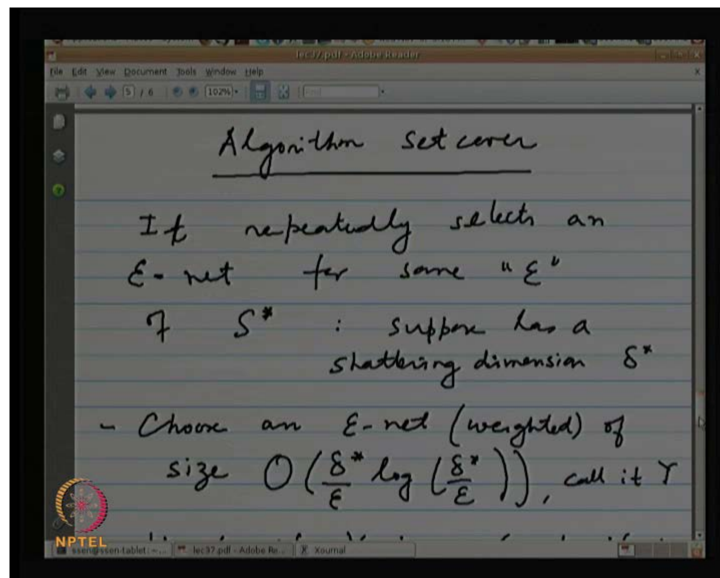
Computational Geometry
Prof. Sandeep Sen
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Module No. # 13
Epsilon Nets, VC Dimension and Applications
Lecture no. # 04
Geometric Set cover
(With Bounded VC Dimension)

So, we resume our discussion on geometric set cover. I would like to recap the algorithm that we did yesterday and we had just about started the analysis right.

(Refer Slide Time: 00:42)

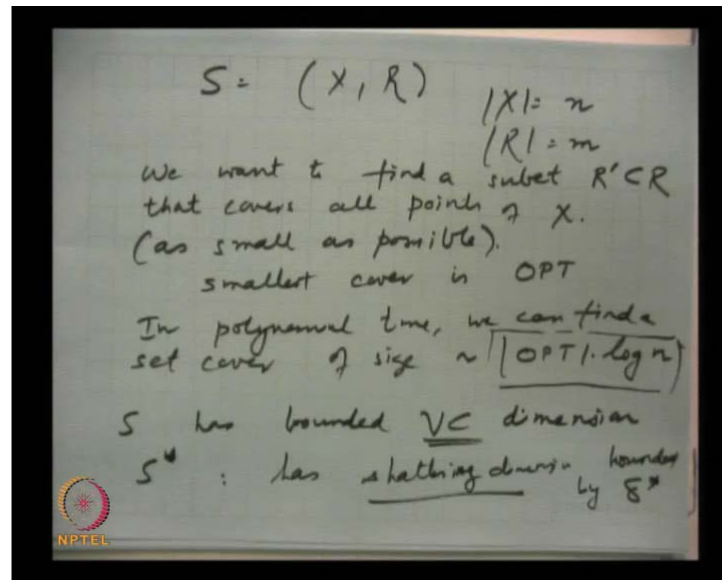


So, what was the algorithm, what was the problem? We have a geometric range space with some bounded VC dimension.

(No audio available: 00:51-1:00)

And given some ranges and given set of points X. We want to cover all those points X. Let us write it. So, problem was given, this range space X, R.

(Refer Slide Time: 01:22)



Were number of points is n, number of ranges is m. We want to find a subset, let say, R prime of R that covers all points of X as small as possible. Of course, the smallest size is often known as point optimum set cover. Smallest cover is some opt. So, this, the smallest size, smallest number of ranges that will contain all the points of X. It is known to be an interactive problem in the general case of set cover and it is also known to be interactable many geometric cover problems, including the covering of points by this. We want to avoid the cost incurred by a generic set cover algorithm, which is in polynomial time. We can find a set cover of size approximately opt times log n.

So, this is just using a greedy approach to the set cover right. You you you start with a set that contains the largest number of uncovered points, include it, update your points cover and then, look at the next set that covers the most numbers of uncovered points. You go on till all points are covered and that is your greedy algorithm and that achieves this bound. Surprisingly, using even more sophisticated methods which are more complicated than greedy method, you still cannot obtain a faster algorithm in the general case. For special cases, people know like in a vertex cover or you know where your sets are bounded size etcetera, you can you get better, better approximation bounds.

