Advanced Operations Research

Prof. G. Srinivasan

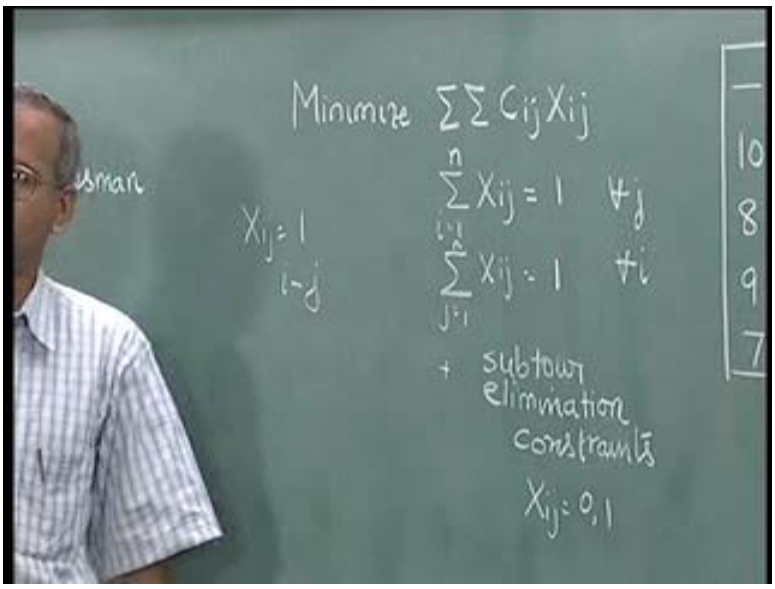
Department of Management Studies

Indian Institute of Technology, Madras

Lecture No. - 25
Branch and Bound Algorithms for TSP

In this lecture we continue our discussion on the Travelling Salesman Problem. In the last lecture we saw a branch and bound algorithm to solve the TSP. In this lecture we will see another form of a branch and bound algorithm to solve the same problem.

(Refer Slide Time: 00:42)



We have earlier seen the formulation of the Travelling Salesman Problem where X_{ij} equal to 1, if the person goes from i to j immediately equal to 0 otherwise. This is the formulation of the Travelling Salesman Problem where we try to minimize sigma $C_{ij}X_{ij}$ subject to X_{ij} equal to 1 summed over i for every j and X_{ij} equal to 1 summed over j here for every i.

These are called the subtour elimination constraints which make sure that the feasible solution to the Travelling Salesman Problem is a tour. Another way of looking at this formulation is, if we temporarily pull out or leave out the subtour elimination constraints what we have is actually the assignment problem.

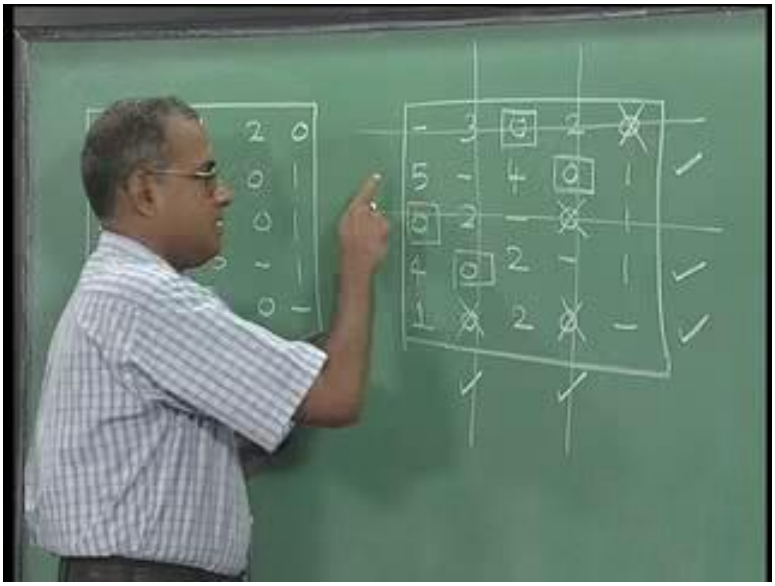
So the Travelling Salesman Problem can be looked upon as a further constrained assignment problem or alternately removing a set of constraints from the Travelling Salesman Problem can actually give us the assignment problem. So the assignment problem can be seen as a relaxed Travelling Salesman Problem

So we use a simple principle that if there is an optimization problem and we remove a certain constraints or relax the optimization problem and solve the relaxed problem, if it turns out that the optimal solution to the relaxed problem satisfies these constraints, then it is optimal to the original problem or the restricted problem

So we use that idea and we relax the subtour elimination constraints so that the resultant problem is an assignment problem. So if we solve this assignment problem and if it turns out that the optimal solution to the assignment problem is feasible to the TSP, then it is optimal to the TSP as well. So we use that idea and first solve an assignment problem. We also use another principle that since it's a minimization problem and we are solving a relaxed problem; the optimum solution to the relaxed problem is always a lower bound to the given problem or to the given restricted problem

So in more than one way we use these ideas to develop a branch and bound algorithm to solve the TSP optimally. First we relax or leave out the subtour elimination constraints and solve the relaxed assignment problem and see whether the optimal solution to the assignment problem is feasible to the TSP. So first let us solve the assignment problem using the Hungarian algorithm, the way which we have already seen in the fundamentals course.

(Refer Slide Time: 03:50)



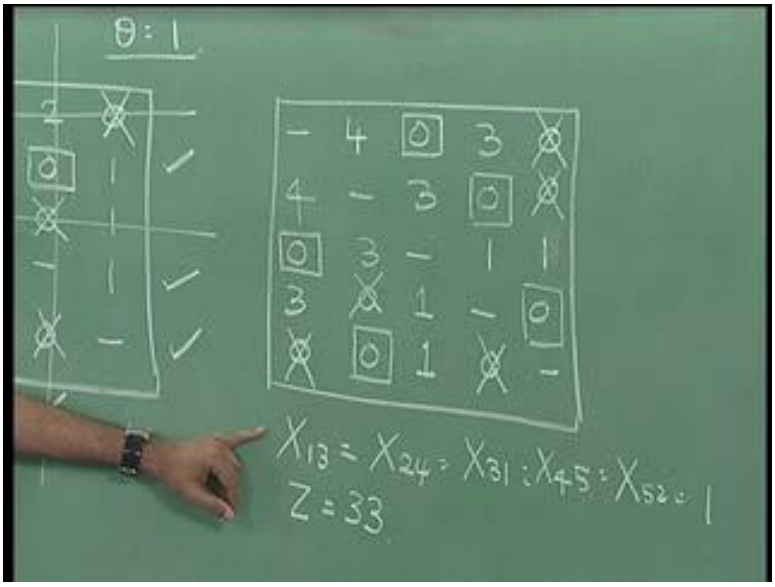
So we first take this matrix. In order to solve the assignment problem we first subtract the row minimum to get: dash 3 1 2 0, 5 dash 5 0 1, 0 2 dash 0 1, 4 0 3 dash 1,1 0 3 0 dash. So, we have subtracted the row minimum from every element in the row. Now we subtract the column minimum. The column minima are 0, 0, 1, 0, and 0 respectively. So we get dash 5 0 4 1, 3 dash 2 0 0, 0 4 dash 2 2, 2 0 0 dash 0, 0 1 1 1 dash. We have subtracted the row minima and the column minima to get the reduced matrix to solve the assignment problem.

