Chapter 2

The Multiplicative Weights Update method

The *Multiplicative Weights method* is a simple idea which has been repeatedly discovered in fields as diverse as Machine Learning, Optimization, and Game Theory. The setting for this algorithm is the following. A decision maker has a choice of n decisions, and needs to repeatedly make a decision and obtain an associated payoff. The decision maker's goal, in the long run, is to achieve a total payoff which is comparable to the payoff of that fixed decision that maximizes the total payoff with the benefit of hindsight. While this best decision may not be known *a priori*, it is still possible to achieve this goal by maintaining weights on the decisions, and choosing the decisions randomly with probability proportional to the weights. In each successive round, the weights are updated by multiplying them with factors which depend on the payoff of the associated decision in that round. Intuitively, this scheme works because it tends to focus higher weight on higher payoff decisions in the long run.

This idea lies at the core of a variety of algorithms. Some examples include: Freund and Schapire's AdaBoost algorithm in machine learning [42]; algorithms for game playing studied in economics (see Section 2.4), the Plotkin-Shmoys-Tardos algorithm for packing and covering LPs [85], and its improvements in the case of flow problems by Garg-Könneman [44] and Fleischer [40]; etc. The analysis of the running time uses a potential function argument and the final running time is proportional to $1/\varepsilon^2$.

It has been clear to most researchers that these results are very similar, see for instance, Khandekar's PhD thesis [64]. In this chapter, we develop a unified framework for all these algorithms. This meta algorithm is a generalization of Littlestone and Warmuth's *Weighted Majority* algorithm from learning theory [78]. We call this the Multiplicative Weights algorithm (a similar algorithm, *Hedge*, was developed by Freund and Schapire [42]). This algorithmic framework, and the derivation of previously known algorithms using it, have been studied in much more detail in the survey paper [15]. We also present some applications of this framework in designing algorithms to approximately solve zero-sum games, feasibility problems with concave constraints over a convex domain, and fractional packing and covering linear programs.

2.1 The Weighted Majority Algorithm

Consider the following setting. We are trying to invest in a certain stock. For simplicity, think of its price movements as a sequence of binary events: up/down. (Below, this will be generalized to allow non-binary events.) Each morning we try to predict whether the price will go up or down that day; if our prediction happens to be wrong we lose a dollar that day.

In making our predictions, we are allowed to watch the predictions of n "experts" (who could be arbitrarily correlated, and who may or may not know what they are talking about). The algorithm we present will be able to limit its losses to roughly the same as the *best* of these experts. At first sight this may seem an impossible goal, since it is not known until the end of the sequence who the best expert was, whereas the algorithm is required to make predictions all along.

The algorithm does this by maintaining a *weighting* of the experts. Initially all have equal weight. As time goes on, some experts are seen as making better predictions than others, and the algorithm increases their weight proportionately. The Weighted Majority algorithm is given in Figure 2.1.

Weighted majority algorithm

Initialization: Fix an $\varepsilon \leq \frac{1}{2}$. For each expert i, associate the weight ${w_i}^{(1)} := 1$.
For $t = 1, 2, \ldots, T$:

1. Make the prediction that is the weighted majority of the experts' predictions based on the weights ${w_1}^{(t)}, \ldots, {w_n}^{(t)}$. That is, predict "up" or "down" depending on which prediction has a higher total weight of experts advising it (breaking ties arbitrarily).
2. For every expert i who predicts wrongly, decrease his weight for the next round by multiplying it by a factor of $(1 - \varepsilon)$:

$$ {w_i}^{(t+1)} = (1 - \varepsilon){w_i}^{(t)} \qquad \text{(update rule).} \tag{2.1} $$

Figure 2.1: The Weighted Majority algorithm.

Theorem 1. *After T steps, let ${m_i}^{(T)}$ be the number of mistakes of expert i and $m^{(T)}$ be the number of mistakes our algorithm has made. Then we have the following bound for every i:*

$$ m^{(T)} \leq \frac{2 \ln n}{\varepsilon} + 2(1 + \varepsilon){m_i}^{(T)}. $$

In particular, this holds for i which is the best expert, i.e. having the least ${m_i}^{(T)}$.

Proof: A simple induction shows that ${w_i}^{(t+1)} = (1 - \varepsilon)^{{m_i}^{(t)}}$. Let $\Phi^{(t)} = \sum_i {w_i}^{(t)}$ ("the potential function"). Thus $\Phi^{(1)} = n$. Each time we make a mistake, the weighted majority

of experts also made a mistake, so at least half the total weight decreases by a factor $1-\varepsilon$. Thus, the potential function decreases by a factor of at least $(1-\varepsilon/2)$:

$$\Phi^{(t+1)} \;\leq\; \Phi^{(t)}\left(\frac{1}{2}+\frac{1}{2}(1-\varepsilon)\right) \;=\; \Phi^{(t)}(1-\varepsilon/2).$$

Thus another simple induction gives $\Phi^{(T+1)} \leq n(1-\varepsilon/2)^{m^{(T)}}$. Finally, since ${\Phi_i}^{(T+1)} \geq {w_i}^{(T+1)}$ for all i, the claimed bound follows by comparing the above two expressions and using the fact that $-\ln(1-\varepsilon) \leq \varepsilon+\varepsilon^2$ since $\varepsilon < \frac{1}{2}$. $\square$

The beauty of this analysis is that it makes no assumption about the sequence of events: they could be arbitrarily correlated and could even depend upon our current weighting of the experts. In this sense, this algorithm delivers more than initially promised, and this lies at the root of why (after generalization) it can give rise to the diverse algorithms mentioned earlier. In particular, the scenario where the events are chosen adversarially resembles a zero-sum game, which we consider later in Section 2.3.1.

