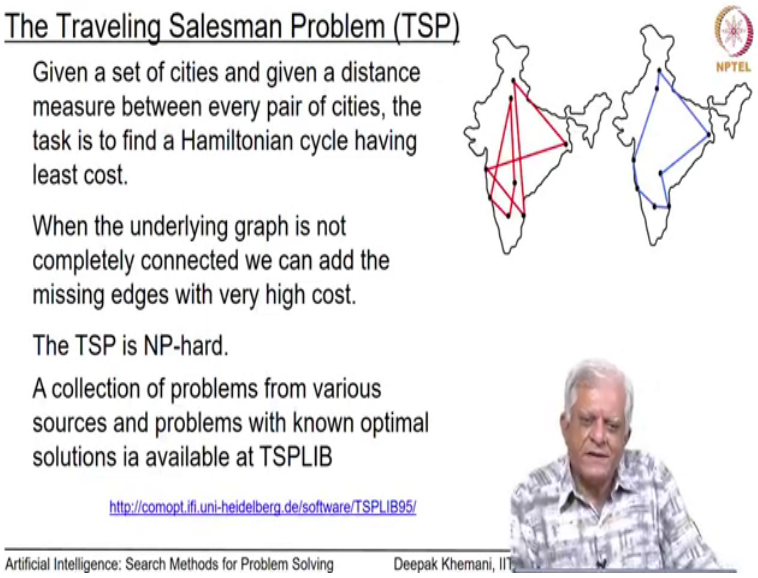
Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science & Engineering
Indian Institute of Technology, Madras

Lecture – 26
The Traveling Salesman Problem

(Refer Slide Time: 00:15)



So, let us look at another problem that we keep talking about which is a Travelling Salesman Problem. The problem of travelling salesman which many of you should be familiar with, is says that given a set up cities and given a distance measure between every pair of cities, the task is to find a Hamiltonian cycle having the least cost. So, a Hamiltonian cycle is that you start from some city and visit every other city exactly once, and then you come back to the original city.

If the underlying graph is not a completely connected graph, so for example, if you were to look at the road network; road networks are not connected completely, they are only connected to the neighbouring cities and so on. So, for our algorithm we can convert that into a completely graph by adding edges with very high cost which would be typically not reserved.

The thing about TSP is that, it is a problem which is much harder than the sat problem. In fact, it is in a class which is called as NP-hard. As a very simple calculation you can see that if you have n cities to go to you can choose the first one in n ways, the second one in n minus 1 ways, the third one in n minus 2 ways and so on, so essentially the number of possible ways that you could traverse this n cities is factorial n.

And if you look at the factorial function, you can see that it goes much much faster than the than the exponential function which is 2 raise to n for sat problem. And the TSP problem is in fact much harder than the sat problem. For that reason many people considered it to be the Holy Grail of the computer science to solve the TSP problem.

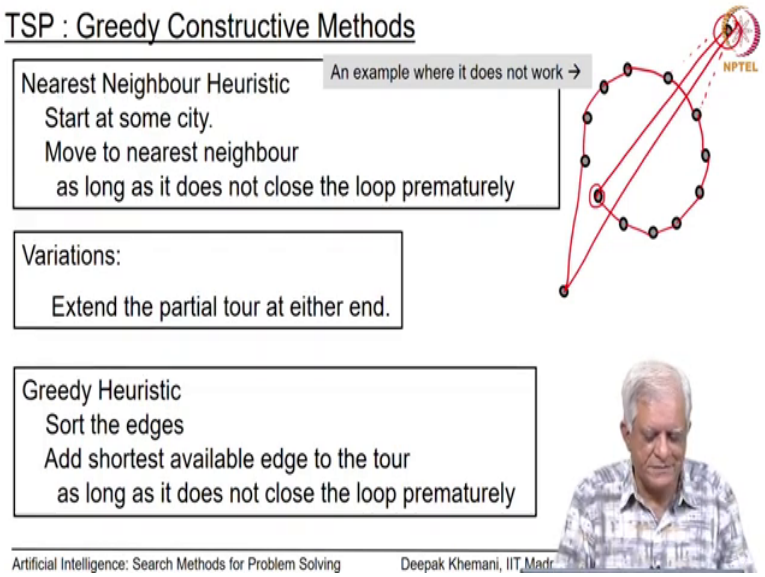
Now, if you are interested in the TSP problem, I would strongly recommend that you go to this library called TSPLIB which is maintained by the University of Heidelberg. And they not only have a nice history of the TSP problem, they have a study of different kinds of distance functions.

