Advanced Operations Research

Prof. G. Srinivasan

Dept of Management Studies

Indian Institute of Technology, Madras

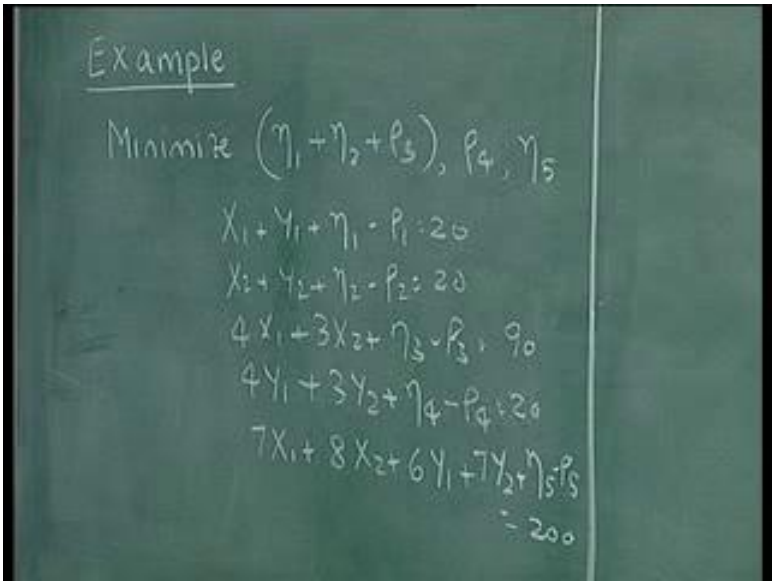
Lecture- 10

Goal Programming Solutions

Complexity of Simplex Algorithm

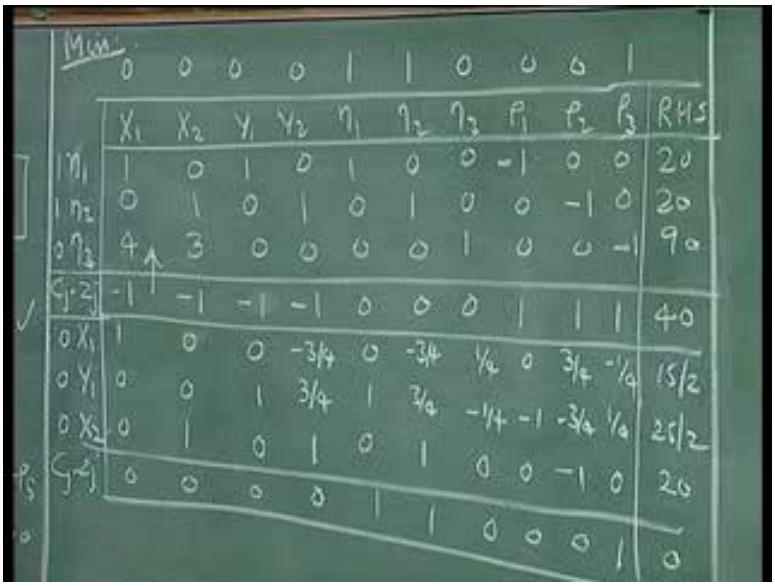
In the last lecture, we were solving goal programming problems using the graphical method for those problems that had only two decision variables and the rest of the variables were the deviation variables.

(Refer Slide Time: 00:30)



Today we will take an example that has more than two decision variables and more deviation variables and try to solve this using a simplex algorithm. The objectives are minimize eta_1 plus eta_2 plus rho_3 and then rho_4 and eta_5, subject to these set of constraints because, eta_1, eta_2 and rho_3 are the first part of the objective function. The objective function has three parts. The first part contains these three, which implies that these three constraints are the rigid constraints. We start the simplex algorithm by considering this objective function and only the rigid constraints to begin with. We can set up the simplex table like this.

(Refer Slide Time: 01:24)



This table will have X_1, X_2, Y_1, Y_2, eta_1, eta_2, eta_3, rho_1, rho_2, rho_3 and a right hand side. We can start the simplex table this way. Then we write this X_1 plus Y_1 plus eta_1 minus rho_1 equal to 20. X_2 plus Y_2 plus eta_2 minus rho_2 equal to 20. 4 X_1 plus 3 X_2 plus eta_3 minus rho_3 equal to 90. We could start with this and we want to minimize eta_1 plus eta_2 plus rho_3. So this is what we want to minimize. This is how your first simplex table will look like. Because every constraint is an equation and every constraint has an eta or a rho, simply starting with eta_1, eta_2, eta_3, the eta's will always qualify to be initial basic variables and eta's will have an identity matrix associated among themselves. For example, if you can have eta_1 1 0 0, 0 1 0, 0 0 1 you can do that. Then your first simplex table will look like this, with a 1 for eta_1, with a 1 for eta_2 and 0 for eta_3, with value equal to 40 here on the right hand side and then all these are 0s. You calculate your C_j minus Z_j or Z_j minus C_j; you could calculate any one of them. But you should remember that it is a minimization problem and also remember that we have not converted it to a maximization problem. We are solving the minimization problem as it is.

If you evaluate C_j minus Z_j the usual way, you will get 1 into 1, 1. So this will become minus 1, this will also become minus 1, this will become minus 1, this will also become minus 1, this will become 0, this will become 0, this will also become 0. You will get a plus 1 here, you will get a plus 1 here and you will get a 1 here and 40. It is a minimization problem; you have calculated C_j minus Z_j. So for a minimization

problem what will happen is, a negative C_j minus Z_j will enter. You could enter either this or this or this or this and so on and may be you can start with this and then compute your thetas and proceed in your simplex iteration, till you reach the optimum solution.

