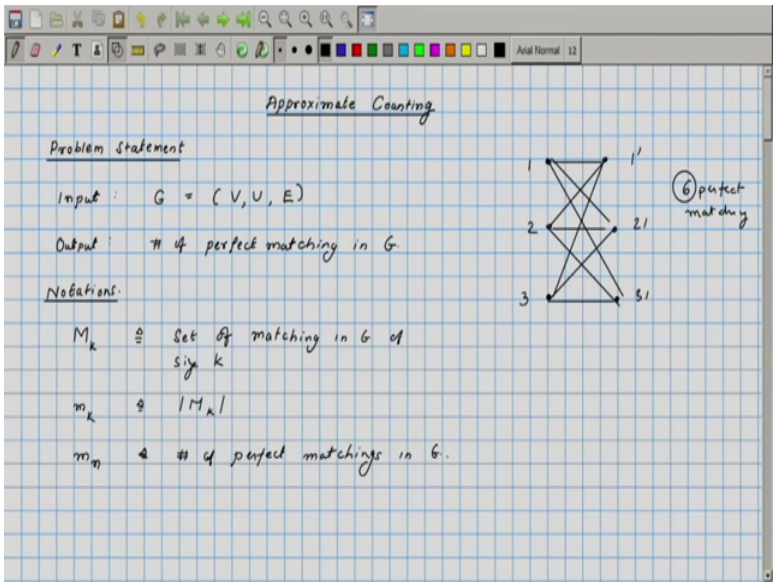
Randomized Algorithms
Prof. Benny George Kenkireth
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 31
Perfect Matching - I

So, in this lecture we will look at the problem of approximately counting some combinatorial objects.

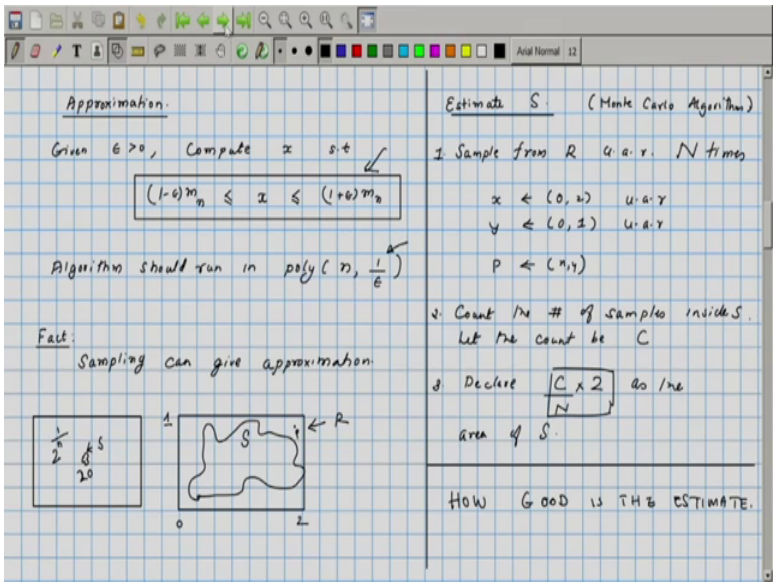
(Refer Slide Time: 00:37)



So, let us look at the particular problem that we are looking at, our input is a bipartite graph. So G, this is a bipartite graph, the parts are V and U and E is the set of edges and the output will be the number of perfect matchings in G. For example, if we had this particular graph, bipartite graph. This is a complete bipartite graph. We can show that there will be 6 perfect matchings ok.

So, we need to get this number 6, we introduce some notations. So, when we write capital M k this will denote the set of matchings in G of size k and small m k we will denote the number of matchings of size k. What we are interested in is computing small m n. So, the exact count of number of perfect matchings is difficult, it is a very hard problem.

(Refer Slide Time: 02:51)



So, we will look at the problem of approximating this value m n. So, what does approximation mean here? So, given an epsilon which is a it is a real number between 0 and 1. We have to compute a number x, an integer x such that x is the estimate of the number of perfect matchings. So, this should lie between 1 plus epsilon times m n and 1 minus epsilon times m n.