Now, in in in our context, again we do not have such restriction. Again, these are sets, you know these are arbitrary large sets, but they have this property. So, S has bounded VC dimension. Most geometric set cover problem will have this property. So, using,

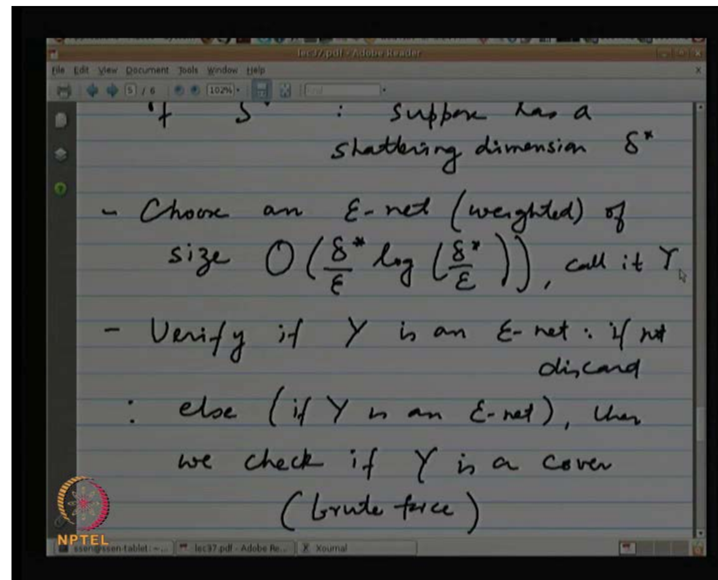
exploiting this bounded VC dimension, we want to get a set cover which is superior to this size, namely you know it does not have, we do not want any dependence on this n and this number of points. We want to have something that is a function of only opt, may be log of opt or something.

So, this is algorithm that we were discussing yesterday. What do we do for some epsilon; we will define what epsilon is exactly. So, for some epsilon, we pick epsilon net by the way. So, this problem, this set cover problem works in the dual range space because we are trying to cover, we are trying to cover points by by by ranges and we are using, going to use epsilon nets. So, normally when when you define epsilon nets, we do a random sample of the points, but here we are covering the points using the ranges. So, we are going to random sample the ranges. So, we look at the the dual range spaces. So, we look at S star. So, whatever we are doing is basically with S star. So, we are working on S star and S star we are saying has; let us say a shattering dimension. Now, I am not using VC dimension, I am using shattering dimension.

Again, there is a relationship between VC dimension and shattering dimension, but for for simplicity of calculation, we will just use shattering dimension bounded by VC delta star. The subset that we are picking from this dual range will be a sample of the ranges and that sample of the range is basically done by this epsilon net, the technique of epsilon net. So, we are going to construct epsilon net and we keep constructing this epsilon net with with by by by, it is a weighted epsilon net. So, we begin that all elements have unit weights that we revise the weights and we keep revising the weights and picking up this epsilon net and every time we pick up the epsilon net, we actually test whether or not it covers all the points. The moment it cover all the points, the algorithm stops and says ok, this is the cover and size of the cover is is you know as we will see is what we were looking at, namely that it does not have the algorithm factor.

So, let me just again quickly step through this, so that we can do the analysis. So, it repeatedly selects an epsilon net for some epsilon of S star we were working on the dual range space, which has shattering dimension delta star. So, we choose an epsilon net. Now, we are talking about choosing an epsilon net in a weighted set because initially, we start with weight equal to 1 for every range and then, we revise depending on which ranges we prefer to pick up in our epsilon net because you know certain points.

(Refer Slide Time: 07:20)



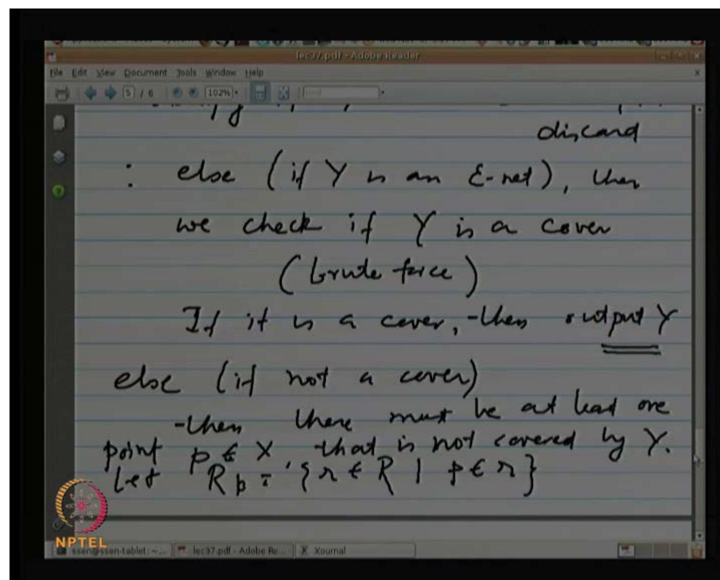
So, if it is not a cover, so what happens? So, first of all we pick up this subset Y, which is epsilon net. It is supposed to be epsilon net. If it is not an epsilon net, then we discard it. We do not even look at it further. You first verify whether it is an epsilon net. How do you verify if it is an epsilon net? Basically, just brute force verifies if it is epsilon net or not. So, look at all subsets of, so we have m subsets right, we have m ranges and n points. So, we brute force of that, so whatever time it takes n times, it is polynomial right. So, we will we will look at that, verify whether it has hit all the large subsets of size at least epsilon net. If it is not, then we discard it. Do not process it further, but if it is an epsilon net, then we examine Y with respect to whether or not it is a cover.