2.2 The Multiplicative Weights algorithm

In the general setting, we still have n experts. The set of events/outcomes may not be necessarily binary and could even be infinite. To model this, we dispense with the notion of predictions altogether, and instead suppose that in each round, every expert recommends a course of action, and our task is to pick an expert and use his advice. At this point the costs of all actions recommended by the experts is revealed by nature. We suffer the cost of the action recommended by the expert we chose.

To motivate the Multiplicative Weights algorithm, consider the naïve strategy that, in each iteration, simply picks an expert at random. The expected penalty will be that of the "average" expert. Suppose now that a few experts clearly outperform their competitors. This is easy to spot as events unfold, and so it is sensible to reward them by increasing their probability of being picked in the next round (hence the multiplicative weight update rule).

Intuitively, being in complete ignorance about the experts at the outset, we select them uniformly at random for advice. This maximum entropy starting rule reflects our ignorance. As we learn who the hot experts are and who the duds are, we lower the entropy to reflect our increased knowledge. The multiplicative weight update is our means of skewing the distribution.

We now set up some notation. Let $t = 1, 2, \ldots, T$ denote the current round, and let i be a generic expert. In each round t, we select a distribution $\mathbf{p}^{(t)}$ over the set of experts, and select an expert i randomly from it (and use his advised course of action). At this point, the costs of all the actions recommended by the experts are revealed by nature in the form of the vector $\mathbf{m}^{(t)}$ such that expert i incurs cost ${m_i}^{(t)}$. We assume that the costs lie in the range $[-1, 1]$. This is the only assumption we make on the costs; nature is completely free to choose the cost vector as long as these bounds are respected, even with full knowledge of the actions recommended by the experts.

The expected cost to the algorithm for choosing the distribution $\mathbf{p}^{(t)}$ is

$$\mathop{\mathbb{E}}_{i \in \mathbf{p}^{(t)}} \left[{m_i}^{(t)} \right] \;=\; \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}.$$

The total expected cost over all rounds is therefore $\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$. Just as before, our goal is to design an algorithm which achieves a total expected cost not too much more than the cost of the best expert, viz. $\min_i \sum_{t=1}^{T} {m_i}^{(t)}$.

Multiplicative Weights algorithm

Initialization: Fix an $\varepsilon \leq \frac{1}{2}$. For each expert i, associate the weight ${w_i}^{(t)} := 1$.
For $t = 1, 2, \ldots, T$:

1. Choose expert i with probability proportional to his weight ${w_i}^{(t)}$. I.e., use the distribution $\mathbf{p}^{(t)} = \{ {w_1}^{(t)}/\Phi^{(t)}, \ldots, {w_n}^{(t)}/\Phi^{(t)} \}$ where $\Phi^{(t)} = \sum_i {w_i}^{(t)}$.
2. Observe the costs of the experts $\mathbf{m}^{(t)}$.
3. Penalize the costly experts by updating their weights as follows: for every expert i,

$${w_i}^{(t+1)} = \begin{cases} {w_i}^{(t)} (1-\varepsilon)^{{m_i}^{(t)}} & \text{if } {m_i}^{(t)} \geq 0 \\ {w_i}^{(t)} (1+\varepsilon)^{-{m_i}^{(t)}} & \text{if } {m_i}^{(t)} < 0 \end{cases}$$

Figure 2.2: The Multiplicative Weights algorithm.

The following theorem —completely analogous to Theorem 1— bounds the total expected cost of the Multiplicative Weights algorithm (given in Figure 2.2) in terms of the total cost of the best expert:

Theorem 2. *In the given setup, the Multiplicative Weights algorithm guarantees that after T rounds, for any expert i, we have*

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;\leq\; \sum_{t=1}^{T} {m_i}^{(t)} + \varepsilon \sum_{t=1}^{T} |{m_i}^{(t)}| + \frac{\ln n}{\varepsilon}.$$

Proof: We use the following facts, which follow immediately from the convexity of the exponential function:

$$\begin{aligned} (1-\varepsilon)^x &\leq (1 - \varepsilon x) \quad \text{if } x \in [0,1] \\ (1+\varepsilon)^{-x} &\leq (1 - \varepsilon x) \quad \text{if } x \in [-1,0] \end{aligned}$$

The proof is along the lines of the earlier one, using the potential function $\Phi^{(t)} = \sum_i {w_i}^{(t)}$.

