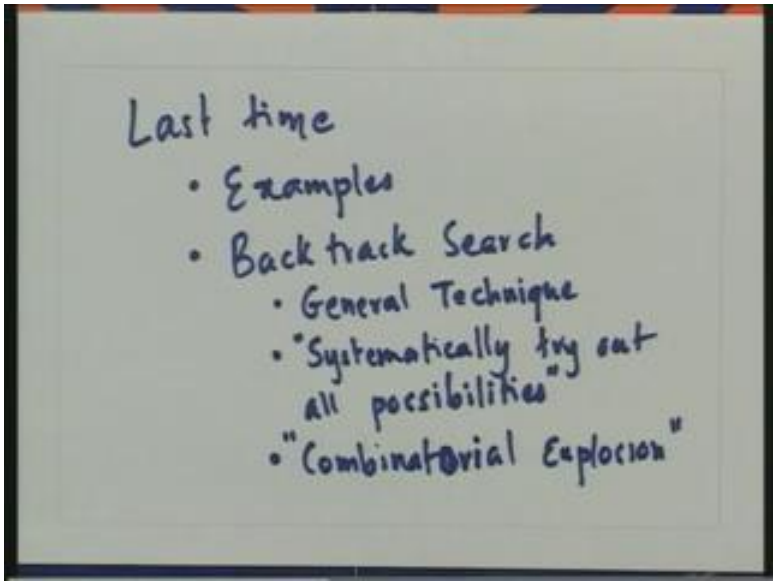
Design and Analysis of Algorithms
Prof. Abhiram Ranade
Department of Computer Science Engineering
Indian Institute of Technology, Bombay

Lecture – 17
Combinatorial Search and Optimization – II

Welcome to the course on Design & Analysis of Algorithms. Our topic today is Combinatorial Optimization and Search. And this is going to the second lecture, on this topic. Let me start by summarizing, what we did last time.

(Refer Slide Time: 01:09)

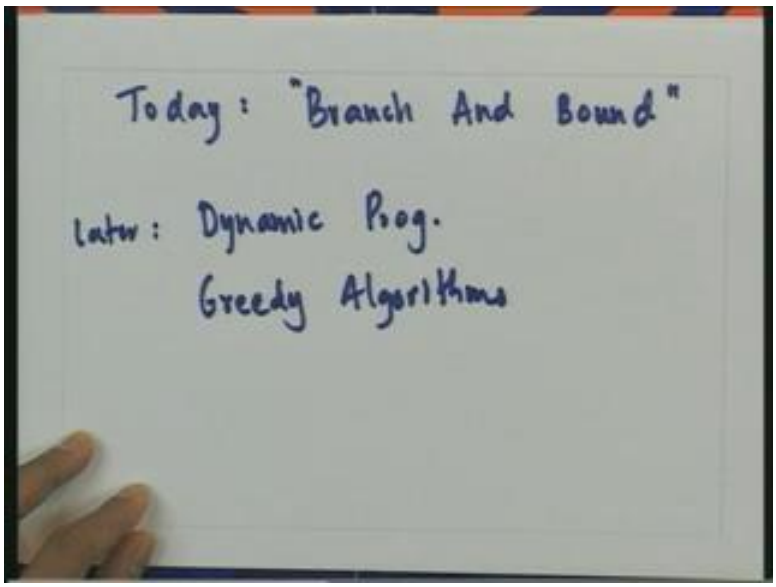


In the last lecture, we gave examples of what we mean by combinatorial search. And then we discussed, a fairly general technique called backtrack search. This is sort of the obvious idea, just executed systematically. So, let me just summarize by saying that it is a general technique. It will work essentially for every combinatorial optimization problem or combinatorial search problem. And the basic idea, over here is to systematically try out all possibilities.

The problem of course, with these techniques was so called combinatorial explosion. As we explained last time, this just means that the number of all possibilities, which we are going to trial is really large. And therefore, this technique typically takes a lot of time. However for many problems, this is the only way to go. Our topic today is to consider

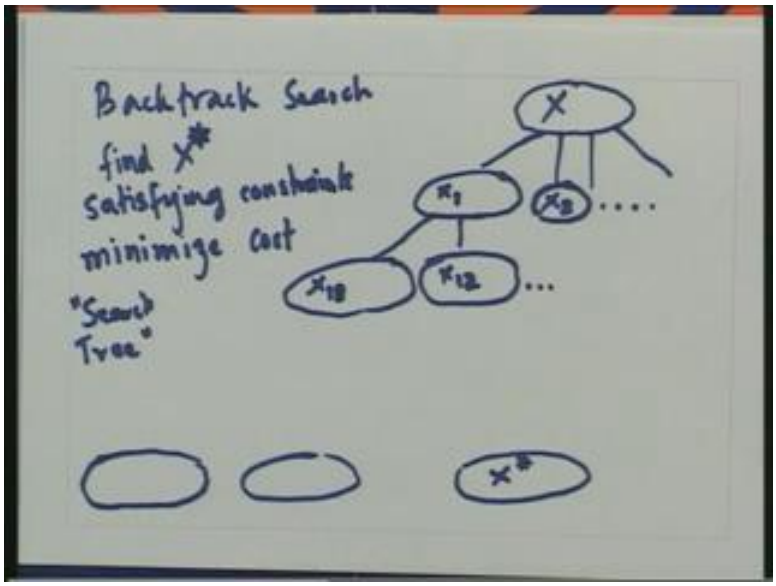
cases, in which we can do better. So, today's topic is going to be how to improve upon backtrack search.

(Refer Slide Time: 02:50)



And specifically, we are going to see a variation called branch and bound. This is going to be the main topic. But, we will eventually also look at other ideas called dynamic programming and also greedy algorithms, which are also used to solve combinatorial optimization problems. But this will come later, all this will come later. And we will deal with dynamic programming, as well as greedy algorithms are quite some length. And today, we will mostly focus on branch and bound. So, the main question is, we know how to do backtrack search and we will review it, in a minute. How do we make it more efficient that remains the main question. So, let me start by reviewing what backtrack search is…

(Refer Slide Time: 03:57)



So, the idea over here is that we are expected to find the combinatorial object. Let us call it some X star, which satisfy certain constraints. So, we know all the constraints and we require that the combinatorial object X star must satisfy all those constraints. And further there is cost function, which is also given for these objects. And we are, we need to minimize the cost. So, there is a set of possibilities for X star and somehow, we have to be sure that we have found the one with the least cost among them.

