Advanced Operations Research

Prof. G. Srinivasan

Department of Management Studies

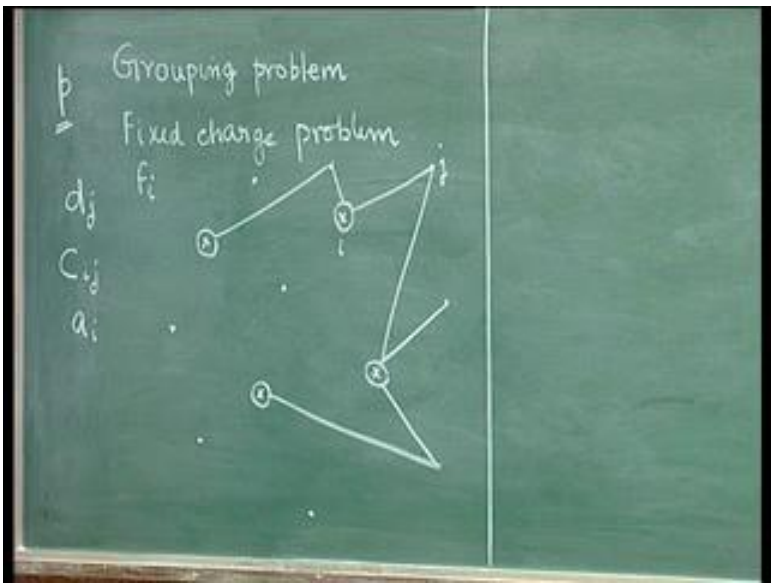
Indian Institute of Technology, Madras

Lecture 12

Integer Programming Formulations

In today's lecture, we will continue our discussion on integer programming formulations.

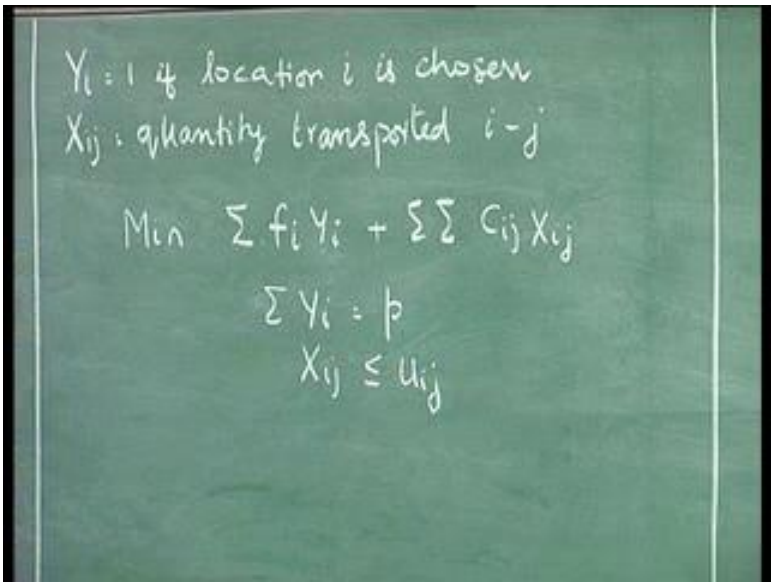
(Refer Slide Time: 00:17)



In the last lecture, we formulated the grouping problem as an integer programming problem. Today, we will begin with something called the fixed charge problem, which is related to the grouping problem but slightly different. Now, the problem is like this. Suppose there are say three or four locations where somebody would like to establish a factory or a warehouse or a distribution center or something like that; so these (Refer Slide Time: 00:54) will be the central nodal points. From this factory or distribution center we have to send items to different places. These will be the customer points; this will be there and so on. There is already an established connectivity saying, for example, this point is connected to this, this point is connected here, this point is connected here, this point is also connected to this, connected to this, connected here connected here and so on.

We will have certain connectivity associated; you have to make sure that every point is connected to at least one of these or more than one of these and so on. For example, we will call these (Refer Slide Time: 01:48) as i, these are the i's and these are the j's, so we will say that there is a requirement of d_j for each of these j's and there is a capacity a_i, we will look at that little later. There is a transportation cost C_{ij} of transporting it from i to j. There could be an a_i, which is the capacity restriction also. The most important thing is that, it is not that you have these plants located in all these four places, what you want to do is, let us say, you want to locate p number of plants, say p could be 2. Let us assume that these are candidate locations for plants and these are the demand points and there is transportation from the plants or the factories to the demand points. There are four candidate points but we want to locate it only in two places, not on all the four. There is the fixed charge f_i of locating something in location i, that is also there. In order to create a facility I have a fixed expenditure which is called a fixed charge; that is why this problem is called fixed charge problem.

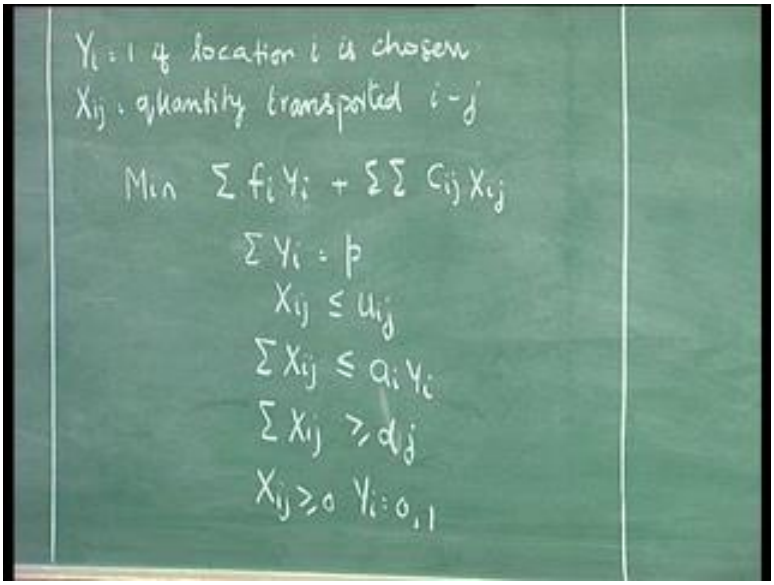
(Refer Slide Time: 03:33)



The variables will look like this; you can have $Y_i = 1$, if location i is chosen and X_{ij} is the quantity transported from i to j. So your objective function for this problem will be to minimize sigma f_i y_i plus sigma C_{ij} X_{ij}. You may say for example, if there is an absolute connectivity every one of the j's is connected to all the i's and so on, a complete connectivity between i and j then you may formulate this problem for a fixed number. For example, you may say sigma Y_i equal to

p implies that I want my factories to be located exactly in p out of the m possible locations; you could have that. At the same time if you do not have this complete connectivity, what you would like to do is, to have a minimum number of locations you will ignore this constraint and let the problem decide the number of Y_i's that it wants to have. You may have this constraint you may not have this constraint depending on the situation. You may also have X_{ij} less than or equal u_{ij}, which is the link. For example, this will be u_{ij}, which is the capacity connected with the arc i-j. I may be able to transport the maximum of u_{ij} between i and j.

