CHAPTER 4

Learning, Regret Minimization, and Equilibria

Avrim Blum and Yishay Mansour

Abstract

Many situations involve repeatedly making decisions in an uncertain environment: for instance, deciding what route to drive to work each day, or repeated play of a game against an opponent with an unknown strategy. In this chapter we describe learning algorithms with strong guarantees for settings of this type, along with connections to game-theoretic equilibria when all players in a system are simultaneously adapting in such a manner.

We begin by presenting algorithms for repeated play of a matrix game with the guarantee that against any opponent, they will perform nearly as well as the best fixed action in hindsight (also called the problem of combining expert advice or minimizing external regret). In a zero-sum game, such algorithms are guaranteed to approach or exceed the minimax value of the game, and even provide a simple proof of the minimax theorem. We then turn to algorithms that minimize an even stronger form of regret, known as internal or swap regret. We present a general reduction showing how to convert any algorithm for minimizing external regret to one that minimizes this stronger form of regret as well. Internal regret is important because when all players in a game minimize this stronger type of regret, the empirical distribution of play is known to converge to correlated equilibrium.

The third part of this chapter explains a different reduction: how to convert from the full information setting in which the action chosen by the opponent is revealed after each time step, to the partial information (bandit) setting, where at each time step only the payoff of the selected action is observed (such as in routing), and still maintain a small external regret.

Finally, we end by discussing routing games in the Wardrop model, where one can show that if all participants minimize their own external regret, then overall traffic is guaranteed to converge to an approximate Nash Equilibrium. This further motivates price-of-anarchy results.

4.1 Introduction

In this chapter we consider the problem of repeatedly making decisions in an uncertain environment. The basic setting is we have a space of \( N \) actions, such as what route to use to drive to work, or the rows of a matrix game like \{rock, paper, scissors\}. At each time step, the algorithm probabilistically chooses an action (say, selecting what route to take), the environment makes its “move” (setting the road congestions on that day),
and the algorithm then incurs the loss for its action chosen (how long its route took). The process then repeats the next day. What we would like are adaptive algorithms that can perform well in such settings, as well as to understand the dynamics of the system when there are multiple players, all adjusting their behavior in such a way.

A key technique for analyzing problems of this sort is known as regret analysis. The motivation behind regret analysis can be viewed as the following: we design a sophisticated online algorithm that deals with various issues of uncertainty and decision making, and sell it to a client. Our algorithm runs for some time and incurs a certain loss. We would like to avoid the embarrassment that our client will come back to us and claim that in retrospect we could have incurred a much lower loss if we used his simple alternative policy \( \pi \). The regret of our online algorithm is the difference between the loss of our algorithm and the loss using \( \pi \).

Different notions of regret quantify differently what is considered to be a “simple” alternative policy. External regret, also called the problem of combining expert advice, compares performance to the best single action in retrospect. This implies that the simple alternative policy performs the same action in all time steps, which indeed is quite simple. Nonetheless, external regret provides a general methodology for developing online algorithms whose performance matches that of an optimal static offline algorithm by modeling the possible static solutions as different actions. In the context of machine learning, algorithms with good external regret bounds can be powerful tools for achieving performance comparable to the optimal prediction rule from some large class of hypotheses.

In Section 4.3 we describe several algorithms with particularly strong external regret bounds. We start with the very weak greedy algorithm, and build up to an algorithm whose loss is at most \( O(\sqrt{T \log N}) \) greater than that of the best action, where \( T \) is the number of time steps. That is, the regret per time step drops as \( O(\sqrt{\log N}/T) \). In Section 4.4 we show that in a zero-sum game, such algorithms are guaranteed to approach or exceed the value of the game, and even yield a simple proof of the minimax theorem.

