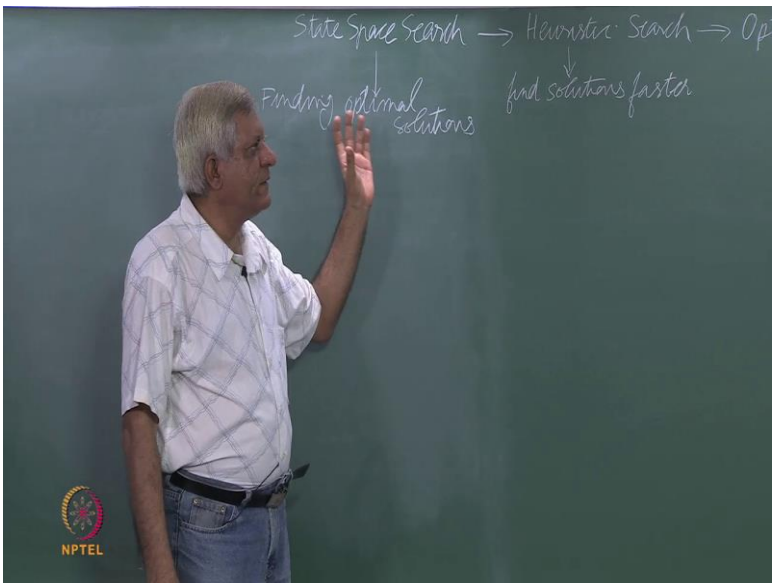
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science
Indian Institute of Technology, Madras

Lecture - 18
Branch and Bound, Dijkstra's Algorithm

Today, we are going to shift focus, a little bit to see, where we are going. Let us just do a quick recap of what we have done so far.

(Refer Slide Time: 00:24)



We started with state space search and then, we went on to heuristic search and from there, we went to optimization. The idea was, that the idea of optimization came in, because we said that instead of looking for the goal state, we will try to optimize the heuristic value of the evaluation function, or the objective function, as the case maybe. The idea of using heuristics was to find solutions faster, because we discovered that search spaces tend to grow exponentially. We were looking for a function or some kind of heuristic knowledge, which comes from the domain, which will guide the search towards the solution, and not go off in other directions, in the hope that we will find it faster, essentially, because we do not want to run into exponential times.

Today, we want to shift focus and look at the other aspects. So, once you find a solution, let us say, you are solving problems in some domain, or let us say it is a logistics domain,

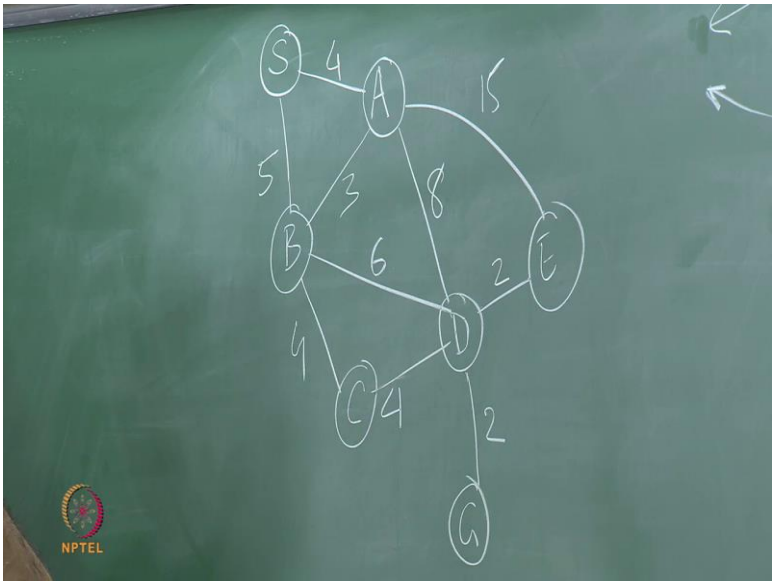
or you are running a courier company, and you have to send people all over the place to collect parcels and deliver parcels, and things like that. The solution that you find has to be executed, and has an associated cost, essentially. Today, we will look, we will shift focus to finding solutions, which have an optimal or least cost solutions, essentially. That is critical in many domains. To take an extreme example; if 10 years from now, you start a company in which you are setting up a colony on Mars, then you can imagine, that the cost of each trip is going to be significant, and you would like to have a solution in which, there are minimum number of trips, essentially. So, maybe, on the way you stop on the Moon, or something like that, or more coming down to Earth. If you are traveling like a traveling salesman, going through a few cities; you may want to optimize the cost of your tour, or if you are using some vehicle, let say, daily in a city, because of your job. Unless you happen to be a politician in the government or a bureaucrat in the government, who get apparently unlimited fuel supplies. You would be worried about the cost of fuel, and you would want to find solutions, which are optimal, essentially.

So, our focus is going to be on finding optimal solutions, now. Of course, one might ask as to how is this different, from the optimization that we have been studying so far because, for example, we said that we want to solve the TSP, when we were looking at example of optimization. So, it is not really different; it is a same process; it is just that I want to be clear on the motivation. Earlier, we went for this process of devising the idea of a heuristic function, and trying to find optimal solutions for that, or trying to find optimum values for the heuristic function. In the process, we, sort of wondered into optimization, which is what we interested in, but we never looked at solutions, which guarantee optimality. We only looked at randomized methods like simulated, unhealing and genetic algorithms, and colony optimization, which are likely to work most of the time, but not necessarily guarantee optimal solutions. So, what we want to do now is to look at methods, deterministic methods, which will guarantee optimal solutions.

