Advanced Operations Research

Prof. G. Srinivasan

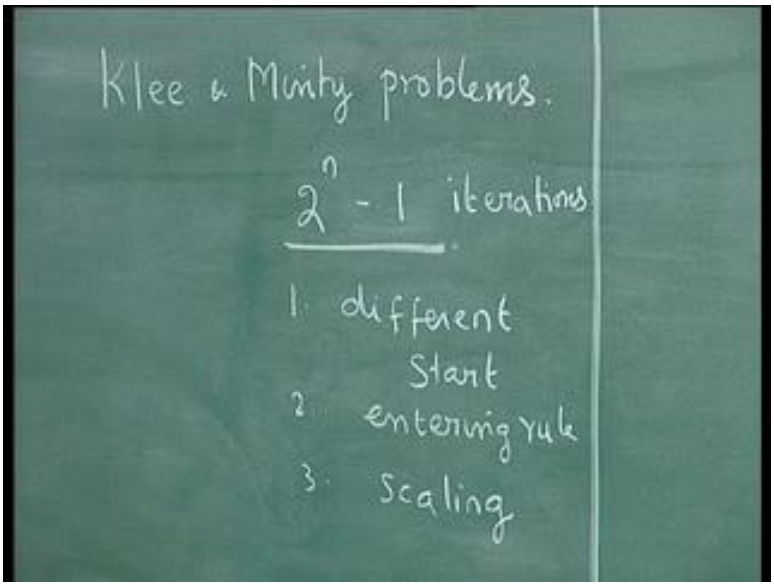
Department of Management Studies

Indian Institute of Technology, Madras

Lecture - 11
Complexity of Simplex Algorithm (Continued)

Integer Programming

(Refer Slide Time: 00:17)

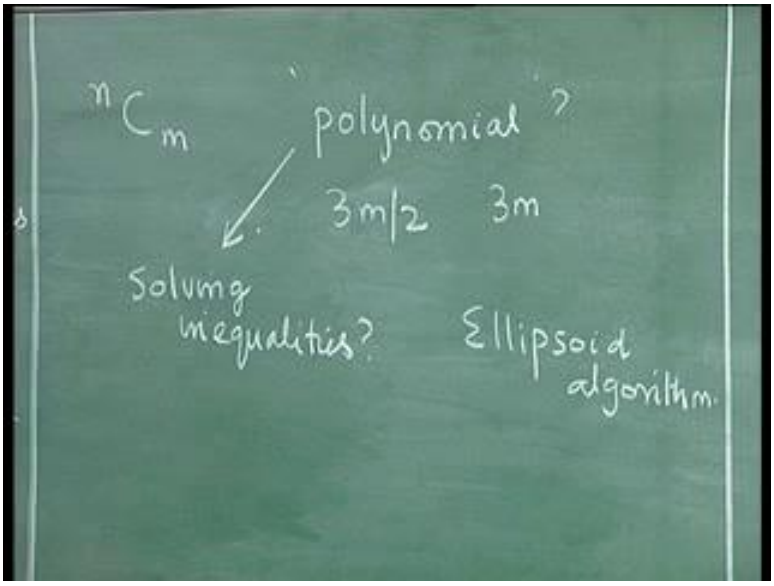


In the last class we looked at the Klee and Minty problems, and said that the Klee and Minty problems would take 2 to the power of n minus 1 iterations if there are n decision variables in the problem. We also saw that there could be indirect ways of overcoming this difficulty particularly for the Klee and Minty problems. Those ways of overcoming the difficulty could be, to have a different starting solution or it could be a different entering variable rule, or scaling the variables could also get us out of the Klee and Minty difficulty.

We should keep in mind that the idea of the Klee and Minty problem was not to solve the Klee and Minty problem, but to expose the fact that simplex algorithm particularly when you use the maximum or the largest coefficient rule: the C_j minus Z_j rule, can turn out to be exponential for certain types of problems.

The algorithm is called a polynomial algorithm, if the time taken to solve is a polynomial function of the problem parameters. It becomes non-polynomial; it becomes exponential when time taken can be shown to be an exponential function. At this point, we are not going to go deeper into complexity, but based on the Klee and Minty example; understand that simplex can take an exponential number of iterations, particularly when you are looking at Klee and Minty. Very few examples of Klee and Minty have been published. There could be many more which have not been published yet. But this was enough to show that simplex is a worst case exponential algorithm and it is not the worst case polynomial algorithm.

(Refer Slide Time: 02:32)



The fact that it could be exponential was already evident by the algebraic method which will take superscript nC_m iterations or it evaluates all the superscript nC_m possible solutions. One could go back and perhaps argue that there are no Klee and Minty examples for the largest product rule or the largest increase rule. But it would not be too difficult to create an equivalent Klee and Minty example, because they have been created for the largest coefficient. The main understanding is the fact that, simplex is an exponential algorithm, when it comes to looking at it, in terms of complexity.

The next question that people started asking was can we have polynomially bounded algorithms to solve linear programming problems? In this course, we are not going to go very deep into the

understanding of the polynomial algorithm. We will just give an outline or an overview of at least one of them; also try to explain some of the issues related to this.

When this question was raised after the Klee and Minty examples were published, then people also started arguing that, simplex however was an excellent algorithm when it came to average case performance. It is only for the very small amount of Klee and Minty and related problems it behaves in an exponential manner. Whereas when you test it on normal problems, problems that are taken from real life, problems for which a linearly programming solution is applicable, meaningful, practical, can be implemented and so on.

In all those cases, simplex was only taking about 3 m by 2 iterations on an average and very rarely going up to 3 m. So simplex, in average case was a very good algorithm. However, average case performance of algorithms is meant for people who practice and in worst case, meant for people who developed. So theory still felt that, simplex is an exponential algorithm.

Now the question was, are there polynomials algorithms and how do we develop that. When this thing was going on, there was also another issue that was being discussed which is called solving inequalities. Remember that, even though all the linear programming constraints are inequality, there could be equations, but predominantly all of them are inequalities. We never tried solving them directly. We converted the inequalities to equations and then we solve them. That is because we know or we have learnt how to solve equations, whereas we have not learnt how to solve the inequalities.

Solving inequalities simply means, given a set of inequalities can we find a solution?

For example, if we give you say four or five inequalities of two variables. For example: X_1 plus X_2 less than and equal to something; $3X_1$ plus $2X_2$ greater than or equal to something. So, you define a certain number of variables and certain number of inequalities. Now solving this inequalities simply means, finding an X_1, X_2, which satisfies all the inequalities. Or if there are n variables, finding even one X_1, X2, X_3, X_n which satisfies all the constraints is called solving inequalities.