And furthermore of those, we first once which satisfy the constraints; and from those we find the one which minimize the cost. So, last time we said that a good way of organizing, this search is to start up with an object techniques called X. X is really not an object in the sense that we have defined over here, X is sort of a template. So, X think of X as consisting of slots, which need to be filled.

The first slot of X can be filled in several ways. And corresponding to that will have branches going out of this node. So, the first way of filling the first slot, will give me a partially constructed object, let me call it X 1. The second way of filling, this first slot will give me partially constructed object X 2 and so on. So, the first slot can be filled in many, many ways. And I will get, different partially constructed objects based on; however, choose to fill that slot.

I can keep going in this manner say from X 1 or from any of these. Now, I take any of these partially constructed objects. And from these, I am going to filled the second slot.

So, let me denote this by calling this new object X 1 2. Well it is not a fully constructed object yet X 1 1. But two of its slots have been filled. The first slot has been filled, in the first possible way. The second here, has been filled in the first possible way as well.

But, the second slot could also be filled in another way. And that another way, could be written down in this manner. So, in this manner I keep going and eventually, I will get the complete object constructed over here. Of course, when I get the complete object constructed, I have to stop. So, at this point, when I get to the leaves of this tree I have to stop and at each leaf I am going to make sure that may constraints are satisfy.

And then I will evaluate the cost function, given to me and then from that, I will pick the object with the minimum cost. So, let us say the minimum cost object X star appear somewhere over here and that is how I am going to find it. And that is the way that I am going to written. We said, last time as well that this is a tree, which is often called the search tree, this tree which is the search tree is explored in a depth first manner.

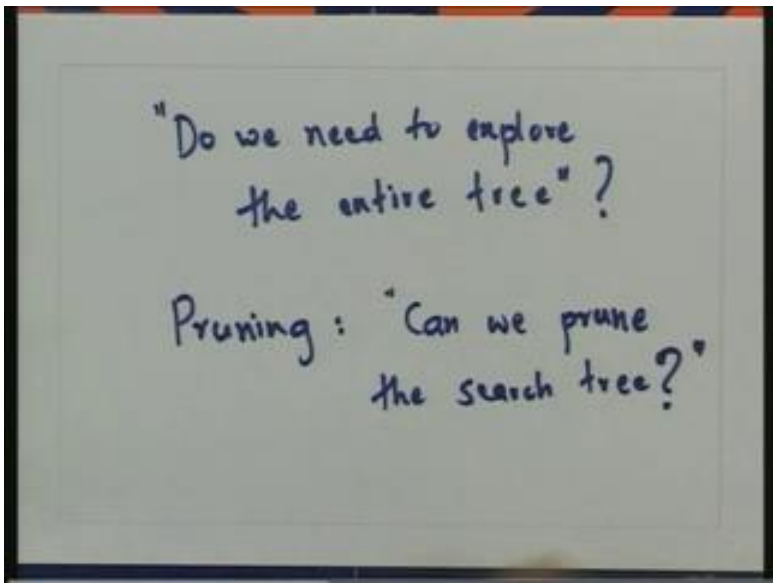
So, from X we generate, we fill the first slot. And we generate X 1, from X 1 we filled the second slot and which of the rate X 1 1 and so on, till we get to the end. At this point, we check the constraints, we evaluate the cost if necessary. And then we go back and then we consider the second way, of the next possible way of filling, the next possible slot or the same slot. And then we go down in the tree, but in the different direction now.

And. So, whenever we come to the leaf, we go back. And in this manner, we go through the entire tree, going back and forth and this back, this going back gives a name backtrack search for this method. So, we will have to explore, the entire tree and in order to be sure that we have found the minimum cost leaf extra. So, this is the organization of backtrack search.

Although, I have drawn the tree in parallel, I have drawn the entire tree for you. Remember that the way a program would execute this procedure. It will start with the top node under it and will go along some particular path, then come back, then go down again. So, at any point and time, it will only be having certain path in this entire search tree. And it will either go down that path or it will roll back that path. And then take a different path that is how the algorithm will execute.

So, this is the search tree that we mentioned and now the natural question is, do we need to explore the entire tree.

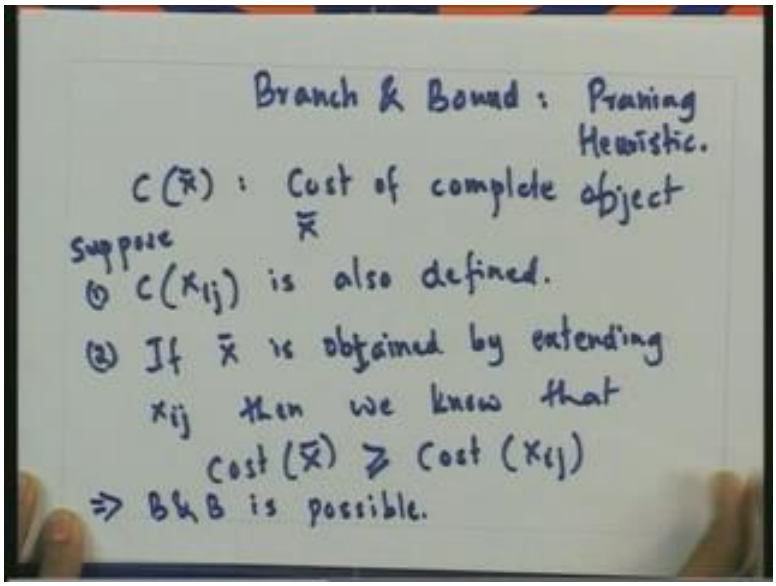
(Refer Slide Time: 09:36)



This is an extremely important question. We said that there is combinatorial explosion and that because the number of leaves is a huge number, the leaf coming back to this picture. At every time as we go down the tree, the numbers of nodes doubles are get multiplied by a large number as we go down. And therefore, the number of leaves is enormous. So, suppose it was possible for as to say, that look anything that is underneath these X 2, really does not need to be explored and without even seeing, anything underneath it, If we could do that, then we will be saving ourselves a lot of work.