So, these are the two different aspects of problem solving; one is, how long do you take to solve the problem. Heuristic function is devised to speed up that process. The second aspect is, how good a solution you find, and that is the aspect that we are going to focus on today. How to look at optimal, how to find optimal solutions? So far, we have not had a notion of cost, in the solution finding process and whenever, we spoke about quality of a solution, we said the number of steps or the length of the solution. So, now, let us

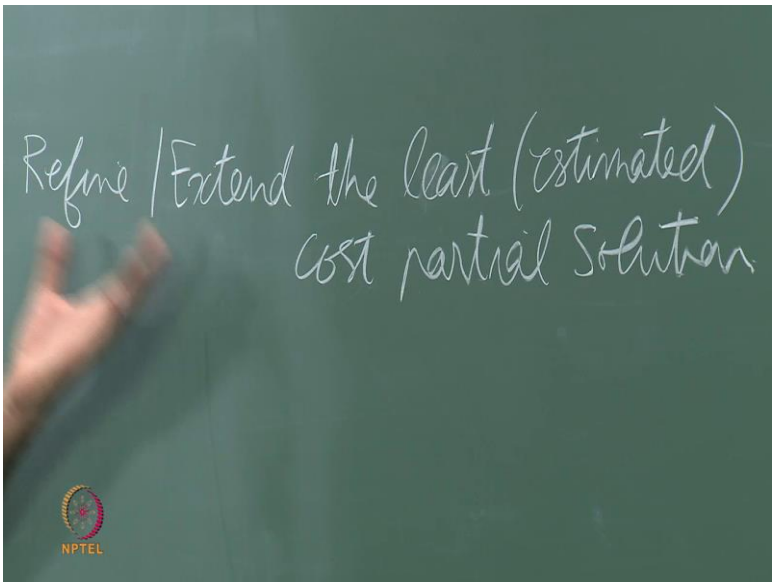
introduce cost, which means we will introduce a weight for every edge in the search space and then, we want to find solutions, which are optimal cost. So, it is like finding the shortest path, and that is an abstraction of all these problems that we are looking at, essentially; finding shortest path in a graph. Let us start with a small example graph, just to illustrate the algorithm set that we are looking at.

(Refer Slide Time: 06:02)



So, let us say, this is a start node and then, you go to some node called A. The cost of going to that node is 4, or you can go to B. The cost of going to b is 5 and the cost of this edge is, let us say, 3. Then, you have, let us say, a few more nodes; C, D, E. Let us give some cost to these. So, let us say, this is 6 and let us say, this is 8 and let us say, this is 4 and let us say, this is also, 4. Let us say, this is an expensive edge, costing 15 and this is 2. Let us say, this is goal node that you want to reach, and the cost of going to goal node from here, is 2. Let us say, there are some more edges that I am not drawing, because we do not want to have an exploding search space. Let us say, this is an example, you want to start from S, and you want to find the path to G, which is of the shortest paths, actually.

(Refer Slide Time: 07:45)



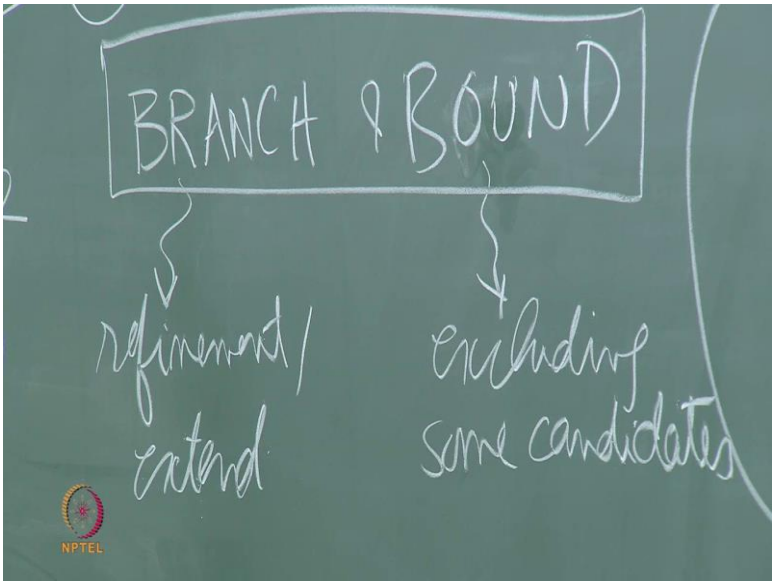
So, the general algorithm that we will follow can be abstracted as follows; that refines or extends the least cost, and I must emphasize least estimated cost, because we do not know the actual cost. So, the main problem solving step, which is basically, very similar to what we were doing earlier. We have a move gen function, which generates successors and you have to choose one of them, and so on. But, we can view this now, a little bit more, abstractly, in the sense that we are working in the space of possible solutions. We do not have all the solutions; we have some of them, partially defined, essentially with every partial solution, we have an estimated cost. When we say estimated cost, we mean them; estimation of the cost of the total solution, if that was to be completely refined. So, we will use a term; refine, to say that if we have a partial solution, and we had a little bit more information about that solution.

