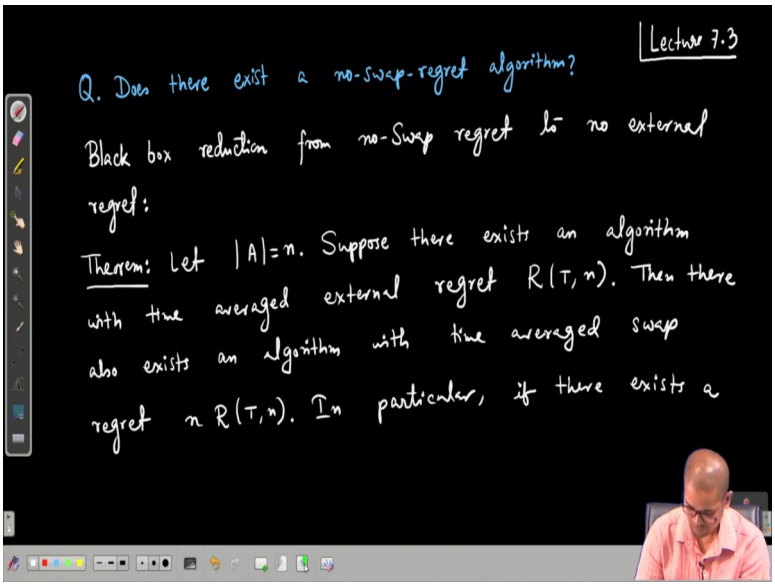
Algorithmic Game Theory
Prof. Palash Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 33
Swap Regret to External Regret Reduction

Welcome. So, in the last class we have studied no-swap-regret algorithm and we have seen that how the no-swap-regret algorithm allows, enables all the players to simultaneously converge to a correlated equilibrium. But we have left one important question untouched that whether there exist a no–swap-regret algorithm.

Although we have shown that there exist a no–external-regret algorithm namely multiplicative weight algorithm, but we have not shown existence of no–swap-regret algorithm and that is our point of discussion in today's class.

(Refer Slide Time: 01:09)



So, question. Does there exist a no-swap-regret algorithm? And we do not have to build up or design a no-swap-regret algorithm from scratch as we did for no–external-regret algorithm. What we will do is that we will leverage our knowledge about external no-external-regret algorithm.

We know that there exist an algorithm with no-external-regret namely multiplicative weight and we will use that algorithm in a Black box fashion that is a very beautiful

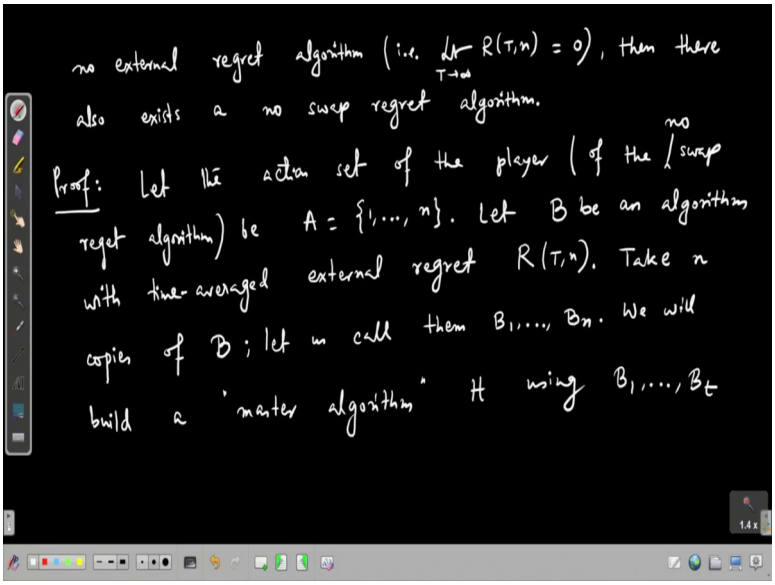
technique and this what is called Black box reduction from no-swap-regret to no-external-regret.

What do I mean by that? It means that if I want to design a no–swap-regret algorithm meaning that it is time average swap regret must go to 0 as time goes to ∞, then it is enough to design a no-external-regret algorithm meaning that an algorithm whose time averaged external regret goes to 0 as t goes to ∞.

