

# The Multiplicative Weights Update Method: A Meta Algorithm and its Applications

Sanjeev Arora*	Elad Hazan†	Satyen Kale†
Princeton University	IBM Almaden	Microsoft Research
35 Olden St,	650 Harry Rd,	1 Microsoft Way,
Princeton NJ 08540	San Jose, CA 95120	Redmond, WA 98052
arora@cs.princeton.edu	hazan@us.ibm.com	sakale@microsoft.com

## Abstract

Algorithms in varied fields use the idea of maintaining a distribution over a certain set and use the *multiplicative update rule* to iteratively change these weights. Their analyses are usually very similar and rely on an exponential potential function.

In this survey we present a simple meta algorithm that unifies these disparate algorithms and drives them as simple instantiations of the meta algorithm. We feel that since this meta algorithm and its analysis are so simple, and its applications so broad, it should be a standard part of algorithms courses, like “divide and conquer.”

## 1 Introduction

Algorithms in varied fields work as follows: a distribution is maintained on a certain set, and at each step the probability assigned to  $i$  is multiplied or divided by  $(1 + \epsilon C(i))$  where  $C(i)$  is some kind of “payoff” for element  $i$ . (Rescaling may be needed to ensure that the new values form a distribution.) Some examples include: the Ada Boost algorithm in machine learning [FS97]; algorithms for game playing studied in economics (see references later), the Plotkin-Shmoys-Tardos algorithm for packing and covering LPs [PST91], and its improvements in the case of flow problems by Young, Garg-Konneman, Fleischer and others [You95, GK98, Fle00]; methods for

---

\*This project was supported by David and Lucile Packard Fellowship and NSF grants MSPA-MCS 0528414 and CCR-0205594.

†Work done while the author was at Princeton University.

convex optimization like gradient descent, lagrangian multipliers, and sub-gradient methods, Impagliazzo’s proof of the Yao XOR lemma [Imp95], etc. The analysis of the running time uses a potential function argument and the final running time is proportional to  $1/\epsilon^2$ .

It has been clear to most researchers that these results are very similar. For example Khandekar’s PhD thesis [Kha04] makes this point about the varied applications of this idea to convex optimization. The purpose of this survey is to clarify that these applications are instances of the same (more general) algorithm. This meta algorithm is a generalization of Littlestone and Warmuth’s *weighted majority* algorithm from learning theory [LW94]. (A similar algorithm has been independently rediscovered in many other fields; see below.) The advantage of deriving the above algorithms from the same meta algorithm is that this highlights their commonalities as well as their differences. To give an example, the algorithms of Garg-Könemann [GK98] were felt to be quite different from those of Plotkin-Shmoys-Tardos [PST91]. In our framework, they can be seen as a clever trick for “width reduction” in the PST framework (see Section 3.3).

We feel that our meta algorithm and its analysis are simple and useful enough that they should be viewed as a basic tool taught to all algorithms students together with divide-and-conquer, dynamic programming, random sampling, and the like. Note that the matrix multiplicative weights update rule may be seen as a “constructive” version of LP duality —equivalently, von Neumann’s minmax theorem in game theory— and it gives a fairly concrete method for competing players to arrive at a solution/equilibrium (see Section 3.1). This may be an appealing feature in introductory algorithms courses, since the standard algorithms for LP such as Simplex, ellipsoid, or Karmarkar lack such a game-theoretic interpretation. Furthermore, it is a convenient stepping point to many other topics that rarely get mentioned in algorithms courses, including *online algorithms* (see the basic scenario in Section 1.1) and *machine learning*. Furthermore our proofs seems easier and cleaner than the entropy-based proofs for the same results in machine learning.

The current paper is chiefly a survey. It introduces the main algorithm, gives a few variants (chiefly having to do with the range in which the payoffs lie), and surveys the most important applications —often with complete proofs. There are a few small results that appear to be new, such as the improved lowerbound of Section 4 and the variant of the Garg-Könemann algorithm in Section 3.3.

**Related work.** An algorithm similar in flavor to the Multiplicative Weights algorithm were proposed in game theory in the early fifties [BvN50, Bro51, Rob51]. Following Brown [Bro51], 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 from a 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 [GK95] 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 [Lit87, Lit89]. It is somewhat reminiscent of the older *perceptron* learning algorithm of Minsky and Papert [MP69]. The Winnow algorithm was generalized by Littlestone and Warmuth [LW94] in the form of the Weighted Majority algorithm. More recent learning algorithms in the so-called *maximum entropy* framework are also related. We note that most relevant papers in learning theory use an analysis that relies on entropy (or its cousin, Kullback-Leibler divergence) calculations. This analysis is closely related to our analysis, but we use *exponential* functions instead of the *logarithm* used in those paper. The underlying calculation is the same: whereas we repeatedly use the fact that  $e^x \approx 1 + x$  when  $|x|$  is small, they use the fact that  $\ln(1 + x) \approx x$ . We feel that our approach is cleaner.

The multiplicative update rule (and the exponential potential function) was also discovered in Computational Geometry in the late 1980s [CW89] and several applications in geometry are described in Chazelle [Cha00] (p. 6, and p. 124). See also our Section 3.11, which also mentions some more recent applications to geometric embeddings of finite metric spaces.

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 [Cov96], Foster and Vohra [FV99], and also Blum [Blu98] 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 [Han57]. Kalai and Vempala showed how to derive efficient algorithms via methods similar to Hannan’s [KV03].

Within computer science, several researchers have previously noted the close relationships between multiplicative update algorithms used in different contexts. Young [You95] notes the connection between fast LP algorithms and Raghavan’s method of pessimistic estimators for derandomization of randomized rounding algorithms; see our Section 3.4. Klivans and Servedio [KS03] relate boosting algorithms in learning theory to proofs of Yao’s XOR Lemma; see our Section 3.5. Garg and Khandekar [GK04] describe a common framework for convex optimization problems that contains Garg-Könemann and Plotkin-Shmoys-Tardos as subcases.

To the best of our knowledge our framework is the most general and arguably, the simplest. We readily acknowledge the influence of all previous papers (especially Young [You95] and Freund-Schapire [FS99]) on the development of our framework. *Disclaimer:* We do not claim that *every* algorithm designed using the multiplicative update idea fits in our framework, just that most do. Some applications in multiparty settings do not easily fit into our framework; see Section 3.8 for some examples.

## 1.1 The weighted majority algorithm

Now we briefly illustrate the weighted majority algorithm in a simple and concrete setting, which will naturally lead into our generalized meta algorithm.

Imagine the process of picking good times to invest in a stock. For simplicity, assume that there is a single stock of interest, and its daily price movement is modeled 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.

We make the *arbitrary* and even *adversarial*. To balance out this pessimistic assumption, we assume that while making our predictions, we are allowed to watch the predictions of  $n$  “experts. ” These experts could be arbitrarily correlated, and who may or may not know what they are talking about. The algorithm’s goal is to limit its cumulative losses (i.e., bad predictions) to roughly the same as the *best* of these experts. At first sight this seems 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.

Indeed, the first algorithm one thinks of is to compute each day’s up/down prediction by going with the majority opinion among the experts that day. But, this algorithm doesn’t work because a majority of experts may be consistently wrong on every single day.

The weighted majority algorithm corrects the trivial algorithm. It maintains 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 algorithm’s prediction of up/down for each day is computed by going with the opinion of the weighted majority of the experts for that day.

**Weighted majority algorithm**

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

**For**  $t = 1, 2, \dots, T$ :

1. Make the prediction that is the weighted majority of the experts’ predictions based on the weights  $w_1^{(t)}, \dots, w_n^{(t)}$ . 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  $(1 - \epsilon)$ :

$$w_i^{(t+1)} = (1 - \epsilon)w_i^{(t)} \quad (\text{update rule}). \tag{1}$$

**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  $i$ :*

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

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

*Remark:* When  $m_i^{(T)} \gg \frac{2 \ln n}{\epsilon}$  we see that the number of mistakes made by the algorithm is upperbounded by roughly  $2(1 + \epsilon)m_i^{(T)}$ , i.e., approximately

two times the number of mistakes made by the best expert. The factor of 2 can be removed by substituting the above deterministic algorithm by a randomized algorithm that predicts according to the majority opinion with probability proportional to its weight. (In other words, if the total weight of the experts saying “up” is  $3/4$  then the algorithm predicts “up” with probability  $3/4$  and “down” with probability  $1/4$ .) Then the number of mistakes after  $T$  steps is a random variable and the claimed upperbound holds for its *expectation* (see Section 2 for more details).

PROOF: A simple induction shows that  $w_i^{(t+1)} = (1 - \epsilon)^{m_i^{(t)}}$ . 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  $1 - \epsilon$ . Thus, the potential function decreases by a factor of at least  $(1 - \epsilon/2)$ :

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

Thus simple induction gives  $\Phi^{(T+1)} \leq n(1 - \epsilon/2)^{m^{(T)}}$ . 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  $-\ln(1 - \epsilon) \leq \epsilon + \epsilon^2$  since  $\epsilon < \frac{1}{2}$ .  $\square$

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 obvious 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 3.1.

## 2 Our generalization to the Weighted Majority 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.

Denote the set of events/outcomes by  $\mathbf{P}$ . We assume there is a matrix  $\mathbf{M}$  such that  $\mathbf{M}(i, j)$  is the *penalty* that expert  $i$  pays when the outcome is  $j \in \mathbf{P}$ . We will assume that for each expert  $i$  and each event  $j$ ,  $\mathbf{M}(i, j)$  is in the range  $[-1, 1]$ . This is the only assumption we make on the penalties.

The prediction algorithm will be *randomized*. In each round, we select a distribution over the set of experts, and select an expert randomly from it (and use his advised course of action). At this point, nature chooses an event from  $\mathbf{P}$ , and reveals the corresponding penalties of all the experts. We desire that the *expected penalty* is not much worse than that of the best expert in hindsight<sup>1</sup>.

The expected penalty for outcome  $j^{(t)} \in \mathbf{P}$  is

$$\sum_i p_i^{(t)} \mathbf{M}(i, j^{(t)}) = \sum_i w_i^{(t)} \mathbf{M}(i, j^{(t)}) / \sum_i w_i^{(t)},$$

which we denote by  $\mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})$ . By linearity of expectations, the expected total loss after  $T$  rounds is  $\sum_{t=1}^T \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})$ . The following theorem — completely analogous to Theorem 1 — bounds the expected total loss in terms of the loss of the best fixed expert in hindsight:

**Theorem 2 (Main)** *Let  $\epsilon \leq \frac{1}{2}$ . After  $T$  rounds, the Multiplicative Weights*

---

<sup>1</sup>Note that this setting generalizes the binary events setting of the weighted majority algorithm as follows: the penalty matrix  $\mathbf{M}$  has a row for each of the  $n$  experts and  $2^n$  columns, corresponding to the  $2^n$  possible penalty vectors in  $\{0, 1\}^n$ . For a prediction vector  $\langle x_1, x_2, \dots, x_n \rangle \in \{0, 1\}^n$  of the  $n$  experts, there are only 2 possible events: those corresponding to the penalty vectors  $\langle x_1, x_2, \dots, x_n \rangle$  and  $\langle 1 - x_1, 1 - x_2, \dots, 1 - x_n \rangle$  depending on whether the outcome is 0 or 1.