A second category of alternative policies are those that consider the online sequence of actions and suggest a simple modification to it, such as “every time you bought IBM, you should have bought Microsoft instead.” While one can study very general classes of modification rules, the most common form, known as internal or swap regret, allows one to modify the online action sequence by changing every occurrence of a given action \( i \) by an alternative action \( j \). (The distinction between internal and swap regret is that internal regret allows only one action to be replaced by another, whereas swap regret allows any mapping from \( \{1, \ldots, N\} \) to \( \{1, \ldots, N\} \) and can be up to a factor \( N \) larger.) In Section 4.5 we present a simple way to efficiently convert any external regret minimizing algorithm into one that minimizes swap regret with only a factor \( N \) increase in the regret term. Using the results for external regret this achieves a swap regret bound of \( O(\sqrt{TN \log N}) \). (Algorithms for swap regret have also been developed from first principles—see the Notes section of this chapter for references—but this procedure gives the best bounds known for efficient algorithms.)

The importance of swap regret is due to its tight connection to correlated equilibria, defined in Chapter 1. In fact, one way to think of a correlated equilibrium is that it is a distribution \( Q \) over the joint action space such that every player would have zero
internal (or swap) regret when playing it. As we point out in Section 4.4, if each player can achieve swap regret $\epsilon T$, then the empirical distribution of the joint actions of the players will be an $\epsilon$-correlated equilibrium.

We also describe how external regret results can be extended to the partial information model, also called the multiarmed bandit (MAB) problem. In this model, the online algorithm only gets to observe the loss of the action actually selected, and does not see the losses of the actions not chosen. For example, in the case of driving to work, you may only observe the travel time on the route you actually drive, and do not get to find out how long it would have taken had you chosen some alternative route. In Section 4.6 we present a general reduction, showing how to convert an algorithm with low external regret in the full information model to one for the partial information model (though the bounds produced are not the best known bounds for this problem).

Notice that the route-choosing problem can be viewed as a general-sum game: your travel time depends on the choices of the other drivers as well. In Section 4.7 we discuss results showing that in the Wardrop model of infinitesimal agents (considered in Chapter 18), if each driver acts to minimize external regret, then traffic flow over time can be shown to approach an approximate Nash equilibrium. This serves to further motivate price-of-anarchy results in this context, since it means they apply to the case that participants are using well-motivated self-interested adaptive behavior.

We remark that the results we present in this chapter are not always the strongest known, and the interested reader is referred to the recent book (Cesa-Bianchi and Lugosi, 2006) that gives a thorough coverage of many of the the topics in this chapter. See also the Notes section for further references.

4.2 Model and Preliminaries

We assume an adversarial online model where there are $N$ available actions $X = \{1, \ldots, N\}$. At each time step $t$, an online algorithm $H$ selects a distribution $p_t$ over the $N$ actions. After that, the adversary selects a loss vector $\ell_t \in [0, 1]^N$, where $\ell_t^i \in [0, 1]$ is the loss of the $i$-th action at time $t$. In the full information model, the online algorithm $H$ receives the loss vector $\ell_t$ and experiences a loss $\ell_t^H = \sum_{i=1}^{N} p_t^i \ell_t^i$. (This can be viewed as an expected loss when the online algorithm selects action $i \in X$ with probability $p_t^i$.) In the partial information model, the online algorithm receives $(\ell_t^i, k_t)$, where $k_t$ is distributed according to $p_t$, and $\ell_t^H = \ell_t^{k_t}$ is its loss. The loss of the $i$-th action during the first $T$ time steps is $L_T^i = \sum_{t=1}^{T} \ell_t^i$, and the loss of $H$ is $L_T^H = \sum_{i=1}^{N} \ell_t^i$. The aim for the external regret setting is to design an online algorithm that will be able to approach the performance of the best algorithm from a given class of algorithms $G$; namely, to have a loss close to $L_T^{\min} = \min_{g \in G} L_T^g$. Formally we would like to minimize the external regret $R_G = L_T^H - L_T^{\min}$, and $G$ is called the comparison class. The most studied comparison class $G$ is the one that consists of all the single actions, i.e., $G = X$. In this chapter we concentrate on this important comparison class, namely, we want the online algorithm’s loss to be close to $L_T^{\min} = \min_i L_T^i$, and let the external regret be $R = L_T^H - L_T^{\min}$.