That is if someone gives me a Black box or no-external-regret algorithm I can use that in a Black box fashion. I do not need to know it is functioning and using that I can design a no-swap-regret algorithm. So, the main theorem in today's lecture is this. So, let as usual I have n actions players have n actions.

Suppose there exist an algorithm with extra with time averaged external regret $R(T,n)$. Of course, $R(T,n)$ will go to 0 if the algorithm is a no-external-regret algorithm. So, suppose there exist an algorithm with time average external regret $R(T,n)$, then there also exists an algorithm with time averaged swap regret n times $R(T,n)$. In particular, if there exists a no-external-regret algorithm that is limit T tends to ∞ $R(T,n)$ is 0.

(Refer Slide Time: 06:25)



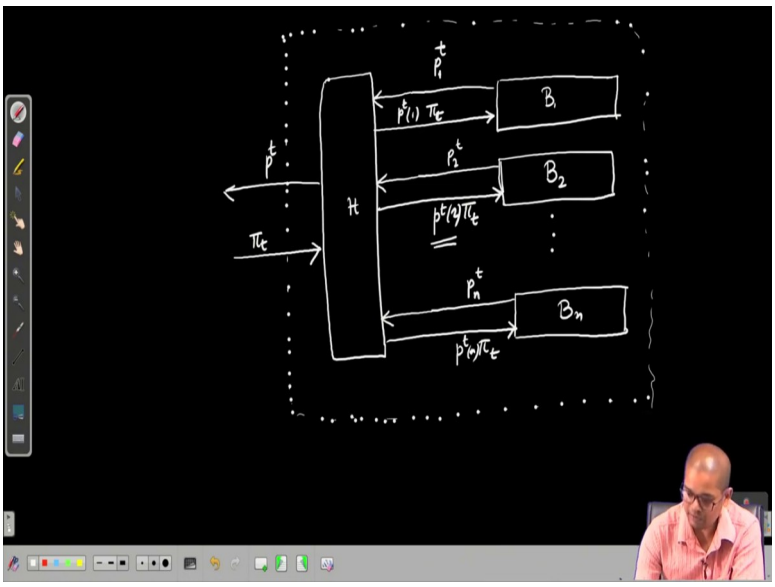
If there exist no external regret algorithm then there also exists a no swap regret algorithm. Proof: So, let the action set be let the action set of the player, player of the

external regret algorithm not external regret algorithm the swap regret algorithm. So, we will be designing a algorithm whose swap regret is n times $R(T,n)$.

So, we are designing a no-swap-regret algorithm of the swap regret algorithm. Action set of the player be say A equal to 1 to n. And let B be an algorithm with time averaged external regret $R(T,n)$. What we first do is that, we take n many copies of B. So, take n copies of B. What is n? The number of actions. So, each the idea is each copy of B will mimic one action of the player. Take n many copies of B, let us call it let us call them $B_1,\ldots,B_n$ ok.

Now, the idea is that we will build a master algorithm, let us call it H using $B_1,\ldots,B_n$. How does the algorithm look? Let us first understand it pictorially.

(Refer Slide Time: 11:08)



So, here the n copies of the algorithm B whose external regret is at most $R(T,n)$; $B_1,\ldots,B_n$ ok and we let them run independently. So, in every iteration B 1 will commit to a probability distribution P_1 in the t-th iteration t, B_2 will commit to a probability distribution P_2 ,..., B_n will commit to a probability distribution P_n.

And there is a master algorithm H which will consider all these probability distributions that each of these n copies of no-external-regret algorithm; no-external-regret algorithm

commits to and combining these somehow it computes some function of $P_1^t, P_2^t, \ldots, P_n^t$ and commits to a probability distribution P^t.

