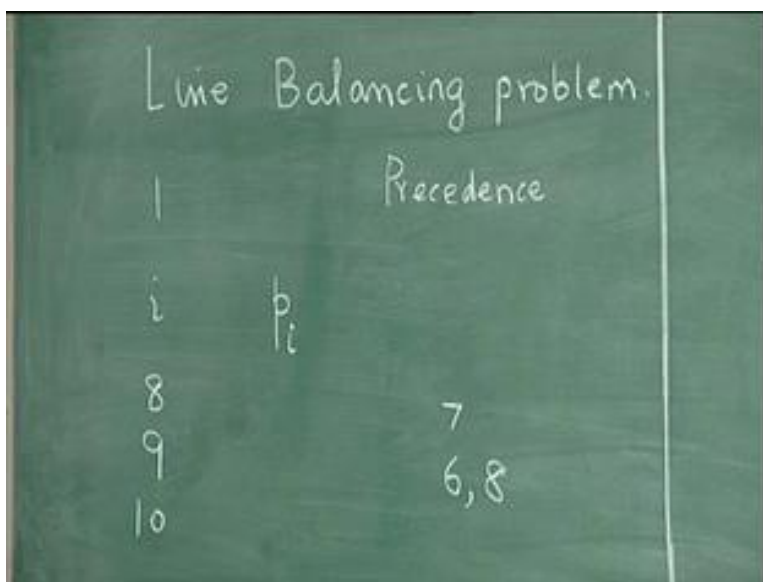


**Advanced Operations Research**  
**Prof. G. Srinivasan**  
**Department of Management Studies**  
**Indian Institute of Technology, Madras**

**Lecture - 13**  
**Solving Zero-one Problems**

Today, we look at another formulation of integer programming which is called the Line Balancing problem.

(Refer Slide Time: 00:19)



The problem is as follows. You can assume that there is an assembly line and say 10 items have to be assembled; we call them 1 to 10. These 10 items have to be assembled and they make a final product. For each of these items, for item  $i$ , the time taken to assemble one piece of item  $i$  into the assembly is given by say  $p_i$ . Sometimes there will be precedence relationships and you might have a relationship which says that, for example, 8 can be assembled only after 7 is assembled or 9 can be assembled only after 6 and 8 are assembled and so on.

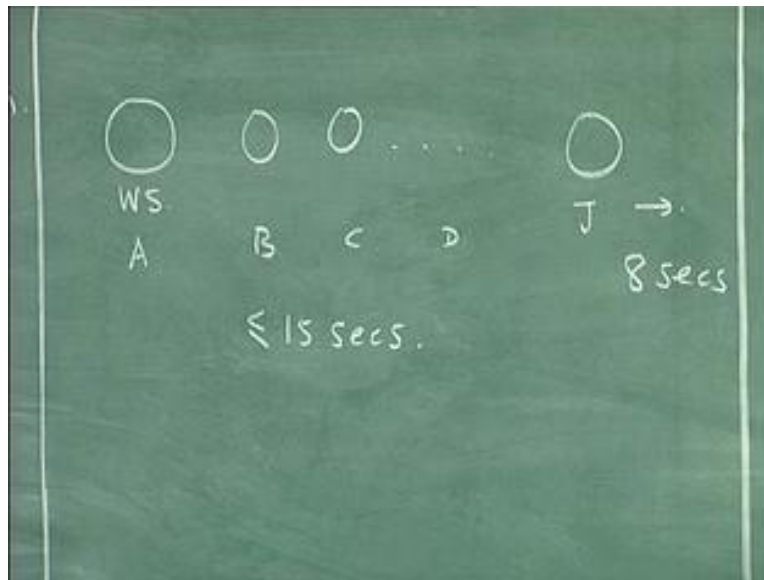
(Refer Slide Time: 01:40)

Line Balancing problem.

Item	Duration (secs)	Precedence
A	8	-
B	6	-
C	5	A, B
D	4	
E	6	
F	3	
G	7	
H	3	
I	4	
J	5	

To take a typical example, we will assume that there are 10 items; we call them A, B, C, D, E, F, G, H, I and J. Let us say the duration given in seconds to assemble would be something like 8, 6, 5, 4, 6, 3, 7, 3, 4, and 5. It adds up to 51 time units. Let us say there is precedence; A does not have any precedence, say B does not have any precedence, C can be assembled only after A and B can be assembled and so on. Let us assume there are some more precedence relationships that exist. Now, these assemblies are done in what are called workstations. The simplest thing to do is we make an assumption in all these problems that a feasible solution exists and the order A to B, B to C, C to D, D to E, I to J is feasible. You will not be violating any precedence relationship if we start assembling in the order A, B, C, D, E, F, H, I, J.

(Refer Slide Time: 03:19)



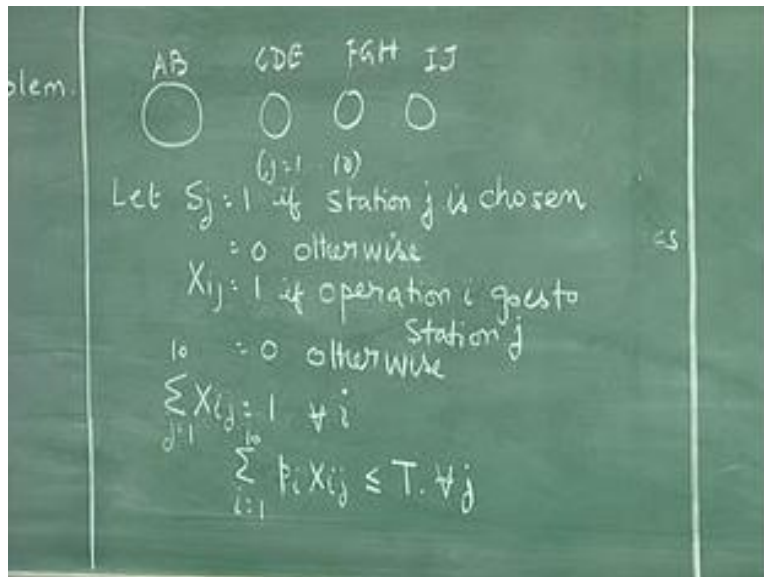
The easiest thing to do is to have only one workstation, we call it one workstation and we have one person who is managing this workstation and all the items are assembled here. So, every 51 time units or seconds one piece will come out of this system. We will assume that we need to assemble a large number of these finished products say 100 or 1000 or some such number. If we have one person doing the whole thing in one workstation every 51 minutes or seconds a piece will come out of this system. Since we know and assume that the order A, B, C, D up to J is feasible, the other extreme would be to have 10 workstations and each does A, B, C, D and so on. The first person will assemble A, and then give it to the next person who will put B in it and then C, D, E and so on. The last person who is having workstation J will feed it to the output. When we do such a thing, the steady state output of this system will be 8 seconds. The first piece will come out in 51 seconds and then for example, every 8 seconds another piece will come out. What we are trying to do is that if we give an intermediate value for this cycle time, say for example 15 seconds my demand is such that I want an output every 15 seconds.