Since ${m_i}^{(t)} \in [-1, 1]$, using the facts above we have,

$$\begin{aligned}
\Phi^{(t+1)} &= \sum_i {w_i}^{(t+1)} \\
&= \sum_{i:\ {m_i}^{(t)} \geq 0} {w_i}^{(t)}(1-\varepsilon)^{{m_i}^{(t)}} + \sum_{i:\ {m_i}^{(t)} < 0} {w_i}^{(t)}(1+\varepsilon)^{-{m_i}^{(t)}} \\
&\leq \sum_i {w_i}^{(t)}(1-\varepsilon {m_i}^{(t)}) \\
&= \Phi^{(t)} - \varepsilon\Phi^{(t)} \sum_i {m_i}^{(t)} {p_i}^{(t)} \\
&= \Phi^{(t)}(1 - \varepsilon \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}) \\
&\leq \Phi^{(t)} \exp(-\varepsilon \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}).
\end{aligned}$$

Here, we used the fact that ${p_i}^{(t)} = {w_i}^{(t)}/\Phi^{(t)}$. Thus, by induction, after T rounds, we have

$$\Phi^{(T+1)} \ \leq \ \Phi^{(1)} \exp(-\varepsilon \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}) \ = \ n \cdot \exp(-\varepsilon \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}).$$

Furthermore, for every expert i,

$$\Phi^{(T+1)} \geq {w_i}^{(T+1)} = (1-\varepsilon)^{\sum_{\geq 0} {m_i}^{(t)}} \cdot (1+\varepsilon)^{-\sum_{<0} {m_i}^{(t)}},$$

where the subscripts "≥ 0" and "< 0" in the summations refer to the rounds t where ${m_i}^{(t)}$ is ≥ 0 and < 0 respectively. Now we get the desired bound by taking logarithms and simplifying as before. We used the facts that $\ln(\frac{1}{1-\varepsilon}) \leq \varepsilon + \varepsilon^2$ and $\ln(1+\varepsilon) \geq \varepsilon - \varepsilon^2$ for $\varepsilon \leq \frac{1}{2}$. □

REMARK: From the proof, it can be seen that the following multiplicative update rule:

$${w_i}^{(t+1)} \ = \ {w_i}^{(t)}(1 - \varepsilon {m_i}^{(t)})$$

regardless of the sign of ${m_i}^{(t)}$, would also give the same bounds. Such a rule may be easier to implement.

Corollary 1. *If the costs of all experts lie in the range* $[0, 1]$*, then the Multiplicative Weights algorithm also guarantees that after* T *rounds, for any distribution* $\mathbf{p}$ *on the experts,*

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \ \leq \ (1+\varepsilon) \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p} + \frac{\ln n}{\varepsilon}.$$

PROOF: This corollary follows immediately from Theorem 2, by taking a convex combination of the inequalities for all experts i with the distribution $\mathbf{p}$. □

2.2.1 Gains instead of losses

There are situations where it makes more sense for the vector $\mathbf{m}^{(t)}$ to specify *gains* for each expert rather than losses. Now our goal is to get as much total expected payoff as possible in comparison to the total payoff of the best expert. We can get an algorithm for this case simply by running the Multiplicative Weights algorithm using the loss vector $-\mathbf{m}^{(t)}$.

The algorithm that results updates the weight of expert i by a factor of $(1+\varepsilon)^{m_i{}^{(t)}}$ when $m_i{}^{(t)} \geq 0$, and $(1-\varepsilon)^{-m_i{}^{(t)}}$ when $m_i{}^{(t)} < 0$. The following theorem follows directly from Theorem 2 by simply negating the quantities:

Theorem 3. *In the given setup, the Multiplicative Weights algorithm (for gains) guarantees that after T rounds, for any expert i, we have*

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;\geq\; \sum_{t=1}^{T} m_i{}^{(t)} - \varepsilon \sum_{t=1}^{T} |m_i{}^{(t)}| - \frac{\ln n}{\varepsilon}.$$

2.3 Applications

Typically, the Multiplicative Weights method is applied in the following manner. A prototypical example is to solve a constrained optimization problem. We then let an expert represent each constraint in the problem, and the events correspond to points in the domain of interest. The penalty of the expert is made proportional to *how well* the corresponding constraint is satisfied on the point represented by an event. This might seem counterintuitive, but recall that we *reduce* an expert's weight depending on his penalty, and if an expert's constraint is well satisfied on events so far we would like his weight to be smaller, so that the algorithm focuses on experts whose constraints are poorly satisfied. With these weights, the algorithm generates a *maximally adversarial* event, i.e. the event whose corresponding point maximizes the expected penalty, i.e. the weighted sum of penalties. With this intuition, we can describe the following applications.

2.3.1 Solving zero-sum games approximately

We show how the general algorithm above can be used to approximately solve zero-sum games. This is a duplication of the results of Freund and Schapire [43], who gave the same algorithm but a different proof of convergence that used KL-divergence.

Let $\mathbf{A}$ be the payoff matrix of a finite 2-player zero-sum game, with n rows (the number of columns will play no role). When the row player plays strategy i and the column player plays strategy j, then the payoff to the column player is $A(i,j) := A_{ij}$. We assume that $A(i,j) \in [0,1]$. If the row player chooses his strategy i from a distribution $\mathbf{p}$ over the rows, then the expected payoff to the column player for choosing a strategy j is $A(\mathbf{p},j) := \mathbb{E}_{i\in\mathbf{p}}[A(i,j)]$. Thus, the best response for the column player is the strategy j which maximizes this payoff. Similarly, if the column player chooses his strategy j from a

distribution $\mathbf{q}$ over the columns, then the expected payoff he gets if the row player chooses the strategy i is $A(i, \mathbf{q}) := \mathbb{E}_{j \in \mathbf{q}}[A(i, j)]$. Thus, the best response for the row player is the strategy i which minimizes this payoff. John von Neumann's min-max theorem says that if each of the players chooses a distribution over their strategies to optimize their worst case payoff (or payout), then the value they obtain is the same:

$$\min_{\mathbf{p}} \max_{j} A(\mathbf{p}, j) \;=\; \max_{\mathbf{q}} \min_{i} A(i, \mathbf{q}) \tag{2.2}$$

where $\mathbf{p}$ (resp., $\mathbf{q}$) varies over all distributions over rows (resp., columns). Also, i (resp., j) varies over all rows (resp., columns). The common value of these two quantities, denoted λ^*, is known as the value of the game.