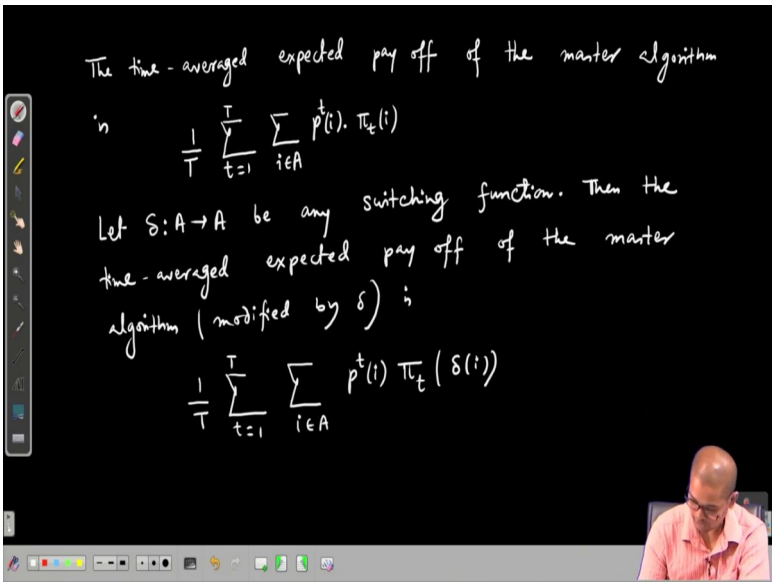
So, this is our algorithm box. So, it is interacting with the outside world which does not know about its inner functionality and it is supposed to commit to a probability distribution in each iteration P^t. And after committing this probability distribution that so, it picks a payoff function π_t which it receives and what it does is that to run $B_1, B_2, \ldots, B_n$ each of them they needs to be supplied a payoff function and it supplies the payoff function $B_1 \pi_t$, but it scales it down with p^t of 1.

Similarly, it supplies B_2 also π_t the same payoff function, but it scales it down to $p_t(2)$ and so on. So, what I have not discussed till now is that how I compute p^t from $P_1^t, P_2^t, \ldots, P_n^t$. Once I discuss this once I tell how p^t is computed then that finishes the description of the algorithm.

So, it takes n copies of no-external-regret algorithm and from their commuted probability distribution it somehow computes the probability distribution p^t which we will see now and then it receives a probability a payoff function π_t from the adversary from the from outside and it supplies those the same utility function π_t, but scales it down appropriately. So, it multiplies with P_1^t and supplies it to B_1 it multiplies with P_2^t and supplies to B_2 and so on.

So, in the rest of the rest of the proof we will see what or how p_t should be connected with $P_1^t, \ldots, P_n^t$ so that the swap regret is sort of minimum. So, let us see.

(Refer Slide Time: 15:39)

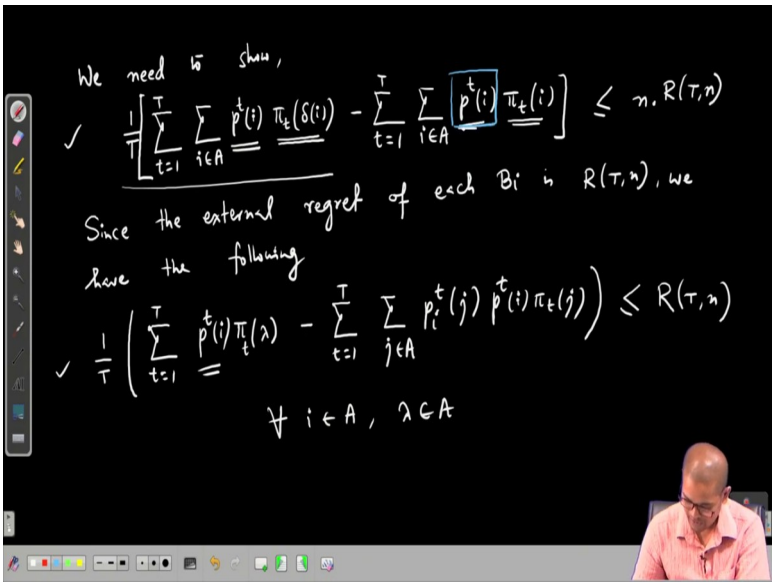


So, first what is the time averaged expected payoff? Time averaged expected payoff of the master algorithm is the transfer t capital T iterations and each iteration it commits to the probability distribution p^t; actions are 1 to n that is why let me index use i $p^t(i)\pi_t(i)$. This is its total utility to total payoff and I divide it with capital T to get time average expected payoff ok.

So, let us pick any switching function and see what would have been its payoff if we apply the switching function. So, let $\delta: A \to A$ be any switching function then the time averaged expected payoff of the master algorithm modified by δ; that means, what?