For example, if you are solving a TSP, if you have put in five edges, and if you had one more sixth edge, then that is a little bit more refined. So, in the process of you keep refining partial solutions, till you have a complete solution and then, you would stop. The algorithm that we will use is that; extend the least estimated cost partial solution, till such a solution is fully refined because, in the end, we are interested in a solution, complete solution. So, actually, I must clarify what I mean by this, till such, and we will see this is, when the example comes out; that till the least cost solution is fully refined. I, instead of writing that, I have just written; till such as solution. But, you must read this such as; a least cost solution is fully refined. Just imagine this situation where, you have some

partial solutions and some, at least one fully refined solution, or more than one fully refined solution, and you have estimated cost for them. Now, for fully refined solutions, there is no notion of estimate; you actually, know the actual cost, whereas, only for partial solution you have to estimate cost.

Now, if one fully refined solution has a lowest cost, then we will say; we can terminate. That is going to be the basic idea. We will discuss this as to, when is this idea sound; or when this is idea, guarantee to give you the shorter solution. This is the loop that in which we will operate, in the next few classes. It is just that our notion of partial solution may change, as you go along little bit. So, let us look at this from the state space prospective, as we have done earlier. When I look at this algorithm, you must keep in mind, that the similarity with Dijikistra's algorithm, essentially. So, I take it that everybody is familiar with Dijikistra's shorter spark algorithm, which takes a single source and finds shorter spark, to the rest of the graph, essentially. So, we are in some sense going to mimic that algorithm, but our goal is not to find solutions to all the nodes; but we will be doing something similar, essentially.

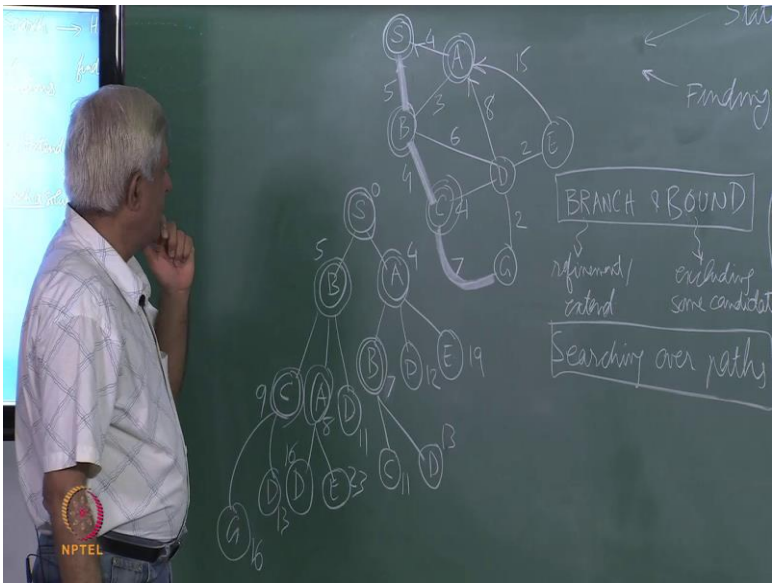
(Refer Slide Time: 11:42)



The algorithm that we are looking at is called Branch and Bound. By branching, we mean this process of refinement, or extension, and by bounding, we mean excluding some solutions. So, remember what I said that at the point, which we terminate, we will not bother about some solutions, which are not fully refined, because we would be able

to exclude them, or we would be, they have gone beyond some bound, that we are working with, and we do not need to refine them, any further. So, this general idea is called Branch and Bound. So, let us first simulate this algorithm on this graph.

(Refer Slide Time: 12:56)



So, you start with a search node and it is very much like the other algorithm that we have seen. We generate its children, in this case; A and B. In this stimulation, we will assume that we are not going to remove duplicates, essentially, because it is possible that we may later find a shorter path, to a given node, essentially. So, we do not want to say if you have seen that node, do not generate it again, essentially. So, you can say this is searching over paths. That instead of saying, that we are visiting a node, for example, D or C, we will say, we are inspecting a path from source to D or source to E, and we are looking at different possible paths, and we will. So, if there is one path S, A, D, then this is a different candidate from another path, which is S, B, D, essentially. So, we will treat them as separate. So far, we are saying, if you have come to D before, do not bother. You have already; so, do not put it into open. That is what you use to say.

Now, the cost of s is 0. The cost of this path is 4, and the cost of this path is 5, so the cost of that edges. The algorithm is simple; refine or extend the least estimated cost partial solution. So, instead of saying that, we are taking the cost of the complete partial solution and estimate of the complete partial solution. We will work with the known cost that we have, which is a cost up to A, essentially. So, as far as we are concerned, we have two

partial solutions here; one goes, say, goes from S to A; the other, say, goes some S to B. This one cost 4 and this one cost 5, and we will say that this is better than S. So, we do not have any sense of direction. So, this is a simple process, we will follow. We say that this has been inspected. In Dijikistra's algorithm, you would do something, very similar.