(Refer Slide Time: 05:22)

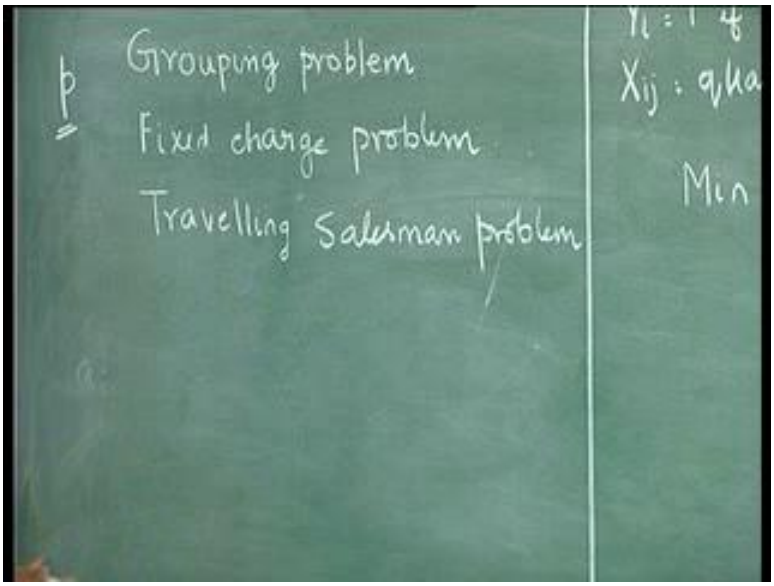


If you do that then it is called a capacitated problem. If you leave out the capacity restriction, then it becomes an uncapacitated problem. For example, the standard transportation problem that you have formulated is an uncapacitated problem. It does not put any restriction on the limit on X_{ij}. All capacity constraints or capacitated problems have constraints which are upper bound to that X_{ij}. So you may have a capacitated problem, you may have an uncapacitated problem. If you have a capacitated problem then you will get this constraint. If it is uncapacitated, you will not limit the X_{ij}, it can be anything. Sigma X_{ij} should be less than or equal to a_iY_i. If location i is chosen, which means Y_i equal to 1, only then the a_i capacity is available for you to be distributed, otherwise it is not. The standard transportation constraint would not have this Y_i. It will simply be sigma X_{ij} less than or equal to a_i, j is equal to 1 to n. Because of the location decision that comes along with the transportation decision, you will have sigma a_iY_i.

If you do not have this capacity constraint a_i and if you assume that the plant can produce an infinite number or whatever is needed, then this will look like sigma X_{ij} is less than or equal to M into y_i, where big M is large and positive and tends to infinity. When the location is chosen, it can produce a maximum big M. If it is not chosen, it can produce only 0, so nothing can come out of it. Sigma X_{ij} is greater than or equal to b_j because what is needed in destination j should be met. All these will work very nicely if there is complete connectivity among the i's and j's otherwise what you can do is, you can leave out those X_{ij}'s for which there is no connectivity; that is one, or you can set that u_{ij} equal to 0 so that the whole thing does not exist, you can do that or you can even go back and put another a_{ij}, which is like an incidence which models an incidence; a_{ij} equal to one if there is a link between i and j, you can do that also and suitably bring that a_{ij} or b_{ij}. You should not confuse with a and b. Call it some c; c is also used, so some number r_{ij} as an incidence between i and j and use it suitably. So, X_{ij} will be enabled only when that r_{ij} is 1 otherwise it will not be enabled; you can do it that way. Easiest thing to do is put all the u_{ij}'s to 0, where you do not have the link; that is the easiest thing and so on.

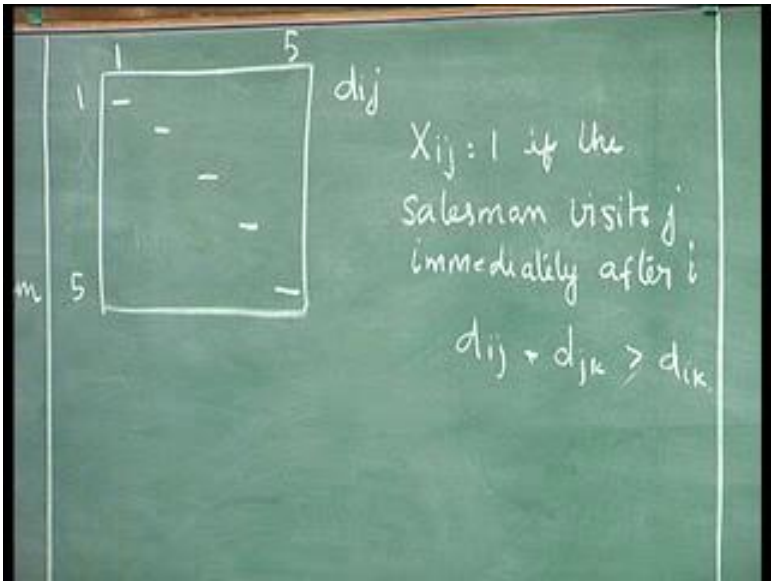
So X_{ij} greater than or equal to 0. What is b_j? b_j is the requirement. What is that d_j? Where is the d_j? I should be consistent. I will call this d_j. So Y_i is equal to 0, 1. This is called a fixed charge problem. This is a very important problem in location, allocation, distribution, supply chain and in all those areas, particularly when you are looking at decisions at multiple levels, for example, you could have factories, you could have distribution centers, you could have warehouses and then retailers. The problem will look like, I would want to make say p_1 out of M_1 factories, p_2 out of M_2 distribution centers, p_3 out of M_3 warehouses, p_4 out of M_4; M_4 is ultimate customer so you will not have that; you will get a series of fixed charge and becomes a fairly complicated problem. This is reasonably similar to the grouping problem but it has additional constraints and restrictions.

(Refer Slide Time: 10:25)



We will look at some more examples of integer programming formulation. You will also take the travelling salesman problem as the next example.