So, for example, if you were to do the TSP on a globe like our earth, then the distance function is different from the distance on a plane which is a Euclidean distance, and how does that effect the search and so on. And not only that they have a collection of problems which are completely solved, we said that the complexity can be factorial in nature. So, they have actually taken the trouble to take some problems run them for months and months on end on their machines and giving you the optimal solution. So, you could always use it for comparing an algorithm that you are doing to using to solve TSP.

So, the figure on the right hand show some cities in India which travelling sales person would have to visit. So, I think the cities are Chennai, and Bangalore, and Goa, and Hyderabad, and Bombay, Kolkata, Delhi, and Chandigarah essentially.

So, the tour on the left hand side which is shown in red as you can see it does not look like very efficient or optimal tour, whereas the tour on the right hand side which is shown in blue looks like that it might be the least cost or close to the least cost tour. And the task is to find such tours. Even if we cannot find the optimal tours we would be happy if we find as low cost tours as possible.

(Refer Slide Time: 03:52)



Now, before we move onto the perturbation methods that we are interested in let us quickly look at the Greedy constructive methods which have been popularly used for solving the TSP. And the most institutive and the simplest is using the nearest neighbour heuristic which simply

says that start at some city and move to the nearest neighbour from there; and from there move to the nearest neighbour and keep constructing the tour. Make sure that you do not construct shorter loop then the n cities that you want to cover essentially.

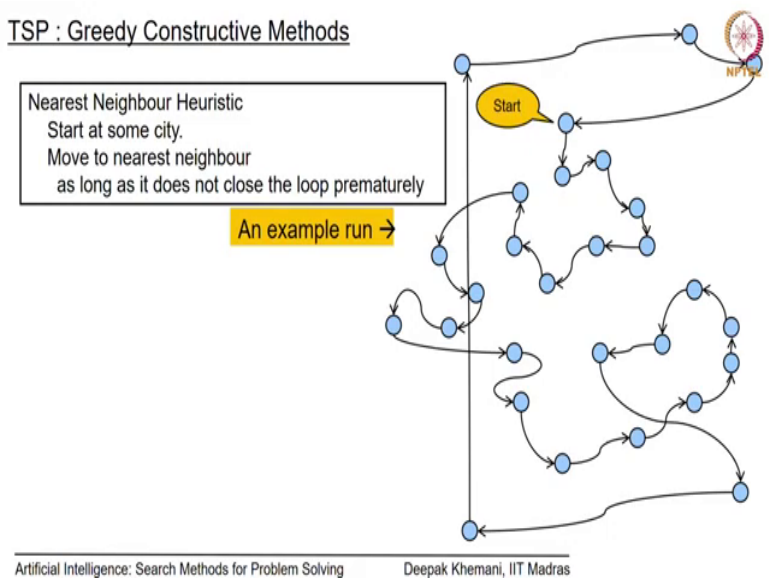
Now, if you imagine this algorithm running on this set of cities, you can see that there are cases as exemplified here in this diagram where this algorithm will not give you the optimal solution. So, for example, if you were to start at this city, then you will end up going here, then here, then here, then here, because you are moving to the nearest neighbour.

And of course, we cannot close this gap from here. So, then you will go to this one; and from here you will go this one; and from here you will come to this one. Now, you can see that this is not clearly the optimal tour, because to go to this city it would have been much better if you had connected in to these two cities, but the nearest neighbour algorithm cannot see that. We will see another example very shortly.

A variation of that would be why extend the tour only in one direction. You can extend the partial tours at either end its possible that might give you a better solution. Another very popular algorithm is called the Greedy Heuristic. And for those of you have studied minimum spanning tree, you will see that this is reasonably close to the Kruskal's algorithm for doing that.