When you represent the solving inequalities problem and solving equations problem, one may be tempted to think that solving inequalities is actually easier than solving equations. If you take a 2

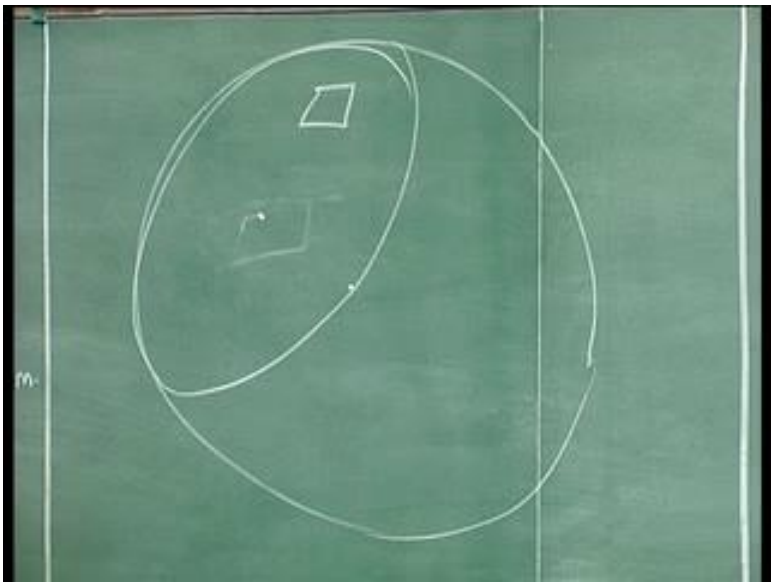
by 2 set and plot the inequalities on the graph, but if you end up getting a feasible region you can actually close your eyes and put your finger at any point in the region that will satisfy all the constraints, that is the solution to inequalities whereas you cannot do that to equations.

Most of the time there is only one point which satisfies all the equations. You could have other cases like, no solution, infinite solution, and so on. In practice, actually speaking, solving inequalities is not easy; people have not developed an algorithm that could solve inequalities, a polynomial algorithm that could solve inequalities, till 1979 or so.

When this issue is being discussed, the ellipsoid algorithm was developed in 1979 to solve inequalities. It was a polynomially bonded algorithm to solve the inequalities. We will not go into too much of detail on the ellipsoid algorithm, but we will give you some basics of the ellipsoid algorithm. If you make an assumption that there exists a feasible region in certain number of dimensions. If you initially make an assumption, that the given set of inequality has a solution, which means there exists a feasible region for this. It is only a matter of finding a point which is inside that feasible region.

The basic idea of the ellipsoid algorithm is to try and imagine, a very large the biggest possible ellipsoid of the same number of dimensions, as the number of variables that you have. For example, if your inequalities have 10 variables, irrespective of the number of constraints, you should imagine a very huge ellipsoid which is of 10 dimensions. Then create such an ellipsoid and locate its centre. When that center satisfies all these constraints that you have, then you have solved it. If you have not, then the underlying assumption is, if the existing feasible region is the centre of the ellipsoid does not satisfy the existing feasible region, and if such a region exists, it is always possible to put that region in a smaller ellipsoid, which is proportionately reduced in terms of volume. There are lots of governing equations and mappings and all that which we will not get into.

(Refer Slide Time: 08:57)



The underlying assumption is, to just to put it on the black board. For example: if this is your region and if this is your huge ellipsoid, it need not be a circle. If it is a huge ellipsoid and this is the centre and it does not satisfy the feasible region, then you can always make an assumption that, we can put something called a half ellipsoid, which will be like this, whose center in this case satisfies. If the feasible region is here, then the half ellipsoid center also will not satisfy.

How do you get this half ellipsoid from the bigger ellipsoid is a matter of a lot rigour; we mean, so much of mapping, so much of equations, you cannot arbitrarily choose half ellipsoid. It has to be proportionately reduced. All those, there is a lot of theory, which we are not going to look at today. We are only going to see the principle in which the algorithm was developed. You try to put it in a half ellipsoid and then try to take the center of the half ellipsoid. If the center of the half ellipsoid again it satisfies this, then you have solved it. If it does not put another half ellipsoid from the given half ellipsoid and keep doing it, till it satisfies. This is the basic idea.

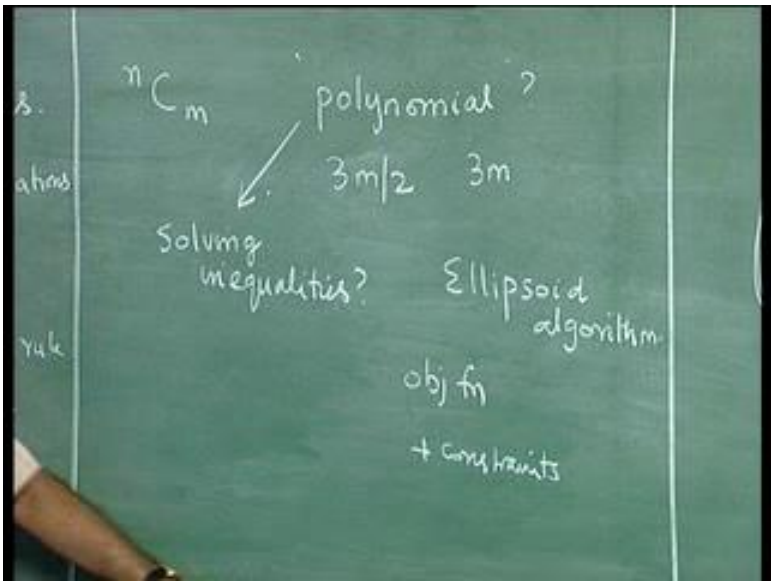
There are a whole lot of issues associated with it. The most important thing being, if there is a solution, then you have to prove that you will reach that point in polynomial number of steps and that is the most important thing. All those things were done and we are not going to see how they were done, but all those things were done, so that, this algorithm was developed, to solve inequalities.

This algorithm is called the Kachiyan's algorithm. It came in 1979 and Kachiyan proved that, there is a polynomially bounded algorithm to solve inequalities. This was a major development, particularly when people were thinking about whether we could develop a polynomial algorithm for simplex. The moment this was available, of course, there were other issues. For example, this is under the assumption that the given set of inequalities has a feasible region. There are other issues like, what if it does not have, what if it is not solvable. All those are, in some sense, a little more involved than even trying to understand the algorithm which will solve the case where a solution exists.

We will right now leave that; we will only take the learning from this part. The principle is you start with a largest possible ellipsoid, get it centered. Center does not satisfy you, assume that there are going to be two half ellipsoids and it will be in one of them. Then try to get that. It is not that you made 2 into 4, 4 into 8 and so on it, becomes exponential, but then you choose the one and gradually reduce the volumes of ellipsoids, identify the centres, and then keep proceeding till you are able to get it.

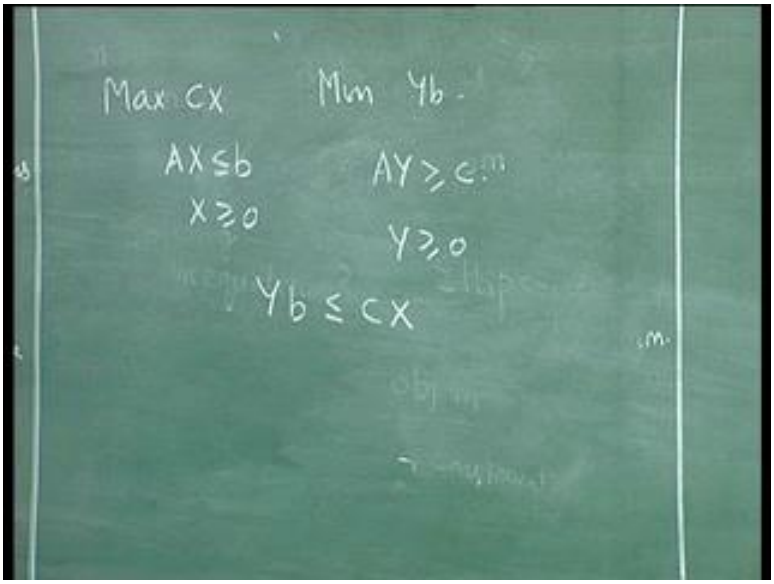
When this algorithm was developed, people said, we can use this algorithm to solve linear programming problems. How we do that?