(Refer Slide Time: 10:47)



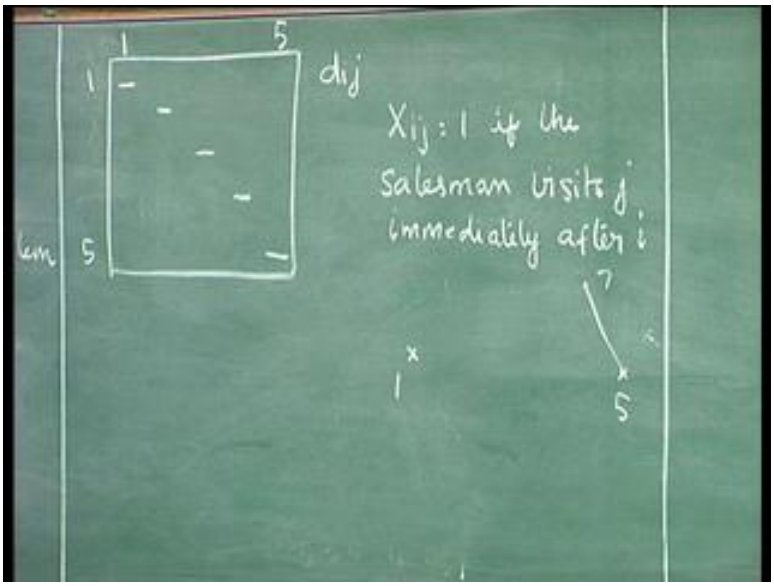
To begin with, let us take a travelling salesman problem, where the distance matrix is given. We can generalize it to n there is a no problem at all, except that we need to consider the case where n is odd and n is even separately. We will right now take five, where n is odd and we will assume

that we have the distance matrix that is given to us. Distance d_{ij} is known. We can assume that it is symmetric but it is not absolutely necessary that it is symmetric in a travelling salesman problem.

Normally, distance between i and itself, any point and itself is 0. But it is customary to define $d_{i,i}$ as infinity in all travelling salesman problems. So, we will put a dash here in all these five places, indicating infinity. We will see why we do that very soon. In all travelling salesman problem you can almost close your eyes and put infinity along the diagonal, you will never have a 0 in the diagonal. Let us define X_{ij} equal to 1 if the salesman visits j immediately after i. There is one more aspect which we will touch upon. Unless otherwise stated explicitly, it is assumed that the travelling salesman distance matrix for any TSP will have to be square because it is distance among the certain number of cities. It will be symmetric because it is under the assumption that you are modeling Euclidean distances unless otherwise you are bringing some other factor as a distance measure in a TSP.

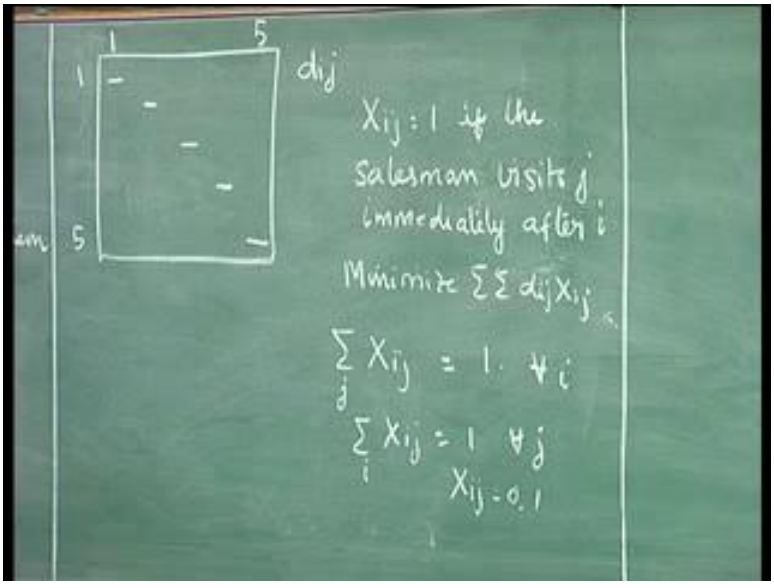
The third important assumption is it satisfies triangle inequality. Triangle inequality means if you take three points i, j, k, d_{ij} plus d_{jk} is greater than or equal to d_{ik}; I think strictly greater than is triangle inequality. So you will have this. This comes from the fact that in a triangle sum of two sides is always greater than the third. When you have this triangle inequality satisfied, it reflects itself in a different form. If you define a TSP, the normal definition of a TSP is a salesman starts from city 1 or any city, visits every city once and only once and comes back. This once and only once comes in because if your matrix satisfies triangle inequality you can prove that the person will visit once and only once. For example, you cannot have a solution where the salesman visits a city twice other than the starting city of course. If you assume the city 1 is the starting city, the salesman will start from city 1, go back complete the circuit and come back to 1. Every other city in a TSP, he or she is expected to visit only once and that comes because of the triangle inequality. If you satisfy triangle inequality, any solution where the person visits a city more than once will always be inferior to a solution where the person visits once and only once. So, that once and only once comes because of the triangle inequality. Distances are positive, distances are more Euclidean; therefore they satisfy this. For example, if you have a distance matrix that does not satisfy triangle inequality, then it is always advantageous for you to go from i to k and k to j instead of i to j. So the person will end up visiting a node more than once.

(Refer Slide Time: 15:00)



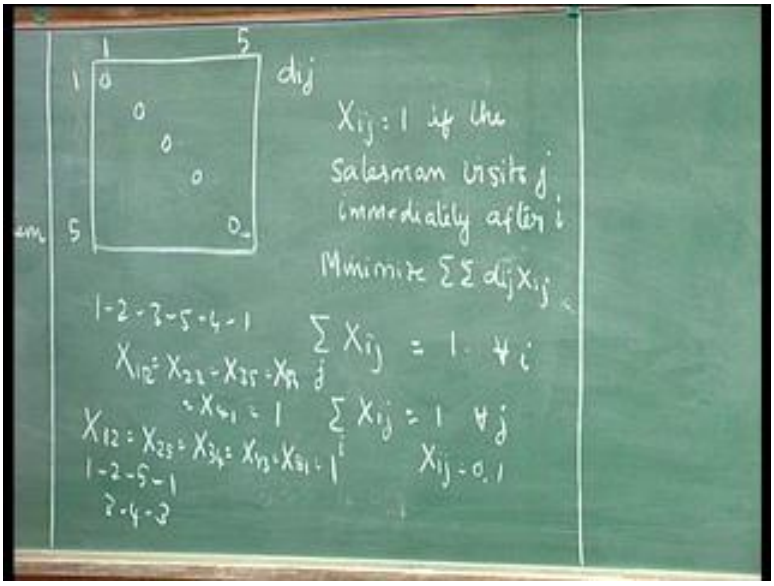
For example, if I want to go from say 1 to 5, if going through 4 is cheaper I will always do that. For example, from 5 to 7 I am going again and if it turns out that going through 4 is cheaper, I will visit 4 twice. But when this condition is satisfied I will never do that. So that is one thing which we will assume unless otherwise stated that the matrix is square symmetric and satisfies triangle inequality, which implies that the person will visit every city once and only once. Let us go back and start formulating. The person has to leave each city once.

(Refer Slide Time: 15:43)



Sigma X_{ij} is equal to 1. From every i, he has to leave, so summed over j equal to 1 for every i; person has to enter every city so sigma X_{ij} summed over i equal to 1 for every j. X_{ij} equal to 0 or 1 and minimize sigma d_{ij} into X_{ij}. Let us assume that we have formulated the TSP this way. Is this answer correct or is this formulation correct? It is not, simply because this is the formulation for the assignment problem. There has to be something more which makes it the TSP, so what is that? Suppose we look at this matrix and we just apply this formulation saying that it is a TSP formulation and let us also assume that we have not done this and we have used that the distance between any point and itself is zero.

