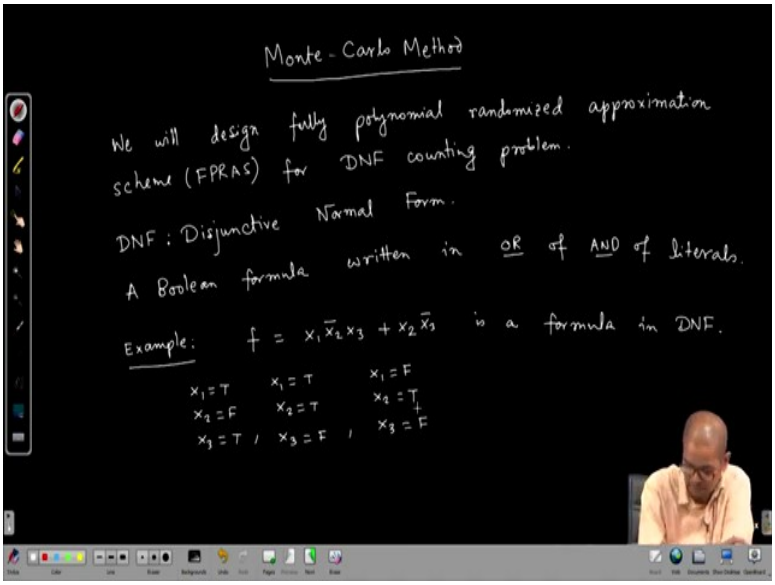
Selected Topics in Algorithm
Prof. Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

Lecture – 27
DNF Counting

Welcome. So, in the last class we started Monte-Carlo methods and explained that method on estimating the value of π. In today's class we will see more examples of Monte-Carlo methods for designing algorithms.

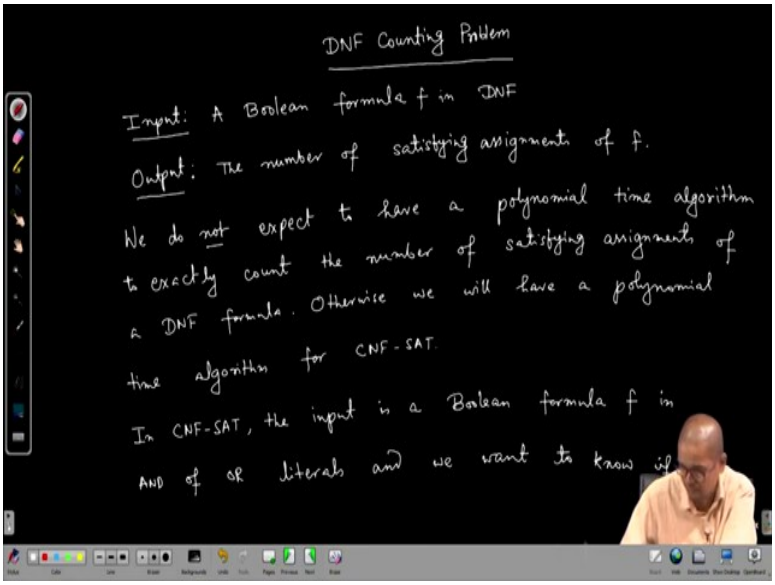
(Refer Slide Time: 00:45)



So, today's topic is Monte-Carlo method. So, using Monte-Carlo method, we will design what is called fixed fully polynomial randomized approximation scheme FPRAS for DNF counting problem. So, this is our second example. So, what is DNF counting problem? DNF stands for disjunctive normal form. So, what is disjunctive normal form? It is like a Boolean formula written in logical OR of AND of literals.

For example, so suppose $x_1 \wedge \bar{x}_2 \wedge x_3 \vee x_2$. So, this is an OR of AND, so this is a formula in DNF form disjunctive normal form. Now, what is the DNF counting problem?