Now we start making the assignments. There are two assignable 0s so at the moment we don't make an assignment. There is only one assignable 0 so we make an assignment here; this goes, this also goes. Here there is only one assignable 0, so we make an assignment here. We have one assignable 0, so we make one assignment here; this goes. Here we cannot make an assignment. Now we look at the columns, these two columns have assignments. This has only one assignable column, so we make an assignment here; this goes. Now we cannot make any more assignment. So the initial portion of the assignment part of the algorithm is complete. We have made four assignments as against five that we require.

So we go to the ticking procedure. The ticking procedure is tick an unassigned row. If there is a 0 in that column tick that column. If there is an assignment in a ticked column then tick the corresponding row. If there is a 0, tick that column. There are no 0s here. There is one 0 which is already ticked.

Now draw lines through unticked rows and ticked columns. So one line comes here; one line comes here. For ticked columns, one comes here; one comes here. Now we have crossed or we have made sure that every 0 in the assignment matrix either has a horizontal line or a vertical line or both passing through it.

(Refer Slide Time: 07:16)



Now find out the minimum of the numbers which do not have any line passing through them. That happens to be one which is either this or this or this. Now go back and redo this matrix, such that if there are no lines passing through then subtract the theta; if there are two lines passing through add the theta; and if there is only one line passing through, keep it as it is. So we get: dash, 4 (4 comes because there are two lines passing through this, 3 plus 1, theta is 1, which is the minimum of the uncrossed numbers which could be either this or this or this) 0, 3, 0. These 0s remain as 0, because there is only one line passing through them. Now this becomes: 4, dash, 3, 0, 0; 0, 3, dash, 1, 0; this 1 comes because there are two lines passing through, so 0 plus theta so you get 1. Here there is only one line passing through so this will remain as 1. Now this is 3, 0, 1, dash, 0, now here this 2 becomes 1, there are no line passing through; this 1 becomes 0, dash, remains 0 remains as 0. Here it becomes: 0, 0, 1, 0, dash. So now we have recreated or changed this matrix.

We have already seen the optimality of such an approach when we studied the assignment algorithm in the earlier course. Now for this reduced matrix we start making allocations. There are two 0s, so temporarily we don't make an allocation.

There are again two 0s, so we don't make an allocation, now here there is only one 0, so we make an allocation here. Now here and this goes. There are two 0s, so we don't make an allocation and once again there are two 0s, so we don't make an allocation.

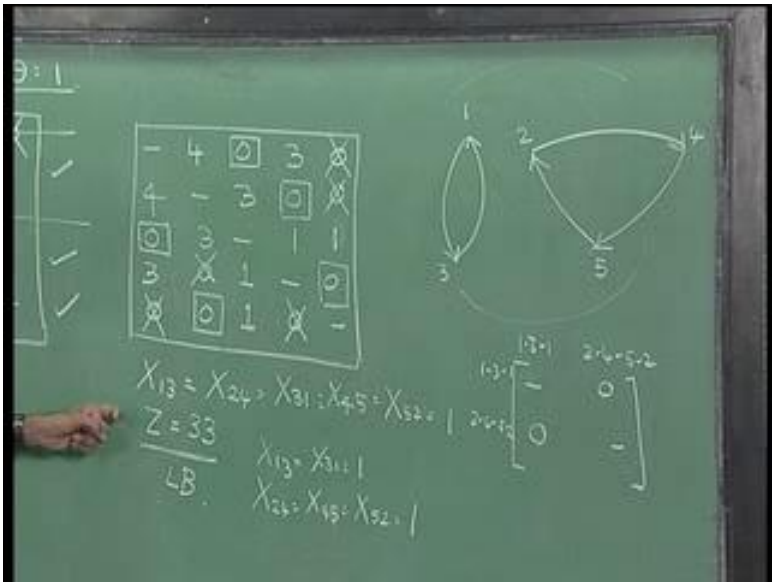
Now we look at it column wise. This column has two 0s, so we don't make an allocation. This column has only one 0, so we make an allocation. Now this column has, so this goes. Now this column has two allocations, so we don't make an allocation; again two 0s, so we don't make an allocation.

Again go through the rows once again; two 0s here, two 0s here, and two 0s here. So we get into a situation that we are not able to make allocations with 0s. So we make an arbitrary allocation here. We start with two 4s which is an arbitrary allocation; so this goes, this also goes. So straight away this row now has only one assignable 0; so make an allocation here, this goes and therefore this has one assignable 0.

So we have got five allocations here and the assignment solution is X_{13} equal to X_{24} equal to X_{31} equal to X_{45} equal to X_{52} equal to 1 with the corresponding value Z equals 1,3 is 8; 2, 4 is 8 plus 5 is 13; 13 plus 8 is 21; 21 plus 6 is 27; 27 plus 6 is 33. So we get into an assignment solution with Z equal to 33.

Now if we look at this assignment solution carefully and we need to verify whether this is feasible to the TSP. If this is feasible to the TSP then this is optimal. If we look at this carefully, we have a subtour which is X_{13} equal to X_{31} equal to 1; which is a subtour. We have X_{24} equal to X_{45} equal to X_{52} equal to 1.

(Refer Slide time: 11:34)



So there are two subtours in this solution and these two subtours are shown like this: 1, 3 and 3, 1 and then we have 2 to 4, 4 to 5 and 5 to 2, so there are two subtours here. Now because the optimal solution to the assignment is infeasible to the TSP and because assignment problem is a relaxed form of the TSP, now this 33 becomes a lower bound to the optimum value of the TSP, so this 33 becomes a lower bound to this.

Now we can start a branch and bound algorithm with 33 as a lower bound to the optimum value of the TSP. We can also do one more thing where we can see if we can tighten this lower bound a little bit. What is the motivation to do that? There are two subtours here and in order to get a tour we have to break one of these links as well as one of these links. Two links have to be broken and those two broken links have to be replaced by one that moves from here to here and one that moves from here to here.

So we will either leave this out or leave this out or leave one of these and then we need to add one from this to this and add one from this to this. Now if we go back to the last reduced matrix and if we create something like this. We create a subtour to subtour matrix which is 1 - 3 - 1, 2 - 4 - 5 - 2 versus 1 - 3 - 1 and 2 - 4 - 5 - 2.

Now 1 to 2, 1 to 4, 1 to 5, the minimum is 0, 3 to 2, 3 to 4, 3 to 4 the minimum is 1. So from 1 - 3 to 2 - 4 - 5 - 2, the minimum is 0. All these will obviously have a dash and from 2 - 4 - 5 - 2 to 1 - 3 - 1, 2 to 1, 2 to 3, 4 to 1, 4 to 3, 5 to 1, 5 to 3; the minimum is also 0.