The problem is what is the minimum number of workstations with which I can do this such that, the workstation time does not exceed these 15 time units and the precedence relationships are satisfied, they are not violated. For example, C can be done only after A and B can be done, so I

cannot allocate item. Now, I have a few numbers, I know for sure that I need at least 4 workstations; because the total time is 51 and I want a 15 second output, I need at least 4 stations and so on. Suppose I am looking at a solution with 5 stations. I may give, for example, AB here, which takes 14 time units, CDE takes 15 time units, EFGH takes 13 and say IJ; so with 4 workstations I can do it, provided I have only one precedence relationship which is given or we will assume that the other precedence relationships are such that this is feasible. For example, I cannot have a solution where C is given here, A and C are given to this, and B is given. In my anxiety to make all these closer to the 15, I cannot put C and A into one workstation and then B goes to the next station because it is violating the precedence relationship. I should not violate the precedence relationships. I would like to have a maximum of 15 time units which is the output and so on.

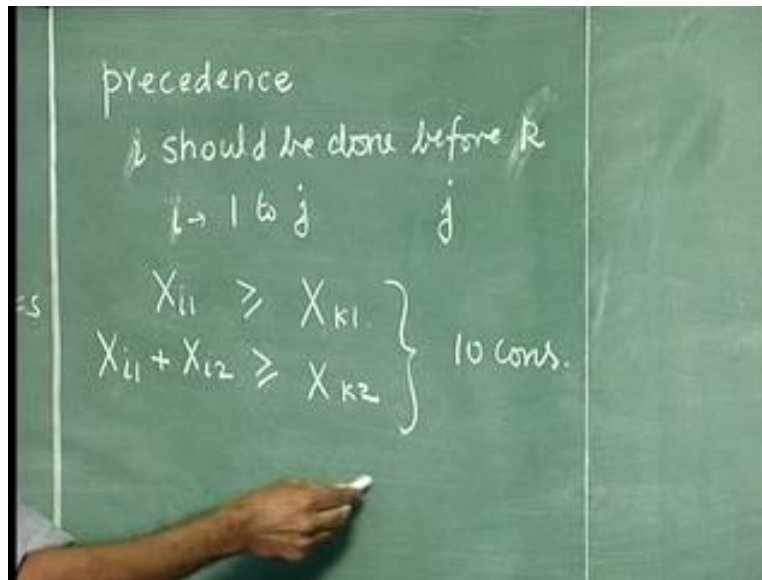
I may allow A and B, C here, which is fine if it satisfies the less than 15 second constraint. I may allow an A here followed by a B, C because I will always assume that a person who is doing the work here will first assemble B and then C, so that the precedence is not violated. So when there is a precedence relationship it is not absolutely necessary that this A and B should be done before or in an earlier workstation. A can be done in an earlier one, B and C can be done in the same workstation in the order BC, but I can put all ABC in one if it does not violate my 15 and so on. The idea is that the precedence constraint should not be violated; the cycle time per workstation should not exceed this 15. Let us try formulating an integer programming problem that will do this. The idea is to minimize the number of workstations such that this time constraint is maintained.

(Refer Slide Time: 08:03)



We will say let  $S_j$  be equal to 1 if station  $j$  is chosen, equal to 0 otherwise. Remember that when you define  $S_j$  equal to 1 you need to define an upper limit for  $j$ .  $j$  has to be from 1 to something and that upper limit cannot be a variable. It has to be a fixed parameter in the problem. The problem tells you that a maximum of ten stations are only possible because you have a maximum of 10 operations to be performed.  $S_j$  equal to 1 will have  $j$  equal to 1 to 10. The maximum of 10 workstations are possible out of which I am choosing a certain number. Whichever I choose, I put a 1; when I do not choose it, I put a 0.  $X_{ij}$  is equal to 1, if operation  $i$  goes to workstation  $j$ ; equal to 0 otherwise. The first constraint would be every operation goes to only one workstation. So,  $\sum_{j=1}^{10} X_{ij} = 1$  summed over  $j$  equal to 1 to 10 in this example, for every  $i$ . The next constraint will be the cycle time restriction that you have. So this will become  $\sum_{i=1}^{10} P_i X_{ij} \leq T$ , summed over  $i$ ,  $i$  equal to 1 to 10 in this case, should be less than or equal to some capital  $T$ . This is for every station. The time that is taken in every workstation should not exceed the upper limit of capital  $T$ . In this case it is 15. This  $T$  will become 15 and the individual  $P_i$ s are this 8, 6, 5, 4 and so on. The next thing we need to model is the precedence.