Let $\delta > 0$ be an error parameter. We wish to approximately solve the zero-sum game up to additive error of δ, namely, find mixed row and column strategies $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{q}}$ such that

$$\lambda^* - \delta \;\leq\; \min_{i} A(i, \tilde{\mathbf{q}}) \tag{2.3}$$

$$\max_{j} A(\tilde{\mathbf{p}}, j) \;\leq\; \lambda^* + \delta. \tag{2.4}$$

The algorithmic assumption about the game is that given any distribution $\mathbf{p}$ on experts, we have an efficient way to pick the best event, namely, the pure column strategy j that maximizes $A(\mathbf{p}, j)$. This quantity is at least λ^* from the definition above. Call this algorithm the ORACLE.

Theorem 4. *Given an error parameter $\delta > 0$, there is an algorithm which solves the zero-sum game up to an additive factor of δ using $O(\frac{\log n}{\delta^2})$ calls to* ORACLE*, with an additional processing time of $O(n)$ per call.*

PROOF: We map our general algorithm from Section 2.2 to this setting by considering (2.3) as specifying n linear constraints on the probability vector $\tilde{\mathbf{q}}$: viz., for all rows i, $A(i, \tilde{\mathbf{q}}) \geq \lambda^* - \delta$. Now, following the intuition given in the beginning of this section, we make our "experts" to correspond to pure strategies of the row player. Thus a distribution on the experts corresponds to a mixed row strategy. "Events" correspond to pure strategies of the column player. The penalty paid by an expert i when an event j happens is $A(i, j)$.

In each round, given a distribution $\mathbf{p}^{(t)}$ on the rows, we will set the event $j^{(t)}$ to be the best response strategy to $\mathbf{p}^{(t)}$ for the column player, by calling ORACLE. Thus, the cost vector $\mathbf{m}^{(t)}$ is the $j^{(t)}$-th column of the matrix $\mathbf{A}$.

Since all $A(i, j) \in [0, 1]$, we can apply Corollary 1 to get that after T rounds, for any distribution on the rows $\mathbf{p}$, we have

$$\sum_{t=1}^{T} A(\mathbf{p}^{(t)}, j^{(t)}) \;\leq\; (1+\varepsilon) \sum_{t=1}^{T} A(\mathbf{p}, j^{(t)}) + \frac{\ln n}{\varepsilon}.$$

Dividing by T, and using the fact that $A(\mathbf{p}, j^{(t)}) \leq 1$ and that for all t, $A(\mathbf{p}^{(t)}, j^{(t)}) \geq \lambda^*$, we get

$$\lambda^* \;\leq\; \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}^{(t)}, j^{(t)}) \;\leq\; \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}, j^{(t)}) + \varepsilon + \frac{\ln n}{\varepsilon T}$$

Setting $\mathbf{p} = \mathbf{p}^*$, the optimal row strategy, we have $A(\mathbf{p}, j) \leq \lambda^*$ for any j. By setting $\varepsilon = \frac{\delta}{2}$ and $T = \lceil \frac{4 \ln n}{\delta^2} \rceil$, we get that

$$\lambda^* \;\leq\; \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}^{(t)}, j^{(t)}) \;\leq\; \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}, j^{(t)}) + \delta \;\leq\; \lambda^* + \delta. \tag{2.5}$$

Thus, $\frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}^{(t)}, j^{(t)})$ is an (additive) δ-approximation to λ^*.

Let $\tilde{t}$ be the round t with the minimum value of $A(\mathbf{p}^{(t)}, j^{(t)})$. We have, from (2.5),

$$A(\mathbf{p}^{(\tilde{t})}, j^{(\tilde{t})}) \;\leq\; \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}^{(t)}, j^{(t)}) \;\leq\; \lambda^* + \delta.$$

Since $j^{(\tilde{t})}$ maximizes $A(\mathbf{p}^{(\tilde{t})}, j)$ over all j, we conclude that $\mathbf{p}^{(\tilde{t})}$ is an approximately optimal mixed row strategy, and thus we can set $\mathbf{p}^* := \mathbf{p}^{(\tilde{t})}$. [1]

We set $\mathbf{q}^*$ to be the distribution which assigns to column j the probability $\frac{|\{t:\, j^{(t)} = j\}|}{T}$. From (2.5), for any row strategy i, by setting $\mathbf{p}$ to be concentrated on the pure strategy i, we have

$$\lambda^* - \delta \;\leq\; \frac{1}{T} \sum_{t=1}^{T} A(i, j^{(t)}) \;=\; A(i, \mathbf{q}^*)$$

which shows that $\mathbf{q}^*$ is an approximately optimal mixed column strategy. □

2.3.2 Approximating Linear Feasibility Programs on Convex Domains

Plotkin, Shmoys, and Tardos [85] generalized some known flow algorithms to a framework for approximately solving *fractional packing and covering* problems. Their algorithm is a quantitative version of the classical *Lagrangian relaxation* idea, and applies also to general linear programs. Below, we derive the algorithm for convex programs which can be stated as trying to find a point in a convex domain satisfying a number of linear inequalities. We will then mention the slight modification that yields better running time for fractional packing and covering LPs.