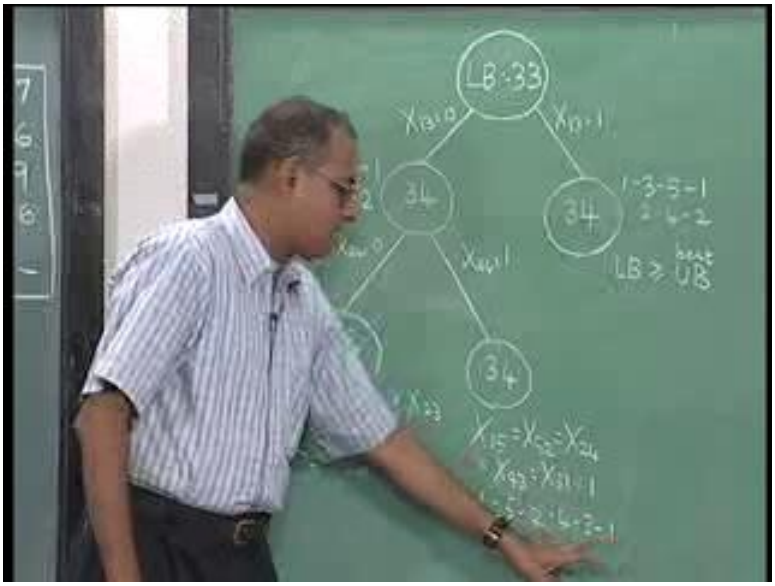
Now what does this matrix convey to us now; this has been pulled out from the last of the reduced matrix from which we got the assignments. We also said that in order to get a feasible solution to the TSP which is a tour; we have to pullout some edges from here and add some edges there. Now all these edges which are here in the subtours have a weight of 0 from this matrix, because these represent an assignment solution using this reduced matrix.

So whatever we pullout have weights equal to 0 with respect to this matrix and whatever we have to put in; so at least one arc has to go from either this to this or this to this. There are two arcs that have to go; one going from this to this and one coming from this to this. So this is "from to path" of an assignment matrix and these weights are 0 respectively. So if we can solve these also represent the minimum weight that are there, taken from this reduced matrix.

If we have to really add something into this so the minimum weight that we may have to add is 0 and again a minimum weight that we may have to add is 0. Solving this assignment problem would give us the minimum weight that we may add to this 33 which we will have to add this 33.

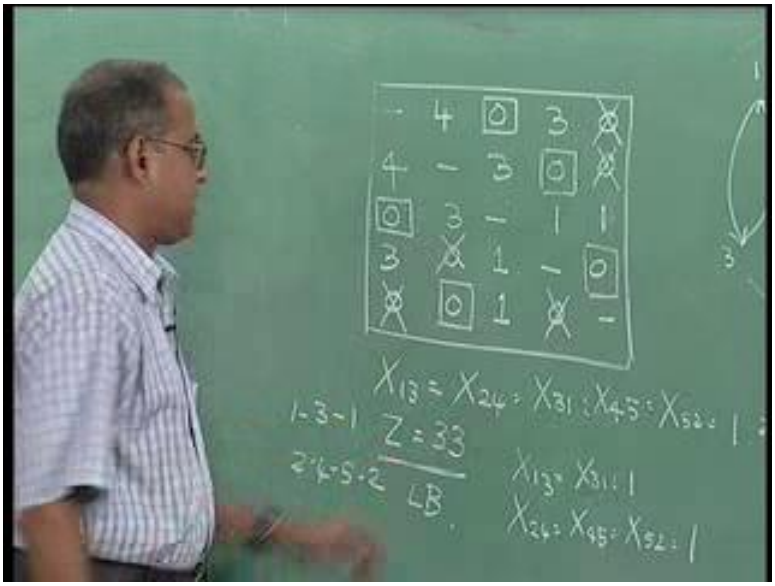
For this numerical illustration it turns out that the solution is 0 here, so the lower bound 33 does not increase. If on the other hand, one of the 0s say had been 1, then we would go back and say that the minimum that we may have to add is 1 and therefore the lower bound would have moved or increased from 33 to 34. So this is one more way of effectively tightening this lower bound or making this lower bound slightly more than that given by the assignment solution. This can also be used suitably so that the lower bound gets better and better. In this particular instance the increase in lower bound happens to be 0; so the lower bound stays at 33. So now how do we start with the branch and bound algorithm?

(Refer Slide Time: 16:40)



So we start the branch and bound algorithm with the lower bound equal to 33 given by this 33 it happens that the increase is 0, so 33 plus 0 remains as 33. Now there are two subtours which is 1 - 3 - 1 and 2 - 4 - 5 - 2; so we have to eliminate these subtours. We take the smaller one which is 1 - 3 - 1, subtour of length 2 and then create two branches here. To begin with we would say X_{13} equal to 0 and X_{13} equal to 1. These two branches are possible so 1 - 3 either lies in the solution or 1 - 3 does not lie in the solution. When X_{13} equal to 0 what we have to do is we will go back to this matrix and put a dash or infinity, the way you model X_{13} equal to 0 is by changing this to infinity, so that 1 - 3 does not remain in the solution, so we can change this to dash.

(Refer Slide Time: 17:47)

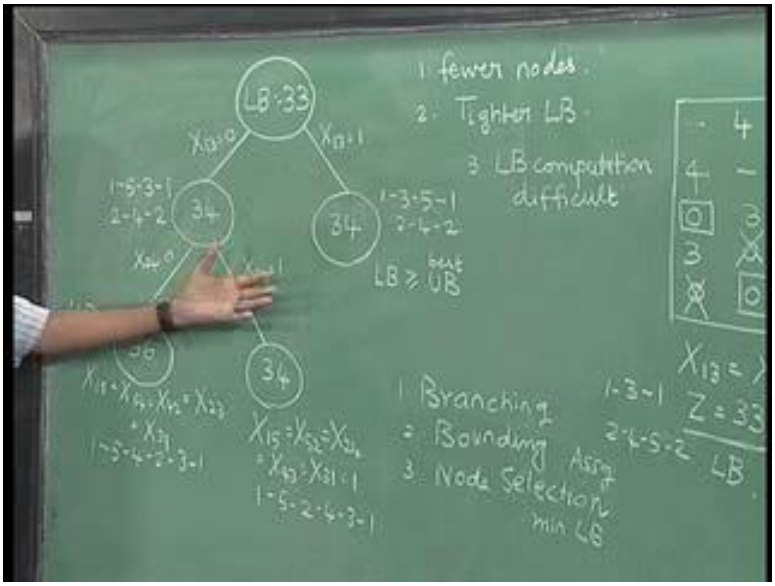


So X_{13} equal to 0, we can simply put a dash here and then solve this assignment problem again. We may get a final matrix like this, so whatever we have here will be a lower bound and we will check if it is feasible to the TSP.

Now this lower bound could also be further increased by considering this, if there are subtours. So we can use this as an internal routine, to try and get these lower bounds. So when we put X_{13} equal to 0, which means, we put a dash here and solve another assignment problem. Now we get into, the lower bound now increases from 33 to 34, we do not get a feasible solution; we still get two subtours; we get 1 - 5 - 3 - 1 and 2 - 4. We have two subtours when we solve the assignment problem with X_{13} equal to 0 which means we put a dash here and solve it.

The optimal solution to the assignment problem has objective function value of 34, which is a lower bound and it has two subtours 1- 5 - 3 - 1 and 2 - 4 - 2.

(Refer Slide Time: 19:00)



When we solve the other one; when we put X_{13} equal to 1, now what effectively happens is that, since we force X_{13} equal to 1; it means that we can actually leave out the first row and third column and solve a resulting four by four problem. We add this 8 to the objective function value of the optimal solution; in which case we also get a lower bound equal to 34, but we get 1-3-5-1 and 2-4-2.

One can easily observe that actually these two solutions are one and the same with respect to the TSP, because 1-5-3-1 is the same as 1-3-5-1 and so on. We have now made two branches. Remember that for this one, we put a dash here and solved a five by five; for this one we forced X_{13} to 1, solved the remaining four by four and added that 8 into it and updated the solution.