So, it may be that the one that we discarded, you know may have contained the cover, but we discarded never the less. So, because you know the the probability of not being an epsilon net is less than half. So, whatever we do, even if you consider only those samples that are epsilon net, we are missing out may be a factor or 2 in the process. It will be much easier for us to analyze only those samples that are actually epsilon nets. That is why we do not even retain or consider those samples that are not epsilon nets. If it only passes the test, we go to the next step.

The next step is to verify if the subset, all the epsilon net covers all the points of X, and if it covers, then you stop an output, which means that essentially this epsilon net Y of the size delta star over epsilon log of delta star epsilon is the size of the set cover. You can

see that this does not have any, well we do not know epsilon really, may be epsilon have some dependency on it. We have not defined epsilon net yet. Let us assume for the time being that epsilon does not have any dependence. So, this is the size of the set cover which is only dependent on the shattering dimension, which is the property of the range space and epsilon that we are choosing carefully. Ok, but if it is not an epsilon net, what you do? If it is not an epsilon net, sorry if it is not a cover that you check whether, so you you find out some point that is not covered. So, there has to be at least one point, let us say p that is not covered by this set of ranges.

(Refer Slide Time: 09:36)



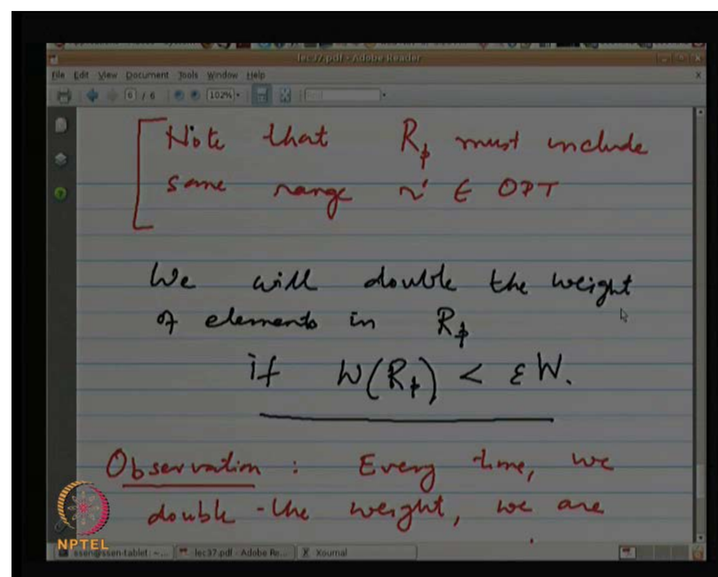
So, look at all those ranges that contain p. So, that we are denoting as R sub p. So R sub p is a set of all ranges that contain the point p. Now, one of those ranges that must also be present in the optimum set cover because in the optimum set cover, the point P must be covered, and the point P can only be covered by one of those ranges. So, what we are doing? So, now we do not know which of those ranges is in the optimum set cover. So, we blindly sort of increase the weight of all those ranges in the R sub p by factor of 2, so it doubles the weight. So, that what we are doing is, so that we sample next time, we are actually preferring that at least one of those ranges get picked up by sample, so that the point p is covered next time.

So, we just keep repeating this process. We we pick a sample of this epsilon net size, check whether it is epsilon net or not, if no discard it. So, we process to see that if the set

of ranges Y actually covers all the points. If so, we are done. If not, we look at which point it missed out, pick up any of those point, arbitrarily pick any of those points, look at all the ranges that contain point and for those ranges, we double the weight. So, in the process what happens is that every time we fail to pick a set cover, we are certainly going to increase or not increase, double the weight of at least once range in the optimum set cover, right. Every time we miss a point, at least one range that contains the p must be in the optimum set cover and since, we are doubling the weight of all the ranges, so even that range will double its weight because certainly, every time we have to repeat the the we the the the the optimum set is certainly gaining very rapidly weight.

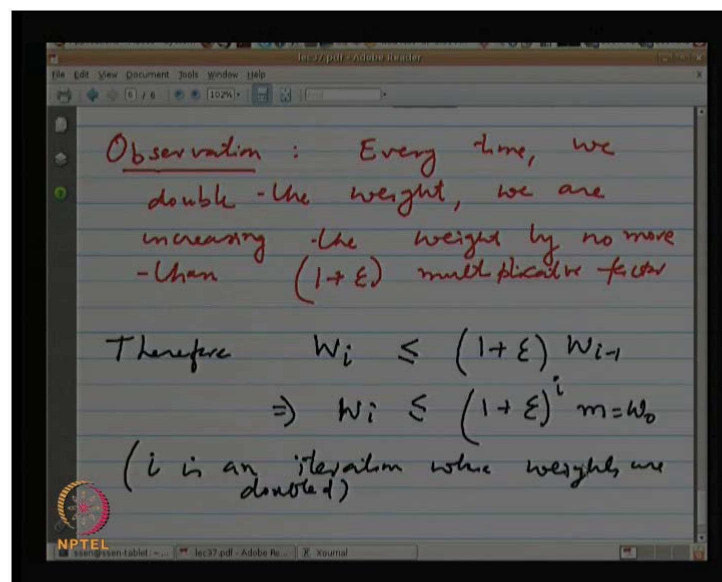
So, the whole idea is that, therefore very soon we should basically stop because this we cannot go on forever because the weight of the optimum set is really doubling and since, it is doubling, it is going to sort of you know dominate it at some point. We have no choice with, the sampling algorithm will have no choice, but to actually output that set or something that is very close to that set. That is an intuitive idea, but when we do the vigorous analysis, you know what is happening is we we we make, we do one more change and that is we will double the weight. So, we do not double all the time, we double only when the weight of this ranges that contain the point p is less than epsilon W. The reason I gave for that was see, if the since it is an epsilon net, if the weight was greater than epsilon W, we would have certainly covered p right, because the weight of those weight of those ranges that contain p exceeds epsilon W.