### Multiplicative Weights Update algorithm

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

**For**  $t = 1, 2, \dots, T$ :

1. Choose an expert from the distribution  $\mathcal{D}^{(t)} = \{p_1^{(t)}, p_2^{(t)}, \dots, p_n^{(t)}\}$  where  $p_i^{(t)} = w_i^{(t)} / \sum_k w_k^{(t)}$ , and incur the associated penalty.
2. Based on the outcome  $j^{(t)} \in \mathbf{P}$  in round  $t$ , at step  $t + 1$ , the weight of expert  $i$  is updated as follows for each  $i$ :

$$w_i^{(t+1)} = \begin{cases} w_i^{(t)}(1 - \epsilon)^{\mathbf{M}(i, j^{(t)})} & \text{if } \mathbf{M}(i, j^{(t)}) \geq 0 \\ w_i^{(t)}(1 + \epsilon)^{-\mathbf{M}(i, j^{(t)})} & \text{if } \mathbf{M}(i, j^{(t)}) < 0 \end{cases}$$

algorithm guarantees that for any expert  $i$ , we have

$$\sum_t \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)}) \leq (1 + \epsilon) \sum_{\geq 0} \mathbf{M}(i, j^{(t)}) + (1 - \epsilon) \sum_{< 0} \mathbf{M}(i, j^{(t)}) + \frac{\ln n}{\epsilon}$$

where the subscripts  $\geq 0$  and  $< 0$  refer to the rounds  $t$  where  $\mathbf{M}(i, j^{(t)})$  is  $\geq 0$  and  $< 0$  respectively.

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

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

The proof is along the lines of Theorem 1, using the potential function  $\Phi^{(t)} = \sum_i w_i^{(t)}$ . Since  $\mathbf{M}(i, j^{(t)}) \in [-1, 1]$ , using the facts above we have,

$$\begin{aligned} \Phi^{(t+1)} &= \sum_i w_i^{(t+1)} \\ &= \sum_{i: \mathbf{M}(i, j^{(t)}) \geq 0} w_i^{(t)}(1 - \epsilon)^{\mathbf{M}(i, j^{(t)})} + \sum_{i: \mathbf{M}(i, j^{(t)}) < 0} w_i^{(t)}(1 + \epsilon)^{-\mathbf{M}(i, j^{(t)})} \\ &\leq \sum_i w_i^{(t)}(1 - \epsilon \mathbf{M}(i, j^{(t)})) \\ &= \Phi^{(t)}(1 - \epsilon \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})) \\ &\leq \Phi^{(t)} e^{-\epsilon \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})}, \end{aligned}$$

where we used the fact that  $p_i^{(t)} = w_i^{(t)}/\Phi^{(t)}$  by definition. After  $T$  rounds, we have  $\Phi^{(T+1)} \leq \Phi^{(1)} e^{-\epsilon \sum_t \mathbf{M}(\mathcal{D}^{(t)}, x^{(t)})}$ . Furthermore, for every  $i$ ,

$$\Phi^{(T+1)} \geq w_i^{(T)} = (1 - \epsilon)^{\sum_{\geq 0} \mathbf{M}(i, j^{(t)})} \cdot (1 + \epsilon)^{-\sum_{< 0} \mathbf{M}(i, j^{(t)})}$$

Now we get the desired bound by taking logarithms and using  $\Phi^{(1)} = n$ , and simplifying as before. We used the facts that  $\ln(\frac{1}{1-\epsilon}) \leq \epsilon + \epsilon^2$  and  $\ln(1 + \epsilon) \geq \epsilon - \epsilon^2$  for  $\epsilon \leq \frac{1}{2}$ .  $\square$

REMARK: From the proof it is clear that the multiplicative update rule could also be

$$w_i^{(t+1)} = w_i^{(t)}(1 - \epsilon \mathbf{M}(i, j^{(t)}))$$

regardless of the sign of  $\mathbf{M}(i, j^{(t)})$ . Such a rule may be more practical to implement and is also used in the analysis of some algorithms such as SET COVER (c.f. section 3.4).

## 2.1 Bounded penalties

In many applications of the Multiplicative Weights algorithm, especially for solving linear programs, the range for the penalties is not  $[-1, 1]$ , rather, we have some parameters  $0 \leq \ell \leq \rho$  such that for any expert  $i$ , one of  $\mathbf{M}(i, j) \in [-\ell, \rho]$  or  $\mathbf{M}(i, j) \in [-\rho, \ell]$  holds for all  $j \in \mathbf{P}$ . We will call  $\rho$  the *width*<sup>2</sup>.

It is simple to apply the Multiplicative Weights algorithm to this situation: we just scale the penalties down by  $\rho$ , to get them in the range  $[-1, 1]$ , and apply the algorithm. The following theorem results immediately:

**Theorem 3 (Main)** *Let  $\epsilon \leq \frac{1}{2}$ . After  $T$  rounds, the Multiplicative Weights algorithm applied with penalties  $\mathbf{M}(i, j)/\rho$  for the experts guarantees that for any expert  $i$ , we have*

$$\sum_t \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)}) \leq (1 + \epsilon) \sum_{\geq 0} \mathbf{M}(i, j^{(t)}) + (1 - \epsilon) \sum_{< 0} \mathbf{M}(i, j^{(t)}) + \frac{\rho \ln n}{\epsilon}$$

where the subscripts  $\geq 0$  and  $< 0$  refer to the rounds  $t$  where  $\mathbf{M}(i, j^{(t)})$  is  $\geq 0$  and  $< 0$  respectively.

---

<sup>2</sup>The *width* parameter plays an important role in all previously cited papers. Following their convention, the width in our setting should be defined as  $\rho + \ell$ , which is a number in  $[\rho, 2\rho]$  and thus differs from our definition of width by at most a factor 2. Furthermore, in previous papers it was usually assumed that  $\ell = \rho$  but allowing these two to differ can be useful in some situations.

In several applications, the following corollaries are useful:

**Corollary 4** *Let  $\delta > 0$  be an error parameter. Then with  $\epsilon \leq \min\{\frac{\delta}{4\ell}, \frac{1}{2}\}$ , after  $T = \frac{2\rho \ln(n)}{\epsilon\delta}$  rounds, we have the following (additive and multiplicative) bound on the average expected loss: for any expert  $i$ ,*

$$\frac{\sum_t \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})}{T} \leq \delta + (1 \pm \epsilon) \frac{\sum_t \mathbf{M}(i, j^{(t)})}{T}$$

where the  $+$  or  $-$  sign depends on whether  $\mathbf{M}(i, j) \in [-\ell, \rho]$  or  $[-\rho, \ell]$  respectively.

PROOF: For concreteness, we will prove the case when  $\mathbf{M}(i, j) \in [-\ell, \rho]$ . The other case is similar. In this case,

$$(1 - \epsilon) \sum_{<0} \mathbf{M}(i, j^{(t)}) \leq (1 + \epsilon) \sum_{<0} \mathbf{M}(i, j^{(t)}) + 2\epsilon\ell T.$$

Substituting this bound in the inequality of Theorem 2 and dividing by  $T$  we have

$$\frac{\sum_t \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})}{T} \leq \frac{\rho \ln n}{\epsilon T} + 2\epsilon\ell + (1 + \epsilon) \frac{\sum_t \mathbf{M}(i, j^{(t)})}{T} \quad (2)$$

Choosing  $\epsilon$  and  $T$  as given, we get the required bound.  $\square$

REMARKS: (i) Again, we note that bound of Theorem 3 holds even if the events/outcomes could be picked by an adversary who knows our algorithm's current distribution on experts. (ii) In Corollary 4 two subcases need to be highlighted. When  $\ell = 0$  —i.e., all penalties are positive—then the running time is proportional to  $\rho/\epsilon\delta$ . When  $\ell = -\rho$ , then  $\epsilon \leq \delta/4\rho$ , and the running time is proportional to  $\rho^2/\delta^2$ . This issue is at the root of the difference between algorithms for general LP and for packing-covering LP problems (see Section 3.2).

**Corollary 5** *Let  $\delta > 0$  be an error parameter. Then with  $\epsilon = \min\{\frac{\delta}{4\rho}, \frac{1}{2}\}$ , after  $T = \frac{16\rho^2 \ln(n)}{\delta^2}$  rounds, we have the following (additive) bound on the average expected loss: for any expert  $i$ ,*

$$\frac{\sum_t \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})}{T} \leq \delta + \frac{\sum_t \mathbf{M}(i, j^{(t)})}{T}$$

PROOF: For concreteness, we will prove the case when  $\mathbf{M}(i, j) \in [-\ell, \rho]$ . The other case is similar. As in Corollary 4, we first derive inequality (2). We have  $(1 + \epsilon) \frac{\sum_t \mathbf{M}(i, j^{(t)})}{T} \leq \frac{\sum_t \mathbf{M}(i, j^{(t)})}{T} + \epsilon\rho$ , since for any  $j^{(t)}$ ,  $\mathbf{M}(i, j^{(t)}) \leq \rho$ . Finally, choosing  $\epsilon$  and  $T$  as given, we get the required bound.  $\square$

## 2.2 Gains instead of losses

In some situations, the entries of the matrix  $\mathbf{M}$  may specify *gains* instead of losses. Again, we assume that the gains are in the range  $[-1, 1]$ . Now our goal is to gain as much as possible in comparison to the gain of the best expert. We can get an algorithm for this case simply by considering the matrix  $\mathbf{M}' = -\mathbf{M}$  instead of  $\mathbf{M}$ .

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

**Theorem 6** *After  $T$  rounds, for any expert  $i$ , we have*

$$\sum_t \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)}) \geq (1 - \epsilon) \sum_{\geq 0} \mathbf{M}(i, j^{(t)}) + (1 + \epsilon) \sum_{< 0} \mathbf{M}(i, j^{(t)}) - \frac{\ln n}{\epsilon}$$

## 3 Applications

Now we describe how apply our meta-algorithm to a host of settings. First we describe the high-level intuition.

Usually we are interested in trying to find a suitable  $x \in \mathfrak{R}^n$  that satisfies some set of constraints. Each constraint corresponds to an “expert” in our earlier scenario, and we will use the Multiplicative Weights algorithm to update the distribution on experts. Each choice of the vector  $x$  corresponds to a possible “event”. The penalty of the expert on this event is made proportional to *how well* the corresponding constraint is satisfied by  $x$ . 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.

In many applications (though not all) the choice of event is also under our control. Typically we will need to generate the *maximally adversarial* event, i.e. the event  $x$  that maximizes the expected penalty, i.e. the weighted sum of penalties. Then the overall algorithm consists of two subprocedures: an “oracle” for generating the maximally adversarial event at each step, and the MW algorithm for updating the weights of the experts.

### 3.1 Solving zero-sum games approximately

We show how our general algorithm above can be used to approximately solve zero-sum games. (This is a duplication of the results of Freund and Schapire [FS99], who gave the same algorithm but a different proof of convergence that used KL-divergence. Furthermore, convergence of simple algorithms to zero-sum game equilibria were studied earlier in [Han57].) Let  $\mathbf{M}$  be the payoff matrix of a finite 2-player zero-sum game, so that when the row player plays strategy  $i$  and the column player plays strategy  $j$ , then the payoff to the column player is  $\mathbf{M}(i, j)$ . Assume that  $\mathbf{M}(i, j) \in [0, 1]$  for all  $i, j$ . We wish to approximately compute the game value, which according to von Neumann’s MinMax theorem is characterized as:

$$\lambda^* = \min_{\mathcal{D}} \max_j \mathbf{M}(\mathcal{D}, j) = \max_{\mathcal{P}} \min_i \mathbf{M}(i, \mathcal{P}), \quad (3)$$

where  $\mathcal{D}$  (resp.,  $\mathcal{P}$ ) varies over all distributions of rows (resp., columns) and  $j$  (resp.,  $i$ ) varies over all columns (resp., rows), and the notation  $\mathbf{M}(\mathcal{D}, j)$  denotes  $E_{i \in \mathcal{D}}[\mathbf{M}(i, j)]$ .

Let  $\delta > 0$  be an error parameter. We wish to approximately solve the zero-sum game up to additive error of  $\delta$ , namely, find mixed row and column strategies  $\mathcal{D}_{\text{final}}$  and  $\mathcal{P}_{\text{final}}$  such that

$$\lambda^* - \delta \leq \min_i \mathbf{M}(i, \mathcal{P}_{\text{final}}) \quad (4)$$

$$\max_j \mathbf{M}(\mathcal{D}_{\text{final}}, j) \leq \lambda^* + \delta. \quad (5)$$

We map our general algorithm from Section 2 to this setting by making the “experts” 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  $\mathbf{M}(i, j)$ . The algorithmic assumption about the game is that given any distribution  $\mathcal{D}$  on experts, we have an efficient way to pick the best event, namely, the pure column strategy  $j$  that maximizes  $\mathbf{M}(\mathcal{D}, j)$ . This quantity is at least  $\lambda^*$  from the definition above.