(Refer Slide Time: 12:03)



The linear programming problem is actually an objective function plus constraints. If we hide this, then this becomes solving inequalities, but linear programming is something more. Here, this algorithm takes care of finding just one point among the many that may be available which satisfies this. But on the other hand, what you want is among the infinite or many available, you want the best. So you cannot directly apply this algorithm to this. What people said is, let us do a small trick and get it. The trick goes like this.

(Refer Slide Time: 12:50)



If we have a linear programming problem, which is maximise CX subject to AX less than or equal to b, X greater than or equal to 0. Now its dual will be, to minimise Yb, subject to say AY greater than or equal to C, Y greater than or equal to 0. You need not worry so much about the matrix notation, consistency of the matrices and so on. We will just take it crudely, that this is how the dual will look like.

What will happen is, your duality theorems have told you, that if there is an optimum, at the optimum, the CX star should be equal to Y star b, because the values of the objective function will be equal for both the primal and the dual at the optimum. Your weak duality theorem also tells you, that for every feasible solution to the primal and to the dual; this Yb is greater than CX.

All you need to do is, you need to put Yb either equal to CX, or more importantly less than or equal to CX. So the optimum to both, if you solve this set of inequalities, you have actually got

the optimum to both the primal and the dual. To repeat again, from the weak duality theorem, Yb is greater than or equal to CX, but at the optimum Yb is equal to CX. You change the constraint to, Yb less than or equal to CX. So the only point that will satisfy, the only XY that will satisfy this, is the optimum. Because you are dealing with inequalities you do not want to put an equation because you may have to again convert the equation into two inequalities.

You just take it as the opposite. This is the opposite of the weak duality theorem result, but this is consistent with what we are trying to do. So add this one, so when you add this one, you get a set of inequality. The objective function does not come any more, so you get a set of inequalities. So you are trying to solve a set of inequalities, apply this algorithm and try to solve it.

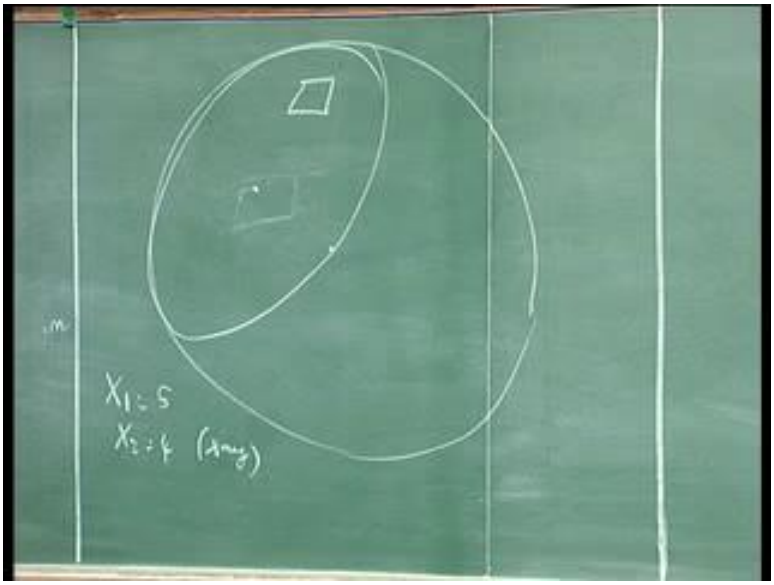
People said if this can solve inequalities in polynomial time, now this can solve linear programming problems in polynomial time. But then there were another issue. The issue is very simple. If this system has a solution, then there is no feasible region for this. There is only one point, there is no feasible region. This algorithm had to be modified to look at a situation, when the set of inequalities has only one solution, which is unique, if it exists. Whereas, in the general case of trying to check whether the centre satisfies the region, now it is more like defining the ellipsoids in such a way that ultimately the centre will coincide with that unique point; so that had to be looked at.

Obviously, this algorithm was modified and then there were perturbation schemes introduced. You try to put a small area around this point and perturb, and then check that you define a small feasible region around this optimum point. Then if it comes into that, you say it is, you have to define the epsilons, and then you have to define the number of iterations in which it will converge into that. All those things happened and then people were kind of happy and convinced that the ellipsoid algorithm is capable of solving the linear programming problems in polynomial time.

This was ascertained and the ellipsoid algorithm was the first algorithm in principle, to be developed, not necessarily or not with the primary objective of solving LP, but then it could be adapted to solving LPs, in providing an algorithm that solves it in polynomial time. Let us try to solve some more problems in this. Now there is a theoretical proof that ellipsoid algorithm can solve it in polynomial time. Let us try out some problems.

If you go back and try out the ellipsoid algorithm, for a simple problem like: maximise $5X_1$ plus $6X_2$, subjected to 2 sets of constraint and all that. Ellipsoid algorithm will take at least about 1000 iterations, for it to converge whereas simplex will do it in 2 iterations.

(Refer Slide Time: 17:13)



For example, if your optimum is X_1 equal to 5, X_2 equal to 4, say what will happen is ellipsoid will kind of alternate. It will start alternating between say, 5.5 and 3.5 on one hand then 4.5 and 4.5 on the other. It will look at 5.5, 3.5 here; it will look at 4.5, 4.5., it will move a little bit closer and then it will slowly try to converge towards this 5 and 4.

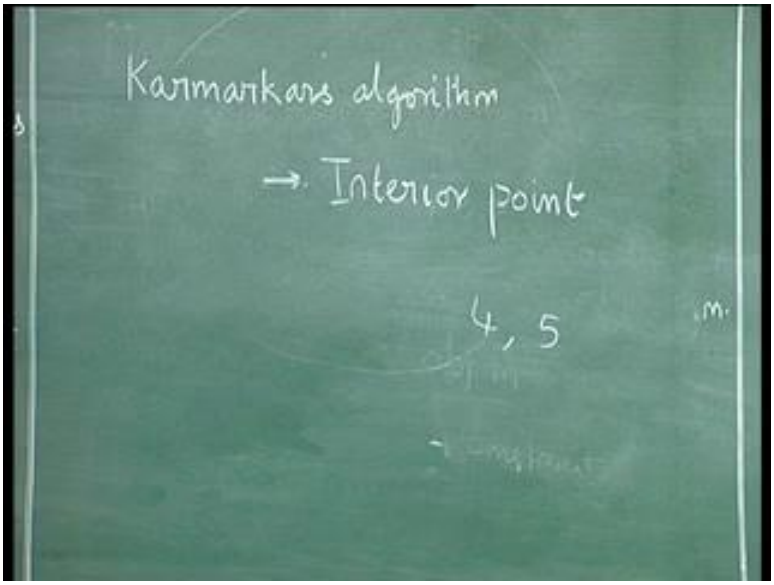
What you have to do is you need to put an epsilon and say that, the moment it comes within a certain range, you terminate. But you do not know what that epsilon is. You run into those kinds of difficulties. What was evident from such a study, not by a single person but many people was that this has a very poor average performance. In terms of average performance, simplex will just beat the ellipsoid algorithm hands down for the kind of problems that people were solving.

