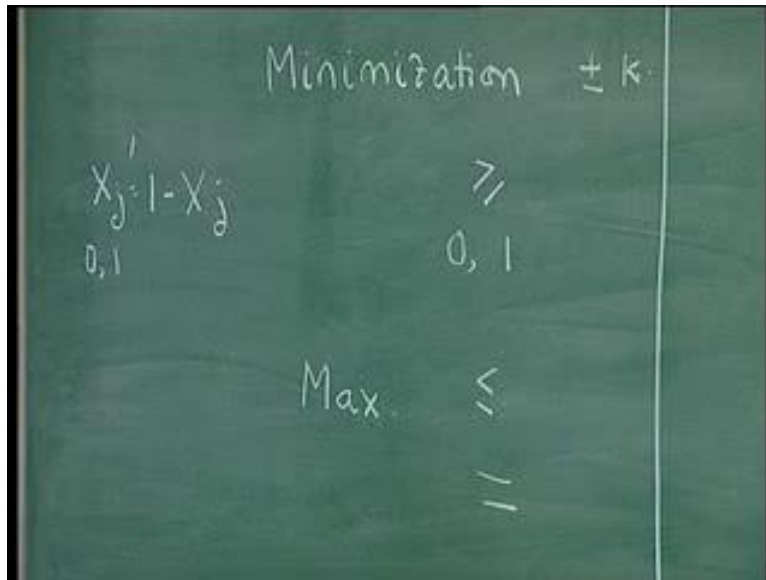


Advanced Operations Research
Prof. G. Srinivasan
Department of Management Studies
Indian Institute of Technology, Madras

Lecture No. - 15
Branch and Bound Algorithm for Integer Programming

Today we continue our discussion on zero-one problems. We try to look at a few things.

(Refer Slide Time: 00:23)



We said that our standard problem is a minimization problem with all constraints of the greater than or equal to type and of course all variables are 0 and 1. More importantly, the minimization objective function is such that the objective function coefficients are all non-negative. So, if we have a less than or equal to constraint then we multiply the constraint with a minus 1 and convert it to a greater than or equal to which is a known way of doing it and we do not have unduly worry about the right-hand side values. The standard problem does not assume that the right-hand side values are non-negative; right-hand side values can still be negative, so it does not matter.

Step 1: If you have a less than or equal to inequality, multiply with a minus 1 to get a greater than or equal to inequality here. If you have a maximization objective, once again multiply

the objective function with a minus 1 to make it a minimization problem. So, these two things are possible.

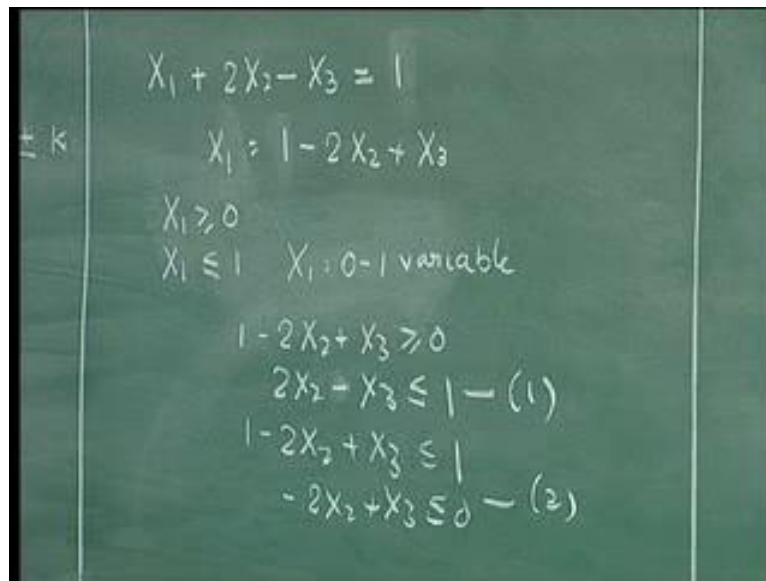
Now, how do we ensure that every term in the objective function has a non-negative coefficient?

If there is a variable X_j with a negative coefficient in the objective function, replace it with an X_j dash equal to $1 - X_j$ so that it effectively takes a non-negative coefficient in the case of a positive coefficient and because X_j is a 0, 1 variable X_j dash is $1 - X_j$, X_j dash is also a 0, 1 variable. So, this way, we will be able to convert every given problem into the standard form.

Standard form is the form where you have a minimization objective function, all constraints of the greater than or equal to type and all variables in the objective function having a non-negative value. There can be a constant. As a result of this conversion we might have at the end a plus or minus constant and that constant is all right. You can always take a constant out of the objective function, solve it and then add or subtract that constant depending on what the sign of the constant is. The only thing that we have not looked at is what happens when I have an equation?

The variable definition is only one way which is 0, 1 so we do not worry about initialization with respect to the nature of the variables. We are only worried about initialization with respect to nature of the constraints and the objective function. We have seen objective function, constraints except the equation.

(Refer Slide Time: 03:32)


$$\begin{aligned} X_1 + 2X_2 - X_3 &= 1 \\ \Leftrightarrow X_1 &= 1 - 2X_2 + X_3 \\ X_1 &\geq 0 \\ X_1 &\leq 1 \quad X_1: 0-1 \text{ variable} \\ 1 - 2X_2 + X_3 &\geq 0 \\ 2X_2 - X_3 &\leq 1 - (1) \\ 1 - 2X_2 + X_3 &\leq 1 \\ -2X_2 + X_3 &\leq 0 - (2) \end{aligned}$$

Suppose we have an equation X_1 plus $2X_2$ minus X_3 equal to 6. Now, what we do is, we write X_1 equal to 6 minus $2X_2$ plus X_3 . I was just giving you a value; if you do not like 6 you could keep it as 1. So, you have 1 minus $2X_2$ plus X_3 and then this X_1 being a 0 or 1 would simply give you two constraints with an either or relationship. So, when X_1 is 0, this becomes $2X_2$ minus X_3 equal to 1. When X_1 is 1, it becomes $2X_2$ minus X_3 equal to 0. That's what happens here when you do this.