And so the idea that we are looking for over here is called pruning. So, another way of expressing the same version is, can we prune the search tree. So, if we could do that, then you reduce our work and then we would have, a more efficient way of searching.

(Refer Slide Time: 11:11)



So, the branch and bound is just one possible heuristic of pruning, is a pruning heuristic. So, let us come back to this picture here again.(Refer Time: 11:28) So, let us say that we have explored this whole thing, we found certain solution X over here and we evaluated its cost and this cost turned out to be some C of X. So, we went down along the left branch in the tree and we found, a certain C of X.

Now, suppose this cost function C can be used to evaluate the cost of X, which is a complete object or let me call this say some X bar, X bar is a completely defined object. But, let us suppose that this cost function C is also, will also be able to tell us the cost of partial object. So, this is an important idea. So, we are given a cost function C, some C of say X bar, which gives the cost of a complete object X bar.

But, suppose C of say some Xij is also defined. Suppose so as we construct our object little by little, we can still associate a cost with it. This may not always be true, but in some cases, it will be and we are talking about those some cases and we are going to give an example in a minute. And furthermore, suppose this extended cost function, now has the following interesting property.

So, in addition to this, this is the first condition that we need, in the second condition is that if say X bar is obtained by extending Xij, then we know that cost of X bar is greater than or equal to cost of Xij. Suppose the cost function satisfy these two properties. What

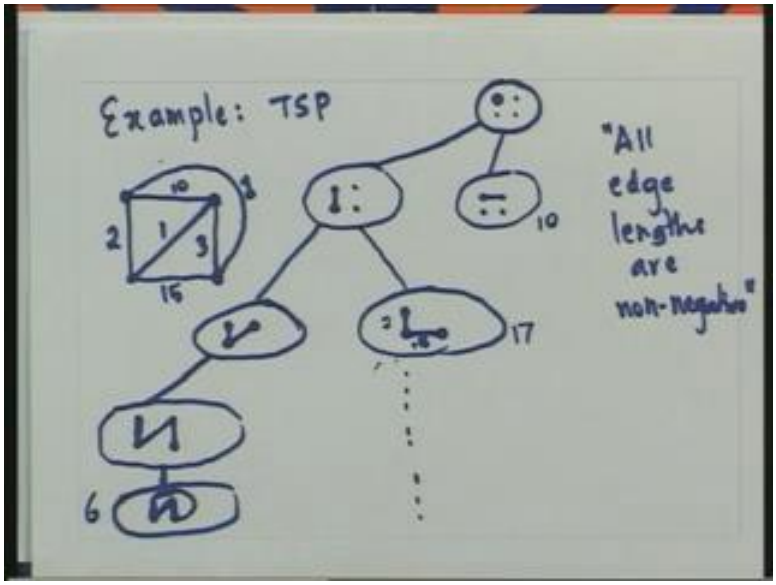
are the two properties that it must be defined, even for partially constructed objects and furthermore, we should only increase as we go towards completely building that object.

Another way of saying the something is that, this is our tree.(Refer Time: 14:26) So, the cost function should be defined at every node in this tree. And furthermore, it has to only increase, actually does not have to increase but it cannot decrease, it should not decrease as we go down from the root towards the leaves. So, if these two properties are met then we can have a branch and boundary, ((Refer Time: 14:46)).

So, let me write that down and then I will explain why. So, this implies that branch and bound is possible, so here is the idea. So, suppose I have found some solution over here, a complete object and its cost.(Refer Time: 15:13) And now I am exploring this tree and I go back and I go back to this point X 2, at this point avail you evaluate the cost of X 2. Now, if the cost of X 2 is itself larger than the cost of the best solution that I have found so far, then something interesting has happen.

I know because of the property, of this cost function that no matter how I complete this object (Refer Time: 15:39). How I extend this object into a full object, the cost is only going to increase. But, this cost itself is bigger than the cost of the best object found so far. And therefore, there is no point in searching further below X 2, this is an important idea. This is the idea of branch and bound. So, let us take an example.

(Refer Slide Time: 16:06)



So, my example is the travelling salesman problem. So, I am going to use a very simple graph, to explain how branch and bound can be used with TSP. So, here is my simple graph. So, it is just a problem on four cities. So, here are the edges and vertices in my graph, this is of course, ridiculously small problem, but it will do for explaining our ideas.

Let me put down the weights along with this. So, may be this edges weight 2, let us say this edges to be 1, let us say this edges be 15, this edge over here has be 10, say this edge has be 3 and this outer edge has be 1. So, this is the graph that we have given and our goal is to find a tour through this graph, a tour is simply a sequence of vertices, which contains all vertices and this is the complete graph. And furthermore, the weight of the edges traverse according to the tour should be as simple as possible.

I am going to first explain, how backtrack search should work on this. And then I will tell you how, a branch and bound heuristic will allow us to prove away a lot of the searching. So, let us start with the empty tour, the empty tour is, so these are my four vertices and let us say this is my starting point, I could have anything as my starting point, does not really matter.

So, this is the starting point. At this starting point, my idea is going to be that I am going to extend the tour, just I have been given a partially constructed object, an object which represents a tour which is not fully constructed here. So, I am going to extend it edge by edge. So, the first edge I can add in several ways.

So, for example, here is one possibility. So, I can start over here and I can go down. So, these are my other two cities. So, I could have gone down over here. Of course, I could have something else what could I have done, I could have say taken the horizontal edge instead. This I will explore little bit later, but right now this is what I am going to explore. So, as we said earlier the algorithm is going to proceed in a depth first manner. So, the algorithm will make a choice that the first edge is the downward edge over here.