People are not the researchers, people were everybody, we mean in linear programming terminology, problems even with 500 constraints or 1000 constraints are considered to be small. It becomes large only when you are looking at a million constraints, a million variables and things like that. If you are looking at 50,000, 100,000, you are somewhere in the medium, today.

That is a kind of problem size that is associated with the definition of it being small, large, medium, and so on.

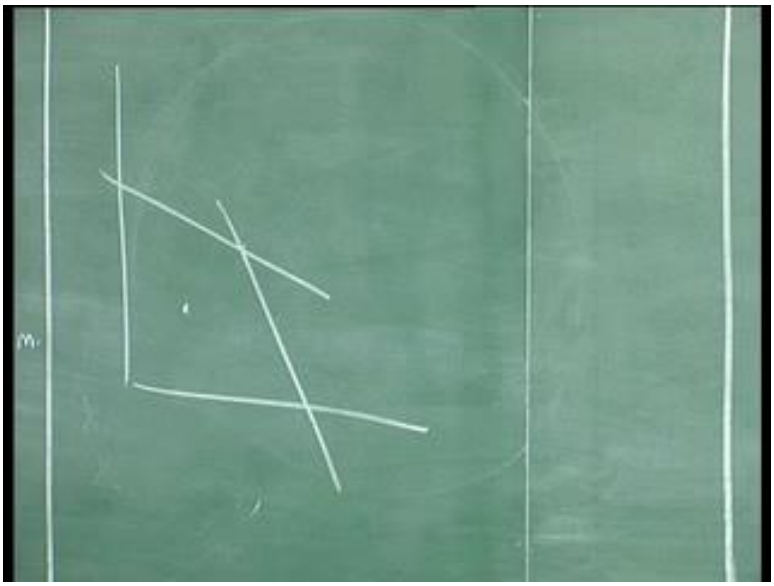
We are not looking at the 2 by 2's and 3 by 3's that we are familiar with. If you want look at a real life application that comes, say communication, or comes from scheduling in airline, or comes from forecasting, demand in an airline industry, or in an FMCG firm or whatever be the example, which can be modeled as linear programming related problem, then you are running into tens and thousands or may be hundreds and thousands of variables. Even when it comes to solving medium sized problems, the simplex was beating the ellipsoid algorithm, and ellipsoid algorithm had a kind of a poor average scale performance. But it did have a very good worst case performance which is polynomial.

(Refer Slide Time: 19:40)



When this was going on then came a most important break-through which is called the Karmarkars algorithm, which was also a kind of an interior point algorithm. If you go back to the simplex, simplex is not an interior point algorithm. Simplex is basically a corner point algorithm. It start with one corner point and keeps moving along the corner points in a certain way so that you reach the optimum because you have a result that, the optimum solution is always a corner point solution.

(Refer Slide Time: 20:30)



For example, if this were a feasible region. In a 2 by 2 case what simplex would do is, it would start here, it either moves here or here, here and then check whichever is optimum (20:38). It keeps moving along the corner points. For example, nothing prevents you from starting with the point here, knowing fully well that there it is not a corner point, provided you can move in the right direction from this. Not necessarily in one jump, go to this corner point or this corner point and so on, but keep jumping in a certain way, till you finally reach the corner point. This was the basic idea. Now such algorithms are called interior point algorithms.

In a 2 by 2, you cannot actually appreciate the whole thing. If you can think of a three-dimensional representation then what you can do is you can be in an intermediate point then you try to move either in x or in y or in z up to a certain limit and then keep moving that way. It is not in one jump you can get here, you still have to move progressively, you have to make multiple jumps and then slowly move towards the corner point that is optimum. That was the basic idea of the Karmarkar's algorithm, wherein there are mapped problems. You map all the x's into the y's and then you try put something into a huge sphere of higher dimensions, instead of ellipse and ellipsoids, use circles and spheres higher dimensions and then you solve the y problem and then get the optimum solution to the y in a certain way, and then map it back to x.

Your x moves from certain place from inside the feasible region to some other point inside the feasible region. Keep doing this back and forth, till you reach the corner point. Reaching the corner point is not exactly sitting on the corner point, it will be like you are within some 2 to the power minus n of the corner point and then you will terminate it. It is not that, if the answer is (4, 5), it will beautifully sit at (4, 5) and tell you that this is the optimum, with simplex would do. It would always converge a 3.998 and 5.001, if you are giving 3 digit accuracy or whatever. If you want 25 digit accuracy, then it will be some 2 to the power something error and so on. That was the basic idea of the Karmarkars algorithm.

There were lots of similarities and differences between the ellipsoid algorithm and the Karmarkars algorithm. There are lots of research papers which talk about, may be the basic idea is from the ellipsoid, some people would say, it is not. There are other related issues about the origin. Both these algorithms came from scientists who worked in different countries which compete all the time. So there were issues of origin and so on. But that is beyond our discussion. Our discussion is only to look at algorithms that help solving linear programming problems.

Today Karmarkars algorithm is accepted which is exclusively developed to solve linear programming problems unlike the ellipsoid algorithm; perhaps a little faster. Both of them have a worst case polynomial nature of the algorithm. The important thing is that, the Karmarkars algorithm should start with an interior point, which means the interior point implies that the point is inside the feasible region. What Karmarkars algorithm would do is, it will solve particular linear programming problem for which the value of the objective function is 0 at the optimum. The point 1 by n, 1 by n, 1 by n, 1 by n is feasible and so on.

There are certain conditions associated with the Karmarkar problem. The problem that he used as a base problem to develop the polynomial algorithm, would satisfy certain things, at least some of which are the value of the objective function is 0 at the optimum, and the point 1 by n, 1 by n, 1 by n is feasible. He also devised a mechanism by which, any problem that has an optimum can be converted into the Karmarkar form, by suitably writing the dual, by suitably scaling some variables. It is a slightly complicated way to convert a given problem into a Karmarkar problem.

For example, if you take a simple 2 by 2 that you are very familiar with. If you convert it into the Karmarkars form, you will end up getting some 10 constraints and some 10 variables. That does

not matter; it is still theoretically a polynomially bounded algorithm. Once again, you have to go back and look at in terms of average case performance and complexity; simplex was still better than Karmarkar's algorithm for small and medium sized linear programming problems. Only for very large problems, which run into millions of constraints and tens of millions of variables and so on the Karmarkar's algorithm could be faster than the simplex. This is where the theory is, with respect developing efficient algorithms for linear programming.