And what it says is that sort the edges, and add the shortest available edge to the tour as long as it does not close the loop prematurely essentially. So, as you can see it also make sense because you want tours to have shortest edges, and this algorithm is saying that you know take the short edges first, and then keep adding them essentially.

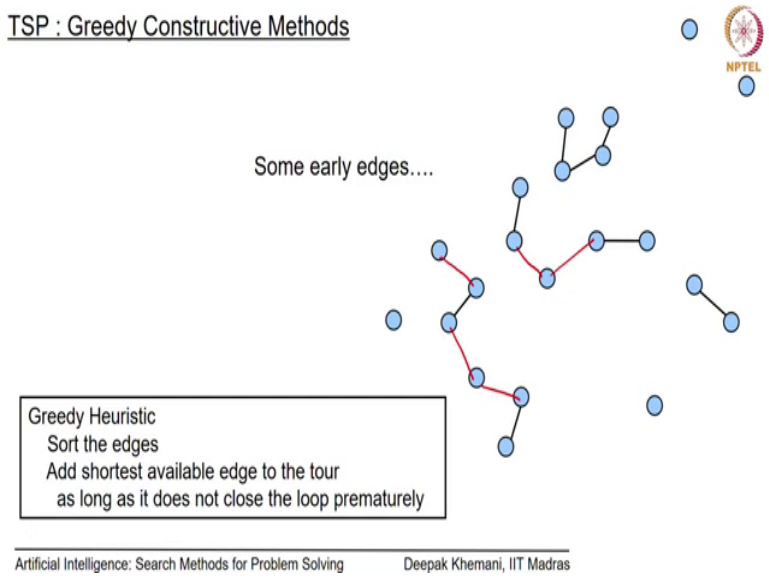
(Refer Slide Time: 06:10)



So, here is an example of the nearest neighbour heuristic with a slightly bigger example. And supposing this was to be your start state, and you kept moving to the nearest neighbour. Then reasonably assuming that you know the distance function is reflected on the location of the these cities on this map here which means that the distance are proportional to the distances on the map. Then you would get the your which looks something like this. And this is how the algorithm will proceed moving on to the nearest neighbour at every stage.

So, we are assuming that the actual distance of every cost of each edge is proportional to the Euclidean distance as shown on this diagram. And you can see that the nearest neighbour algorithm typically performs well in the beginning; but towards the end when it has exhausted some of those earlier parts, it ends up picking some very long edges.

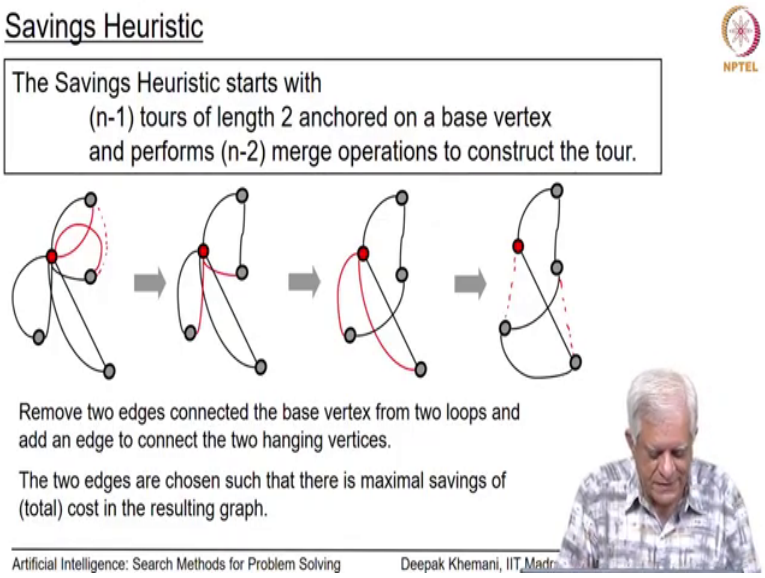
(Refer Slide Time: 07:21)



If you were to do the TSP, the if you have to do the greedy algorithm, then what this diagram is showing is some of the early edges that would get added to the tour. And you can see that from then onwards it might try to add more. So, the remember that the heuristic is add the shortest edge that you can add to the tour.

