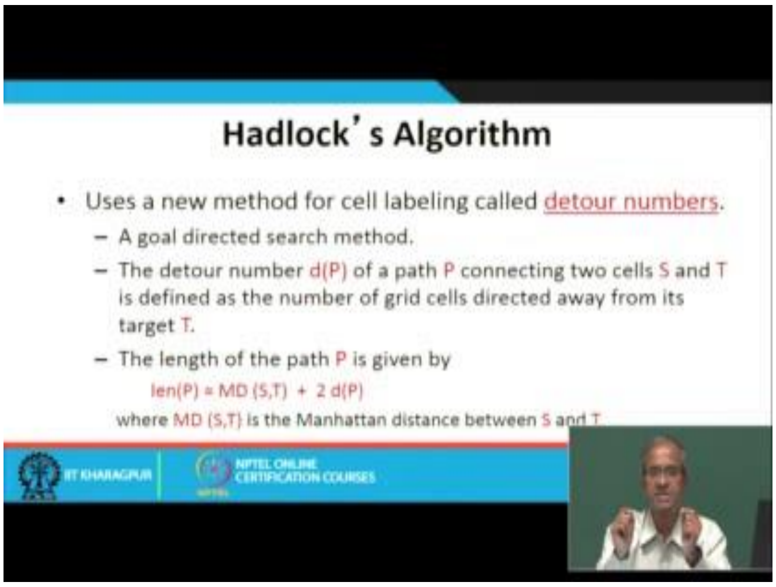
VLSI Physical Design

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

Lecture – 17
Grid Routing (Part III)

So, we continue with our discussions on the Grid Routing algorithms. So, as I said we shall be discussing some more algorithms in this lecture which are more efficient in terms of the running time also the storage complexities. Let see.

(Refer Slide Time: 00:38)



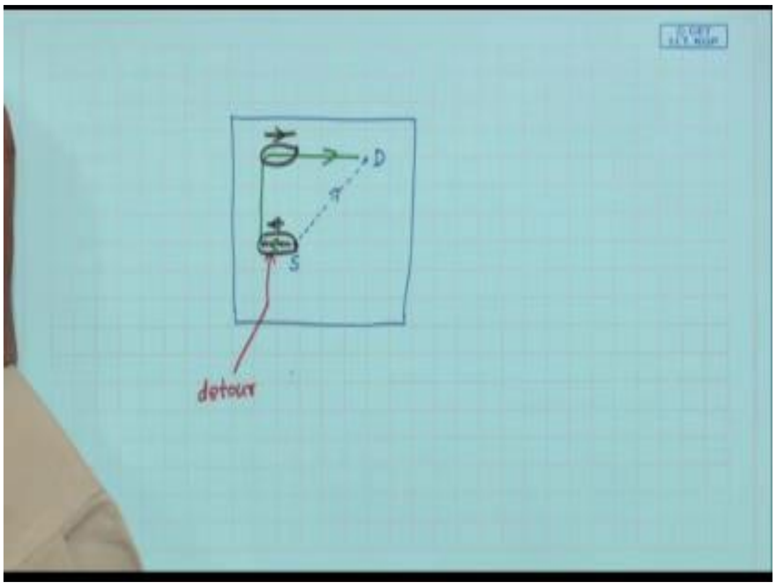
Hadlock's Algorithm

- Uses a new method for cell labeling called detour numbers.
  - A goal directed search method.
  - The detour number d(P) of a path P connecting two cells S and T is defined as the number of grid cells directed away from its target T.
  - The length of the path P is given by
    len(P) = MD (S,T) + 2 d(P)
    where MD (S,T) is the Manhattan distance between S and T

The first one is called Hadlocks algorithm. This this method let me tell you first. This is similar to Lee's algorithm, in the sense that here also we are labeling some cells with numbers, but the wave we are labeling, the way the wave fronts are progressing are very different because in Lee's algorithm you just try to recall the basic mechanism you are trying to propagate the numbering or the wave front in all directions without any consideration as to which direction the target is actually located.

So, if the target was located there you are also propagating wave front in the other direction in Lee's algorithm, but in Hadlocks algorithm you specifically keep track of a of the fact that whether you are moving in the right direction or in the reverse direction. You always give priority if you are moving in the right direction. So, you are avoiding unnecessary expansion of the wave fronts in the reverse direction which will; obviously, require much less number of cells to be numbered which of course, will result in much faster and times, but let see the basic concept here. So, this cell labeling scheme that is used in Hadlocks algorithm, I mean use a something called detour numbers.

(Refer Slide Time: 02:28)

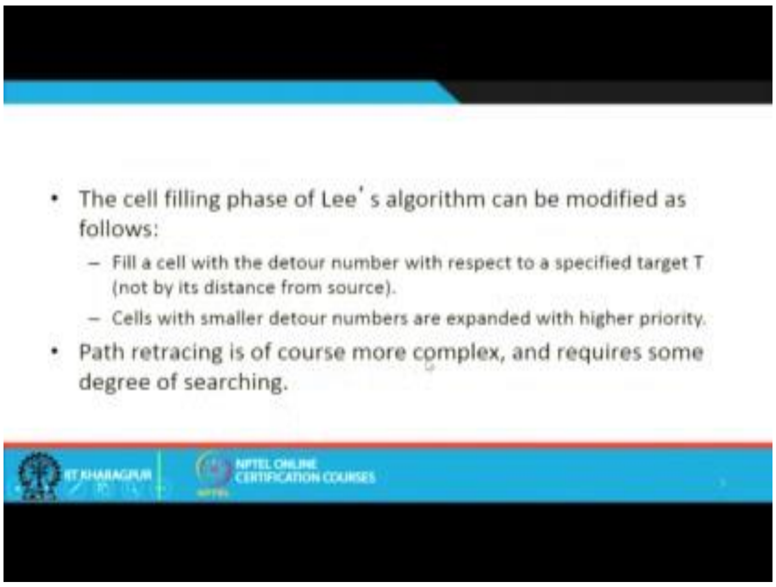


Now, what is a detour? Let me just this explain the concept of detour with the help of a diagram first let say. So, I have a source here I have a destination here. So, I want to find out a path from S to D like this. Now suppose due to some reason in the, I cannot find a path like this. I may see that in order to find a path, I may have to go away from d then again I may have to come back towards T a path like this. Now, this move where I moving away from D this is called a detour, here I am moving away from D, but other than this segment you see nowhere else in this path we are using a detour. Say from here we are moving in the right direction, here also we are moving in the right direction, but here we are having a detour.