It can deviate from m n, but not by much ok and we need an algorithm which when given an epsilon and the input graph computes an x which satisfies this requirement. Further the algorithm should run in polynomial time in poly n. So, the input graph has some number of vertices that is denoted by the parameter n. So, the algorithm should run in poly n and it should be polynomial in 1 by epsilon. It is a slightly relaxed requirement, in the sense epsilon if you think of as a number, the number of bits required to represent it may be only log 1 over epsilon; but we are allowing the polynomial to be little larger than that ok.

So, basically given any epsilon, we should be able to produce an approximation to m n such that value that we produce lies between 1 minus epsilon times m n and 1 plus epsilon times m n. That will be what we will call as an approximation algorithm for perfect matching. Now how do we compute approximate answers to these kind of problems? So, we lose the following fact ok.

So, this is the key thing that we will use in most approximation algorithms. So, sampling can give approximations. What does this mean? So, let us say that there is a rectangle of length 2 and breadth 1 and inside this there is some particular set. So, the rectangle we will call as R and the set we will call as S. The problem at hand is estimate S. Now, how does this related to approximation algorithms? Well, this rectangle is some set whose size we already know and we want to estimate the size of a subset of that rectangle.

We will spend a little more time on it later; when how is exactly translates into an approximation algorithm for perfect matching. But let us just look at this problem in isolation. We have the rectangle R and inside that, that is a small area S and we need to estimate the area of this particular region. So, one algorithm would be sample from R uniformly at random. This would mean, let us say x you take uniformly at random from 0 2 and y also you pick uniformly at random from 0 1 and your point is basically x comma y ok.

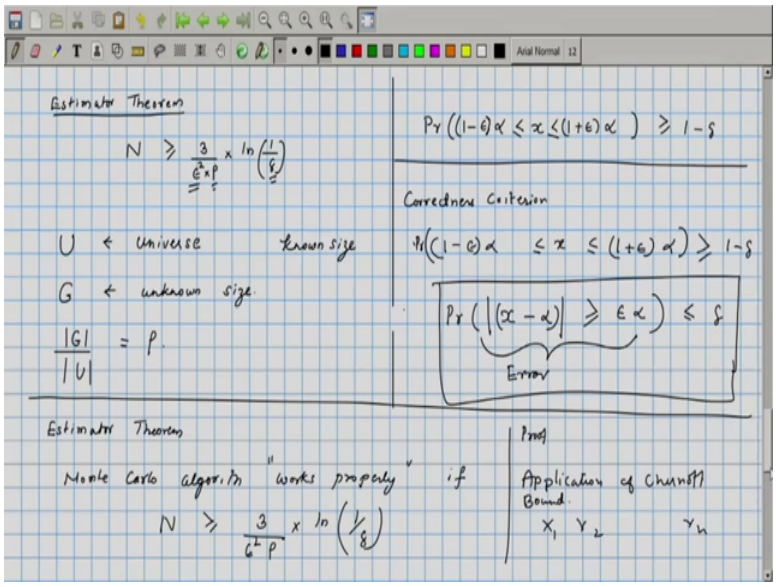
So, that is how you get one sample and check whether that point lies inside or I mean, outside the region S. Count the number so, we will do the sampling let us say N times. Count the number of samples inside S. So, let the count be C and we will declare C divided by N times area of the rectangle that is 2 here as the area of S ok. So, this is an estimate we need to wonder whether this estimate is a reasonable estimate that is one thing. And, the second thing is we should be able to sample uniformly from it say this rectangle which we can do if we have an access to a uniform random variable. And, we also need to check whether a given point lies inside this region or does not lie inside region.

Now, this region may be presented in a wide variety of ways and we need to basically check quickly in polynomial time whether the sample point lies inside the region that we had chosen. So, if you are able to do that, then this entire estimation works in polynomial time. Now, the important question is how good is the estimate? So clearly our intuition will tell us that larger the N is the better will be the estimate. But how large an N can be offered?

Suppose the area S was very small, suppose this was S then we know that we might have to sample more times than what is required here. So, if this is 10 times smaller than this, let us say 20 times smaller than this, will we will it suffice if we just sample 20 times

more? Or, do we have to do let us say 2 power 20? So, what is the dependency between the size of the area that we are estimating and the quality of our approximation? So, this algorithm is we will call it as the Monte Carlo algorithm ok.