For want of time and because of our own familiarity, we will not go through the simplex iterations. We will only give you the optimum table and then see something interesting from the optimum table. The optimum table alone will look like this. This will actually happen after three iterations and you will finally get something like this. You will have X_1, Y_1, X_2, with 0 0 and 0. You will have 1 0 0; Y_1 you will have 0 1 0, X_2 you have 0 0 1, Y_2 minus 3 by 4, plus 3 by 4 and 1, eta_1 0 1 0, eta_2 minus 3 by 4, 3 by 4, 1, eta_3 1 by 4, minus 1 by 4, 0. Rho_1 will be 0, minus 1, 0; rho_2 3 by 4, minus 3 by 4, minus 1 and rho_3 minus 1 by 4, 1 by 4, 0; 15 by 2, 25 by 2 and 20 with Z equal to 0.

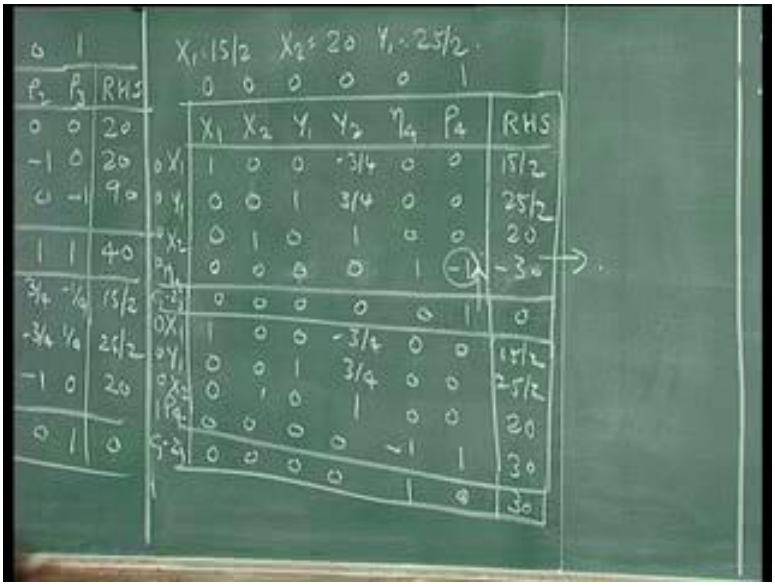
The C_j minus Z_j values will be 0 0 0. For Y_2 it is also 0 because, all this will be 0; this will be 1 1 0 0 0 1. This is all you will get because, now all your basic variables have 0, so automatically the C_js will repeat. There is no negative C_j minus Z_j, the present solution is optimal. Basically, the purpose of solving the first problem is to verify that there is a feasible region associated with the rigid constraints. If there is no feasible region associated with rigid constraints, then you will not have the 0 coming here. The presence of the 0, the optimal value of the objective function being 0, indicates that, there is a feasible region. If there is no feasible region, then you will get some other number that comes here. So this happens after three iterations. It turns out it happens after three iterations here, so all three of them go away.

One way to do is to start the problem in the normal traditional simplex way, by looking at the positive slack variables, as the basic variables and then, one could proceed starting from this 40, till you get this 0 and then understand that the first phase of the problem is solved optimal. But if you look at this problem very carefully, you could have started with three other variables which have an identity. You could have started with Y_1 right at the beginning. Instead of eta_1, eta_2, eta_3, you could have started with Y_1, Y_2 and eta_3, which has 1 0 0, 0 1 0, 0 0 1, which have an associated identity matrix. If you had started with Y_1, Y_2 and eta_3, then straight away you will get Z equal to 0 here, because Y_1, Y_2 and eta_3 have 0 contributions. So straight away

your Z will be 0 here and because they have 0 contributions, you would have the same C_j minus Z_j appearing. Such a thing is possible only when you look at the problem in little more detail and try to choose the correct set of basic variables. If you normally solve the problem the way you are used to, you would end up taking the slack variables or the positive deviation variables as your starting basic variables and then end up performing three or four iterations till it comes. So either you can do this or you could have just straight away said I will start with Y_1, Y_2, eta_3 to get Z equal to 0.

Now that we have got this, let us take this solution and proceed further. We assume that, we started with the actual thing and then we have solved this, we have got this optimal solution for the first phase and from this we will proceed. The 0 indicates we are doing all right, which means we have a feasible region which is symbolized by the Z equal to 0.

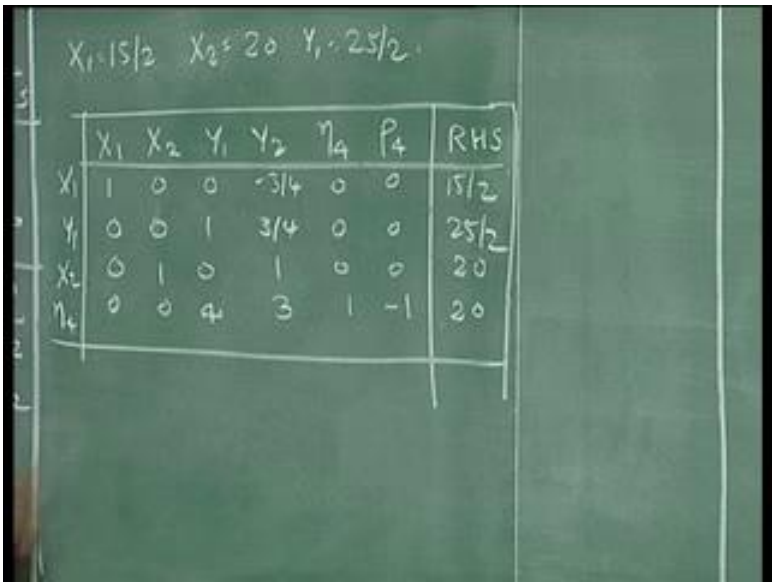
(Refer Slide Time: 10:25)