I will just do the two algorithms in parallel. In Dijikistra's algorithm, you would initialize S to cost 0, and everything else to cost infinity, and maybe, in some versions of the algorithm, some description of the algorithm, you will color them white, and you will color this; or color everything white to start with, which is like putting them on open, or something like that, and pick one node from there. So, there is a stage of relaxation in the rest of the algorithm, which says that once you relax this node, or once you inspect this node S, you relax all edges going out of that, which means this cost originally was infinity, and now, it is reduced to 4. Because, we know that you can get from S to A is 4. So, we revise this cost to 4 and revise this cost to 5; exactly, what we are doing here, except that in the Dijikistra's algorithm, you would do it on the graph itself. Then, you would pick the lowest cost note. So, the Dijikistra's would also pick this and color it black, for example, and then relax these three edges, which I am also saying, that we will generate these three edges. So, one simplifying assumption while drawing, we will make is, that we are not going to go back; we are not going to allow loops, because we know that loops are only going to increase the cost. So, if I go back, go from S to A and A to S, it is not going to help me, in any way. One assumption, that we are working with like, the Dijikistra does, that cost are all positive here. So, pick the least cost node S A. So, let us say, this also, I am coloring it black here. The Dijikistra also would color it black, and there would be a pointer pointing back to this. Then Dijikistra would color this one, black, and try to; when you relax this edge, you find that 4 plus 3, 7. So, it remains like that, whereas, this becomes 8 this, in arrow, which comes here, this becomes 19 and an arrow comes here.

So, there are these costs associated with nodes, which I am not writing there; which I am writing here. From A, you can go to B, or you can go to D, or you can go to E, and the costs are, as we said, 19 for this; D is 12 and B is 5. So, very similar to what we were doing in heuristic search, always pick the lowest cost node, except that when you are doing heuristic search, it used to be lowest heuristic value, note; here, it is the lowest known partial solution cost. So, this is, sorry, this is not 5; this is 7. From S from A to B,

then if you come like this, it is 7, essentially. Then, you pick this B and from B, you can go to C, or you can go to A, or you can go to D. If you go to C, then you have a cost of 9. If you go to A, then you have a cost of 8. If you go to D, then you have a cost of 11. So, this process continues, the rest of the couple of more rounds and then, we will stop. The lowest node now is B. So, we do this. As I said, we will not allow loops. So, we will not allow A or B to B, because A or b is in the path here. So, the only thing you can do is, go to C or to D; C D. So, 7 plus 4; this is 11 and 7 plus 6 is 13. So, you can see, we have found two paths to D; one path costing 11, which goes from S to B to D, which is cost 11; the other is from S to A to B to D, which cost 13.

Let us say, there is another edge, which I have not drawn earlier; forgot to draw earlier. This is costing 6, 7 units, let us say. Now, this is, A is the lowest node that I can expand. From A, I can go to; I cannot go to S; I cannot go to B; I can go to D or to E, and the cost of going to D is 16, here, and cost of going to E is 23, here. I have done away with it. Now, something interesting is happening that I am going to expand C. C and D is interesting, because they are the two nodes, which lead us to the goal, essentially. So, at this point, this node D has become the lowest cost node. So, this is 9, 16, 23, 11, 12, 13, and 19. I expand this C and from C, I can go either to D, along this path, or I can go to G. If I go to D from here, it is going to be 9 plus 4, 13. If I go to G from here, it is going to be 9 plus 7, 16.

So, I have found one path to the goal. So, let me highlight that path. I am going from S to B, B to C and C to G, and that is the path represented by this node G here; S to B, B to C and B to G. So, this is gone into closed. So, should I terminate, or if you go back to what Dijikistra would have done. What have we done? We have colored A, we have colored B, we have colored C, and we have not yet, colored D, or E. Now, if you look at this graph, there is a path going from S to B to D, which is 11 plus 2, 13. Now, that path is better than this path, which is of length 16. If I want my algorithm to find the optimal path, I cannot stop at this stage, which is why, we have this condition, till such a solution is fully refined, and such, I mean, the least cost solution is fully refined. Now, in this case, the least cost solution is this D and this C. So, let us say, without loss of generality, we pick D from here. Once we pick D from here, we will add this; we have already seen C. So, from D, you can go to G to C and to E. So, let us not worry about C and G. The cost from S to B is 5 and then, another 6 and then, 2 and we have this cost of 13. Now,

we have added, in some sense to open, if you want to say; this path S, B, D, G to open, and we also have another path S, B, C, G to open, but none of them is at the head of the queue. Remember, that we will use something like a priority queue to increment this. Before we come to this G, we will exhaust this option of C, we will exhaust this option of D, and we will exhaust; yes, only these two options, you have to exhaust. This one, because this is lower cost, and this is lower cost; once we expand them or refine them, we will get more expensive solutions, and at that point, this G will become the least cost and then, we can terminate.