(Refer Slide Time: 10:07)



Now, the estimator theorem will tell us why this estimate, I mean why this algorithm can work when the size of S is comparable to the overall size. So, what does this estimator theorem says? So, this says that, if you choose N to be greater than 3 by epsilon square times rho, that is the quantity that we have not discussed yet; times log 1 over delta ok. So, epsilon we already know what it is. We do not know what delta is, we do not know what rho is, but the estimator theorem says that if it is greater than this quantity then the estimate is reasonably good.

So, let us look at what is rho and delta. So, let us look at some particular set U whose size is known. So, we will call this as the universe which is rectangle for us in the earlier case and the universe is of known size ok. And, there is a set G whose size we need to estimate. If we are given one particular sample point from the universe, we can tell whether it belongs to G or not in polynomial time, but the total number of elements in G we do not know. The universe may be a large set its size is known. If we thought of the algorithm of generating all elements in the universe and checking whether it belongs to G or not; that is an exponential time algorithm usually because the size of the set U could be very large and we will call this ratio of these sizes as rho ok.

So, this in some sense captures our intuition because if rho was really small then the dependency on N is inversely proportional. So, as rho becomes smaller and smaller, the number of iterations it will have to run could become very large. When you are looking at algorithms, we can afford to have rho small, but not let us say exponentially small. 1 by let us say n raise to k is ok because, then the number of times we will have to run our algorithm is something like a polynomial will be 1 by a polynomial will be ok for us. Now, we need to understand what is delta. So, this algorithm this can give various answers at various times. It depends on how mean what all we sampled.

This was our correcting correction I mean; this is our correctness criterion that is the answer return should lie between 1 minus epsilon and 1 plus epsilon of the correct answer. So, we can look at probability that the let us call x as the estimate. It lies between 1 minus epsilon times the true value, let us call that as let us say m n and 1 plus epsilon times m n or whatever it was, is the value that we were estimating minus epsilon times let us say, if the alpha was the true value ok.

So, we could look at what is a probability that the estimate lies between 1 minus epsilon of the true value and between 1 minus epsilon times true value and 1 plus epsilon times true value. We could compute this probability or we could say that we want an algorithm wherein, this probability is let us say significantly high. This should be greater than 1 minus delta.

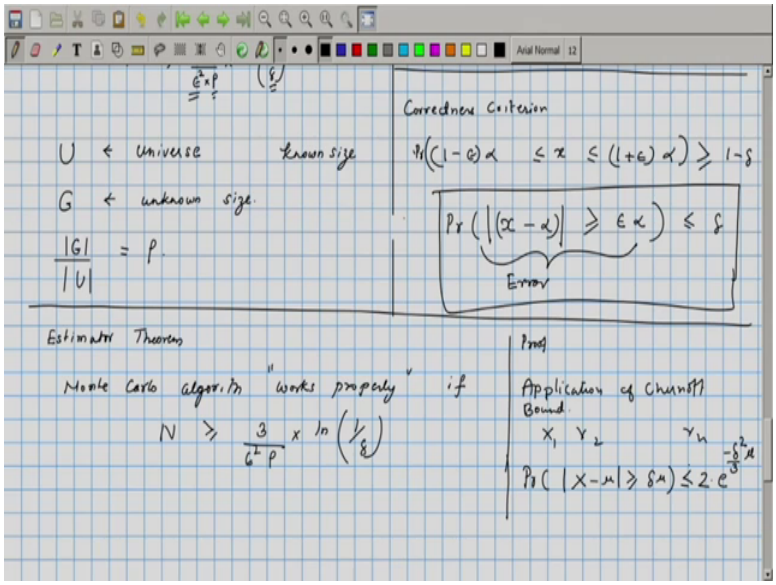
In other words, we want the, our algorithms correctness criterion are as follows, 1 x should lie between 1 minus epsilon times alpha and 1 plus epsilon times alpha with probability; x should lie between these, but that will happen with a certain probability. So, this probability should be greater than 1 minus delta. In other words, probability that the complimentary event happens that is x minus alpha the absolute value of that being greater than epsilon times alpha there is less than delta ok.

So, if you think of this as the error ok; then we are saying that the probability of error is at most delta. So, if we have this condition, how large an N should we choose so that this algorithm, the Monte Carlo algorithm works properly? And the estimator theorem states the following, Monte Carlo algorithm works properly. So, works properly means it satisfies the correctness condition. If N is greater than 3 by epsilon square rho times log 1

by delta ok. We will quickly see the proof of this. The proof is just an application of Chernoff bound ok.