What we do is this is X_1 equal to this, X_1 being a 0, 1 variable is now written as X_1 greater than or equal to 0, X_1 less than or equal to 1. X_1 defined as a 0, 1 variable. Now X_1 greater than or equal to 0 would give you 1 minus $2X_2$ plus X_3 greater than or equal to 0 so $2X_2$ plus X_3 less than or equal to 1 and 1 minus $2X_2$ plus X_3 less than or equal to 1 would give us minus $2X_2$ plus X_3 less than or equal to 0.

Every equation now becomes two constraints. So, we end up adding one more constraint for every equation, the variable retains, but we have eliminated one variable from the analysis. So the variable X_1 does not exist in anymore.

Typically if you have to solve zero-one problem with many equations you would get into this kind of a situation. Now if you say five equations then we end up writing five constraints for the greater than or equal to five constraints. For the less than or equal to 1 and the greater than or equal to 0. One of them you can simply add.

Particularly the less than or equal to type constraints you can add and convert five of them into a single constraint depending on the problem. But as a general rule if you have an equation, then you end up creating two constraints for you eliminate one variable from the equation but then you have to create one additional constraint so there will be two constraints.

To that extent the problem becomes little more involved and cumbersome whenever you are having equations because the time taken obviously depends on the number of constraints and with equations the number of constraints increases considerably.

(Refer Slide Time: 07:43)

$$\text{Minimize } Z: \underbrace{X_2 X_3}_{y_1} + X_2 + X_3$$

$$X_2 + X_3 - y_1 = 1$$

$$-X_2 - X_3 + 2y_1 \leq 0$$

$$X_2, X_3, y_1 = 0-1$$

The last part that we have to see in this is what happens when we have variables of this form. Suppose, I have this minimize Z equal to $X_2 X_3$ plus X_2 plus X_3 or to generalize it we put some X_2 square plus X_3 cube where all X are 0,1 variables. We know that this X_2 square and X_3 cube will become simply X_2 and X_3 because of the 0,1 nature of the variables. Only thing we need to do is to model this $X_2 X_3$ and just write down the equations for you.

(Refer Slide Time 09:07)

$$\begin{aligned} \text{Minimize } Z &= X_2 X_3 + X_2 + X_3 \\ x_2 + x_3 - y_1 &\leq 1 \\ -x_2 - x_3 + 2y_1 &\leq 0 \\ x_2, x_3, y_1 &= 0-1 \end{aligned}$$

You can take any variable you can choose it or you it is not necessary that you should eliminate only X_1 you could eliminate X_2 or X_3 equations will change accordingly. So you will have Y you will replace $X_2 X_3$ by Y and then say that X_2 plus X_3 minus y_1 less than or equal to 1. minus X_2 minus X_3 plus $2y_1$ less than or equal to 0 $X_2 X_3 y_1$ equal to 0, 1, $X_2 X_3$ will be replaced by y_1 and then you will have this.

For every product term you will have two constraints which are coming in and an additional variable that also come in. There is another way of representing it also by which you do not define y_1 as a 0, 1 variable you define y_1 as continuous variable and add three constraints instead of two. Obviously this is a more desirable one than the other one simply because in the other one you add three constraints so you do not increase the size of the problem. Secondly, you also bring in a y which is not a 0, 1 variable, it is a continuous variable, so the problem loses all its 0, 1 structure and it will be difficult to solve that problem using the implicit enumeration algorithm.

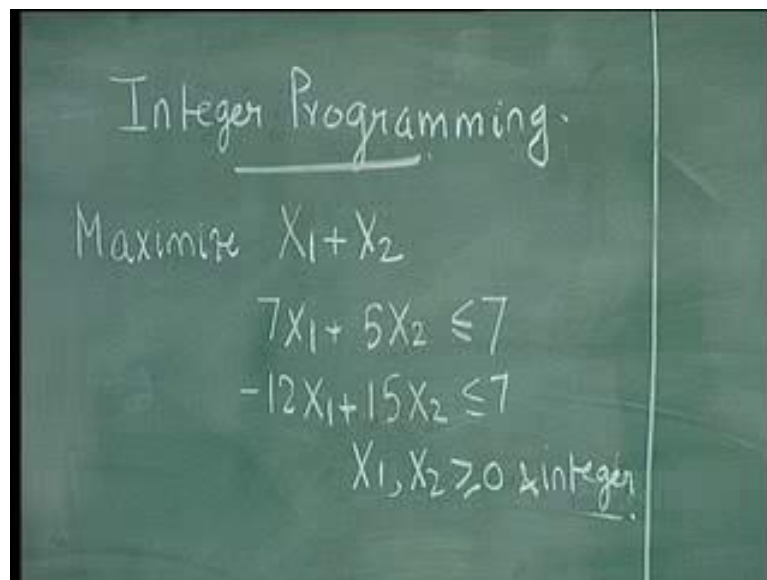
We will not look at that but we would rather keep this as the way to convert a product form into a single variable. Nonlinear problems become linear zero-one problems by this method. Any higher power automatically becomes the same variable only when you have product forms you will get this. When you have three terms appearing you can write a corresponding

the spirit is y_1 will take a value 1 only when X_2 and X_3 take 1. If either of them takes 0 then y_1 will be forced to 0 that is the underlying assumption which is solved by both these.