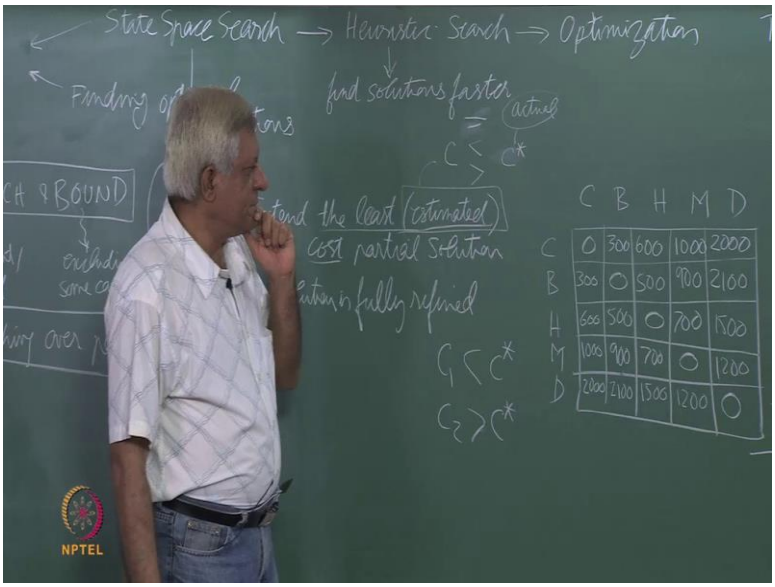
So, this is the idea of Branch and Bound, essentially; that what do we mean by bound here? That the moment, when we expand this node with 13, we are going to bound; we are not going to be interested in this node with 19, or this node with 16, or this node with 16, or this node with 23. Even though, they are not complete solutions, we know that if we are to refine them, there cost would be more than 613 here, and we have found this complete solution of paths cost 13 and the partial solution are of higher cost. Therefore, I can terminate at this stage. So, at the moment, when you pick the goal node or the path to the goal node, you can terminate.

Let us look at another example, which is that traveling salesman problem, which we are so interested in, but before I do that, I want to ask a question, a few. I used a term estimated cost here, and when I say estimated cost, I mean the estimated cost of the full solution; not the partial solution. Here, we are working only with partial solution. So, when you say the cost of this path is 12, we mean that to go from S to A and A to D is 12. It is not telling you how much it would cost through, if you went to the goal, along with this path. Now, I am talking about that; that instead of using this exact known cost of partials solutions, let us work with estimated cost of full solutions.

Let us say, the estimated cost of any solution is C, and the actual cost of the same solution, and by actual cost, I mean, if I want to refine that solution completely, let me use the term; C star. The question that I want to ask is the mechanism that I used for estimation; it can do three things. One is, it can give me a perfect estimate, in which case, this would be; I would use an equality sign C equal to C star, but that is only a wishful hope, that you will find an estimate, which is perfect. So, let us hope that, let us say, that we cannot find an estimate, which is perfect, which leaves us two choices; either greater than or less than. So, we have eliminated that choice, because we have granted that we

will never find such an estimate. Now, if I had a choice between an estimating function, which underestimates a cost, and a choice with a different estimation function, which over estimates the cost. So, let say C 1 and C 2 are such functions. Let us say, C 1 is less than C star, and C 2 is less than, is greater than C star; always greater then C star; which one, should I use? In other words, while devising an estimating function, what property should I take care to satisfy? Should I use C 1 or should I use C 2? How many people here, feel it is C 2, and how many feel, it is C 1? The rest are undecided or not awake. Anyway, just keep this in mind, I will ask this question in a little while again, essentially. Now, let us spend a little bit of time on the travelling salesman problem, again. This time, we are looking for an exact solution; even though it is NP hard problem. We want to look at methods, which will give us exact solutions. Maybe, we cannot solve very big problems, but at least, for the smaller problems, we want that, essentially.

(Refer Slide Time: 27:58)



We look at something called refinement search. By refinement search, I mean the following; that consider the set of all possible tours, that will be the root node of our search three, the set of all possible tours, and then by some operator, I will partition the set into smaller sets, essentially. What is an operator? One operator could be the choosing an edge, for example. Then, in the process, I want to refine so, essentially. I also want to talk about estimated cost. So, one thing that while I draw a small example, you should think about is; given a problem, which means given a set of cities, and the cost of edges between those cities; how can you estimate the cost of a tour? In other

words, can you find a lower bound of a tour, or maybe, if you are interested, can you find a upper bound of a tour; any tour? Or in other words, all tours are going to be greater than some lower bound; can we find such a lower bound? So, let me write a small example. Let us say, we have these five cities; Chennai, Bangalore, Hyderabad, Bombay and Delhi and. So, Chennai, Bangalore, Hyderabad, Mumbai; I should say, otherwise, there is always a danger; and Delhi. So, I need the edge cost between these cities. So, I am just drawing the edge matrix to capture those costs. So, these are 0s; the diagonal elements, because you are already in that city. Let us take some simple values. Let us say, between Chennai and Bangalore, the cost is 300 Kilometers; between Chennai and Hyderabad, it is 600 Kilometers; Chennai and Mumbai; it is, let us say, 1000 Kilometers, and Delhi is 2000 Kilometers. These are not correct figures, but there was a list. So, given these figures, can you find an estimated cost? I am asking you a specific question now; can you find a lower bound on the tour cost; which means that no tour can be cheaper than that cost.