The penalties for experts (which are the same as payoffs) lie in  $[0, 1]$ . We use Corollary 5 with the parameters  $\ell = 0$  and  $\rho = 1$ , and we set  $\epsilon = \delta/4$ . We run the game for  $T = 16 \ln(n)/\delta^2$  as specified by Corollary 5.

Let  $j^{(1)}, j^{(2)}, \dots, j^{(T)}$  be the event sequence. Note that  $\mathbf{M}(\mathcal{D}^{(t)}, j^{(t)}) \geq \lambda^*$  since  $j^{(t)}$  is the best column response to distribution  $\mathcal{D}^{(t)}$ . Furthermore, Corollary 5 implies that the expected payoff of the MW algorithm is at

most  $\delta$  plus the payoff of the best fixed row strategy. Hence we have

$$\lambda^* \leq \frac{\sum_{t=1}^T \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})}{T} \leq \delta + \min_i \left\{ \frac{\sum_{t=1}^T \mathbf{M}(i, j^{(t)})}{T} \right\}. \quad (6)$$

Trivially this is upperbounded by  $\delta + \frac{\sum_{t=1}^T \mathbf{M}(\mathcal{D}, j^{(t)})}{T}$  for every row strategy  $\mathcal{D}$ . When  $\mathcal{D} = \mathcal{D}^*$  (the optimal row strategy) we have  $\mathbf{M}(\mathcal{D}, j) \leq \lambda^*$  for every  $j$ . The inequality (6) becomes:

$$\lambda^* \leq \frac{\sum_{t=1}^T \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})}{T} \leq \delta + \lambda^*$$

Thus,  $\frac{\sum_{t=1}^T \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})}{T}$  is an (additive)  $\delta$ -approximation to  $\lambda^*$ .

We set  $\mathcal{D}_{\text{final}}$  to be the distribution  $\mathcal{D}^{(t)}$  which has the minimum  $\mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})$  over all  $t$ . Let  $j_{\text{final}}$  be the optimal column player response to  $\mathcal{D}_{\text{final}}$ . We have, from (6),

$$\mathbf{M}(\mathcal{D}_{\text{final}}, j_{\text{final}}) \leq \frac{\sum_{t=1}^T \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)})}{T} \leq \lambda^* + \delta$$

Since for any  $t$ ,  $j^{(t)}$  maximizes  $\mathbf{M}(\mathcal{D}^{(t)}, j)$  over all  $j$ , we conclude that  $\mathcal{D}_{\text{final}}$  is an approximately optimal mixed row strategy<sup>3</sup>.

We set  $\mathcal{P}_{\text{final}}$ , the final column strategy, to be the distribution which assigns to column  $j$  probability equal to the fraction of times it was used during the game, namely  $\frac{|\{t: j^{(t)}=j\}|}{T}$ . From (6), we have, for any row distribution  $\mathcal{D}$ ,

$$\lambda^* - \delta \leq \frac{\sum_{t=1}^T \mathbf{M}(\mathcal{D}, j^{(t)})}{T} = \mathbf{M}(\mathcal{D}, \mathcal{P}_{\text{final}})$$

which shows that  $\mathcal{P}_{\text{final}}$  is an approximately optimal mixed column strategy.

## 3.2 Plotkin, Shmoys, Tardos framework for packing/covering LPs

Plotkin, Shmoys, and Tardos [PST91] 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

---

<sup>3</sup>Alternatively, we can set  $\mathcal{D}_{\text{final}} = \frac{1}{T} \sum_t \mathcal{D}^{(t)}$ . For let  $j_{\text{final}}$  be the optimal column player response to  $\mathcal{D}_{\text{final}}$ . Then we have  $\mathbf{M}(\mathcal{D}_{\text{final}}, j_{\text{final}}) = \frac{1}{T} \sum_t \mathbf{M}(\mathcal{D}^{(t)}, j_{\text{final}}) \leq \frac{1}{T} \sum_t \mathbf{M}(\mathcal{D}^{(t)}, j^{(t)}) \leq \lambda^* + \delta$

*Lagrangian relaxation* idea, and applies also to general linear programs. Below, we derive the algorithm for general LPs and then mention the slight modification that yields better running time for packing-covering LPs. For convenience we will discuss only covering LPs since packing LPs are treated similarly. Also, we note that we could derive this algorithm as a special case of game solving, but for concreteness we describe it explicitly.

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

$$Ax \geq b, \quad x \in P \quad (7)$$

where  $A$  is an  $m \times n$  matrix,  $x \in \mathbb{R}^n$ ,  $b \in \mathbb{R}^m$  and  $P$  is a convex set in  $\mathbb{R}^n$ . Intuitively, the set  $P$  represents the “easy” constraints to satisfy, such as non-negativity, and  $A$  represents the “hard” constraints to satisfy. We wish to design an algorithm that, given an error parameter  $\delta > 0$ , either yields an approximately feasible solution up to error  $\delta$ , i.e., an  $x \in P$  such that  $A_i x \geq b_i - \delta$  for some small  $\delta > 0$ , or failing that, proves that the system is infeasible. Here,  $A_i$  denotes the  $i^{\text{th}}$  row of  $A$ .

Plotkin, Shmoys and Tardos assume the existence of an oracle which solves the following feasibility problem:

$$\exists x \in P : \quad c^\top x \geq d \quad (8)$$

where  $c = \sum_i p_i A_i$  and  $d = \sum_i p_i b_i$  for some distribution  $p_1, p_2, \dots, p_m$ . It is reasonable to expect such an optimization procedure to exist (indeed, such is the case for many applications) since here we only need to check the feasibility of one constraint rather than  $m$ .

Using this oracle we can describe the required algorithm using the Multiplicative Weights algorithm. To map our general framework to this situation, we have an expert representing each of the  $m$  constraints. Events correspond to vectors  $x \in P$ . The penalty of the expert corresponding to constraint  $i$  for event  $x$  is  $A_i x - b_i$ . We assume that the oracle’s responses  $x$  satisfy  $A_i x - b_i \in [-\rho, \rho]$  for all  $i$ , for some parameter  $\rho$  known to our algorithm. Thus the penalties lie in  $[-\rho, \rho]$ . We run the Multiplicative Weights Update algorithm for  $T$  steps as in Corollary 5 using  $\epsilon = \delta/4\rho$ . (Note that  $T$  is proportional to  $\rho^2$ .) Note that if  $p_1, p_2, \dots, p_m$  is the distribution at any time, then we call the oracle with  $c = \sum_i p_i A_i$  and  $d = \sum_i p_i b_i$ .

We have the following cases:

**Case 1:** The oracle returns a feasible  $x$  for (8) in every iteration.

Then from Corollary 5, we have, for any  $i$

$$\frac{\sum_{t=1}^T \sum_j p_j^{(t)} [A_j x^{(t)} - b_j]}{T} \leq \delta + \frac{\sum_{t=1}^T [A_i x^{(t)} - b_i]}{T}$$

The LHS is  $\geq 0$  by assumption. Then we let  $\bar{x} = \sum_t x^{(t)}/T$  be the final answer, since the previous line implies that for every row  $i$ ,  $A_i \bar{x} \geq b_i - \delta$ . So we have an approximately feasible solution  $\bar{x}$ .

**Case 2:** In some iteration, the oracle declares infeasibility of (8).

In this case, we conclude that the original system is infeasible. This is correct because if there were a feasible solution  $x$ , then  $Ax \geq b$ , and so taking the linear combination of the inequalities given by the distribution  $p_1, p_2, \dots, p_m$  in the current iteration, we have  $\sum_i p_i A_i x \geq \sum_i p_i b_i$ , which means that the oracle incorrectly declared infeasibility.

**FRACTIONAL COVERING PROBLEMS:** The framework is the same as above, with the crucial difference that the coefficient matrix  $A$  is such that  $Ax \geq 0$  for all  $x \in P$ , and  $b > 0$ .

The algorithm will exploit this fact by using our earlier Remark that appeared after Corollary 4. We assume without loss of generality (by appropriately scaling the inequalities) that  $b_i = 1$  for all rows. Let  $\rho$  be a number (known to the algorithm in advance) such that for all  $x \in P$ ,  $A_i x \in [0, \rho]$ . Again, we have an expert corresponding to each of the  $m$  constraints and the events correspond to vectors  $x \in P$ . However, the penalty for the expert for constraint  $i$  for the event corresponding to vector  $x$  is  $A_i x$  instead of  $A_i x - b_i$  used above.

The rest of the analysis is unchanged, except the running time is now proportional to  $\rho$  instead of  $\rho^2$ .

**FRACTIONAL PACKING PROBLEMS:** Fractional packing problems are essentially the same as fractional covering ones, except that the inequalities are reversed. We can obtain algorithms to test feasibility of fractional packing problems in an exactly analogous way, the only difference being, we need to have the penalty matrix specify gains instead of losses as in section 2.2.

### 3.3 Approximating Multicommodity Flow Problems

Multicommodity flow problems are represented by packing/covering LPs and thus can be approximately solved using the PST framework outlined above. The resulting flow algorithm is outlined below together with a brief analysis. Unfortunately, the algorithm is not polynomial because its running time is a polynomial of the edge capacities (as opposed to the logarithm of the capacities, which is the number of bits needed to represent them) and thus the algorithm is not even polynomial-time. Garg and Könemann [GK98] fixed this problem with a better algorithm whose running time does not

depend upon the edge capacities.

Here we derive the Garg-Könemann algorithm using our general framework. This will highlight the essential new idea, namely, a reweighting of penalties to reduce the *width* parameter. Note that algorithm is not quite the same as in [GK98] (the termination condition is slightly different) and neither is the proof.

For illustrative purposes we focus on the *maximum multicommodity flow problem*, in which we are given a set of  $k$  source-sink pairs and capacities  $c_e$  on edges, and the objective is to maximize the total flow between these pairs. The LP formulation is as follows:

$$\begin{aligned} & \max \sum_p f_p \\ \forall e : & \sum_{p \ni e} f_p \leq c_e \end{aligned} \tag{9}$$

Here,  $f_p$  represents the flow on path  $p$  connecting the source-sink pairs for any of the commodities.

Before presenting the Garg-Könemann idea we first present the algorithm one would obtain by applying our packing-covering framework (Section 3.2) in the obvious way. The LP (9) is a packing LP, thus, we need to apply the Multiplicative Weights algorithm of Section 2.2.

First, note that by using binary search we can reduce the optimization problem to feasibility, by iteratively introducing a new constraint that gives a lower bound on the objective. So assume without loss of generality that we know the value  $F^{\text{opt}}$  of the total flow in the optimum solution. Then we want to check the feasibility of the system of inequalities,  $\forall e : \sum_{p \ni e} f_p \leq c_e$ , where the flows come from the polytope  $P = \{\sum_p f_p = F^{\text{opt}}\}$ . As outlined in Section 3.2, the obvious algorithm would maintain at each step  $t$  a weight  $w_e^{(t)}$  for each edge  $e$ . The optimization routine needed at each step is to find the flow in  $P$  which minimizes  $\sum_e w_e^{(t)} \sum_{p \ni e} f_p / c_e = \sum_p f_p \sum_{e \in p} w_e^{(t)} / c_e$ . This is minimized by a flow that is supported on a single path, namely, the shortest path  $p^{(t)}$  between a source-sink pair in the graph where edge  $e$  has length  $w_e^{(t)} / c_e$ . Thus an “event” corresponds to this path  $p^{(t)}$  and consists of passing a flow  $F^{\text{opt}}$  on this path. (Note that the final flow will be an average of the flows in each event, and hence will also have value  $F^{\text{opt}}$ .) Penalties for the experts/edges are defined as in Section 3.2.

