CS364A: Algorithmic Game Theory
Lecture #17: No-Regret Dynamics*

Tim Roughgarden†

November 18, 2013

This lecture continues to study the questions introduced last time. Do strategic players reach an equilibrium of a game? How quickly? By what learning processes? Positive results on these questions justify equilibrium analysis, including bounds on the price of anarchy.

Last lecture focused on best-response dynamics. These dynamics are most relevant for potential games, which cover many but not all interesting applications. This lecture, we study a second fundamental class of learning dynamics — *no-regret dynamics*. An attractive feature of these dynamics is their rapid convergence to an approximate equilibrium — a coarse correlated equilibrium (Lecture 13), not generally a Nash equilibrium — in arbitrary games.

1 External Regret

1.1 The Model

Most of the this lecture studies the *regret-minimization problem*, which concerns a single decision-maker playing a game against an adversary. Section 3 connects this single-player theory to multi-player games and their coarse correlated equilibria.

Consider a set A of $n \geq 2$ actions. The setup is as follows.

- At time $t = 1, 2, \ldots, T$:
  - A decision-maker picks a mixed strategy p^t — that is, a probability distribution — over its actions A.
  - An adversary picks a cost vector $c^t : A \to [0, 1]$.[1]

†Department of Computer Science, Stanford University, 462 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

[1]The important assumption is that costs are bounded. See the Exercises for extensions of today's results to negative costs (i.e., payoffs) and to costs in $[0, c_{\max}]$ instead of in $[0, 1]$.

– An action a^t is chosen according to the distribution p^t, and the decision-maker incurs cost $c^t(a^t)$. The decision-maker learns the entire cost vector c^t, not just the realized cost $c^t(a^t)$.[2]

For example, A could represent different investment strategies, or different driving routes between home and work. When we return to multi-player games (Section 3), the action set will be the strategy set of a single player, and the cost vector will be induced by the strategies chosen by all of the other players.

1.2 Lower Bounds

We seek a "good" algorithm for online decision-making problems of the above type. But the setup above seems a bit unfair, no? The adversary is allowed to choose a cost function *after* the decision-maker has committed to a mixed strategy. This asymmetry motivates asking what kind of guarantee we could possibly hope for in such a model. We next consider three examples that show limitations on what is possible.

Example 1.1 (Impossibility w.r.t. the Best Action Sequence) There is no hope of comparing the cost of an online decision-making algorithm to the cost of the best action sequence in hindsight — the latter quantity $\sum_{t=1}^{T} \min_{a \in A} c^t(a)$ is simply too strong a benchmark.

For instance, suppose $A = 2$ and fix an arbitrary online decision-making algorithm. Each day t, the adversary chooses the cost vector c^t as follows: if the algorithm plays the first strategy with probability at least $\frac{1}{2}$ then c^t is $(1 \;\; 0)$; otherwise the cost vector is $(0 \;\; 1)$. The adversary has forced the expected cost of the algorithm to be at least $\frac{T}{2}$ while ensuring that the cost of the best action sequence in hindsight is 0.

Example 1.1 motivates the following important definitions. Rather than comparing the expected cost of an algorithm to that of the best action *sequence* in hindsight, we compare it to the cost incurred by the best *fixed action* in hindsight. That is, our benchmark will be $\min_{a \in A} \sum_{t=1}^{T} c^t(a)$ rather than $\sum_{t=1}^{T} \min_{a \in A} c^t(a)$.

Definition 1.2 The *(time-averaged) regret* of the action sequence $a^1, \ldots . a^T$ with respect to the action a is

$$\frac{1}{T}\left[\sum_{t=1}^{T} c^t(a^t) - \sum_{i=1}^{T} c^t(a)\right]. \tag{1}$$

In this lecture, "regret" will always refer to Definition 1.2. Next lecture we discuss another notion of regret.

Definition 1.3 (No-Regret Algorithm) Let $\mathcal{A}$ be an online decision-making algorithm.

[2]The *bandit model*, where the decision-maker only learns the realized cost, has also been studied extensively (e.g. [2]). The guarantees presented in this lecture carry over, with somewhat worse bounds and more complex algorithms, to the bandit model as well.

(a) An *adversary for* $\mathcal{A}$ is a function that takes as input the day t, the mixed strategies $p^1, \ldots, p^t$ produced by $\mathcal{A}$ on the first t days, and the realized actions $a^1, \ldots, a^{t-1}$ of the first $t-1$ days, and produces as output a cost vector $c^t : [0,1] \to A$.