(Refer Slide Time: 10:34)

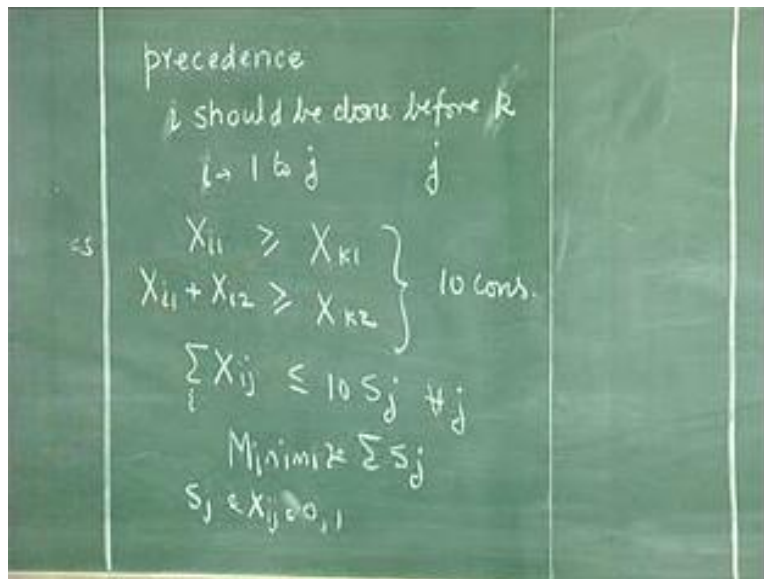


If  $i$  and  $j$  have precedence that is, if  $i$  should be done before  $k$ , since we have used  $j$  for the station let us say, before  $k$ , operations  $i$  and  $k$  have precedence. Then, if operation  $k$  goes to station  $j$  then operation  $i$  can go to anything from 1 to  $j$  and that has to be modeled. If we assume that  $X_{k1}$  is 1, then  $i$  should definitely go to 1, so there will be a  $X_{i1}$ . If  $X_{k2}$  is 1, then  $X_{i1}$  plus  $X_{i2}$  should be 1 that is the idea. How do you model this?  $X_{k1}$  should be greater than or equal to  $X_{i1}$ . Let us go back and check what happens here. If this is 1, then this has to be greater than or equal to this, this cannot be 0. If this is 2, then this can be 1 or this can be 1 but one of them will have to be 1 if this condition is satisfied. Just put the greater than or equal to; that will take care. Because you may have a situation where this can be 0, this can still be 1, but  $k$  can go to 3. So, you will have a series of constraints which run like this.

For a single precedence relationship between a pair  $i$  and  $k$  you will have ten constraints which happens and all these constraints are “either or” constraints. If  $k$  goes to the first workstation I have one constraint; if  $k$  goes to the second workstation I have one constraint; if  $k$  goes to the third workstation I have one constraint. They need not be “either or” constraints because  $k$  will go to only one of them. So automatically only one of these ten will be valid because  $X_{kj}$  equal to 1 summation from here, every  $k$  will go to only 1 which automatically implies that only one of

them will be 1 the rest of them will be 0 and so on. So they are not “either or” constraints. Will all ten be valid? All ten will be valid but eventually only some of them will be active, some of them will not be active. You do not know which one is active and which one is not active till you solve it. The point is, for every precedence you will have these 10 constraints which will come.

(Refer Slide Time: 14:30)



Most importantly there is one more constraint which is this.  $\sum X_{ij}$  is less than or equal to 10 times  $S_j$  summed over  $i$  for every  $j$ . What does this tell you? This tells you two things: It tells you that I can assign an operation  $i$  to a station  $j$  only when that station  $j$  is chosen. I have ten stations and I am not going to choose all ten of them. I am going to choose a certain number of them which I am trying to minimize. So only when I choose a workstation I can put things into it and I can put a maximum of ten, this in general a big  $M$ . This became ten because you have ten operations like your fixed charge.

If you go back to an earlier formulation, we said  $X_{ij}$  will be less than equal  $M$  into  $y_j$ . Only when it is chosen you can put something and you can put a maximum of 10 in this case instead of a big  $M$  because only a maximum of 10 will go. This is the constraint which links the rest of the things to the objective function because your objective function is to minimize  $\sum S_j$ .  $S_j$  is a 0, 1

variable, so this will minimize the number of stations that you will finally end up with.  $S_j$  and  $X_{ij}$ , both are 0, 1 variables. This is the formulation of the line balancing problem. The only thing is, for every precedence constraint or condition you will have these many constraints that appear. Therefore, the problem becomes very large in terms of the number of constraints as well as number of variables because when  $X_{ij}$  is  $i$  equal to 1 to  $n$ ,  $j$  equal to 1 to  $n$ . If you are looking at 10 operations, you are looking at 100 variables and then there are 10  $S_j$ 's, so there are 110 variables and many constraints. For example, this is a set of 10 constraints, for this problem, this is another 10, this is another 10, plus every precedence will have 10 more constraints. So, you will have too many constraints for this. What you do if you want to really solve this problem in practice is this. This is exactly why we wrote this.

(Refer Slide Time: 17:06)

Line Balancing problem

Item	Duration (secs)	Precedence
A	8	-
B	6	-
C	5	A, B
D	4	
E	6	
F	3	
G	7	
H	3	
I	5	

If you go back, assuming that this is the only precedence relationship, which means you have only 10 constraints here and not more than 10.

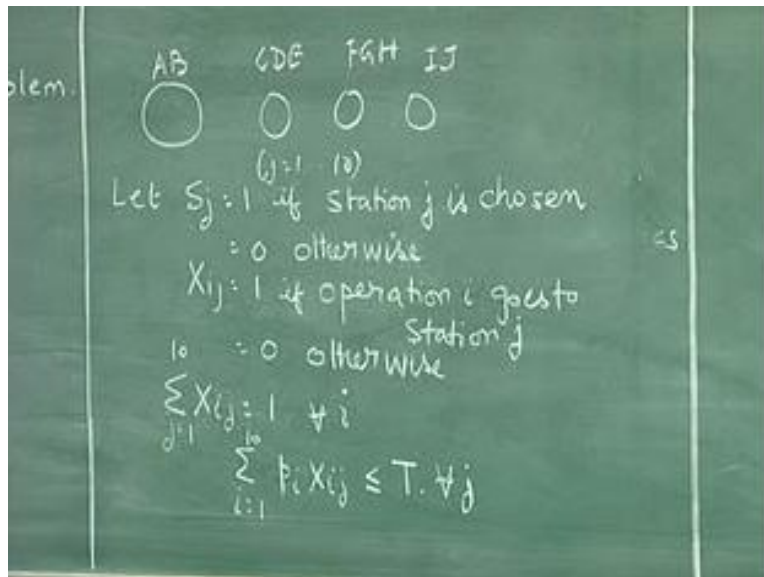


(Refer Slide Time: 17:25)

precedence  
i should be done before k  
i → 1 to j      j  
cs  
 $X_{i1} \geq X_{k1}$   
 $X_{i1} + X_{i2} \geq X_{k2}$  } 10 cons.  
 $\sum_i X_{ij} \leq 10 S_j + j$   
Minimize  $\sum S_j$   
 $S_j, X_{ij} \geq 0, 1$