Look at this solution now; the solution for you is X_1 equal to 15 by 2, X_2 equal to 20 and Y_1 equal to 25 by 2. The rest of them are all zeroes, which means, your eta_1, eta_2, eta_3, rho_1, rho_2, rho_3 are all zeroes here. Remember that you have already solved up to this. These things are non-basic and they are not going to appear in your solution. They do not appear in the subsequent constraints. The eta_1, eta_2, eta_3, rho_1, rho_2, rho_3 do not appear in the subsequent constraints and they are at 0 in the iteration. You can eliminate all these variables from now on. So you can eliminate eta_1, eta_2, eta_3, rho_1,

rho_2, rho_3; you can do that and then you need to keep the Y_2 because Y_2 appears later. You cannot eliminate Y_2. Right now, Y_2 is at 0 but you cannot eliminate the variable Y_2 because it appears in the subsequent constraint. So you will keep this. Your next table will only have this portion.

(Refer Slide Time: 11:46)

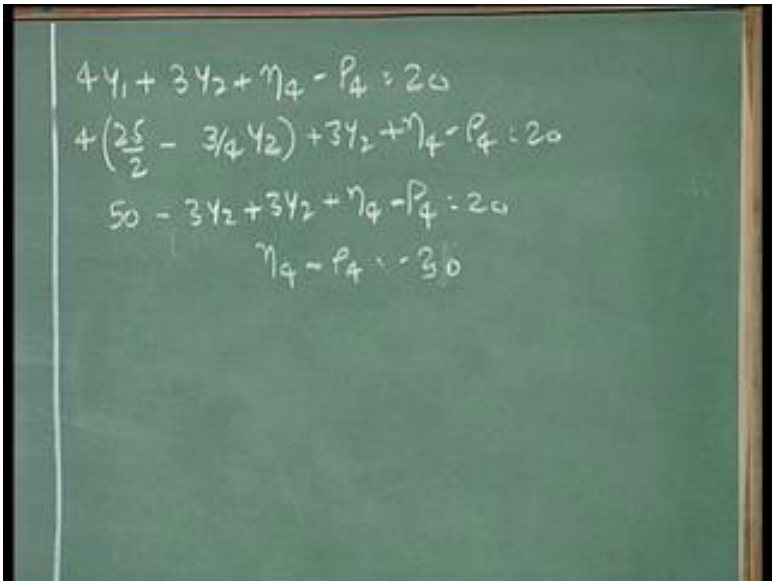


It will start with X_1, X_2, Y_1, Y_2. It will not have eta_1, eta_2, eta_3, rho_1, rho_2 rho_3, all of them are non-basic and they do not appear; they are at 0. Therefore your minimize eta_1 plus eta_2 plus rho_3 is 0. You look at the next constraint. The next constraint contains eta_4 and rho_4. So just add them, eta_4 and rho_4 and then put your right hand side values here. Now repeat this portion of the table into that, at the same time you have to add this also. Repeat this; somewhere you may have five constraints; so you will have to have a five row simplex table right. You have taken the first three, so repeat from this table, whatever is relevant here. So X_1, Y_1 and X_2 and you will repeat 1 0 0 minus 3 by 4, 15 by 2. Y_1 has 0 0 1, 3 by 4, 25 by 2. This is 0 1 0, 1 and 20. You have 0 0 0 0 0 0. Now this constraint has to be written 4 Y_1 plus 3 Y_2 plus eta_4 minus rho_4 equal to 20.

Let us see if we can straight away write it here. You have eta_4, so you have 4 Y_1 plus 3 Y_2 plus eta_4 minus rho_4. Try writing equal to 20. You cannot directly write it because you will lose your identity structure on this. You have to write Y_1 which is

presently a basic variable. You have to substitute for Y_1 in terms of the non basic variable and then retain the identity column for Y_1. What you need to do is this.

(Refer Slide Time: 14:30)



The constraint now becomes 4 Y_1. The constraint is 4 Y_1 plus 3 Y_2 plus eta_4 minus rho_4 equal to 20. Now Y_1 has to be written from this. So 4 times 25 by 2 minus 3 by 4 Y_2 plus 3Y_2 plus eta_4 minus rho_4 equal to 20. This becomes 50 minus 3Y_2 plus 3Y_2 plus eta_4 minus rho_4 equal to 20. This gives you eta_4 minus rho_4 equal to minus 30. The Y_1 becomes 0 here, Y_2 also becomes 0 here; eta_4 minus rho_4 is equal to minus 30, with C_j minus Z_j retained at 0 0 0 and 0 here. Under Y_2 you have a 0. Before that we need to write this; we have 0 0 0 0. The second objective is to minimize rho_4. So you will have a plus 1 here; so you will have 1, these are all 0 0 0 and 0, so you will have 1 coming in here and the right hand side value is 0. There is one more thing before we do this. Remember again, right here at this iteration, this is not a unique optimum. It is not a unique optimum because Y_2 is non basic and can enter and so on. That is precisely the reason why, if you had chosen as I said here, Y_1, Y_2 and eta_3, you would have still got an optimum. That is the alternate optimum; there could be many, that is one of them. So irrespective of what you start it is perfectly fine.