The basic problem is to check the feasibility of the following convex program:

$$\exists ? \mathbf{x} \;\in\; \mathcal{P} : \quad \mathbf{A}\mathbf{x} \;\geq\; \mathbf{b} \tag{2.6}$$

where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{x} \in \mathbb{R}^n$, and $\mathcal{P}$ is a convex set in $\mathbb{R}^n$. Intuitively, the set $\mathcal{P}$ represents the "easy" constraints to satisfy, such as non-negativity, and $\mathbf{A}$ represents the "hard" constraints to satisfy.

We wish to design an algorithm that given an error parameter $\delta > 0$, either solves the problem to an additive error of δ, i.e., finds an $\mathbf{x} \in P$ such that for all i, $\mathbf{A}_i \mathbf{x} \geq b_i - \delta$, or failing that, proves that the system is infeasible. Here, $\mathbf{A}_i$ is the i^{th} row of $\mathbf{A}$.

[1] Alternatively, we can set $\mathbf{p}^* = \frac{1}{T} \sum_t \mathbf{p}^{(t)}$. For let j^* be the optimal column player response to $\mathbf{p}^*$. Then we have $A(\mathbf{p}^*, j^*) = \frac{1}{T} \sum_t A(\mathbf{p}^{(t)}, j^*) \leq \frac{1}{T} \sum_t A(\mathbf{p}^{(t)}, j^{(t)}) \leq \lambda^* + \delta$.

We assume the existence of an algorithm, called ORACLE, which, given a probability vector $\mathbf{p}$ on the m constraints, solves the following feasibility problem:

$$\exists ?\mathbf{x} \in \mathcal{P}: \quad \mathbf{p}^\top \mathbf{A}\mathbf{x} \ \geq \ \mathbf{p}^\top \mathbf{b} \tag{2.7}$$

It is reasonable to expect such an optimization procedure to exist (indeed, such is the case for many applications) since we only need to check the feasibility of one constraint rather than m. If the feasibility problem (2.6) has a solution $\mathbf{x}^*$, then the same solution also satisfies (2.7) for *any* probability vector $\mathbf{p}$ over the constraints. Thus, if there is a probability vector $\mathbf{p}$ over the constraints such that no $\mathbf{x} \in \mathcal{P}$ satisfies (2.7), then it is proof that the original problem is infeasible.

We assume that the ORACLE satisfies the following technical condition, which is necessary for deriving running time bounds:

Definition 1. *An* (ℓ, ρ)**-bounded** ORACLE*, for parameters* $0 \leq \ell \leq \rho$*, is an algorithm which given a probability vector* $\mathbf{p}$ *over the constraints, solves the feasibility problem (2.7). Furthermore, there is a fixed subset* $I \subseteq [m]$ *of constraints such that whenever the* ORACLE *manages to find a point* $\mathbf{x} \in \mathcal{P}$ *satisfying (2.7), the following holds:*

$$\begin{aligned} \forall i \in I: \quad & \mathbf{A}_i\mathbf{x} - b_i \ \in \ [-\ell, \rho] \\ \forall i \notin I: \quad & \mathbf{A}_i\mathbf{x} - b_i \ \in \ [-\rho, \ell] \end{aligned}$$

The value ρ *is called the* **width** *of the problem.*

In previous work, such as [85], only (ρ, ρ)-bounded ORACLEs are considered. We separate out the upper and lower bounds in order to obtain tighter guarantees on the running time. The results of [85] can be recovered simply by setting $\ell = \rho$.

Theorem 5. *Let* $\delta > 0$ *be a given error parameter. Suppose there exists an* (ℓ, ρ)*-bounded* ORACLE *for the feasibility problem (2.6). Assume that* $\ell \geq \frac{\delta}{2}$*. Then there is an algorithm which either solves the problem up to an additive error of* δ*, or correctly concludes that the system is infeasible, making only* $O(\frac{\ell\rho\log(m)}{\delta^2})$ *calls to the* ORACLE*, with an additional processing time of* $O(m)$ *per call.*

PROOF: The condition $\ell \geq \frac{\delta}{2}$ is only technical, and if it is not met we can just redefine ℓ to be $\frac{\delta}{2}$. To map our general framework to this situation, we have an expert representing each of the m constraints. Events correspond to vectors $\mathbf{x} \in \mathcal{P}$. The loss of the expert corresponding to constraint i for event $\mathbf{x}$ is $\frac{1}{\rho}[\mathbf{A}_i\mathbf{x} - b_i]$ (so that the costs lie in the range $[-1, 1]$).

In each round t, given a distribution over the experts (i.e. the constraints) $\mathbf{p}^{(t)}$, we run the ORACLE with $\mathbf{p}^{(t)}$. If the ORACLE declares that there is no $\mathbf{x} \in \mathcal{P}$ such that ${\mathbf{p}^{(t)}}^\top \mathbf{A}\mathbf{x} \geq {\mathbf{p}^{(t)}}^\top \mathbf{b}$, then we stop, because now $\mathbf{p}^{(t)}$ is proof that the problem (2.6) is infeasible.