Next, it is going to extend the tour starting at this vertex. So, how will you do that, well it will consider all possibilities out of that. So, from here, from here there are two possibilities, now either it can either take the horizontal edge or it can take this diagonal edge going back. So, let us say take this diagonal edge, the horizontal edge will come in somewhere over here.

Then from here what are the possibilities. So, from here the possibilities are that till valid it is gone down and it is gone up. And this point really there is only one thing left to do, it has to go down here again, it has to go down. And then finally, from here there is only one possibility. In fact, so there is no, even here there was one possibility; here as well there is one possibility. So, this is what we had done so far, and then from here it will go back.

So, this is the tour that we would have got, by doing depth first search. So, take the first alternative at each step and we would come to this tour. At this point we would ask, what is the cost of the tour? What is the cost of the tour here, so it is 2 for this 2 plus 1 3 3 plus 3 plus 1. So, the cost of this is 6. So, in backtrack search, we would just record this 6 and we would say this is the best object we have gone, so far.

What we do next then we would go back from here. When we go back here, we say is there another way in which this tour could have extended no. There is no other way because from here, we have to go back to the starting point. So, there is no other way. So, then we go back over here again. So, at this point again we need to ask is there another way of, in which this tour could have been extended.

Well even here, there is no other way because we cannot go back to town which we have already seen, because we have to go through all the towns first. So, the only way could have been to go down, which is what we did over here. So, even here there was no other choice. So, we come back over here and at this point, we could ask well instead of going in this direction could be go straight. So, yes we could go straight. In fact, and therefore, we would go and we would explore this part of the search page

So, from here, instead of going back diagonally we could go down horizontally. Now, from this point onwards backtrack search would go down and search the rest of the tree ((Refer Time: 21:43)). But, here is how branch and bound would make a different decision. Now, branch and bound could say, let us evaluate the cost of the tour that we have constructed so far.

So, what is the cost of this tour, the cost of these edges is 2 the cost of this edge is 15. So, the total cost of this partial object that we have constructed is 17. We know that as we add more and more edges into this, the edge length that the total distance covered is only

going to increase and this is based on the important assumption, which is that all edge lengths are positive.

Let me write the term because that is really an important assumption, all edge lengths are non-negative. If this assumption when not true, there we would not have a branch and bound algorithm at least not the way, I have described so far. But, this is the very natural assumption and so we can conclude at this point that. In fact, any possible way in which this tour is to be extended, does not need to be considered, because its cost is going to be bigger than 17.

And therefore, it cannot beat this best cost tour that we have found so far. So, branch and bound would not do, any of the work of exploring anything below this. So, at this point branch and bound will say, oh I do not need to do anything and I would return back. And it would keep the 6 as the only tour that we have found so far. And that is the best and it would know that with assurance ((Refer Time: 23:34)).

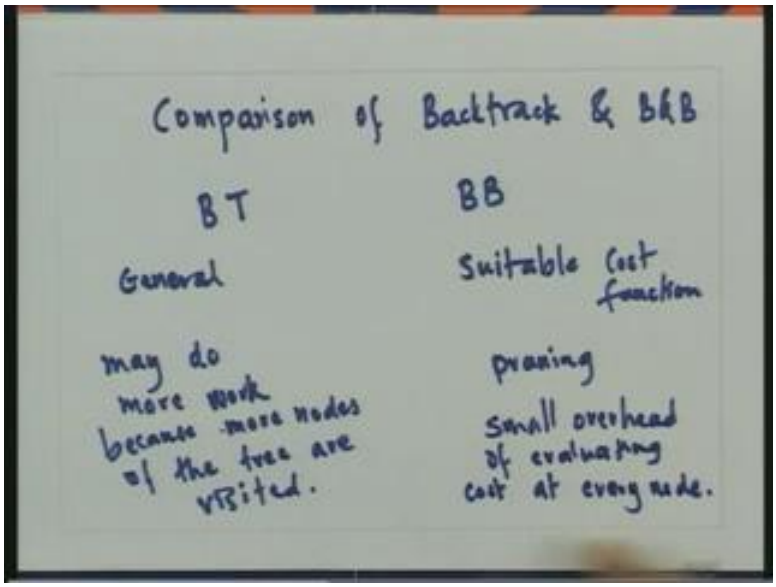
We have essentially proved. In fact, that noting underneath over here needs to be explored. What would happen next, we come back over here, ((Refer Time: 23:44)). And then we say, instead of exploring in this manner, instead of taking the first edge itself going downwards, let us take it some other way. So, what is another way, it is this way, let us take the horizontal edge. Again branch and bound will evaluate, what is the cost of this partial tour that we have constructed.

What is the cost over here, the cost of this partial tour is10, because this horizontal edge has cost 10. Again, without exploring things below this vertex, we can safely conclude that this is large enough, that we do not need to explore anything underneath it. Because whatever comes underneath is, is going to be obtained by extending this tour, which already has cost 10.

And therefore, we can say forget it, the cost of anything below this has to be bigger than at least, it has to be at least 10, but we already found that tour of cost 6. And therefore, we can return back over here and then we can take the next stage and so on. So, in any case in backtrack search we would have explored everything underneath this, in backtrack search we would have also explored everything underneath this.

In branch and bound, we come to this vertex and we know we do not need to do this work. Also we come to this vertex and we do not need to do this work. And therefore, we have saved on the total amount of work that we do. So, let me just summarize this.