So, for example, the next edge that you might add might be this one and then perhaps next the next stage could be this one, and may be the next one could be this one, but that you cannot add because this node would have three edges. So, you are supposed to take that back perhaps you could add this one, and may be this one and may be this one and so on. As you can see that we are constructing a reasonably good tour, but it may not be the best possible tour that you would imagine.

(Refer Slide Time: 08:30)



Another heuristic, so these are all heuristic algorithm, these are all greedy algorithms. In the sense they do not take the they do not take exponential amount of time, they do not take factorial amount of time which is what a brute force algorithm would do, and they try to solve it in some polynomial time and that is why they also called the heuristics, some people call them as meta heuristics, but they try to find a solution as quickly as possible.

So, the savings heuristic says that you first construct n minus 1 tours of length 2 anchored on some base vertex. So, choose one base vertex and construct n minus 1 tours. And then you perform n minus 2 merge operations to construct the tour; in each merge operation you will take two small tours and combined them into a larger tour.

So, the operation that you would do is remove two edges connected to the base vertex from some two loops and add an edge to connect the two hanging vertices. Which two edges? The

two edges are chosen such that there are maximal savings in cost in the resulting graph essentially.

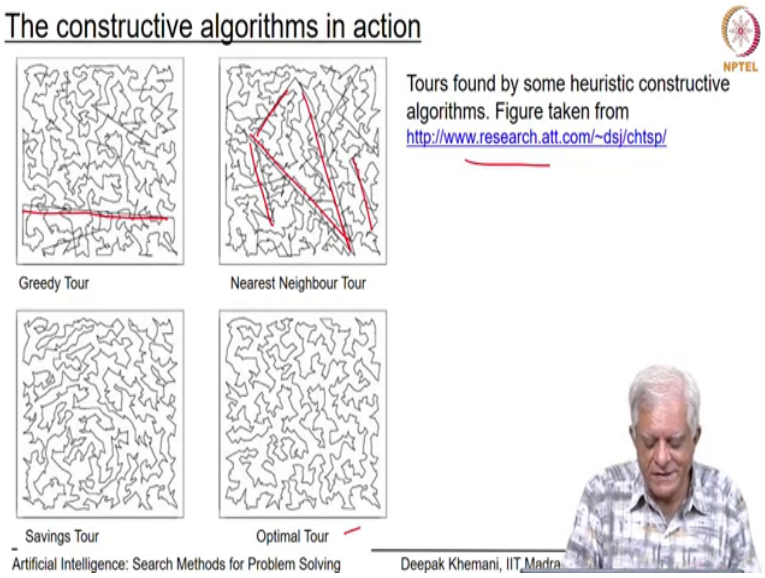
So, here is the small illustration of how this works. So, you start off by constructing n minus 1 tours in this example, there are five cities. And therefore, you can see that there are 4 tours which have been constructed. And the red node is a base vertex and then in the first step we removed those two red edges that we have seen marked in this, and connect the resulting hanging edges by joining them here.

So, then you would get a tour which looks like this. And next time we remove the two red edges that are marked again on this side, and we will get a tour like this. And then again we remove the two red edges, again remember that we are taking two red edges which are going out of the base vertex, the base vertex was connected to n minus 1 cities to start with.

And in the end, it should be connected only to two cities because that is the property of a tour for the travelling salesman problem that every city should be connected exactly to two cities once that you visit from and then once that you move to essentially.

So, we are moving these two red edges here and now we have a final tour. As you can see every cities is connected to two cities and we have a tour. It does not look like the best tour, probably the best tour would be like this. Assuming that the heuristic function is Euclidean in nature, the distance function is Euclidean in nature. But it is a reasonably good tour; it is not too bad essentially.