If we have three terms product terms coming in a single variable then you will have one more additional constraint that would come in and that has to be written accordingly. So, this kind of brings us to the end of the discussion on zero-one problems we have just seen one algorithm of course there are so many other refinements the algorithm that we saw is a very old 1959 algorithm so in the last forty-five years you could imagine much more development in this field.

Nevertheless, it gives us a feel of what implicit enumeration is and how easy it becomes, particularly, to write a computer program or a code which would solve zero-one problems. We will then move into the next important topic which is to solve problems using a general integer programming problem.

(Refer Slide Time: 11:30)



The image shows a chalkboard with the following handwritten text:

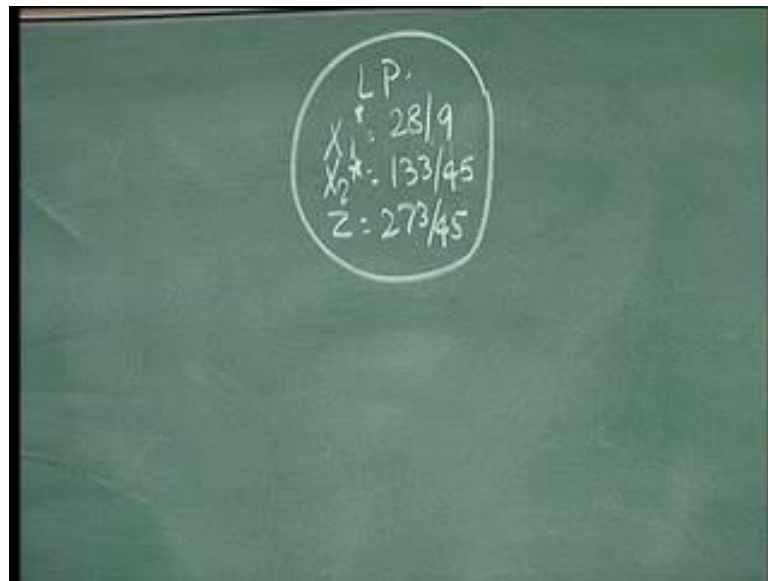
$$\begin{aligned} &\text{Integer Programming.} \\ &\text{Maximize } X_1 + X_2 \\ &7X_1 + 5X_2 \leq 7 \\ &-12X_1 + 15X_2 \leq 7 \\ &X_1, X_2 \geq 0 \text{ \underline{integer}.} \end{aligned}$$

Now we look at, how we solve a general integer programming problem. We will take a specific example and explain the IP (Integer Programming). We will take this problem as a standard problem and try to solve this. Now the first thing that we do is in all integer programming problems, right now this is all integer programming problem because all the variables are integers. When some of the variables are continuous and some of them are integers it is called a mixed integer programming problem. We will first consider the all integer example. In all these kind of problems or in general for any integer programming

problem the first thing to do is, to relax the integer restriction and try to solve it as a linear programming problem.

If the linear programming optimum is integer value for all the integer variables then it is optimal to the IP so the first thing we need to do is to relax the integer assumption or restriction and solve it as a linear programming problem.

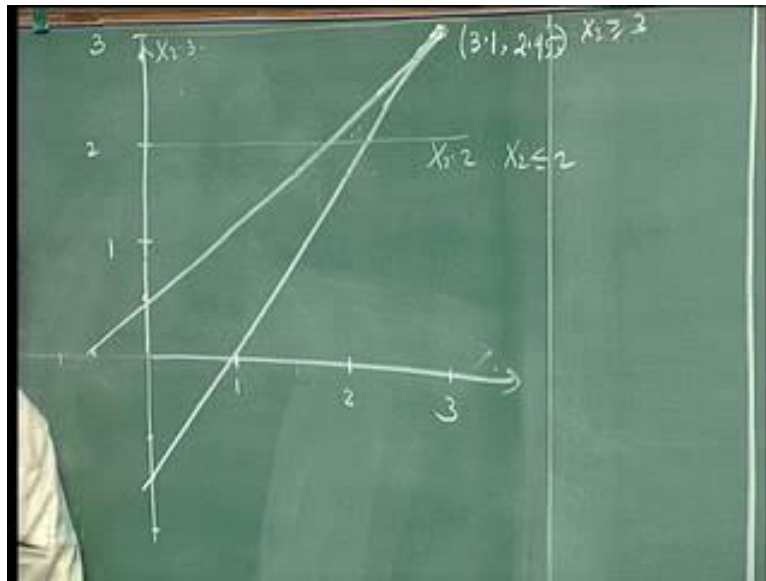
(Refer Slide Time: 13:28)



We will create the first thing by a node which says LP optimum will be X_1 star equal to 28 by 9. Now X_2 star equal to 133 by 45 and Z equal to 273 by 45. Now let us assume, that please remember that we know how to solve a linear programming problem not only that we know how to solve it efficiently. You could use many things like your column generation if needed or the efficient ways of inverting the matrix and then solve it.

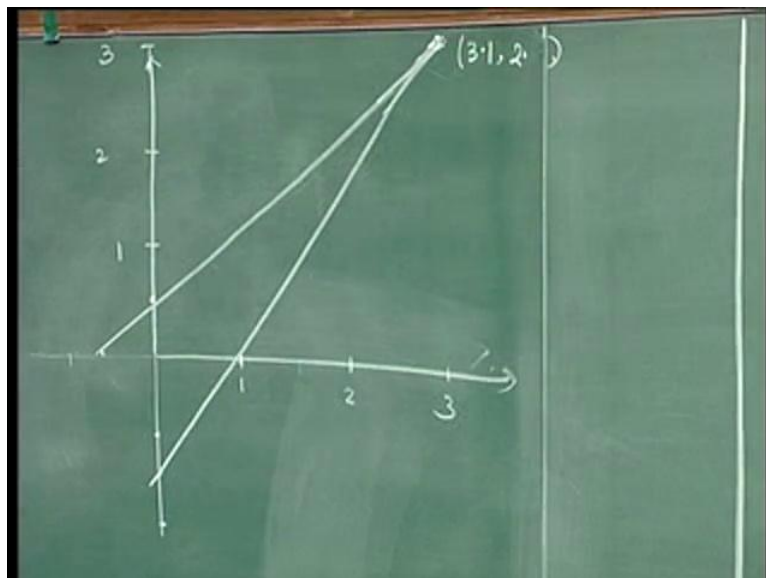
Now, this LP optimum does not have an integer value for either of the decision variables so we need to do something to get to this and what we do is this. Now let us try to pictorially depict this LP optimum.