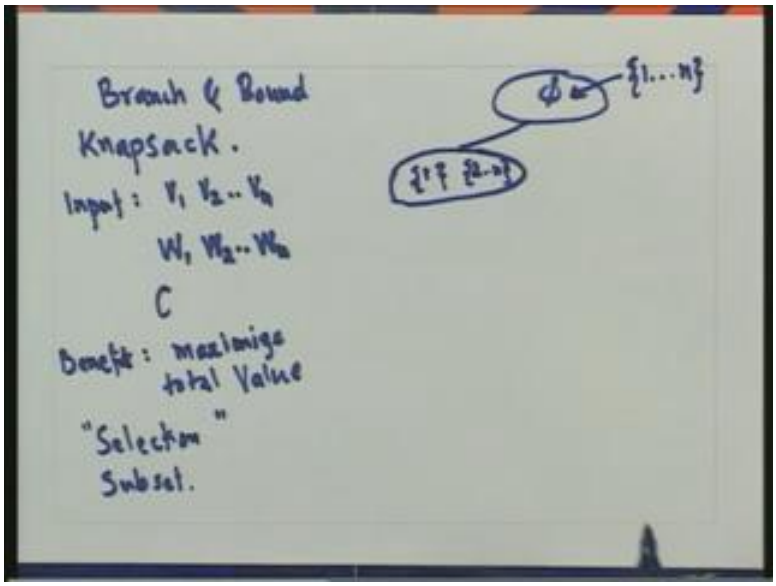
(Refer Slide Time: 25:16)



So, let me just write this as a comparison of backtrack and branch and bound. So, this is backtrack and this is branch and bound. So, branch and bound backtrack is a general method, branch and bound needs suitable cost function. And there might be some amount of cleverness that might be needed, in defining this cost function. This may do more work, because more nodes of the tree are visited. In this, there maybe pruning and there may be fewer notes of the tree, which might get explored in this.

But there is this small overhead of evaluating cost at every node. In general branch and bound, if you have reasonable cost function will work substantially better than backtrack. And I might go as far as saying, that whenever you can find a reasonable cost function, which has the property that I just mention. You would invariably use it, because the gain from the pruning that you get, will typically be much more than the over head of maintaining, the cost function is concerned. So, let us take one more example, before we go on to something else.

(Refer Slide Time: 27:26)



So, we are going to look at branch and bound, but we are going to look at branch and bound this time, for the knapsack problem. We introduce this problem last time, but let me defined it again. So, the input r is the vector is r two vectors, say V 1 2 then V n. So, we have been given n numbers, Vi represents the value of the ith object. And we are also given numbers W 1 W 2 till W n where, Wi represents the width of the ith object. And we are given one more input, which I will call C, which is the capacity.

So, we have been given think of it, as in this manner. So, we have a back of capacity C where, C is also measured as a weight. So, say we have been given a bag, which can carry at most 50 kg. In front of us r n object, we know the weight of each object as well as we know the value of each object. Our goal is to select objects, such that we do not overflow our bag. So, we do not put too much weight in our bag, which might break it. But, subject to this constraint we should put in as much value as possible, that is the goal.

So, I want to explain how they will do branch and bound, how we can implement the branch and boundary heuristics for this. But before that I would like to start with the backtrack search, how backtrack search will work on this. And like the last problem where, there was a natural cost, the natural objective function over here is a different function.

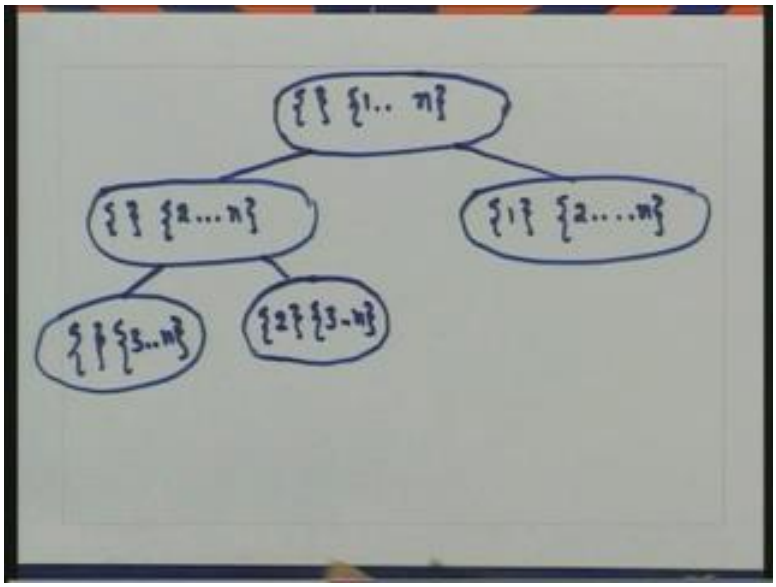
So, the natural function over here is a benefit function. So, our goal is naturally expressed as maximize total value. So, this is; however, object; however, object is

naturally expressed we want to pick up objects, such that we get maximum value, subject to the constraints that our weight, the total weight that we pick up is at most C. So, how will backtrack search work on this, well we need to define the nation of, we need to develop this idea that when we construct solutions, candidate solutions for this.

We have to, can we do that as step by step process in which we start of within empty candidate object and we extended little by little. So, that eventually we have a complete solution, so our nation. So, what we are constructing over here is a selection, selection of objects, which is essentially a subset. So, clearly we should think of this as, we start of by looking at the empty subset.

Then the first decision point that we take and that point we need to, we simply decide, do we take the first object or do we not take the first object. Let us make a firm decision. So, either we take the first object or we reject the first object. So, say let us say we decide to take the first object, then the subset that we have selected so far is consist of 1 and the subset that we have not selected so far consist of 2 through n. So, let me here there is an additional number, which is 1 to n. So, the pointers that our search let me just write this separate, this is beginning fill up to small.

(Refer Slide Time: 32:28)



So, initially our search can be characterized by writing down, what objects we have selected and what are the objects, which are yet to be selected. So, initially we have selected no object what so ever. This is the set of object, about which we have not made

a decision. So, this is the starting point, now the idea is that we are going to make decisions about objects. We will either decide to take an object or either will reject an object.

In the first decision point, we will make the decision about the first object. What could the decision be, well there are into two choices, we will either pick that object or not pick that object. So, if we decide to not pick that object we will get this state. So, we will not have picked any object whatsoever. But the set of objects about which we have to yet to make a decision or going to be 2 to n this time. So, here we have said that we are definitely, not interested in the first object.

Why are we not interested, well there is no real reason for it, this is just one of the possibilities that we are considering. Remember the back track search involves, exploring all possible decisions. So, this is just an exploration of this possibility in which, we do not take the first object, the other possibility is that we in fact, include the first object. So, here will have 1 and those objects about which we have yet to make a decision, will appear over here.