Now, the number of cells that we are moving away from with respect to D that is called the detour number, this is the idea. So, the detour number let see, the detour number of a path P that connects 2 cells is defined as the number of grid cells directed away from the target. Now you look at the entire path you find out in how many grid cells you are actually moving away from the target that will be your number d P. Now one thing also you try to understand now in this example. So, you are moving away in this path. So, the amount of movement you are doing away you will have to come back by the same amount again. So, you were moving away you will again have to come back.

So, that this amount of d p your moving away that d p also will have to added while you are coming back. So, that is why the length of the path with respect to the detour number here you can estimate as the Manhattan distance between S and T which is the expected shortest path which may not be there, that Manhattan distance plus twice the detour number. So, once you go to the away direction will have to again come back by that amount. So, this is how you are estimating the length of the path, and the idea behind Hadlock's algorithm is that you use the detour numbers to label the cells.

(Refer Slide Time: 05:38)

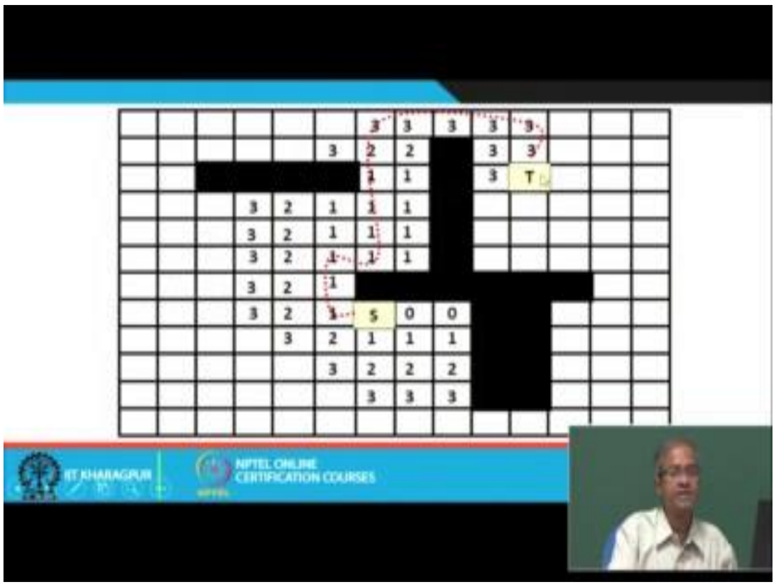


- The cell filling phase of Lee's algorithm can be modified as follows:
  - Fill a cell with the detour number with respect to a specified target T (not by its distance from source).
  - Cells with smaller detour numbers are expanded with higher priority.
- Path retracing is of course more complex, and requires some degree of searching.

So, the cell filling is similar to Lee's algorithm. The difference is that you fill the cell with the detour number and not by it is distance. And this cells which are having smaller detour numbers and expanded with higher priority.

And this another point here. To see in Lee's algorithm at every step we are labeling a neighbor, in the next step the neighbor of the neighbor, but in Hadlock's algorithm we go and labeling as long as you are moving in the same direction in the same step. Like for example, as long as where you are moving in the right direction that will mean detour number of 0. So, if I see I moving in the right direction I can continually label the adjacent cell 0 0 0 0 0 0 in a single step itself. This is the idea, but; however, here the path retracing is not so simple. So, you need some data structure you need to keep track of some additional information and some degree of searching also.

(Refer Slide Time: 06:58)



This is the price your p. So, let us illustrate this Hadlock's algorithm for this example. Where here again we are showing this array of grid cells with the black regions are interfiled as obstacles. This is the source this is your target. So, starting from S we start labeling this cells with detour numbers.

For look at one thing the S is here T is to the right and to the top of S. So, any movement towards right or towards top will not increase the detour number. So, it means you are moving in the right direction, but the top is blocked you cannot move on the top. The only way in which you can move in the right direction is towards right. So, in this way you can move by 2 cells here and here after that again you stop. You cannot move in the right direction because right and top both are blocked.

So, in the first step you are labeling like. This detour number zero; that means, you are moving in the right direction you can move only up to this much. After this you cannot move any further. So, you will have to make a detour. So, what can be detour you can make? So, from 0 we can move this side. 0 you can move this side, from S you can move this side, from S you can move to the left. This all will mean a detour of 1, mean; that means, one cell you are moving means away from T which means like this not only this. So, once you have moved one cell to the left of S and label this cell as 1, you I mean as I said that you are free to label this cells as long as you are moving in the direction not one as many as you can.