Knowing that this is the only precedence relationship and this is the feasible solution with 4 workstations for your 15, what you normally do is you apply a quick heuristic or a quick algorithm, which will tell you a certain number. It will give you a feasible solution and this feasible solution has only four workstations. Based on this feasible solution, you know that there is a solution which is possible for this problem with 4. I am not interested in solutions which give me 5, 6, 7, 8, 9, 10 workstations. So quickly reduce your  $j$ , instead of 1 to 10 to 1 to 4 and solve it again. Automatically, the number of variables and constraints will come down the moment you get a feasible solution. Now you solve the problem all over again and if it turns out that this  $\sum S_j$  is 4 then your feasible solution is optimum. If it gives 3, then you can do it in three workstations. So that is how you look at this problem. This is one case where you reduce the number of variables and constraints by identifying a feasible solution and then trying to solve it to get to the optimal, because the formulation for the optimal without looking at the feasible solution has too many constraints and variables. You can follow this approach for certain types of problems, not all.

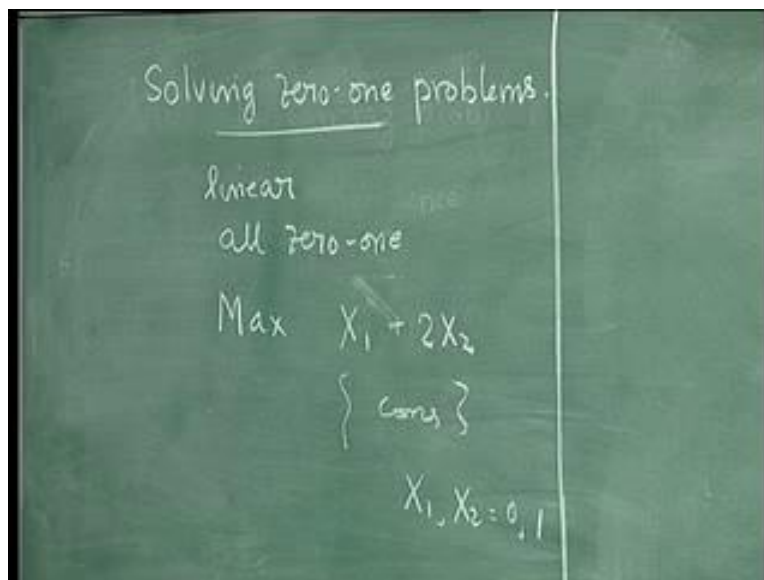
(Refer Slide Time: 18:42)



Not necessarily. If the first one has 5, then this cannot be more than 5. The moment I have a feasible solution with four workstations, my optimal solution can be 4 or 3 or 2 or 1; feasible solution presents an upper bound to the objective function for a minimization problem. The optimum cannot be 5 if you have a feasible solution with 4. Therefore you reduce the 100 variables to 40 variables, consider only four workstations and solve it. You may get 3; if you get 3, then you have a better solution. You can use any greedy algorithm, any algorithm that gives you a feasible solution to begin. If your feasible solution is good enough, then you may end up with 4. If it is not good enough, you may end up with 5. So you have 10 more variables into the problem. No this constraint is absolutely essential because without this constraint you will see that the objective function is independent of the rest of the variables. This is the only constraint which links the objective function to the constraint, to the variables basically. Otherwise,  $S_j$  does not appear anywhere, so this is absolutely essential. This model is a situation that I can assign jobs only when a workstation is chosen. This is very similar to the location allocation model that  $F_i Y_i$  which we did in the earlier formulation. Normally there will be a big  $M$ , because you do not know. Here, because you know that there are 10 operations or activities to be performed, you will say a maximum of 10 can be allocated.

If  $S_j$  is 1 and a station is chosen, a maximum of 10 things can be given. If you want, we can put an additional restriction that every station should have at least one operation assigned to it. You may have that because you do not want null stations. But the way it is formulated it will not happen, because if you have a null station it increases your  $S_j$  by 1 and you are minimizing  $\sum S_j$ . You do not have to explicitly put a condition that every workstation should have at least one. It will not allow a null basically, or even if it allows a null station, and for example, you say it is 4 and from your solution if you see there is a null station, you know that with 3 you can do it. So you do not have to restrict it with more constraints. For example, you will not get a null, it will never be allocated. You know that you do not have to do it. So, with this example we complete our portion on formulating integer programming problems.

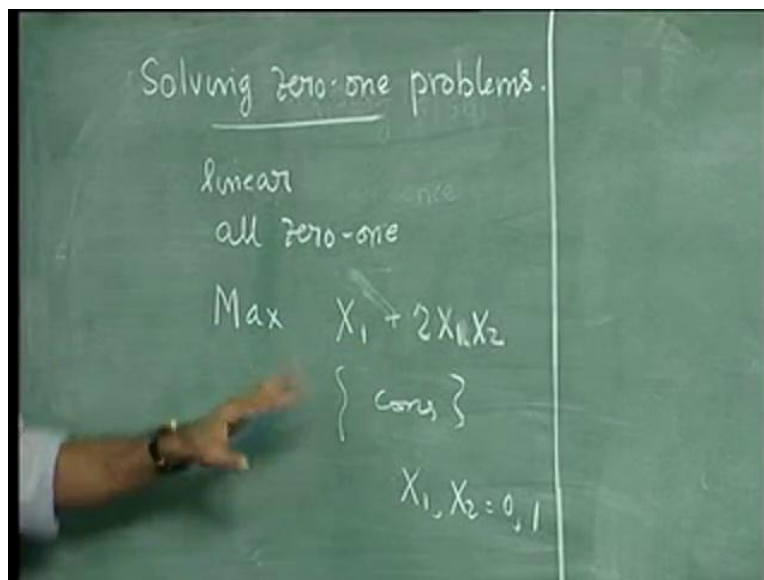
(Refer Slide Time: 22:13)



We will try to move to solving integer programming problems and we take the zero-one to begin with and then we try to solve general integer programming problems. We will look at solving zero-one problems. When we say we are looking at zero-one problems, we are looking at a linear objective function comprising of 0, 1 variables. We are looking at what is called an all zero-one problems. Unless otherwise stated, if you say that a particular problem is a zero-one problem it implies that it has a linear objective function and it has all variables restricted to either 0 or 1.