(Refer Slide Time: 14:39)



Let us say we need to plot both the constraints so $7X_1 + 5X_2$ less than or equal to 7, say if we call this as 1 and 2. So, the first constraint would have a 1, 0 here X_1 equal to 1, X_2 equal to 0 and then we have 0 and 7 by 5. I have 1, 0 for the first one and X_2 equal to 7 by 5 which is roughly ah which is which is 1.4. So this is the first line which is here second one would give us X_2 equal to 7 by 15 which is somewhere here and X_1 equal to minus 7 by 12 which would be say somewhere here so this would be our... one second, let me just check ... $7X_1 + 5X_2$.

The $7X_1 + 5X_2$ less than or equal to 7, minus $12X_1 + 15X_2$ less than or equal to 7. (Refer Slide Time 16:38)



The first one is 1, 0 and this would be minus 1.4 which is somewhere here first line would go like this. Second line would be 7 by 15 which is somewhere here and minus 7 by 12.

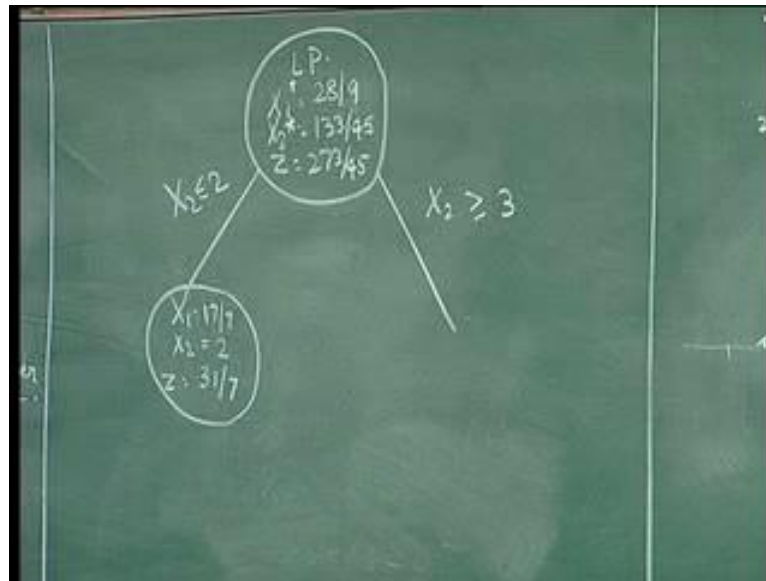
Let us say 28 by 9 is somewhere here and 133 by 45. This should be somewhere here and assume that it goes like this and somewhere here is the LP optimum. We will say that this is 3.1, 2.45 three are 135 so 2.0 something.

We have X_1 equal to 3.1 X_2 is 273 by 45 which is like 2.95. Now what we understand from this is this corner point solution is not optimal that is number 1 number 2 is we would we would not be interested in. For example, if I take a variable say X_2 under consideration and I kind of draw a line here that represents X_2 equal to 2 and there is a line here which is X_2 equal to 3. I know for sure that every point here on the feasible region which is above X_2 equal to 2, is not actually feasible with respect to the integer programming because I have a restriction that X_2 should be an integer value.

X_2 can only take a value 2 or 3. If I am in this territory, X_2 could take 1 if I am in another territory but looking at the optimum if I basically take one variable which is X_2 I could do the same interpretation for X_1 . If I take variable X_2 which is at 2.95 I know that I am not interested in this value it is infeasible to the IP.

I am interested in all integer values where X_2 is less than or equal to 2 or X_2 is greater than or equal to 3. I am not interested in any value which is in between this. So, what I will do now is I take any one of the variables which has a non-integer or a fractional value at the optimum in this cases I might take X_2 . I can do the same thing with X_1 .

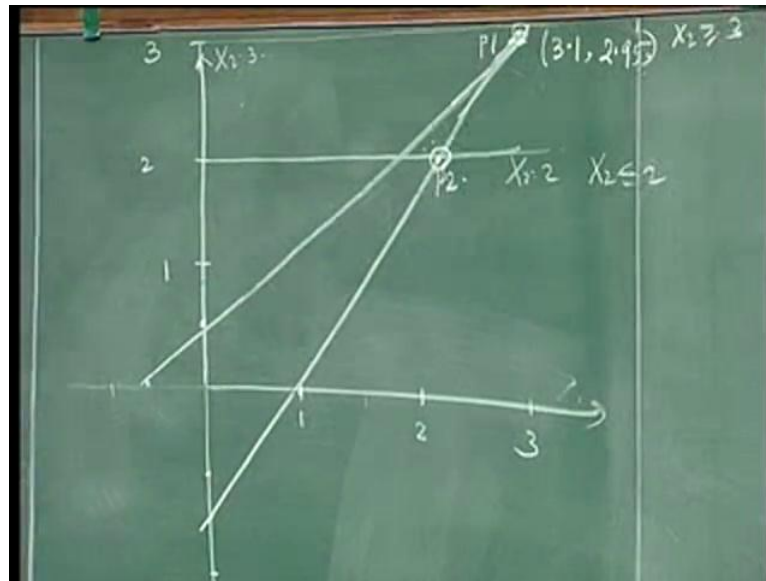
(Refer Slide Time: 20:04)