Now it turns out that both of them are still infeasible to the TSP and have subtours. So we have to eliminate the subtours again. So what we can do here is by doing this; we take this subtour 2, 4, so we put X_{24} equal to 0 and X_{24} equal to 1. When we do this, we have to solve another assignment problem and when we solve this particular assignment problem, we have to force X_{13} to 0 and X_{24} to 0, which means we will have a dash here as well as a dash here.

When we do this, the lower bound goes to 36, but we have a solution which is X_{15} equal to X_{54} equal to X_{42} equal to X_{23} equal to X_{31}, which is a feasible solution; 1-5-4-2-3-1 which is a feasible solution to the TSP.

Now we have a feasible solution to the TSP, so this acts as an upper bound. The moment we know that we have a feasible solution with value equal to 36, we are not interested in any other solution which will have value 36 or more. Now if it turns out that one of these nodes had a value greater than 36, then we can fathom that node by using this feasible solution; using this condition that lower bound is greater than or equal to the current best upper bound.

Right now this node has a lower bound of 34, so we cannot fathom it based on this result. We go back and evaluate this. Now when we start doing this node, we have to put a dash here, but we make an assignment here; so we leave this row and this column; solve the remaining four by four and add this into the solution, so this gives us a lower bound of 34. We get a solution X_{15} equal to X_{52} equal to X_{24} equal to X_{43} equal to X_{31} equal to 1. So the lower bound is 34, but it also happens that we have a feasible solution to the TSP with value equal to 34.

We can verify this; 1, 5 is 7, 5, 2 is 6; 7 plus 6 is 13; 13 plus 5 is 18; 18 plus 8 is 26; 26 plus 8 is 34. We have a feasible solution with Z equal to 34. Now the moment we have a feasible solution with Z equal to 34, we can actually fathom this, because branching from here would only give solutions with value 34 or more. So this is fathomed based on the lower bound is greater than or equal to current best upper bound. Because the best upper bound has been updated from 36 to 34, the moment we got a feasible solution with Z equal to 34.

So now we do not have any node which is right now unfathomed or open, therefore the algorithm terminates with the current best upper bound as the optimal solution. So the optimal solution is Z equal to 34.

In fact for this example, we already know that the optimum solution is Z equal to 34 and we have found out that the solution is the same. If we had branched from this, then we would have got the mirror image of this solution. This solution was 1 - 5 - 2 - 4 - 3 - 1 and if we had branched from here, we would get the mirror image solution which is, 1 - 3 - 4 - 2 - 5 - 1, the problem being symmetric with the same value 34 for the objective.

This is how we use a branch and bound algorithm that uses ideas from the assignment problem. The assignment problem comes because we have shown here assuming that if we relax certain subtour elimination constraints the Travelling Salesman Problem becomes an assignment problem. So, the optimal solution to the relaxed problem will provide a lower bound to the restricted problem that we looked at. So at every branch of the branch and bound

tree we found out that the optimum solution to the assignment problem gives a lower bound to the TSP. If it turns out that it is feasible to the TSP, then we can fathom that node and that is how this branch and bound has been developed.

Now this branch and bound was developed by Eastman earlier somewhere in the late 50s or early 60s, so this is called the Eastman's branch and bound for the Travelling Salesman Problem. How does this branch and bound compare with the earlier branch and bound? The earlier was a very rudimentary branch and bound based on a simple bounding principle, that some of the row minima would give us a lower bound. Now we have a quick comparison of this and the earlier would tell us that this branch and bound actually has a fewer nodes. It has fewer nodes than the earlier one, earlier one had several nodes. This had a tighter lower bound.

In general, the lower bound at any node was actually better than the lower bound which was there in the earlier branch and bound algorithm, but the only difference is the lower bound computation is difficult in this compared to the earlier where the lower bound computation was much easier. Here it takes more CPU time or computational time in every node because we are solving an assignment problem in every node.

So in terms of number of nodes which means in terms of memory requirement, this is slightly better than the earlier branch and bound, but in terms of speed or time taken to evaluate each node, the earlier one was faster. So, there could be a chance that based on speed the earlier one was slightly better, but the earlier one would consume more memory than this.

In general this branch and bound would be preferred to the earlier one, because it evaluates fewer nodes even though the time spent on evaluating each node is a little more than the earlier one. Every branch and bound algorithm has three important aspects; branching strategy, the bounding strategy and the node selection strategy.

In most branch and bound algorithms, the node selection will be the node which has minimum lower bound, so that the branch from there. The branching strategies here is in terms of a 0, 1 variable; in the earlier case the branching strategy was, you take a particular city and then explore different ways of either coming into the city or leaving the city.

Now the bounding strategy is for solving an assignment problem here, whereas earlier there sum of the row minima gave us a lower bound. Depending on the branching strategy, the

bounding strategy and the node selection strategy, performance of different branch and bound algorithms are different.

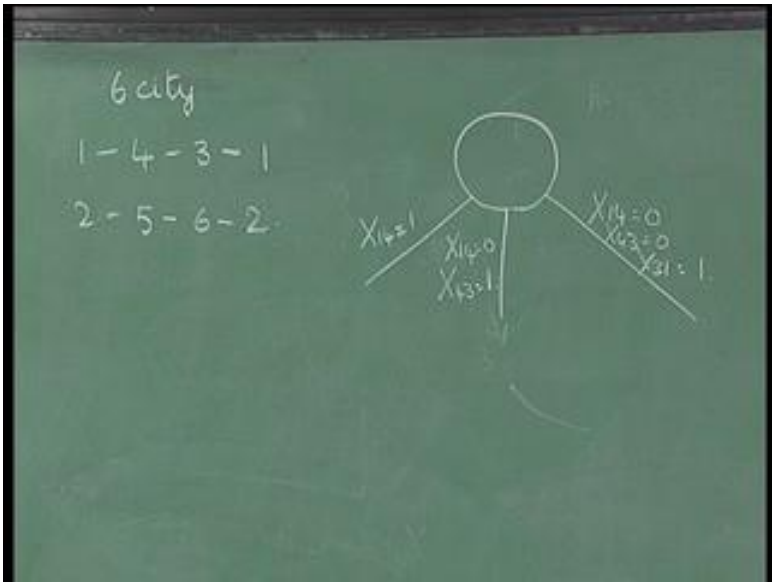
Now this particular branch and bound is important because it is based on the principle that the relaxed version of a Travelling Salesman Problem is an assignment problem. The assignment problem is unimodular; it has an efficient algorithm. So the optimum solution to the assignment problem is a lower bound to the TSP and if it is feasible it could be optimal to the corresponding TSP. So we use that idea in the Eastman's version of the branch and bound and then created this branch and bound tree.

This branch and bound compared to the earlier one has fewer nodes, but spends more time in evaluating the bound in every node. We also saw that even this itself is a bound but the bound could be further tightened by something like this. Now there is just one more aspect that we will have to look at before we wind up our discussion on this particular type of a branch and bound.

Now the example that we have used actually is a symmetric matrix. Whenever we solve a symmetric matrix, there will always be a tendency for to have subtours of smaller length; subtours of 2 or 3. The reason being if X_{ij} equal to 1; the symmetry would kind of force X_{ji} also to be equal to 1. So when we solve symmetric TSPs, this will give a large number of smaller sized subtours; so we would have that possibility or if we are solving a non-symmetric TSP, then we could have a smaller number of large subtours, so both are possible. So, smaller number of large sized subtours; and large number of small sized subtours.