So let us assume that this doesn't happen, i.e. in all rounds t, the ORACLE manages to find a solution $\mathbf{x}^{(t)}$ such ${\mathbf{p}^{(t)}}^\top \mathbf{A}\mathbf{x} \geq {\mathbf{p}^{(t)}}^\top \mathbf{b}$. Since the cost vector to the Multiplicative

Weights algorithm is specified to be $\mathbf{m}^{(t)} := \frac{1}{\rho}[\mathbf{A}\mathbf{x}^{(t)} - \mathbf{b}]$, we conclude that the expected cost in each round is non-negative:

$$\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;=\; \frac{1}{\rho}[\mathbf{A}\mathbf{x}^{(t)} - \mathbf{b}] \cdot \mathbf{p}^{(t)} \;=\; \frac{1}{\rho}[{\mathbf{p}^{(t)}}^{\top}\mathbf{A}\mathbf{x} - {\mathbf{p}^{(t)}}^{\top}\mathbf{b}] \;\geq\; 0.$$

Let $i \in I$. Then Theorem 2 tells us that after T rounds,

$$\begin{aligned} 0 \;&\leq\; \sum_{t=1}^{T} \frac{1}{\rho}[\mathbf{A}_i\mathbf{x}^{(t)} - b_i] + \varepsilon \sum_{t=1}^{T} \frac{1}{\rho}|\mathbf{A}_i\mathbf{x}^{(t)} - b_i| + \frac{\ln m}{\varepsilon} \\ &=\; (1+\varepsilon)\sum_{t=1}^{T} \frac{1}{\rho}[\mathbf{A}_i\mathbf{x}^{(t)} - b_i] + 2\varepsilon \sum_{<0} \frac{1}{\rho}|\mathbf{A}_i\mathbf{x}^{(t)} - b_i| + \frac{\ln m}{\varepsilon} \\ &\leq\; (1+\varepsilon)\sum_{t=1}^{T} \frac{1}{\rho}[\mathbf{A}_i\mathbf{x}^{(t)} - b_i] + \frac{2\varepsilon\ell}{\rho}T + \frac{\ln m}{\varepsilon} \end{aligned}$$

Here, the subscript "< 0" refers to the rounds t when $\mathbf{A}_i\mathbf{x}^{(t)} - b_i < 0$. The last inequality follows because if $\mathbf{A}_i\mathbf{x}^{(t)} - b_i < 0$, then $|\mathbf{A}_i\mathbf{x}^{(t)} - b_i| \leq \ell$. Dividing by T, multiplying by ρ, and letting $\bar{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{x}^{(t)}$ (note that $\bar{\mathbf{x}} \in \mathcal{P}$ since $\mathcal{P}$ is a convex set), we get that

$$0 \;\leq\; (1+\varepsilon)[\mathbf{A}_i\bar{\mathbf{x}} - b_i] + 2\varepsilon\ell + \frac{\rho\ln(m)}{\varepsilon T}.$$

Now, if we choose $\varepsilon = \frac{\delta}{4\ell}$ (note that $\varepsilon \leq \frac{1}{2}$ since $\ell \geq \frac{\delta}{2}$), and $T = \lceil \frac{8\ell\rho\ln(m)}{\delta^2} \rceil$, we get that

$$0 \;\leq\; (1+\varepsilon)[\mathbf{A}_i\bar{\mathbf{x}} - b_i] + \delta \quad\Longrightarrow\quad \mathbf{A}_i\bar{\mathbf{x}} \;\geq\; b_i - \delta.$$

Reasoning similarly for $i \notin I$, we get the same inequality. Putting both together, we conclude that $\bar{\mathbf{x}}$ satisfies the feasibility problem (2.6) up to an additive δ factor, as desired. □

Concave constraints

The algorithm of Section 2.3.2 works not just for linear constraints over a convex domain, but also for concave constraints. Imagine that we have the following feasibility problem:

$$\exists ?\mathbf{x} \in \mathcal{P} : \quad \forall i \in [m] : \; f_i(\mathbf{x}) \geq 0 \tag{2.8}$$

where, as before, $\mathcal{P} \in \mathbb{R}^n$ is a convex domain, and for $i \in [m]$, $f_i : \mathcal{P} \to \mathbb{R}$ are concave functions. We wish to satisfy this system approximately, up to an additive error of δ. Again, we assume the existence of an Oracle, which, when given a probability distribution $\mathbf{p} = \langle p_1, p_2, \ldots, p_m \rangle^{\top}$, solves the following feasibility problem:

$$\exists ?\mathbf{x} \in \mathcal{P} : \quad \sum_i p_i f_i(\mathbf{x}) \geq 0 \tag{2.9}$$

An Oracle would be called (ℓ, ρ)-bounded there is a fixed subset of constraints $I \subseteq [m]$ such that whenever it returns a feasible solution $\mathbf{x}$ to (2.9), all constraints $i \in I$ take values in the range $[-\ell, \rho]$ on the point $\mathbf{x}$, and all the rest take values in $[-\rho, \ell]$.