(Refer Slide Time: 11:30)



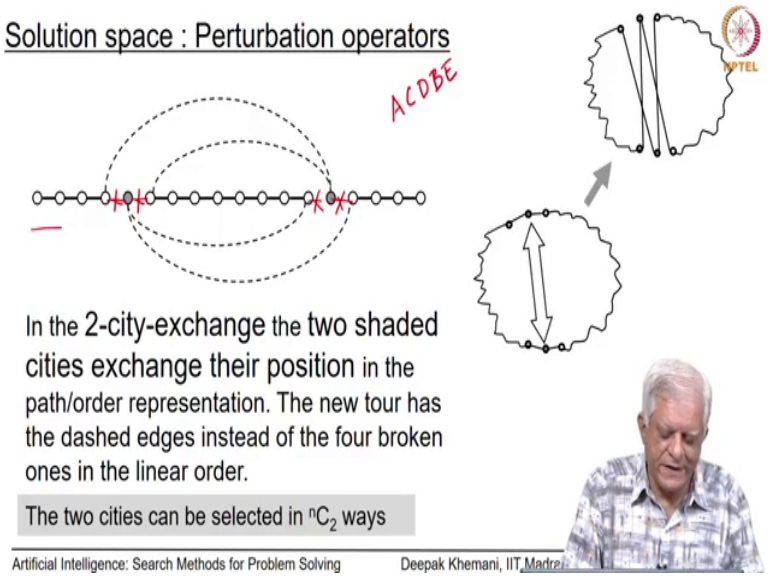
So, here is a illustration of a some experimentation that has been done taken from this site here. It shows how what are the nature of the tours that these different algorithms that we have just looked at do that. You can see that the nearest neighbour tour is probably the worst; it has got this many long edges which it has ended up adding. And as we saw this typically happens towards the end of the (Refer Time: 12:05).

When it is exhausted the shorter neighbours then it has to go to neighbour far away. The greedy, the greedy tour, remember the greedy tour says that add short edges first, but towards the end it ends up at again adding some very long edges essentially.

The savings tour seems must better. If you look at it visually ah at least there are no criss crossing edges this is what you would expect in a good tour. So, if there are two edges which are crossing each other, it probably means that you have a bad tour essentially. The savings

tour does not having any criss crossing edges and it looks reasonably good and on the bottom right we see the optimal tour and you can see that has a similar look as a savings edges, but is in fact, the optimal tour.

(Refer Slide Time: 12:57)



So, let us now move to the solution space problem of a perturbation of a tour. So, if you have this tour which is given here and very often we represent a tour as a permutation of cities. So, for example, if we say that ACDBE is a tour, then this representation is called the path representation. And what it says is that you start at the city A, go to the city C, go to the city D from there, then to B, then to the E, and implicitly you come back to A.

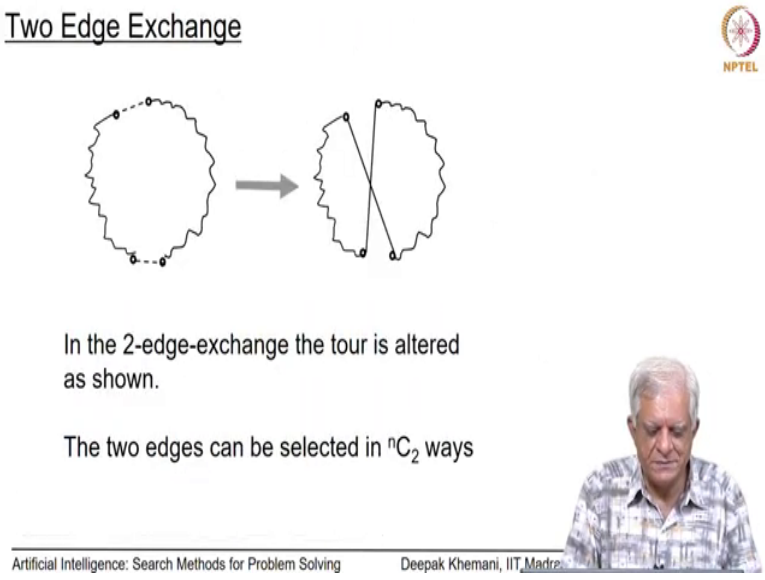
So, if this were to be a path representation, then you can see that we have a list of cities. So, one perturbation operator can be two city exchange. What it says is that exchange the location of two cities in the path representation which basically means that you removed the edges that

were originally connected to the two cities which would have been like this, this here, and this here, and this here and this here. So, you remove this you remove this you remove this and remove this and then connect the edges again to give you a new tour.