Let us put in some more values. Let us say, Bangalore and Hyderabad, let us say, 500; Bangalore and Mumbai is, let us say, 900; and Delhi is the farthest from everything. So, let us say, 2100 here, and 1500 here, and just some random, a close to random figures, and that leaves me with, between Mumbai and Hyderabad, let us say, that is 700. So, this is my matrix given to me; edge cost, and I want find the solution for the travelling salesman. My refinement search is going to do the following; that my root is going to be S; I will just call it S, and this is a set of all tours. What I am asking you is that for this node S or route S, which consist of all possible tours; what would be a cost that I would want to associate with that, which is the lowest possible cost that I can think of, essentially? Now obviously, you can say 0 is a lower bound, because definitely, every node, every tour will have cost greater than 0, but I am not interested in such a trivial lower bound. You can even say 300 is a lower bound, for example, but I am not interested in that. The reason for that is that, if I am going to do this Branch and Bound, I am interested in excluding candidates from my search space, and I can only exclude candidates, if their estimated cost is higher than my actual cost of some known solutions. To see an example here, if I want to expand this known solution, is cost 13, this estimated cost is 19. Now, this 19 is actually, the actual cost of going from S to A to E, but I can be overly optimistic and say, that is actually, the estimated cost of going to G from this path, essentially. The rest of the edges, I have cost 0; I can be overly optimistic

about that. So, I can treat this actual cost of going to E, as estimated cost of going to G, via this path, essentially, but even, if that were to be the case, I know that the estimated cost is 19, and this actual cost is 13, and this is only going to increase as I refine the solution further.

So, it can never become better than 13. So, I do not really need to refine that. I am going to; that is a bounding, I mean. I am just not looking at that, this thing. So, for such reasons, that I should be able to exclude bad candidates, as quickly as possible, I need estimated costs, which are as high as possible. So, as I said, you can always give me 0 as an estimated cost, or 300 as an estimated cost, but I am not interested in that, because they will not exclude solutions from a search space, essentially. You were saying something.

Student: (( ))

Prof: Yes. But I am not interested in higher bounds, so much. So, let us talk about lower bounds, essentially.

Student: Lower shortest four edges.

Prof: Lower shortest four edges; why not five?

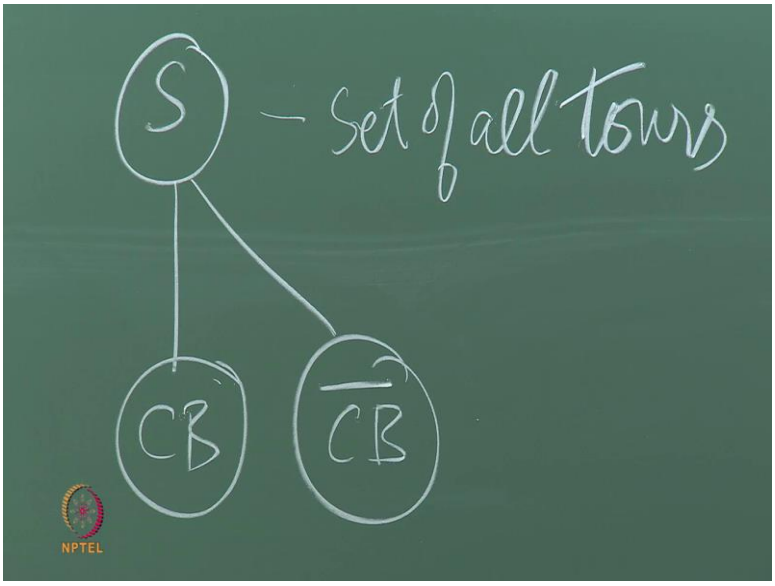
Student: (( ))

Prof: Yes, but that would give me what; 300 plus 500, even assuming that I will look at only the diagonal matrix, I mean, triangular matrix; 300 plus 500 plus 600 plus 700. That would, you know, something like that, essentially.

But, all my desire is to get as high an estimate as possible. So, the general idea is that I want as high an estimate as possible, but it is should be a lower bound, essentially; these two things, remember. So, there are, of course, I know that there are more than one way of doing it. So, we will just use one method here, which is that we will take the lowest two elements from every row, essentially. So, I will take 300 plus 600 from here. I will get 900 from this row. I will take 300 plus 500; I will get 800 from this row. Then, these two, which will give me 1100 from this row; 7 plus 9 is 1600 from this row, and 12 plus 15, which is 2700 from this row. I will sum them up and divide the answer by 2.

What is the rationale behind this; taking two lowest from every row? Yes. In a tour, every city will have two edges; one incoming and one outgoing, if you can distinguish between them. We are sort of being optimistic here, and saying that the two edges are the lowest cost to edges, because they cannot be better than that. So, if the lowest cost two edges are taken from each row and we sum that up, and divide by 2, because we do not want 10 edges; we want only 5 edges; we will get an estimate of the lower bound essentially. So, we follow this similar, this high level algorithm, we follow; refine the cheapest cost partial solution. In this case, we have only one at this moment. So, we will refine that, till the solution is fully refined. Let us say, we use some heuristic and we say, that we will add a cheap edge to the tour, essentially. So, the cheap edge here is, for example; between Chennai and Bangalore.

(Refer Slide Time: 37:49)



So, I have now, I will just draw it a little bit this side, because I want to write things here. I have two successors. This one, I will call C B, and the other, I will call C B bar. By this, I mean, I have portioned the whole set into two subsets; in one subset, which I call C B; the Chennai Bangalore segment will always be present; and in the other subset, it will always be absent, essentially.