(b) An online decision-making algorithm has *no (external) regret* if for every adversary for it, the expected regret (1) with respect to every action $a \in A$ is $o(1)$ as $T \to \infty$.

Remark 1.4 (Combining Expert Advice) The problem of designing a no-regret algorithm is sometimes called "combining expert advice" — if we think of each action an "expert" that makes recommendations, then a no-regret algorithms performs asymptotically as well as the best expert.

Remark 1.5 (Adaptive vs. Oblivious Adversaries) The adversary in Definition 1.3 is sometimes called an *adaptive adversary*. An *oblivious adversary* is the special case in which the cost vector c^t depends only on t (and on the algorithm $\mathcal{A}$).

For this lecture, we'll adopt the no-regret guarantee of Definition 1.3 as the "holy grail" in the design of online decision-making algorithms. The first reason is that, as we'll see in Section 2, this goal can be achieved by simple and natural learning algorithms. The second reason is that the goal is non-trivial: as the examples below make clear, some ingenuity is required to achieve it. The third reason is that, when we pass to multi-player games in Section 3, no-regret guarantees will translate directly to coarse correlated equilibrium conditions.

Remark 1.6 In regret-minimization, one usually thinks of the number n of actions as fixed as the time horizon T tends to infinity. In a no-regret algorithm, the (time-averaged) regret can be a function of n but tends to 0 as the time horizon grows.

The next example rules out deterministic no-regret algorithms.

Example 1.7 (Randomization Is Necessary for No Regret) A simple consequence of the asymmetry between the decision-maker and the adversary is that there does not exist a no-regret deterministic algorithm. To see this, suppose there are $n \geq 2$ actions and fix a deterministic algorithm. At each time step t, the algorithm commits to a single action a^t. The obvious strategy for the adversary is to set the cost of action a^t to be 1, and the cost of every other action to be 0. Then, the cost of the algorithm is T while the cost of the best action in hindsight is at most $\frac{T}{n}$. Thus, even when there are only 2 actions, the algorithm has constant regret (as $T \to \infty$) with respect to some action a.

The next example does not rule out (randomized) no-regret algorithms, though it does limit the rate at which regret can vanish as the time horizon T grows.

Example 1.8 ($\Omega(\sqrt{(\ln n)/T})$ Regret Lower Bound) The next example shows that, even with only $n = 2$ actions, no (randomized) algorithm has expected regret vanishing faster than the rate $\Theta(1/\sqrt{T})$. A similar argument shows that, with n actions, expected regret cannot vanish faster than $\Theta(\sqrt{(\ln n)/T})$.

Consider an adversary that, independently on each day T, chooses uniformly at random between the cost vectors (1 0) and (0 1). No matter how smart or dumb an online decision-making algorithm is, its cumulative expected cost is exactly $\frac{T}{2}$. In hindsight, however, with constant probability one of the two fixed actions has cumulative cost only $\frac{T}{2} - \Theta(\sqrt{T})$. The reason is that if you flip T fair coins, while the expected number of heads is $\frac{T}{2}$, the standard deviation is $\Theta(\sqrt{T})$. This implies that there is a distribution over 2^T (oblivious) adversaries such that every algorithm has expected regret $\Omega(1/\sqrt{T})$, where the expectation is over the algorithm's coin flips and the choice of the adversary. It follows that for every algorithm, there exists an (oblivious) adversary for which the algorithm has expected regret $\Omega(1/\sqrt{T})$.

2 The Multiplicative Weights Algorithm

2.1 No-Regret Algorithms Exist

The most important result in this lecture is that *no-regret algorithms exist*. Next lecture we'll see that this fact alone has some amazing consequences. Even better, there are simple and natural such algorithms — while not a literal description of human player behavior, the guiding principles of the following algorithm are recognizable from the way many people learn and make decisions. Finally, the algorithm we discuss next has optimal regret, matching the lower bound provided by Example 1.8.

Theorem 2.1 ([3, 4], etc.) *There exist simple no-regret algorithms with expected regret $O(\sqrt{(\ln n)/T})$ with respect to every fixed action.*

An immediately corollary is that a logarithmic (in n) number of iterations suffice to drive the expected regret down to a small constant.

Corollary 2.2 *There exists an online decision-making algorithm that, for every $\epsilon > 0$, has expected regret at most ϵ with respect to every fixed action after $O((\ln n)/\epsilon^2)$ iterations.*

2.2 The Algorithm

We next discuss the *multiplicative weights (MW)* algorithm, which is also sometimes called *Randomized Weighted Majority* or *Hedge*. This algorithm has numerous applications beyond online decision-making [1]. Its design follows two guiding principles.