Unfortunately the width parameter is  $\rho = \max_{\bar{f} \in P} \max_e \sum_{p \ni e} \bar{f}_p / c_e = F^{\text{opt}} / c_{\min}$  where  $c_{\min}$  is the capacity of the minimum capacity edge in the graph. Thus the algorithm requires  $T = \rho \ln(n) / \epsilon^2$  iterations. (As already

mentioned, this is not polynomial since  $\rho$  depends upon  $1/c_{\min}$  rather than the logarithm of this value.) The overall running time is  $\tilde{O}(F^{\text{opt}}T_{\text{sp}}/c_{\min})$  where  $T_{\text{sp}} \leq O(mk)$  is the time needed to compute  $k$  shortest paths.

Now we describe the Garg-Könemann modification. It continues to maintain weights  $w_e^{(t)}$  for every edge  $e$ , where initially,  $w_e^{(1)} = 1$  for all  $e$ . The events correspond to paths, as before. However, instead of routing the same flow  $F^{\text{opt}}$  at each time step, the event consists of routing only as much flows as is allowed by the minimum capacity edge on the path. In other words, the “event” at time  $t$  is a flow of value  $c_{p^{(t)}}$  on path  $p^{(t)}$ , where  $c_{p^{(t)}}$  is the minimum capacity of an edge on the path  $p^{(t)}$ . The penalty incurred by edge  $e$  is  $\mathbf{M}(e, p^{(t)}) = c_{p^{(t)}}/c_e$ . (In other words, a penalty of  $1/c_e$  per unit of flow passing through  $e$ .) *The width is therefore automatically upper bounded by 1.*

The Multiplicative Weights Update rule in this setting consists of updating the weights of all edges in path  $p^{(t)}$  and leaving other weights unchanged at that step:

$$\forall e \in p^{(t)} : w_e^{(t+1)} = w_e^{(t)}(1 + \epsilon)^{c_{p^{(t)}}/c_e}$$

The *termination rule* for the algorithm is to stop when as soon as for some edge  $e$ ,  $\frac{f_e}{c_e} \geq \frac{\ln m}{\epsilon^2}$ , where  $f_e$  is the total amount of flow routed by the algorithm on edge  $e$ .

### 3.3.1 Analysis

We apply Theorem 6. Since we have  $\mathbf{M}(e, p) \in [0, 1]$  for all edges  $e$  and paths  $p$ , we conclude that for any edge  $e$ , we have

$$\sum_{t=1}^T \mathbf{M}(\mathcal{D}^{(t)}, p^{(t)}) \geq (1 - \epsilon) \sum_{t=1}^T \mathbf{M}(e, p^{(t)}) - \frac{\ln(m)}{\epsilon}. \quad (10)$$

We now analyze both sides of this inequality. In round  $t$ , for any edge  $e$ , we have  $\mathbf{M}(e, p^{(t)}) = \frac{c_{p^{(t)}}}{c_e}$  if  $e \in p^{(t)}$ , and 0 if  $e \notin p$ . Thus, we have

$$\sum_{t=1}^T \mathbf{M}(e, p^{(t)}) = \frac{f_e}{c_e}, \quad (11)$$

where  $f_e$  is the total amount of flow on  $e$  at the end of the algorithm, and

$$\sum_{t=1}^T \mathbf{M}(\mathcal{D}^{(t)}, p^{(t)}) = \sum_{t=1}^T \frac{\sum_{e \in p^{(t)}} \frac{c_{p^{(t)}}}{c_e} \cdot w_e^{(t)}}{\sum_e w_e^{(t)}} = \sum_{t=1}^T c_{p^{(t)}} \cdot \frac{\sum_{e \in p^{(t)}} \frac{w_e^{(t)}}{c_e}}{\sum_e w_e^{(t)}}. \quad (12)$$

Now, suppose the optimum flow assigns  $f_p^{\text{opt}}$  flow to path  $p$ , and let  $F^{\text{opt}} = \sum_p f_p^{\text{opt}}$  be the total flow. For any set of edge lengths  $w_e/c_e$ , the shortest path  $p$  satisfies

$$\frac{\sum_e w_e}{\sum_{e \in p} \frac{w_e}{c_e}} \geq \frac{\sum_e w_e \cdot \sum_{p' \ni e} \frac{f_{p'}^{\text{opt}}}{c_e}}{\sum_{e \in p} \frac{w_e}{c_e}} = \frac{\sum_{p'} f_{p'}^{\text{opt}} \cdot \sum_{e \in p'} \frac{w_e}{c_e}}{\sum_{e \in p} \frac{w_e}{c_e}} \geq \sum_{p'} f_{p'}^{\text{opt}} = F^{\text{opt}}.$$

The first inequality follows because for any edge  $e$ , we have  $\sum_{p' \ni e} f_{p'}^{\text{opt}} \leq c_e$ . The second inequality follows from the fact that  $p$  is the shortest path with edge lengths given by  $w_e/c_e$ . Using this bound in (12), we get that

$$\sum_{t=1}^T \mathbf{M}(\mathcal{D}^{(t)}, p^{(t)}) \leq \sum_{t=1}^T \frac{c_{p^{(t)}}}{F^{\text{opt}}} = \frac{F}{F^{\text{opt}}}, \quad (13)$$

where  $F = \sum_{t=1}^T c_{p^{(t)}}$  is the total amount of flow passed by the algorithm.

Plugging (11) and (13) into (10), we get that

$$\frac{F}{F^{\text{opt}}} \geq (1 - \epsilon) \max_e \left\{ \frac{f_e}{c_e} \right\} - \frac{\ln(m)}{\epsilon}.$$

We stop the algorithm as soon as  $\max_e \left\{ \frac{f_e}{c_e} \right\} \geq \frac{\ln(m)}{\epsilon^2}$ , so that we get

$$\frac{F}{F^{\text{opt}}} \geq (1 - 2\epsilon) \max_e \left\{ \frac{f_e}{c_e} \right\}.$$

Now,  $C := \max_e \left\{ \frac{f_e}{c_e} \right\}$  is the maximum congestion of the flow passed by the algorithm. So, the flow scaled down by  $C$  respects all capacities. For this scaled down flow, we have that the total flow is

$$\frac{F}{C} \geq (1 - 2\epsilon) F^{\text{opt}},$$

which shows that the scaled-down flow is within  $(1 - 2\epsilon)$  of optimal.

**Running time.** In every iteration  $t$  of the algorithm, there is some edge  $e$  on the chosen path  $p^{(t)}$  that has minimum capacity. It gets congested by the flow of value  $c_{p^{(t)}} = c_e$  sent in that round. Since we stop the algorithm as soon as the congestion on any edge is at least  $\frac{\ln(m)}{\epsilon^2}$ , any given edge can be the minimum capacity edge on the chosen path at most  $\lceil \frac{\ln(m)}{\epsilon^2} \rceil$  times in the entire run of the algorithm. Since there are  $m$  edges, the number of iterations is therefore at most  $m \cdot \lceil \frac{\ln(m)}{\epsilon^2} \rceil = O\left(\frac{m \log(m)}{\epsilon^2}\right)$ .

Each iteration involves  $k$  shortest path computations. Recall that  $T_{\text{sp}}$  is the time needed for this. Thus, the overall running time is  $O\left(\frac{m \log m}{\epsilon^2} \cdot T_{\text{sp}}\right)$ .

### 3.4 $O(\log n)$ -approximation for many NP-hard problems

For many NP-hard problems, typically integer versions of packing-covering problems, one can compute a  $O(\log n)$ -approximation by solving the obvious LP relaxation and then using Raghavan-Thompson [RT87] randomized rounding. This yields a randomized algorithm; to obtain a deterministic algorithm, derandomize it using Raghavan’s [Rag86] method of *pessimistic estimators*.

Young [You95] has given an especially clear framework for understanding these algorithms which as a bonus also yields faster, combinatorial algorithms. He observes that one can collapse the three ideas in the algorithm above —LP solving, randomized rounding, derandomization— into a single algorithm that uses the multiplicative update rule, and does not need to solve the LP relaxation directly. (Young’s paper is titled “*Randomized rounding without solving the linear program.*”) At the root of Young’s algorithm is the observation that Raghavan’s pessimistic estimator is also an exponential potential function<sup>4</sup> and the approximation algorithm only needs to drive down this potential function at each step. This is easily achieved by a multiplicative update rule algorithm.

Below, we illustrate this idea using the canonical problem in this class, SET COVER. (A similar analysis works for other problems.) Since we have developed the multiplicative weights framework already, we do not go over Young’s original intuition involving Chernoff bound arguments and can proceed directly to the algorithm. In fact, the algorithm can be simplified so it becomes exactly the classical greedy algorithm, and we obtain a  $\ln n$ -approximation, which is best-possible for this problem (assuming reasonable complexity-theoretic conjectures [Fei98]).

In the SET COVER problem, we are given a universe of  $n$  elements, say  $U = \{1, 2, 3, \dots, n\}$  and a collection  $\mathcal{C}$  of subsets of  $U$  whose union equals  $U$ . We are required to pick the minimum number of sets from  $\mathcal{C}$  which cover all of  $U$ . Let this minimum number be denoted OPT. The Greedy Algorithm picks subsets iteratively, each time choosing that set which covers the maximum number of uncovered elements.

We analyze the Greedy Algorithm in our setup as follows. Since the elements of the universe represent the constraint that the union of sets picked by the algorithm must cover each of them, we let “experts” correspond to elements in the universe, and “events” correspond to the sets  $C_j \in \mathcal{C}$ . The penalty of the expert corresponding to element  $i$  for the event corresponding

---

<sup>4</sup>Recall that Raghavan’s algorithm is a derandomization of a Chernoff bound argument, and Chernoff bounds are derived using the exponential generating function  $e^{tX}$ .

to set  $C_j$  is  $\mathbf{M}(i, C_j) = 1$  or 0 depending on whether  $i \in C_j$  or not.

We run the Multiplicative Weights Update algorithm with this setup with  $\epsilon = 1$ . The update rule to be used is (see the remark following the proof of Theorem 2):

$$w_i^{(t+1)} = w_i^{(t)}(1 - \epsilon \mathbf{M}(i, C_j))$$

This update rule implies that elements that have been covered so far have weight 0 while all the rest have weight 1. The maximally adversarial event in this case is the set  $C_j$  which maximizes, given weights  $w_1, w_2, \dots, w_n$  and the corresponding distribution  $p_i = w_i / \sum_j w_j$ ,

$$\sum_i p_i \mathbf{M}(i, C_j) = \sum_{i \in C_j} p_i$$

which is simply the set which covers the maximum number of uncovered elements. Thus, this is the Greedy Set Cover algorithm.

Note that for any distribution  $p_1, p_2, \dots, p_n$  on the elements, we know that OPT sets cover all the weight, so one set must cover at least  $1/\text{OPT}$  fraction. So for the maximally adversarial event, we have  $\max_{C_j} \sum_{i \in C_j} p_i \geq 1/\text{OPT}$ .

Thus, the change in potential for each round is:

$$\Phi^{(t+1)} < \Phi^{(t)} e^{-\epsilon/\text{OPT}} = \Phi^{(t)} e^{-1/\text{OPT}}.$$

The strict inequality holds because we always get a strictly positive penalty. Thus, the potential drops by a factor of  $e^{-1/\text{OPT}}$  every time.

We run this as long as some element has not yet been covered. We show that  $T = \lceil \ln n \rceil \text{OPT}$  iterations suffice, which implies that we have a  $\lceil \ln n \rceil$  approximation to OPT. We have

$$\Phi^{(T+1)} < \Phi^{(1)} e^{-T/\text{OPT}} = n e^{-\lceil \ln n \rceil \text{OPT}/\text{OPT}} = n e^{-\lceil \ln n \rceil} \leq 1$$

Note that with  $\epsilon = 1$ ,  $\Phi^{(T+1)}$  is exactly the number of elements left uncovered after  $T$  iterations. So we conclude that all elements are covered.

