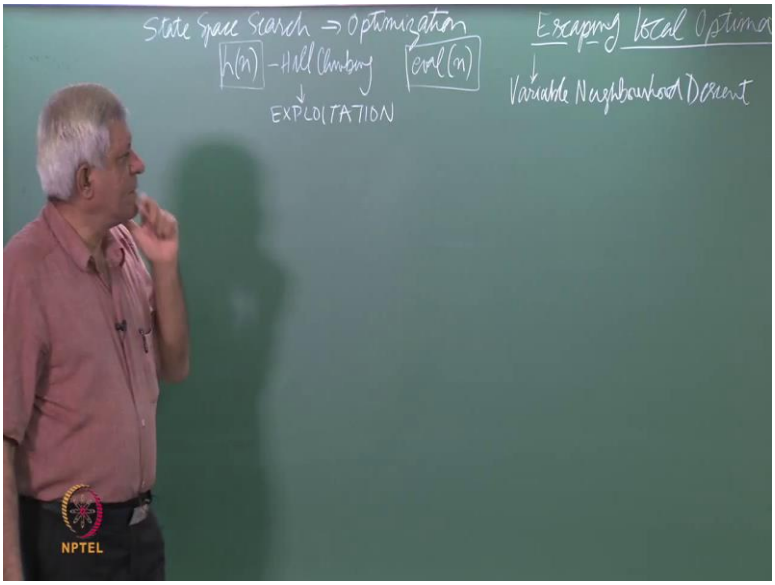
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 12
TSP Greedy Methods

So, you were looking at optimization. And remember that we came to optimization from state space search.

(Refer Slide Time: 00:20)



And the way with this was that, we looked at the heuristic function. That was used in best for search. And said that this, defines at again over which we use an algorithm called hill climbing, which is a local search algorithm. As the first, a global algorithm which best first search was, which means at the algorithm looks only at it is neighborhood. Immediate neighborhood in the state, or in the solutions space and moves to one of them especially.

And then, we saw that local search algorithms can get stuck in local maxima, or local minimize is the case may be. And so the theme for our said this movement, is escaping from those minima. That, how can we improve upon hill climbing. So, what does hill

climbing do? Hill climbing does, what we will call as exploitation. And exploitation of the heuristic function essentially, that it basically follows the heuristic function.

It looks in around it is neighborhood. And wherever the heuristic function is getting, a maximum increase. Or in other words, in the direction of the steepest gradient, it makes one move. And then, repeats it process still it cannot find the better neighbor. And in escaping local maxima, we said that one of the first algorithm that we looked at; or in fact, the only algorithm that we have seen is variable neighborhood descent.

And what this algorithm did was that, it tried out a variety of neighborhood functions. So, for us the neighborhood functions is given by the MoveGen function. And we also said that, we are moving from state space to solution space search, which means that we are perturbing candidate solutions to generate new solutions. And we saw the example with said, that you can flip 1 bit or you can flip 2 bits or up to 2 bits or 3 bits and so on.

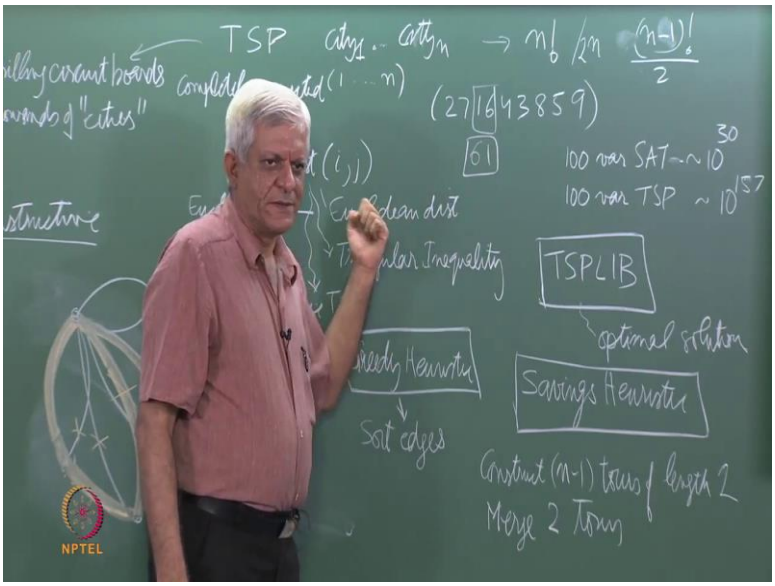
You can generate a variety of neighborhood functions. And what neighbor, variable neighborhood descent is that when it get stuck, it does a series of hill climbing's. And when it get stuck, at a local maxima or at a maxima it does not know, that is local or not. It increases the density of the neighborhood function and tries hill climbing, all over again essentially. So, we want to move, we want to look at different other approaches which will help us get around local maxima.