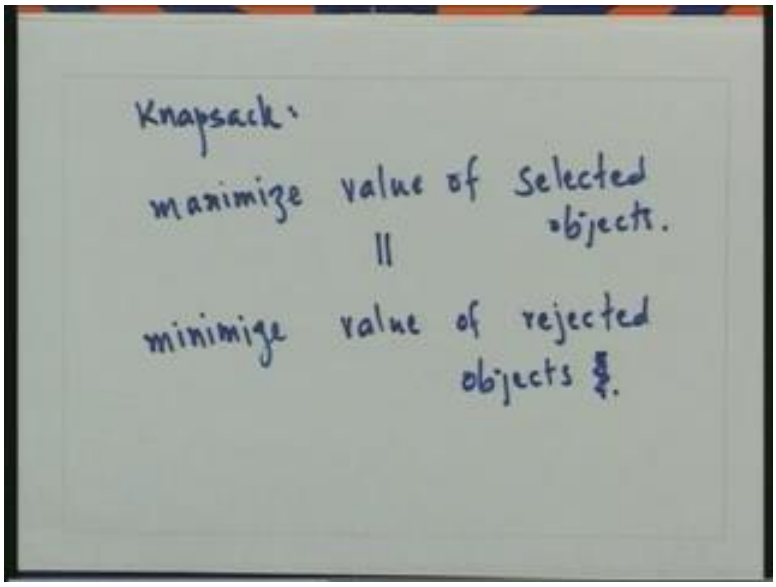
Of course, we will not be building this set immediately, we will first just make a decision about this and will come to this point. Then what will be do, then will we will make a decision about the second object. Again there are two possibilities and therefore, we could get something like this, again may be we decide no, we are not interested in the second object either. So, we will get 3 to n over here or we could decide that we are, in fact interested in the second object this time. So, we will get something like 3 and 3 to m and similarly over here.

So, in this manner, if we proceed we will get to two to the leaves, which will correspond to all possible ways of taking subsets of n objects. And then at that point, we could evaluate our benefit function and then we would have to keep track up of the best possible benefit function. At the best possible completely generated subset with what its benefit is.

Going back to our definition; however, we said that in order to apply a branch and bound, we need to write down a proper cost function. ((Refer Time: 35:54)) And further more the cost function must have this property, of course we could do the same reasoning with the benefit function as well that is indeed possible.

But I am going to take slightly different rule, I am going to keep our definition of branch and bound the same, that is we will vary about cost function rather than benefits function. And now I am going to express the knapsack problem, in terms of a benefit functional rather than a cost function.

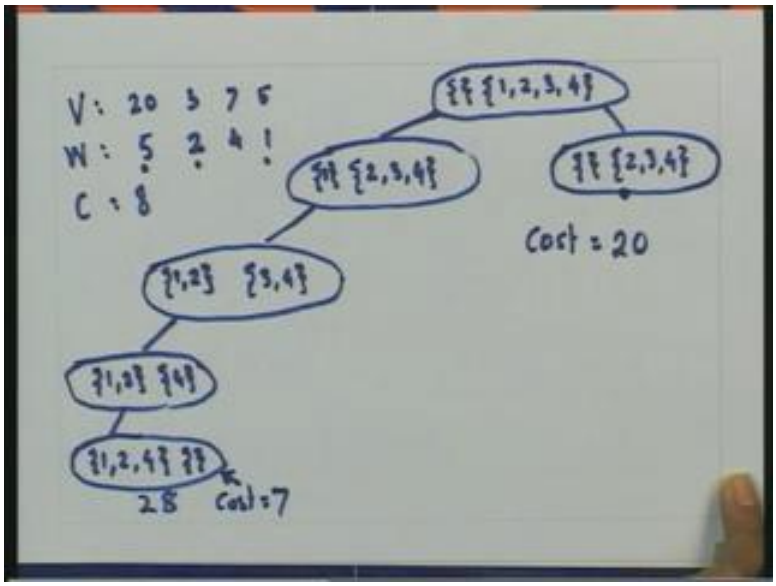
(Refer Slide Time: 36:26)



So, our original function is maximize value of selected objects is there a natural cost function, which we can minimize. I could simply do the negative of this, but that negative is not really very interesting, it does not really give us any insight into the problem. Instead of that, why not ask for minimizing the value of rejected objects. I claim that these two are identical.

So, if you give you a subset, in which the value of the selected objects is as large as possible, I claim that the value of the rejected objects must also be as small as possible. This is simply waste of the observation that the total value is fixed and therefore, if you select large value then you are rejecting the small value. So, this is the key to applying branch and bound to this problem. So, now we are going to think of this knapsack problem, not in terms of maximizing the value of the selected objects, but in terms of minimizing the values of rejected objects. In fact, let us take a numerical example, which will make this idea completely clear.

(Refer Slide Time: 38:13)



So, our numerical example is that our array, this is our array V, so V consists of value 20 3 7 and 5 let us say and W consists of say 5 2 4 and 1. So, the third object has value 7 and weight 4 and let us says our capacity is equal to 8. What will be the branch and bound tree for this looks like. So, originally we will start with no selection. So, the selected set is this, the set of out which we are to make a decision consists of all four objects 1 2 3 4.

Then we make a decision about the first object. So, let us say that is n says that the first object is not to be taken. So, now we still have to make a decision about 2 3 4, over here let us say we do make. In fact, let me change my idea little bit, let us say here we make the decision that the first object is to be taken. So, here the decision could be that the first object is not taken and 2 3 4 we have again really made up our mind yet.

So, if we take the first object, we will have used up V 5, so now we need to make a decision about the next object. So, let us say all the left going paths are the once in which we are greedy, we are keep taking those objects. So, what would happen over here, so we will take the second object as well. So, our bag would contain or our knapsack would contain both 1 and 2 and we will not yet have made a decision about 3 and 4.

And then at this point, we could either decide to take third object or we could check the constraints, remember do not have to be always we checked at the end, we could make the check earlier as well. So, if we decide to check the constraints earlier, if we have

already taken the first two objects then taking in the third object would make the weight be 11, which is bigger than 8. And therefore, there is no question of taking in the third object.