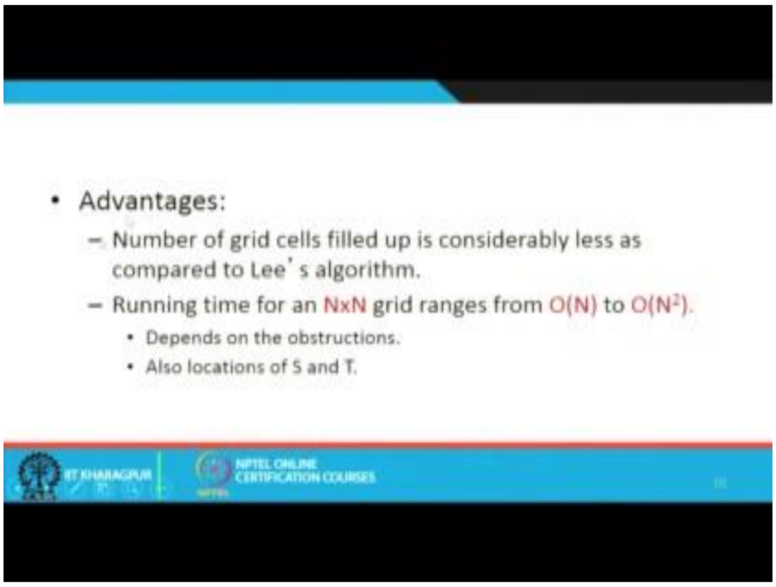
So, you continue the process from here. So, as soon as you label this particular cell has 1, from here you can see your T is to the right and top. So, any cell to the top or right you can go on labeling with 1 1 1, because you are not making any other detour. Detour is not changing d P is not increasing. So, right you go on labeling till you reach here, but if you reach any further this will involve another detour because you are going above T. So, now, these will become 2 labels if you want to move up. So, you go and go on labeling here as long as your detour number does not increase.

So, you feel so many cells in the second step. After you have done this the third step you have to take another detour; that means, you have either move up or in this case move to the left or in this case move down. So, like this, this cells you will be marked as 2, 2 here. Now you see from here again you have take another detour to reach T because there is no other path in the right direction. You have to move up or here or here all 3, but you see this 2 if you move here has 3 in this cell, from there your T will be on the right and down say you can continue with 3 3 3 3 3 like this; that means, you will get something like this.

So, here detour number 3 you are getting, but here as you get 3 you can continue with 3 because this is no further detour here. So, you have got a path, the path will look something like this, so where you have taken 3 detours. One detour here another detour here another detour here. So, your total path length will be Manhattan distance between S and T plus twice into d P which is 3, but as I had said in this case the retracing is not the trivial. You have to do some kind of searching of this cells which you have labeled. So, I am not going into the detail of that, but just remember that you retracing is a little more complex here, but the advantage is that you are means at every step you are labeling a number of cells as long as your moving in the right direction.

You are not spreading unnecessarily the labels in the wrong direction. You are moving away only when required. Only when you see that you cannot move in the right direction. Then only you move away. This is the basic concept behind Hadlock’s algorithm in the performance is means of course, better then Lee's algorithm because it labels much less number of cells in general and runs faster naturally, but the problem is that the memory complexity or storage complexity does not change much because here also you are keeping the data structure as have 2 dimensional array.

(Refer Slide Time: 12:33)

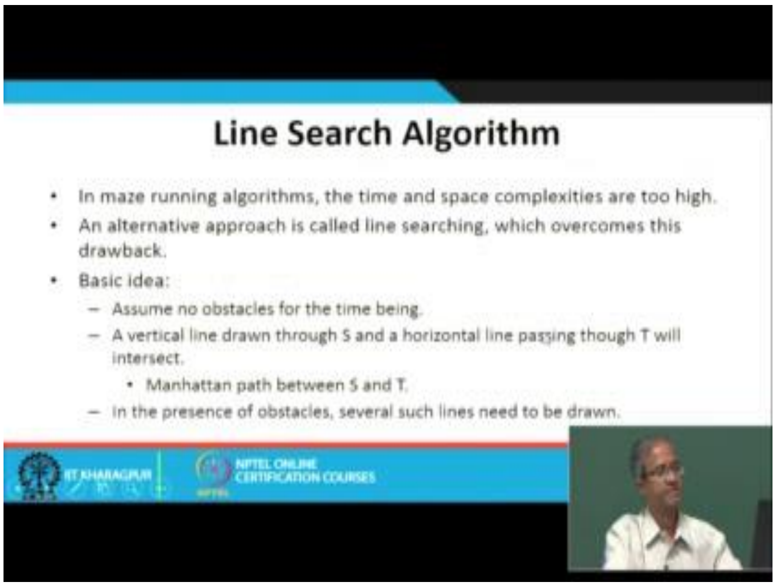


- Advantages:
  - Number of grid cells filled up is considerably less as compared to Lee's algorithm.
  - Running time for an NxN grid ranges from O(N) to O(N^2).
    - Depends on the obstructions.
    - Also locations of S and T.

So, if it is a 2000 by 2000 grid you have to maintain the 2000 by 2000 array to store this status of each of this cells right.

So, these are the advantages these are mentioned, and running time of course, it is difficult to analyze exactly, but it has been found that for n by n grid the running time for connecting 2 points ranges from means order n which is the linear time algorithm to order n square. It of course, depends on where the obstructions are located which are the positions of S and T and so on. Now, but this is the average case.

(Refer Slide Time: 13:04)



Line Search Algorithm

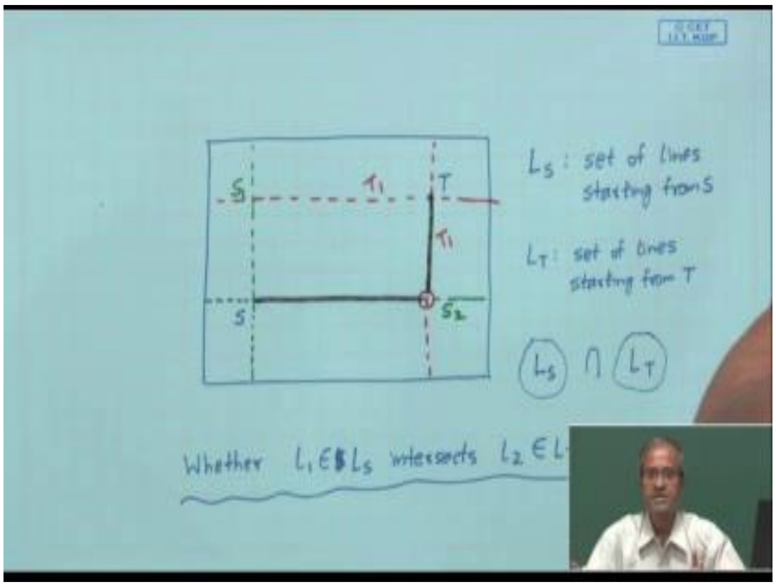
- In maze running algorithms, the time and space complexities are too high.
- An alternative approach is called line searching, which overcomes this drawback.
- Basic idea:
  - Assume no obstacles for the time being.
  - A vertical line drawn through S and a horizontal line passing though T will intersect.
    - Manhattan path between S and T.
  - In the presence of obstacles, several such lines need to be drawn.