There can be non-linear problems with all zero-one and I had mentioned briefly earlier that, if you have for example, something which says  $X_1$  square plus  $2X_2$  subject to a set of constraints and  $X_1 X_2$  is equal to 0, 1. If you have a non-linear objective function which is only a power of the  $X$ s, you can always replace this  $X_1$  square as  $X_1$  because  $X_1$  is a 0, 1 variable; any higher power of  $X_1$  is either a 0 or a 1. It does not take any other value, so there is no problem associated with having powers.

(Refer Slide Time: 23:36)



The real difficulty comes only when you have something like a product term, you have 2 into  $X_1$ ,  $X_2$ . We will take an example little later and see how we convert this into a linear system. We will just leave this portion. The only problem there is when you have this power term that comes in it will influence it. The product term is not as straight forward as the other one. You have to model the situation that only when both are 1, this will become 1, otherwise it will become 0. That has to be modeled explicitly. You will replace it with a 2 into  $X_3$  and bring a relationship between  $X_1$ ,  $X_2$  and  $X_3$ , you need to do that. We will do that after we solve the zero-one problem. There are a few other things that can be modeled comfortably with 0, 1 variables.

(Refer Slide Time: 24:31)

Solving zero-one problems.

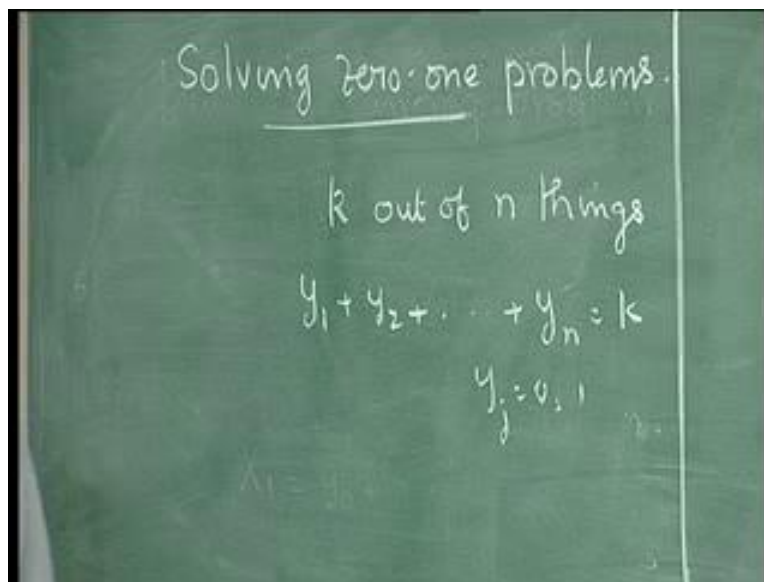
$$\text{Max } 6X_1 + 3X_2$$
$$X_1 \leq 8$$
$$X_2 \leq 7$$
$$X_1, X_2 \text{ integer.}$$
$$X_1 = y_0 + 2y_1 + 4y_2 + 8y_3$$
$$y_0, y_1, y_2, y_3 \text{ 0-1}$$

Normally, we do not do that but for example, if you have a problem which says maximize 6 into  $X_1$  plus 3 into  $X_2$ ,  $X_1$  less than or equal to 8,  $X_2$  less than or equal to 7, say  $X_1, X_2$  integer. Let us not worry about the solution to it. We would only take an example to illustrate something. What happens is, this is an integer problem and therefore, you have bounds. Particularly, when you have bounds for integer variables you can use it to do something. For example, this 8;  $X_1$  can take any value from 0 to 8, so you can write  $X_1$  as  $y_0$  plus 2 into  $y_1$  plus 4 into  $y_2$  and I think you need an 8 into  $y_3$  as well, because there is an 8, so 8 into  $y_3$ . Then you can replace  $y_0, y_1, y_2, y_3$  as 0, 1 variables, binary. Therefore, if it turns out that it takes 8 then, this will take 1. If this takes 7, the rest of them will take 1 and so on.

In some sense, every integer programming problem with some judicious computations, general integer programming problem can be converted into a zero-one problem this way. But then it becomes very cumbersome. When the numbers increase, the number of 0,1 variable also increases. It does not become a function of the actual number of constraints and variables. It is not dependent on the problem size, it is also dependent on the numbers that appear. It may not be a very efficient thing to do.

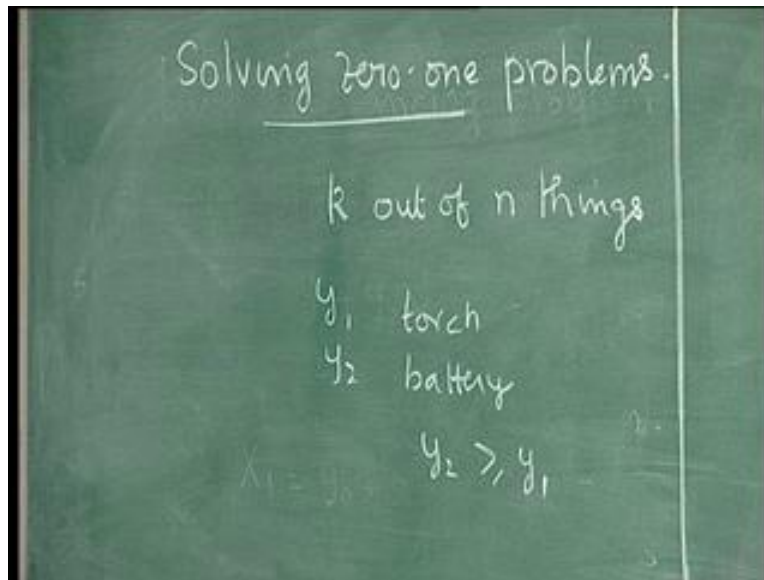
A simple problem like this with smaller numbers can give rise to fewer variables; whereas, if this 8 becomes 8000 or 800,000 then the number of variables becomes much more. But this is something which can be done. When you follow this approach to solve a general integer programming problem that is valid. This is only used to show that it can be done, but it is usually never done. You normally do not like or want to have an algorithm whose running time depends on the numbers that you give. You get into other kind of complexities. Not necessarily. When we look at algorithm to solve integer programming problems, you will see that it is not. Whether the number is 1000 or 10000, the algorithm behaves more or less in the same way. We can use zero-one to model a few other things also.