Look at the value 2.95 and I create two nodes from this tree one which says, X_2 less than or equal to 2 less than or equal to the lower integer value and X_2 greater than or equal to 3 which is greater than or equal to the higher integer value. Because I know that any X_2 which is in between these two for example anything that is slightly above 2 or slightly below 3 but fractional value is not desirable because it is going to be infeasible to the integer programming problem. So, you create two branches from the original LP by adding these two constraints X_2 less than or equal to 2 and X_2 greater than or equal to 3 and solve two resultant linear programming problems. You should not put both the constraints into the same problem because if you put both the constraints into the same problem you end up creating X_2 infeasible.

The very fact that you have written two constraints there is no feasible area corresponding to these two but then you know for sure that such an area you do not want look at because it is not feasible to the IP. So, you will create two integer programming problem two linear programming problems at every node that your branch. One less than or equal to the lower integer value of a fractional variable and the other greater than or equal to the higher integer value of the fractional variable. In this case you get less than or equal to 2 and greater than or equal to 3 and solve this problem all over again solve the original LP, with the additional constraint that X_2 less than or equal to 2, when you do that you get a solution X_1 equal to 17 by 7 X_2 equal to 2 and Z equal to 31 by 7 which means basically we put this constraint.

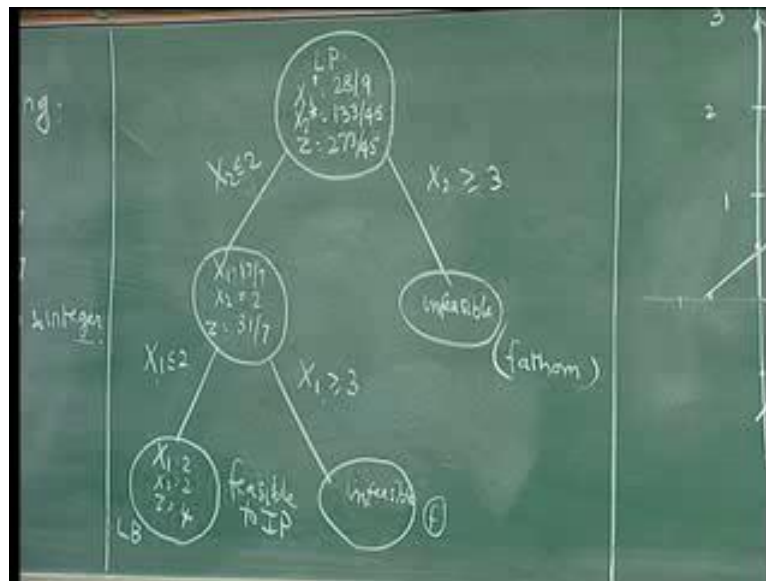
(Refer Slide Time: 22:11)



Now with this feasible region, I am trying to solve the problem and I realize that this is my corner point. Now, from here I have moved to this point this is for my original P_1 this is for my P_2 . Now when you put X_2 greater than or equal to 3 to the original set of constraints you should not put the this one also again this has already been solved. You put X_2 greater than or equal to 3 into the original two constraints and solve the problem. You get infeasibility here. Now, that is understandable from the graphical region one can easily see that X_2 greater than or equal to 3 results in infeasibility.

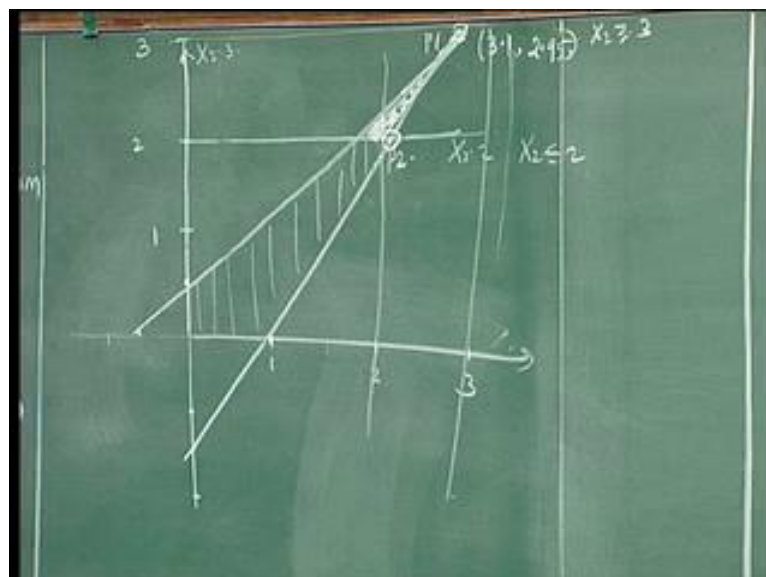
The moment a node is infeasible you fathom it by infeasibility fathoming by infeasibility we have already seen in the 0, 1. So, you do not proceed further, because proceeding further would mean putting some more restriction already it is infeasible so, putting another restriction can never bring in feasibility. So, it will continue to be infeasible. You do not do that. The moment it is infeasible you fathom it. You fathom which means, you do not move further from the infeasibility, there is no back tracking involved in this algorithm so we just fathom by infeasibility.