So, in the process when we are talking of optimization, we will use the notion evaluation. So, this is just particular to the optimization community. So, but you must keep in mind that, we are talking about the same function; whether it is h of n or evolve of n. It is basically a value that you get, for a given candidate. And the task is to maximize, to find the candidate which has the maximum, this value. So, when heuristic functions, if you are thinking of heuristic as a distance, then you want to find them.

State with the smallest distance, which is the goal state or if you have pause it, like within the block problem. That some heuristic function, which has the maximum value at the goal state, then you are maximizing essentially. So, instead of calling it heuristic function, we call it evolve function. But, the process is still the same that, we all want to

find the note with the maximum value or the minimum value, the case may be. So, today let me introduce this. One of the most talked about problems in computer science, which is the tabling salesman problem.

(Refer Slide Time: 05:17)



I am sure, you are all familiar with it. So, I will just write the acrolein here. TSP stands for Travelling Salesman Problem. And it is one of the simplest problems to state and one of the hardest problems to solve this easily. So, a travelling salesman problem is basically, the motivation is set to be to help a travelling salesman, who has to visit many cities. Let us in one day or something like that. And come to his home city. And do the whole thing with some minimum cost, in some manner essentially.

So, I suppose everyone is familiar with the TSP problem. We will say that, we have city 1 to city n, n cities which we will also denote by 1 to n. Just has a short hand. Easier to refer to is essentially. So, there is n cities, 1 to n. And you have to go from one place to another essentially. one city to another, without visiting the same cities twice. Now, in practice off course, in some problems it may be necessary for you to visit the same city twice essentially.

So, for example, if there is a small Ireland near the coastline, to go you go to that Ireland

from one city. Let us say Chennai for example. And you come back to Chennai and then, go somewhere else, essentially. But, we will ignore all those problems. And we will assume that, the classical problem of visiting each city exactly once. Essentially is what we want to solve. Now, in practice a city network may not be completely connected.

In the sense that, if it is a road network, you may have roads from some cities to other cities and so on and so forth. But, not for example a direct road from, let say here to Nagpur or something like that. But, we have indirect roads, in the sense. That may be, you go from here to Vijayawada. And from Vijayawada, you go Nagpur or something like that. But, in general it has been found that, it is easier to solve TSP when the graph, underline graph is completely connected.

So, we will assume that, this is completely connected. And we can always convert a non-connected graph to a completely connected graph. I adding new edges. And making sure those edges have very high weight. So, that they never figure in the solution actually. But, the solving process becomes easier. Now, TSP occurs in many practical problem. So, for example drilling circuit boards, if you want to manufacture circuit boards then, you have to drill many holes on that board.

And if you think of each hole as a city then, you have to visit all these locations keep drilling holes, essentially. So, we have something like a TSP to solve them. And such problems have something like thousands of. So, it is not uncommon to have a problem with eight thousand holes to be drilled and think like that. And then, you can see that the complexity of the problems grows very quickly. So, what is the number of tours that we have. For, if you number of n cities, how many tours can we have for the n cities.

So, we can have n factorial tours. And one of the notations that we will use for tour is, something like this. So, for example, if you have nine cities then, a tour could be something like 2 7 1 6 4 3 8 5 9. So, list city names or in this case city indexes or city numbers, gives you a candidate tour, essentially. So, this is a tour says that, you started cities 2 then, go to city 7, then go to city 1 then, go to city 6 and so on essentially.

We can represent a tour in this session. And we can see that, we can honest choose the

first number is n ways, the second one in and so on. And that is, how we get that number factorial n. But, many of these tours are duplicates of each other. In particular, if I rotate this number, if you think of this is a number, I start with 7 and 2 comes at this end. Then, it is a same tour essentially. Because, remember that in the travelling salesman problem, you have to come back to the same cities.

So, after 9 we are going to come back with 2. So, I could started with 7 and return to here. I could start with 1 and return 27 here and that would be the same essentially. So, you should device this way n, because there are these n rotations that are possible. And in the addition, if I write it in the reverse order 9 5 8 3 4 6 1 7 2, then also we will assume it is the same tour. So, we will assume that the cost of going from city a to b is the same as the cost of going from city b to a.