Now, let us come to an alternate class of algorithms line search algorithm. Let me try to tell you the basic concept here first. Now in the maze learning algorithms what we are doing we were basically representing the layout surface of the 2 dimensional array where each element of the array represented as cell and we were labeling the cell and carrying out some kind of a search. So, that from source we reach the target right, but in case of line search algorithm the concept is somewhat different. I am not representing the area as a 2 dimensional array any more. I only have the total area available to be here.

Here we maintain information about a number of straight lines. So, our data structure here will be different. You have to maintain number of straight lines straight line can be represented by the n coordinates to coordinates means the straight line and also. You will have to search whether 2 straight lines are intersecting each other.

If the intersect what is the point of intersection. Well you can your simple coordinate geometry for that you already know this is means school math staff that is easy. So, let us try to explain the motivation behind it before I explain the salient details.

(Refer Slide Time: 14:42)


S
T₁
T
T₁
S
S₂
Lₛ : set of lines starting from S
L_T : set of lines starting from T
Lₛ ∩ L_T
Whether L₁ ∈ Lₛ intersects L₂ ∈ L

The idea is that suppose I have a rectangular area in which I have to lay out the points. Let say I have a source here I have a target here. Well in the earlier case what we are doing we were propagating the wave forms labeling the adjacent cells in a grid. So, that the numbers finally, will reach T, but here we use a very simple concept just assume for the timing that there are no obstacles well in presence of obstacles we shall see later.

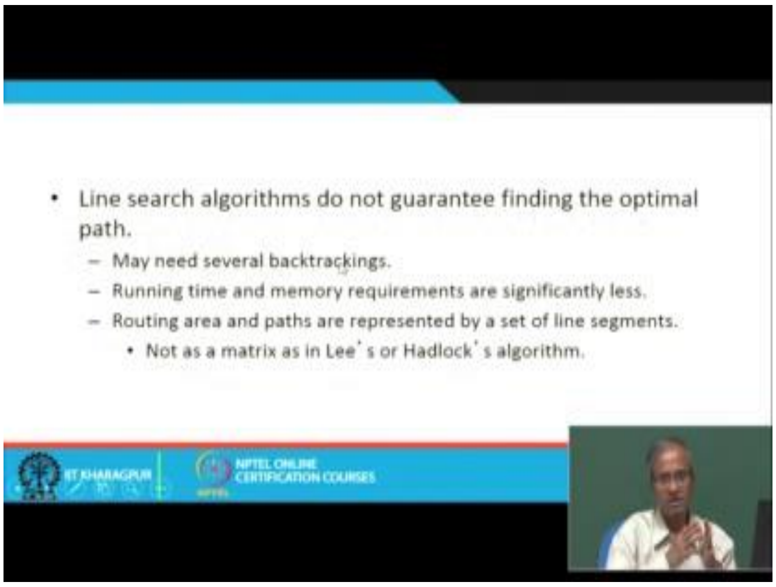
What I do from this this point S you just imagine 2 straight lines one horizontal and one vertical. These 2 straight lines are coming from S call them S 1 and S 2. Similarly, from T you start in a similar way a horizontal and a vertical straight line call them T 1 and T 1. Now if we find by well here just again talking about the data structure you have L S. L S is set of lines starting from S. Similarly, you have L T which is the set of lines starting from T. Now at every stage we check whether any line from the set L S whether the intersect L T or not. We say I am not saying this is set intersection what you mean to say is that is line from this L whether intersects it; that means, what you are checking whether let say L 1 belongs to S belongs to L S, sorry L S intersects another line L 2 which belongs to L T.

So, we are checking this at every step. There can be multiple intersections like in this case one intersection is happening here one intersection is happening. Let say we take this as the intersection, once we find the intersection we can trace back from there up to T and up to S. We can trace the red lines to go up to T we can trace the green lines to go up to S. So, you see what you have we have already got a path between S and T. The concept is very simple this is how we get a path between S and T. The advantage you can immediately see. So, we are not labeling cells we are not using very large storage. Only in in this particular example we have to keep track of 4 straight lines, means 4 into 2 8 coordinates. And some simple coordinate geometry algebra to detect whether the lines are crossing or not if they cross what is the coordinate.

So, once we have the coordinate you can get the line segments that constitute the path of the obstacles. Of course, you have to keep some information about the obstacles in your data structure that these are the places where some obstacles are there. So, your data structure will be slightly more complex then straight lines not only straight lines you also have to represent some rectangles which represent some obstacles already placed you cannot drive a line across an obstacle, that you have to follow some other path. Let us come back to this now. So, in this line search algorithm the basic idea is that just the example that I have shown. So, here is state the something.

So, assuming no obstacles a vertical line through S and a horizontal line through T will intersect and vice versa. So, in semi some horizontal line through a S and a vertical line to through T can also intersect. And once we get an intersection you get the Manhattan path between S and T that will be the shortest path, but when there are obstacles the complexity increases and you may have to draw several such set of lines.