(Refer Slide Time: 23:37)



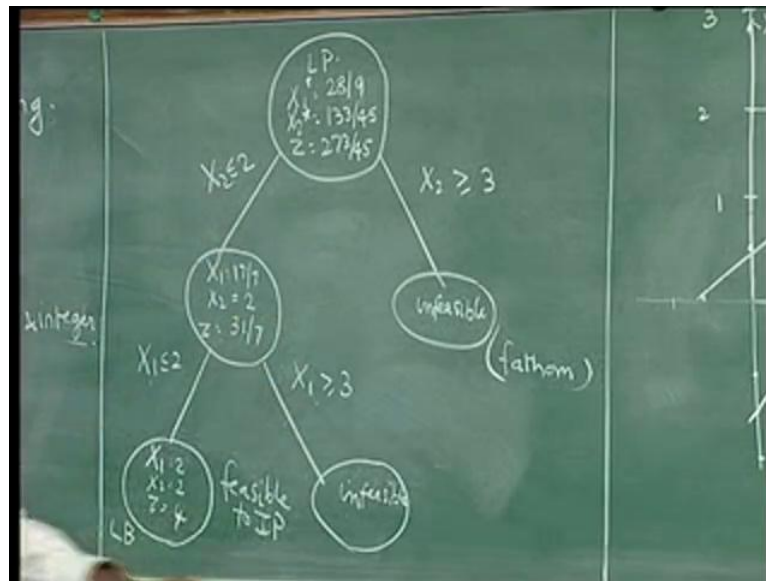
Now look at this node, which is active. Now here this has an integer value this does not have an integer value. So what do I do? I branch on this. I take this one 17 by 7 is 2.0 and something so I create two nodes one with X_1 less than or equal to 2 and the other with X_1 greater than or equal to 3 . When I put an additional restriction that X_1 less than or equal to 2 ...

(Refer Slide Time: 24:09)



Now I come back here. Now, this is where my 17 by 7 is so. Let me assume that my X_1 less than or equal to 2 is somewhere here and X_1 greater than or equal to 3 is somewhere here.

(Refer Slide Time 25:42)



Less than equal to 2 will give me another one which is somewhere here so when I do that I am able to get this point which is in the feasible region. So I get X_1 equal to 2 X_2 equal to 2 and Z equal to 4 which is a feasible solution to the IP. This is feasible to the integer programming problem and I update the solution. I have one integer solution which I update now I do not proceed further here because I already have an integer solution so this is fathomed by feasibility.

The moment I get a feasible solution to a maximization problem acts as a lower bound., because my optimum to the IP, can only have a value of 4 or more because I already have a feasible solution with Z equal to 4. Now, I go back and do the additional one and get infeasible here when I solve it. Remember that when I am solving for this or for this I should add both X_2 less than equal to 2 and X_1 less than equal to 2. When I am solving here I should add X_2 less than equal to 2 and X_1 greater than or equal to 3 to these two.

There are four constraints now two additions that have come in to this. Now, this infeasibility is also indicated by the graph that it is totally out of the feasible region. So, it is infeasible and it is fathomed by infeasibility now there are no nodes that are right now hanging the algorithm terminate then I cannot branch. Now this is a simple branch and bound algorithm. To solve integer programming problems, we have taken a very small and simple example which kind of terminates in five nodes and gives us the optimum.

Now what we were trying to do in all these aspects whenever we branched from a node, we were basically trying to eliminate a set a certain feasible region from this feasible region. For example, when we branched on the first time, we were actually trying to leave out this much of a feasible region by putting an X_2 less than or equal to 2 and an X_2 greater than or equal to 3, we eliminated this portion of the feasible region. So, every time you branch, you end up eliminating or leaving out a certain portion of the feasible region and somewhere, in the order in which you do it you will end up if the problem has the IP has an optimum then you will reach the optimum in a finite number of iterations but there is no way to guarantee what is that finite number. It can be a very large number of iterations but it does converge in a finite number of iterations that's one. The idea is whenever we branch we kind of create either two horizontal lines in the graphical representation or create two vertical lines and that part of the feasible region that lies between those two. In this case two horizontal lines if you branch on X_2 that part of the feasible region is eliminated.

You look at either this or the other it turns out in this case that it is infeasible and so on. Next now another important thing is this; every branch and bound algorithm basically has three things. It has a branching strategy. It has a bounding strategy and it has a node selection strategy. These are the three things that are associated with any branch and bound algorithm. Now, what is the branching strategy? For example, what is the bounding strategy? What is the node selection strategy?

Let us look at the bounding strategy that's easy. Bounding strategy here is you fathom by infeasibility you fathom by feasibility and you also realize that at every stage you are solving a linear programming problem, you are solving a maximization linear programming problem therefore the LP optimum is an upper bound to the IP optimum. I have a maximization problem. For example, here the optimum is 4, now the value here; if you divide 273 by 45 would be anything between 5 and 6. It is a 6.0 and something 6 and 3 by 45 we got 6.0 something here now that is an upper bound to the IP optimum now when you branch and get 31 by 7 you get 4.0 something which also acts as an upper bound. In fact right here you could have done one more thing. This is 4.0 and something all your coefficients are integers so 4.0 and something would imply that four is a realistic upper bound for the integer programming problem. Here I have a feasible solution with Z equal to 4 which is a lower bound feasible solution is a lower bound and right here I've got it. Even if I proceed here I can only get 4 or

less and do not even has to evaluate this infeasibility I can just say right here that I have found the optimum. Now those things are possible depending on certain situations.

The bounding strategy would be every node solves a linear programming problem with a maximization objective. So, what we get is an upper bound. Bounding strategy would be bound by infeasibility bound by feasibility, and the fact that every bound we solve every node we solve a linear programming problem which gives an upper bound to the IP optimum for a maximization problem.