So, you have to divide further by 2. So, you have divide by 2 n. So, what we get is, n minus 1 factorial divided by 2. So, this is a number of distinct tours, which means say no 2 tours are identical. Of course, in factors it may not be easy to recognize the tours, the distinct tours. So, we can say that we have about n factorial. The size of the space is end factorial, essentially. How bad is the factorial function?

So, remember that we has said when we have talking about sat, we had said that a 100 variable SAT problem has to raised to 100, the size of the space is 2. So, 100 variable SAT has 2 raised n, which we said was about 10 raised to 30. And we have in sought of talking about this large numbers, and say how big these numbers. We try to imagine how big these numbers, really or essentially. So, you must whenever you get time, look at this book call mathematical themes.

A mathematical theme, which is by I keep talking about quite often. And one of the articles in that book is or one of the chapters in this book is, about how we cannot distinguish between large numbers. So, we cannot if I say 25 billion or if I say 25 million, they basically appears same to you essentially. I mean in the sense, you do not have a sense for large number. Essentially, if I say that, it takes 10 is to 27 let say seconds. And if I say it takes 10 to 29 seconds, it sounds almost the same to us.

But in fact, is of course, 10 is to 29 is 100 times 10 is to 27 and we sort of 10 to lose sense, of in essentially. So, we have seen the 10 raised to 30, extremely large numbers. And if you have to explore all of them, it would be billions of years essentially. But, 100 variable TSP is about 10 raised to 157. So, TSP is much, the factorial function is much much much verse in terms of how fast it grows, has compare to the explanation function.

So, the SAT problem is exponentially hard. In fact, it known to be n p completes. It for the first problem, for that was shown to be n p complete. TSP is verse an exponentially, it is factorial and factorial tends to go much much faster. And we can see that, the 100 variable TSP has about 10 is to 157 possible different, possible to us. And that is the number, that we cannot even began to imagine essentially. So, as something to compare with, we can say how many fundamental particles.

Let say as some level electrons for example, are there in this entire universe. Does anyone have an idea? How many particles are there in the universe, at some level of detail, essentially?

Student: (Refer Time: 15:24)

Yes. It is in similar, that somewhere 10 raise to 75, 10 raise to 80 depending on what level you looking at. So, the total number of particles in the universe is about 10 raise to 80. See each of them was a super computer, examining billions of states per second. We would still need billions and billions of years. So, these are the very large numbers that, you have to tackle. And that is why, you cannot even hope to solve this completely.

You cannot even hope to find them optimal solutions for something like, TSP. And in that sense, many people call TSP has the holy grail of computer science that, how to find good algorithms to solve TSP is a great motivator, essentially. Now, one thing that we have talking about so what is the problem? The problem is that we have to find the tours, the cost of the tour is minimal, essentially. That is the problem essentially.

Now, the cost of course depends on the individual distances between. So, distance city distance from i to j, what kind of a function are we using for distance function, much

depends upon that as well. If we assume that, the distance function is arbitrary then, of course the problem is completely hard to solve essentially. Then, we have to go back this thing. But, sometimes the problems are not so arbitrary. In the sense, distances are not necessarily arbitrarily.

So, one of course example is, simple example is the Euclidean distance. So, if the distance is the Euclidean distance. So, for example, if you have manufacturing a circuit board then, you can think of the distances of the Euclidean distance. And then, at least indubitably you can imagine that, you will be able to find an optimal solution. So, supposing it was the great, and you have to drill many holes in the grid.

Then, you can at least hope to say that, I will find a good solution, if not necessarily an optimal solutions. Now, in terms of that for, so Euclidean distance we will call it as Euclidean TSP. If this is the Euclidean distance, as the distance measure it turns out. That if the problem is the Euclidean TSP then, you can solve it in polynomial time, not optimally but, the community calls approximate solutions.

So, we can find approximate solutions, in polynomial time and you can specify it, what degree it is approximate. So, it is something like 1 plus 1, 1 upon c or something like that times optimal cost. So, in polynomial time you can find very good solutions and you can put a bound on, how bad they can be essentially apart. From Euclidean distance, we can say that the distance should satisfy, what we call as a triangular inequality which is the weaker condition.

Then, Euclidean distance also satisfies triangle inequality. But, what triangle inequality says that, if you want to imagine a triangle between three cities. Then, the distance between of the length of one side is smaller than the sum of the lengths of the other two sides. So, that is the, this is known as the triangular inequality. And if the distance function satisfies the triangular inequality then, also it is relatively easier to solve.