Instead of playing i, it plays $\delta(i)$. Then the time average expected payoff of the master algorithm modified by delta is same thing, but instead of playing i it should play $\delta(i)$ and thus it will receive a payoff of $\pi_t(\delta(i))p^t(i)$ this is the probability with respect to which action i is picked, but when action i is picked it is not playing action i it is playing $\delta(i)$ and so, the payoff received is $\pi_t(\delta(i))$ ok.

(Refer Slide Time: 19:10)



What is the goal? We need to show we need to show that the time averaged external time averaged swap regret 1 over T t equal to 1 to capital T $i \in A$ $p^t(i)\pi_t(\delta(i))$ minus summation t equal to 1 to capital T summation $i \in A$ $p^t(i)\pi_t(i)$. This is this should go to 0 as t goes to ∞. In particular we need to show for this particular theorem. This is less than equal to n times $R(T,n)$.

It is a good practice when you are proving some theoretical result to remind, constantly remind yourself that what we need to show, what is the goal, where we need to reach and where we are ok. And what do we have? We have that the external regret of each of the B_i's is at most $R(T,n)$ n. So, let us use that.

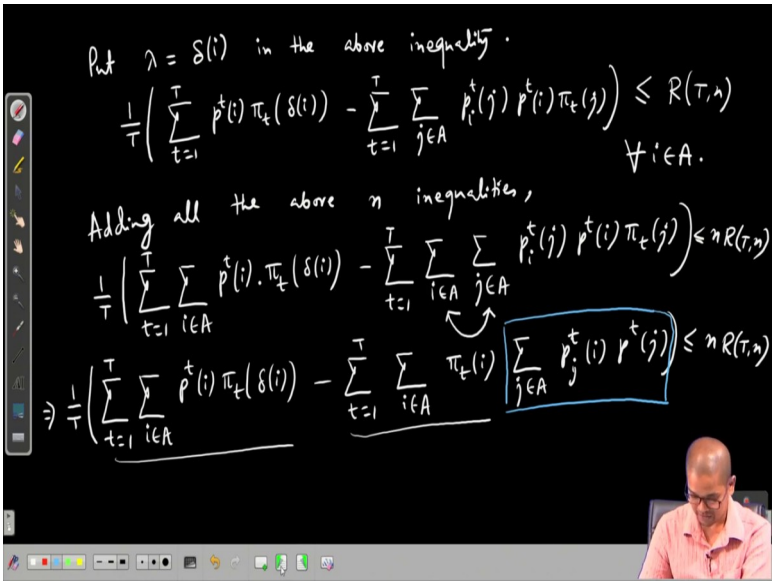
Since the since each since the external regret of each B_i is $R(T,n)$, we have the following. Recall external regret tries to perform as good as any fixed action. So, let us fix any action say λ and what will be its utility? If it keeps on playing λ. So, if it is keep on playing λ, so, its utility will be and what is the utility function? Utility function is $p^t(i)$. What is the utility function given to i th given to B_i? It is the $p^t(i)$, given to B_2 is $p_2^t \pi_t$.

So, given to B_i is $p^t(i)\pi$. So, $p^t(i)\pi(\lambda)$ because $\pi_t(\lambda)$ because it plays the fixed action lambda minus so, from B_i's perspective you see what is the committed probability distribution and what is it play. $j \in A$ the committed probability distribution is $p_i^t(j)$.

With this probability the i th player the i th box B_i plays the action j and the utility that it receives is $p^t(i)\pi_t(j)$ and the time averaged regret is this less than equal to $R(T,n)$ this should hold for all $i \in A$ and λ for i because for each player for each action i have a copy of the algorithm B. So, that is why $i \in A$ and for each $\lambda \in A$ because it is a no external regret algorithm, external regret is at most $R(T,n)$, good.

Now, what we do is that I somehow need to connect the equation on top with the equation on the bottom and you see that one easy way to connect is that here we have $p^t(i)$ here also you have $p^t(i)$, but the multiplied term is $\pi_t(\delta(i))$, here is $\pi_t(\lambda)$, but this the second inequality holds for all λ. So, what we do is that we put $\lambda = \delta(i)$.

(Refer Slide Time: 25:06)



So, put $\lambda = \delta(i)$ in the above inequality, then what do we get? 1 over T times $\sum_{t=1}^{T} p^t(i)\pi_t(\delta(i))$ minus t equal to 1 to capital T $j \in A$ those things the second term remains same $\pi_i^t(j)\, p^t(i)\pi_t(j)$ this is less than equal to $R(T,n)$.