(Refer Slide Time: 04:30)

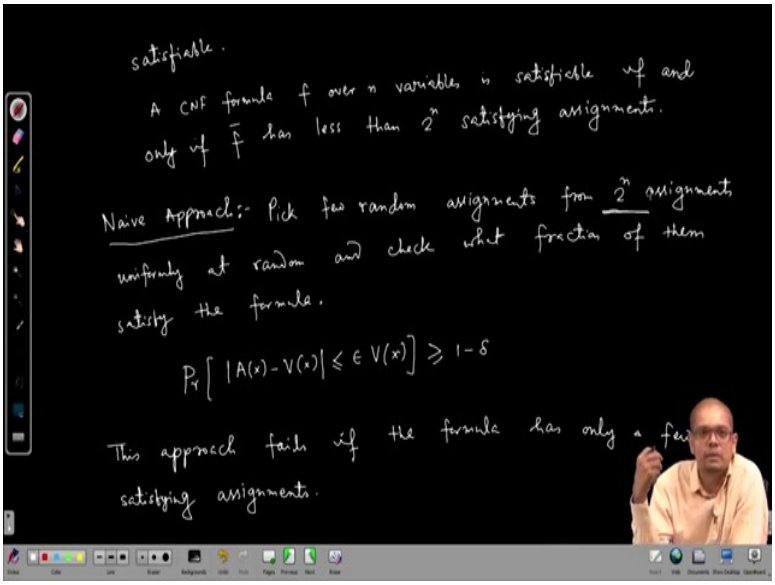


So, go in the next page, DNF counting problem. What is input? Input is Boolean formula in disjunctive normal form a Boolean formula f. Output is a number we want to count the number of satisfying assignments output of f. So, for example, in this formula if what are the satisfying assignments? That x_1 is true x_2 is false x_3 is true. It is one satisfying assignment.

Another setting showing assignment is x_1 is true, x_2 is true, x_3 is false. Another satisfying assignment is x_1 is false, x_2 is true, x_3 is false. So, this formula f has three satisfying assignments. Now, we will see later that you know we do not expect to have a polynomial term algorithm to output the exact count of number of solutions. We do not expect to have a polynomial time algorithm to exactly count the number of satisfying assignment of our DNF formula.

Why we do not expect? Otherwise, we will have a polynomial time algorithm for CNF-SAT. So, this problem we will discuss more in NP completeness but for the time being this CNF-SAT is a problem where the input is a Boolean formula in OR of AND format. So, in CNF-SAT the input is Boolean formula f in AND of OR clauses OR literals. And we want to know if f is satisfiable.

(Refer Slide Time: 10:25)



Now, for this we do not expect to have a polynomial time algorithm. And then how this problem is connected to DNF counting? So, CNF formula f over n variables is satisfiable if and only if f bar which is a DNA formula. If f is a CNF formula then $\bar{f}$ is a DNF formula $\bar{f}$ has 2^n satisfying assignments has less than 2^n satisfying assignment.

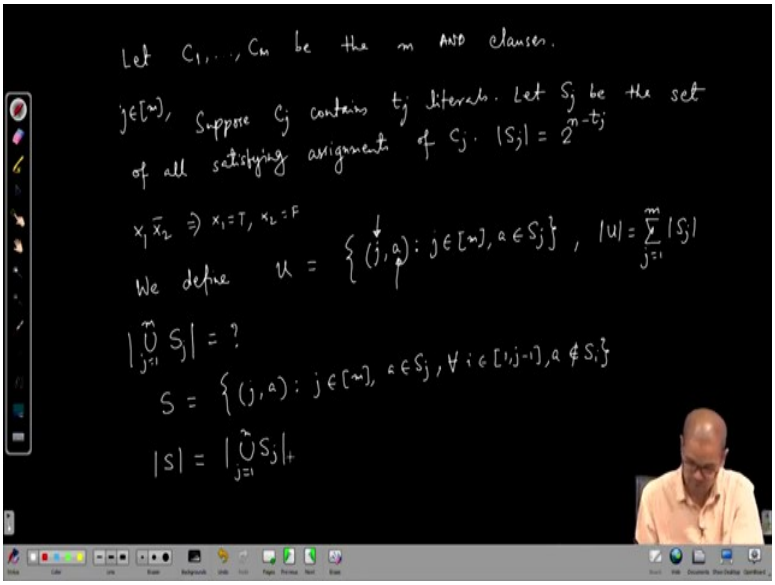
Hence, if we can exactly count the number of satisfying assignments of a DNF formula using that we can solve the CNF-SAT problem. For which you do not expect to have a polynomial time algorithm. So now, next, what we will do? Is that we will design a FPRAS for DNF counting problem, so, the naive approach. What is the naive approach? Naive approach is pick a random assignment from 2^n assignments.