(Refer Slide Time: 20:15)

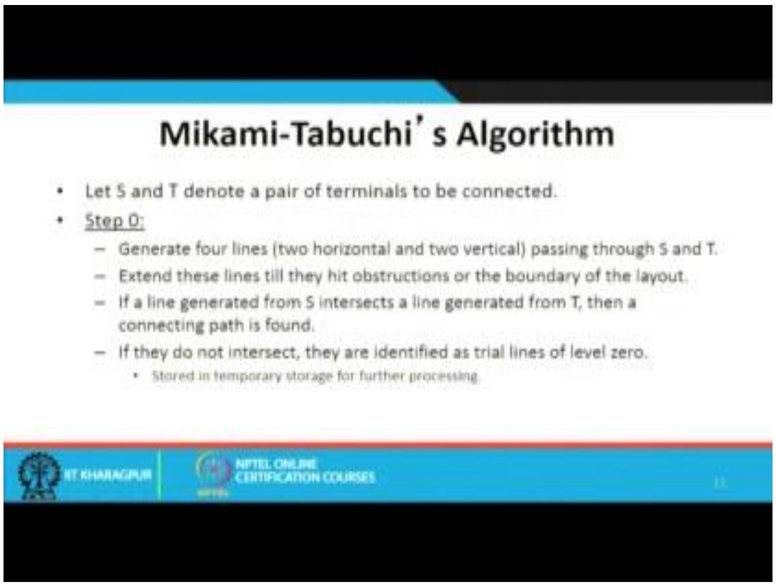


- Line search algorithms do not guarantee finding the optimal path.
  - May need several backtrackings.
  - Running time and memory requirements are significantly less.
  - Routing area and paths are represented by a set of line segments.
    - Not as a matrix as in Lee's or Hadlock's algorithm.

Now, one point to note in this line search algorithms are there they are very fast and so, most of the cad algorithms which do area routing they rely on these line search algorithms, but these algorithms do not always guarantee generating the best path of the shortest path. And Secondly, they may need backtrackings because you are following a path you fine in that you have reach the dead end you may have to go back and try an alternate path.

Such scenarios can also occur and as I had said the routing area and also paths are represented by the set of lines and not as a 2 dimensional matrix as in the maze routing algorithms.

(Refer Slide Time: 21:09)



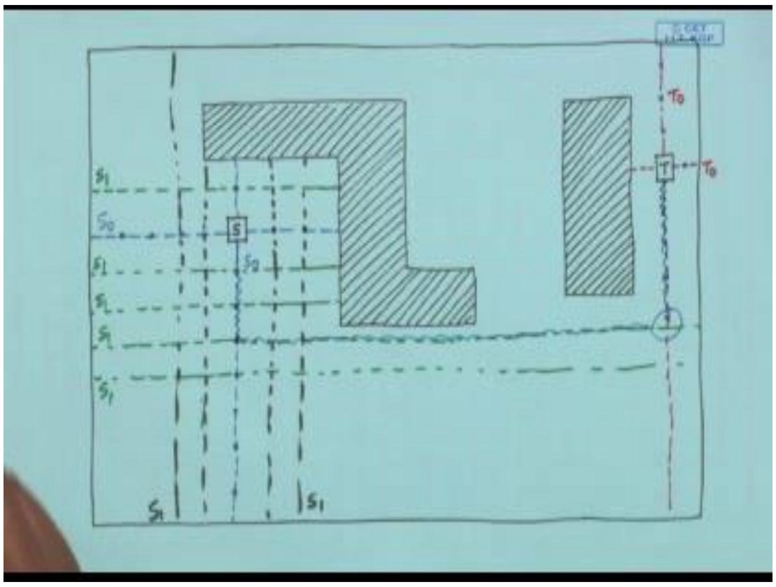
Mikami-Tabuchi's Algorithm

- Let S and T denote a pair of terminals to be connected.
- Step 0:
  - Generate four lines (two horizontal and two vertical) passing through S and T.
  - Extend these lines till they hit obstructions or the boundary of the layout.
  - If a line generated from S intersects a line generated from T, then a connecting path is found.
  - If they do not intersect, they are identified as trial lines of level zero.
    - Stored in temporary storage for further processing.

So, we shall be looking at 2 different algorithms the first one we refer to as Mikami and Tabuchi algorithm. These are the 2 persons who propose this algorithm. So, here let me just explain this step then I will illustrate with an example. So, the 2 terminals to be connected are S and T. So, the initialization step let us call it step 0, generate 4 lines 2 horizontal and 2 vertical passing through S and T. Let us jut work step by step with an example. Generate 4 lines passing through S and T.

Let us take an example like this. See this example this example shows source point target point and some obstacles already there.

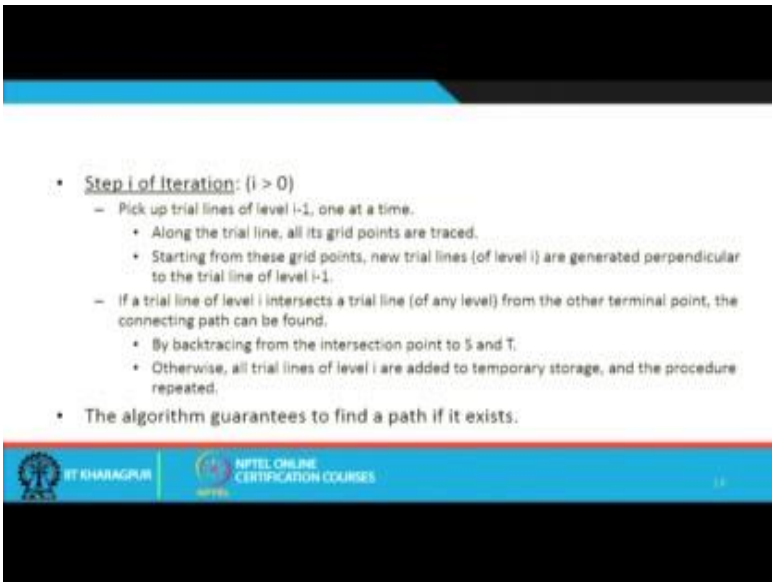
(Refer Slide Time: 22:05)



So, the step 0 says we generate 4 lines. So, I generate a horizontal line passing through S like this and a vertical line passing through this. I call them S 0 and S 0. Similarly, from the target I am using a different color a horizontal line and a vertical line. Let us again call it T 0 and T 0. This is the first step. Now as you can see because of the obstacles this lines are not intersecting because this lines are getting stopped. So, you need some more iterative steps. So, what next? The next step says to extend this lines till the hit obstructions or the boundary of the layout. If a line generated from S intersects a line generated from T, then a connecting path is found.