1. Past performance of actions should guide which action is chosen now. Since the action choice must be randomized (Example 1.7), the probability of choosing an action should be increasing in its past performance (i.e., decreasing in its cumulative cost).

2. Many instantiations of the above idea yield no-regret algorithms. For optimal regret bounds, however, it is important to aggressively punish bad actions — when a previously good action turns sour, the probability with which it is played should decrease at an exponential rate.

Here is a formal description of the MW algorithm. It maintains a weight, intuitively a "credibility," for each action. At each time step the algorithm chooses an action with probability proportional to its current weight. The weight of each action can only decrease, and the rate of decrease depends on the cost of the action.

1. Initialize $w^1(a) = 1$ for every $a \in A$.
2. For $t = 1, 2, \ldots, T$:
   (a) Play an action according to the distribution $p^t := w^t/\Gamma^t$, where $\Gamma^t = \sum_{a \in A} w^t(a)$ is the sum of the weights.
   (b) Given the cost vector c^t, decrease weights using the formula $w^{t+1}(a) = w^t(a) \cdot (1 - \epsilon)^{c^t(a)}$ for every action $a \in A$.[3]

For example, if all costs are either 0 or 1, then the weight of each action either stays the same (if $c^t(a) = 0$) or gets multiplied by $(1-\epsilon)$ (if $c^t(a) = 1$). We'll choose an exact value for ϵ later; it will be between 0 and $\frac{1}{2}$. For intuition, note that as ϵ grows small, the distribution p^t tends to the uniform distribution. Thus small values of ϵ encourage exploration. As ϵ tends to 1, the distribution p^t tends to the distribution that places all its mass on the action that has incurred the smallest cumulative cost so far. Thus large values of ϵ encourage "exploitation," and ϵ is a knob for interpolating between these two extremes. The MW algorithm is obviously simple to implement — the only requirement is to maintain a weight for each action.

2.3 The Analysis

This section proves the bound in Theorem 2.1 for the MW algorithm. We can restrict attention to oblivious adversaries (as defined in Remark 1.5) that fix a sequence of cost vectors $c^1, \ldots, c^T$ up front. The intuitive reason is that the behavior of the MW algorithm is independent of the realized actions $a^1, \ldots, a^t$: the distribution p^t chosen by the algorithm is a deterministic function of $c^1, \ldots, c^{t-1}$. Thus, there is no reason for a worst-case adversary for the MW algorithm to condition its cost vectors on previous realized actions. Similarly, a worst-case adversary does not need to explicitly condition on the distributions $p^1, \ldots, p^t$, since these are uniquely determined by the adversary's previous cost vectors $c^1, \ldots, c^{t-1}$.[4]

Fix an arbitrary sequence $c^1, \ldots, c^T$ of cost vectors. Recall $\Gamma^t = \sum_{a \in A} w^t(a)$ denotes the sum of the actions' weights at time t. The weight of every action (and hence Γ^t) can only decrease with t. The plan for the proof is to relate the two quantities we care about, the expected cost of the MW algorithm and the cost of the best fixed action, to Γ^T. The bound will then follow from some simple algebra and approximations.

[3] Other update steps, like $w^{t+1}(a) = w^t(a) \cdot (1 - \epsilon c^t(a))$, also work; see the Exercises.

[4] A bit more formally, one can solve for the worst adaptive adversary for the MW algorithm using backward induction, and the result is an oblivious adversary.

The first step is to show that if there is a good fixed action, then the weight of this action single-handedly shows that the final value Γ^T is pretty big. Formally, define $OPT := \sum_{t=1}^{T} c^t(a^*)$ as the cumulative cost for the best fixed action a^*. Then,

$$\begin{aligned}\Gamma^T &\geq w^T(a^*) \\ &= \underbrace{w^1(a^*)}_{=1} \prod_{t=1}^{T} (1-\epsilon)^{c^t(a^*)} \\ &= (1-\epsilon)^{OPT}.\end{aligned}$$

This connects our intermediate quantity Γ^T with one of the two quantities that we care about, namely OPT.

The second step, and the step which is special to the MW algorithm and its close cousins, is that the sum of weights Γ^t decreases exponentially fast with the expected cost of the MW algorithm. This implies that the algorithm can only incur large cost if all fixed actions are bad.