When we look at the second one, we could simply take any one of the subtours and keep branching with a 0 and 1, which makes it easier. But when we have a smaller number of large sized subtours, then over a period of time people came up with some interesting ways of creating this branch and bound tree, which we will possibly explain here.

(Refer Slide Time: 31:26)



Let us say if we have two subtours. Let us say we solve a 6 city problem. If we are solving a six city problem and let's say we had one like this: 1 4 3 1 and then we had another one like this, 2 5 6 2; we could branch by considering either this or this. So, if we take this subtour, let us say, this is your starting lower bound; there are three links here. We create three here with X_{14} equal to 1; we have X_{14} equal to 0 and X_{43} equal to 1; and X_{14} equal to 0, X_{43} equal to 0, X_{31} equal to 1.

Now, whenever we create something like this; when we say X_{14} equal to 1, then it means we go back here, we fix this to 1, solve the remaining four by four and update. Whenever you have a 0 and a 1, here it means, you put the corresponding C_{ij} to infinity, fix this to 1, solve the resulting problem and then update it. It means you put two values to 0, fix one of them to 1, solve the resultant problem and update it. So whenever you have larger sized subtours, this is a very convenient way of doing this branch and bound. This is attributed to Held and Karp somewhere in the early seventies, who gave this method of actually making this branch and bound slightly better.

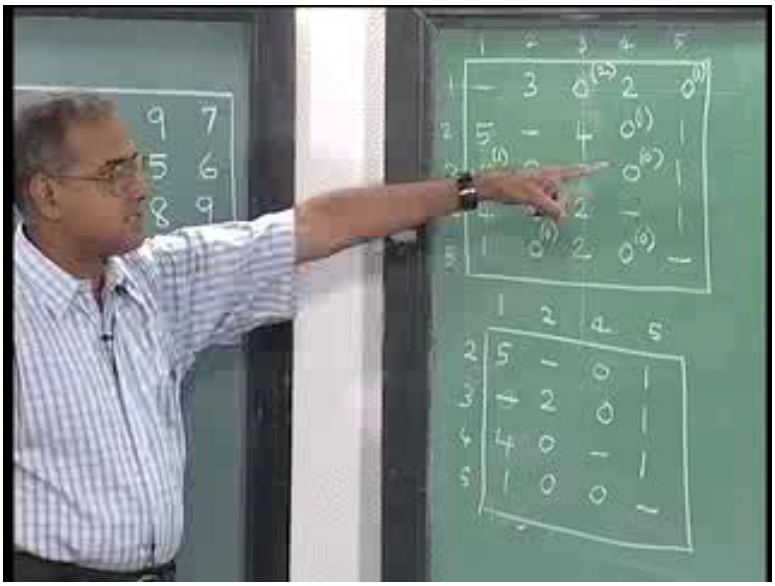
The difference here, of course, is if we follow this type of a branch and bound then every time we branch, we will only branch with two nodes, one fixed at 0 and the other fixed at 1; whereas, when we use this type of a branch and bound here, if there are three links here, then there will be three branches and so on.

This is generally found to make this whole implementation of this type of branch and bound algorithm better. This is how we use the assignment idea to solve the Travelling Salesman Problem optimally through a branch and bound algorithm. Now we will go and see another type of a branch and bound algorithm to solve TSP.

Now, let us go back to this starting matrix. These are the distances. We have already seen that sum of the row minima would give us a lower bound, because sum of the row minima is centered around the idea that if we have reached city 1 from here, we have to leave city 1 and reach either 2 or 3 or 4 or 5 and the minimum distance is a minimum that we have to travel. So sum of the row minimas would give us a lower bound.

First, let us make a reduced matrix based on subtracting the row minima from this. The reduced matrix will look like this.

(Refer Slide Time: 34:54)



We have: dash 3 1 2 0, 5 dash 5 0 1, 0 2 dash 0 1, 4 0 3 dash 1, 1 0 3 0 dash. This is the row minima that we have. Now the sum of the row minima happens to be 7 plus 5 is 12, 12 plus 8 is 20, 20 plus 5 is 25, 25 plus 6 is 31, 31 is the sum of the row minima. If we look at this reduced matrix, there is one 0 here, there are 0s here, there is no 0 here, there are 0s here and there are 0s here.

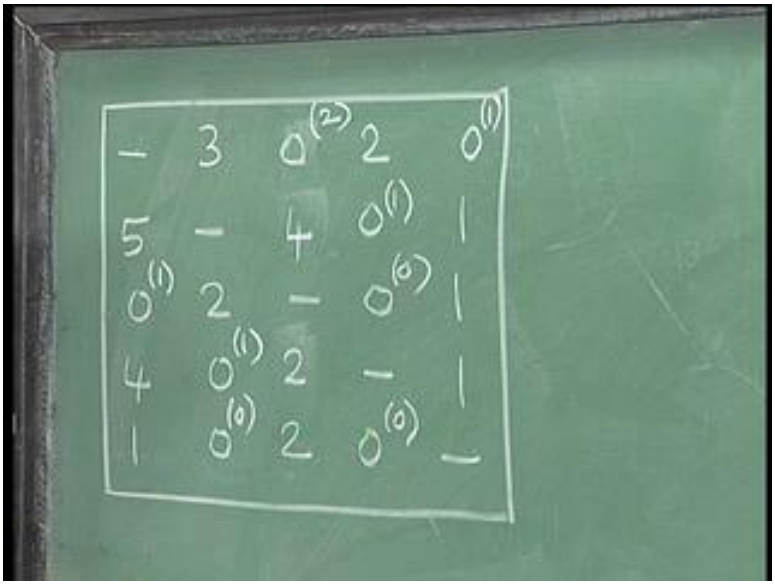
Now if we try and apply the column minima principle, which means for a particular city we will have to come into that city from some other city. When we look at this city, there is a way by which you can come into the city by incurring an additional 0 which is from here.

Similarly from here, whereas, from here we have to incur an additional one if we want to come to city number 3, so we can subtract this column minimum again to get 0s, so that in principle we have a 0, which means, that from some city we can come into the city by incurring 0 cost.

We subtract again the column minimum from this and subtracting the column minimum would give us 0, 4, 2 and 2; and then we have added plus 1. So the lower bound now becomes 32 and we start the branch and bound algorithm with lower bound equal to 32.

Now we look at this matrix and as in the assignment problem, we would like to make assignments where there are 0s so that we do not incur additional distance either to reach that node or while leaving that node. So we are interested now in these 0s to make assignments in these 0s. We start asking another question as what happens if we are not able to make an assignment in a particular 0 though assigning at 0s are desirable.

(Refer Slide Time: 37:58)



So if we take this particular 0, if I am not able to make an assignment in the 0, what happens is we would like to make an assignment in the next highest element, which also happens to be 0 for the row, but it happens to be 2 for the column. So if I am not able to make an assignment here it means that my additional penalty that I will incur has a row penalty of 0 and column penalty of 2. The row penalty is the difference between that 0 and the next highest element and the column penalty is again the difference between the 0 and the next highest element.

So the sum of the row penalties and column penalties is 2 for this, 0 plus 2; if we take this 0, the row penalty is 0, the column penalty is 1; so we get a 1 here for the penalties.