Now, what is the branching strategy we could have in the same problem we could have branched on X_1 or X_2 at any stage, we started here branching on X_2 . We could have started the same thing branching on X_1 sometimes branching on X_1 would result in fewer nodes sometimes branching on X_2 would result in fewer nodes. Branching strategy would simply imply having chosen a node to proceed further from that node which variable are you going to branch on. Very conventionally what you would do is you would pick a variable which has the largest fractional component and then you proceed. It is a very customary practice that you take a variable that has the largest fractional component.

In this case this had 3.1 and something. The fractional portion was a 0.1, here you had a 2.95 fractional portion is 0.95. So, it is customary to take the one with that. There is no proven evidence that if you branch on the variable that has the largest fractional value you will get the optimum. In few iterations there is no such proven rule. It is only a greedy approach where we think that because it is very close to an integer value you might get that value in the next one. So, for all you know the $2X_2$ equal to 3 never figured in the solution. The 2.95 would make it closer to 3 but the optimum did not have X_2 equal to 3. In fact X_2 equal to 3 was infeasible but it is customary that you branch on a variable that has the largest fractional component.

Now, what is the node selection? Strategy node selection strategy we actually did not have any node selection strategy in this example. The reason being we started with a first node which was a linear programming solution and then you branched. One of the branches became infeasible and only one branch you could proceed further. This was the only node available for selection from here also we branched and right here we got the optimum. So, we did not have any node selection to do. By default we were having only one node in this example.

There can be situations where we may have more than one active node. In fact, in almost all situations there will be a lot of active nodes and you may have to choose one out of the existing active nodes to branch further. An active node is one, which has not yet been fathomed and it has a fractional value for at least one of the integer variables. That's called an active node.

Normally, you branch on that node that has the largest value of the upper bound. LP optimum indicating the upper bound that node among the active nodes that has the largest value of Z . LP is there space between z and n LP. The LP optimum is usually the node that is selected for branching. Again it is a greedy approach; for example, when I say here 6.0 and something if I choose a node with there is a node with an upper bound of 6.0 and something there is a node with an upper bound of 5.0 something. Let us say, if it turns out that the LP optimum is 4 that it is it is pretty much possible that move proceeding from the node with 5.0 and something may give it may reach the optimum faster.

We always end up being optimistic saying if there is a node with an LP value of 6; it gives us an outside chance of getting an IP value of 6, whereas, proceeding from a node with an LP value of 5.0 and something can only give us at the maximum an integer programming solution with Z equal to 5 so, we do not do that. Eventually all nodes have to be evaluated. There is nothing like saying that I won't do this I will only ah branch on one node.

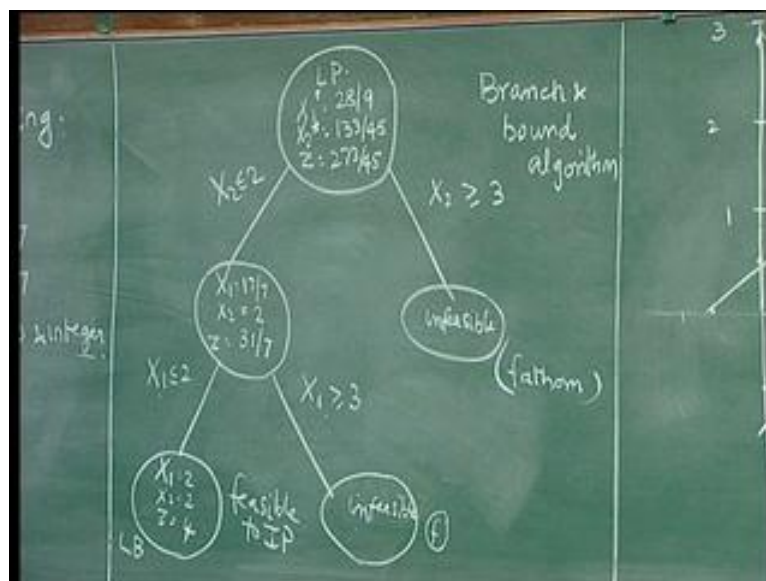
By and large all these three things have to be done very meticulously and it is also well known, that depending on the choice of these three both the memory requirement as well as the time requirement of the IP problems are defined. Why does the memory requirement come in here? We do not in the earlier 0, 1 implicit enumeration algorithm the variable to branch could take only two values. Backtracking was a very convenient way of moving from one node to another in a branch and bound tree a variable that takes a value 1 can only take a value 0 here that cannot happen. For example X_2 may be between 2 and 3 so, you branch on X_2 with 2 and 3 somewhere later when you move down X_2 would take a value between 1 and 2 and you might do some branching. You do not have that a variable appearing only once in the branching with the 0, 1 and so on. So, here you have to store all the active nodes. Storage becomes an issue in this kind of a branch and bound storage was not an issue in the implicit enumeration algorithm.

Normally, if right at the time of storing you start in the decreasing order of LP optimum and then store at the right place so, that your node selection becomes easy you always branch on the first active node the node that has the largest LP optimum.

So, these are the various issues associated with a branch and bound algorithm for integer programming. The three strategies branching bounding and node selection as well as the branch every time we branch from this we actually eliminate a certain area from the feasible region which is bounded by two horizontal lines or two vertical lines. If we have this is for an all integer problem where all the variables are integers if we have a mixed integer programming problem, where some of the variables are continuous and some other variables are integers this becomes a little easier.

Out of all the variables set there are only going to be fewer variables that have to be integer value. In some sense with fewer nodes we should be able to terminate at the optimum you do not have to branch on the variables that have greater than or equal to. You need to branch only on those variables that have integer restrictions and at that point do not have integer values. So, this is how the general branch and bound algorithm works with the three things branching bounding and node selection. Now we look at another way, to solve integer programming problems yeah in fact before that we need to look at one more aspect which is this.