(Refer Slide Time: 17:02)

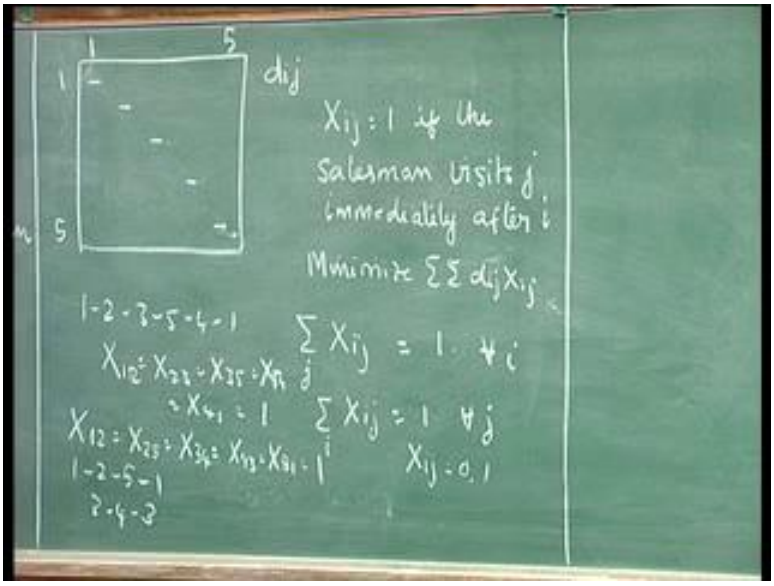


Now if you apply this formulation, what will you get? You will get all diagonal assignments, X_{11} equal to 0, X_{22} equal to 0, z equal to 0. Why is it not a TSP, because it is not a tour. Now number one is if you put 0s here, then your travelling salesman problem for this formulation, which we will refine this formulation later we will give you all diagonal assignments. Each of the diagonal assignment is a subtour. There are two things in a TSP. If I have a 5 by 5, if I have 1, 2, 3 say 5, 4, 1 is a tour; it is a feasible solution to the travelling salesman problem.

For example, this will be reflected by X_{12} equal to X_{23} equal to X_{35} equal to X_{54} equal to X_{41} equals 1. There will be five allocations and this will be the order of allocations, this has a tour. Suppose I have X_{12} equal to X_{25} equal to X_{34} equal to X_{43} equal to X_{51} equal to 1; let us try to get

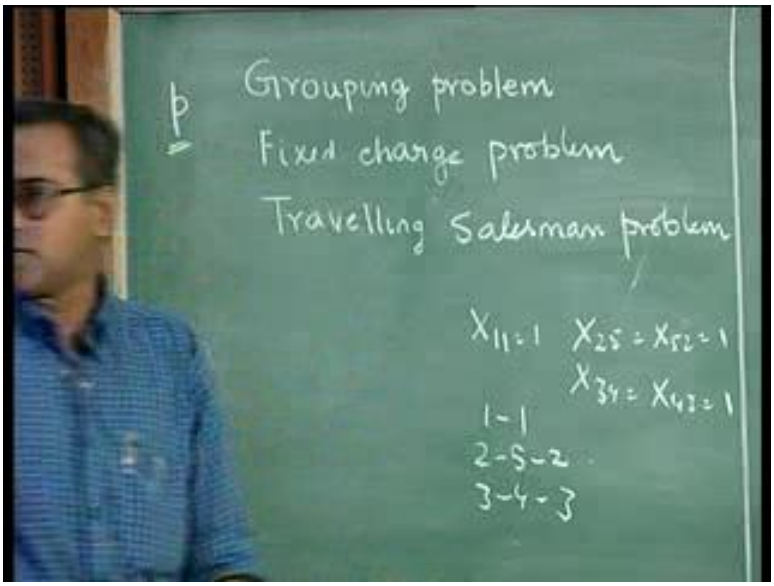
a tour or a subtour from it, then you get 1 to 2, 2 to 5, 5 to 1, 3, 4, and 3. This is not the solution to the TSP, this is the subtour and there are two subtours. A solution to the TSP should not have subtours, so what we need to do is, we need to add what are called subtour elimination constraints into this and if you are able to do that, then we get the solution or the formulation for a travelling salesman problem. So you need to add subtour elimination constraints. For a travelling salesman problem of length n or size n, then you can have subtours of length 1, 2, 3 up to n minus 1. You need to eliminate all these kinds of subtours. Now, every subtour elimination constraint would involve a constraint and you want to minimize constraints in any formulation. Now what you do is you put a dash here, which is like a big M, so that you avoid subtours of length 1.

(Refer Slide Time: 19:35)



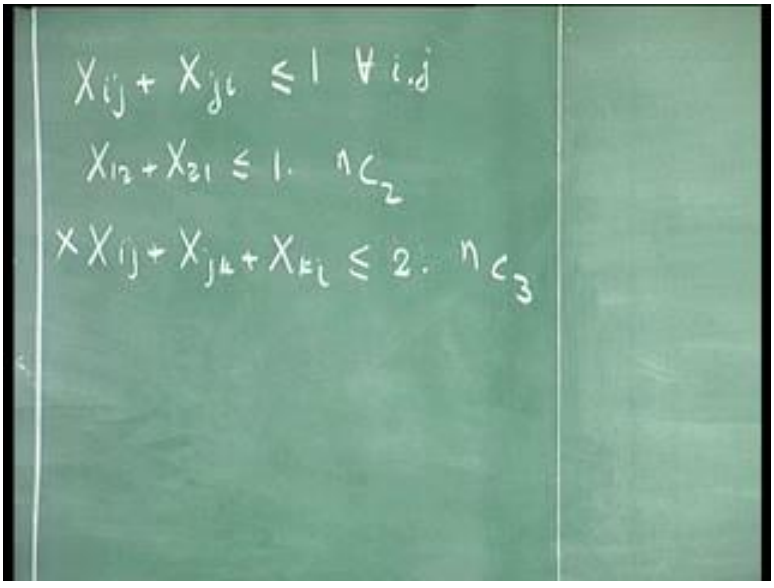
The moment you start putting infinity here or dash here, then X_{ii} will not become an allocation in a TSP. So indirectly you eliminate subtours of length 1 by forcing the diagonals to infinity. That is the reason why you put infinity in the diagonals. You have eliminated subtours of length 1.

(Refer Slide Time: 20:16)



Suppose I have a solution which is like this; X_{11} equal to 1, X_{25} equal to X_{52} equal to 1, X_{34} equal to X_{43} equal to 1. There are three subtours, 1 to 1 is a subtour, 2 to 5, 5 to 2 is a subtour, 3 to 4, 4 to 3; there are three subtours. This is the subtour of length 1, this is the subtour of length 2, subtour of length 2. There are two links so subtour of length 2. There is only one link so its subtour of length 1. Now by forcing the diagonal elements to infinity we have eliminated subtours of length 1.