So, we could say we could definitely go down to 1 2 and only 4 over here. So, we have rejected object three. So, from this, we will go down to leaf which is 1 2 and 4, this is an acceptable leaf because 1 2 and 4 are these objects which have a weights 5 2 and 1, which adapt to eight, server capacity is not being violated. And at this point, I am putting an empty set over here, indicating that all our decisions have been we made, we have made decisions about all the object

So, we have come to leaf and we have found the solution and what its value, well this is an object which we took, this is an object that we took and this is an object that we took. So, the value is 20 plus 3 plus 5, which is 28. So, here found a leaf, we have found a solution with value 28. So, now we can go back and we could explore this, I would like you to focus your attention on what happens, when the search reach us this point over here.

Well this is the benefit, but we let me just remind you that we said that we are not going to worry about the benefit, we are going to worry about the cost. So, what is the cost over here. Remember that the cost is nothing, but the value of the object, which were rejected. So, what is the value of the object which is rejected, well the object which was rejected is the third object and so the cost over here is equal to 7, this is the benefit, which is interesting of course, because in the end we want to report the benefit.

But the cost of this solution is 7, in the 7 that we defined, the cost as the value of the object that was rejected. So, if we want to execute branch and bound on this, what is going to happen. We are going to evaluate the cost at each particular point, in this whole exploration. So, when we come to this point, we will also be evaluating the cost. So, now I would like you to tell me, what the cost of this partial solution is.

At this point, the object that we have firmly rejected is the first object, because that is the decision that we made, so what is the cost of this. So, the cost of this is going to be the cost of the object which we have definitely rejected, all objects which we have definitely rejected, so far. We may reject further objects in the future, but that we are not going to

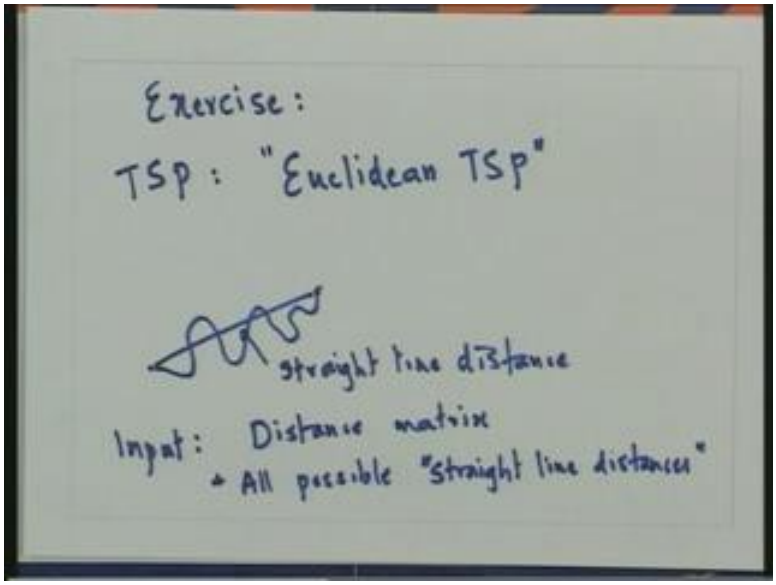
cover. So, the cost of the object that we have definitely rejected, so far is 20. So, now we can conclude that no further exploration underneath this, are really interesting.

Because as we go down, we will only reject more objects. We will include more objects as well, but remember that right now we are focusing on the object which will reject. So, in that sense if we think of the cost, the cost of this is only going to increase if at all, it is not going to decrease. And therefore, whatever leaf we reach underneath this is going to have cost at least 20.

If it cost at least 20, then what is the maximum benefit that we can; however, here well the total value is going to be 20 plus 3 plus 7 plus 5 which is 35. So, the total benefit, the total value is 35, we have already rejected an object of cost 20. So, when we come down to over here, our benefit is going to be at most 15. The interesting thing is that we know that even without seeing all possible subsets. And that is what makes the method powerful.

So, at this point itself, we can reject the entire sub tree and we can say forgive it, go back and this has to be the best cost subset. Whereas, backtrack search could, in fact has search everything. Because it is not keeping track of what is the best and it is not putting, the bound on how much, how good as a solution we can get. So, again even in this case branch and bound will work quite well.

(Refer Slide Time: 45:41)



I want to give an exercise. So, basically if you want to use the branch and bound heuristic, you need to come up with good ways of constructing these cost functions. So, we already saw two problems over which, we constructed cost functions in one case the cost function was fairly natural. In another case we have to do a little bit, we have to be slightly cleaver in order to construct the cost function.

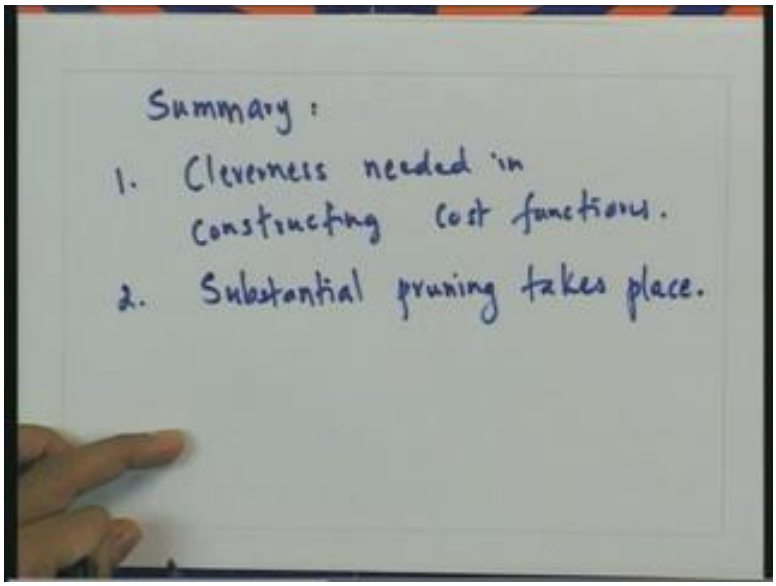
So, let me give you a variation of the TSP problem. Say let us call this the geographical TSP problem, the Euclidean TSP problem. In the Euclidean TSP problem, what we know well think of this as a Euclidean or the geographical TSP problem, which means that, in fact we think of vertices as towns and edges as roads correcting them. So, suppose in addition to knowing the road distances, we also know the latitudes and longitudes or the X Y coordinates of the town is themselves.