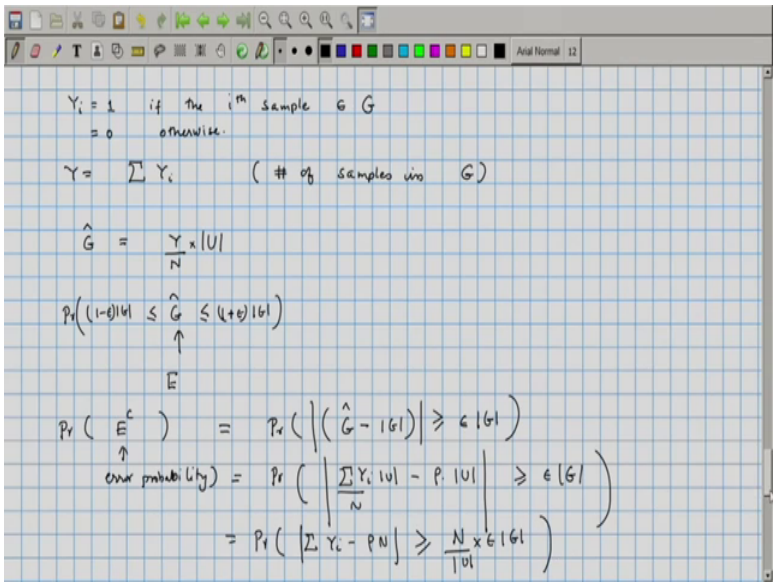
So, what does Chernoff bound say? If you look at i i d samples, of a random variable, let us say X 1 or X 2 X N are i i d samples and each X i is a random variable whose mean is some fixed number.

(Refer Slide Time: 17:25)



The probability that the sum of them X minus mu; mu is the expectation of the sum. This being greater than let us say delta times mu happens with probability which is less than 2 times e raised to minus delta square by 3 times mu ok. So, basically Chernoff bound says, this sum of Poisson trials, what is the probability that it is greater than delta times its expectation that is extremely small? So, we will just apply that to prove the estimator theorem.

(Refer Slide Time: 18:13)

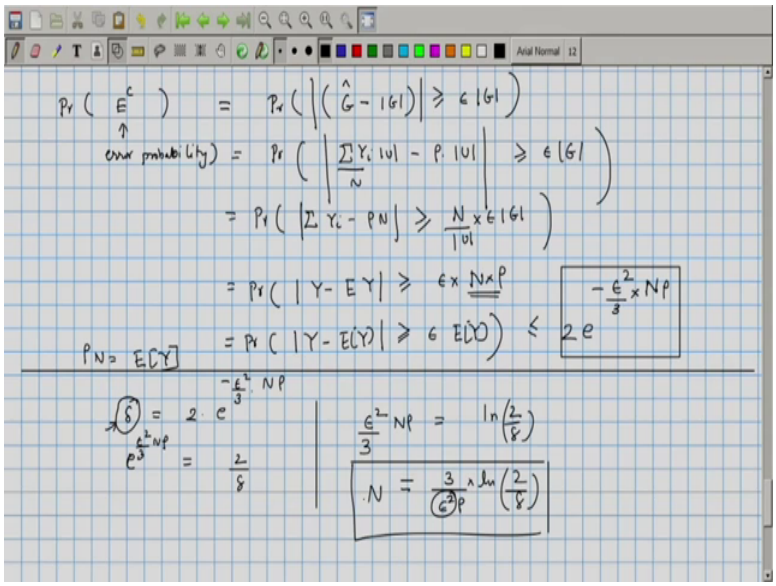


So, let us set up our random variables properly. So, a random variables are as follows Y i equals 1. If the i th sample belongs to G, this is equal to 0 otherwise and Y is equal to summation Y i which can be viewed as number of samples in G. We use Y to estimate G. So, the estimate for G, we will call it as G tilde that is equal to Y by N times size of U. We want to compute the probability that the estimate lies between 1 plus 1 minus epsilon times size of G and 1 plus epsilon times size of G.

Let us call this as event E. So, this probability we want it to be high. We could look at the complimentary event E complement that is probability that probability of E complement or the error probability. So, this is same as probability that G complement minus size of G is greater than epsilon times size of G and the absolute value is greater than epsilon time size of G.