### 3.5 Learning theory and boosting

Boosting [Sch90] —combining several moderately accurate rules-of-thumb into a singly highly accurate prediction rule— is a central idea of AI today. Freund and Schapire’s *AdaBoost* [FS97] uses the Multiplicative Weights Update Rule and fits in our framework. Here we explain the main idea using some simplifying assumptions.

Let  $X$  be some set (*domain*) and suppose we are trying to learn an unknown function (*concept*)  $c : X \rightarrow \{0, 1\}$  chosen from a concept class  $\mathcal{C}$ . Given a sequence of training examples  $(x, c(x))$  where  $x$  is generated from a fixed but unknown distribution  $\mathcal{D}$  on the domain  $X$ , the learning algorithm is required to output a *hypothesis*  $h : X \rightarrow \{0, 1\}$ . The *error* of the hypothesis is defined to be  $\mathbf{E}_{x \sim \mathcal{D}}[|h(x) - c(x)|]$ .

A *strong learning algorithm* is one that, for every distribution  $\mathcal{D}$  and every  $\epsilon, \delta > 0$ , outputs with probability  $1 - \delta$  a hypothesis whose error is at most  $\epsilon$ . A  $\gamma$ -*weak learning algorithm* for  $\gamma > 0$  is similar, except its error is as high as  $1/2 - \gamma$ . Boosting shows that if a  $\gamma$ -weak learning algorithm exists for a concept class, then a strong learning algorithm exists. (The running time of the algorithm and the number of samples may depend on  $\gamma$ .)

We prove this result in the so-called *boosting by sampling framework*, which uses a fixed training set of  $N$  examples drawn from the distribution  $\mathcal{D}$ . The goal is to make sure that the final hypothesis erroneously classifies at most  $\epsilon$  fraction of this training set. Using VC dimension theory —details omitted— this is sufficient to ensure (with probability  $1 - \delta$  over the choice of the sample) that the error of the hypothesis over the entire domain  $X$  (under distribution  $\mathcal{D}$ ) is at most  $2\epsilon$ .

The idea in boosting is to repeatedly run the weak learning algorithm on different distributions defined on the fixed training set. The final hypothesis has error  $\epsilon'$  under the *uniform* distribution on the training set. We use the Multiplicative Weights Update algorithm, but to avoid notational confusion, we use  $\alpha$  instead of  $\epsilon$  for the multiplicative update factors. The “experts” correspond to samples in the training set and “events” correspond to the set of all hypotheses that can be generated by the weak learning algorithm. If the event corresponding to the hypothesis  $h$  happens, the penalty for expert  $x$  is 1 or 0 depending on whether  $h(x) = c(x)$  or not. (Notice, we want the weight of an example to *increase* if the hypothesis labels it incorrectly.)

In each iteration, the algorithm presents the current distribution  $\mathcal{D}^{(t)}$  on the examples to the weak learning algorithm, and in return obtains a hypothesis  $h^{(t)}$  whose error with respect to the distribution  $\mathcal{D}^{(t)}$  is not more than  $1/2 - \gamma$ , in other words, the expected penalty,  $\mathbf{M}(\mathcal{D}^{(t)}, h^{(t)})$ , in each iteration is at least  $1/2 + \gamma$ . The algorithm is run for  $T$  rounds, where  $T$  will be specified shortly. The final hypothesis,  $h_{\text{final}}$ , labels  $x \in X$  according to the majority vote among  $h^{(1)}(x), h^{(2)}(x), \dots, h^{(T)}(x)$ .

Let  $S$  be the set of  $x \in X$  incorrectly labelled by  $h_{\text{final}}$ . The penalty for each  $x \in S$ ,  $\sum_t \mathbf{M}(x, h^{(t)})$  is at most  $T/2$  since the majority vote is incorrect for it. We adapt the proof of Theorem 2 in the following manner. We have,

as in the proof,

$$\Phi^{(T+1)} \leq \Phi^{(1)} e^{-\alpha \sum_t \mathbf{M}(\mathcal{D}^{(t)}, h^{(t)})} \leq n e^{-\alpha T(1/2+\gamma)}.$$

Now

$$\Phi^{(T+1)} \geq \sum_{x \in S} (1 - \alpha)^{\sum_t \mathbf{M}(x, h^{(t)})} \geq |S| (1 - \alpha)^{T/2} \geq |S| e^{-(\alpha + \alpha^2)(T/2)},$$

using the fact that  $(1 - x) \geq e^{-(x+x^2)}$  for  $x < 1/2$ . Thus, we conclude that the error of  $h_{\text{final}}$  on the training set under the uniform distribution is  $\frac{|S|}{n} \leq e^{-\alpha(\gamma-\alpha/2)T}$ . Choosing  $\alpha = \gamma$  and  $T = \frac{2}{\gamma^2} \ln \frac{1}{\epsilon'}$ , we get that the error is  $\leq \epsilon'$  as desired.

### 3.6 Hardcore sets and XOR Lemma

A function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  is  $\epsilon$ -strongly hard for circuits of size  $S$  if for every circuit  $C$  of size at most  $S$ ,

$$\Pr_x [C(x) = f(x)] \leq \frac{1}{2} + \epsilon.$$

It is  $\gamma$ -weakly hard on the other hand if

$$\Pr_x [C(x) = f(x)] \leq 1 - \gamma.$$

Yao's XOR Lemma [Yao82] shows that if  $f$  is  $\gamma$ -weakly hard against circuits of size  $S$  then  $f^{\oplus k} : \{0, 1\}^{nk} \rightarrow \{0, 1\}$  is  $\epsilon + (1 - \gamma)^k$ -strongly hard for circuits of size  $S\epsilon^2\gamma^2/8$ .

The original proofs were difficult but in the 1990s Impagliazzo [Imp95] suggested a simpler proof that as a byproduct proves an interesting fact about weakly hard functions: there is a reasonably large subset of inputs on which the function behaves like a strongly hard function! This subset is called a *hard core set*. Klivans and Servedio [KS03] observed that Impagliazzo's proof is a version of boosting. Phrased in our setting, experts correspond to inputs. At each step the algorithm maintains a distribution on experts. Events correspond to circuits of size  $S\epsilon^2\gamma^2/8$  that predict  $f$  with probability at least  $1 - \gamma$  on the current distribution. (We are simplifying a little; see [KS03] for details.) Recently, the third author has obtained an alternative, simpler proof of the existence of hard-core sets by directly applying the MW framework along with a projection trick [Kal07]. This method obtains the best known parameters in hard-core set constructions directly without having to recourse to composing two different boosting algorithms as in [KS03].

### 3.7 Connection to Chernoff bounds

*Chernoff bounds* show that the sum of bounded independent random variables  $X_1, X_2, \dots, X_n$  is sharply concentrated about its mean. They are proved by applying the standard Markov's inequality to the variable  $e^{t(\sum_i X_i)}$ . As pointed out in Young [You95], this technique has formal similarities to the multiplicative update rule. If we imagine the values of  $X_1, X_2, \dots, X_n$  being revealed sequentially, then the value of the “potential”  $e^{t(\sum_i X_i)}$  may be seen as being multiplied by  $e^{tX_i}$  at the  $i$ th step. Assuming  $tX_i \ll 1$  this multiplication factor is approximately  $(1 + tX_i)$ . As Young showed, this formal similarity can be used to design more efficient approximation algorithms for integer packing-cover problems where the original algorithms used randomized rounding (see Section 3.4).

### 3.8 Multiplicative update rules in network congestion control

The multiplicative update framework is usefully viewed as a *feedback mechanism* in several situations; e.g. *fictitious play* in economics that was discussed earlier. A similar view is also useful in congestion control in flow networks. For example, the congestion control in the classic TCP/IP protocol. Whenever clients detect that their packets are not getting through (implying that the network is too congested) they respond by drastically decreasing the rate at which they send packets. Conversely, when they detect that packets are getting through they increase their send rate. Below we analyze its convergence properties using our standard analysis.

We note that more advanced network protocols have also been designed, and sometimes analysed using a *utility* framework; see the survey [LPD02]. This multiparty analysis doesn't seem to fit into our multiplicative update framework, though it certainly seems related. Within theoretical computer science, a distributed flow algorithm of Awerbuch and Leighton [AL94] also uses a variant of the multiplicative update rule, but its analysis again does not exactly fit into our framework.

Here our goal is to analyse the convergence rate of the congestion control rule in TCP/IP: the *multiplicative decrease, additive increase* rule, which expects senders to additively increase their send rate until the network gets congested, and then multiplicatively decrease their send rate by a half. We show that this algorithm quickly converges to allocating equal bandwidth for all the senders; in fact, for  $n$  senders, convergence within error  $\epsilon$  occurs in  $\log(\frac{n}{\epsilon})$  steps of multiplicative decrease, additive increase.

This can be seen as follows. For convenience, let the total bandwidth be 1. Consider the time at which the network is at full capacity, and the senders reduce their transmit speeds by half. At this point, only 1/2 bandwidth is used. Observe that the subsequent additive increase part can be done away with; since all senders increase their transmit speeds at the same rate, all that is accomplished by the additive increase step after a multiplicative decrease step is to assign an extra bandwidth of  $\frac{1}{2n}$  to every sender so that the network is again full. Thus, if initially the sender has bandwidth  $x$ , then after  $T$  steps of multiplicative decrease, additive increase, the bandwidth for the sender is given by

$$\underbrace{\left( \dots \left( \left( x \cdot \frac{1}{2} + \frac{1}{2n} \right) \cdot \frac{1}{2} + \frac{1}{2n} \right) \dots \right) \cdot \frac{1}{2} + \frac{1}{2n}}_T = \frac{x}{2^T} + \frac{1}{2n} \left( 1 + \frac{1}{2} + \dots + \frac{1}{2^{T-1}} \right) \\ = \frac{\left( x - \frac{1}{n} \right)}{2^T} + \frac{1}{n}$$

After  $T \geq \log\left(\frac{n}{\epsilon}\right)$  steps, all senders have bandwidth within  $(1 \pm \epsilon)\frac{1}{n}$ .

### 3.9 Approximately Solving certain Semidefinite Programs

*Semidefinite programming* (SDP) is a special case of convex programming. A semidefinite program is derived from a linear program by imposing the additional constraint that some subset of  $n^2$  variables form an  $n \times n$  positive semidefinite matrix. Since the work of Goemans and Williamson [GW95], SDP has become an important tool in design of approximation algorithms for NP-hard optimization problems. Though this yields polynomial-time algorithms, their practicality is suspect because solving SDPs is a slow process. Therefore there is great interest in computing approximate solutions to SDPs, especially the types that arise in approximation algorithms. Since the SDPs in question are being used to design approximation algorithms anyway, it is permissible to compute approximate solutions to these SDPs.

Klein and Lu [KL96] use the multiplicative weights framework to derive a more efficient 0.878-approximation algorithm for MAX-CUT than the original SDP-based method in Goemans-Williamson [GW95]. The main idea in Klein-Lu is to approximately solve the MAX-CUT SDP. However, their idea does work very well for other SDPs. The main issue is that the width  $\rho$  (see 3.2) is too high for certain SDPs of interest.

The Klein-Lu approach was of limited uses in many cases because it does not do too well when the additive error  $\epsilon$  is required to be small. (They

were interested in the MAX-CUT problem, where this problem does not arise. The reason in a nutshell is that in a graph with  $m$  edges, the maximum cut has at least  $m/2$  edges, so it suffices to compute the optimum to an additive error  $\epsilon$  that is a fixed constant.) We have managed to extend the multiplicative weights framework to many of these settings to design efficient algorithms for SDP relaxations of many other problems. The main idea is to apply the Multiplicative Weights framework in a “nested” fashion: one can solve a constrained optimization problem by invoking the MW algorithm on an subset of constraints (the “outer” constraints) in the manner of 3.2, where the domain is now defined by the rest of the constraints (the “inner” constraints). The oracle can now be implemented by another application of the MW algorithm on the inner constraints. Alternatively, we can reduce the dependence on the width by using the observation that the Lagrangian relaxation problem on the inner constraints can be solved by the ellipsoid method. For details of this method, refer to our paper [AHK05]. For several families of SDPs we obtain the best running time known.