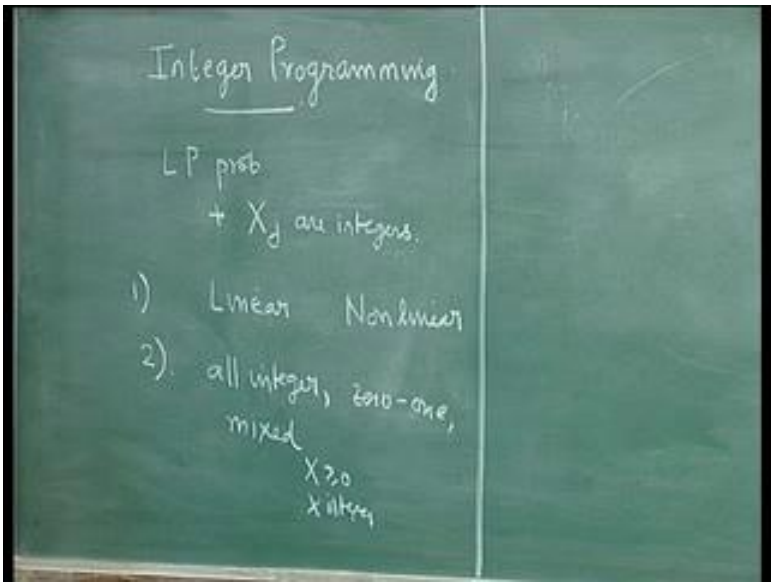
As far as this course is concerned, simplex will be the algorithm that you will use to solve linear programming, as well as integer and the other kind of programming problems. As people who learned operations research, one should be aware of the limitations of the simplex algorithm and the availability of algorithms that are theoretically very superior. Depending on the size of the problem that is being solved, may be faster than the simplex, even for average case, if the problem size is exceptionally large. For the kind of small and medium problems, that people encounter in practice, simplex is still the best, when it comes to average case performance and general average running performance. With this we will wind up the discussion on how fast or how efficient the simplex is.

We have not looked at, for example, there are obviously results available, if the given problem with hundred thousand constraints is infeasible or if the given problem is hundred thousand constraints may be unbounded, one does not know. But then there are results available. Both for the ellipsoid as well as the Karmarkar algorithms there are efficient versions, the moment the Karmarkar algorithms came in 1985, nearly for ten years there were papers with extended Karmarkar algorithm to various situations discussing implementation of the Karmarkar algorithm, discussed their own experience in solving the large real life problems and so on. There are plenty of researches available and people should still be working in several places to develop polynomially bounded algorithm better than Karmarkar's in terms of performance.

As far as we are concerned and this course is concerned, we would look at simplex and keep simplex as the algorithm to solve linear programming problems. We will wind up the discussion on Klee and Minty problems, as well as the goodness of the simplex algorithm.

We will move on to the next topic today which is called the integer programming.

(Refer Slide Time: 28:04)



We will start integer programming; we will also look at a few things here. I will give you an introduction to integer programming and explain why it is different from linear programming. We will also look at some formulations. We will also try to see that certain nice things can be modeled using IP. Then we will go to solving integer programming problems. We will first look at this 0 1 case and then we look at the Gomory cutting plane algorithm and then some all integer algorithms. That will be the topic of integer programming.

What is an integer programming called?

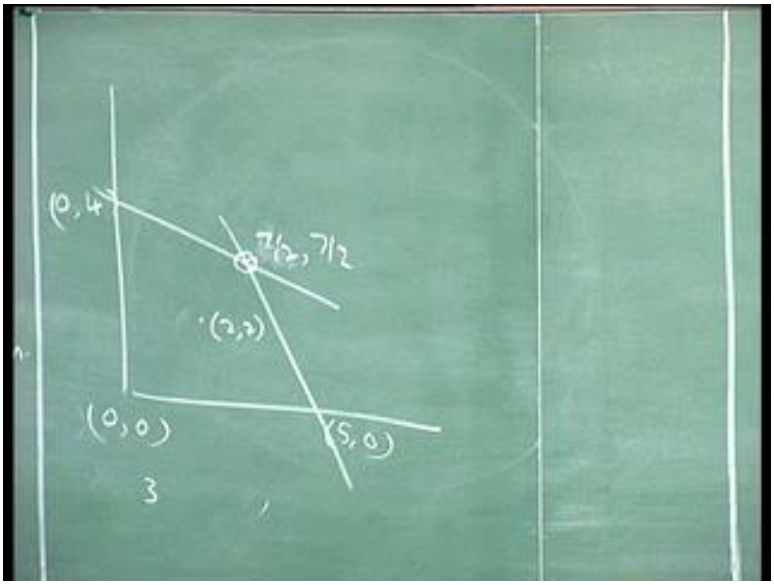
It is a linear programming problem plus the condition, that X_j are integers. Moment you add an integer restriction it becomes an integer programming problem.

The question is can we define integer programming problem as one which has a linear objective function, a set of linear constraints, non-negative restriction plus an additional condition that variables are integers. The answer is no. It is actually beyond that.

An integer programming problem can be a linear integer programming problem, it can be a nonlinear integer programming problem; it can be both. In terms of objective function, it can be linear integer; it can be non linear integer. But common terminology would mean that, if you say it is an integer programming problem, it means, it is linear integer; unless otherwise stated,

integer programming means linear integer, but it could be nonlinear integer. You could have nonlinear objective functions.

(Refer Slide Time: 30:11)



What do we mean by an integer variable?

Integer variable, for example, if we have a feasible region to an LP like this, now it simply means, like for example, if this is 0 0 and say this is 5 0, this is 7 by 2, 7 by 2 and say this is 0 4. Now what does integer mean? Integer would mean that, for example, a point here, say (2, 2), which is an integer point, is feasible, whereas, this point itself is not feasible to the integer programming problem, because it is not integer. In terms of the integer restriction of the variables there are two categories.

One category is called an all integer problem; where all variables are forced to take integer values. You have zero-one problems, where the variable can take either a 0 or a 1 or binary. It cannot take value 2 or 3 or whatever. In the assignment problem that we have seen in the earlier course, this is zero-one problem; X_{ij} is 0 or 1. But then we solved it as a linear programming due to other reasons but the problem as such is a zero-one problem where variables take either 0 or 1.

Then you could have mixed integer programming problems. Normally when you say mixed integer it means you have some variables that are greater than or equal to 0 and some variables

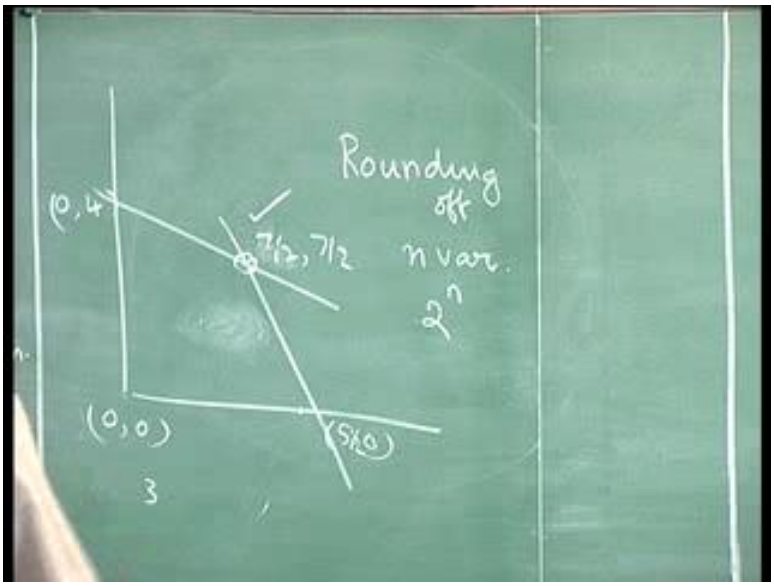
that are integer. When you say a variable is integer, it means you are looking at this category. You have to explicitly say that it is a binary variable, if it is a binary variable. When you say the variable is integer, it does not include binary. So variables can be of three types: continuous, integer and binary.