Now, how do we compute this probability ok? So, this we can simply write it as probability that the estimate for G is summation Y I by N times U and the value of mod G is equal to rho times size of U. So this should be greater than or equal to epsilon times mod G. This is the event whose probability we need to estimate. So, we can write this as probability that summation Y I minus rho, its absolute value is greater than N divided by mod U times epsilon times size G. Ok just rearrange terms.

(Refer Slide Time: 21:13)



So, this is the probability that sum of random variables 0 1 random variables is greater than I mean, minus rho N its absolute value is greater than; you can write it as Y minus rho N. What is rho? Rho N is actually expectation of expectation of Y because each Y I, its expectation is rho and therefore, when you have N such sample that is just expectation of Y. So, Y minus expectation of Y greater than epsilon times N into G by U is again rho. So, this is again expectation of Y. So, this is probability that Y minus expectation of Y is greater than epsilon times expectation of Y.

Here, since Y is sum of random variables which takes 0 1 value, we can say, we can apply Chernoff bound and so therefore, this probability is less than 2 times E raised to minus epsilon square by 3 times mu. The mu being the mean of Y which is, N times rho ok. So, our error probability is definitely less than 2 times epsilon square by 3 times N rho. So, error probability if you think of it as delta and if we equate to this quantity ok, we will get delta is equal to 2 times E raised to minus epsilon square by 3 times N rho ok.

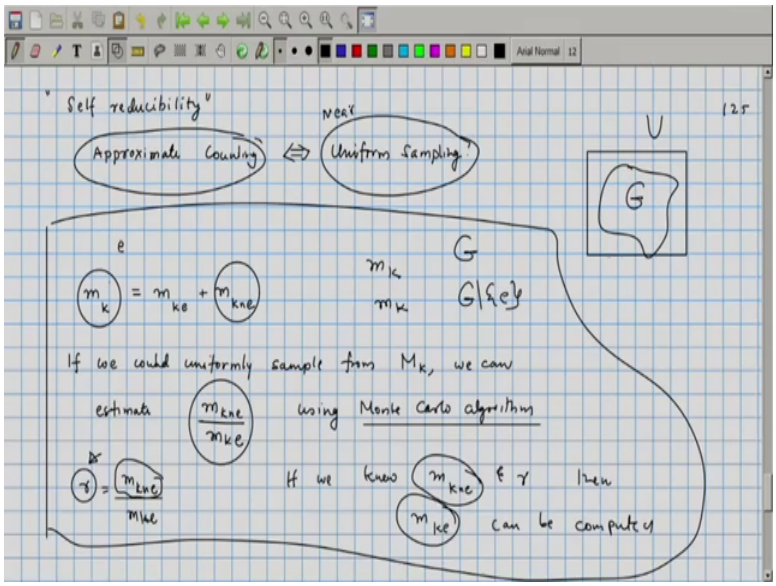
In other words, if we choose our N accordingly, I mean, whatever be the N this is the maximum amount of error that we can have. This is the amount of error we can have how large should N be? So, an N which satisfies this equation will be sufficient. So, we can write it as E raised to minus epsilon square by 3 N rho is equal to 2 by delta. Taking logs on both sides, we will get epsilon square by 3 times N rho is equal to log 2 by delta

or we can say, N is equal to 3 by epsilon square rho into log N by delta which is what our estimator theorem stated. So, as long as error is reasonable if it is let us say one by poly and the fraction of good cases or the size of G is bounded by a polynomial in the size of the total universe; this algorithm you can choose N to be polynomially bounded?

So, the estimator theorem works properly when rho is reasonably small. If it is extremely small, yes we cannot use the Monte Carlo algorithm. So, when it is very small let us say the size of S is 1 by 2 raise to N of the entire size, then we will have to run it to raise 2 raise to N times to get a reasonable success probability. So, those so we need to show that what whenever we are running Monte Carlo algorithm the set is of reasonable size. that is the first requirement ok. So now, how do we use this notion of Monte Carlo algorithm to generate or to count the number of perfect matchings?