(Refer Slide Time: 12:21)



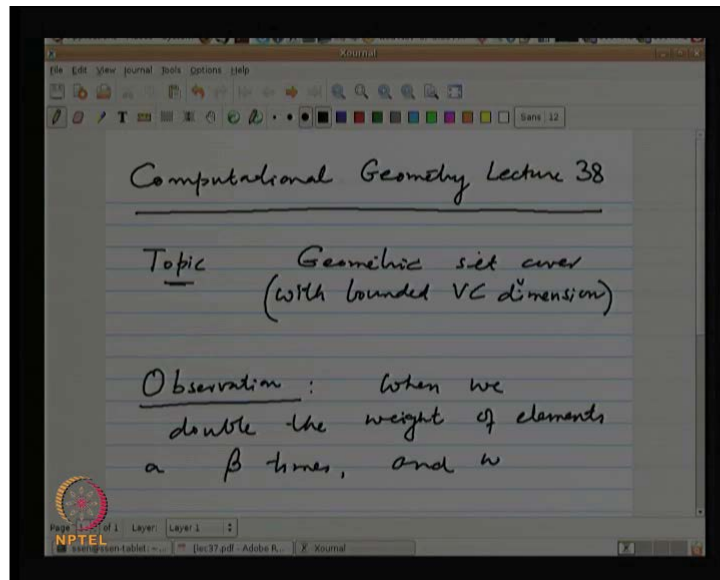
So, my epsilon net must hit one of those ranges. So, only if it is less than epsilon w, we will double because if it is less greater than epsilon w, in any case it is good it would have been picked. So, if it is an epsilon net and this point p was not covered, it means that the weight of those ranges is less than epsilon W. So, when we double the weight, what are we doing? We are increasing the weight, the total weight of all the ranges by upmost epsilon W right. So, this will help us analysis right. So, every time we double the weight, we are increasing the weight by no more than epsilon W, which is basically 1 plus epsilon multiplicative factor of the of the entire weight right.

(Refer Slide Time: 13:59)



Therefore, W i is less than equal to 1 plus epsilon W i minus 1, which is basically this quantities and so, we will carry on from here. So, this is one way of bounding the weight, this is upper bounding the weight. So, the weight at the ith stage of all the elements put together is no more than 1 plus epsilon to the power i. Now, let us continue with this calculation.

(Refer Slide Time: 14:20)



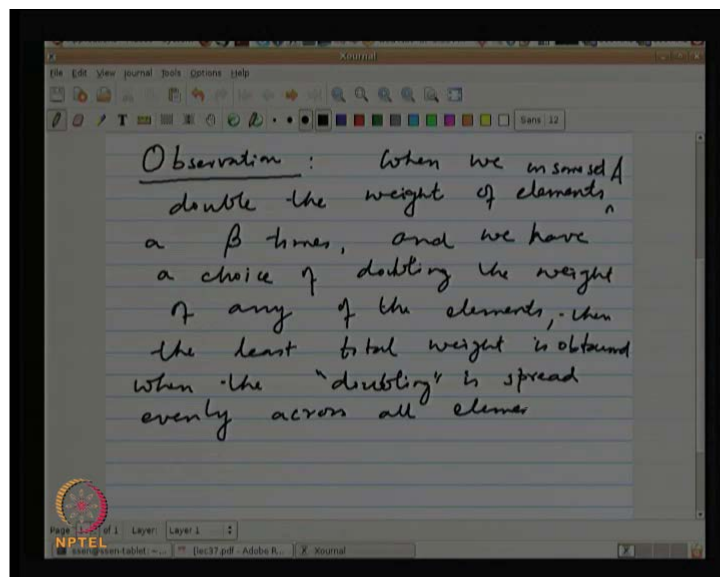
So, the other way that we want to bound the weight is when we double, so, here is another observation.

(No audio available: 14:28-15:12)

Let us say, let me use some numbers, elements. Let me take some value, let say, beta times.