So, you see here we have extended this lines till they either hit the boundaries of the obstructions, or the boundaries of the layout. And if the intersect which they are not. So, if they would have intersected I would have, for example, if the target was here then this horizontal line would have intersected here directly. So, I would have got a path, but it has not intersected so, far. So, these 4 lines they are considered as trail lines of level 0, this 0, and the suffix indicates the level. So, what is mentioned here, if they do not intersect in this example it is so, that identified as trail lines of level 0 which we store temporarily for further processing we have to manipulate this 4 straight lines further.

(Refer Slide Time: 24:33)



- <u>Step i of Iteration</u>: (i > 0)
  - Pick up trial lines of level i-1, one at a time.
    - Along the trial line, all its grid points are traced.
    - Starting from these grid points, new trial lines (of level i) are generated perpendicular to the trial line of level i-1.
  - If a trial line of level i intersects a trial line (of any level) from the other terminal point, the connecting path can be found.
    - By backtracing from the intersection point to S and T.
    - Otherwise, all trial lines of level i are added to temporary storage, and the procedure repeated.
- The algorithm guarantees to find a path if it exists.

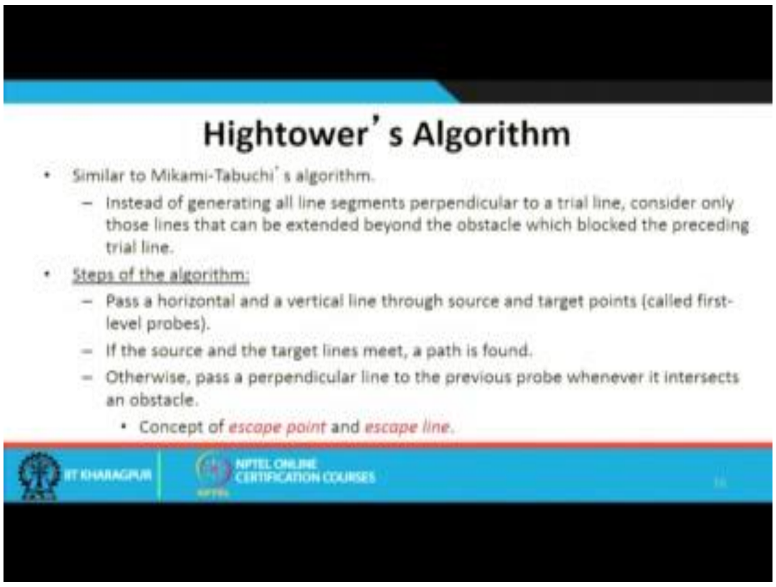
So, how do manipulate? That is your step i of the iteration where i greater than 0. Because step 0 you have already done. See the idea is simple you try to understand. Pick up trail lines of level i minus 1, one at a time. So, you are now at step 1, i equal to 1. So, you pick up trail lines of level 0 there are 4 search one at a time. So, along the trail line all it is grid points are traced. Starting from the grid points new trail lines of this new level i are generated that are perpendicular to the trail line of level i minus 1. Let us try to work out this. So, what I am saying 2 things. Along the trail line the grid points are traced and starting from this grid points new trail lines which are perpendicular to the original trail line they are generated. Let see.

So, let us look at one of the time. Let us say this one. This horizontal, let say along the grid I have one grid point let say here and one grid point here. So, the grid points are imaginary grid points are not actually stored in your so, you know that what is the minimum separation. So, you just imagine this grid point. Let say we have there are 4. Similarly, in this direction for this line there may be one grid point here. And there can be some grid points here right. Now along this grid points you through next level trail lengths which are perpendicular to this let me use a different color. Let say I use a trail length like this, I call it S 1. So, across all the grid points I am doing this I am drawing so, many trail lines. Some of them will be hitting the obstacles some of them will be crossing.

Like this will go, similarly in the in this direction they the other direction will be doing the same thing. These all will be labeled as S 1 - S 1, S 1, S 1, S 1 and S 1. Similarly, in this direction there will be trail lines like this so on. They will all be labeled again as S 1. These are all S 1. Right now you see here you need not have to the same thing you would have done here. Like here also you would have identified the grid points, and you have drawn the horizontal line vertical things, but what you see that you have already got an intersection of a red and green lines here. One coming from T and the other coming from S, this T 0 and S 1 are intersecting here. Now as you got a path intersection like this, you can trace back a path. Like up to here then you go here up to this grid point and you have got a path.

So, you see the idea is very simple. You do in this way and you will reach stage where some line starting with from T and some line starting with S will intersect. And you keep carrying out this check at every step. And once the intersect you have been being means you have obtained a path you can trace back and find a path. So, this is the essential idea behind the Mikami and Tabuchi algorithm. And this algorithm actually guarantees to find a path if it exists right. Now this example you have shown.