More recently Arora and Kale [AK07] have designed a new approach for solving SDPs that involves a variant of the multiplicative update rule at the matrix level; see Section 5 for details.

### 3.10 Approximating Graph Separators

A recent application of the multiplicative weights method is a combinatorial algorithm for approximating the SPARSEST CUT of graphs [AHK04]. This fundamental graph partitioning problem seeks the cut in the input graph of minimum expansion, viz. the number of edges crossing the cut divided by the size of the smaller side in the cut. This problem arises as a useful subroutine in many other algorithms, such as in divide-and-conquer algorithms for optimization problems on graphs, layout problems, clustering, etc. Furthermore, the expansion of a graph is a very useful way to quantify the connectivity of a graph and has many important applications in computer science.

The work of Arora, Rao and Vazirani [ARV] gave the first  $O(\sqrt{\log n})$  approximation algorithm to the SPARSEST CUT problem. However, their best algorithm relies on solving an SDP and runs in  $\tilde{O}(n^{4.5})$  time. They also gave an alternative algorithm based on the notion of *expander flows*, which are multicommodity flows in the graph whose demand graph has high expansion. However, their algorithm was based on the ellipsoid method, and was thus quite inefficient. In the paper [AHK04], we obtained a much more efficient algorithm for approximating the SPARSEST CUT problem

to an  $O(\sqrt{\log n})$  factor in  $\tilde{O}(n^2)$  time using the expander flow idea. The algorithm casts the problem of routing an expander flow in the graph as a linear program, and then checks the feasibility of the linear program using the techniques described in Section 3.2. The oracle for this purpose is implemented using a variety of techniques: the multicommodity flow algorithm of Garg and Könemann [GK98] (and its subsequent improvement by Fleischer [Fle00]), eigenvalue computations, and graph sparsification algorithms of Benczúr and Karger [BK96] based on random sampling.

### 3.11 Multiplicative weight algorithms in geometry

The multiplicative weight idea has been used several times in computational geometry. Chazelle [Cha00] (p. 6) describes the main idea, which is essentially the connection between derandomization of Chernoff bound arguments and the exponential potential function noted in Section 3.7.

The geometric applications consist of derandomizing the obvious randomized algorithm by using a deterministic construction of some kind of small set cover or low-discrepancy set. Formally, the analysis is similar to our analysis of the Set Cover algorithm in Section 3.4. Clarkson used this idea to give a deterministic algorithm for Linear Programming [Cla88]. Following Clarkson, Bronnimann and Goodrich use similar methods to find Set Covers for hyper-graphs with small VC dimension [BG94].

Recently, multiplicative weights arguments were also used in context of geometric embeddings of finite metric spaces. The approximation algorithm for sparsest cut in Arora et al. [ARV] involves a “Structure Theorem” about metric spaces of negative type. This theorem says that in an  $n$ -point metric space of negative type in which the average interpoint distance is  $\Omega(1)$ , there are two sets  $S, T$  of size  $\Omega(n)$  such that the distance between each  $i \in S, j \in T$  is  $\Omega(1/\sqrt{\log n})$ . Chawla et al. [CGR05] noted that this can be viewed as an embedding into  $\ell_1$  that preserves the “average” pair of distance  $\Omega(1)$  up to  $\sqrt{\log n}$  factor. They used the multiplicative update idea to construct an embedding (which can be viewed as a probability distribution on  $O(\log n)$  ARV-style embeddings) in which *every* distance of  $\Omega(1)$  distorts by at most  $O(\sqrt{\log n})$  factor. Using a similar idea for distance scales other than 1 and combining the resulting embeddings using the ideas of Krauthgamer et al. [KLMN04] they obtained an embedding of the negative type metric into  $\ell_1$  in which every internode distance distorts by at most a factor  $O(\log^{3/4} n)$ . Recently Arora et al. [ALN] gave a more complicated construction to improve the distortion bound to  $O(\sqrt{\log(n)} \log \log n)$ . As a consequence, they obtain a  $O(\sqrt{\log(n)} \log \log n)$ -approximation for non-uniform sparsest cut,

thus almost matching the  $O(\sqrt{\log n})$  bound for uniform sparsest cut from [ARV].

### 3.12 Online convex optimization

Online convex optimization [Zin03] is a very general framework encompassing many of the online problems discussed in the applications setting and many more. Here “online” means that the algorithm does not know the entire input at the start, and the input is presented to it in pieces over many rounds. In this section we describe the framework and the central role of the multiplicative weights method. Roughly speaking the story is that multiplicative weights method is the right analog in the “online” setting to the standard “gradient descent” in “offline” optimization. For a much more detailed treatment of online learning techniques see [CBL06].

In online convex optimization, in each round  $t = 1, 2, \dots$ , the online player picks a point  $x_t$  from a convex domain  $K \subseteq \mathbb{R}^n$ . A loss function  $f_t$  is presented, and the decision maker incurs a loss of  $f_t(x_t)$ . The goal of the online algorithm  $\mathcal{A}$  is to minimize loss compared to the best fixed offline strategy. This quantity is called **regret** in the game theory and machine learning literature.

$$\mathcal{R}_T(\mathcal{A}) = \sum_{t=1}^T f_t(x_t) - \min_{x^* \in K} \sum_{t=1}^T f_t(x^*)$$

Online convex optimization generalizes the standard expert setting by having the convex set  $K$  be the  $n$ -dimensional simplex (i.e. the set of all distributions over  $n$  “experts”), and having the payoff functions  $f_t$  be inner products with the appropriate column of the payoff matrix, i.e.  $f_t(x) = x^\top j_t$ , where  $x$  is the distribution over the experts, and  $j_t$  (with some abuse of notation) denotes the column vector of losses for the experts on event  $j_t$ . It also generalizes other online learning problems such as the portfolio management problem and online routing (see [Haz06] for more discussion on applications).

Below we describe how to extend the Multiplicative Weights Update algorithm to the online convex optimization framework, for the special case where the convex set is the  $n$  dimensional simplex. Zinkevich [Zin03] gives algorithms which apply to arbitrary convex sets with similar performance guarantees (although with worse dependence on the dimension, see [Haz06] for more details). We remark that recently more efficient algorithms for online convex optimization were found, which are not based on the Multiplicative Weights Update method [HAK07].

In order to apply the algorithm to the online convex optimization setting, simulate the standard Multiplicative Weights Update algorithm with the penalties defined as

$$\mathbf{M}(p, f_t) \triangleq p^\top \nabla f_t(p_t)$$

where  $p_t$  is the point played in round  $t$ , and  $\nabla f_t(p)$  is the gradient of the function  $f_t$  at point  $p$ . Let

$$\rho = \max_{p \in S} \max_{f_t} \|\nabla f_t(p)\|_\infty$$

Then the width of this game is  $\rho$ .

**Theorem 7** *After  $T$  rounds of applying the Multiplicative Weights Update algorithm to the online convex optimization framework, for any distribution on experts  $p^*$*

$$\frac{1}{T} \sum_t [f_t(p_t) - f(p^*)] \leq 4\rho \sqrt{\frac{\ln n}{T}}$$

PROOF: By the Taylor series:

$$f(y) = f(x) + (y - x)^\top \nabla f(x) + \frac{1}{2}(y - x)^\top \nabla^2 f(\zeta)(y - x)$$

Where  $\zeta$  is a point on the line connecting  $x$  and  $y$ ,  $\nabla f(x)$  is the gradient of  $f$  at point  $x$ , and  $\nabla^2 f(\zeta)$  is the Hessian of  $f$  at point  $\zeta$ . If  $f$  is convex, then  $\nabla^2 f(\zeta) \geq 0$  and therefore

$$f(x) - f(y) \leq (x - y)^\top \nabla f(x) \quad (14)$$

Next, reasoning as in Corollary 4, the inequality (2) with  $\ell = \rho$  gives, for any  $i$ ,

$$\frac{1}{T} \sum_t \mathbf{M}(p_t, f_t) \leq \frac{\rho \ln n}{T\varepsilon} + (1 + \varepsilon) \cdot \frac{1}{T} \sum_t \mathbf{M}(i, j^t) + 2\varepsilon\rho$$

Since the RHS holds for any  $i$ , we can replace  $\mathbf{M}(i, j^t)$  on the RHS by  $\mathbf{M}(p, j^t)$  for any distribution  $p$ , in particular,  $p = p^*$ . Now we have

$$\begin{aligned} \frac{1}{T} \sum_t [f_t(p_t) - f_t(p^*)] &\leq \frac{1}{T} \sum_{t=1}^T (p_t - p^*)^\top \nabla f_t(p_t) && \text{from (14)} \\ &= \frac{1}{T} \left[ \sum_t \mathbf{M}(p_t, f_t) - \mathbf{M}(p^*, f_t) \right] \\ &\leq \frac{\rho \ln n}{T\varepsilon} + 3\varepsilon\rho && \because \mathbf{M}(i, j^t) \in [-\rho, \rho] \end{aligned}$$

Choosing  $\varepsilon = \sqrt{\frac{\ln n}{T}}$  completes the proof.  
 $\square$

## 4 Lowerbounds

Can our analysis of the Multiplicative Weights Update algorithm be improved? This section shows that the answer is “No,” at least when the “width” parameter is not too large. Proving a lowerbound for larger values of width is an open problem.

Recall that our MW update algorithm specializes to a host of known algorithms, and therefore lowerbounds known for those specialized settings (e.g., Littlestone and Warmuth [LW94], Klein and Young [KY99], Freund and Shapire [FS99]) also carry over to our MW update algorithm. These lowerbounds show that at least  $\Omega(\frac{\rho \log n}{\varepsilon^2})$  iterations are required to obtain an  $\varepsilon$ -approximation. Now we sketch a better  $\Omega(\frac{\rho^2 \log n}{\varepsilon^2})$  lowerbound when the payoffs can be negative.

We prove the lowerbound in the expert-event prediction framework described in Section 2, where the experts are trying to maximize their gain. Events are revealed sequentially and remain unknown to the algorithm until revealed. The idea is to use random payoffs; this is similar to Klein and Young [KY99].

Let  $\mathbf{M} \in \{+1, -1\}^{n \times m}$  be the payoff matrix where there are  $n$  experts and  $m$  events. Define

$$V(\mathbf{M}) = \max_{\mathcal{D}} \min_{j \in [m]} \mathbf{M}(\mathcal{D}, j)$$

where the minimization is over all distributions  $\mathcal{D}$  on experts (note here that we are working with gains instead of losses, as in subsection 2.2). Let  $\mathcal{D}_{\mathbf{M}}^*$  be the distribution on the experts that achieves this payoff  $V(\mathbf{M})$ . Note that by definition of  $V(\mathbf{M})$ , for *every* distribution  $\mathcal{D}$  on the experts, there exists an event  $j$  for which the expected gain is at most  $V(\mathbf{M})$ .

The main technical theorem we prove is:

**Theorem 8** *For any  $n \in \mathbb{Z}^+$ , let  $p = \Omega(\frac{1}{n^{1/8}})$ ,  $m = \tilde{O}(n^{0.5})$ . Let  $T$  be any number  $\leq \frac{\log n}{p^2 \varepsilon^2}$ . Then there exists a payoff matrix  $\mathbf{M} \in \{\pm 1\}^{n \times m}$  for which  $V(\mathbf{M}) = \Omega(p)$  and the following holds:*

*For every subset of  $T$  events, there exists a distribution on the experts such that the minimal expected payoff over these  $T$  events is greater than  $V(\mathbf{M})(1 + \varepsilon)$ .*

Take the  $\mathbf{M}$  from Theorem 8, scaled by a factor of  $\frac{1}{2p}$ , and consider this new payoff matrix. Since all entries of  $\mathbf{M}$  were 1 in absolute value before the scaling, the width is bounded by  $\rho = O(\frac{1}{p})$ . We note here that the Multiplicative Weights Update algorithm takes  $T = O(\frac{\rho^2 \log n}{\epsilon^2})$  rounds to get a payoff within a factor of  $(1 + \epsilon)^{-1}$  of the payoff achieved by the best expert: this follows by using Corollary 4 with  $\delta = \frac{\epsilon}{2} \cdot \frac{1}{2p} V(\mathbf{M}) = \Omega(\epsilon)$  and using the fact that the best expert achieves an average payoff of at least  $\frac{1}{2p} V(\mathbf{M})$ .