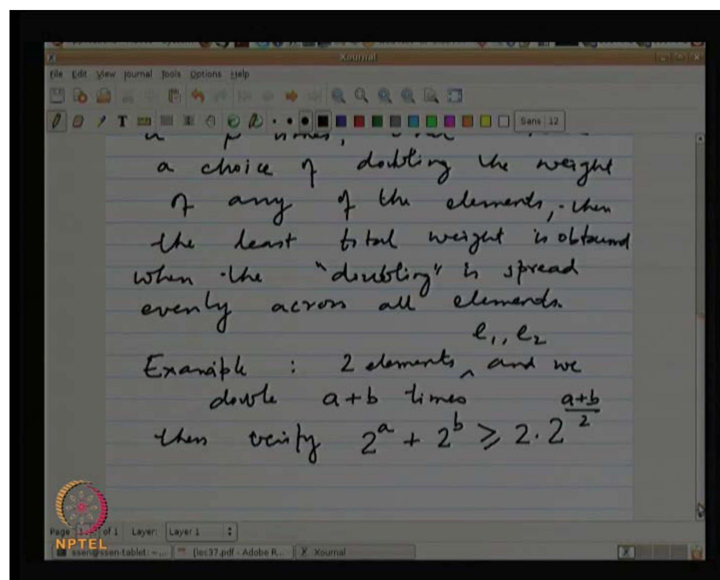
(No audio available: 15:35-16:20)

(Refer Slide Time: 16:23)



So, a is a set of elements that we are doubling the weight. At any time, every time we double, we have the option of picking one elements of a, and double the weight. So, when we have this option of picking up any arbitrary a, and doubling, the claim is that the least and after while, we look at what is the total weight of all the elements. Suppose, all the elements started being in weight 1, we are doubling, picking up the arbitrary element, double, you put it back again, again double, put it back. So, the claim is that the least weight is obtained, when the doubling is kind of spread across evenly across all the elements. Then, the least total weight is obtained when the doubling is spread evenly across all elements.

(Refer Slide Time: 17:41)



So, you can actually formally prove it, but let us say that you have just 2 elements, let say a and b. So, well not 2 elements. Yeah we have only 2 elements, so example say 2 elements and we double 2 elements, epsilon 1, epsilon 2 and we double a plus b times. Then, verify that 2 to the power a plus 2 to the power b is less than equal to 2 times 2 to the power a plus b plus 2. Can we verify this?

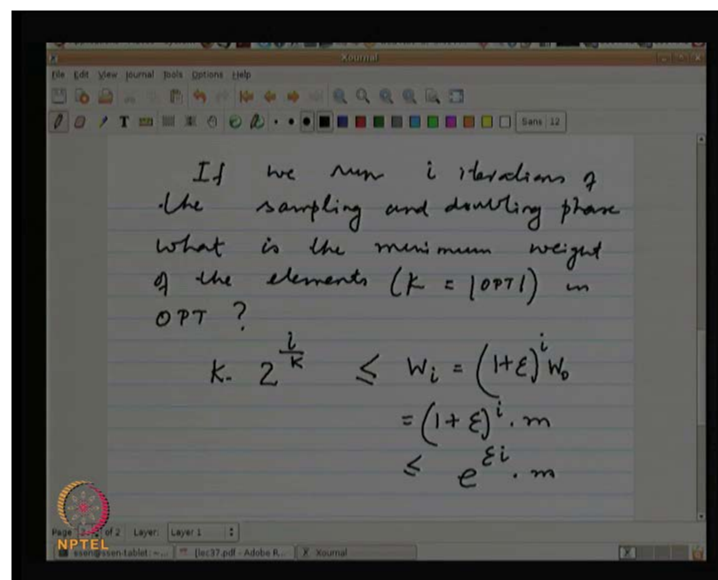
(Audio not available: 18:40-19:00)

Yeah, you can yeah, but I think i think even in this example you can probably verify right. So, I am not going to give a proof of this.

(Audio not available: 19:10-19:17)

You want to put in some figure and check, a equal to 3, b equal to 3. Yeah yeah ok. So, you you basically extend it to any, I mean you are right absolutely. So, you extend it to other elements. So, will what means what it implies is that, when we look at the set of elements in the optimum set cover, so among other elements; those elements could also be increasing their weight right. What now what we are claiming is that, this algorithm as as we are done more and more iterations of the algorithm, the elements, also the ranges in optimum cover, some of them are going to double their weight and this one claims, this observation basically says that after, let us say some running it at some t times, what is the minimum weight gained by the elements in the, sorry the ranges in the optimum set cover right.

(Refer Slide Time: 20:40)



(No audio available: 20:34-20:40)

So, if we run i iterations of the sampling and doubling phase.

(No audio available: 21:02-21:24)

So, there is K elements in the optimum set cover. So, we will make an assumption that we know K. If we do not know K, we will see what to do later. Suppose the optimum set cover has size, so suppose K.

(Audio not available: 21:36-21:46)

So, what is the minimum weight of the elements in opt of iterations. So, according to our previous observation, it should be spreaded evenly across all the elements, that should be the least right. So, it will be so each of the K elements should be doubled i over K times right. I am missing some floors and ceilings you know. Do not worry too much about that right. Do you agree with this?

(No audio available: 22:25-22:34)

So, this is minimum weight of this, right. We also have from the previous lecture an upper bound on the weight, right. So, I can write less than W i, W i must be greater than because this is the weight of the, it is just the weight of the elements in the optimum set cover. There are other elements also, so this is certainly lower bound of that. So, this must be, so inequality must hold. So, W i which is when we are saying 1 plus epsilon to the power i times w naught. W naught is the initial weight which is each range has weight exactly equal to 1 and we have m ranges right. So, this will be equal to 1 plus epsilon i time m, right and we can upper this bound, this by something like e to the power epsilon to the power epsilon m, i times m, like 1 plus epsilon net than e to the power epsilon.