Continuous is greater than or equal to 0, which is in linear programming. Integer means, it takes integer values 0, 1, 2, 3, 4, any integer. Zero-one means it takes either a 0 or 1 or binary. When you say the problem is mixed, it could be any two out of the three or all the three. Once again, normally when you say mixed, it means, it is continuous and integer. The zero-one, unless otherwise stated does not figure in a mixed problem. But in reality, a problem may have all three types of variables; continuous, integer, and binary, it could be anything.

Typically you can have something like six or seven different types of IPs. You can have a linear integer program, all integer linear, linear objective function and all integers; you can have a linear and a mixed integer; you could have a linear and a zero-one; you could have a nonlinear integer, nonlinear zero-one, nonlinear mixed and so on. The moment you get into the nonlinear segment, it becomes tougher and within that if you get into mixed, it is much tougher. The only solvable thing in the nonlinear is a nonlinear binary. Nonlinear binary is reasonably easy because if the variable X is binary any power of X is also binary. It is only 0 or 1. Therefore it becomes simple. Only difficulty is, when you have a product, you get into trouble there. As long as you have an X that is binary, any X to the power n is also 0 or 1. So we will look at that as we proceed.

Typically a linear programming problem has all these things. Let us look at the first one, the simplest one, with a linear and an all integer. This would simply mean a linear programming problem with the additional restriction that variables are integers. In a 2 by 2 case, you would go back and say, this is my LP solution. Now we know that this point is not feasible to the integer problem. All we want to do is to find out that integer point, which has the best value of the objective function. Assume maximization, say we want to find that integer point, which maximises my objective.

(Refer Slide Time: 34:44)



What are the issues?

The first and most important issue is, if this happens to be a corner point, if we leave the integer restriction, relax it into an LP and solve it, and if we say that this is the corner point solution. The normal feeling that one would have is, that the integer point that is closest to this corner point is likely to be optimum. A feasible integer point which is nearest to the corner point is likely to be the optimum, is a very common assumption that we can make. That assumption need not be correct all the time. You can have a problem where this could be optimum. You can have an objective function where, a point, a different corner point; a different integer corner point could be optimum. It is all a matter of how the constrains are defined and how the objective function is defined.

It is not absolutely necessary that the feasible integer point closest to the corner point is optimum. That hypothesis need not be correct all the time. You could get a different integer corner point or a different integer point which is away from the existing LP. For example, this could be optimum, if this is 5 ½ 0, then 5 0 could be optimum. So that is number one.

Now, if we compromise and say that, we are finding it difficult, we know that the nearest integer point to the infeasible corner point need not be optimum to the IP. However, let me try and find an approximation, after all it cannot be a bad solution because the LP optimum is here, we just

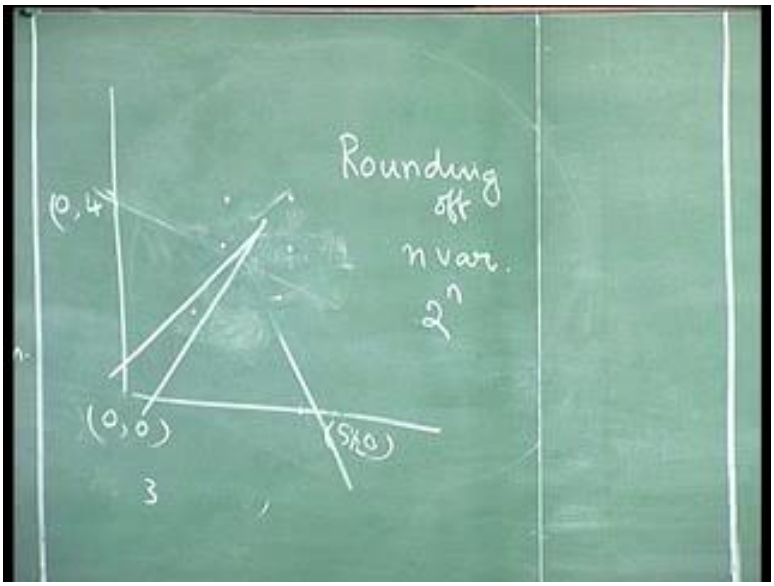
put an integer restriction, it cannot be a very bad solution. It can be a descent solution, so we want to compromise and we find out that point.

Now we are compromising, we are not looking at the exact optimum, but we are compromising somewhere. We start doing that, then what we have to do, is to round off this. So, rounding off, is most of the time seen as a very efficient strategy to solve IP. When you round it off, what will happens is, if you have n variables, then each variable is rounded off to its upper limit and the lower limit. You can have 2 to the power n rounded solution that are possible.

2 to the power n is not a polynomial number in n, as n increases, 2 to the power n is exponential and therefore rounding off is not a polynomial way of solving it. But as you are always interested in algorithm that is polynomial, you will not take an algorithm the moment it is established, that it is exponential. Even if you do an exponential search and get these 2 to the power n and get the best, it is still not optimum; there is no guarantee. Therefore rounding off is not a very good strategy, even though it is followed by many equations.

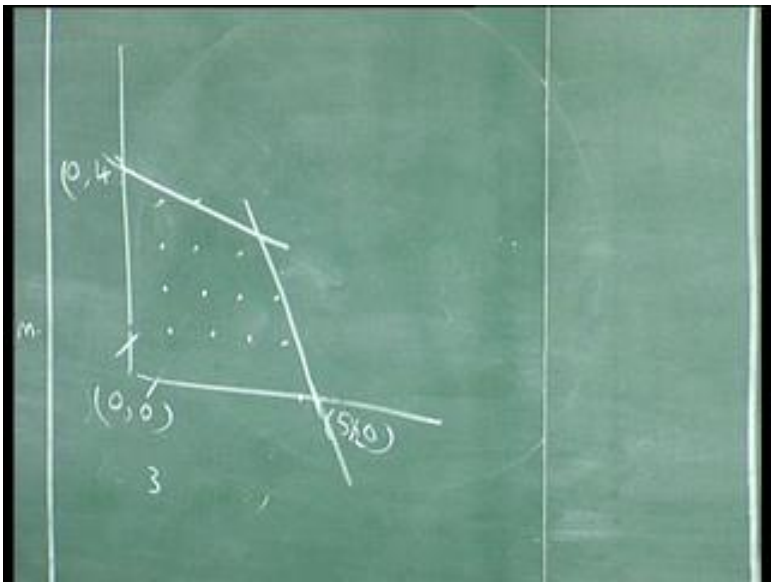
You will also know that out of these 2 to the power n, there could be a very large number that would be infeasible, because when you start rounding the mass on the higher side, for example, you will get points like this. This you will have four rounded points and in this case, it can turn out three of them is infeasible and only one of them is feasible. You can also have a situation where none of the rounded points are feasible, but there is an optimum somewhere else.

(Refer Slide Time: 38:39 min)



For example, you can create a feasible region like this. This is the LP corner point and let us say these are the four rounded points; none of them will be feasible to the IP. You could have an integer point somewhere here, which is optimum. These are all extreme examples. However, they are relevant, because we are trying to understand the principles involved. There are three aspects to it; one - rounding is exponential and therefore difficult to do; two - the best feasible solution among the rounded, need not be optimum; three - you can encounter situations where all the rounded solutions are infeasible, but there is an optimum somewhere. Therefore one should not do this. One should try and formulate the problem separately and solve the problem separately. What are the other implications? If we have an algorithm which will directly solve integer programming without using ideas from linear programming, it is another issue.