Easier to solve meaning, you can find good solutions faster. You cannot solve it optimally, essentially. Then, we have something call the geographic TSP. So, imagine that you are one of those globe floating executives, who flies from here to Delhi and

from Delhi to Tehran and Tehran to Budapest and so on and so far. Going all over the world then, your domain is a spherical domain. The earth is the sphere as we all know. Now, at least and distance is do not necessarily, we are not necessarily we cannot talk of Euclidean distance.

So, we have to talk about distance on the curved surface. So, that is the slightly different problem. People have try to solve these kind of problems. Now, interestingly you must look up for this website, it is called TSP LIB. So, I do not remember the address but, you must look up for TSP LIB website. It is a website, maintained in one of the German universities. And it is a collection of very interesting TSP problems.

So, you know there is a problem of all European cities. For example, and some circuit board problems and things like that. And more interestingly, it has got problems in which somebody has taken the travel to find the optimal solution. If you have a problem of 15000 cities for with the optimal solution is known then, you can write some algorithm that will be used to calculate them when it exploring.

And compare your algorithm with one of those with the actual optimal solution. So, this has the optimal solutions. So, there are examples big examples, which their optimal solutions given. And so you could use that as a benchmark, to see how good your algorithm is essentially. And we probably give you one exercise, along those lines essentially. Some of those optimal solutions, see the only way we can guarantee an optimal solution and we will study optimal solutions, sub it later.

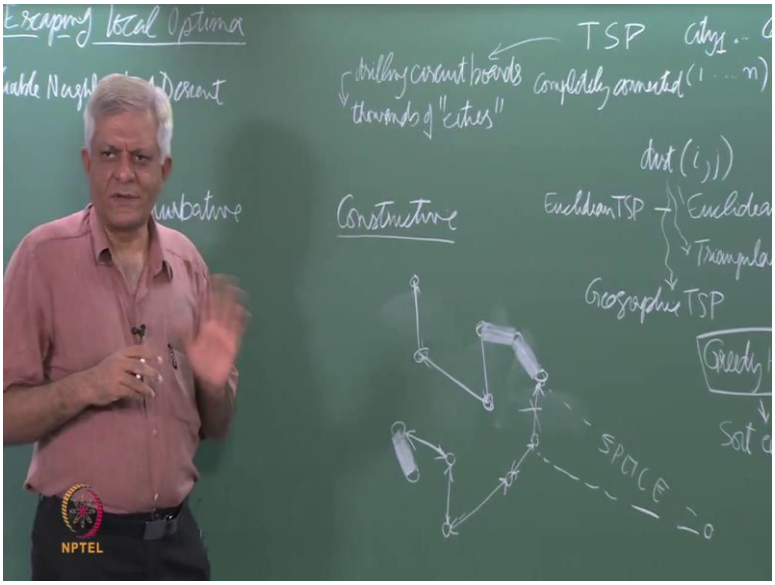
Is to say that, I mean this sound like a circular argument but, to say that there is no better solution, which is possible essentially. And if you then, guarantee that there is no better solution then, you will have an optimal solution. And it is taken them, if you look at some of the references. For example, this book (Refer Slide Time: 21:49) Something, like thousands of computing years of computing time.

So, as they are many machines working in parallel. And then, they compute this exhaustibly anumulating all TSP and finding optimal solutions. So, you must go and look at this site, which will give you optimal solutions essentially. So, today let us spent a

little bit of time. So, I am brought TSP here, because one of the things you want to do is look at this variable neighborhood descent again.

From and see how TSP can be solved using this essentially. So, basically the idea being that, what are the different neighborhood functions that we can construct? But, before we come to that, so remember that we had said that, there are two ways of solving problems.

(Refer Slide Time: 22:38)



One is constructive and the other is perturbative. So, in constructive method we construct a solution, bit by bit. And that is the whole state space search that we started with. In perturbative methods, we take a candidate solution and perturbative to look at another candidate solution. So, we did this for SAT. For example, we said that any bits string is a candidate solution. Then, you can change some number of bits, which is the perturbation you have doing.

And look at another solution but, it also constructs a solution edge by edge, if you want to say. So, let us first look at some algorithm. So, constructive solving the TSP with constructive methods and then, we will come to Perturbative methods in which case. We will look at variable neighborhood descent. And whatever other algorithm, that we are going to look at after that. So, for example given a set of cities. So, let us assume that we

have working in this Euclidean space.