Maybe, there is a universe which contains elements which are perfect matchings and let us say which are not so perfect matchings or other objects. If we could, and the universes size we could somehow estimate and then we could sample from that and generate, but this is not a I mean we do not have any algorithm of that kind. So, instead what we will use is an idea which can be used in a wide variety of case. It's called self reducibility.

(Refer Slide Time: 25:51)



So, self-reducibility is a key property that is required for running the kind of Monte Carlo algorithms will be thinking ok. So, what the self reducibility bring in? So whenever, so we have been defined what a self-reducibility, but whenever there is self-

reducibility the following statement is true. Approximate counting is equivalent to uniform sampling ok. So, once again whenever a problem is self-reducible, we haven't defined what is self-reducible, approximate counting which we had earlier done by a Monte Carlo algorithm can now be done by a uniform sampling ok?

In Monte Carlo algorithm, you require the good cases to be part of a large universe and the good cases is a significant fraction of the universe that we have, we are considering. Now, uniform sampling essentially means, we will generate only instances which are good instances ok, but we can do that uniformly. So, the earlier picture was we had this universe U and from that we were sampling elements G. We were sampling U uniformly at random. Now, if we could sample G uniformly at random, even when we do not know the size of G, this uniform sampling can be used to do approximate counting. In order to do this, a key feature required as what is known as self-reducibility. It's a natural notion.

So, let us understand it via let us say perfect matchings itself. So, imagine that we had access to a an oracle or an oracle or program which generates perfect matchings uniformly at random ok. So there were let us say, 125 perfect matchings for a particular graph. Each of those is being generated with 1 over 125 probability ok. Now how can we use that to count the size? The idea is the following, let us look at all the possible matchings and let us look at one particular matching edge E ok. Now, the matchings could either contain E or not contain e. So, if you look at the number of matchings of size K, this is equal to m k e plus m k n e. The number of matchings of size k essentially is a sum of matchings which contains the edge e and this doesn't contain the edge e.

If we could uniformly sample, we could estimate the ratio. So, this is the key claim. If we could uniformly sample from capital M k we can estimate m k n e divided by m k e. using our Monte Carlo algorithm. Ok use when we say using Monte Carlo algorithm, you cannot assume that these mean whatever we are approximating from whichever space we were sampling ok. So, we could sample from m k and let us say, we sample for let us say a thousand times and we look at the number of elements which have the edge e and the number of edges which doesn't have the edge e. Their ratio is essentially a good estimate of m k n e by m k e ok. But if so, this ratio could be estimated, but what is the advantage? If we estimate this ratio can be really compute m k ok.

So, m k n e by m k e this is a ratio that we have estimated. Now suppose, we knew m k n e we could clearly compute m k e as well ok. So, if we knew m k n e and r then m k e can be computed; just a multiplication alright? m k e is m k n e into 1 by R. So, this can be computed and once this is computed we can compute the sum which is m k. So, if we could compute this ratio and m k n e we are done, in a certain sense. We have to argue that the errors in the approximation do not add up to render our algorithm futile ok. That we will do later. So, m k n e is what we needed to estimate, but that is just another instance of approximate counting, it is a smaller instance.

M k n e is the number of perfect matchings without the edge e. So, that is a smaller graph. You need to count the number of perfect matchings in a smaller graph. We could again use the same idea to estimate that ok? So, we needed to estimate the number of perfect matchings in some particular or the matchings of size k in a certain graph. Now we need to find the matchings in a smaller graph; matchings of size k in a smaller graph ok.

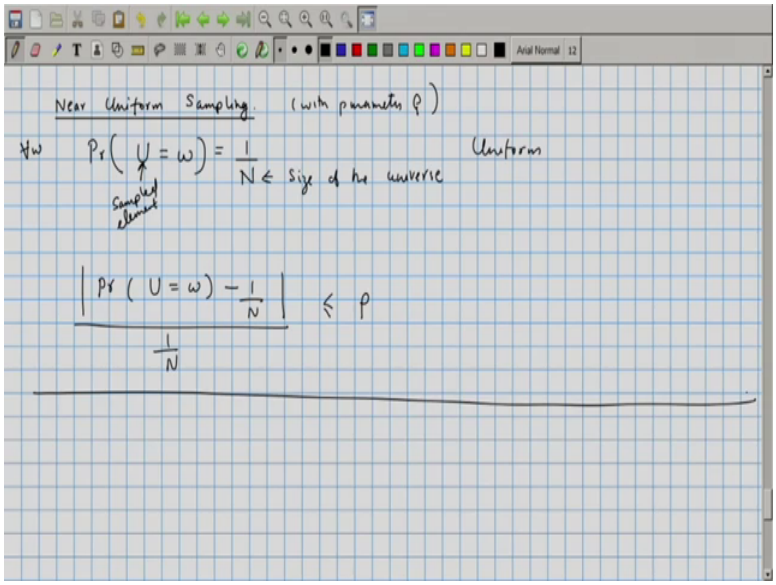
So for that again, we can use the same idea. So, this is called self-reducibility. So, when we could do self-reducibility. So, this is an intuitive proof of why self-reducibility would mean approximate counting because if we could self -educe; that means, reduce the problem to a smaller instance ok. So, we wanted to count M k now, I mean on some particular let us say graph G, now we have again m k on a smaller graph this graph minus the edge e.