Precisely, the expected cost of the MW algorithm at time t is

$$\sum_{a \in A} p^t(a) \cdot c^t(a) = \sum_{a \in A} \frac{w^t(a)}{\Gamma^t} \cdot c^t(a). \tag{2}$$

Next, to understand Γ^{t+1} as a function of Γ^t and the expected cost (2), we write

$$\begin{aligned}\Gamma^{t+1} &= \sum_{a \in A} w^{t+1}(a) \\ &= \sum_{a \in A} w^t(a) \cdot (1-\epsilon)^{c^t(a)} \\ &\leq \sum_{a \in A} w^t(a) \cdot (1 - \epsilon c^t(a)) \qquad (3) \\ &= \Gamma^t(1 - \epsilon\nu^t),\end{aligned}$$

where (3) follows from the fact that $(1-\epsilon)^x \leq 1 - \epsilon x$ for $\epsilon \in [0, \frac{1}{2}]$ and $x \in [0,1]$ (see the Exercises), and ν^t denotes the expected cost (2) of the MW algorithm at time t. As promised, the expected algorithm cost (and ϵ) govern the rate at which the total weight Γ^t decreases.

Our goal is to upper bound the cumulative expected cost $\sum_{i=1}^{t} \nu^t$ of the MW algorithm in terms of OPT. We've related these quantities through Γ^T:

$$(1-\epsilon)^{OPT} \leq \Gamma^T \leq \underbrace{\Gamma^1}_{=n} \prod_{t=1}^{T} (1 - \epsilon\nu^t)$$

and hence

$$OPT \cdot \ln(1-\epsilon) \leq \ln n + \sum_{t=1}^{T} \ln(1 - \epsilon\nu^t).$$

We want to extract the ν^t's from the clutches of the logarithm, so it makes sense to recall the Taylor expansion

$$\ln(1-x) = -x - \tfrac{x^2}{2} - \tfrac{x^3}{3} - \cdots .$$

By throwing away all (negative) terms but the first, we can use this expansion to upper bound $\ln(1-\epsilon\nu^t)$ by $-\epsilon\nu^t$. While we're at it, we may as well lower bound $\ln(1-\epsilon)$ by throwing out all terms but the first two, and doubling the second term to compensate (assuming here that $\epsilon \leq \frac{1}{2}$), yielding $-\epsilon - \epsilon^2$. Summarizing, for $\epsilon \in (0, \frac{1}{2}]$ we have

$$OPT \cdot \left[-\epsilon - \epsilon^2\right] \leq \ln n + \sum_{t=1}^{T} (-\epsilon\nu^t)$$

and hence

$$\sum_{t=1}^{T} \nu^t \leq OPT \cdot (1+\epsilon) + \frac{\ln n}{\epsilon} \leq OPT + \epsilon T + \frac{\ln n}{\epsilon}, \tag{4}$$

where in the second inequality we use the very weak upper bound that, since costs are at most 1, OPT is at most T.

We now set the free parameter ϵ in the MW algorithm to equalize the two error terms in (4) — that is, to $\sqrt{\ln n/T}$. Then, the cumulative expected cost of the MW algorithm is at most $2\sqrt{T \ln n}$ more than the cumulative cost of the best fixed action; dividing both sides by T shows that (per-time-step) regret is at most $2\sqrt{\ln n/T}$. This completes the proof of Theorem 2.1.

Remark 2.3 (When T Is Unknown) In setting the parameter ϵ, we assumed that the time horizon T is known a priori. When this is not the case, the algorithm can be modified as follows: at day t, use the value $\epsilon = \sqrt{\ln n/\hat{T}}$, where $\hat{T}$ is the smallest power of 2 larger than t. The regret guarantee of Theorem 2.1 carries over to this algorithm (see the Exercises).

Recall that Example 1.8 shows that Theorem 2.1 is optimal up the constant in the additive term. Corollary 2.2 is also worth remembering — only $\frac{4 \ln n}{\epsilon^2}$ iterations of the MW algorithm are necessary to achieve expected regret at most ϵ.

3 No-Regret Dynamics

We now pass from single-player to multi-player settings. We use the language of cost-minimization games (Lecture 13); there is an obvious analog for payoff-maximization games. In each time step $t = 1, 2, \ldots, T$ of *no-regret dynamics*:

1. Each player i simultaneously and independently chooses a mixed strategy p_i^t using a no-regret algorithm.

2. Each player i receives a cost vector c_i^t, where $c_i^t(s_i)$ is the expected cost of strategy s_i when the other players play their chosen mixed strategies. That is, $c_i^t(s_i) = \mathbf{E}_{\mathbf{s}_{-i} \sim \sigma_{-i}}[C_i(s_i, \mathbf{s}_{-i})]$, where $\sigma_{-i} = \prod_{j \neq i} \sigma_j$.