(Refer Slide Time: 39:58)



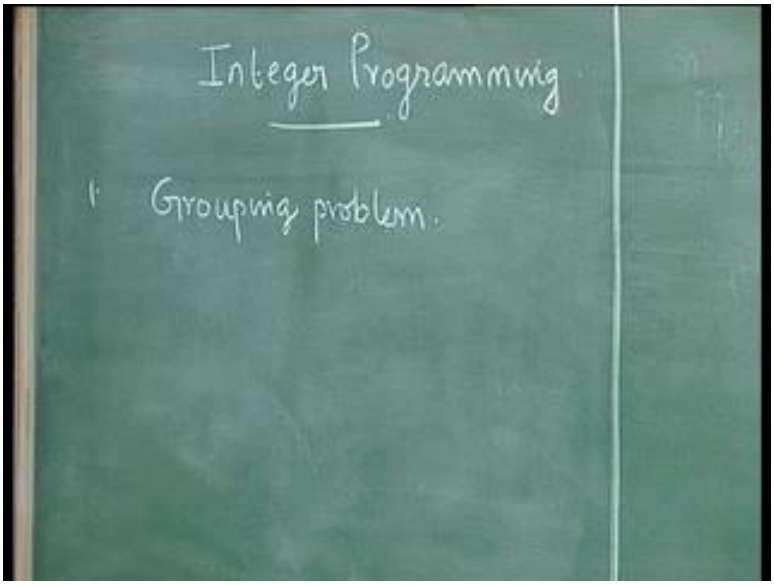
If you look at it, this discussion was based on the fact that you relax the integer assumption. Solve the resultant LP and then try to get the integer optimum from the linear programming optimum, that was the hypothesis or that was the assumption which led us to the discussion.

Let us go back and see: let this be the feasible region,, what we actually want to do? If we can identify all the integer points in this feasible region and then we substitute all the integer points into the objective function we can get the best, which is optimum. It is only a question of trying or to be able to find out all the integer points which are there. For example, your integer points are, the moment you start looking at this way, these are your integer points and so on. Your feasible region for the IP, is not continuous; it is a collection of discrete points. You see, this point is infeasible, this point is infeasible. Whereas, in linear programming, your feasible region is convex and the entire results of linear programming is built around the convexity property of your feasible region.

You cannot directly apply results from linear programming and say we can do this, look at the corner point. You need to still be within the linear programming territory and do things in such way that the convexity of the feasible region is not lost. Then somehow you try to get corner point to an LP, which is integer. Most integer programming algorithms are developed with that thing in mind.

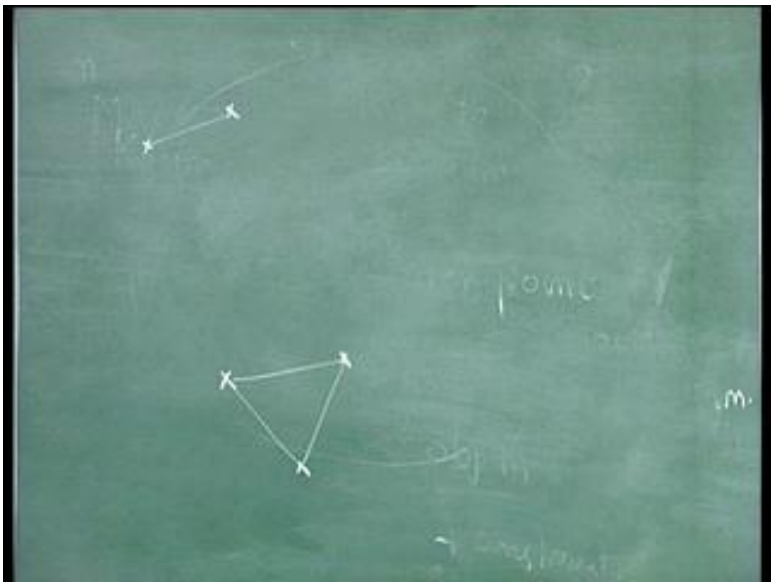
We will always be solving LPs, because we do not want to lose the convexity property of the feasible regions, and somehow do something such that the LP optimum becomes integer. Please remember, that in all IPs if you relax the integer assumption and solve it as an LP and if the LP optimum happens to be integer valued we have solved the LP. That is the best thing that you can hope for whenever you are solving IP. You solve it as an LP and you hope that the LP optimum is integer valued so that straight away you have to solve the IP. If it is not, then you still do things which are under the purview of LP, keep solving LPs in successive iterations and somehow get the optimum LP to be integer valued so that it is also optimum IP. Algorithms to solve integer programming problems are developed that way. After this, let us go back and try to formulate some practical situations which are integer programming problems.

(Refer Slide Time: 43:11)



We start with one simple example called the grouping problem. Let us define the grouping problems and see whether we can complete the formulations.

(Refer Slide Time: 43:27)

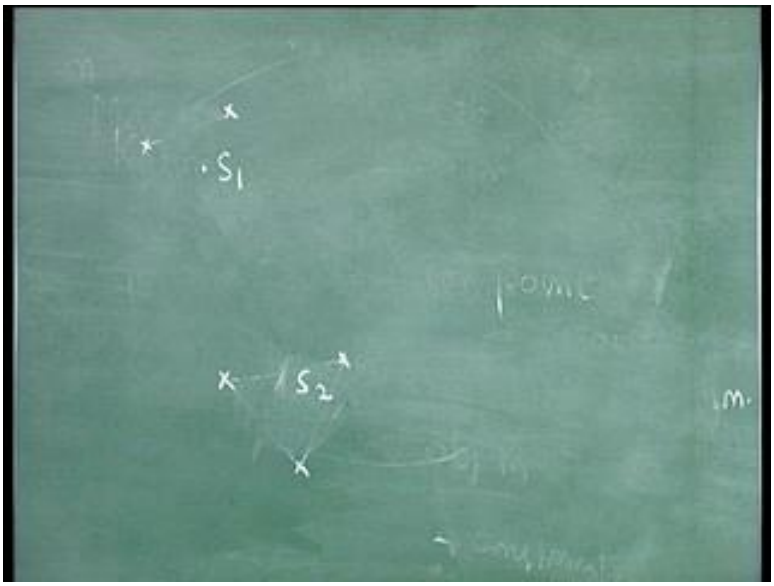


Let us assume that we have five points which are like this. If you look at it normally and if we ask anyone of you to come up with two groups, you will quickly go back and say, these two will form a group, and these three will form a group, very obvious. This is under the assumption of minimizing the distance among the groups.

How do we first formulate this problem and then try and solve them, to get this, how can we formulate this?

One of the ways of formulating it is this, even before we define the decision variables let us look at this.