External regret uses a fixed comparison class $G$, but one can also envision a comparison class that depends on the online algorithm’s actions. We can consider modification
rules that modify the actions selected by the online algorithm, producing an alternative strategy which we will want to compete against. A modification rule $F$ has as input the history and the current action selected by the online procedure and outputs a (possibly different) action. (We denote by $F^t$ the function $F$ at time $t$, including any dependency on the history.) Given a sequence of probability distributions $p^t$ used by an online algorithm $H$, and a modification rule $F$, we define a new sequence of probability distributions $f^t = F(p^t)$, where $f^t_i = \sum_j F^t(j) p^t_j$. The loss of the modified sequence is $L_{H,F} = \sum_t \sum_i f^t_i \ell^t_i$. Note that at time $t$ the modification rule $F$ shifts the probability that $H$ assigned to action $j$ to action $F^t(j)$. This implies that the modification rule $F$ generates a different distribution, as a function of the online algorithm’s distribution $p^t$.

We will focus on the case of a finite set $F$ of memoryless modification rules (they do not depend on history). Given a sequence of loss vectors, the regret of an online algorithm $H$ with respect to the modification rules $F$ is

$$R_F = \max_{F \in F} \{ L^T_H - L^T_{H,F} \}.$$  

Note that the external regret setting is equivalent to having a set $F^\text{ex}$ of $N$ modification rules $F_i$, where $F_i$ always outputs action $i$. For internal regret, the set $F^\text{in}$ consists of $N(N - 1)$ modification rules $F_{i,j}$, where $F_{i,j}(i) = j$ and $F_{i,j}(i') = i'$ for $i' \neq i$. That is, the internal regret of $H$ is

$$\max_{F \in F^\text{in}} \{ L^T_H - L^T_{H,F} \} = \max_{i,j \in X} \left\{ \sum_{t=1}^T p^t_i (\ell^t_i - \ell^t_j) \right\}.$$  

A more general class of memoryless modification rules is swap regret defined by the class $F^\text{sw}$, which includes all $N^N$ functions $F : \{1, \ldots, N\} \to \{1, \ldots, N\}$, where the function $F$ swaps the current online action $i$ with $F(i)$ (which can be the same or a different action). That is, the swap regret of $H$ is

$$\max_{F \in F^\text{sw}} \{ L^T_H - L^T_{H,F} \} = N \max_{i,j \in X} \left\{ \sum_{t=1}^T p^t_i (\ell^t_i - \ell^t_j) \right\}.$$  

Note that since $F^\text{ex} \subseteq F^\text{sw}$ and $F^\text{in} \subseteq F^\text{sw}$, both external and internal regret are upper-bounded by swap regret. (See also Exercises 4.1 and 4.2.)

### 4.3 External Regret Minimization

Before describing the external regret results, we begin by pointing out that it is not possible to guarantee low regret with respect to the overall optimal sequence of decisions in hindsight, as is done in competitive analysis (Borodin and El-Yaniv, 1998; Sleator and Tarjan, 1985). This will motivate why we will be concentrating on more restricted comparison classes. In particular, let $G_{\text{all}}$ be the set of all functions mapping times $\{1, \ldots, T\}$ to actions $X = \{1, \ldots, N\}$.

**Theorem 4.1** For any online algorithm $H$ there exists a sequence of $T$ loss vectors such that regret $R_{G_{\text{all}}}$ is at least $T(1 - 1/N)$. 
The sequence is simply as follows: at each time \( t \), the action \( i_t \) of lowest probability \( p_t^i \) gets a loss of 0, and all the other actions get a loss of 1. Since \( \min \{ p_t^i \} \leq 1/N \), this means the loss of \( H \) in \( T \) time steps is at least \( T(1 - 1/N) \).