So, if this was the tour for example, and these are two cities that you are exchanging, then the resulting tour would look something like this essentially. What you see on the left hand side is something like a path representation, what you see on the right hand side is something like a tour arranged in a visual visually appealing format essentially.

So, how many neighbours will this perturbation operator give you can choose two cities in n C 2 ways, and therefore, if you started with 5 cities, then you would have 10 operators 10 neighbours and so on.

(Refer Slide Time: 15:03)



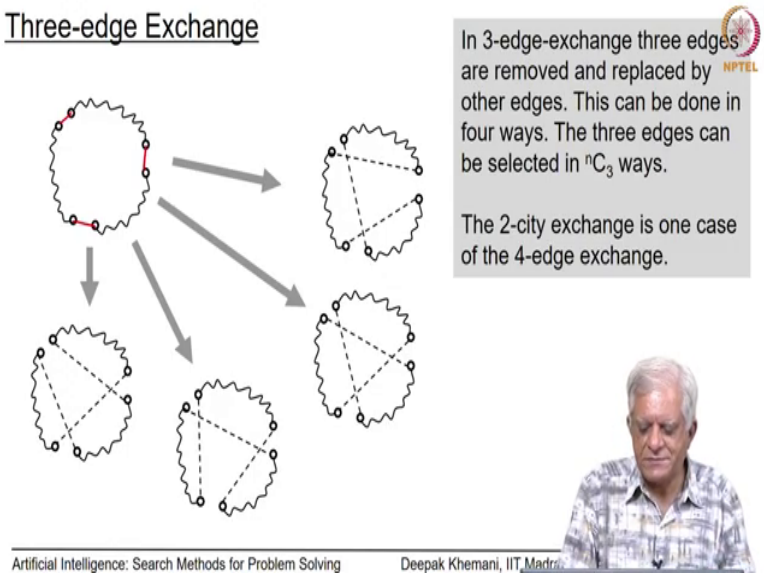
Very often one prefers to use an edge exchange instead of a city exchange, because what is demonstrated here is an edge exchange. So, the two edges on the left hand side which are shown on the dotted lines are removed, and two new edges are added. But in practise the move from right to left would have been preferable.

And since in edge exchange in two edge exchange, what are you doing you are saying that remove two edges from your tour and reconnect the four cities which were connecting those two edges, it could be three also in some cases with new edges.

So, since you are allow to remove some two edges one heuristic that you could follow here is that remove two large edges. So, you construct a tour for example, using a greedy algorithm or the nearest neighbour algorithm or the savings algorithm, and then you would do this two edge exchange on the constructed tour.

And what you do is you remove two large edges and replace them with two hopefully shorter edges, and hopefully that will be done in a better way. But if you look at the space generated by this algorithm, it also requires it generates n C 2 neighbours because you can remove two edges remember that with n cities there are n edges in a tour.

(Refer Slide Time: 16:22)



Another possibility is to do 3-edge exchange in three edge exchange you remove 3-edges, so the edges that are we have removed I am drawing here with red. So, if you remove these 3-edges, you can fill them in as you can see in 4-different ways. And you can remove 3-edges in n C 3 ways, so that gives a much dense neighbourhood function.

So, you can see that even in the TSP, you could have use an algorithm like variable neighbourhood descent. And typically what you might do is you might start off with some greedy algorithm which will give you an initial tour and then apply one of these perturbation operators to try and get better tour essentially.

In the next few classes, you will also see stochastic methods for doing that. So, you could follow this of its stochastic methods which still care to look better tours. Now, if you go back

to the 2-city exchange that we started with, you can see that essentially these 2-city exchange is removing 4-edges and putting back 4-edges.

So, it is just a special case of the 4-edge exchange if you remove 4-edges you could remove them in n C 4 ways and then you could put them back in many ways which I will leave as small exercise for you to figure out. But the 2-city exchange is just one of those neighbours that you can generate by a 4-edge exchange.

(Refer Slide Time: 17:58)

So, with this we end the deterministic methods. There is still one more deterministic method called tabu search that is still left for us to pursue we will look at that in the next class. And then we will move to stochastic methods. And our goal all this while is going to be how to look at algorithms which escape from local maxima or local minima or in general from local optima.