(Refer Slide Time: 21:02)



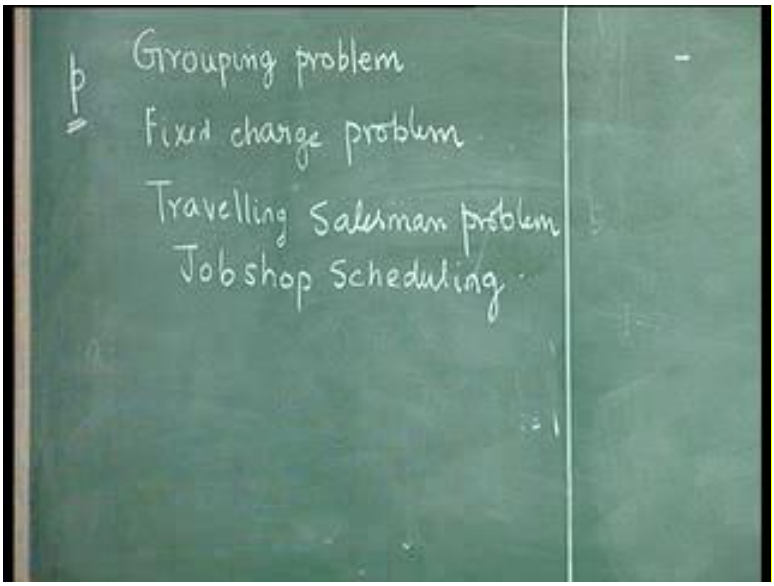
Now you need to eliminate subtours of length 2 and that will be like this; X_{ij} plus X_{ji} less than or equal to 1 for all i and j. For i equal to 1, j equal to 2, actually you can also put i not equal to j so that you can reduce some more constraints. If I look at i_2, I am trying to say X_{12} plus X_{21} less than or equal to 1. What will happen? One of them will have to become 0. If X_{12} is in the solution, X_{21} will not be in the solution. If both are not in the solution it is still acceptable to you. That is why I have put a less than or equal to 1. It may turn out that both 1, 2 or 2, 1 may not be in the optimal solution. Therefore you put a less than or equal to 1 or in general for a subtour of length k, you put k minus 1. This will take care. The only problem is there are too many constraints. There are nC_2 or nC_2 minus n if you leave out that i not equal to j, and you put nC_2-n, that many constraints you will have to put. If you add all these nC_2, you have now eliminated the subtours of length 2. Go back and look at subtours of length 3. When I have to eliminate subtours of length 3, then I have to put X_{ij} plus X_{jk} plus X_{ki} less than or equal to 2, which means I have nC_3 constraints because i j k can be chosen in nC_3 ways. I will have another nC_3 set of constraints and I will have another nC_4 set of constraints.

The number of constraints becomes exceedingly large, but let us try to do something else. Now, let us go and back look at this three constraints again. I have a TSP with five cities. Let us assume I have a subtour of length 3. If I have a subtour of length 3, the remaining two cities which are not included in my subtour can either form a 2 city subtour or two 1 city subtour. It cannot form anything else; therefore, if I eliminate all 1 city subtours and 2 city subtours, I am automatically eliminating all 3 city subtours. Is that clear? I will repeat again. If I have a 3 city subtour in a 5 city TSP, then the remaining 2 cities can become either a single subtour of length 2 or 2 subtours of length 1 each, no other possibilities exist. Therefore, if I eliminate all 1 city subtours and all 2 city subtours, I automatically eliminate all 3 city subtours, which I have already done. I do not have to put this nC_3. Similarly, if I have a 4 city subtour there is only one remaining and that has to be a 1 city subtour.

I have already eliminated 1 city subtours, so I do not have to do the nC_4. So in general, if I have n city travelling salesman, problem where n is odd it is enough if I do till n minus 1 by 2. I do not have to do anything beyond. If n is even, I will do up to n by 2. When I have 6-city TSP I have to necessarily also eliminate the 3 city subtour, because a 3 can result in another 3. It can also result in another 3, so you do it up to n by 2, when n is even and n minus 1 by 2, when n is odd. You

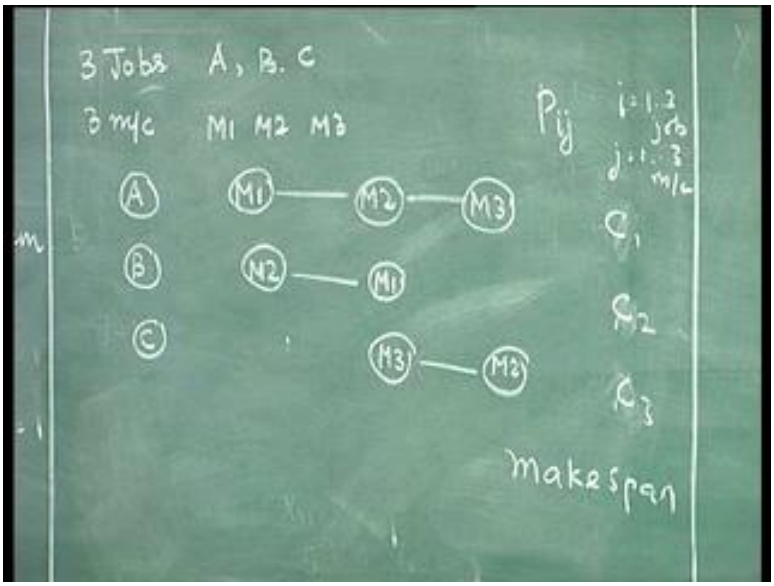
will have nC_2 plus nC_3 plus dash, dash $nC_{n-1/2}$ constraints if n is odd in addition to this 2n and nC_1 plus nC_2 up to $nC_{n/2}$, when n is even in addition to these. You will realise that the number of constraints itself is exponential, because $nC_{n/2}$ or n minus 1 by 2 is exponential. Number of constraints itself is exponential. Much later in the course when we look at TSP, we will at least see that this nC_1 up to nC_n minus 2, which is an exponential number, can be replaced by a set of n constraints or n square constraints. We will see that little formulation much later, but right now the understanding is there will be whole set of subtour elimination constraints, which we have to make. Much later we will look at that formulation where we will have n square or nC_2 or whatever number like that. nC_2 constraint which is capable of modeling the entire thing, that will be a more efficient formulation of the TSP. The understanding here is to add this subtour elimination constraint; also the understanding is that we do it systematically, so you can stop up to n minus 1 by 2, when n is odd and n by 2 when n is even. We will look at another formulation now. We look at job shop scheduling problem.

(Refer Slide Time: 26:54)



Let us say there are three jobs; there are three machines. You call the jobs A, B and C, you call the three machines M1, M2 and M3.