On the other hand, there exists \( g \in G_\text{all} \), namely \( g(t) = i_t \), with a total loss of 0.

The above proof shows that if we consider all possible functions, we have a very large regret. For the rest of the section we will use the comparison class \( G_a = \{ g_i : i \in X \} \), where \( g_i \) always selects action \( i \). Namely, we compare the online algorithm to the best single action.

4.3.1 Warmup: Greedy and Randomized-Greedy Algorithms

In this section, for simplicity we will assume that all losses are either 0 or 1 (rather than a real number in \([0, 1]\)), which will simplify notation and proofs, although everything presented can be easily extended to the general case.

Our first attempt to develop a good regret minimization algorithm will be to consider the greedy algorithm. Recall that \( L_i^t = \sum_{\tau=1}^{t} \ell_{\tau}^i \), namely the cumulative loss up to time \( t \) of action \( i \). The Greedy algorithm at each time \( t \) selects action \( x_t = \arg\min_{i \in X} L_i^{t-1} \) (if there are multiple actions with the same cumulative loss, it prefers the action with the lowest index). Formally:

\[
\text{Greedy Algorithm}
\]

Initially: \( x^1 = 1 \).

At time \( t \):
- Let \( L_{\min}^{t-1} = \min_{i \in X} L_i^{t-1} \), and \( S_{\min}^{t-1} = \{ i : L_i^{t-1} = L_{\min}^{t-1} \} \).
- Let \( x^t = \min S_{\min}^{t-1} \).

**Theorem 4.2**  The Greedy algorithm, for any sequence of losses has

\[
L_{\text{Greedy}}^T \leq N \cdot L_{\min}^T + (N - 1).
\]

**Proof**  At each time \( t \) such that Greedy incurs a loss of 1 and \( L_{\min}^t \) does not increase, at least one action is removed from \( S_t \). This can occur at most \( N \) times before \( L_{\min}^t \) increases by 1. Therefore, Greedy incurs loss at most \( N \) between successive increments in \( L_{\min}^t \). More formally, this shows inductively that \( L_{\text{Greedy}}^T \leq N - |S_T| + N \cdot L_{\min}^T \).

The above guarantee on Greedy is quite weak, stating only that its loss is at most a factor of \( N \) larger than the loss of the best action. The following theorem shows that this weakness is shared by any deterministic online algorithm. (A deterministic algorithm concentrates its entire weight on a single action at each time step.)

**Theorem 4.3**  For any deterministic algorithm \( D \) there exists a loss sequence for which \( L_D^T = T \) and \( L_{\min}^T = \lfloor T/N \rfloor \).
Note that the above theorem implies that \( L^T_D \geq N \cdot L^T_{\min} + (T \mod N) \), which almost matches the upper bound for Greedy (Theorem 4.2).

**Proof** Fix a deterministic online algorithm \( D \) and let \( x^t \) be the action it selects at time \( t \). We will generate the loss sequence in the following way. At time \( t \), let the loss of \( x^t \) be 1 and the loss of any other action be 0. This ensures that \( D \) incurs loss 1 at each time step, so \( L^T_D = T \).

Since there are \( N \) different actions, there is some action that algorithm \( D \) has selected at most \( \lfloor T/N \rfloor \) times. By construction, only the actions selected by \( D \) ever have a loss, so this implies that \( L^T_{\min} \leq \lfloor T/N \rfloor \).

Theorem 4.3 motivates considering randomized algorithms. In particular, one weakness of the greedy algorithm was that it had a deterministic tie breaker. One can hope that if the online algorithm splits its weight between all the currently best actions, better performance could be achieved. Specifically, let \( \text{Randomized Greedy} \) (RG) be the procedure that assigns a uniform distribution over all those actions with minimum total loss so far. We now will show that this algorithm achieves a significant performance improvement: its loss is at most an \( O(\log N) \) factor from the best action, rather than \( O(N) \). (This is similar to the analysis of the randomized marking algorithm in competitive analysis.)