The desired lowerbound is a simple corollary of Theorem 8. We show that if the number of rounds  $T = o(\frac{\rho^2 \log n}{\epsilon^2})$  then no prediction algorithm can achieve total payoff within a factor of  $(1 + \epsilon)^{-1}$  of the payoff achieved by the best expert. In each round, the adversary picks a new event such that the payoff for the current distribution on experts is at most  $\frac{1}{2p} V(\mathbf{M})$ . Thus after  $T$  rounds the algorithm's gain is at most  $\frac{1}{2p} V(\mathbf{M})T$ .

Now consider the submatrix of  $\mathbf{M}_-$  consisting of the  $T$  columns revealed. According to Theorem 8, there exists a distribution over the experts,  $\mathcal{D}_{\mathbf{M}_-}^*$ , that achieves an expected gain larger than  $\frac{1}{2p} V(\mathbf{M}) \cdot (1 + \epsilon)$  for each of the  $T$  events. Therefore, the gain of an offline algorithm which uses the fixed distribution  $\mathcal{D}_{\mathbf{M}_-}^*$  in every iteration is at least  $\frac{1}{2p} V(\mathbf{M})T(1 + \epsilon)$ . The gain of the best expert for these  $T$  events is only larger.

To finish, we prove Theorem 8. The proof shows that a random payoff matrix  $\mathbf{M} \in \{+1, -1\}^{n \times m}$  satisfies the required properties with high probability. Each entry of  $\mathbf{M}$  is a binomial random variable, chosen independently to be  $\pm 1$  with probabilities  $\frac{1}{2} + p, \frac{1}{2} - p$  correspondingly.

In order to prove the required property, we prove an upper bound on  $V(\mathbf{M})$  and a lower bound for  $V(\mathbf{B})$  for every payoff submatrix  $\mathbf{B} \subseteq \mathbf{M}$  of size  $n \times T$ . For both cases we use estimates for the tails of the binomial distribution. For the *upper bound* on  $V(\mathbf{M})$ , let us consider the uniform distribution on the experts. The payoffs for each event separately are tightly concentrated, and we use the Chernoff tail estimate to bound the deviations.

**Claim 1** *With probability at least  $1 - o(1)$ , we have  $V(\mathbf{M}) = O(2p(1 + \sqrt{\frac{\log m}{pn}}))$  and  $V(\mathbf{M}) = \Omega(p)$ .*

PROOF: Consider the uniform distribution on the experts. The expected gain for any event is  $\mu = 2p$ . Using the Chernoff bounds, we can bound the probability that any event  $j$  produces a total gain much larger than the expected

gain.

$$\Pr \left[ \frac{1}{n} \sum_j \mathbf{M}(i, j) - \mu \geq t\mu \right] \leq e^{-t^2\mu n}$$

Plugging in  $t = O(\sqrt{\frac{\ln m}{pn}})$  and using the union bound over all  $m$  events, we obtain that with very high probability all events occur a gain of at most  $2p(1+t)$ . The lower bound on  $V(\mathbf{M})$  is obtained similarly.  $\square$

Let us now proceed to bound  $V(\mathbf{B})$  from below for all submatrices  $\mathbf{B} \subseteq \mathbf{M}$  of size  $n \times T$ . The idea is again to observe the tail of the binomial distribution, but this time to bound the deviations from below (and the deviations will be larger then before since we consider a smaller number of random variables).

As for the parameters, we fix  $\epsilon$  later (think of it as  $O(p)$ ), and let  $T = \frac{\log n}{p^2\epsilon^2}$ . We assume  $T$  is small enough such that  $T^2 \log m = o(n)$  (this is consistent with the parameters of Theorem 8).

**Claim 2** *With probability at least  $1 - o(1)$  it holds for all  $\mathbf{B}$  that  $V(\mathbf{B}) \geq 2p(1 + \epsilon)$  for some  $\epsilon = \omega(\sqrt{\frac{\log m}{pn}})$ .*

PROOF: Consider certain expert  $k \in [n]$ . Denote by  $P_k$  the number of successful predictions of expert  $k$  on the events of  $\mathbf{B}$  (which is equal to the number of positive entries in row  $k$  of  $\mathbf{B}$ ). The expected value of  $P_k$  is naturally  $T(\frac{1}{2} + p)$ . Using a lower bound on the tail of the binomial distribution, which essentially matches the Chernoff upper bounds, we have

$$\Pr \left[ P_k \geq (1 + \nu)\left(\frac{1}{2} + p\right)T \right] \geq e^{-\Omega(T\nu^2)}$$

By our choice of  $T$  we have  $\epsilon = \frac{\sqrt{\log n}}{p\sqrt{T}}$ . Take  $\nu = \bar{\epsilon}p$  and set  $\bar{\epsilon} = \sqrt{\frac{\log(n/2r)}{p^2T}}$ , and the above probability becomes  $\frac{2r}{n}$ . Let  $r = T^2 \log m = o(n)$  (recall we assume that  $m, n$  are such that  $T^2 \log m = o(n)$ ).

Call any expert for which  $P_k \geq (1 + \nu)\left(\frac{1}{2} + p\right)T$  a *good* expert. Then by our choice of  $\bar{\epsilon}$ , the probability that a certain expert is good is at least  $\frac{2r}{n}$ , and every expert has independent performance. The expected number of good experts is  $2r$ . Because of the tight concentration around the mean, we can ensure, with high probability, that for every choice of  $\mathbf{B}$  there are  $r$  good experts. Specifically, the probability that there exists a submatrix  $\mathbf{B}$  that does not have at least  $r$  good experts is bounded by  $e^{-8r} \cdot \binom{m}{T}$ . By our

choice of  $r$  this probability is  $o(1)$ . So we assume that  $\mathbf{B}$  has at least  $r$  good experts.

We now lower bound  $V(\mathbf{B})$ . Let  $\mathbf{C}$  be the  $r \times T$  payoff sub-matrix of  $\mathbf{B}$  restricted to the  $r$  good experts of  $\mathbf{B}$ . Naturally  $V(\mathbf{C}) \leq V(\mathbf{B})$ , and hence it suffices to bound  $V(\mathbf{C})$  from below. For this, we use the von Neumann min-max theorem, which states:

$$V(\mathbf{C}) = \max_{\mathcal{D}} \min_{j \in [T]} \mathbf{C}(\mathcal{D}, j) = \min_{\mathcal{P}} \max_{i \in [r]} \mathbf{C}(i, \mathcal{P}) \quad (15)$$

Here,  $\mathcal{P}$  is a distribution on the events (i.e. columns) of  $\mathbf{C}$ , and  $\mathbf{C}(i, \mathcal{P}) = \sum_j \mathbf{C}(i, j) \mathcal{P}(j)$ .

For the role of  $\mathcal{P}$  in the equation above, consider the uniform distribution on the  $T$  events of  $\mathbf{C}$ . For any event of  $\mathbf{C}$ , the gain incurred by a random good expert is  $+1$  with probability  $\geq \frac{1}{2} + p + \frac{1}{2}\nu$  and  $-1$  with probability  $\leq \frac{1}{2} - p - \frac{1}{2}\nu$ , since all the experts of  $\mathbf{C}$  are good, and is independent from the other experts (according to the construction of  $\mathbf{M}$ ). The expected gain for any expert is thus at least  $(2p + \nu)r$ . Using the Chernoff bound for each event  $j$  separately, we have:

$$\Pr \left[ \left( \frac{1}{r} \sum_i \mathbf{C}(i, j) - (2p + \nu) \right) \geq \xi \right] \leq e^{-r\xi^2}$$

Set  $\xi = O(\sqrt{\frac{T \ln m}{r}}) = O(\sqrt{\frac{1}{T}}) = o(\nu)$ . This implies that

$$\Pr \left[ \frac{1}{r} \sum_i \mathbf{C}(i, j) \leq \Omega(2p + \nu) \right] \leq e^{-\Omega(T \log m)}$$

Taking the union bound over all events,  $\Pr[V(\mathbf{C}) \leq \Omega(2p + \nu)] \leq T e^{-\Omega(T \log m)}$ . And another union bound over all  $\binom{m}{T} \leq m^T$  sub matrices  $\mathbf{B}$  of  $\mathbf{M}$ , the probability that there are  $r$  good experts *and*  $V(\mathbf{B}) \geq V(\mathbf{C}) = \Omega(2p + \nu) = \Omega(2p(1 + \epsilon))$  is  $1 - o(1)$ .

In addition, notice that for sufficiently large  $m$ :

$$\epsilon = \Omega(\bar{\epsilon}) = \Omega \left( \sqrt{\frac{\log(n/2r)}{p^2 T}} \right) = \Omega \left( \sqrt{\frac{\log n}{p^2 T}} \right) = \omega \left( \sqrt{\frac{\log m}{pn}} \right).$$

□

Theorem 8 now follows from Claims 1 and 2.

## 5 The Matrix Multiplicative Weights algorithm

In the preceding sections, we considered an online decision making problem with experts. We refer to that setting as the *basic* or *scalar* case. Now we briefly consider a different online decision making problem with experts, which is seemingly quite different from the previous one, but has enough structure that we can obtain an analogous algorithm for it. We move from loss vectors to loss matrices, and from probability vectors to density matrices. For this reason, we refer to the current setting as the *matrix* case. We call the algorithm presented in this setting the *Matrix Multiplicative Weights algorithm*. The original motivation for our interest in this matrix setting is that it leads to a constructive version of SDP duality, just as the standard multiplicative weights algorithm can be viewed as a constructive version of LP duality. In fact the standard algorithm is a special subcase of the algorithm in this section, namely, when all the matrices involved are diagonal.

Applications of the matrix multiplicative weights algorithm include solving SDPs [AK07], derandomizing constructions of expander graphs, and obtaining bounds on the sample complexity for a learning problem in quantum computing. The details of these applications are beyond the scope of this survey; please see the third author's PhD thesis [Kal06] for details. The algorithm given here is from a paper of Arora and Kale [AK07] and a very similar algorithm was discovered slightly earlier by Warmuth and Kuzmin [WK06].

We stick with our basic experts scenario but now associate an expert with every unit vector  $\mathbf{v}$  in  $\mathbb{S}^{n-1}$ , the unit sphere in  $\mathbb{R}^n$ . As in the basic case, in every round, each expert recommends an action, and our task is to pick an expert  $\mathbf{v} \in \mathbb{S}^{n-1}$  and follow his suggested course of action. At this point, the losses of all actions recommended by the experts are revealed by nature. These losses are not arbitrary, but they are correlated in the following way. A *loss matrix*  $\mathbf{M} \in \mathbb{R}^{n \times n}$  is revealed, and the loss of an expert  $\mathbf{v}$  is then  $\mathbf{v}^\top \mathbf{M} \mathbf{v}$ . We assume that all these losses are either in the range  $[0, 1]$  or in  $[-1, 0]$ . Again, as in the basic case, this is the only assumption we make on the way nature chooses the costs; indeed, the costs could even be chosen adversarially. Equivalently, we assume that all the eigenvalues of the matrix  $\mathbf{M}$  are in either in  $[0, 1]$  or  $[-1, 0]$ .