(Refer Slide Time: 27:04)

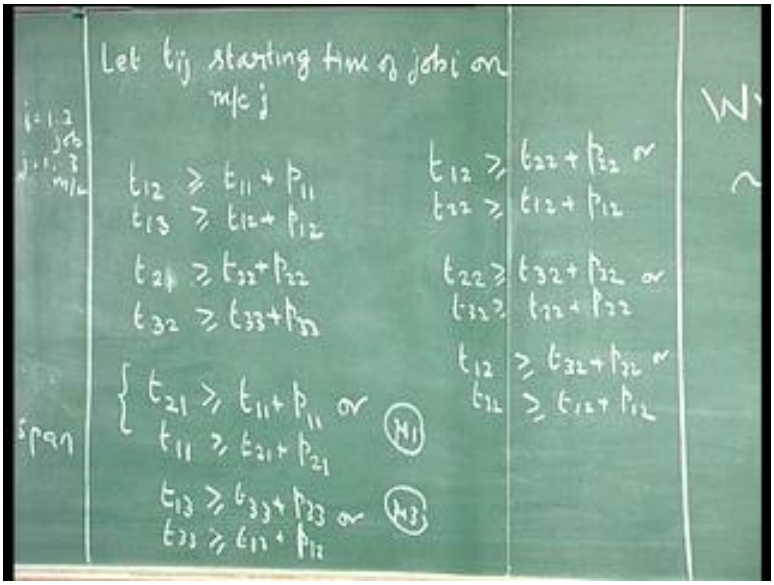


Now let us say job A will have M1, M2 and M3. This will be the route. Job B will have M2 and M1 and say job C will have M3 and then M2. The associated processing time will be P_{ij}, where i equal to 1 to 3 represents job, j equal to 1 to 3 represents machine. For example, this will need P_{11}, P_{12}, P_{13}, P_{22}, P_{21} and so on. This is called the job shop because the order of visit is different for different jobs. If for example, we say that all the jobs will first visit M1 and then M2 and then M3, then it becomes something called a flow shop, where all the jobs have the same order of visit. If the route or order of visit depends or is different for different jobs then it comes under a general job shop scheduling problem. All the jobs are available at time equal to 0. When we start processing, each job will be completing at a certain time. For example, let us say all these three will be over if they are over by certain time, say A_1 and this is A_2 and this is A_3 or you call it some capital C_1, capital C_2, capital C_3.

What I want to minimise is called the makespan, which is the time at which all the jobs are over. It is something like all the three jobs are to be done for the same customer. When all the three jobs are finished I can pack them and send it to the customer. So I want to send it to the customer at the earliest and this is what I want to minimise. I want to find out how I schedule these jobs in the system such that I minimize this. C is the time at which this is over. C is the time at which this is over; this is not given to you C_1, C_2, C_3 are not known. C_1, C_2, C_3 will have to be found.

The objective is to try and minimize the make span, which is the time at which all the jobs are ready and over. That is the objective. Now, let us formulate this problem.

(Refer Slide Time: 30:23)



Let t_{ij} be the starting time of job i on machine j. For example I take job A. For this certain things are very clear. t_{12} from 1 it goes to 2, so t_{12} is the start time of job A on machine 2. t_{12} has to be greater than or equal to t_{11} plus p_{11} because job A first visits machine 1 and only then it has to visit machine 2, which means it should have completed its work in machine 1. If t_{ij} is the start time on machine 1 for job 1, then t_{11} plus p_{11} is the time it is over and it can go to machine 2 only after this is over; this is very straight forward constraint. Similarly t_{13} is greater than or equal to t_{12} plus p_{12}, which is again a very straight forward constraint. As far as this job is concerned t_{21} is greater than or equal to t_{22} plus p_{22}. That is the only thing I have and as far as this is concerned, t_{32} is greater than or equal to t_{33} plus p_{33}.

For example, I can finish my stuff with M1. Any machine can process only one job at a time that is an assumption. So, I may have to wait to get this M2. My start time t_{12} need not be immediately after this is over. This could be over, it could be waiting for machine 2 and machine 2 may be busy doing something else, it waits and once the machine 2 is free, it can go, which means t_{12} is greater than or equal to t_{11} plus p_{11}. These are simple constraints which are easy to model. The difficulty comes here; if I take machine M1 then machine M1 now has job A and job

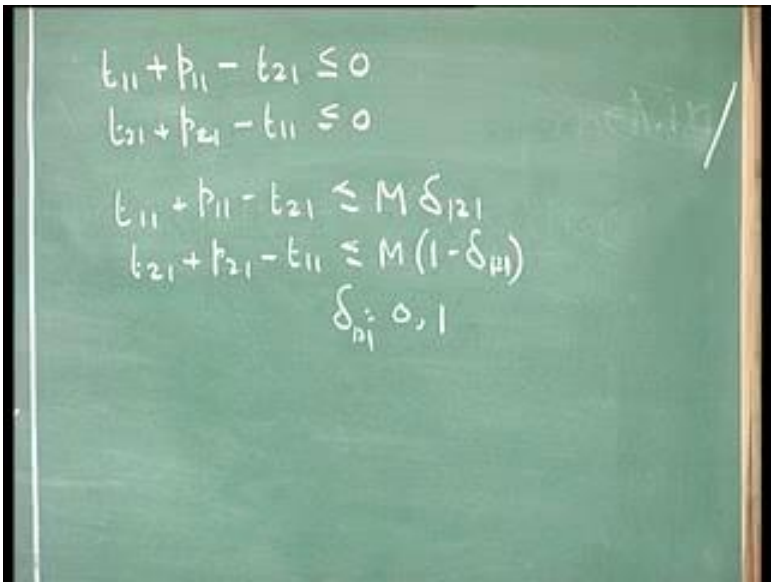
B going into machine M1. I do not know which one is going to go first. I have to decide such that it is advantageous to me. What I will do is, if I take a machine M1, either job A can go first and then job B can go or job B will go first and then job A can go. You get into an either or situation. If you take M1, if job A goes first and then job B goes, then it means that t_{21} job B going into machine 1, will have to be greater or equal to t_{11} plus p_{11} or t_{11} is greater than or equal to t_{21} plus p_{21}.

You have an either or constraint that comes in which we have not modeled yet in our linear programme. We always know how to model the rigid constraints but not an either or constraint. Similarly, if you take M3 it is either A and C. This is for M1. For M3 you will get a similar either or constraint which will be like this. M3 has A and C, so you will have either t_{13} is greater than or equal to t_{33} plus p_{33} or t_{33} is greater than or equal to t_{13} plus p_{13}. You take M3, A and C; so jobs 1 and 3. Starting time of job 1 on machine 3 should be after the completion time of job 3 on machine 3, which is t_{13} is greater than or equal to t_{33} plus p_{33} or the other way. Only one of them is valid. Only when both are valid you can do the addition, etc. Only one of them is eventually valid. We will see how we model that.