(Refer Slide Time: 44:13)



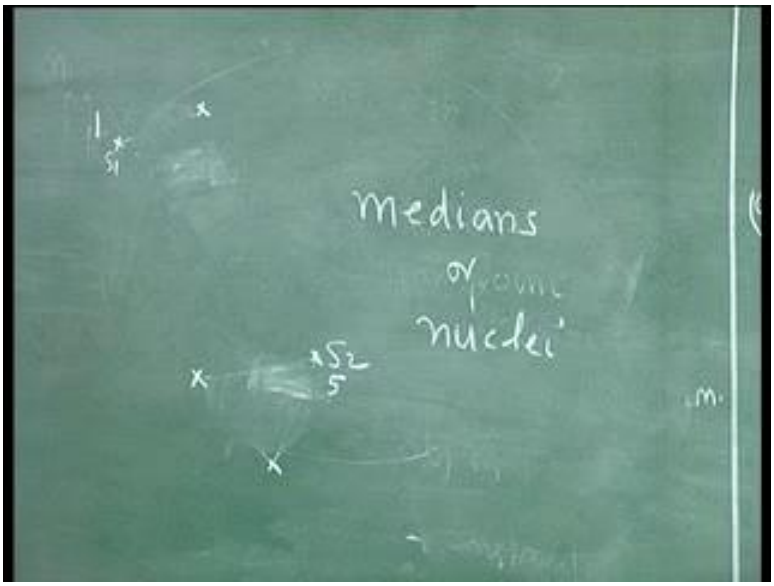
If we know the number of groups and if we want to solve or identify the groups, for a known number of groups, say two in this case. What we can do is we arbitrarily define two seed points called, S_1 and S_2. The moment you defined two seed points S_1 and S_2, we will go back and try to find out the given points 1, 2, 3, 4, 5; we will try to assign them or cluster them around the seed points. That is a very simple thing to do. These two will go to this seed point; these two will go this seed point.

Now what is the difficulty there?

The difficulty is not in assigning or clustering around the seed point, but the difficulty is in creating the seed points. For example, if we had arbitrarily created, my S_1 to be here and my S_2 to be somewhere here, then you will see that all the points will cluster around S_1 and you will have a null cluster around S_2.

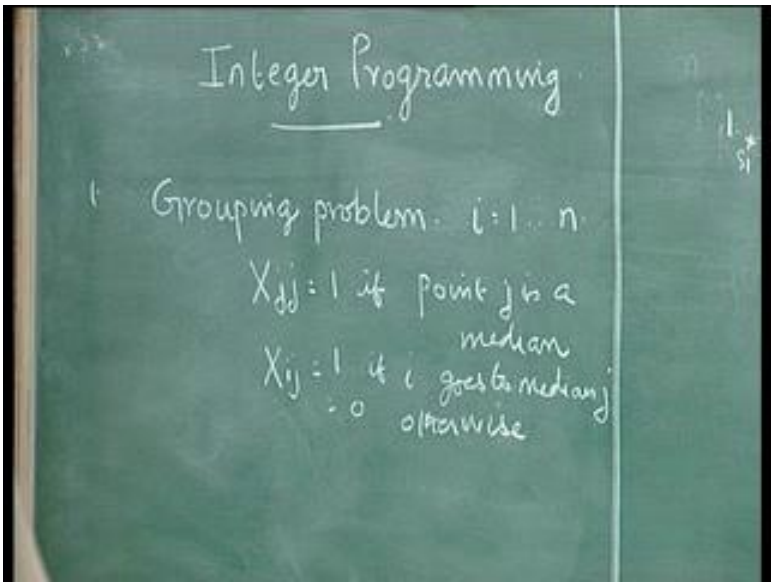
When we say that we want a group, in that case the first group will have all the 5 and the second will have 0, which means you have only one group. When we say that we want two groups, a group is characterised by the presence of at least one element. Therefore, what we do is, we try to not identify seed points that are different from the given points. We will simply go back and assume 2 out of the 5 given points as seed points, so that we know for sure that we will not get null groups.

(Refer Slide Time: 45:50)



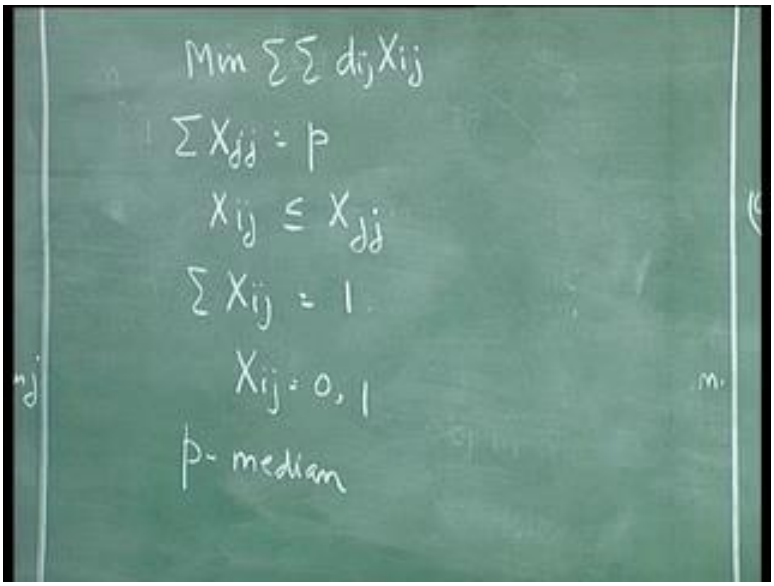
I could assume this one to be S_1 and this one to be S_2. So we are guaranteed that, this 1 will form a group; 5 will be in another group, and the rest of them will get mapped into different groups. If we are going to form a group based on the distance criterion, then we assume that we know the seed points, we can quickly do a distance criterion and form the groups. What we want to do is, a good way by which we pick 2 out of the 5, or in general if we want p groups, we fix p out of the n. as seeds. The moment we fix p out of n as seeds, because of my objective function which is to minimise distance, we will quickly cluster the rest of the points around the p seed points. These p seeds points are called medians or nucleus points, nucleus, nuclei and so on. We will call them medians.

(Refer Slide Time: 47:03)



The formulation will look like this: we have i points, i equal to 1 to n. We will first put this condition: X_{jj} equal to 1, if point j is a median. X_{ij} equal to 1, if i goes to median j; equal to 0 otherwise. The formulation looks like this. This is under the assumption that we know that p, the number of groups that we want.

(Refer Slide Time: 47:53)



What we will do is, my first constraint will be, sigma X_{jj} equal to p. 2 out of the 5, will take 1 and the remaining 3 will take 0. Whichever j is a median will take that X_{jj} will be 1. If it is not a median, then it will be 0. X_{ij} should be less than or equal to X_{jj}, because if we are attaching for example, if point 1 goes to median to point 2, then point 2 should be a median. If 2 is not a median, then 2 cannot build clusters around itself.

You can build clusters only around the median. So X_{ij} less than equal to X_{jj} would imply, that any point that is not a median, can only be clustered around a median. Sigma X_{ij} equal to 1 implies, that any i that is not a median can go to only one cluster. Then your X_{ij} is 0 or 1 and then you minimise double sigma $d_{ij}X_{ij}$ and your d_{ij} is 0, the distance between the point and itself is 0.

You assume that for the given 5 points, you have a 5 by 5 distance matrix and if there are points here, they will be symmetric, so you can define it right. You will get a formulation like this. This is one example of how integer programming can be used to formulate a reasonably complex problem. You will know that the grouping problem is actually a very difficult problem. This is called the p median problem or a k median problem, depending on whether you call this as p or k. This is an integer programming formulation of a grouping problem under known number of groups. You cannot solve this if the p is not known, the only thing you can do is you can solve for p equal to 2, 3, 4, 5, up to n minus 1 and then do something. But this is under the assumption that you know the number of groups that you want and having given the number of groups you want to get the best grouping that is possible out of this.

We will look at other types of formulation in the subsequent lectures.