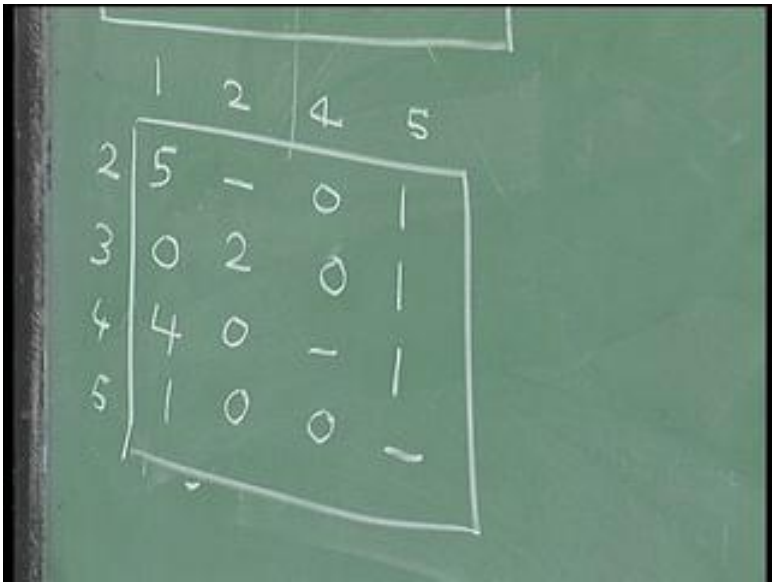
For this we get a 1 plus 0, so we get a penalty of 1 for this we get 1 plus 0, so penalty of 1. Here the penalty is 0, because it is 0 plus 0, here the penalty is 1, here the penalty is 0 and here also the penalty is 0. So this is the way we compute the penalties that we would incur by not assigning to the 0 position. That penalty has two components a row penalty and a column penalty. So the row penalty is a difference between the 0 and the next highest element. If the next highest element happens to be 0, then the penalty is 0, otherwise there is a positive penalty.

For this 0, for example, we have 0 penalty, because row penalty is 0 and column penalty is also 0; whereas for this row penalty is 0, but column penalty is 2. We have now computed the penalties for all the 0s. Now what do these penalties tell us these penalties tell us? If by some for some reason we are not able to make an assignment here, we incur a total penalty of 2. What does that mean? More the penalty is, the more we should actually try and assign there, because the opportunity cost of not assigning is very high. Therefore, we look at that 0, which has the highest penalty and in this example it turns out to be 1, 3.

This 0 which has the highest penalty should be the first one that we would try to assign, because by not assigning here we incur the maximum total penalty of 2. So we choose this and decide to branch on this particular variable. So we branch on the variable X_{13} and create two nodes. This is X_{13} equal to 1 and this is X_{13} equal to 0.

Now we have already computed that if we do not make an assignment here or if this does not form part of the TSP solution, then we will have an increased cost of 2. Therefore the lower bound for this will become 32 plus 2 which is 34. Now, in this case, we have decided to allocate X_{13}, so what we do here is, leave this row and this column and create a four by four matrix.

(Refer Slide Time: 41:30)

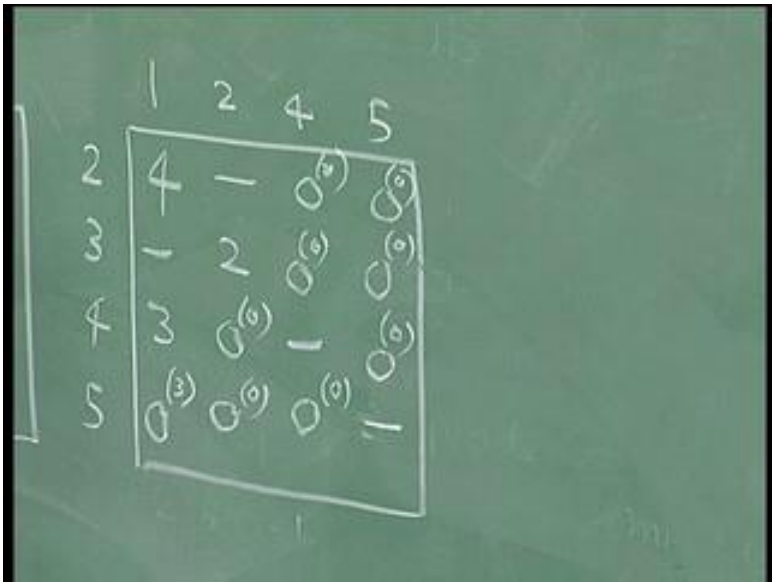


Now, this 4 into 4 matrix will look like this, this will have 2, 3, 4, and 5. Remember this has 1, 2, 3, 4, 5; 1, 2, 3, 4, 5. So this has 2, 3, 4, and 5 and 1, 2, 4, and 5. So you get 5 dash 0 1, 0 2 0 1, 4 0 dash 1, 1 0 0 dash. This is what we have here. Now, having allocated X_{13} to 1, means we should not have X_{31} in the solution, otherwise it will become a subtour. We will not have this X_{31}, so we will put a dash here, so that we do not have X_{31} in the solution, since we have already have X_{13} in the solution.

Now in this reduced matrix, we now have to check whether every row and every column has an assignable 0. To do that, we realize that all the four rows have 0s, but, in this column and this column, we do not have 0s; so subtract the column minimum. So this will now change to: 2, 3, 4, 5; 1, 2, 4, 5; so this becomes 4 dash 3 0, dash 2 0 0, 0 0 dash 0, 0 0 0 dash. Now, we subtracted 1 from here and we subtracted 1 from here; so effectively we have subtracted 2, therefore the lower bound now increases from 32 to 34.

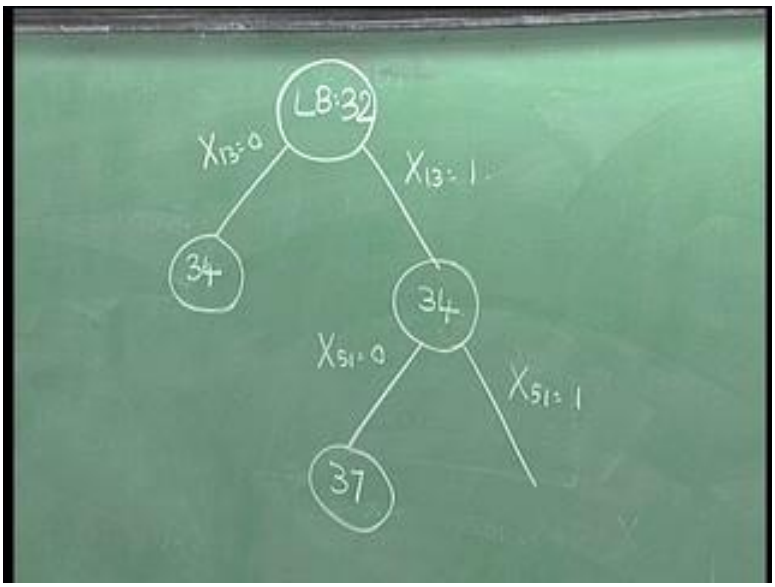
Please note that the computations are different; the motivation to get these numbers are different. It is only incidental that we have got the same number for both of them. Now increase in 2 on the left hand side comes, because of the maximum penalty of 2; increase in 2 on this comes because we have subtracted 1 from here, we subtract 1 from here. These subtractions are motivated by the fact that in order to reach this 1 from 2, 3, 4, or 5; we need to incur a minimum 1, so we make that 0 and add that 1 to the lower bound. So this is how we get both these lower bounds. This is the reduced matrix.