(Refer Slide Time: 27:45)



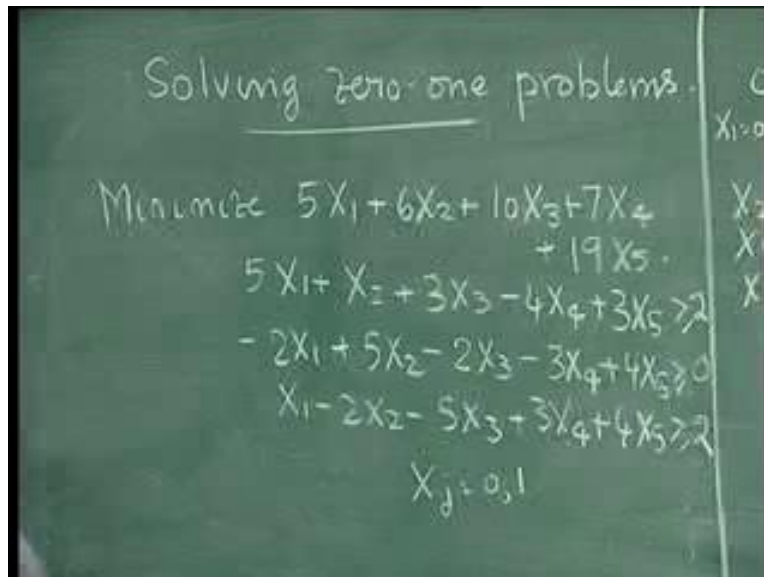
A very simple thing to do based on this is if I want to choose k out of n things, I can easily write it in the binary mode. For example, I will have  $y_1$  plus  $y_2$  plus  $y_n$  equal to k,  $y_j$  equal to 0, 1 will model this situation. I just want to choose k out of n, I get this. Particularly, if you are looking at Knapsack problems and say that a torch is an item and a battery is another item, so when I decide to take a torch I have to take a battery. So something like that can also be modeled.

(Refer Slide Time: 28:22)



For example, if  $y_1$  represents torch and  $y_2$  represents battery, I have to necessarily take a battery if I take a torch, so put  $y_2$  greater than or equal to  $y_1$ . Say  $y_1$  equal to 1,  $y_2$  will have to be 1. Such conditions can also be modeled. I may take a battery without taking a torch that is fine. The condition is if I take a torch then I have to take a battery. I may use the battery for some other purpose as well, so this is all right to do that. Such situations can also be modeled comfortably with zero-one problems.

(Refer Slide Time: 29:17)



Solving zero-one problems.

Minimize  $5X_1 + 6X_2 + 10X_3 + 7X_4 + 19X_5$ .

$5X_1 + X_2 + 3X_3 - 4X_4 + 3X_5 \geq 2$

$-2X_1 + 5X_2 - 2X_3 - 3X_4 + 4X_5 \geq 0$

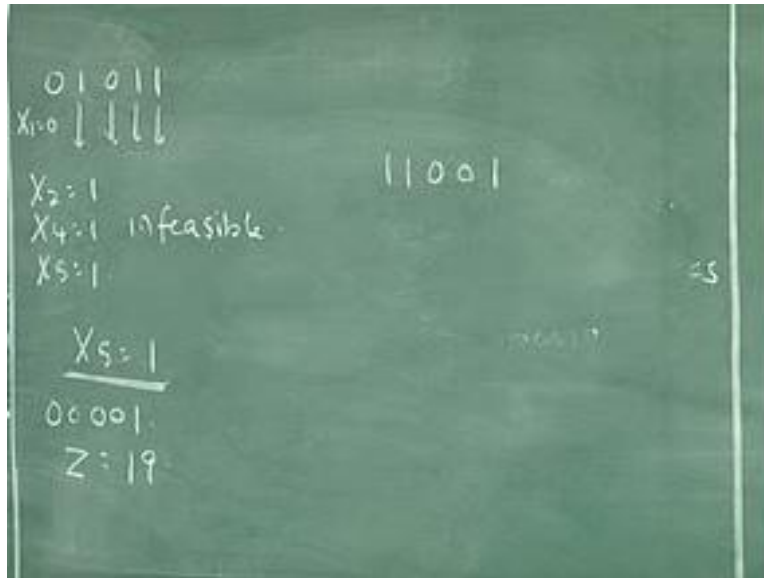
$X_1 - 2X_2 - 5X_3 + 3X_4 + 4X_5 \geq 2$

$X_j = 0, 1$

Now, let us go back and see how we solve zero-one problems. We will take an example and try to explain a few things with zero-one problem. Minimize  $5X_1$  plus  $6X_2$  plus  $10X_3$  plus  $7X_4$  plus  $19X_5$ ;  $5X_1$  plus  $X_2$  plus  $3X_3$  minus  $4X_4$  plus  $3X_5$  greater than or equal to 2; Minus  $2X_1$  plus  $5X_2$  minus  $2X_3$  minus  $3X_4$  plus  $4X_5$  greater than or equal to 0;  $X_1$  minus  $2X_2$  minus  $5X_3$  plus  $3X_4$  plus  $4X_5$  greater than or equal to 2;  $X_j$  equal to 0, 1. The standard problem that we will solve will have two important properties. It will be a minimization problem. All coefficients in the objective function are non-negative. The standard problem again will be a minimisation problem with all objective function coefficients non-negative; number one. Second is all the constraints are greater than or equal to constraints, no matter what the coefficients become in the left-hand side and in the right-hand side; no matter what happens to them, they will be treated as greater than or equal to constraints. These are the two conditions which we will have in our standard problem.