Uniformly at random not pick few random assignments from 2^n assignments uniformly at random. And check what fraction of them satisfy the formula? Indeed, this was the approach for estimating the value of π. The problem with this approach is that this approach will fail if the number of satisfying assignments is too small, say one or two or something like that.

And if the number of assignments is say one then the algorithm is supposed to output correctly. Because if it is an FPRAS recall, the requirement is probability that $A(X)-V(X)$, $V(X)$ is the value of the solution, value of the problem, value of the instance. In this case, this is the number of solutions and $A(X)$ is the output of the algorithm. This is supposed to be within ϵ times value of $V(X)$ with probability at least $1-\delta$.

So, for small δ if $V(X)$ is very small, say 1 or 2. Then $A(X)$ essentially has to output correctly and this is where this approach fails. This approach let me write this approach, fails if the formula has only a few satisfying assignments.

(Refer Slide Time: 16:18)



So now, we see the second approach which will work. So, what we do is that? Let $C_1, \ldots, C_m$ be the m AND clauses recall that the formula is given in DNF form and it is a DNF means it is OR of AND clauses each $C_1, \ldots, C_m$ is AND of some literals. So, suppose for $j \in [m]$. Suppose C_j contains t_j literals. So, let S_j be the set of all satisfying assignments of C_j. Then if it contains t_j literals then on those t_j literals, we can fix exactly one way.

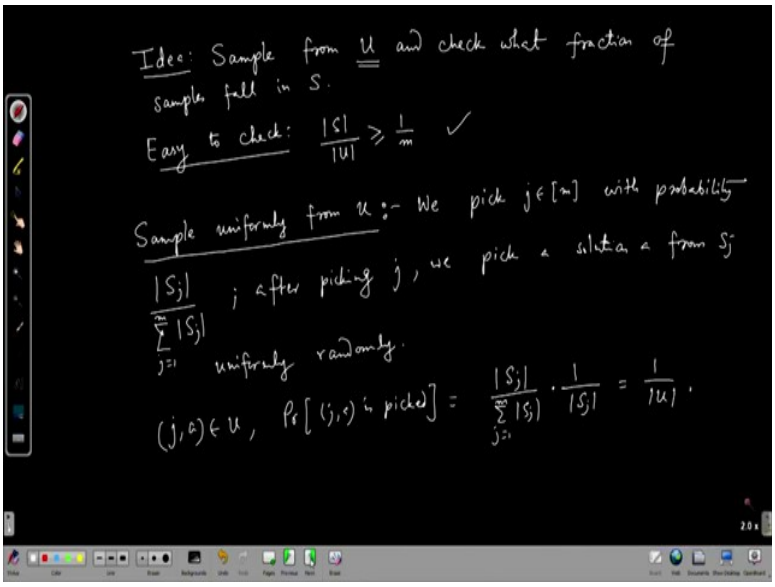
Those t_j literals can be fixed exactly one way to solve to satisfy C_j but other remaining literals can be set arbitrarily. It is like if there is a clause $x_1 \bar{x}_2$. And then to satisfy this clause x_1 must be set to true and x_2 must be set to false. But the remaining variables say $x_3, x_4, \ldots, x_n$ can be said arbitrarily. So, the number of satisfying assignments is very easy to compute this is to 2^{n-t_j}.

Now, we define so, for each 1 clause, the number of satisfying assignments is very, very easy to compute but they can have overlapped same as solution may satisfy many clauses. And that is where the difficulties we should not be doing over counting. So, to count that approximately we define some quantities, we define this set U to be (j, a), where $j \in [m]$ and a is a satisfying assignment.