(Refer Slide Time: 24:30)

Now, we just have to solve this. So, that will give us some bound on the number of iterations. So, if you take log on both sides, it may be so, let may be, so let us to make the calculation easier, suppose i equal to k times q or some such thing. So, then we can

rewrite as k times 2 to the power q greater than e to the power epsilon. Can you just tell me what the solution should be if it logs on both sides? Log k plus q equal to m plus, I am missing some constant factor. This is e, this is 2, so there is some constant factor that I am missing here. I am just being a little sloppy. So, what does it say? i is, should i be less than or i should be greater than?

(Audio not available: 25:42-26:34)

Oh right right right right. So, I am getting, I should not have i, I should have I thought this something. We are solving for q right .

(Audio not available: 26:58-27:19)

So, it should be that is what we want, right.

(Audio not available: 27:22-27:59)

Now, what this says is that q should be is less than or equal to something. For that inequality to hold, we cannot go on forever. So, you must end somewhere. Q cannot be too large, so we have and for this to be positive, now you can define your epsilon, right. So, as long as epsilon is less than 1 over k or something, this left hand side will be positive, right, so that we can basically have a solution that q is less than equal to. So, by let us, now we can choose epsilon right. So, we have not chosen an epsilon till now, so suppose, we can choose epsilon, right. We are choosing epsilon of the epsilon net.

Suppose, epsilon net equals to 1 over 2 k, right then q is less than or equal to order of log of 2 times. So, this says that we will end in some finite number of iterations. Number of iterations is bounded by k times q, right. So, number of iterations. Well, it is not quite a number of iterations; it is a number of doubling steps. Iterations, where weight doubles, right is less than equal to k times q, right. So, order k times.

(No audio available: 29:45-30:00)

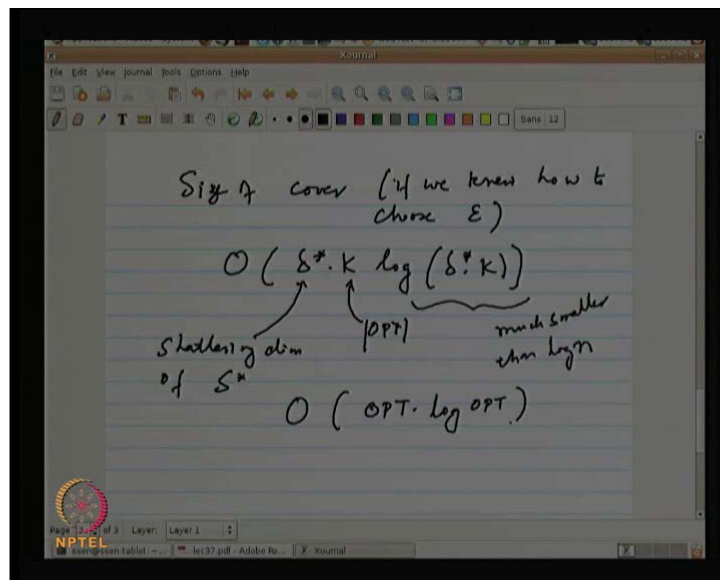
So, m is the number of ranges and k is the size of the optimum set cover. Well, how do you choose the epsilon, if I do not know k. If I knew k, then I can choose epsilon to be this, right and I have to know epsilon because I am picking up epsilon net for some known epsilon, fix epsilon. So, if I knew epsilon, then my total number of iterations is is

this quantity. If I knew that, if somehow could choose epsilon to be, actually if I choose epsilon to be less than 1 over k, then itself you know we are going to converge, but how do we know epsilon. Which epsilon are we going to use within the algorithm? The algorithm is very simple. Choose an epsilon net for some epsilon. If it is not an epsilon net, discard. If it is an epsilon net, see it is a set cover. If it is not a set cover, then double the values of some ranges that contain the uncovered point p and keep on going. So, the size of the cover is fixed. So, the size of the cover is this.

(Audio not available: 31:13-31:22)

Now, we know epsilon also, right. So, the size of the cover is delta star, epsilon is let us see if it is 1 over 2 times k. So, size of my cover is delta star times K log of delta star times K. That is the size of my cover. So, size of cover, if we knew how to choose is order delta star times K log of delta star times k. Now, in this quantity, delta star is the shattering dimension, the dual space, does not depend on n. K is the size of the optimum cover, right, is the size of the optimum cover which do not know really. This is the shattering dimension of S star.

(Refer Slide Time: 31:50)



So, we have succeeded in our goal that we have a set cover whose size does not depend on, does not have that logarithm factor, log n factor right. It only depends on delta star; it only depends to the optimum size. So, this log could be much smaller than log of n. That was the goal.