So, there are actual distance is the distance, what kind of algorithm can you think of. You must have tried something some time. So, let simplest constructive method is to start. Try to stimulate what you would do, if you are doing this in the real world, so to speak. So, you start the some city, let say this one. And then, look at it is neighborhood. Now, remember this entire set is a neighborhood because, assume that our graph is completely connected it, essentially in this case.

But, when we come to perturbative methods, we will look at smaller neighborhoods where you know, all those in this the neighborhood is different. In sense that, it is not a candidate solution but, it is a neighboring city. So, you will go to the city which is nearest to you it. So, let us say you go from here to here. And then, you repeat this process. You go to the nearest city, you go from here to here, when you go I mean, it looks like this is the nearest and you will get some solution and solve.

So, this is one simple heuristic algorithm. It is a greedy algorithm. It says, started some city and go to the nearest neighbor. Then, from there go to the nearest neighbor and so on and so far, essentially. Now, obviously in some situations it will give you a very good solution, if not the optimal solution. But, you can imagine that what will happen with situations, where I have a city somewhere here. If I have a city somewhere there, what will my algorithm do, ideally what should it do?

It should go from here to that and come back here and then resume its thing. But, it will not do that because, my algorithm says go to nearest neighbor. So, I will go from here to then, I will go from here to here, go from here to here then here, here, here to here, here to here. Then I will go there and then I will come back here. So, obviously you can this is just to illustrate, that this algorithm will not always give you a optimal solution.

But, it will give you depending on what kind of problem it is, it will give you reasonably a good solution, essentially. Now, a simple variation to this is, that instead of thinking going like this, you can think of extending you towards. So, at the any given time you have a partial toward it. So, let say we have done only, till this much. And instead of

saying that, this is a one directional thing you can say that, you can think of it is going in both directions.

And then, you can say you can extend at either end of the tour. Instead of saying that, only extend that where you verse trying to stimulate a physical person moving around. You can say which ever end of the tour has a closer city extend, that towards end. That is as a simple extension of that essentially. So, I am now how do you talk about cities like this, which are far away from the rest essentially. So, one algorithm, so I am not writing these things but, any of you look at any text book on TSP, they will talk about these algorithm.

You could say that, I splice this. So, in the science that supposing I have already constructive this toward, which is not a very good toward but, I have constructed this and I only left with that city. Instead of going from this place to this place but, I will do is that, I will find out which point in the tour is closes to this city. And connects that with this and then break the stage and connect this with this. So, this will give you some improvement essentially.

So, that is but obviously, every time we do that, the complexity increases. Because, notice now that if you write a general algorithm for that, you will say that for every new point in the city, which is the closes point in the tour. So, you have to inspect in the whole toward. So, complexity will go by a factor of n in that case. So, this is one set of constructive algorithms for solving the TSP. Another popular thing, which is known a greedy heuristic says that, you sort the edges.

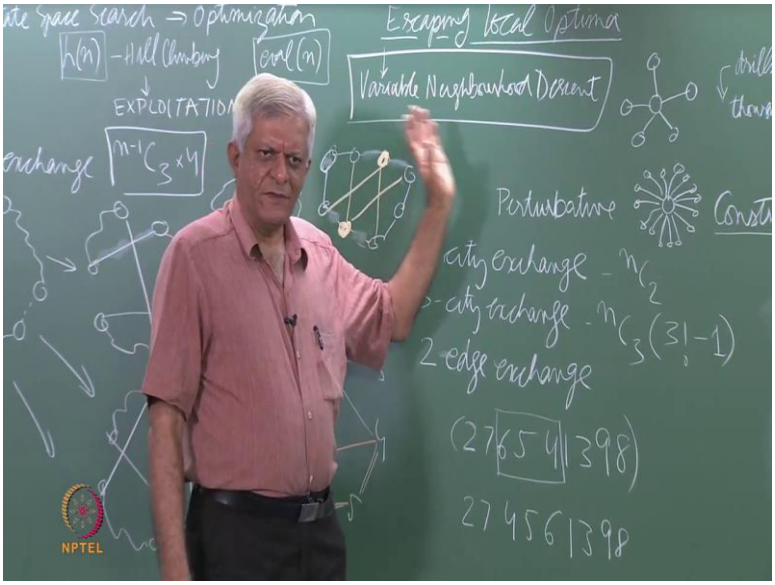
Maintain a sorted set of edges and work with edges instead of working with cities essentially. So, you can now imagine what the algorithm is. It says that, you have all the edges that are available to you in this thing. So, pick the shortest edge that is available to you and add that. So, in this example for example, it could be this one. So, this could be my first says that I add, because that is the shortest edge. Then, the next shorted could be this one. Then, it could be this one and so on and so far.