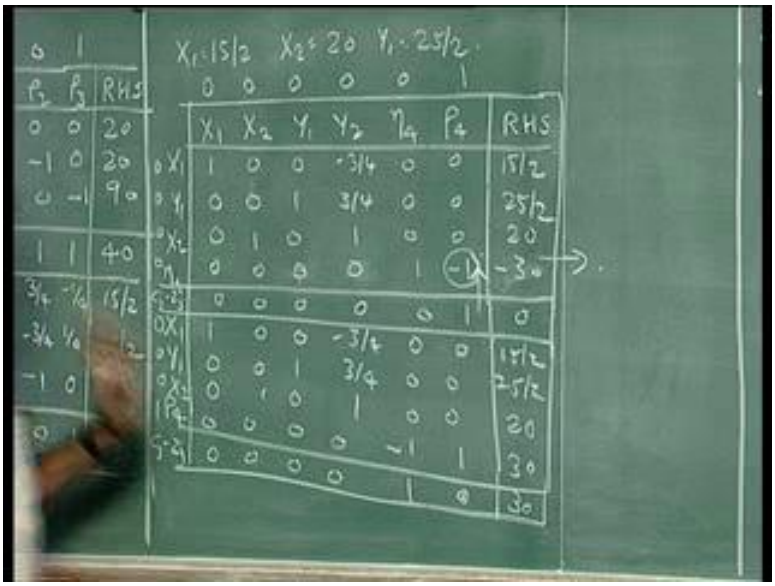
Instead of doing three iterations to get this, if you had started with Y_1, Y_2 and eta_3, you have got an optimum table right here, but you cannot eliminate the variable eta_3 here. Eta_3 would have got a value, so eta_3 column will keep coming again. Only those

that are non basic at that stage and not repeating can be eliminated. So depending on how you do it, you may have the advantage of fewer variables or fewer iteration. It is common sense to take the advantage of fewer iteration than fewer variables, but then you have to be aware of that. If you had chosen Y_1, Y_2 and eta_3, then eta_3 will appear here. That is number one.

Number two is, this solution is optimum already, so you add a constraint here. When you add a constraint, the only thing that will happen is the constraint may be satisfied, the constraint may be violated. If the constraint is satisfied then straight away this would be optimum. What you should do even before writing this is, you should look at the value of X_1 equal to 15 by 2, Y_1 equal to 25 by 2, X_2 equal to 20. Substitute it here. If we had substituted X_1 equal to 15 by 2, it is okay; Y_1 equal to 25 by 2, so, that becomes 50. 50 and 20 here is violated, therefore this constraint becomes binding and you need to do an iteration. Go back to the basics; if I have an optimum solution and I add a constraint, in sensitivity analysis, the first thing you did was to check whether the constraint is satisfied or violated. If it is satisfied, then you will not take the trouble of carrying out another simplex iteration. Only when it is violated you will carry out the simplex iteration and when it is violated it is always indicated by infeasibility of our right hand side but the optimality condition will be satisfied.

Now when you realize that it is violated and then, you go back and write this. You realize now that the feasibility is violated, so you get a minus here, which is infeasible; the optimality condition remains intact. What you should do is a dual simplex iteration on this and try to get the next iteration. Leave out this fellow, now go back and do a dual simplex iteration. Remember again, a dual simplex iteration should have a negative pivot. As there is only one negative pivot here, you do not have to add a theta row; so straight away this is your entering variable and that is your leaving variable.

(Refer Slide Time: 19:50)

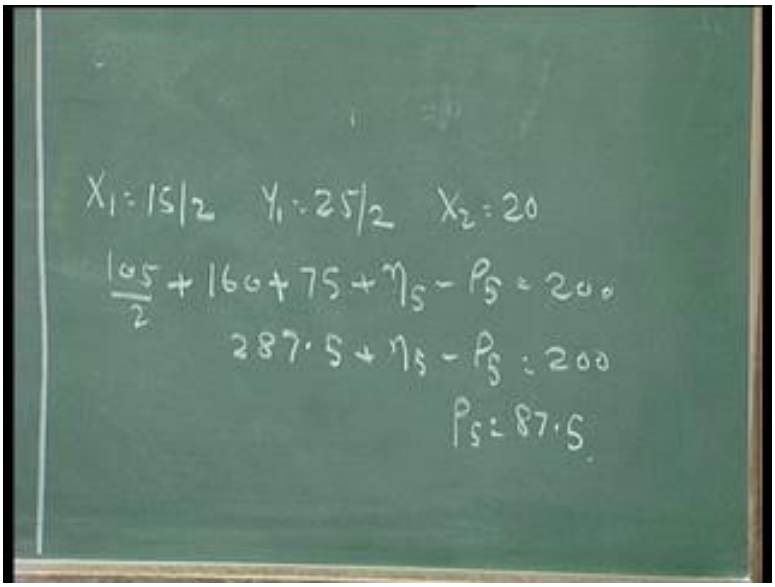


Complete your dual simplex iteration here. With variable rho_4 replacing eta_4 you will have X_1, Y_1, X_2. You now have 0 0 0 and 1. Rho_4 comes in into the basis with 1. What will happen is, first divide by the pivot element here, so you get 0 0 0 0 minus 1, 1 and 30 and luckily for you all of them are 0s here. You just have to write the same thing again. When you write the same thing again, you get 1, 0, 0, minus 3 by 4, 0, 0, 15 by 2, 0, 0, 1, 3 by 4, 0, 0, 25 by 2, 0, 1, 0, 1, 0, 0, 20. You actually do not have to calculate the C_j minus Z_j .The reason being, the moment primal becomes feasible it is optimal in a dual simplex iteration. In a dual simplex iteration the moment primal becomes feasible, it is optimal. The optimality condition will any way be maintained. Just for the sake of completeness, you just put C_j minus Z_j here. So I get 0 here 0 0 0. This is the 1 place, this is 0; I get a plus 1 here, this is a 0 and I get 30.

I have got the optimal solutions here and the optimal solution is X_1 equal to 15 by 2, Y_1 equal to 25 by 2, X_2 equal to 20 and rho_4 equal to 30. I have solved the problem up to this and the 4th constraint. Now what do I observe? I observed that at the optimum the objective function is not 0. If the objective function value is not 0, it implies that I need already a deficit here somewhere. I am not able to get that rho_4 equal to 0; therefore, I stop. I do not proceed to include the other objectives and constraints and optimize further.