No-regret dynamics are well defined because no-regret algorithms exist (Theorem 2.1). Each player can use whatever no-regret algorithm it wants. Players move simultaneously, unlike in best-response dynamics, but the following results also extend to players that move sequentially (see the Exercises).

No-regret dynamics can be implemented very efficiently. If each player uses the MW algorithm, for example, then in each iteration each player does a simple update of one weight per strategy, and only $O(\frac{\ln n}{\epsilon^2})$ iterations of this are required before every player has expected regret at most ϵ for every strategy. (Here n is the maximum size of a player's strategy set.) The next result is simple but important: the time-averaged history of joint play under no-regret dynamics converges to the set of coarse correlated equilibrium — the biggest set in our hierarchy of equilibria (Lecture 13). This forges a fundamental connection between a static equilibrium concept and outcomes generated by natural learning dynamics.

Proposition 3.1 *Suppose after T iterations of no-regret dynamics, every player of a cost-minimization game has regret at most ϵ for each of its strategies. Let $\sigma^t = \prod_{i=1}^k p_i^t$ denote the outcome distribution at time t and $\sigma = \frac{1}{T}\sum_{t=1}^T \sigma^t$ the time-averaged history of these distributions. Then σ is an ϵ-approximate coarse correlated equilibrium, in the sense that*

$$\mathbf{E}_{\mathbf{s}\sim\sigma}[C_i(\mathbf{s})] \leq \mathbf{E}_{\mathbf{s}\sim\sigma}[C_i(s_i', \mathbf{s}_{-i})] + \epsilon$$

for every player i and unilateral deviation s_i'.

Proof: By definition, for every player i,

$$\mathbf{E}_{\mathbf{s}\sim\sigma}[C_i(\mathbf{s})] = \frac{1}{T}\sum_{t=1}^T \mathbf{E}_{\mathbf{s}\sim\sigma^t}[C_i(\mathbf{s})] \tag{5}$$

and

$$\mathbf{E}_{\mathbf{s}\sim\sigma}[C_i(s_i', \mathbf{s}_{-i})] = \frac{1}{T}\sum_{t=1}^T \mathbf{E}_{\mathbf{s}\sim\sigma^t}[C_i(s_i', \mathbf{s}_{-i})]\,. \tag{6}$$

The right-hand sides of (5) and (6) are the time-averaged expected costs of player i when playing according to its no-regret algorithm and when playing the fixed action s_i' every day, respectively. Since every player has regret at most ϵ, the former is at most ϵ more than the latter. This verifies the approximate coarse correlated equilibrium condition. ■

Remark 3.2 For Proposition 3.1, it is important that the decision-making algorithms used by the players have no regret with respect to adaptive adversaries (Remark 1.5). The mixed strategy chosen by player i at time t affects the cost vectors c_j^t received by the other players $j \neq i$ at time t, hence it affects their chosen strategies at future time steps, and hence it affects the cost vectors received by player i at future time steps. That is, when other players are themselves using adaptive learning algorithms to choose strategies, they correspond to an adaptive adversary.

In Lecture 14 we proved that POA bounds for smooth games — including all of the examples we've studied in this course — automatically extend to coarse correlated equilibria. With an approximate equilibrium, the POA bounds remain approximately correct.

Corollary 3.3 *Suppose after T iterations of no-regret dynamics, player i has expected regret at most R_i for each of its actions. Then the time-averaged expected objective function value $\frac{1}{T}\mathbf{E}_{\mathbf{s}\sim\sigma^i}[cost(\mathbf{s})]$ is at most*

$$\frac{\lambda}{1-\mu}cost(\mathbf{s}^*) + \frac{\sum_{i=1}^k R_i}{1-\mu}.$$

In particular, as $T \to \infty$, $\sum_{i=1}^k R_i \to 0$ and the guarantee converges to the standard POA bound $\frac{\lambda}{1-\mu}$. We leave the proof of Corollary 3.3 as an exercise.

4 Epilogue

The key take-home points of this lecture are:

1. Very simple learning algorithms lead remarkably quickly to (approximate) coarse correlated equilibria (CCE).

2. In this sense, CCE are unusually tractable, and hence unusually plausible as a prediction for player behavior.

3. Since POA bounds in smooth games apply to no-regret dynamics, they are particularly robust to the behavioral model.

References

[1] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[2] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[3] J. Hannan. Approximation to Bayes risk in repeated play. *Ccontributions to the Theory of Games*, 3:97–139, 1957.

[4] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.