So, this is a different algorithm. It is shorting the edges. So, you short them once and

then pick the shortest edges. The hope is that, you will pick all the small edges because, for the optimal solution the more the number of shorter edges, the more likely it is optimal essentially. Obviously, when you are doing this greedy heuristic, you have to be careful that. You do not have a loop on the way essentially.

So, if you have connect a some number of cities and if the next shortest edge is forming a shorter loop then, you should not take the edges essentially. Used that one thing you should know. Then, there is another heuristic, call the savings heuristic. And all these have you know, available in many books. The savings heuristic says that, first construct n minus 1 tours of length 2. So, let me illustrate this with this example.

(Refer Slide Time: 30:58)



You take some arbitrary city and you construct tours with every other city. So, I will just take a smaller example. So, there is some city and from there you constructing this. So, there are, in this example there are five cities. So, I have constructed four tours of length two. And then, you merge to tours. So, how can you merge two tours? So, for example, I can say that, I will merge this two let most two tours. So, I will take this one from here.

I will take this one from here. And I will delete this edge and I will delete this edge and I will add in a edge, here new edge. So, the only question is which two tours should I

merge and that is where the name comes from savings heuristic. It says, select that pair of tours in which you get the maximum savings. What is the savings? That, your length you had four edges in this to start with. So, you add at the lengths. So, l 1 l 2 l 3 l 4 and now after merging, you have l 1 l 4 and a new one, that is call it l 5.

How much is the saving, how much is l 1 l 4 l 5 better than l 1 l 2 l 3 l 4. Choose that combination, which gives you the maximum savings essentially. So, now that you have merge these two. Then, you could merge this larger tour, with this third one. For example, so you could delete this edge and add this edge. And add this one and so on and so far. We keep merging. So, in the n minus 2 merge operations you will finally, get the tour essentially.

Again, it is heuristic algorithm. Not guarantee to give you an optimal solution but, in general all these algorithms give you a reasonably good solution for Euclidean TSP, at least essentially. So, let us now go to the perturbative approach. What does perturbative approach says? That it, you take some candidate and generate it is neighborhood essentially. The which is, kind of different from this. Here you are constructing the tours gradually.

Here you are saying, I have one tour given to be and I am going to produce a set up neighborhood tours and choose one of the essentially. Exactly like, what we did in SAT. I have a candidate solution. I will perturb it by changing some number of bits, to get new solutions and move to that new solution. This was like hill climbing like algorithm, we are approaching now. In this, the question is what are the neighborhood functions at we can talk about here.

Can we think of that? So, given that tour for example, 2 7 1 6 4 3 8 5 9. How can I generate a neighborhood around? What is the property that a tour by satisfy? And if basically should be a permutation of those n numbers essentially. So, essentially you want to generate some new permutations and explore one of them essentially.

Student: (Refer Time: 35:04)

Swapping adjacent numbers, would give you a new tours. So, instead of saying this I, so I can pick two, any place and swap two numbers. So, for example, I can replace 1 6 by 6 1, that is what you have saying. So, that is one. So, you can imagine how many such moves are possible. You can make n minus 1 swaps here. So, this will have n minus 1 neighbors essentially. So, either the first two are the second two are the third like in line it.

So, you can journalize that to something which we call as a 2 city exchange. And what that says is that, take any two cities in the tour and exchange their positions. So, this is a particular case of that, it says that take two adjacent numbers and exchange that. This is the operator says, the take any two cities and generate essentially. So, how many neighbors will this have. I can take n c 2. Two cities in n c 2 ways and for each way that I pick, there is only one neighbor I would be generate because, I can only exchange their positions essentially.

So, I will have n c 2 neighbors essentially. Or I can have 3 city exchange, what this says is that, takeout any three cities from the tours. And put them back in some different order essentially. So, I can pick them in n c 3 ways and then I can put them back in 3 factorial minus 1 because, 1 I do not want to put it back in the same order. So, 3 factorial minus 1 order. So, I will generate that many these things. So, what is this two city exchange is doing?

If I have a tour like this then, if I am going to exchange this one with this one for example. Now, what it means is that, I was going in this order. But, now I am going to go, instead of this I am going to come here. And from here, I am going to go there. So, this will go away. Then, instead of coming here I will go like this. If you must visualize this, this what happening when I am exchanging this city with this city.