(Refer Slide Time: 38:05)



If you take this branch and bound tree, for example we should remember that when we created an additional node from here, with the addition of a constraint X_2 less than or equal to 2. One need not solve the linear programming problem all over again one could use sensitivity analysis as an additional constraint and do it or one could simply do an upper bounded simplex from the existing LP optimum one can easily do that.

You have two constraints you have because all these constraints are only bounds you have two constraints you will continue to have two constraints. Even if you do sensitivity analysis, you will be introducing a third constraint explicitly whereas, when you straight away plug in an upper bounded simplex from the existing optimum basis you still retain the two constraint thing where the bound goes out of the problem.

Typically the problem size the number of nodes may increase significantly but the problem size is always a two constraint. There could be some other issues when you start doing it but certainly the problem the constraint set does not become large because of the addition of these and each one is only a kind of a bound.

In spite of that the problem is a worst case hard or enumerative kind of an algorithm because you do not know how many nodes you will evaluate before you terminate. Let us look at another way to solve integer programming problem.

(Refer Slide Time: 40:12)

	X_1	X_2	X_3	X_4			
ng.	0	7	-5	1	0	7	→
	0	-12	15	0	1	7	
	1	1	0	0	0	0	
	1	X_1	$-5/7$	$1/7$	0	1	
	0	X_2	$45/7$	$12/7$	1	19	→
	0	$12/7$	$-1/7$	0	1		
integer	1	X_1	0	$1/3$	$1/9$	$28/9$	
	1	X_2	0	$4/15$	$7/45$	$133/45$	
	0	0	$-9/15$	$-9/15$	$273/45$		

As usual we solve this first by the simplex algorithm to get an LP optimum so, let us construct the simplex table and then solve it first using a simplex algorithm. You will have X_1

X_2, X_3, X_4 we start with X_3 and X_4 here so, you have X_3, X_4 here. There are $1 \ 1 \ 0 \ 0 \ 0 \ 7$ minus $5 \ 1 \ 0 \ 7$ minus $12 \ 15 \ 0, 7 \ 1 \ 1 \ 0 \ 0 \ 0$. You could enter on X_1 you could enter on X_2 so, let us go back and enter on we enter X_1 here.

Find there's only one leaving variable so, this is a leaving variable because of this negative there is only one leaving variable so, I have X_1, X_4 . So, I have 1 and $0 \ 1$ minus 5 by $7, 1$ by $7, 0, 1$. This plus 12 times this so, $0 \ 15$ minus 60 by 7 is 45 by 7 , this plus 12 times this is 12 by $7, 1, 7$ plus 12 into 1 is 19 so we get a 1 here. This becomes 0 this becomes $0 \ 1$ plus 5 by 7 is 12 by 7 and this is minus 1 by 7 .

X_2 enters again there is only one leaving variable here this here. I have $X_{1.1}, X_{2.1}$ this is the pivot element 45 by 7 is the pivot. So, multiply or divide by the pivot to get $0, 1, 12$ by $45, 7$ by $45, 19$ into 7 is $63 \ 133$ by 45 here.

This plus 5 by 7 times this $1 \ 0, 1$ by 7 plus 5 by 7 into 12 by $45, 12$ by 45 is 4 by 15 so, you get 1 by 7 plus 12 by 7 into this 12 by 35 so 1 by 7 plus 12 by 7 is 17 by 35 and we get something else here; minus 5 by $7 \ 1$ by $7 \ 0 \ 12$ by 7 so this should become 12 by 45 which is 4 by 15 is written down here, this is 4 by 15 and you have a 7 by 45 here.

This is 1 by 7 plus 5 by 7 into this so, 20 by 105 which is 4 by $21 \ 4$ by 21 so, 1 by 7 plus 4 by 21 is 1 by 3 so, 1 by 3 this plus 5 by 7 into this which gives me 5 by 45 which is 1 by 9 . This plus 5 by 7 into this so, 1 plus 5 by 7 into 133 by 45 so that is 28 by 9 which we know anyway so, 28 by 9 .

The Z value is 28 by 9 plus 133 by $45 \ 273$ by 45 and you get $0 \ 0$ minus 9 by 15 minus 4 by 15 so, this is the LP optimum that we have. Now, we realize that both X_1 and X_2 are not integer valued we can choose any one of them and proceed further.

We just now saw that it is customary to take that variable which has the maximum fractional portion for subsequent addition or move in the algorithm so we take this and now we write this in the form of an equation.

(Refer Slide Time: 45:45)

$$X_2 + \frac{4}{15}X_3 + \frac{7}{45}X_4 = \frac{133}{45}$$

$$X_2 + \frac{4}{15}X_3 + \frac{7}{45}X_4 = 2 + \frac{43}{45}$$

This becomes X_2 plus $\frac{4}{15}X_3$ plus $\frac{7}{45}X_4$ equal to $\frac{133}{45}$. Now, because this is a simplex iteration and it represents LP optimum this will have to be nonnegative even if it is a dual simplex iteration, it does not matter because this represents an LP optimum so this has to be non negative. What we do is we rewrite this in such a way that I do not have a fraction for X_2 so I just write X_2 the fraction the coefficient of X_3 turns out to be a fraction less than one right $\frac{4}{15}$ so I just write it as $\frac{4}{15}X_3$.

For example, if this coefficient had been $\frac{19}{15}$ I will write it as plus 1 X_3 plus $\frac{4}{15}$. I will always write it as an integer plus a positive fraction which means if it were minus $\frac{4}{15}$ I would have written it as minus 1 plus $\frac{11}{15}$. I should always write it as an integer plus a positive fraction. This becomes $\frac{7}{45}X_4$ becomes 2 plus $\frac{43}{45}$. Right-hand side is also written as an integer plus a positive fraction. We use this and try to create a cut which we will see in the next class