(Refer Slide Time: 29:09)



Hightower's Algorithm

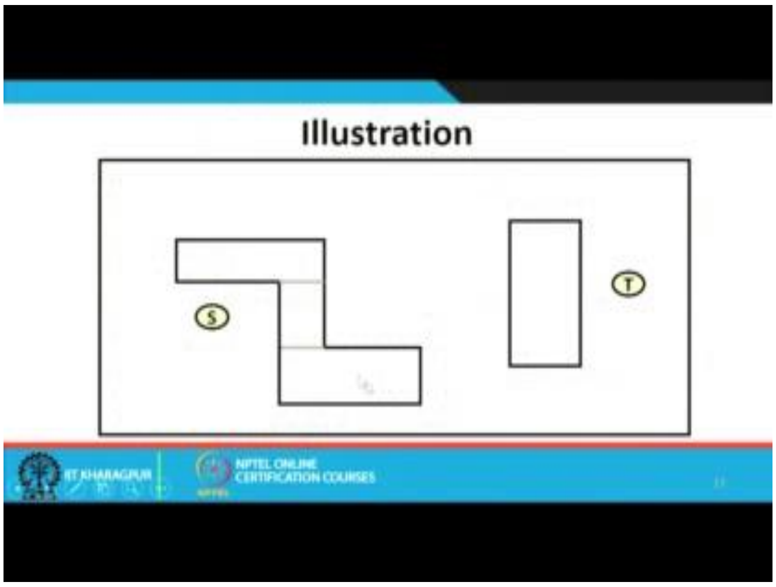
- Similar to Mikami-Tabuchi's algorithm.
  - Instead of generating all line segments perpendicular to a trial line, consider only those lines that can be extended beyond the obstacle which blocked the preceding trial line.
- Steps of the algorithm:
  - Pass a horizontal and a vertical line through source and target points (called first-level probes).
  - If the source and the target lines meet, a path is found.
  - Otherwise, pass a perpendicular line to the previous probe whenever it intersects an obstacle.
    - Concept of *escape point* and *escape line*.

Now, you see this is an improvement over the Mikami Tabuchi algorithm. In the Mikami Tabuchi algorithm along the lines I am looking at every grid points and I am running, so many perpendicular lines along each of the grid points. So, what

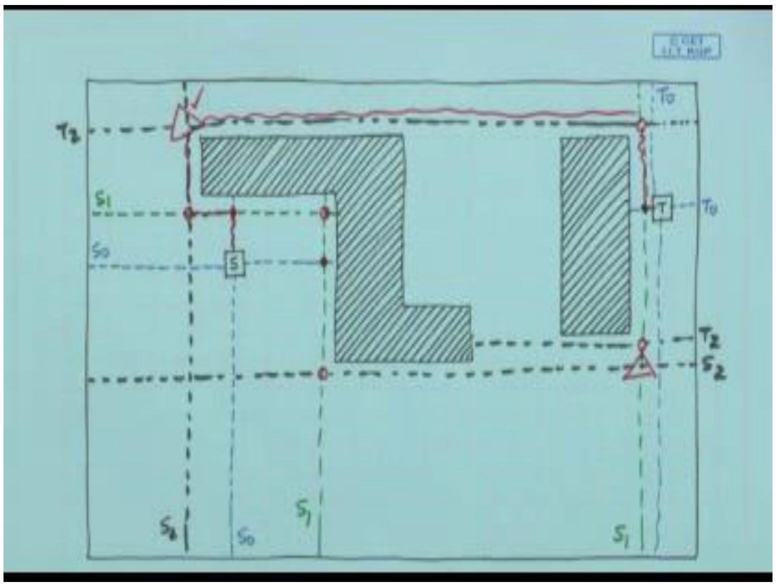
Hightower's of algorithm says is that you will need not have to run so many perpendicular lines. Number of lines can be drastically reduced.

So, very quickly let see this steps. So, instead of generating all line segments that have perpendicular to a trail line, you consider only those lines that can be extended beyond the obstacle which has blocked the line, blocked the preceding line. So, you try to find out something called escape point and escape line and I will explain this with example. So, you try to find out escape point and escape line and you draw the perpendicular lines only at the escape points. This steps of the algorithm are similar. You pass a horizontal and vertical line through source and target, if first level probes if they meet path is found. Otherwise pass a perpendicular line to the previous probe line, but not at all grid points only at the escape points.

(Refer Slide Time: 30:33)



(Refer Slide Time: 30:38)



Let see, the concept with the same example let see. So, we consider this example here, again the same example. So, the first step is same as in Mikami and Tabuchi algorithm we run, horizontal and vertical line both through S and T. Well I am showing it in the color. So, the labels will be S 0 this will be S 0 this will be T 0 this will be T 0. Now let us try to understand the escape points. See escape points means see this line, this line is hitting this obstacle, just a grid point just before the obstacle that is identified as an escape point. So, now, here it is setting similarly this line is hitting the obstacle. So, your line just before that you can take this as an escape point. Similar is the case the vertical line does not hit any obstacle.

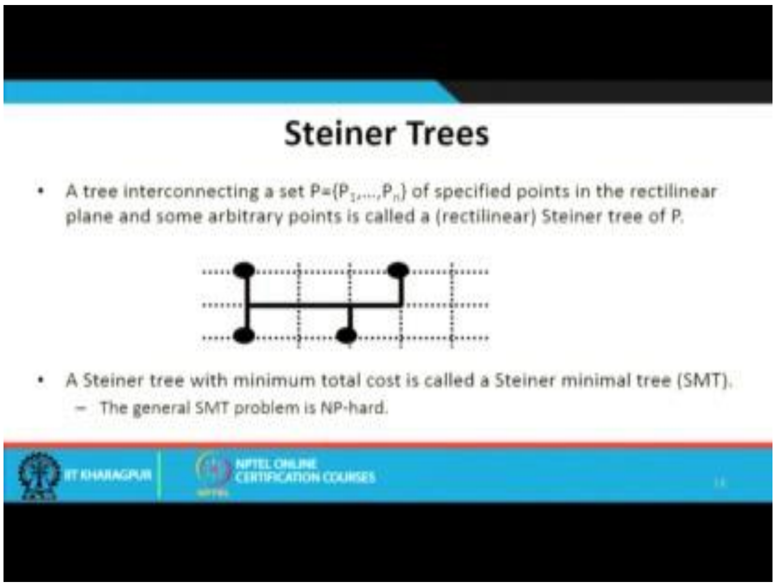
So, you can take grid point just here. So, on these grid points you run perpendicular lines. You run perpendicular line like this call this S 1. Call a perpendicular line draw a perpendicular line here call this S 1. Draw another perpendicular line here call this also S 1. Now with respect to these perpendicular lines you identify escape point like this. You see this line S 1 this is running very close to the obstacle parallel, but it is not hitting it anywhere else, well it is hitting it here. So, means will be getting one escape point here and another thing you also check where it is just crossing the boundary of the obstacle it is here. So, just beyond that this will be another escape point defined. Similarly, for this line S 1. So, it is hitting here of course, and it is here it is just crossing the boundary.