Originally, it was a circular looking tour. Now, because I will exchange the position of this two cities from here, I am going to go here, from here I am going to go there. And then, continue here. And then, this something finger will have essentially. Now, it turns out that city exchanges are not the best way of thinking about things. But, edge exchanges are end of the easier to think about essentially.

So, for example, I have two edge exchanges. It says that, remove some two edges from here tour and insert two new edges, instead of that. So, let us this see, what this means? Supposing, I have tour like this or let us see I have I am just trying them in the circle because, this easier to this in factors. Of course, it is going to be sought of distributed over some space. I mean cities are never arrange in nice circle edges but, it is easier from me essentially.

So, one of the thing that we would want to do is that, you know this Euclidean TSP is to generate random problems, which means take a two dimensional space. Let us say a computers, monitors screen and randomly place it there. And then, say find the optimal tour for that essentially. Now, if I have a tour like this. So, ideally if I have a tour which is something like this, you can see that I have two very long edges in this tour essentially.

If I could somehow remove them and replace them, into two shorter edges, keeping it as a complete tour, I can transform this problem in to another problem in which, instead of these edge and instead of this edge, I add this edge. I add this edge and I add this edge. So, whatever them, I have deleted two edges from my tour. And replaced it with two more edges and there is only one way I can do that essentially.

If I delete any two edges, I have only one new tour that I can create, in this example. How will I do this, two city exchange? So, let us say we have that tour. So, something like that, 2 7 6 5. Supposing, I have representation like this, how can I implement two edge exchange? So, anyways, so it is not comes. So, once you know the answer, it is simple. Essentially, you take a sub slink here and reverse the sub slink.

So, 2 7 4 5 6 1 3 9 8, so let us see whether it is really doing, what we are saying it is doing. So, which other two edges that I have removed here? The edge that I have removed is 1 from 7 to 6 and 4 to 1. The other edges remain the same because, from. So, here instead of going from 7 to 6, I am going from 7 to 4 and 4 is still connect to 5. So, maybe I should draw this. This is called as cities. So, I go from 2 to 7, 7 to 6, 6 to 5, 5 to 4, 4 to 1, 1 to 3, 3 to 9, 9 to 8 and 8 back to 2.

That is the last step. This is the tour that I started with. Now, I am saying just rotate this

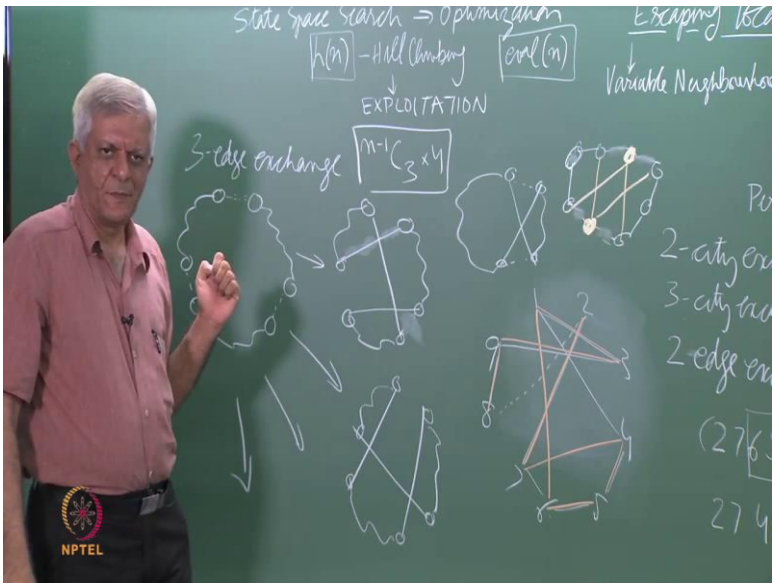
sub slink, 6 5 4 and that gives me, this tour 2 7 4. So, let us follow that tour from 2, I am going to 7, again from 7 I am going to 4, 4 I am going to 5 is part of the same tour, notice. From 5 I am going to 6, from 6 I am going to 1, from 1 I am going to 3, which is also part of the whole 2. Then, from 3 I am going to 9, 9 I am going to 8 essentially.

So, everything all the edges except for this 1 to 4 edge and 7 to 6 has been replaced. So, I have taken up this edge 7 to 6 and 1 to 4. And replace them with this edges, which goes from 7 to 4, which is this edge and which goes from 1 to 6, 6 to 1 in this new edge. So, this is this one and this one is new edges, that I will introduced. So, rotating a sub slink will effectively do a two edge exchange for use essentially.