The moment I realize that I do not have a 0 here, I stop the algorithm and I only evaluate the effect of the solution on the subsequent objective functions and constraints. So I go back and look at my solution.

(Refer Slide Time: 22:29)



My solution is X_1 equal to 15 by 2, Y_1 equal to 25 by 2 and X_2 equal to 20. I go back and substitute here in the last one. So I have 7 X_1, which is 105 by 2 plus 160 plus 6 Y_1. What do I get? 7 X_1 is 105 by 2 plus 8 X_2 is 160 plus 6 Y_1, which is 75 plus eta_5 minus rho_5 equal to 200. This would mean that, this is 235; 235 plus 52.5, which is 287.5. 287.5 plus eta_5 minus rho_5 is equal to 200, would give me rho_5 equal to 87.5. This would give me rho_5 equal to 87.5.

Remember always eta_5 and rho_5 have to be greater than or equal to 0. So you will get rho_5 equal to 87.5. So straight away I realise that I will have an eta_5 equal to 0 here, but I will be having a rho_5 equal to 87.5. Now the value of the objective function here would be 0, 30, 0. The first objective is completely satisfied. Right here I have a deviation of 30, so I stop here and I just evaluate the subsequent ones. It turns out that I still get eta_5 equal to 0; that is fine. The present solution is able to satisfy the condition or the requirement on the eta_4. So the algorithm stops or terminates. This is how you solve a goal programming problem using linear programming.

What we have seen, I will just quickly summarize and then move to the next topic. Under the section of goal programming, we first understood that optimization

problems may have more than one objective. Sometimes the constraints may be rigid, sometimes the constraints may be flexible, they may not be very rigidly defined. Constraints may have priorities that are given. So rigid constraints will always occupy the first priority and the rest of them will occupy priorities as listed or defined in the problem. Let us do the goal programming formulation, where every constraint is represented as an equation with positive or negative deviations and depending on the nature of the requirement you will get an objective function which will typically look like this. It will be a set of etas and rhos which represent the rigid constraints. The non-rigid goals or objectives will come as indicated in the problem or as required by the person who is going to use it and then you get a formulation like this.

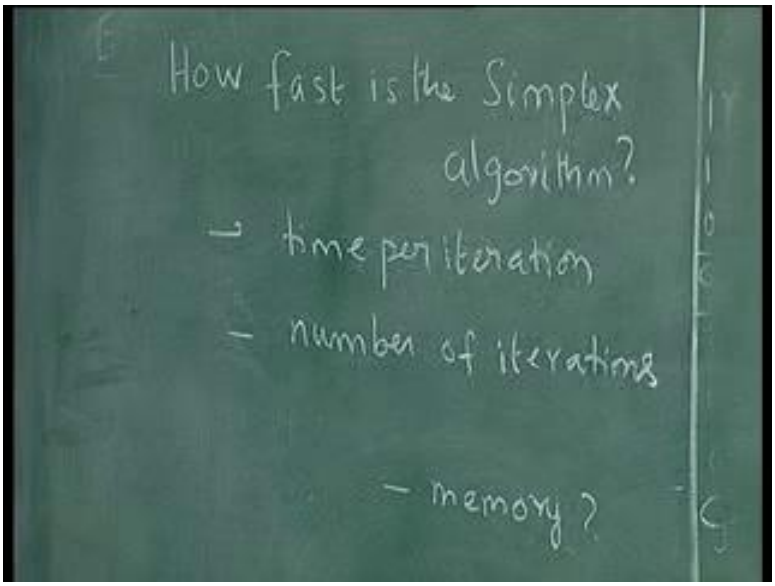
If it has only two variables, you can solve two decision variables. You can solve using the graphical method. If you have more than two decision variables, you can solve using the simplex method in the way we have indicated. Now this is one of the ways of modeling or formulating a problem with multiple objectives. Goal programming, as it is, is only one among the many. There are other ways of solving multiple objective problems, which we are not looking at in this course. Within the goal programming, what we have done is called Lexicographic goal programming. That is, we take the objectives as indicated in a certain order of priority and try to solve. We do not attach weights to the objectives but instead we attach priorities to the objectives and then solve the objectives in the order of priorities. The moment a particular priority is not met, which is indicated by a non-zero value in the objective function, we realise that we do not try and optimize for the rest of the priorities. At that point we stop the algorithm and we go back and evaluate the other objectives for the optimal solution currently available, which is what we did here. Here we realized that this objective in not met in totality. The desirable 0 has now become 30. So we look at this solution and treat this as the best solution and only evaluate the rest of the objectives for this to get 0, 30 and 0 respectively.

There are other ways. For example, one could think of assigning weights to each one and then solving it directly, as a single objective problem. There are many such ways. But what we have covered in two or three lectures is a very small portion of an otherwise large field called multiple objective programming. What we have tried to do is to just give an introduction to it, in terms of a couple of examples and

formulation and the same example is solved by more than one way so that you can get a feel of how multiple objective problems are formulated and solved.

Now with this, we will move to the next part of the topics in linear programming, which is this. We will look at something called how fast is the simplex method?

(Refer Slide Time: 28:14)



We will try to address this question now and perhaps continue the discussion in the next class. The speed or the time taken for a simplex algorithm depends on two things. It depends on time per iteration and it also depends on the number of iterations. There is an additional dimension in terms of goodness of the algorithm, which is also what is the memory required for this simplex algorithm. We will first try to look at this time taken per iteration and number of iterations. Let us also somewhere write down the memory. If we go back and see what all we had covered before the goal programming, we started with the revised simplex algorithm and then we did a column generation. Then we did a bounded upper bounded simplex and then we did a decomposition. Now all these four essentially have tried to address this problem, the first problem in a certain way.