If you take machine 2, you get into a much more different situation because M2 is handling all the three jobs. You have to write three sets of either or. For example, if you take the pair AB, then either A is ahead of B or B is ahead of A. If you take the pair BC, then B is ahead of C or C is ahead of A. If you take the pair AC, either A is ahead of C or C is ahead of A. You will have three pairs of either or constraints. Since we are doing it first time, we will write down all the three pairs. For AB you will have t_{12} is greater than or equal to t_{22} plus p_{22} or t_{22} is greater than or equal to t_{12} plus p_{12}, this is for AB. Now for BC, t_{22} is greater than or equal to t_{32} plus p_{32} and t_{32} is greater than or equal to t_{22} plus p_{22}, this is one pair. For 1 3 you will have t_{12} is greater than or equal to t_{32} plus p_{32} or t_{32} is greater than or equal to t_{12} plus p_{12}, you have another set.

In general, if all the jobs go through all the machines, then you will have M_n pairs of constraints. How do you model this? This is where the integer programming comes in. We just look at one of these; we can similarly model the rest of them. If you take one pair, what you first have to do is to write this in this form.

(Refer Slide Time: 38:02)

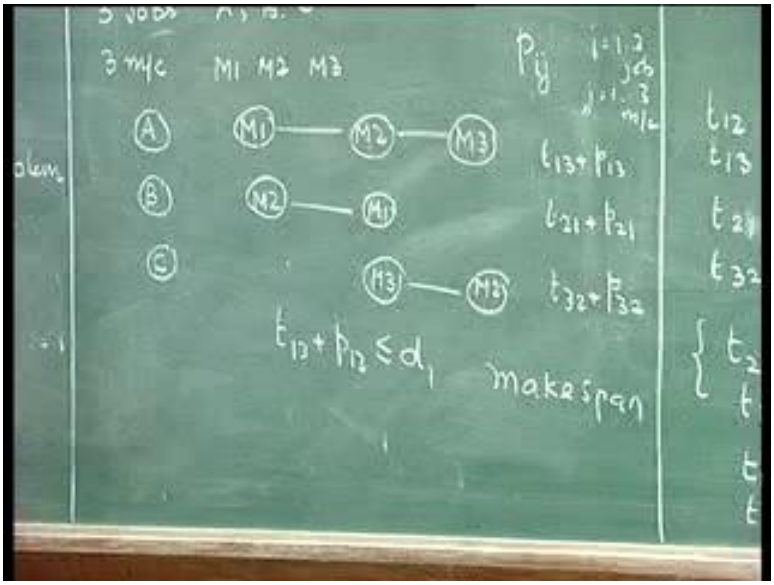


If you take the first constraint this is written as t_{11} plus p_{11} minus t_{21} less than or equal to 0, t_{21} plus p_{21} minus t_{11} less than or equal to 0. The first thing you have to do is to write them as less than or equal to 0. Then you write this as t_{11} plus p_{11} minus t_{21} less than or equal to M into $delta_1$ and t_{21} plus p_{21} minus t_{11} is less than or equal to M into 1 minus $delta_1$. Initially let us call it only delta. Let us not give any subscript to that delta. Later we will do that. This will become M delta and M into 1 minus delta, where M is the well known M; large, positive and tends to infinity, the big M and delta is the 0, 1 variable. Now what will happen? If this constraint is binding, that is, if it is advantageous for you to start 2 after you complete 1 in your final solution, if this constraint is binding, then this is the one that is binding; this is the one that is binding, so your delta will take a value 0. If this is binding, delta will take the value 0 implies this is redundant 1 minus delta is 1. It is a redundant constraint, less than or equal to M is the redundant constraint.

On the other hand if the other one is advantageous to you, then delta will take the value 1 to make this binding and make this redundant. For every pair you will introduce the delta. Every either or situation will have two constraints and a delta which is a 0, 1 variable. You will have to introduce as many deltas as the number of pairs that you have. To give an identity to this delta you can call this delta, say $delta_{121}$ because this represents jobs 1 and 2 on machine 1. For example, this constraint was written for machine 1 for jobs 1 and 2. You can call this as $delta_{121,}$ this will be $delta_{121,}$ $delta_{121}$ as the 0, 1 variable. Similarly, you can name every one of these

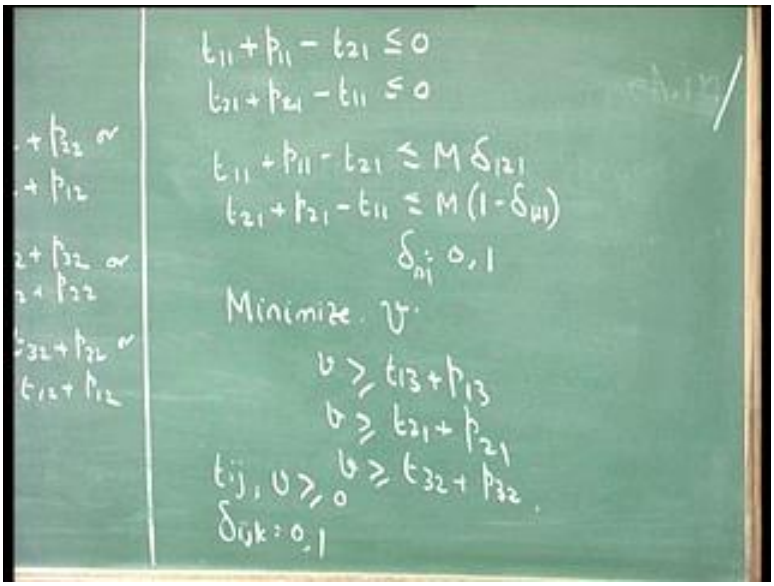
deltas. This will become in this case 1 3 3 and this will have the 1 2 2, 2 3 2, 1 3 2. For every pair you will have a delta.

(Refer Slide Time: 41:18)



Suppose for some reason or other if I put a limit on the completion time of each one. I say that job A has to be over by a certain due date, which is called d_a. Each has a due date or something like that and that is a very simple constraint. Completion time of job A is nothing but t_{13} plus p_{13} should be less than or equal to d_1. Now I have to minimize the make span, so I have the three completion time.