Theorem 6. *Let $\delta > 0$ be a given error parameter. Suppose there exists an (ℓ, ρ)-bounded* ORACLE *for the feasibility problem (2.8). Assume that $\ell \geq \frac{\delta}{2}$. Then there is an algorithm which either solves the problem up to an additive error of δ, or correctly concludes that the system is infeasible, making only $O(\frac{\ell\rho\log(m)}{\delta^2})$ calls to the* ORACLE*, with an additional processing time of $O(m)$ per call.*

PROOF: Just as before, we have an expert for every constraint, and events correspond to $\mathbf{x} \in \mathcal{P}$. The loss of the expert corresponding to constraint i for event $\mathbf{x}$ is $\frac{1}{\rho} f_i(\mathbf{x})$.

Now we run the Multiplicative Weights algorithm with this setup. Again, if at any point the ORACLE declares that (2.9) is infeasible, we immediately halt and declare the system (2.8) infeasible. So assume this never happens. Then as before, the expected cost in each round is $\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \geq 0$. Now, applying Theorem 2 as before, we conclude that for any $i \in I$, we have

$$0 \;\leq\; (1+\varepsilon)\sum_{t=1}^{T} \frac{1}{\rho} f_i(\mathbf{x}^{(t)}) + \frac{2\varepsilon\ell}{\rho} T + \frac{\ln m}{\varepsilon}.$$

Dividing by T, multiplying by ρ, and letting $\bar{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{x}^{(t)}$ (note that $\bar{\mathbf{x}} \in \mathcal{P}$ since $\mathcal{P}$ is a convex set), we get that

$$0 \;\leq\; (1+\varepsilon) f_i(\bar{\mathbf{x}}) + 2\varepsilon\ell + \frac{\rho \ln(m)}{\varepsilon T},$$

since $\frac{1}{T}\sum_{t=1}^{T} f_i(\mathbf{x}^{(t)}) \leq f_i(\frac{1}{T}\sum_{t=1}^{T} \mathbf{x}^{(t)})$, by Jensen's inequality, since all the f_i are concave.

Now, if we choose $\varepsilon = \frac{\delta}{4\ell}$ (note that $\varepsilon \leq \frac{1}{2}$ since $\ell \geq \frac{\delta}{2}$), and $T = \lceil \frac{8\ell\rho\ln(m)}{\delta^2} \rceil$, we get that

$$0 \;\leq\; (1+\varepsilon) f_i(\bar{\mathbf{x}}) + \delta \quad\Longrightarrow\quad f_i(\bar{\mathbf{x}}) \;\geq\; -\delta.$$

Reasoning similarly for $i \notin I$, we get the same inequality. Putting both together, we conclude that $\bar{\mathbf{x}}$ satisfies the feasibility problem (2.8) up to an additive δ factor, as desired. □

Approximate Oracles

The algorithm described in the previous section allows some slack for the implementation of the ORACLE. This slack is very useful in designing efficient implementations for the ORACLE.

Define a δ-approximate ORACLE for the feasibility problem (2.6) to be one that solves the feasibility problem (2.7) up to an additive error of δ. That is, given a probability vector $\mathbf{p}$ on the constraints, either it finds an $\mathbf{x} \in \mathcal{P}$ such that $\mathbf{p}^\top \mathbf{A}\mathbf{x} \geq \mathbf{p}^\top \mathbf{b} - \delta$, or it declares correctly that (2.7) is infeasible.

Theorem 7. *Let $\delta > 0$ be a given error parameter. Suppose there exists an (ℓ, ρ)-bounded $\frac{\delta}{3}$-approximate* ORACLE *for the feasibility problem (2.6). Assume that $\ell \geq \frac{\delta}{3}$. Then there is an algorithm which either solves the problem up to an additive error of δ, or correctly concludes that the system is infeasible, making only $O(\frac{\ell\rho\log(m)}{\delta^2})$ calls to the* ORACLE*, with an additional processing time of $O(m)$ per call.*

PROOF: We run the algorithm of the previous section with the given ORACLE, setting $\varepsilon = \frac{\delta}{6\ell}$. Now, in every round, the expected payoff is at least $-\frac{\delta}{3\rho}$. Simplifying as before, we get that after T rounds, we have, the average point $\bar{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{x}^{(t)}$ returned by the ORACLE satisfies

$$-\frac{\delta}{3} \;\leq\; (1+\varepsilon)[\mathbf{A}_i\bar{\mathbf{x}} - b_i] + 2\varepsilon\ell + \frac{\rho\ln(m)}{\varepsilon T}.$$

Now, if $T = \lceil \frac{18\ell\rho\ln(m)}{\delta^2} \rceil$, then we get that for all i, $\mathbf{A}_i\bar{\mathbf{x}} \geq b_i - \delta$, as required. □

Fractional Covering Problems

In fractional covering problems, the framework is the same as above, with the crucial difference that the coefficient matrix $\mathbf{A}$ is such that $\mathbf{Ax} \geq 0$ for all $\mathbf{x} \in \mathcal{P}$, and $\mathbf{b} > 0$. A δ-approximation solution to this system is an $\mathbf{x} \in \mathcal{P}$ such that $\mathbf{Ax} \geq (1-\delta)\mathbf{b}$.