(Refer Slide Time: 31:39)



Then we will look at two things. For example, let us take a solution which is like this. Suppose we take a solution 0 1 0 1 1 which represents  $X_1$  equal to 0, this is  $X_2$ ,  $X_3$ ,  $X_4$ ,  $X_5$ . There are 5 variables, it is a binary problem. So there are 2 to the power 5, total possible solutions and any binary representation of any number from 0 to 31 would represent a solution. For example, 0 is written as 0 0 0 0 0 with all  $X_1$  to  $X_5$  as 0. 31 would be written as 1 1 1 1 1, with all variables to 1. If you take an intermediate number, it will have a binary representation which would indicate certain  $X$ 's as a 1 and certain  $X$ 's as 0. We will assume that we start from the left. So this is the solution with  $X_2$  equal to 1,  $X_4$  equal to 1 and  $X_5$  equal to 1.

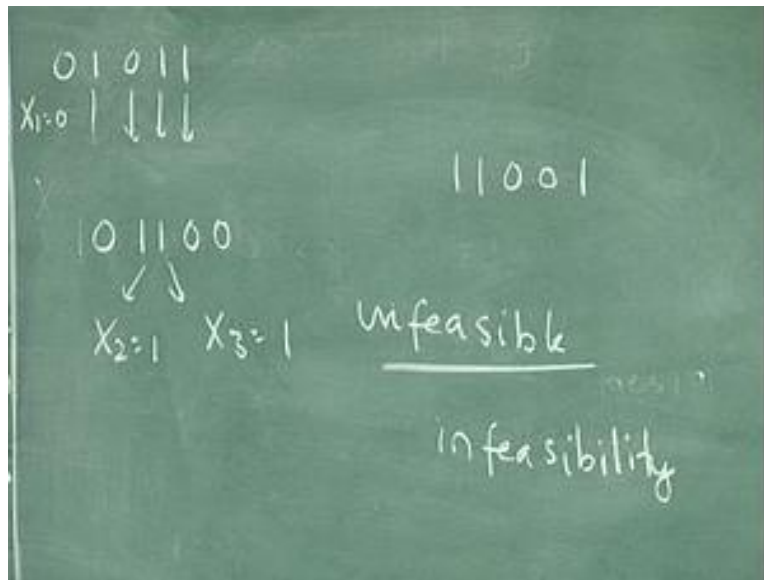
Let us go back and check whether the solution is feasible in the first place.  $X_2$  equal to 1,  $X_4$  equal to 1,  $X_5$  is equal to 1 is not feasible because we have 1 plus 3 is 4, 4 minus 4 is 0, 0 is greater than or equal to 2; it violates this constraint.  $X_2$  equal to 1,  $X_4$  equal to 1,  $X_5$  equal to 1 does not violate this constraint.  $X_2$  equal to 1 does not violate this constraint. So, this solution is infeasible because it violates the first constraint, we will keep it as it is. The second thing, let us take a solution  $X_5$  equal to 1 which is 0 0 0 0 1. If you take a solution  $X_5$  equal to 1, it satisfies this, it satisfies this, it satisfies this. So that, even before we look at this, because our standard problem is a minimization problem with non-negative coefficients in the objective function,

because of that if 0 0 0 0 0 is feasible, then it is optimal. So that is the first thing.

Now having understood that, let us look at  $X_5$  is equal to 1. Now,  $X_5$  equal to 1 is feasible with  $Z$  equal to 19. Every feasible solution to a minimisation problem provides an upper bound to the objective function. The optimum value of the objective function, now, can only be 19 or less, cannot be more than that. If you take this feasible solution 0 0 0 0 1 and let us say I am evaluating another solution from this. Suppose I am evaluating a solution 1 1 0 0 1. Two things can happen: one is this may be infeasible; the second and more importantly, this solution is not going to help us at all because we have non-negative coefficients in the objective function. If you take any existing feasible solution and try to put ones instead of zeroes, you will only end up increasing or not decreasing the value of the objective function and therefore you will not do it. Whether it is feasible or not it is immaterial, it will only have objective function value higher than this 19; higher is a general statement, not lower is the correct one. It will be either 19 or above and therefore you will not do this. The moment you identify a feasible solution, you will not try to evaluate any other solution where a zero in a feasible solution is made into a 1. In this case, there are four zeroes here, so effectively you do not evaluate. The moment you get this solution, you will not evaluate another 2, 4, 8 or 15 solutions. You need not evaluate 15 more solutions, the moment you get this solution. You either say that when I have this solution, I do not evaluate 15 more solutions or you say that I can evaluate or I have already evaluated 15 out to the 32 implicitly by evaluating this one.

The 0, 1 algorithm will have what is called an implicit enumeration component and the moment I have this I am implicitly evaluating a 15. It is also in a branch and bound terminology, you will say that this is fathom by feasibility or has got a feasible solution and therefore I do not evaluate another 15 that can result out of this. There is one thing called implicit enumeration by feasibility. If I have a feasible solution, then I can implicitly evaluate a few more solutions depending on how many zeroes exist in your feasible solution. If you have more zeroes, then more solutions you evaluate implicitly. Now, let us look at another case.

(Refer Slide Time: 37:28)



Suppose, look at a situation where I have a solution like this. For example, suppose we have a solution 0 1 1 0 0. Let us say we have a solution like this. This represents  $X_2$  equal to 1 and  $X_3$  equal to 1. Now, when you have  $X_2$  equal to 1,  $X_3$  equal to 1, you realise that this constraint is violated because  $X_2$  equal to 1, this is a minus 7 here and that is all you have. So, this is clearly infeasible. If a constraint is not satisfied here, it means that you should always add a positive quantity to the left-hand side, because all your constraints are greater than or equal to type. It is a minimisation problem with all greater than or equal to type. You have to add something to the left-hand side to make it feasible and that addition is possible only by putting a variable which has a positive coefficient which is currently at 0 to 1. For example, variables  $X_1$ ,  $X_3$ ,  $X_4$  and  $X_5$  are at 0; they have positive coefficients. Right now, the extent of infeasibility is, this is a minus 7 for a 2, which means the extent of infeasibility is 9. I need to add 9 more things to the left-hand side to do that.

What do I do first? I try to put the one which is most positive into it. The moment you understand this, if there is a constraint which is infeasible, then that constraint can be made feasible only by adding or putting to 1, a variable that has a positive coefficient which is currently at 0. Such a variable is called a helpful variable. A helpful variable is one which is right

now at zero. When you put it to one, it helps you at least in one constraint; it helps you towards getting feasibility.