The revised simplex would look at this. In a normal tabular form, we said we are using Gauss Jordan method, to invert a matrix or solve for the basic variables. We looked at the revised simplex algorithm which used the eta factorization method, so that for very large problems the eta factorization method works faster per iteration

than the Gauss Jordan. It is not for very small problems but for large problems. We did not prove that result, but then we kind of said it or mentioned or indicated that the revised simplex with the product form or the eta matrix is faster than the normal simplex per iteration, if for example, the number of constraints exceeds 1000 or of that order. That tried to address this problem of time per iteration. Revise simplex algorithm also address the problem of memory, because we realize that you store less. You store less in the revised simplex algorithm therefore it is better in terms of memory.

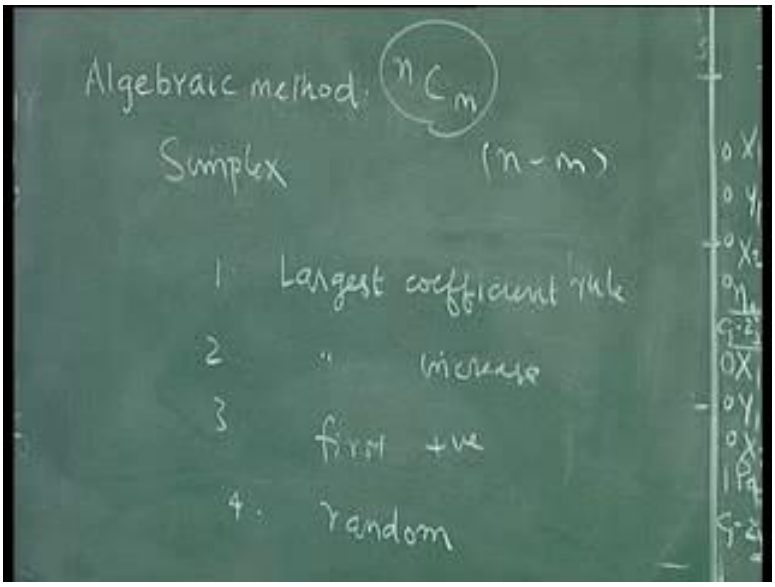
From now on you can assume that, all other forms that we have read, we will now use the revised simplex idea to invert. The upper bounded simplex helped us save a little bit of memory but more importantly time because if there is a bounded variable, the advantage by taking it outside of the iteration would automatically reduce the size of the matrix that we are inverting. Therefore, the time is saved per iteration of the simplex algorithm.

Column generation is primarily a method which concentrates a little more on space rather than time. Column generation would mean you do not save or store the entire coefficient matrix. Instead you store only the B inverse part of it. So instead of storing an m into n, you store an m into m, and even that m into m, if you use revise simplex you store it in a more efficient way. Therefore, for long coefficient matrices n is very large compared to m. The column generation is the way by which you save on memory and may be compromise on time. Because you solve a sub problem every time, it may increase the time a little bit but as a method it helps.

Decomposition is clearly a case where you save on time because you have a bigger problem which can be broken into sub problems and therefore it is easier to invert. Typically the example that we saw, had 6 constraints which had 2 sub problems of 2 constraints each. If you solved it as an original problem, it would be to invert a 6 by 6 matrix at every iteration, but with the sub problems we ended up inverting a 2 by 2, a 2 by 2 and a 3 by 3, which should be faster. All this effort was concentrating on time per iteration.

Then we have to look at the next one. What is the number of iterations a simplex algorithm will take before it terminates? There are a couple of pointers to it. If we go back to the first course, fundamentals of operations research course, we started with what is called the algebraic method.

(Refer Slide Time: 33:10)



Now in the algebraic method, we said that, we always evaluate n C_m or n C_{n-m}, as the case may be, if there are n variables and m constraints, out of which, some of them could be infeasible, etc., and some of them need not give progressively better solutions and so on. Essentially, the point that we tried to address there is, if we want to do an exhaustive enumeration, then we end up doing n C_m number of iterations, some of which may be useful, some of which may not be useful. Now this nC_m is not a polynomial number. nC_m increases exponentially with increase in n and m, and something like m tends to n by 2, it reaches a very large value and so on.

In some sense the algebraic method indicated that, if you look at solving a linear programming problem using an enumeration algorithm, then that enumeration algorithm turns out to be exponential in nature and will have a lot of iterations. Then we moved on to the simplex method and said that, at least some of the things that are undesirable in this exhaustive enumeration algorithm can be avoided. The undesirable ones being the infeasible solution, the undesirable thing being not able to get a better solution in the next iteration; these are the two things that we tried to eliminate. We

tried to eliminate these two things using two important aspects of simplex, which is the choice for leaving variable and entering variable. Choice of leaving variable ensured that the feasibility is maintained. Choice of entering variable ensured that in some sense optimality is maintained or progressively you move towards optimality.

We also saw one more thing that in a simplex algorithm it is always possible for a variable that leaves in an earlier iteration to re-enter. The moment a variable that leaves can re-enter, we do not have something like a polynomial limit on the number of iterations. If for example, we said that a variable once leaves cannot enter, automatically we know that simplex has to terminate in n minus m iterations; that does not happen unfortunately. Now the choice of the entering variable was trying to dictate the rate or speed or the way in which we approach the optimal solution.