**Randomized Greedy (RG) Algorithm**

Initially: \( p_1^0 = 1/N \) for \( i \in X \).

At time \( t \):

- Let \( L_{\min}^{t-1} = \min_{i \in X} L_i^{t-1} \), and \( S^{t-1} = \{ i : L_i^{t-1} = L_{\min}^{t-1} \} \).
- Let \( p_i^t = 1/|S^{t-1}| \) for \( i \in S^{t-1} \) and \( p_i^t = 0 \) otherwise.

**Theorem 4.4** The Randomized Greedy (RG) algorithm, for any loss sequence, has

\[
L_{\text{RG}}^T \leq (\ln N) + (1 + \ln N) L_{\min}^T.
\]

**Proof** The proof follows from showing that the loss incurred by Randomized Greedy between successive increases in \( L_{\min}^T \) is at most \( 1 + \ln N \). Specifically, let \( t_j \) denote the time step at which \( L_{\min}^T \) first reaches a loss of \( j \), so we are interested in the loss of Randomized Greedy between time steps \( t_j \) and \( t_{j+1} \). At time any \( t \) we have \( 1 \leq |S^t| \leq N \). Furthermore, if at time \( t \in (t_j, t_{j+1}] \) the size of \( S^t \) shrinks by \( k \) from some size \( n' \) down to \( n' - k \), then the loss of the online algorithm RG is \( k/n' \), since each such action has weight \( 1/n' \). Finally, notice that we can upper bound \( k/n' \) by \( 1/n' + 1/(n' - 1) + \cdots + 1/(n' - k + 1) \). Therefore, over the entire time-interval \((t_j, t_{j+1}]\), the loss of Randomized Greedy is at most:

\[
1/N + 1/(N - 1) + 1/(N - 2) + \cdots + 1/1 \leq 1 + \ln N.
\]

More formally, this shows inductively that \( L_{\text{RG}}^T \leq (1/N + 1/(N - 1) + \cdots + 1/(|S^t| + 1)) + (1 + \ln N) \cdot L_{\min}^T \). \( \square \)
4.3.2 Randomized Weighted Majority Algorithm

Although Randomized Greedy achieved a significant performance gain compared to the Greedy algorithm, we still have a logarithmic ratio to the best action. Looking more closely at the proof, one can see that the losses are greatest when the sets $S'_t$ are small, since the online loss can be viewed as proportional to $1/|S'_t|$. One way to overcome this weakness is to give some weight to actions which are currently “near best.” That is, we would like the probability mass on some action to decay gracefully with its distance to optimality. This is the idea of the Randomized Weighted Majority algorithm of Littlestone and Warmuth.

Specifically, in the Randomized Weighted Majority algorithm, we give an action $i$ whose total loss so far is $L_i = 1$ and $p_i = 1/N$, for $i \in X$. Initially:

At time $t$:
- If $\ell_i = 1$, let $w_i = w_i^{-1}(1-\eta)$; else ($\ell_i = 0$) let $w_i = w_i^{-1}$.
- Let $p_i = w_i/W_t$, where $W_t = \sum_{i \in X} w_i$.

Algorithm RWM and Theorem 4.5 can be generalized to losses in $[0,1]$ by replacing the update rule with $w_i = w_i^{-1}(1-\eta)^{\ell_i}$ (see Exercise 4.3).

**Theorem 4.5** For $\eta \leq 1/2$, the loss of Randomized Weighted Majority (RWM) on any sequence of binary $\{0,1\}$ losses satisfies

$$L_{RWM}^T \leq (1 + \eta)L_{\text{min}}^T + \frac{\ln N}{\eta}.$$ 

Setting $\eta = \min\{\sqrt{(\ln N)/T}, 1/2\}$ yields $L_{RWM}^T \leq L_{\text{min}}^T + 2\sqrt{T \ln N}$.