Why this intuitively more appealing than city exchange? I hope it is intuitively more appealing than the city exchange because, it is the edges it is the edge cost, bit adds up to the total cost of the solution. So, what would you ideally want to do is, to inspect your solution. And off course, between every two cities there is an edge cost. Pick those edges, which seem to be very high cost. And replace them essentially.

So, the original example that I have drawn, which was like you go some tour like this. So, if you have some tour like this, where you are going like this then, these two are very long edges. If I remove them and replace them with this, I will get a shorter tour. So, in that sense manipulating edges is more appealing because, you can at least apply the slink of removing very long edges essentially. So, 2 edge exchanges just one example. You can look at 3 edge exchange.

(Refer Slide Time: 45:10)



So, 3 edge exchange shows, you must compute how many neighbors are there are and so on. So, you can pick two edges in n minus 1 c 2 ways and then you will get n minus 1 c 2 successes. In 3 edge exchange let say, this is the problem. This is the original tour given to me and I am removing this three edges. So, one is this one, one is this one and one is this one. And now, we can see that we can put them back in different ways.

So, one way you can put them back is, that you can connect this to this, from here. So, you must be careful not to form a cycle. So, from here I cannot go to this. So, that not allowed. So, from here I can go to, let us it is this. And here I can to this. No, something is wrong. Now, here there are 3. So, I cannot do this. So, I can put this here. Then, from here I can go to this. And from there, I can go to this because, I have a new tours essentially.

So, it turns out that there are four different ways to put them back. So, let us try one more and I will leave the other two is an exercise for you. So, you understand this notation. It is a kind of shorthand. This is some tour with some n cities but, we have drawn only six cities, representing those six, three edges that we are going to remove. So, this edge we are removing, this edge we are removing and this edge we are removing.

Less of the tour remains the same but, we have puttng back three edges in slightly different place. So, instead of bliss now I can instead of going from here to here, I can say go from here to here. Then, from here you go like this. Then, from here you go like this. And there are two more different ways of doing it. So, I will leave that as a small exercise for you. So, three edge exchange if you take of three cities three edges, you can put them back in four different ways.

And you can take of three edges in n minus 1 c 3 ways, so into 4. These many neighbors you get essentially. So, what I am trying to illustrate here is that, when you treat the TSP perturbative problem, you take any candidates solution. And you can generate a neighborhood of candidate solution surround it, using two some number of city exchanges or some number of edge exchanges. And they are neighborhood functions of different density essentially, which means that you can apply the variable neighborhood descent functions essentially.

I am not writing this algorithm again. This variable neighborhood descent basically says that, if you have a set of neighborhood functions are raised in a order of increasing density. So, by density means how many neighbors thus, a given candidate have? So, if a candidate has let us say 5 neighbors or if a candidate has let us say 12 or 15 neighbors. Then, this neighborhood functions would more dense and this one essentially. And why did we want denser neighborhood functions because, it is likely that the best amongst them is the local maxima essentially.

Or rather, it just likely that the given no reason local maxima, which means it does not have a better note surrounding it. So, there have the more note surrounding a note, the more the likelihood of a better note existing, if there are better notes in the spaces essentially. So, the idea of variable neighborhood descent is that, to start with the simplest neighborhood function. Why are you do not we want to work with the most dense function first?

Because the cost of making a move is proportional to the number of neighbors, that we have because, we have to inspect all the neighbors and then pick the best amongst them. So, I like in SAT if every, if we can change any number of bits then, it amongst to doing

the complete beautiful search. We do not want to do that, we want to starts with neighborhood functions move on to dense a function and so on and so far.

So, in today's class basically what we have done is, looked at the TSP problem. And we are not really looked at the new method for escaping local maxima, which you will do in the next towards see classes. But, when we are doing that, we will keep in mind how to solve the TSP. So, remember that this kind of this thing. Just a quite comment about these two city exchange, remember this was the two city exchange, I exchange this city and this city.

And as the result, I added this four new edges. Remove the four edges because, the neighboring edges from here and I added for. So, you can see that this is the particular case of four edge exchange essentially. In practice, off course if you remove those four edges, you can put them back in many different ways. So, two city exchange is just a special, one of the cases of those four city exchange. Four edge exchange, which is going to give us a denser functions essentially.

So, the more the cities of the edges we remove, the more the ways you can put them back in and more the ways you can remove them in. And they give you denser functions essentially. So, that is one mechanism for doing that this. I will just stop here. We will, when you comeback we will look at a new algorithm, for try to escape local maxima. How can we escape this local maximize? That is going to be our objective in the next few lectures essentially.