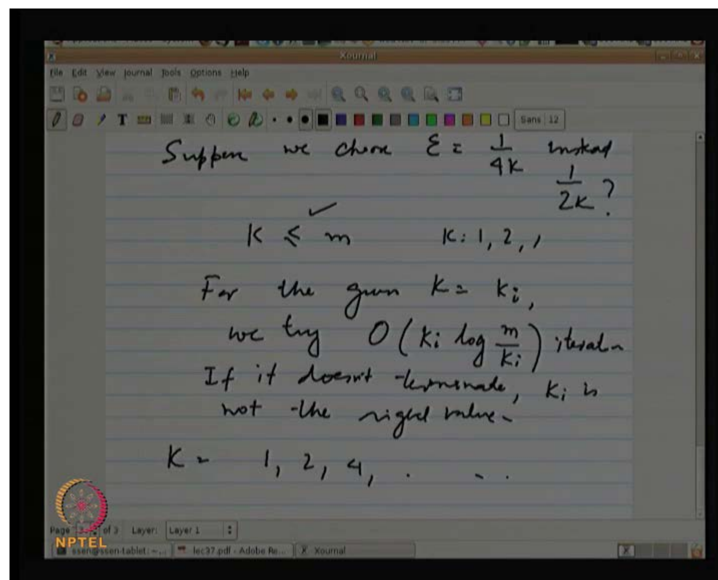
(Audio not available: 33:07-33:17)

Why? See delta star is the shattering dimension, right. That is a fixed quantity; that is a constant. So, we have something that is basically big O of opt of log of opt, right. So, you can think of like, big O of opt times log of opt, instead of opt times log of n. So, the only missing ingredient here is that how do we choose epsilon?

(Audio not available: 33:46-33:56)

I do not know. K epsilon is someone over some 2 K over 1 over 4 K or something. So, will it make any difference, just wondering? Suppose, we choose K, sorry epsilon to be equal to 1 over 4 K instead of 1 over 2 K. Will it make any difference really in the asymptotic calculation? It is again just constant, right.

(Refer Slide Time: 34:05)



So, does it give you any idea? In what strategy? What will be the simplest strategy? Greedy what? So, 1 is that you guess the size of k, right. I mean I can try for k equal to 1, 2, 3 K will be something, right. I can try for all kinds of k, I can try for k equal to 1, 2, 3, 4 whatever we want, right and k has some bounded size. This optimum size k certainly equals to m. That will be one way of achieving it. It will be not a very smart way of doing it. So, we know that K is less than equal to m. This is the size of the range space. This is the number of ranges in the optimum cover, so this is true.

So, I can start with K equal to 1, 2, 3 and run this algorithm, for every value of K run the algorithm and pick the one that give me the best size cover and by this, we are guaranteed that, you know with the cover size is not going to be more than this quantity because for the right value of K, it will be this quantity. For any other value of K if it is small, we accept it. If it is and you know that is all and it cannot. Surely, there will be at least 1 term if we try for all values of k. So, the minimum of that cannot be more than this.

So, you are not be if you do not converge, you know after some, so what you can do if it is not the right value of k, so you will run it for some constant times, whatever value of K we have chosen time log of this, if it does not converge, give up. Ok.

So, let me write it down for the guess, K equal to K I, we try order k's of i that log of iterations. If it does not converge, if it does not terminate, we know that this is not a right value.

(Audio not available: 36:56-37:09)

So, if you only trying to guess the value of k and from our previous observation, if we can guess it even within a factor of 2 years, done. So, why should I try out all the values of K? We should simply try out K equal to, whatever you know 1, 2, 4, just double. If you do the doubling trick, you know you will converge much faster. You go into within a value of, within a factor of 2 very quickly within whatever. Lock it, lock a times, right. So, that will save the, whatever. It will make the running time much better, but even this one is you know trying for all values of K is also a polynomial time. This is also polynomial time after we are looking a polynomial time approximations, but this is much faster and better scheme. So, that is basically the full description of algorithm.

Algorithm is very simple. It is just sample a certain number of element, according to the bound of the epsilon net, sample uniform yet random, but you have to do the weighted sampling because your weights are going to change. So, the only non-trivial part of algorithm is to implement a weighted sampling, but even that is not very difficult. So, once you do the weighted sampling, you keep doing it and you get it. Any questions?

(No audio available: 38:24-38-37)

So, using so what is whatever we discussed actually is a fairly general scheme. What we are doing? We are doing the following. There is some hidden set that we are trying to find. The hidden set is basically the optimum set, optimum set cover here. The hidden set could be, let us say we are trying to solve linear programming. The hidden set is the set of those constraints that define the optimum in d dimension. We know that d constrains or d hyper planes define the optimum, right. Somehow, we need to choose those d hyper planes. So, you can use a very similar approach there, that you, what do you do. You you choose epsilon net of some size, you know according to the linear programming has, you know its own VC dimension etcetera. You choose them, you compute the optimum of the sample and then, you find out if that optimum satisfy all the other constraints.