(Note: The second part of the theorem assumes $T$ is known in advance. If $T$ is unknown, then a “guess and double” approach can be used to set $\eta$ with just a constant-factor loss in regret. In fact, one can achieve the potentially better bound $L_{RWM}^T \leq L_{\text{min}}^T + 2\sqrt{L_{\text{min}} \ln N}$ by setting $\eta = \min\{\sqrt{(\ln N)/L_{\text{min}}}, 1/2\}$.)

**Proof** The key to the proof is to consider the total weight $W_t$. What we will show is that anytime the online algorithm has significant expected loss, the total weight must drop substantially. We will then combine this with the fact that $W_{t+1} \geq \max_i w_i^{t+1} = (1-\eta)L_{\text{min}}^T$ to achieve the desired bound.

Specifically, let $F_t = (\sum_{i: \ell_i = 1} w_i^{t})/W_t$ denote the fraction of the weight $W_t$ that is on actions that experience a loss of 1 at time $t$; so, $F_t$ equals the expected loss of algorithm RWM at time $t$. Now, each of the actions experiencing a loss of 1 has its weight multiplied by $(1-\eta)$ while the rest are unchanged. Therefore, $W_t^{t+1} = W_t - \eta F_t W_t = W_t(1 - \eta F_t)$. In other words, the proportion of
the weight removed from the system at each time $t$ is exactly proportional to the expected loss of the online algorithm. Now, using the fact that $W^1 = N$ and using our lower bound on $W^T + 1$ we have

$$(1 - \eta)^T_{\text{min}} \leq W^T + 1 = W^1 \prod_{t=1}^{T} (1 - \eta F^t) = N \prod_{t=1}^{T} (1 - \eta F^t).$$

Taking logarithms,

$$L^T_{\text{min}} \ln(1 - \eta) \leq (\ln N) + \sum_{t=1}^{T} \ln(1 - \eta F^t)$$

$$\leq (\ln N) - \sum_{t=1}^{T} \eta F^t$$

(Using the inequality $\ln(1 - z) \leq -z$)

$$= (\ln N) - \eta L^T_{\text{RWM}}$$

(by definition of $F^t$)

Therefore,

$$L^T_{\text{RWM}} \leq \frac{-L^T_{\text{min}} \ln(1 - \eta)}{\eta} + \frac{\ln(N)}{\eta}$$

$$\leq (1 + \eta)L^T_{\text{min}} + \frac{\ln(N)}{\eta},$$

(Using the inequality $-\ln(1 - z) \leq -z + z^2$ for $0 \leq z \leq \frac{1}{2}$)

which completes the proof. \(\Box\)

### 4.3.3 Polynomial Weights Algorithm

The Polynomial Weights (PW) algorithm is a natural extension of the R\text{W}M algorithm to losses in $[0, 1]$ (or even to the case of both losses and gains, see Exercise 4.4) that maintains the same proof structure as that used for R\text{W}M and in addition performs especially well in the case of small losses.

**Polynomial Weights (PW) Algorithm**

- **Initially:** $w^1_i = 1$ and $p^1_i = 1/N$, for $i \in X$.
- **At time $t$:**
  - Let $w^t_i = w^{t-1}_i (1 - \eta \ell_i^{t-1})$.
  - Let $p^t_i = w^t_i / W^t$, where $W^t = \sum_{i \in X} w^t_i$.

Notice that the only difference between PW and R\text{W}M is in the update step. In particular, it is no longer necessarily the case that an action of total loss $L$ has weight $(1 - \eta)^L$. However, what is maintained is the property that if the algorithm’s loss at time $t$ is $F^t$, then exactly an $\eta F^t$ fraction of the total weight is removed from the system. Specifically, from the update rule we have $W^{t+1} = W^t - \sum_i \eta w^t_i \ell_i^t = W^t (1 - \eta F^t)$ where $F^t = (\sum_i w^t_i \ell_i^t) / W^t$ is the loss of PW at time $t$. We can use this fact to prove the following.