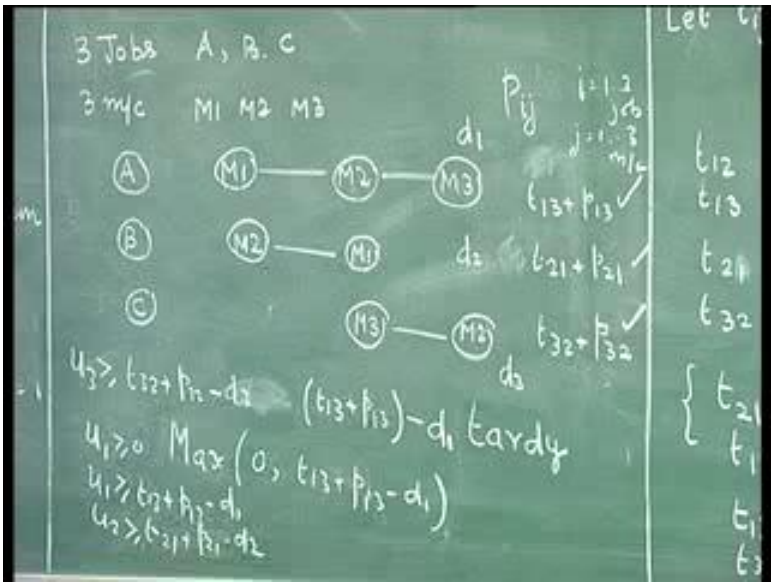
(Refer Slide Time: 42:09)



For this the completion time will be t_{13} plus p_{13}. This will be t_{21} plus p_{21}. This will be t_{32} plus p_{32}. These are the completion times of the three jobs A, B and C, respectively. What do I want to do? I want to minimize the make span, which means I am minimizing the maximum of the 3 numbers. So I end up writing minimize some v and v is the maximum of all of them. So what will happen? v is greater than or equal to t_{13} plus $p_{13,}$ v is greater than or equal to t_{21} plus p_{21}, v is greater than or equal to t_{32} plus p_{32}. You do not have to necessarily restrict the t's to integers or p's to integer; depends on the whole thing. It is likely that if the processing times are integers, then all the t's will also be integers. Right now you do not have to worry about any of them. You will just say t_{ij}, v greater than or equal to 0, $delta_{ijk}$ equal to 0, 1. This is the formulation for the job shop scheduling problem.

What is d_1? d_1 is called the due date for each job. Due date is something like a due time before which you have to complete the job. So if something like a d_1 is given to you, then you will add constraints saying completion time of the job 1 should be before the d_1. Completion time of the job 1 is start time on machine 3 plus processing time on machine 3. So this will be a due date related constraint.

(Refer Slide Time: 43:59)



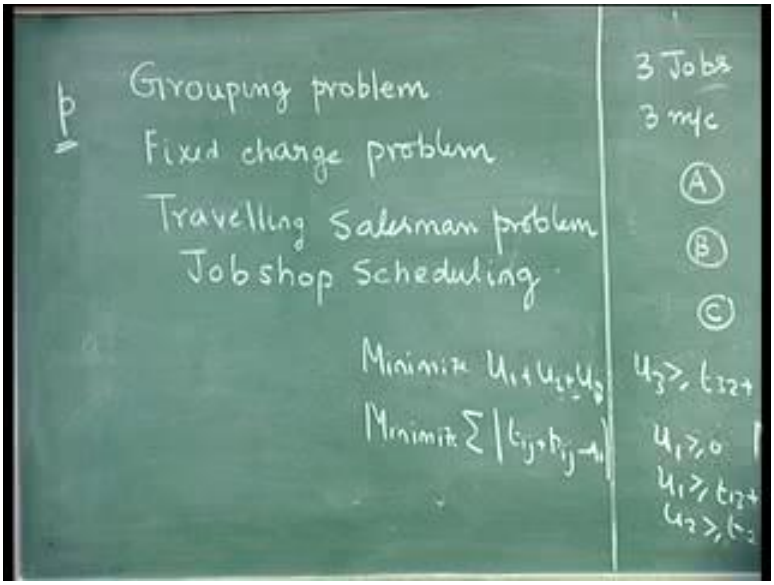
If I do not want to minimize this make span, instead I say that I give you a due date for each one of them, d_1, d_2 and d_3. I am not going to force you to say that you have to complete before $d_1$or d_2 or d_3. I am going to say I want you to minimise the total delay. That is a very reasonable objective. In that case you assume that these three jobs actually go to three different customers and each job has a certain due date given by the customer. Ideally you would like to meet that due date. If you are not able to meet the due date, you would at least like to minimise the total delay that is there, from your point of view. This constraint will not be there. You are not going to be forced to complete within the due date. Then you need to look at one thing. We have these three different completion times and you have associated due times with each one. We call them d_1, d_2 and d_3. We have already written here. Now the delay in scheduling sequencing terminology is called by different terms. If this completion time is before the due date then this job is called early, it is completed early. If this is after the due date then the job is called tardy. So a job can either be early or tardy.

The term late is a very generic term. Lateness is the difference between the due date and the completion time. If the lateness is negative it means the job is completed early. If the lateness is positive, then the job is behind and then it is called tardy. Now what do you want to minimize? Right now you do not want to minimise the earliness. You are interested only in minimizing the tardiness. So what you have to do is you have to define tardiness for each one of these. What

will be your tardiness? Tardiness will be say d_1 minus t_{13} plus p_{13} or minus d_1, this will be your tardiness. This is the completion; this is the due date and if this difference is positive, then it is tardiness. If this difference is negative then tardiness is 0.

The actual tardiness is maximum of 0, t_{13} plus p_{13} minus d_1, this is the tardiness. If the job is early, this is negative and then your tardiness is 0. What you have to do now, is you have to define u_1 greater than or equal to 0, u_1 greater than or equal t_{13} plus p_{13} minus d_1. u_1 greater than or equal to 0 is anyway a non negativity constraint. You do not have to explicitly state this into the formulation as a constraint. You will just define this and similarly u_2 is greater than or equal to t_{21} plus p_{21} minus d_2, your u_3 will be greater than or equal to t_{32} plus p_{32} minus d_3. You will also have u_1 greater than or equal to 0, u_2 greater than or equal to 0, u_3 greater than or equal to 0.

(Refer Slide Time: 48:36)



Now your objective function will be to minimize u_1 plus u_2 plus u_3. You will get this. So depending on the objective function formulation becomes different. If you want to minimize not only the sum of tardiness, but you also want to minimize the sum of earliness as well as tardiness then you will end up doing minimize sigma modulus say some t_{ij} plus p_{ij} minus d_i. There are certain situations where even earliness is not desirable. What will happen is, if you finish the job early you would like to send it to your customer. Customer may not want to receive it, he will say that I have put my schedule in such a way that this item is going to go to production only

tomorrow. It is enough if you send me tomorrow or tomorrow morning; do not send it today morning or two days earlier. I do not want to hold that inventory in my possession. Those are issues related with operations; there are situations. Tardiness is much more undesirable than earliness but there are problems where this is important. We are only interested in modeling, so you will get something like this. Difficulty again will be you have to write this explicitly, because this can be positive or negative and then you have to handle it accordingly. Modulus is not directly a linear thing, so again you have to convert this modulus into another linear thing.

These are the issues. The understanding, particularly from the job scheduling is this "either or"and that is the most important thing. The "either or" constraint will keep repeating again and again in every integer programming formulation, particularly when you are looking at processes and manufacturing related issues where a job or a processor can handle only one job at a time. The moment it can handle multiple jobs you do not have to worry so much, but then if it can handle a maximum of k jobs then you have to put another constraint y_1 plus y_2 plus y_n less than or equal to k and y will be 0, 1. All these are issues in formulating examples. There are still one or two more examples that we will look at in the next lecture.