(Refer Slide Time: 44:36)



Again we find the row penalty and column penalty for every 0 in this matrix. For this 0 it is 0 plus 0 is 0; for this 0, it is 0 plus 0 is 0; for this 0, it is 0 plus 0 is 0; for this also it is 0. Now for this one also, it is 0; for this one also, it is 0; for this one, it is 0 plus 3 is 3; for this one, it is 0; and for this one also, it is 0, because for this, you have a 0 here and a 0 here. Now this has the maximum penalty of 3, so this is the next candidate for us to branch. So X_{51} is a next candidate to branch.

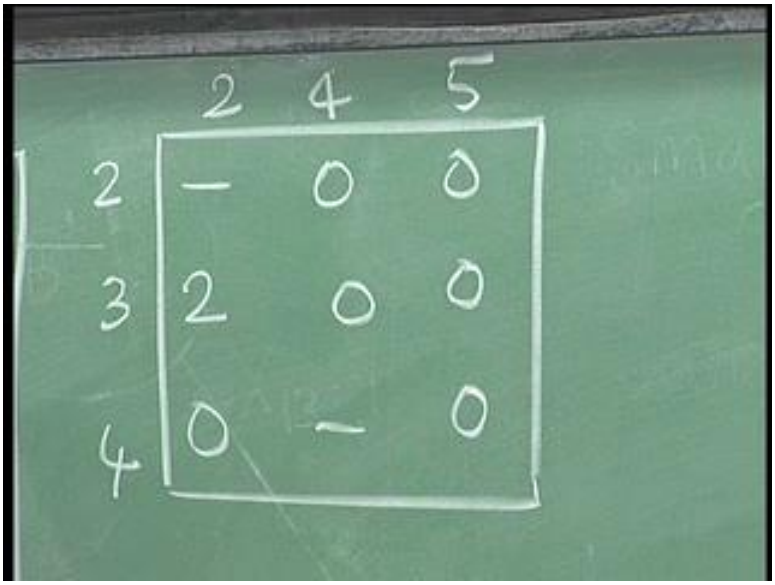
(Refer Slide Time: 45:24)



So we always start from keeping ourselves aligned to the right hand side and we branch. We create two branches here, with X_{51} equal to 1 and X_{51} equal to 0. Now when we have X_{51} to

0, which means we do not have an assignment; which also means, we have computed already that by not having the assignment, we will incur a penalty of 3. Therefore this lower bound becomes 37 from 34, because this is the penalty that we have computed which is the extra cost that we incur by not assigning to this. In order to compute the lower bound on the other one, we have already made an assignment with X_{51}, so we remove this fifth row and the first column and then we create a 3 by 3 matrix with 2, 3, 4 and 2, 4, 5 and the values are dash 0, 0; 2, 0, 0; and 0, dash, 0.

(Refer Slide Time: 46:24)



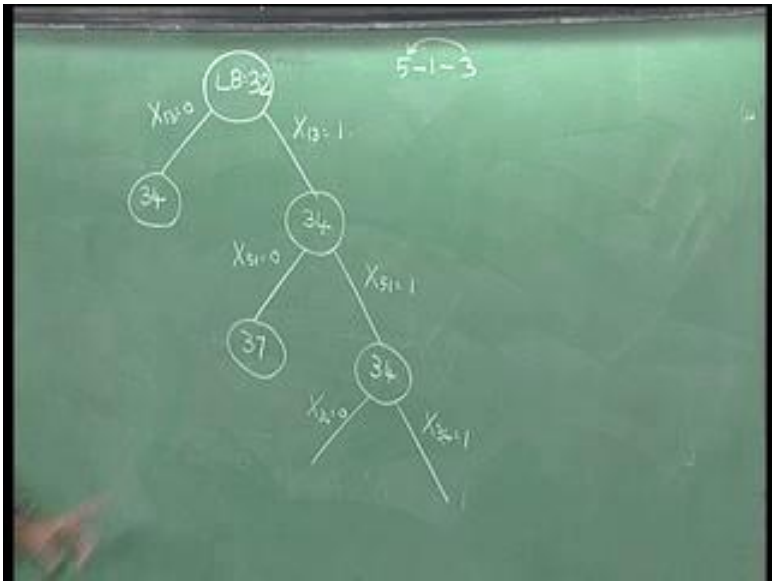
Now again since we have put X_{51} equal to 1, X_{15} has to be 0 but we do not find 15 here now. That means that we need to look at the earlier ones; so we have done 5 to 1, 1 to 3, which means we have moved from 5 to 1, 1 to 3, we should not come from 3 to 5.

I repeat, this represents a solution 5 to 1, 5 to 1, 1 to 3. So you should not have 3 to 5, so C_{35} should be 0, so this becomes dash. So this dash as we said again comes from that part; X_{51} equal to 1; X_{13} equal to 1, would mean 3, 5 should be 0, should not be equal to 1. So for 3, 5 you put a dash. Now after this we see that every row and every column has a 0, so we need not subtract anything from the row minimum and the column minimum and therefore the lower bound stays at 34 for this

Now we again compute the penalties for all of them, so this has a 0 penalty, this has a 0 penalty, 0 plus 0; this has a 0 plus 2, so it has a penalty of 2; this has a 0 plus 2, it has a penalty of 2; now this has a 0 plus 0, so this has a penalty of 0.

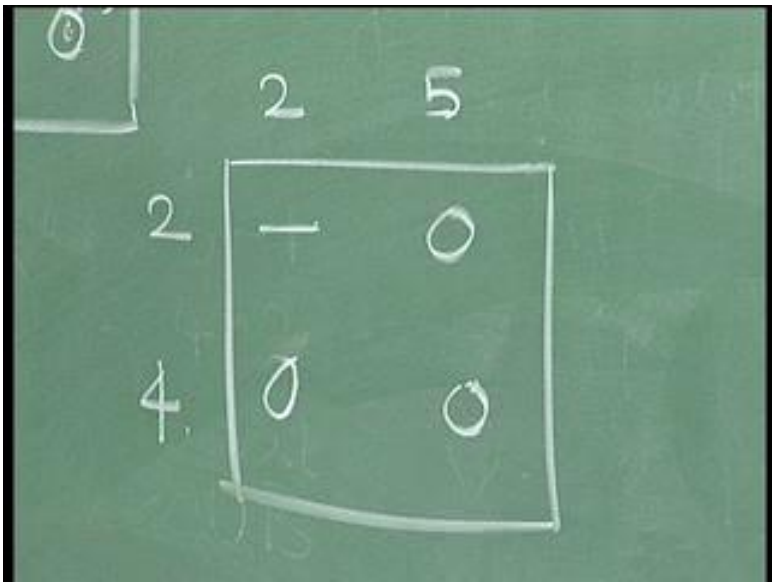
So we could either pick 3, 4 or 4, 2, as the one with maximum penalty both of them have the same penalty of 2.