Now, I want estimated cost of these two. So, for every node, my search three, I need an estimated cost, exactly like, what is happening here. Now, you can see that, let me write it here; C B here, C B bar; let me write here, and C b here. Now, you can see that C B

will not change, because I am going to include this lowest cost edge, I have already done that in my original estimate. So, whatever was my original estimate, will continue there, essentially. In that case, I can just copy these figures from here, but for C B bar, I cannot do that; why? Because I have excluded the edge C B from that set, essentially. So, I have to find revise. I cannot choose 300, because in this, that node C, this thing; can somebody tell me the value for this? I think so, we can use that. Divided by 2? 3550. So, this cost is 3550, and this cost here, it will also be 3550. This cost is not going to be that, because I cannot use this 300. I have to use the next two edges, which means 600 and 1000, which means, this will go up by 700. So, I will write plus 700 here. I cannot use this here. So, I have to use this. Instead of 900, it becomes 1600, which is plus 700 here. Instead of 800, this becomes 1400, which is plus 600. The rest will not change, because I am not using that edge here, essentially.

So, that is 1300, and when I divide by 2, I will get 650. So, this will be plus 650. So, I have a way of devising edges. What have I done? I have said that; so, this is Chennai, then, this is Bangalore; this is Hyderabad; on some scale, this is Mumbai; this is Delhi; I have said I am going to add this edge, and that is my set C B. So, all towards in which, this segment is there, I will call C B, and all towards in which, this segment is not there, I will call C B prime. Out of these, C B seems to be better. So, I will refine that. Let us say, I follow this heuristic. It is not necessary to follow this heuristic, but let us say, we will follow this heuristic, which means, always pick the cheapest available edge. It is like a DD algorithm, which tries to build cheapest tour, essentially. So, I look around this graph here, and the cheapest tour, I can find next is 500, which is between Hyderabad and Bangalore. So, what am I doing now? I am saying, add this edge, and I am refining this graph. So, one side will be called Hyderabad and Bangalore, and the other side would be called its compliment. So, the way to interpret this node is that, this edge H B prime node is, that all those tours, which contain the Bangalore Chennai segment, but exclude the Hyderabad Bangalore segment, and the way to interpret this node is, all those tours, which contain both the Chennai Bangalore, and the Hyderabad Bangalore segment, essentially.

So, that is what I have drawn here. All tours, which have these two edges is this. Now, let us find an estimate for this. How do I do this? At this point, I should mention, before we do this. This process of computing the estimates is computationally, intensive

process, in the sense, you have to spend some computation time, looking at this matrix and doing something. But there is also that you can do a certain amount of reasoning; something, that we would call constraint propagation, in the process of doing estimates. Let me illustrate that. Let us, first of all, because we have chosen the cheapest edge, then for this nod H B, I am anyway going to include both these. So, it is not going to affect my cost. So, the cost for H B is not going to change. You should just convince yourself that is going to be 3550 also, but this cost for H B prime; that is going to change. So, let us look at the changes, essentially. So, one thing is, of course, that you cannot use this Hyderabad Bangalore link, which is; this is Hyderabad; this is Bangalore. So, this link, I cannot use, just as we did first C B prime, for this nod also, we cannot use this link. So, which means now, this will become 300 plus 900, which is 1200. Originally, I had 800, now it is 1200. So, if you compare with this as a basis, I get plus 400 here.

Likewise, here, I cannot choose this 500. So, it becomes 600 plus 700. Originally, it was 1100; now, it is 1300. So, it is plus 100 here. So, I am just adding the incremental cost. Only these two rows will change; plus 200, but coming back to this, how; see, we have this desire, and we will see this in the next class, that having an accurate estimate helps, or if not in the next class, then, the next class after that; that the higher the estimate, the better for us. Intuitively, it means that the higher my estimate is, the more likely it is that it will get excluded from a search space, essentially; if I find a cheaper solution. Now, if you look at the estimate for this, I have this Chennai Hyderabad section; Chennai Hyderabad is this one; this 600, and this 600. Now, if I do a little bit of reasoning, how constraint propagation, and what is the constraint I am propagating is, that I want to find the complete tour, which means that I cannot have a cycle, which is smaller than length 5, in this example. I cannot have a cycle of length C. So, if I am going to have that set H B, which includes, actually, it means that this estimate is not correct, even though, I wrote it there. Why is it not correct? Because I cannot include in my estimate, this Hyderabad Chennai sector, why, because I have already included C B in that path, and then, in this path. So, I am talking of this node here, and I am saying that even, the estimate of this node will go up. The reason, why that will go up is that having included C B, and have having included H B, I am forced to exclude H C from there, because otherwise, I would have a cycle of length C, which means I cannot use this value for computing this; for this one; for computing the estimate of that node, I cannot use this value, 600. Instead, I will be forced to use the next value, which is 300 plus 1000, which

has become 1300, essentially, which is, of course, more reasoning than what I did so far. What I did so far was, if I am including an edge, I cannot count at, when I am using those bar kind of node, essentially, because they are excluded, sorry, if I am excluding an edge, then I cannot count them.

But now, we see that there is certain propagation, which takes place that if I am including this edge, and if I am including this edge; I cannot count this edge. So, let me just draw a zigzag line to say, that I cannot count that line. Why, because then, I would have a cycle. I can go to more extent to do more reasoning, to get better estimates, and what do I mean by better estimates? Higher estimates, which means, if you cannot include something, then do not include it. That something, in this example, is 600 here, which is a low cost in this row, also in this row, but I cannot use it in my estimates. So, I must use something else, which will give me a higher estimate. Another edge here, that I cannot include, after I have included Bangalore Chennai and Bangalore Hyderabad, there are two more edges that I cannot add, because I want a tour, and the tour has this property; that every city is visited, exactly once, which implies that every city has exactly, two edges incident on it. I already have Bangalore, which has two edges incident on it in this set. So, I cannot have this set. Neither, can I have this set.