Now, if you could count that, that along with this estimate the estimate we obtain by a uniform sampling so, that estimate and m k n e together yields the count m k ok. So, we had to count the number of perfect matchings. We said we will approximately count it. In order to approximately count we needed to sample from a universe which contains let us say, the perfect matchings and other elements, but instead of doing that sampling; now say that if we could just generate uniformly at random all the perfect matchings and that is good enough because of the self-reducibility of this particular problem that we are addressing.

So, there are lots of other problems which are self-reducible. In all those problems, we can use this method that is, I mean if we could uniformly sample then we could approximately count. The other direction is also I mean true if you could approximately

count then you can in some instances. You can do a near uniform sampling. So, we need to sample uniformly, but sampling uniformly is difficult.

(Refer Slide Time: 34:25)

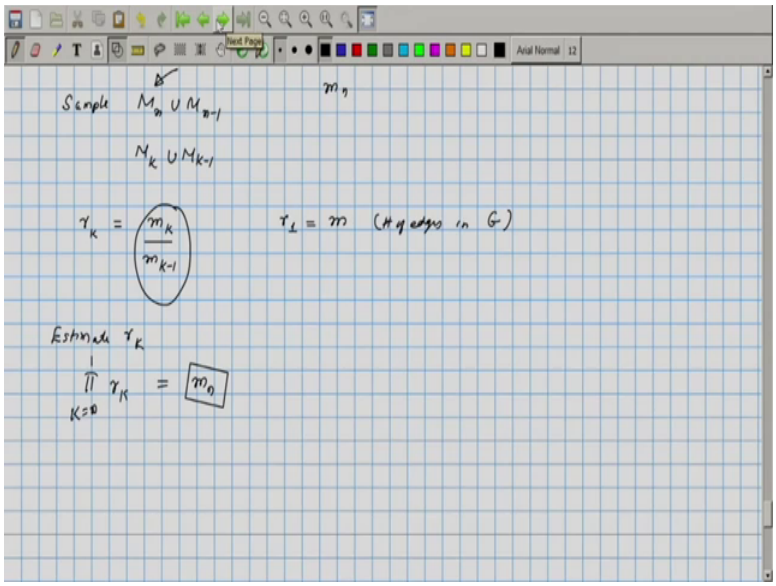


So, what we will do is we will do near uniform sampling and we will have to show that uniform sample, when we looked at our Monte Carlo algorithm, it required uniform sampling, but one can analyze this problem as well that is, if we could sample nearly uniformly. It's not exactly uniform, but very close to uniform. Even in those cases we can compute the approximate values fairly accurately ok. So, what is near uniform sample? So, let us say you are looking at a universe U and let us say the sampled element we denote it by let us say U itself.

If you look at the probability that the sampled element is equal to, let us say a fixed element omega ok. This probability we would expect it to be, let us say I mean 1 by N, where N is the size of the universe ok. So, in uniform sample what we expect is for every omega where omega is an element of the universe that we are sampling, so probability that the sampled element is omega is equal to 1 by N in case of uniform sample ok. When we say that something is near? So, this is uniform. Near uniform means, we are allowed to deviate from 1 by N. So, this minus 1 by N, its absolute value divided by 1 over N should be less than let us say rho. So, we will call that as near uniform sampling with parameter rho.