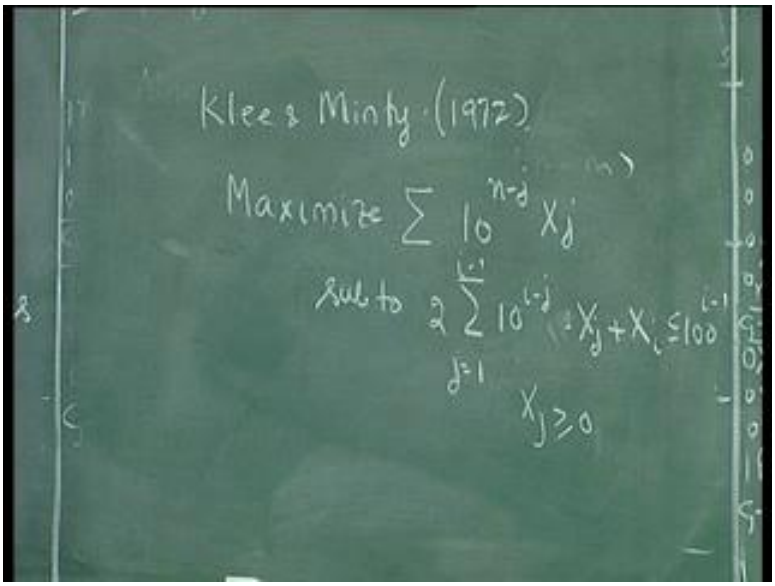
Earlier we had seen four things or four ways of entering a variable into the basis. We will make a quick recap of that. The largest coefficient rule, largest increase rule, first positive rule, random - these are the four rules that we saw. In all the hand calculations, in all the book examples, we will always do this largest coefficient rule, largest C_j minus Z_j rule, the rate of increase being large. The largest increase rule, you are aware of, is like you look at every positive C_j minus Z_j, find out the corresponding theta; product of C_j minus Z_j and theta is the extent of increase from this iteration. So enter that C_j minus Z_j which has the largest value of the product.

This is a typically greedy way of looking at it. It is something like the method of steepest ascent. If we are starting at 0 and you want to reach the optimum for a maximization problem, at every point then I try to jump the farthest, hoping that I will reach my destination at the earliest. Particularly when solving large problems, these two involve a certain amount of computation. Because to find the largest, you need to sort them; to find this out you need the product and store it and then try to sort it and get it. These two basically do not do that. All you need is to look at the C_j minus Z_j and the first one that is positive will enter. Now this is commonly used in implementation where we just try to look at the first positive, do not spend that much time and quickly enter a variable. Last but not the least is random and random is you just pick a random C_j minus Z_j, if it is positive it enters.

Among these four, this is considered to be the best and it is used very widely and extensively. The reason being, based on a lot of experimentation, different researchers carried out experimentation and tried to find out for the same set of problems or for problems that have been defined in a particular order. For example, in randomly generated problems, following certain distributions, with all the problems being generated out of a certain order, they tried it on say 10,000, hundred thousand problems and then tried all the four rules. They tested it on problems of different sizes, different values of number of constraints, number of variables, something like starting from a 5 by 5 to 100 by 100 or even more, for different problem sizes, combinations of problem sizes; all the coefficients, that is the right hand side, the objective function and the constraint coefficients randomly generated out of certain distributions, the range of values being defined and consistent. Which means, effectively they tested the same problem using all these four and came to the conclusion that, the largest coefficient rule, actually fairs better in terms of average number of iterations per problem. One can argue by saying that if they had chosen a different distribution or if they had done something else, may be some other thing would have dominated. But it is an accepted fact that largest coefficient rule, which is the C_j minus Z_j rule, is expected to give a minimum number of iterations in general, based on experimentation.

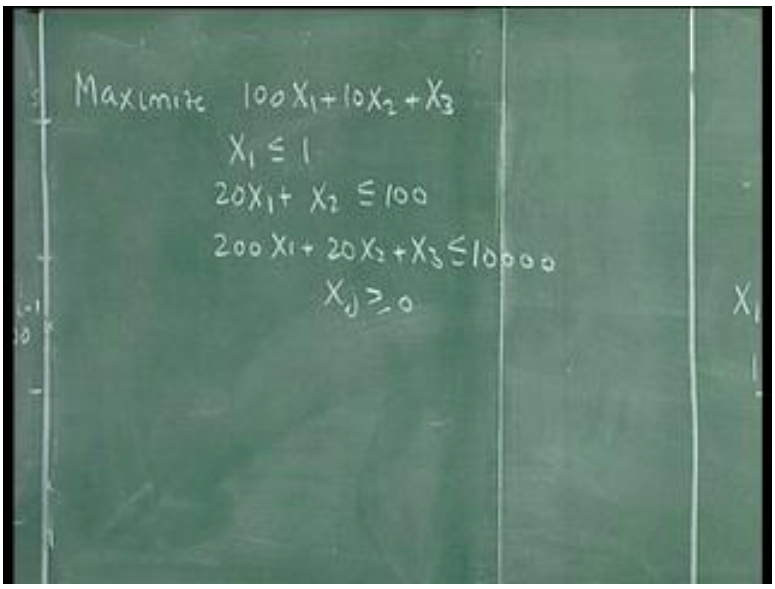
For a particular problem, it may turn out that this is inferior and this is superior. But in general this is considered to be a very good rule. Now this was followed extensively as a rule for entering variable till something called the Klee and Minty problems were proposed.

(Refer Slide Time: 40:00)



Two people called Klee and Minty came up with a set of problems which looked like this. Klee and Minty in 1972, said maximize….(Refer slide time:40:30). This is the generic class of the Klee and Minty problem and for example for n equal to 3, the Klee and Minty problem will look like this.