Now this we have for all $i \in A$. Now for and A is 1 to n. So, we have n such inequalities and what we do is that we add all these n equalities n inequalities. Adding all the above n inequalities, we get what? $\frac{1}{T}\left(\sum_{t=1}^{T}\sum_{i\in A} p^t(i)\pi_t(\delta(i)) - \sum_{t=1}^{T}\sum_{i\in A}\sum_{j\in A} p_i^t(j)p^t(i)\pi_t(j)\right) \leq nR(T,n)$, good.

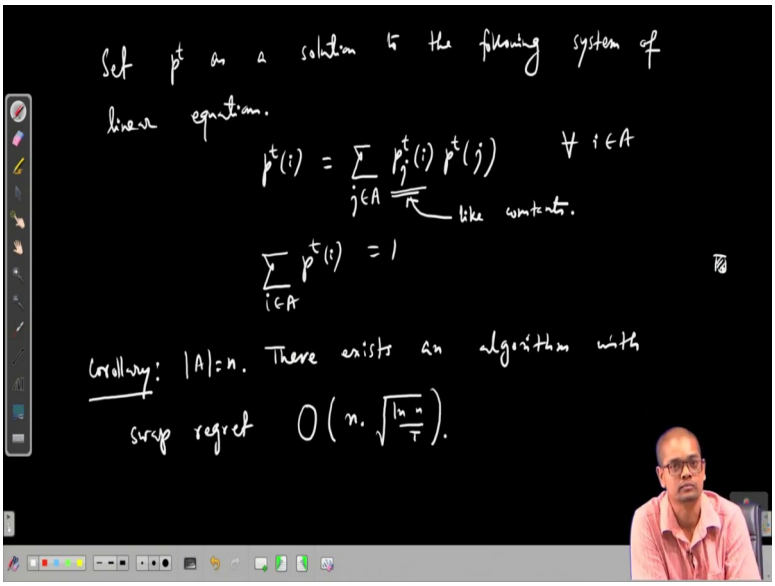
Now, whenever we have double sum it is always it is often good idea to exchange the double sum. So, what we do is that we exchange these two double sums and let us see what we get. We get something interesting. The first term remains as it is minus summation t equal to 1 to capital T ok and what we do is that we basically exchange the role of i and j. So, what we are saying i before we are saying j and j before we are saying i because both are on the same over the same indexing set.

So, exchanging the role of i and j $i \in A$. So, whenever we had I will write j and instead of j I will write i. So, $\pi_t(i)$ goes out and this remains inside $p_j^t(i)p^t(j)$ this is less than equal to $nR(T,n)$ ok. Now you see that what was our goal. Our goal is that so, let us recall what was our goal here.

So, the first term here exactly matches the first term here and the second term till here it matches, but now this part now here you have summation. So, p^t here we have some other color, here we have $p^t(i)$ and there we have this. And this tells us how should we pick we set the p^t probability distribution p^t that the master algorithm commits.

So, let us recall that was the thing that was left to be discussed that how master algorithm H computes p^t from $p_1^t, p_2^t \cdots$ and this is how. It solves. What is the requirement? Requirement is p_i^t should be equal to this sum summation j over A $p_j^t(i)p^t(j)$ for all I this should hold.

(Refer Slide Time: 31:17)



So, we set p^t as a solution to the following system of linear equations $p^t(i)$ is sum over $j \in A$ $p^t_j(i)\, p^t(j)$ this for all $i \in A$ and we have p^t must be a probability distribution. So, $p^t(i)$ $i \in A$ is 1. Now from linear algebra we need linear algebra background to know that this system of linear equation has a unique solution and so, p^t is uniquely defined in terms of this p^t_j's and so, p^t_j's are like constants because when we are computing p^t p^t_j s are given by the $B_1, \ldots, B_n$. So, these are like constants.

So, that makes this a system of linear equations and it follows from linear algebra standard linear algebra that this system of linear equations has a unique solution. So, that concludes the proof and which basically shows like a let me write a corollary because we know that there is a algorithm with external regret $O\left(\sqrt{\frac{\log n}{t}}\right)$. So, let A equal to n then there exist an algorithm with swap regret $O\left(n\sqrt{\frac{\log n}{T}}\right)$. So, we will conclude here today.

Thank you.