So, you can see that problem solving is does not necessarily, one prompt strategy, that you just do search and only search, essentially. Later on, we will see that it is often useful to combine search with reasoning, essentially; some amount of reasoning and some amount of search, and in the process you will try to cut down on the search space, more and more, essentially. What do we gain by excluding this? We get more accurate estimates. For this tour called H B, which is the set of all tours, which include; now, we can describe it more specifically, saying that H B stands for the set of all tours, which include that path here, that C B H, the H B H and which excludes, H C D B and M C, essentially. So, already we have narrowed down our choices and made better estimates.

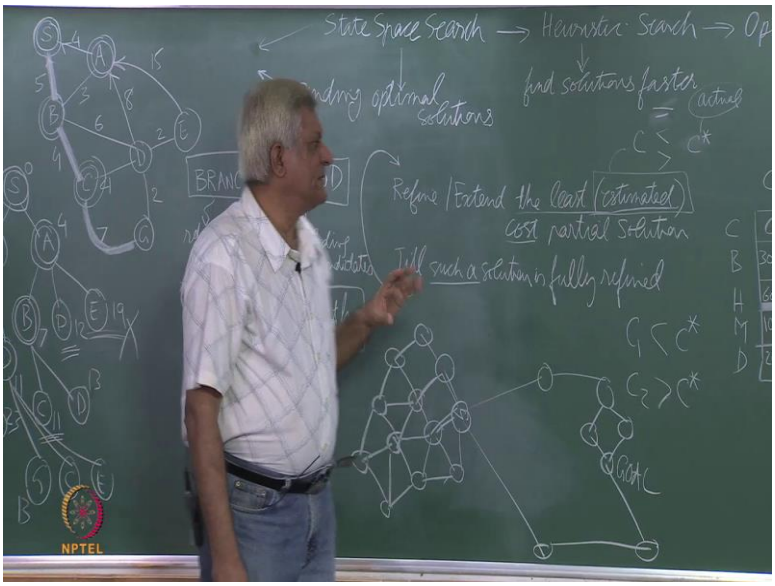
So, this is the same thing that we were doing here; refining the least cost partial solution. We started with 3550 and then, we got two solutions; one was 3550, and the other one was a bit more than that. We refine this; we get these two solutions. We have not computed the actual cost for this, but once we do that, we will refine them. At some point, we will get a complete solution. For example, in this problem, if you want to add one more edge, let us say, this edge; Hyderabad Mumbai. Then, actually, you have

solved the problem, completely, because then, after that, you do not have any more choices left. If you are going to include these three edges, then you have to visit Delhi once. Once you have to visit Delhi from Mumbai, and once from Chennai. So, the rest will be four, essentially.

So, after we have done the search, we have found a complete tour. After we added this search; below H B, if we add M H, and if this node M H, which includes these three edges, happens to be the lowest cost node in my search space. Then, I can terminate. That is the shortest path. So, you must convince yourself, that this termination criterion is sound. By that, I mean that it will guarantee an optimal solution, essentially. One of the reasons behind that is, that I have said that we are going to use lower bounding estimates for estimating the cost of a solution.

You can see that there is some similarity of Branch and Bound with best first search, essentially. You can view best first search as doing Branch and Bound, with the condition that all edge cost are equal, essentially. That will force the; if all edge cost are equal, then you will just go down level by level, because the first level; the cost is 1, the second level; the cost is 2, third level; the cost is 3, and so on and so forth. So, depth first search is a special case of Branch and Bound where, all the edge cost are equal. When the edge cost are not equal, then Branch and Bound is a specific specialization of best first search, sorry, it is a generalization of Depth first search. So, you have to convince yourself that this will give you an optimal solution, but it does not have this. Of course, I said we are going to use a heuristic that uses a minimum cost solution, but it does not have a sense of direction, essentially.

(Refer Slide Time: 57:22)



So, let me illustrate that with a very small example, that if you are doing this city map kind of a thing, and if this is your start node and this is the map that you are looking at, and this, let us say, this is to scale, which means a length of the edge, that I am drawing is actually, the length of the edge. So, let us say, you have some such place. You add this start node and then, of course, there is one node here, let us say and then, some nodes here, and let us say, this happens to be a goal node. What will Branch and Bound do? What is the behavior that Branch and Bound will exhibit? It will explore all this part of the graph, because that is what its mandate is; extend the cheapest partial solution. All the cheapest solutions are on this part of the graph. So, it will explore this; it will explore this; it will explore this; and all possible combinations, which of course, has an intelligent view or you would see is not a very bright thing to do, essentially.

It will guarantee eventually, it will find me the optimal solution. For example, if there is another path from here, which goes like this, which is longer; it will find me the shortest path, but after doing a lot of unnecessary and useless search, in this part of the map, essentially. It does not have a sense of direction. We are focused on this part, finding optimal solutions, and in the process, we have forgotten about this part; finding solutions faster. So, in the next class, we will combine these two together. We will see that how we can combine Branch and Bound with, this was best first, if you remember; best first search.

We will introduce a heuristic function back again, and try to use this diastase frame of working on a graph to look at an algorithm, which is a very well known algorithm called A star algorithm. So, I will stop here and we will take this A star algorithm, up in the next class.