If rho is 0; that means, we have uniform sampling. So, what we will show later on is that near uniform sampling will also suffice ok. So now, approximately count the number of perfect matchings, all that we require is we need to do a near uniform sampling of the perfect matchings in the bipartite graph instead of sampling the perfect matchings.

(Refer Slide Time: 37:09)



So, what we will do is we will sample M n U M n minus 1. So, M N was a set from which we needed to sample. Now here sampling from now on, sampling would mean near uniform sampling. Instead we will do an almost uniform sampling or a near uniform sampling from M N U and M N minus 1 and we will see that our algorithm can be slightly modified; to compute the values of M N from these quantities ok.

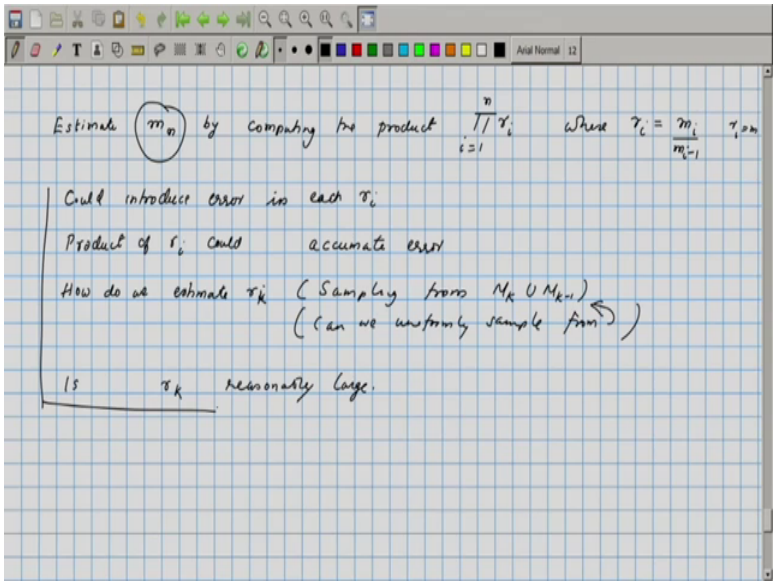
So, let us say that we will be doing near uniform sampling. Let us pretend that we are doing uniform sampling of M N U and M N minus 1 ok. So, when we are doing the sampling of M N U M N minus 1 or later on M k U M k minus 1 we can estimate the following quantity which is let us call this as r k. So, r k is equal to m k that is number of perfect matchings of size k by m k minus 1 because the entire universe from which we are sampling consists of two kinds of matchings.

Matchings of size k and matching the size k minus 1; if these matching formed significant portion of the entire thing, then we know that Monte Carlo algorithm would estimate m k by m k minus 1 with good accuracy ok. So, what we will do is, we will estimate r k. We will say that r 1 is basically equal to m or the number of edges because

anything could have been correct. I mean, when you take r 1; that means, the its the number of matchings of size 1 that is m we can readily estimate it; I mean, we do not have to find the ratio we will just take that as a value.

Now, if you compute r k the product over k varying from let say N to 1, this is equal to m n ok. So, in order to estimate m n, we need to estimate just r k. So, you can write the entire steps. Missed out many steps in between, but we will mean we have come not given complete proofs, but this is our outline.

(Refer Slide Time: 39:49)



Estimate m n by computing the product by i equals 1 to n r i where, r i is equal to m i divided by mi minus one and r 1 is equal to m. Number of edges. Now, were all could be go wrong? The first of all each of these r i's we are just estimating. So, there could be error; could introduce error in each r i ok. Second, each r i error might be small. Product of r i's could have accumulated, product of r i could accumulate error. And the individual r i's are not so much error prone, but maybe the product has ok.

Another issue is, how do we estimate r i? This requires sampling from m k union r k m k union m k minus 1 can be uniformly sampled from this. Even if we could do uniform sampling, is it guaranteed that r k, I mean, that m m k and m k minus 1 are significant portion of the entire collection is m k and m k minus 1. Or we can just simply ask, is r k a reasonably large? That is if one of the m i's, capital m i's is very small in comparison with the m i plus 1, then when you do the Monte Carlo algorithm, the number of

iterations that is required to guarantee the accuracy of r k could be reasonably small. So, is this reasonably large? These are some important questions.

So, we will see that each of these steps could be satisfactorily answered and therefore, we could estimate M N to reasonable accuracy by this method.