This game is repeated over a number of rounds. Let  $t = 1, 2, \dots, T$  denote the current round. In each round  $t$ , we select a distribution  $\mathcal{D}^{(t)}$  over the set of experts  $\mathbb{S}^{n-1}$ , and select an expert  $\mathbf{v}$  randomly from it (and use his advised course of action). At this point, the losses of all the experts are revealed by nature in the form of the loss matrix  $\mathbf{M}^{(t)}$ . The expected cost

to the algorithm for choosing the distribution  $\mathcal{D}^{(t)}$  is

$$\mathbf{E}_{\mathbf{v} \in \mathcal{D}^{(t)}}[\mathbf{v}^\top \mathbf{M}^{(t)} \mathbf{v}] = \mathbf{E}_{\mathbf{v} \in \mathcal{D}^{(t)}}[\mathbf{M}^{(t)} \bullet \mathbf{v}\mathbf{v}^\top] = \mathbf{M}^{(t)} \bullet \mathbf{E}_{\mathbf{v} \in \mathcal{D}^{(t)}}[\mathbf{v}\mathbf{v}^\top].$$

Here, we use the notation  $\mathbf{A} \bullet \mathbf{B} = \sum_{ij} A_{ij} B_{ij}$  to denote the scalar product of two matrices thinking of them as vectors in  $\mathbb{R}^{n^2}$ . Define the matrix  $\mathbf{P}^{(t)} := \mathbf{E}_{\mathbf{v} \in \mathcal{D}^{(t)}}[\mathbf{v}\mathbf{v}^\top]$ . Note that  $\mathbf{P}^{(t)}$  is positive semidefinite: this is because it is a convex combination of the elementary positive semidefinite matrices  $\mathbf{v}\mathbf{v}^\top$ . Let  $\text{Tr}(\mathbf{A})$  denote the trace of a matrix  $\mathbf{A}$ . Then,  $\text{Tr}(\mathbf{P}^{(t)}) = 1$ , again because for all  $\mathbf{v}$ , we have  $\text{Tr}(\mathbf{v}\mathbf{v}^\top) = \|\mathbf{v}\|^2 = 1$ . A matrix  $\mathbf{P}$  which is positive semidefinite and has trace 1 is called a *density matrix*.

We will only be interested in the expected loss to the algorithm, and all the information required for computing the expected loss for a given distribution  $\mathcal{D}$  over  $\mathbb{S}^{n-1}$  is contained in the associated density matrix.

Thus, in each round  $t$ , we require our online algorithm to choose a density matrix  $\mathbf{P}^{(t)}$ , rather than a distribution  $\mathcal{D}^{(t)}$  over  $\mathbb{S}^{n-1}$  (the distribution is implicit in the choice of  $\mathbf{P}^{(t)}$ ). We then observe the loss matrix  $\mathbf{M}^{(t)}$  revealed by nature, and suffer the expected loss  $\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}$ . After  $T$  rounds, the total expected loss is  $\sum_{t=1}^T \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}$ , while the best fixed expert in hindsight corresponds to the unit vector  $\mathbf{v}$  which minimizes  $\sum_{t=1}^T \mathbf{v}^\top \mathbf{M}^{(t)} \mathbf{v}$ . Since we minimize this quantity over all unit vectors  $\mathbf{v}$ , the variational characterization of eigenvalues implies that this minimum loss is exactly  $\lambda_n(\sum_{t=1}^T \mathbf{M}^{(t)})$ . Our goal is to design an online algorithm whose total expected loss over the  $T$  rounds is not much more than the loss of the best expert. The following algorithm uses the notion of matrix exponential,  $\exp(\mathbf{A}) := \sum_{i=0}^{\infty} \frac{\mathbf{A}^i}{i!}$ .

### Matrix Multiplicative Weights algorithm

**Initialization:** Fix an  $\epsilon \leq \frac{1}{2}$ . Initialize the weight matrix  $\mathbf{W}^{(1)} = \mathbf{I}_n$ .

**For**  $t = 1, 2, \dots, T$ :

1. Use the density matrix  $\mathbf{P}^{(t)} = \frac{\mathbf{W}^{(t)}}{\text{Tr}(\mathbf{W}^{(t)})}$ .
2. Observe the loss matrix  $\mathbf{M}^{(t)}$ .
3. Update the weight matrix as follows:

$$\mathbf{W}^{(t+1)} = \exp(-\epsilon \sum_{\tau=1}^t \mathbf{M}^{(\tau)}).$$

Figure 1: The Matrix Multiplicative Weights algorithm.

The following theorem bounds the total expected loss of the Matrix Multiplicative Weights algorithm (given in Figure 1) in terms of the loss of the best fixed expert. This theorem is completely analogous to Theorem 2, and in fact, Theorem 2 can be obtained directly from this theorem in the case when all matrices involved are diagonal. We omit the proof of this theorem; again, it is along the lines of the proof of Theorem 2, but needs an additional inequality from statistical mechanics, the Golden-Thompson inequality. See [Kal06, AK07] for details.

**Theorem 9** *In the given setup, the Matrix Multiplicative Weights algorithm guarantees that after  $T$  rounds, for any expert  $\mathbf{v}$ , we have*

$$(1 - \epsilon) \sum_{\succeq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} + (1 + \epsilon) \sum_{\preceq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \leq \sum_{t=1}^T \mathbf{v}^\top \mathbf{M}^{(t)} \mathbf{v} + \frac{\ln n}{\epsilon}. \quad (16)$$

Here, the subscripts “ $\succeq \mathbf{0}$ ” and “ $\preceq \mathbf{0}$ ” in the summations are used to refer to the rounds  $t$  when  $\mathbf{M}^{(t)} \succeq \mathbf{0}$  and  $\mathbf{M}^{(t)} \preceq \mathbf{0}$  respectively.

## References

- [AHK04] Sanjeev Arora, Elad Hazan, and Satyen Kale.  $O(\sqrt{\log n})$  approximation to sparsest cut in  $\tilde{O}(n^2)$  time. In *45th IEEE FOCS*, pages 238–247, 2004.
- [AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *46th IEEE FOCS*, pages 339–348, 2005.
- [AK07] S. Arora and S. Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, 2007.
- [AL94] Baruch Awerbuch and Tom Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *26th ACM STOC*, pages 487–496, 1994.
- [ALN] Sanjeev Arora, James Lee, and Assaf Naor. Euclidean distortion and the sparsest cut. In *ACM STOC 2005*.
- [ARV] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings, and graph partitioning. In *ACM STOC 2004*, pages 222–231.

- [BG94] H. Bronnimann and M.T. Goodrich. Almost optimal set covers in finite vc-dimension. In *Proc. 10th ACM Symp. on Computational Geometry (SoCG)*, pages 293–302, 1994.
- [BK96] A. A. Benczúr and D. R. Karger. Approximating  $s-t$  minimum cuts in  $\tilde{O}(n^2)$  time. In *ACM STOC*, pages 47–55, 1996.
- [Blu98] A. Blum. On-line algorithms in machine learning. In A. Fiat and G. Woeginger, editors, *Online Algorithms: The State of the Art*, pages 306–325. LNCS 1442, 1998.
- [Bro51] G. W. Brown. *Analysis of Production and Allocation*, T. C. Koopmans, ed., chapter Iterative solution of games by fictitious play, pages 374–376. Wiley, 1951.
- [BvN50] G.W. Brown and J. von Neumann. Solutions of games by differential equations. *Annals of Mathematical Studies*, 24:73–79, 1950.
- [CBL06] Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.
- [CGR05] S. Chawla, A. Gupta, and H. Racke. Embeddings of negative-type metrics and an improved approximation to generalized sparsest cut. In *16th ACM SODA*, 2005.
- [Cha00] B. Chazelle. *The Discrepancy Method — Randomness and Complexity*. Cambridge University Press, Cambridge, 2000.
- [Cla88] K. Clarkson. A las vegas algorithm for linear programming when the dimension is small. In *29th FOCS*, pages 452–456, 1988.
- [Cov96] Thomas M. Cover. Universal data compression and portfolio selection. In *37th FOCS*, pages 534–538, 1996.
- [CW89] B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete & Computational Geometry*, 4, 1989.
- [Fei98] Uriel Feige. A threshold of  $\ln n$  for approximating set cover. *Journal of the ACM*, 45(4):634–652, July 1998.
- [Fle00] Lisa K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discret. Math.*, 13(4):505–520, 2000.

- [FS97] Yoav Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.
- [FS99] Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.
- [FV99] D.P. Foster and R. Vohra. Regret in the on-line decision problem. *Games and Economic Behaviour*, 29:7–35, 1999.
- [GK95] M. Grigoriadis and L. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. In *Operations Research Letters*, volume 18, pages 53–58, 1995.
- [GK98] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *39th FOCS*, pages 300–309, 1998.
- [GK04] Naveen Garg and Rohit Khandekar. Fractional covering with upper bounds on the variables: Solving lps with negative entries. In Susanne Albers and Tomasz Radzik, editors, *ESA*, volume 3221 of *Lecture Notes in Computer Science*, pages 371–382. Springer, 2004.
- [GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.
- [HAK07] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. In *Machine Learning*, volume 69(2-3), pages 169–192, 2007.
- [Han57] J. Hannan. Approximation to bayes risk in repeated plays. In M. Dresher, A Tucker, and P. Wolfe, editors, *Contributaions to the Theory of Games*, volume 3, pages 97–139. Princeton University Press, 1957.
- [Haz06] Elad Hazan. *Efficient Algorithms for Online Convex Optimization and their Applications*. PhD thesis, Princeton University, 2006. Technical Report TR-766-06.

- [Imp95] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *36th FOCS*, pages 538–545, 1995.
- [Kal06] Satyen Kale. *Efficient Algorithms Using The Multiplicative Weights Update Method*. PhD thesis, Princeton University, 2006. Technical Report TR-804-07.
- [Kal07] Satyen Kale. Boosting and hard-core set constructions: a simplified approach. *ECCC Report TR07-131*, 2007.
- [Kha04] Rohit Khandekar. *Lagrangian Relaxation based Algorithms for Convex Programming Problems*. PhD thesis, Indian Institute of Technology, Delhi, 2004. Available at <http://www.cse.iitd.ernet.in/~rohitk>.
- [KL96] Philip Klein and Hsueh-I. Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *28th ACM STOC*, pages 338–347, 1996.
- [KLMN04] R. Krauthgamer, J. Lee, M. Mendel, and A. Naor. Measured descent: A new embedding method for finite metrics. In *FOCS*, 2004.
- [KS03] Adam R. Klivans and Rocco A. Servedio. Boosting and hard-core set construction. *Machine Learning*, 51(3):217–238, 2003.
- [KV03] Adam Kalai and Santosh Vempala. Efficient algorithms for on-line decision problems. In *16th COLT*, pages 26–40, 2003.
- [KY99] P. N. Klein and N. E. Young. On the number of iterations for dantzig-wolfe optimization and packing-covering approximation algorithms. In *IPCO*, pages 320–327, 1999.
- [Lit87] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1987.
- [Lit89] N. Littlestone. *Mistake bounds and logarithmic linear-threshold learning algorithms*. PhD thesis, University of California at Santa Cruz, Santa Cruz, CA, USA, 1989.
- [LPD02] S. H. Low, F. Paganini, and J. C. Doyle. Internet congestion control. *IEEE Control Systems Magazine*, 2002.

- [LW94] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1 February 1994.
- [MP69] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1969.
- [PST91] Serge A. Plotkin, David B. Shmoys, and Tardos Tardos. Fast approximation algorithm for fractional packing and covering problems. In *32nd FOCS*, pages 495–504. IEEE Computer Society Press, 1991.
- [Rag86] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. In *27th FOCS*, pages 10–18. IEEE, 1986.
- [Rob51] J. Robinson. An iterative method of solving a game. *Ann. Math.*, 54:296–301, 1951.
- [RT87] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.
- [Sch90] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.
- [WK06] M. Warmuth and D. Kuzmin. Online variance minimization. In *Proceedings of the 19th Annual Conference on Learning Theory (COLT 06)*, 2006.
- [Yao82] Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd FOCS*, pages 80–91. IEEE, 3–5 November 1982.
- [You95] Neal E. Young. Randomized rounding without solving the linear program. In *6th SODA*, pages 170–178, 1995.
- [Zin03] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *20th ICML*, pages 928–936, 2003.