So, then there is a notion of the shortest distance or the straight line distance or which is often called the distance as the crow flies. So, this distance is definitely going to be a lower bound, on the length of any road correcting, because a road code wind does much. If it goes through other towns it might even reverse, but the road could wind and therefore, there is going to be this direct straight line distance which is going to prove a lower bound, which is going to give a lower bound on the distance possible.

So, in addition to the input that you already have, so the input in this case is this distance matrix. Plus all possible straight line distances. So, using this I would like you to construct a cost function, which will be better than the cost function which we have constructed earlier. What does better mean, that if I give it partially constructed tour, it should give me a value which is bigger than the value which I had earlier.

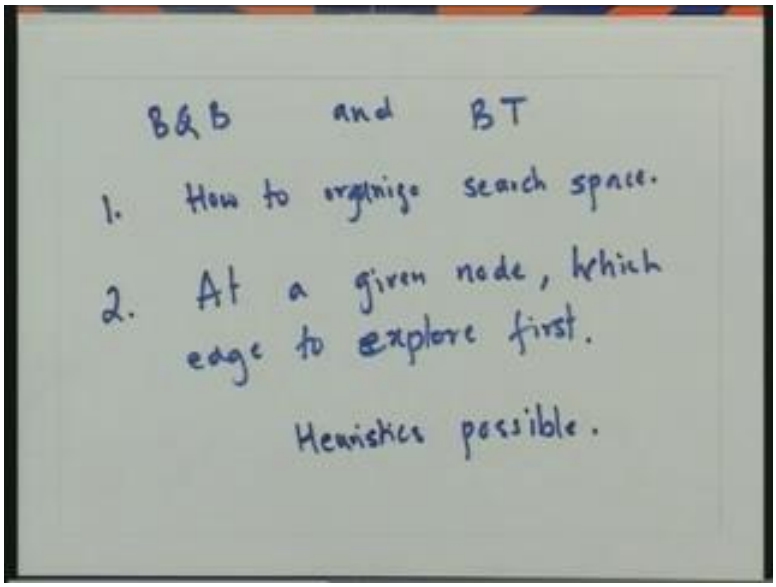
Now, why could that mean. So, that would have to be because, it is somehow has to take into account that I have committed to using these edges. But, I still have to visit all these towns and if I want to visit these towns, I would at least have to have some amount of distance added to the distance that I have calculated so far, so for doing this, the straight line distance information will come ((Refer Time: 49:12)). So, this should be an exercise I will like it to try. So, let me summarize again, just I have summarize this. So, let me write down in the final word.

(Refer Slide Time: 49:28)



So, there is some cleverness needed in constructing cost functions. That is an important idea, but once the cost functions are constructed, usually substantial pruning happens. So, let me look ahead a little bit well, actually know let me look back little bit.

(Refer Slide Time: 50:23)



So, there are number of issues, which you have not talked about in both branch and bound and backtrack. So, the first issue that we have not talked about is, how to organize the search space, what I mean by this is let me come back to this example, ((Refer Time:

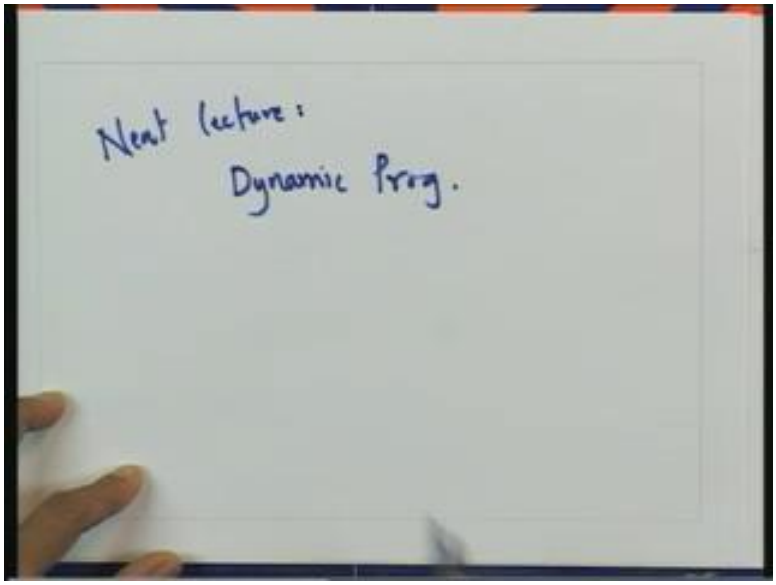
50:51)). Here we said that will look at the first object first, that is what necessary, we could have used some other heuristic of deciding which object to look at first.

So, this initial that ((Refer Time: 51:05)) we do might have great pay offs in the end. So, maybe we could say that we want to look at the object, which has the highest value first or which has the highest value to weight ratio first. So, at a given node, which edge to select first to explore first. Again coming back to this ((Refer Time: 51:46)) at this given node, here we select at the edge in which we included the object. Here we did not include the object.

In this case, we explore this edge first at not this edge and you saw that was the more interesting thing to do because, we got a good object with good cost. And as a result of which could prove in this. If we have gone in different direction, then maybe we did not have what this pruning effect. So, clearly how we which has to explore first is an interesting decision. So, there are additional heuristics possible for all these questions, both these questions.

There are lots of works and there are internet articles as well, these days which will tell you about these things. And which will tell you, how the different heuristics works for different kinds of problems. For our next lecture, we will take a complete different viewpoint.

(Refer Slide Time: 53:12)



And we look at dynamic programming, which is an even more interesting way of doing combinatorial search and optimization and this stop here.