So, same assignment a can satisfy many clauses. And they are coupled with that clause number and put it into U. So, size of U is sum of size of S_j. But what we want to count is? S which is union of S_j. So, we want to count this so, let us call union S_j. This is the goal. So, for that we define another set S which is in one to one correspondence with union S_j. This is (j, a) such that $j \in [m]$, $a \in S_j$ and for all i in 1 to j – 1, $a \notin S_i$.

That means an assignment a satisfies clause C_j but it does not satisfy any of the clauses $C_1, \ldots, C_{j-1}$. So now, it is now clear that S cardinality S is same as cardinality $S = \cup_{j=1}^{m} S_j$. So, our modified goal is to count the size of S. What is cardinality S?

(Refer Slide Time: 22:02)



So, for that first, we observe that we do not have the problem with that we had initially that the favourable outcome is too small than the sample space. This was the case, we were say in the naive approach we are sampling from a sample space of size 2 to the n but the favourable outcome can have a constant number of elements. So, what we do is? The idea is, idea sample from u and check what fraction of samples fall in S.

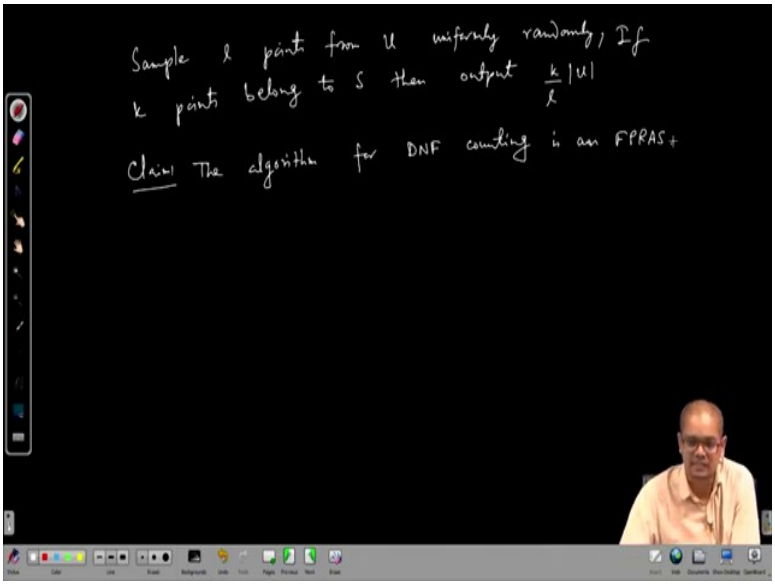
Now, for this approach to succeed and for this approach to do not fail like our naive approach, is that size of s should be considerable with respect to the size of U. But this is easy to check that size of S by size of U is greater than equal to 1 over m. Because size of S is same as union of size of S_j and size of union is at least $\frac{1}{m}$ of the size of the maximum set maximum S_j. So, this is this follows.

So now, we have hope and now the approach follows in the same path as estimating π. Only I need to say how we can sample from U sample uniformly from u? How do you do that? So, we pick first j in 1 to m uniformly at random, pick j with probability. Whichever j has many solutions to sample from uniformly random from u that j should have more probability.

So, the probability of picking j is should be proportional to size of S_j. So that means we pick j with probability size of S_j by summation size of S_j j= 1 to m. And after picking j we pick a solution a from S_j uniformly at random, how? Because S_j is the set of all satisfying assignments of a clause C_j and clause is the AND of some literals. So, those variables involved in the literals has to be set only one way.

And the other variables we set true or false with probability half. So, this way the probability, let us check that you know that some element (j,a) is picked. So, what is the probability? That (j,a) is picked is the probability that first j is picked which happens with this probability 1 to m. And once j is picked, it has size of S_j m solutions, so one of them is picked randomly this. This is and summation of size of S_j is U this is 1 over cardinality U.

(Refer Slide Time: 27:48)



So, again so, hence the algorithm means sample l points from U uniformly randomly. If k points belong to S then output that means $\frac{k}{l}$ fraction of U belongs to S. So then this output $\frac{k}{l}$

of size of U. And this is an FPRAS. So, for that so, we will show that claim. The algorithm for DNF counting is an FPRAS. So, this proof we will see in the next class. Thank you.