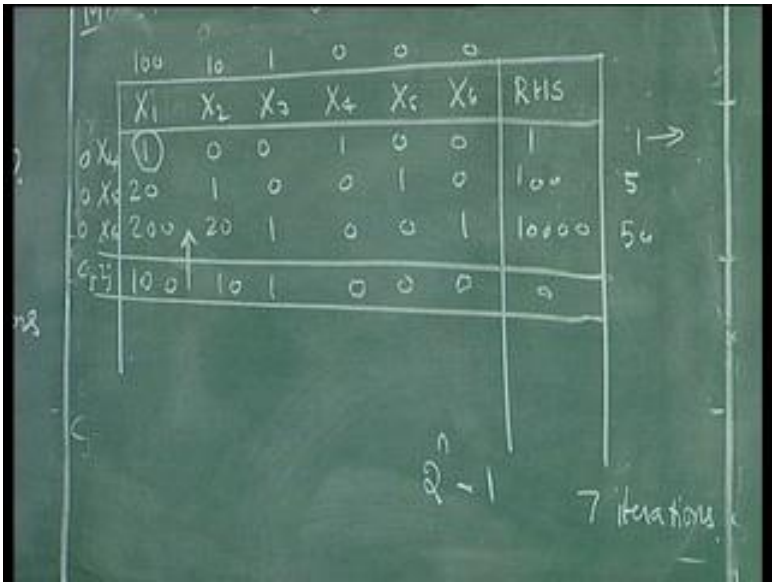
(Refer Slide Time: 41:18)



Maximize 100 X_1 plus 10 X_2 plus X_3; X_1 less than or equal to 1. 20 X_1 plus X_2 less than or equal to 100, 200 X_1 plus 20 X_2 plus X_3 less than or equal to 10,000; X_j greater than or equal to 0. This is the Klee and Minty problem where n equal to 3. The

Klee and Minty problems were defined from n equal to 3 onwards. So n equal to 4, problem will look like 1000 X_1 plus 100 X_2 plus 10 X_3 plus X_4, subject to something X_1 less than equal to 1. This, this, this and 2000 X_1 plus 200 X_2 plus 20 X_3 plus X_4 less than or equal to something; 10,000 into 100, which would be what 10 lakhs or whatever. This is the Klee and Minty problem. Let us do one iteration with the Klee and Minty problem and then see what happens when we try to solve this. We will just do one iteration to understand this.

(Refer Slide Time: 42:40)



We start with X_1, X_2, X_3; we will assume X_4, X_5, X_6 as the slack variables and then, you have a right hand side. You will have 100 X_1 plus 10 X_2 plus X_3 0 0 0. You start with X_4, X_5, X_6; 1 0 0 1 0 0 1; 20 1 0 0 1 0 100; 200, 20, 1 0 0 1 with 10000; With C_j minus Z_j 0 0 0 100, 10, 1 0 0 0 and 0. This is how your simplex will begin. Now the largest C_j minus Z_j is here, so you will enter this and if you go back and compute the thetas, you get a 1 here, you get a 5 here, you get a 50 here. The first iteration will happen with this as leaving variable, this as the pivot and so on till you will reach the optimum. The optimum for this problem is X_4 equal to 1, X_5 equal to 100, X_3 equal to 10,000 with Z equal to 10,000.

Number one, what is wrong? It is like any linear programming problem. I can just solve it till I get 10,000. Only problem is, if you start with X_4, X_5, X_6 and proceed the way we are used to, by using the largest coefficient rule always enter the most

positive C_j minus Z_j; this problem will take 7 iterations till it reaches the optimum. In general, any Klee and Minty problem will take 2 to the power n minus 1 iteration till it reaches the optimum. That is where the problem is. Now 2 to the power n is not a polynomial number. So you will end up getting a very, very large number and exponential number of iterations. There are many things about it.

Number one, you can always go back and look at this problem and say why should I start with X_4, X_5, X_6? Might as well start it with X_4, X_5 and X_3; X_3 with a 0 0 1 qualifies. The moment I start with X_4, X_5 and X_3, I have the optimum right in the first iteration because, the optimum is 10,000. So I would start with X_3 equal to 10,000; Z equal to 10,000.

Second, I do not look at 2 to the power n minus 1, I could have just done it in one iteration. If I use the largest increase rule instead of the largest coefficient rule, look at the largest increase rule. The largest increase rule would give me 100 and 1 which is 100. If I enter this 10, I would get 100 and some other number. So minimum theta would be 10 into 100, which is 1000. If I enter here, I would get 1 into 10,000, which is 10,000. X_3 will straight away enter if I use the largest increase rule instead of the largest coefficient rule. It will go; that is also possible. In one iteration, I will get the answer. If I scale the variables in such a way that I define a Y_1, Y_2, Y_3, such that, Y_3 is $0.0001X_3$, Y_2 equal to $0.01X_2$, Y1 equal to X_1 and solve it. Using the largest coefficient rule in one iteration, I will get the answer because, somewhere here …. 10,000; I will have to scale it in such a way that 10,000 appears here.

There are multiple ways by which you can show that I can get the answer in one iteration but that is not what theory is all about. Theory would close its eyes to all its arguments and tell you that look if I start the simplex the way I know, which is X_4, X_5 and X_6, starting with the slack variable, I get 2 to the power n minus 1 iterations. So Klee and Minty problems, in some sense, they did not challenge anything but they brought out the fact that simplex can be worst case exponential algorithm, which means simplex is not a very good algorithm; it is a worst case exponential algorithm. But any amount of experimentation always proved that simplex is a fantastic algorithm when it comes to average case. When it comes to trying out different problems and trying to see the number of iterations on an average, simplex was doing exceedingly well. The average number of iterations were always close to 3 m by 2,

where m is the number of constraints in the problem, rarely exceeding up to 3 m it never went beyond 3 m. So simplex was a very good algorithm when it came to average case and a very poor algorithm when it came to worst case. What people did to overcome this, we will see that in the next lecture.