So, there will be one escape point here. Similarly, for this green line S 1. So, there will be one escape point defined here where it is just crossing this another escape point defined here just crossing this. So, on this escape points you defined another set of perpendicular lines. So, for this the perpendicular line will be like this. This will be your S 2. On this your perpendicular line will be this. This will also be S 2. So, on this the perpendicular line will be like this. So, this process will continue this will be T 2 and here it will like this. So, now, you have seen that that one line from S and one line from T has intersected now they in fact, many search. So, does one search intersection is here one search intersection is here you can take any one of them.

For example, if you take this one this intersection let say, then your path will be like this. Then your path will be via this, so, you can see. So, in higher algorithm you can generate some paths which run close to the boundaries of the obstacles, which will not unnecessarily congest the remaining path it because if you run a wire like this it can unnecessary stop the other wires from coming in this area. So, it is trying to trace the paths using some lines which are running parallel and close to the boundaries of the obstacles. So, this is the basic idea behind Hightower's algorithm you can see, there is a number of lines you are considering is much less then Mikami Tabuchi algorithm.

But the only computation you have to do is that you have to identify the escape points. So, this is also not very not computationally involved. You have the data structure to store the rectangular obstacles and just using simple coordinate geometry you can identify this escape points, right.

(Refer Slide Time: 35:55)



Steiner Trees

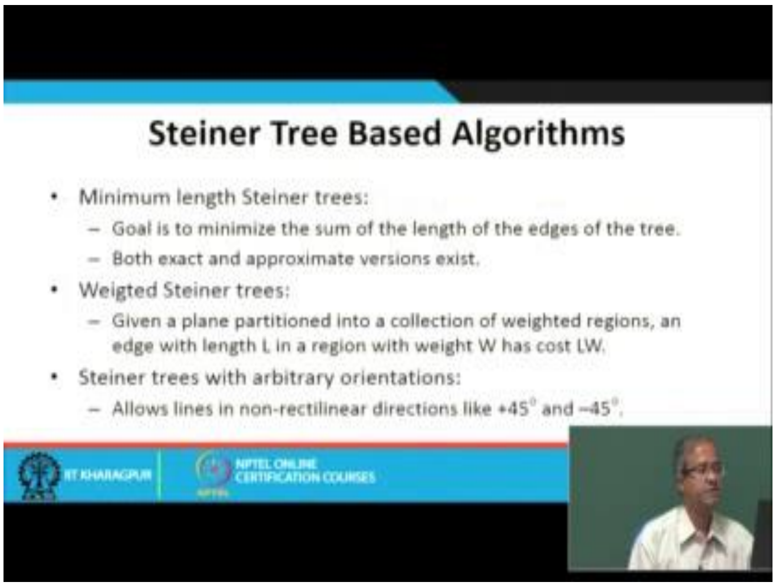
- A tree interconnecting a set $P=\{P_1,\ldots,P_n\}$ of specified points in the rectilinear plane and some arbitrary points is called a (rectilinear) Steiner tree of P.
- A Steiner tree with minimum total cost is called a Steiner minimal tree (SMT).
  - The general SMT problem is NP-hard.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, lastly we look at very briefly some something called Steiner trees which we mentioned earlier. Steiner tree means a set of points interconnected by horizontal and vertical segments is called rectangular Steiner tree. Where some bends can be there allowed even at points where there are terminals like here.

Now the advantage of Steiner tree is that this is how we actually do the layout and the interconnection length is typically less as compare to simple point to point routing.

(Refer Slide Time: 36:53)



Steiner Tree Based Algorithms

- Minimum length Steiner trees:
  - Goal is to minimize the sum of the length of the edges of the tree.
  - Both exact and approximate versions exist.
- Weigted Steiner trees:
  - Given a plane partitioned into a collection of weighted regions, an edge with length L in a region with weight W has cost LW.
- Steiner trees with arbitrary orientations:
  - Allows lines in non-rectilinear directions like $+45^\circ$ and -45°.

But the problem is that the general shortest Steiner tree. Steiner minimal tree problem detection of that is computationally complex. So, there are Steiner tree base algorithms we are not going to detail of this here. So, here we are trying to address the goal to minimize this some of the lengths of edges of the tree. Well exact versions exist, but they as I said it is a np hard problem. You can get the exact solution only for very small problem instances, but for larger problems in approximate heuristics exist. And also you can a weighted Steiner tree is means instead of giving equal weights to all the edges, you multiply the weight with the length of a segment with a weight parameter W. This weight indicates the congestion of that area. So, if you are running a net through our region which is highly congested that weight value should be higher you should try to avoid the congested regions.

And some works have been done where Steiner tree with arbitrary orientation. Particularly diagonal connections 45-degree angle connections are also allowed. So, there are some grid routing algorithms which are also based on Steiner tree.

So, with this we come to the end of this lecture.

Thank you.