(Refer Slide Time: 48:23)



So we can pick 3, 4 to begin with and branch on 3, 4; so we branch here on 3, 4; so you get X_{34} equal to 1; this is X_{34} equal to 0. Please remember that this is 3 to 4 node and that we need should not confuse this with the value 34 that is present here. So this is X_{34} equal to 1, this is X_{34} equal to 0. We have already seen that there is a positive penalty of 2 if we do not make an assignment in 3, 4, therefore by not making this assignment; the lower bound now becomes 36 from 34.

(Refer Slide Time: 49:10)



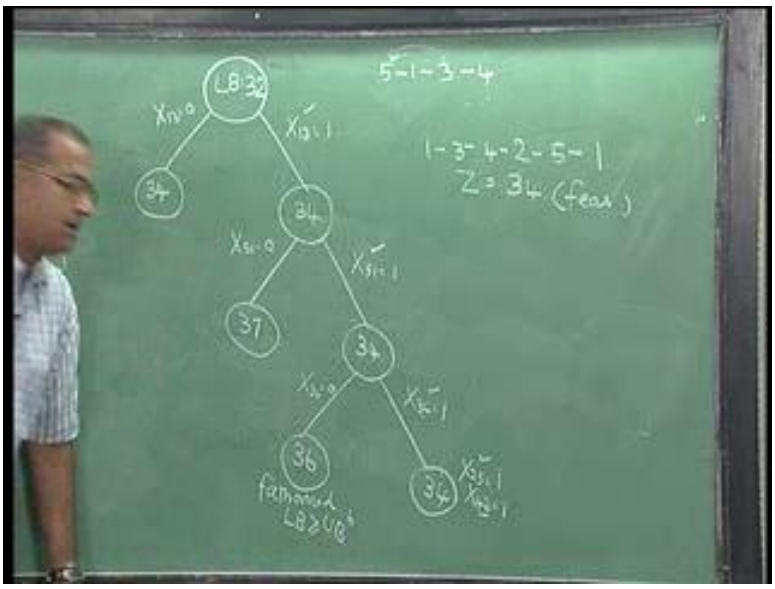
Now once we have made X_{34} equal to 1, so remove the third row fourth column and then create a two by two matrix which has 2, 4 and it has 2, 5. So we have dash, 0; 0, 0 and we made X_{34} equal to 1, which means X_{43} should be 0, but we don't find 4, 3, therefore we need to go back. Now this is 5 to 1; 5 to 1; 1 to 3; 3 to 4, which means 4 to 5 should not exist. So 4 to 5 becomes dash. Now we realize that after making this, every row and every column has an assignable 0, therefore the lower bound does not increase.

So the lower bound stays at 34 again, but we also observe that when we have only two more cities to go, we can always try and get a feasible solution out of it, so these are the only assignments that are possible. This would give us a feasible solution with X_{25} equal to 1 and X_{42} equal to 1. So we now have a solution from this which is X_{13} equal to 1, X_{51} equal to 1, X_{34} equal to 1, X_{25} equal to 1, X_{42} equal to 1.

This gives us a feasible solution 1 3 4 2 5 1; 1 to 3, 3 to 4, 4 to 2, 2 to 5 and 5 to 1. So we get a solution 1, 3, 4, 2, 5, 1, with Z equal to 1 to 3 is 8, 3 to 4 is 16, 4 to 2 is 21, 2 to 5 is 27 and 5 to 1 is 34. So, this gives us a solution with Z equal to 34, which is a feasible solution to the TSP.

Now, we get a feasible solution to the TSP. The moment we get a feasible solution to the TSP, we can do certain things. We already know that this has a lower bound of 34; this has a lower bound of 37; this has a lower bound of 36.

(Refer Slide Time: 52:10)



The moment you have a feasible solution with 34 you fathom this. You do not want to proceed from here, because few moves downwards from this you will get solutions 36 or more. So, this is fathomed based on the principle that the lower bound is greater than or equal to current best upper bound. Right now there is only one upper bound. Every feasible solution is an upper bound, so the upper bound becomes this 34, which we have computed. Now, the current best upper bound is 34; lower bound is greater than or equal to the current best upper bound.

When we move from here, we cannot get anything. This is also fathomed from the idea that lower bound is greater than or equal to current best upper bound being 34. So, we do not proceed from here. This also we can fathom because if we move down we will get either solutions with 34 or more. We already have a solution with 34, so again with the same idea that you fathom this, based on lower bound is greater than or equal to current best upper bound. The equal to comes because even if we proceed from here, we will get either solution with 34 or more. Now we do not have any other node to continue, therefore this algorithm terminates with this solution as an optimal solution. The current best upper bound is now the optimal solution; the algorithm terminates with that.

We already know that Z equal to 34 is the best value that we can get and we have already seen the same solution either in this form or in its mirror image form with Z equal to 34. Now what are the other issues that we will have to look at with respect to this? Now just in case one of the things we always did was, whenever we branched we created two branches and we kind of moved here, we just kept the right thing going and we always continue to evaluate only the right most one till we got a feasible solution.

Now, if it turns out say that one of the left ones is less, say if this were 33 instead of this 34, then what happens? Now if this were 33 then we have to proceed from here. What is this node? This node is X_{13} equal to 0, so we will go back to this, you would create X_{13} equal to 0, and therefore, you would put a dash here for X_{13} equal to 0. Once again do row column subtraction and if we do that or we could go back here, put a dash here. This is equivalent to this; this is equivalent to this 34.

So, we could put a dash here and we proceed. We have realized we have to make one more subtraction so that we get a 0 in every row and every column. When we do that column row column subtraction we would have subtracted two. Therefore if we put a dash here and the

moment we put a dash here, you realize that you do not have to subtract anything from here, but you have to subtract something from here, because this is a dash, 4, dash, 2 and 2; so you would subtract 2 from this.

So the lower bound would now increase from 32 to 34, which we already have here so if it turns out that this lower bound is less; so you will get exactly that lower bound value here, we should not add it to second time. Then from here, we will proceed like we proceeded here, with the only difference that you would have put a dash here and then you will precede. When you proceed from here, you will again create right branches and then keep moving to the right, till you get feasible solutions to the whole thing.

So it does not mean that every time when we reach this, we will get the optimal solution. We need not, but what is guaranteed is, when you reach here you get a feasible solution and many times you get a very good feasible solution right here. So the niceness of this branch and bound is the fact that we keep branching straight away till we get a feasible solution which is given here. If it turns out that we are able the fathom everything, the moment we get the feasible solution, we are very lucky as in this example.

But there could be situations where one of these actually turns out to be strictly lesser than the number that we got here and if so we will have to go back, as I explained here which means something like, this we need to move. Then we have to put a dash here, then again do the column subtraction or row subtraction here; reduce it into a five by five matrix. Do not add what you have subtracted because you have already added it here. And then with the reduced matrix with the dash and making sure that enough row column subtractions are there to get a 0 in every row and every column, we would proceed till we get to the optimal solution.

Now, this is a very famous algorithm. This comes out from a paper by Little and three others Little, Murthy, Swiney and Carol which came in the year 1963. It is a very popular branch and bound to solve the Travelling Salesman Problem. Much later we have had several versions and several better versions of the TSP, but over a period of time this branch and bound algorithm has stayed for over 40 plus years and still seem as a very elegant way to solve the Travelling Salesman Problem. Now this particular thing also gives us a feasible solution, so at some point we may just stop here and use it as a feasible solution.

In the next lecture we will start looking at heuristics to solve the Travelling Salesman Problem.