We assume without loss of generality (by appropriately scaling the inequalities) that $b_i = 1$ for all rows, so that now we desire to find an $\mathbf{x} \in \mathcal{P}$ which satisfies the system within an additive δ factor. Since for all $\mathbf{x} \in \mathcal{P}$, we have $\mathbf{Ax} \geq 0$, and since all $b_i = 1$, we conclude that for any i, $\mathbf{A}_i\mathbf{x} - b_i \geq -1$. Thus, we assume that there is a $(1,\rho)$-bounded ORACLE for this problem. Now, applying Theorem 5, we get the following:

Theorem 8. *Suppose there exists a $(1,\rho)$-bounded* ORACLE *for the program $\mathbf{Ax} \geq \mathbf{b}$ with $\mathbf{x} \in \mathcal{P}$. Given an error parameter $\delta > 0$, there is an algorithm which computes a δ-approximate solution to the program, or correctly concludes that it is infeasible, using $O(\frac{\rho\log(m)}{\delta^2})$ calls to the* ORACLE*, plus an additional processing time of $O(m)$ per call.*

Fractional Packing Problems

A fractional packing problem can be written as

$$\exists ?\mathbf{x} \in \mathcal{P}: \quad \mathbf{Ax} \;\leq\; \mathbf{b}$$

where $\mathcal{P}$ is a convex domain such that $\mathbf{Ax} \geq 0$ for all $\mathbf{x} \in \mathcal{P}$, and $\mathbf{b} > 0$. A δ-approximate solution to this system is an $\mathbf{x} \in \mathcal{P}$ such that $\mathbf{Ax} \leq (1+\delta)\mathbf{b}$.

Again, we assume that $b_i = 1$ for all i, scaling the constraints if necessary. Now by rewriting this system as

$$\exists ?\mathbf{x} \in \mathcal{P}: \quad -\mathbf{Ax} \;\geq\; -\mathbf{b}$$

we cast it in our general framework, and a solution $\mathbf{x} \in \mathcal{P}$ which satisfies this up to an additive δ is a δ-approximate solution to the original system. Since for all $\mathbf{x} \in \mathcal{P}$, we have $\mathbf{Ax} \geq 0$, and since all $b_i = 1$, we conclude that for any i, $-\mathbf{A}_i\mathbf{x} + b_i \leq 1$. Thus, we assume that there is a $(1,\rho)$-bounded ORACLE for this problem. Now, applying Theorem 5, we get the following:

Theorem 9. *Suppose there exists a $(1,\rho)$-bounded* ORACLE *for the program $-\mathbf{Ax} \geq -\mathbf{b}$ with $\mathbf{x} \in \mathcal{P}$. Given an error parameter $\delta > 0$, there is an algorithm which computes a δ-approximate solution to the program, or correctly concludes that it is infeasible, using $O(\frac{\rho\log(m)}{\delta^2})$ calls to the* ORACLE*, plus an additional processing time of $O(m)$ per call.*

2.4 A brief history of various applications of the Multiplicative Weights method

An algorithm similar in flavor to the Multiplicative Weights algorithm were proposed in game theory in the early fifties [29, 28, 86]. Following Brown [28], this algorithm was called "Fictitious Play": at each step each player observes actions taken by his opponent in previous stages, updates his beliefs about his opponents' strategies, and chooses myopic pure best responses against these beliefs. In the simplest case, the player simply assumes that the opponent is playing form an stationary distribution and sets his current belief of the opponent's distribution to be the empirical frequency of the strategies played by the opponent. This simple idea (which was shown to lead to optimal solutions in the limit in various cases) led to many subfields of economics, including Arrow-Debreu General Equilibrium theory and more recently, evolutionary game theory. Grigoriadis and Khachiyan [49] showed how a randomized variant of "Fictitious Play" can solve two player zero-sum games efficiently. This algorithm is precisely the multiplicative weights algorithm. It can be viewed as a soft version of fictitious play, when the player gives higher weight to the strategies which pay off better, and chooses her strategy using these weights rather than choosing the myopic best response strategy.

In Machine Learning, the earliest form of the multiplicative weights update rule was used by Littlestone in his well-known Winnow algorithm [76, 77]. This algorithm was generalized by Littlestone and Warmuth [78] in the form of the Weighted Majority algorithm.

The multiplicative update rule (and the exponential potential function) was also discovered in Computational Geometry in the late 1980s [34] and several applications in geometry are described in Chazelle [33] (p. 6, and p. 124).

The weighted majority algorithm as well as more sophisticated versions have been independently discovered in operations research and statistical decision making in the context of the *On-line decision problem*; see the surveys of Cover [37], Foster and Vohra [41], and also Blum [23] who includes applications of weighted majority to machine learning. A notable algorithm, which is different from but related to our framework, was developed by Hannan in the fifties [52]. Kalai and Vempala showed how to derive efficient algorithms via similar methods [59].

Within computer science, several researchers have previously noted the close relationships between multiplicative update algorithms used in different contexts. Young [103] notes the connection between fast LP algorithms and Raghavan's method of pessimistic estimators for derandomization of randomized rounding algorithms. Klivans and Servedio [70] relate boosting algorithms in learning theory to proofs of Yao's XOR Lemma. Garg and Khandekar describe a common framework for convex optimization problems that contains Garg-Könemann [44] and Plotkin-Shmoys-Tardos [85] as subcases.

In the survey paper [15], we use the framework developed in this chapter to unify previously known applications of the Multiplicative Weights method. In the same paper, we also give lower bounds (inspired by the work of Klein and Young [69]) that show that the analysis of the Multiplicative Weights algorithm is tight in the various parameters involved.