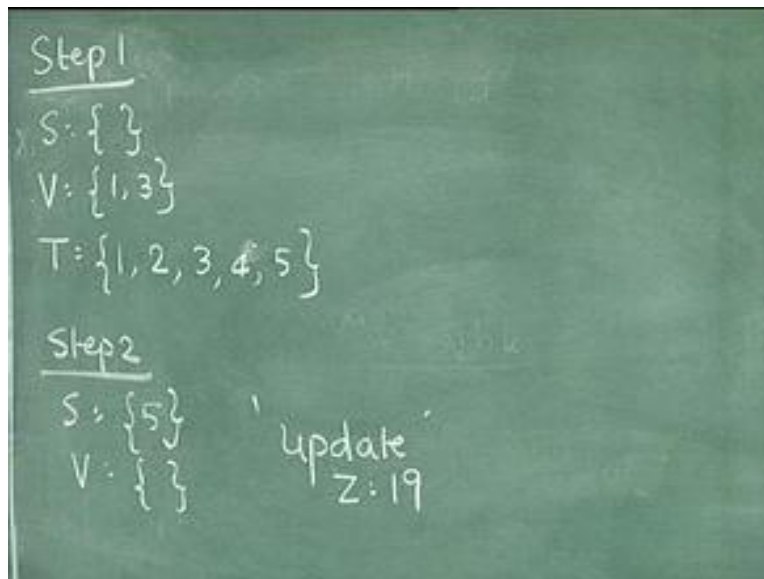
This  $X_5$  is still a helpful variable. Even though putting  $X_5$  to 1 has not given you the feasibility,  $X_5$  is still a helpful variable because it is taking you towards feasibility. If you look at this situation, there are three helpful variables  $X_1$ ,  $X_4$  and  $X_5$ . If you put all the helpful variables to one and it is still not feasible, then any amount of branching from this or moving from this, when you branch or when move, you essentially replace the zeroes by ones, when you move down. When you replace a one by a zero, it is not called branching. We will see that later in much more detail. But the point is if you have a given solution which is infeasible and if you are able to identify a constraint and show that if putting all the helpful variables in that constraint to one, you still cannot get feasibility, as in this example then, I do not have to do it, I implicitly evaluate seven more solutions in this. Because there are seven solutions which can be created by replacing the zeroes into ones and in a single stroke, I know that all these seven will not give me any help. This is called fathoming by infeasibility.

So, let us have a quick recap. The standard problem is a minimization problem which is written in such a form that it has non-negative coefficients in the objective function. It has greater than or equal to constraints irrespective of the coefficients in the left-hand side and irrespective of the right-hand side values. Because of that property that the standard problem has, if the solution 0 0 0 is feasible, then it is optimal; that is number 1. Number 2, there are  $2^n$  possible solutions that have to be evaluated. If you have a feasible solution, then you do an implicit enumeration, that many zeroes that you have in the feasible solution,  $2^n - 1$ , that many solutions you implicitly evaluate whenever you have a feasible solution. You can also do implicit enumeration by infeasibility and when you have an infeasible solution and if you are able to show that for a constraint that is violated if by putting all the helpful variables to 1 you will still get infeasibility to that extent you implicitly evaluate some more. You will use exactly these two results of implicit enumeration and look at an algorithm which does that and tries to get you a solution quickly.

But you should keep in mind at the end which we will ascertain again later that in the worst case,

you will still have to evaluate all  $2^n$  possible solutions. Even though, if it is an implicit enumeration algorithm that we will see, it is a worst case complete enumeration algorithm. You may be forced to evaluate all the  $2^n$ . Let us go back and check and develop this algorithm for the same example and see how we do it.

(Refer Slide Time: 44:31)



We first start with something called step 1 of the algorithm. We define exactly three vectors in this. One is called a solution vector and we begin with a null set. Null set implies that all the variables are at 0. We define another vector which is called V, which is a set of violated constraints. If I put all of them to 0, this constraint is not violated, these two are violated. So 1 and 3 are your violated constraints. Then I define something called T. We will be working only with these three vectors: S, V and T.

T is called a set of helpful variables. Any variable is helpful if it has a positive coefficient in a violated constraint. A variable is helpful if it has a positive coefficient in at least one violated constraint. You do not have to worry about the absolute value of the positive number. Therefore based on that 1 is a helpful variable, 2 is a helpful variable, 3 is a helpful variable, 4 is also a helpful variable, 5 is also a helpful variable. 4 is helpful because of this, 5 is helpful because of

this form. So you have a set of helpful variables which are here.

You can put any one of the helpful variables to 1 and proceed. In order to do that, there are multiple ways, one of which is I take any one of them and put it randomly or otherwise. The other thing that is possible is I will go back to each of these and see. For example if I take 1, you may say 5 plus 1, 6 is a total help that you get out of  $X_1$ ; 2 will give you 1 minus 2, which is minus 1; 3 will give you 3 minus 5 which is minus 2; 4 will give minus 1 and 5 will give 7. You can choose it in anyway. There is no definite rule which says you should take it in this order. But most of the time we end up following what is called a greedy approach when we want to put a variable inside the solution. Now, you can go back and take  $X_5$  because  $X_5$  has the largest contribution towards helping; you do not look at this. For example,  $X_4$  is a very funny thing. You will not go back and say, if I put  $X_4$  there is a minus 4, there is a plus 3, so it is a minus 1. Putting  $X_4$  to 1 is going to force another minus 3 here. You will not consider that at this stage.

When you say helpful variable, you only look at variables that are helpful with respect to the violated constraints. So you can straight away choose any one of them. You could choose 5 because 5 helps more. Let us assume you choose 5, so you go to step 2 and start with  $S$  equal to 5, which means variable  $X_5$  is put to the value 1. The moment  $X_5$  is put to the value 1, you will see that the violated set is a null set because this is satisfied, this is satisfied, and this is satisfied. Therefore, when  $V$  becomes a null set you know that you have got a feasible solution. So you update the solution and say  $X_5$  equal to 1 is feasible with  $Z$  equal to 19. There is no  $T$  because there is no  $V$ ; only when there is an element in a violated constraint, you have a set of helpful variables. So, we have got a feasible solution which is an upper bound. How do we verify that this is optimal or not? We will address it in the next lecture.