So, there has to be whatever we are doing, there has to be a check, whether or not should we terminate. The termination condition in case of set cover was that, whether it covers all elements. The termination condition in the case of linear programming is that, whether or not the optimum of the sample is actually the optimum of the entire thing and you just continue like this. If it is not optimum, there has to be some violating constraint. Then, you are going to double, basically weights of the, weights of all those. So, here you are going to double the weight of all the hyper planes that do not contain that point.

So, there is some optimum we computed for the sample. There are some hyper planes that violate that optimum. So, clearly one of those hyper planes must be in the optimum, so we are again just double all those weights and we keep doing this. So, in this particular algorithm, we will again at some point, once we pick up all the optimum constraints in the sample, then the optimum of that sample is going to satisfy everything. See the d optimum constraints, right and if you can pick up with those d optimums in our samples, the d optimum constraints the sample and we compute the optimum that has to be the optimum of the entire set of constraints. So, here the the hidden set is d. So, again we are going to do this doubling treat, and after a certain number of iterations, we are going to find the optimum which will satisfy everything. So, what we get in the end is the true optimum.

There of course, the critical thing is how many iterations do we require to find that? So, it requires some non-trivial calculation and again the state of the art of linear programming is such that, you know it is not really the polynomial time because there is no what is called no strongly polynomial time for linear programming. So, this method

that we are using will give you something strongly polynomial if it succeeds. So, because it is open problem, whether or not there is a strongly polynomial algorithm for linear programming, but it gives something that is you know, let us say better than what is known by other methods. So, this doubling tricks works there also and there are many other applications of this doubling trick.

(Audio not available: 42:12-42:18)

No, they are polynomial. They are not strongly polynomial. So, the difference between polynomial and strongly polynomial is that, you know if the number of iterations of this algorithm does not depend on the size of the numbers. So, linear programming in the constraints what do you have? You have, essentially each hyper plain have a coefficient, so whether it is (()) algorithm, whether it is (()) algorithm, the running time is proportional to not only you know the number of constraint that you have, number of constrain it says n. Another parameter is dimension, so nd. So, fine it is going to be proportional input size anyways, right. I mean input size meaning the number of coefficients that define the problem.

So, n is the number of coefficient in each constraint, sorry d is the number of coefficients in each constraint because d is the dimension, n is a number of constraints. So, if you have something that is just polynomial n and d, n square d to the power 5 or something, then that is polynomial time, strongly polynomial, but if you have some algorithm that behave just following n square d to the 4 and l to the 10, where l is the size of the coefficients, number of bits in the coefficients. So, of course, you know l is the number of bits in the coefficients. So, that must be counted in the input size.

If you look at the definition of the polynomial time, you know it is fine, it is still in the polynomial time. So, this is absolutely polynomial by the definition, but the strongly polynomial time algorithm definition is that it should not have dependence on the size of each number. So, whatever, be the size of each number, the number of whatever the running time of the algorithm should be immune to that. So, if I am have using largest coefficient I am going to pay some more price for it because when I multiply, when I add I know it is going to be more, but then the convergence of the algorithm should not depend on that.

So, the simplest by the way, the strong, but not polynomial, but it is strongly what is been proved, very very strong property has been proved recently right. I mean it got the (()) award this year. So, you have a situation where do, what is called smooth analysis. So, earlier it is what was known is that average on simplex of the polynomial.

Now, what is averaging done over? Averaging is done over you know all possible instances are equally likely, and that you know that is an assumption you know which is too strong and you know, no one. Why should we make that assumption? So, that assumption is not neither valid nor desirable, but at least there was a result, a deep result. In that case, simplex is not only polynomial time, it is very close to linear. Then, about 7-8 years back, it was shown that some variations, strong variations of those average thing that is, you know you take any instance of linear programming and average in the neighborhood of that instance. So, you give me any arbitrary instance and then, I am allowed to average over a neighborhood of this instance.

If I take it an epsilon net, I perturbed the input by some amount you know that there is a parameter. Let say some epsilon net neighborhood and if I allow to perturb over epsilon neighborhood and look at the average running time in this neighborhood, then that turns out to be polynomial. So, that is called a smooth analysis. So, that holds for that is a notion of analysis, which is called smooth analysis. So, we are looking at something close to worst case, but not quite worst case because you give me an instance, that instance may be very hard, but then I am allowed to look at a neighborhood of that instance and I do my averaging over the neighborhood. Whatever instance you give me in the neighborhood, it is polynomial time. That was shown for linear programming and that result got what is called an (( )) award, which is highest price given to theoretical in computer science scientists. So, that is non-polynomial programming.

Then, again it is not a strong polynomial time, it is some and it is something like, I do not know maybe a 120 page proof or something like that, not pretty. These are essentially what we are saying; these are numerical analysis based methods. Anything that is, these are linear algebraic techniques. So, they are all linear, they are numerical analysis kind of analysis that you have to look at the convergence of the matrices; the size of the number really matters. All right, all those things. So, that problem remains open.

(No audio available: 47:00-47:12)

Anything else? So, we have more or less ended what we wanted to cover, at least all the basic problems that we wanted to address. In the next couple 2 or 3 lectures that we have left, I think Professor Aggarwal will talk mostly about, you know some higher level applications of these things to other areas, like will be graphics or image processing, something like that.