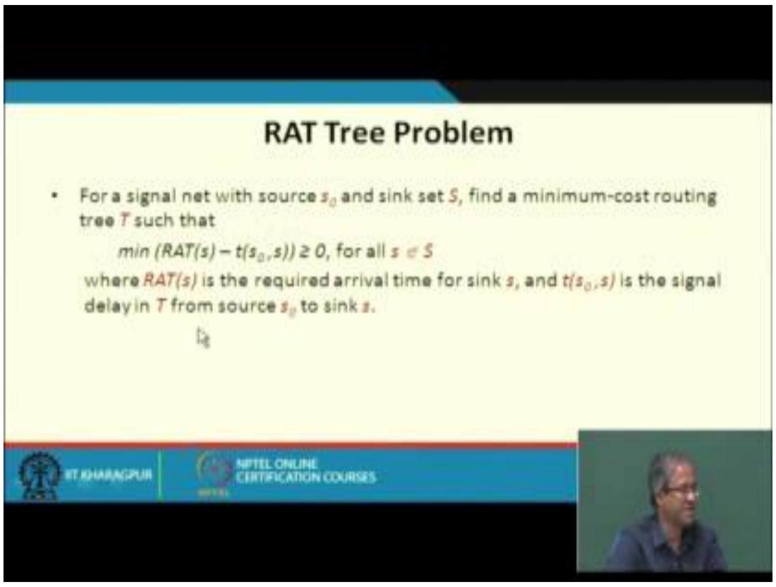


Critical-sink Steiner Tree Heuristic

- This heuristic first constructs a heuristic minimum-cost Steiner tree T_0 over all terminals of S except the critical sink s_c.
- Then, to reduce $t(s_0, s_c)$, the heuristic adds s_c into T_0 by heuristic variants:
  - H_0: introduce a single wire from s_c to s_0.
  - H_1: introduce the shortest possible wire that can join s_c to T_0, so long as the path from s_0 to s_c is monotone (i.e. shortest possible total length).
  - H_{best}: try all shortest connections from s_c to edges in T_0 as well as from s_c to s_0. Perform timing analysis on each of these trees and return the one with the lowest delay at s_c.

Similarly, you can have a critical sink Steiner tree heuristic. So here what it does it first constructs a minimum cost Steiner tree excluding the critical sink. Suppose there is a single critical sink s c it first constructs a Steiner tree say excluding the critical sink, then it uses some heuristic to add s c to the existing to where to add. There are 3 heuristics which are proposed. One heuristic is you directly connect s c to S 0 this source. Second one says you introduce the shortest possible wire that can connect sc to the existing tree. So that the total wire length still becomes minimum this is called a monotone property. And the third one says you try all possible connections try to connect to all possible points in that tree T 0 and for each of these you do timing analysis to find out that for which the delay is minimum so use that.

(Refer Slide Time: 32:24)



RAT Tree Problem

- For a signal net with source s_0 and sink set S, find a minimum-cost routing tree T such that

  $min\ (RAT(s) - t(s_0, s)) \geq 0$, for all $s \in S$

  where $RAT(s)$ is the required arrival time for sink s, and $t(s_0, s)$ is the signal delay in T from source s_0 to sink s.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, there are other methods also like one which uses the required arrival time. So it tries to compute some kind of a slack like this and try to minimize this. So these I am not going into the detail.

So, with this we come to the end of this lecture. We have looked at some of the timing driven routing strategies and techniques that can be used in the routing phase to keep the timing constraints in check, or they are not getting violated.

Thank you.