THEORY OF COMPUTING, Volume 8 (2012), pp. 121–164
www.theoryofcomputing.org

RESEARCH SURVEY

The Multiplicative Weights Update Method: A Meta-Algorithm and Applications

Sanjeev Arora* Elad Hazan Satyen Kale

Received: July 22, 2008; revised: July 2, 2011; published: May 1, 2012.

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 many of these disparate algorithms and derives 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."

ACM Classification: G.1.6

AMS Classification: 68Q25

Key words and phrases: algorithms, game theory, machine learning

1 Introduction

The *Multiplicative Weights (MW) method* is a simple idea which has been repeatedly discovered in fields as diverse as Machine Learning, Optimization, and Game Theory. The setting for this algorithm is the following. A decision maker has a choice of n decisions, and needs to repeatedly make a decision and obtain an associated payoff. The decision maker's goal, in the long run, is to achieve a total payoff which is comparable to the payoff of that fixed decision that maximizes the total payoff with the benefit of

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

 DOI: 10.4086/toc.2012.v008a006

hindsight. While this best decision may not be known *a priori*, it is still possible to achieve this goal by maintaining weights on the decisions, and choosing the decisions randomly with probability proportional to the weights. In each successive round, the weights are updated by multiplying them with factors which depend on the payoff of the associated decision in that round. Intuitively, this scheme works because it tends to focus higher weight on higher payoff decisions in the long run.

This idea lies at the core of a variety of algorithms. Some examples include: the Ada Boost algorithm in machine learning [26]; algorithms for game playing studied in economics (see references later), the Plotkin-Shmoys-Tardos algorithm for packing and covering LPs [56], and its improvements in the case of flow problems by Young [65], Garg-Könemann [29, 30], Fleischer [24] and others; methods for convex optimization like exponentiated gradient (mirror descent), Lagrangian multipliers, and subgradient methods, Impagliazzo's proof of the Yao XOR lemma [40], etc. The analysis of the running time uses a potential function argument and the final running time is proportional to $1/\varepsilon^2$.

It has been clear to several researchers that these results are very similar. For example Khandekar's Ph. D. thesis [46] makes this point about the varied applications of this idea to convex optimization. The purpose of this survey is to clarify that many of these applications are instances of the same, more general algorithm (although several specialized applications, such as [53], require additional technical work). This meta-algorithm is very similar to the "Hedge" algorithm from learning theory [26]. Similar algorithms have 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 [29, 30] were felt to be quite different from those of Plotkin-Shmoys-Tardos [56]. In our framework, they can be seen as a clever trick for "width reduction" for the Plotkin-Shmoys-Tardos algorithms (see Section 3.4).

We feel that this 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 multiplicative weights update rule may be seen as a "constructive" version of LP duality—equivalently, von Neumann's minimax theorem in game theory—and it gives a fairly concrete method for competing players to arrive at a solution/equilibrium (see Section 3.2). This may be an appealing feature in introductory algorithms courses, since the standard algorithms for LP such as simplex, ellipsoid, or interior point 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 seem easier and cleaner than the entropy-based proofs for the same results in machine learning (although the proof technique we use here has been used before, see for example Blum's survey [10]).

The current paper is chiefly a survey. It introduces the main algorithm, gives a few variants (mostly having to do with the range in which the payoffs lie), and surveys the most important applications—often with complete proofs. Note however that this survey does not cover *all* applications of the technique, as several of these require considerable additional technical work which is beyond the scope of this paper. We have provided pointers to some such applications which use the multiplicative weights technique at their core without going into more details. There are also a few small results that appear to be new, such as the variant of the Garg-Könemann algorithm in Section 3.4 and the lower bound in Section 4.

Related work. An algorithm similar in flavor to the Multiplicative Weights algorithm was proposed in game theory in the early 1950's [13, 12, 59]. Following Brown [12], 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 the pure best response 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 numerous developments in economics, including Arrow-Debreu General Equilibrium theory and more recently, evolutionary game theory. Grigoriadis and Khachiyan [33] 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 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 [50, 51]. It is somewhat reminiscent of the older *perceptron* learning algorithm of Minsky and Papert [55]. The Winnow algorithm was generalized by Littlestone and Warmuth [52] in the form of the Weighted Majority algorithm, and later by Freund and Schapire in the form of the Hedge algorithm [26]. 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*, or entropy, used in those papers. 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 (although the entropy based approach yields somewhat tighter bounds that are useful in some applications, see Section 2.2).

Other applications of the multiplicative weights algorithm in computational geometry include Clarkson's algorithm for linear programming with a bounded number of variables in linear time [20, 21]. Following Clarkson, Brönnimann and Goodrich use similar methods to find Set Covers for hypergraphs with small VC dimension [11].

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 [22], Foster and Vohra [25], and also Blum [10] 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 1950's [34]. Kalai and Vempala showed how to derive efficient algorithms via methods similar to Hannan's [43]. We show how Hannan's algorithm with the appropriate choice of parameters yields the multiplicative update decision rule in Section 3.8.

Within computer science, several researchers have previously noted the close relationships between multiplicative update algorithms used in different contexts. Young [65] notes the connection between fast LP algorithms and Raghavan's method of pessimistic estimators for derandomization of randomized rounding algorithms; see our Section 3.5. Klivans and Servedio [49] relate boosting algorithms in learning theory to proofs of Yao's XOR Lemma; see our Section 3.6. Garg and Khandekar [28] 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 [65] and Freund-Schapire [27]) on the development of our framework. We emphasize again that we do not claim that *every* algorithm designed using the multiplicative update idea fits in our framework, just that most do.

Paper organization. We proceed to define the illustrative weighted majority algorithm in this section. In Section 2 we describe the general MW meta-algorithm, followed by numerous and varied applications in Section 3. In Section 4 we give lower bounds, followed by the more general matrix MW algorithm in Section 5.

1.1 The weighted majority algorithm

Now we briefly illustrate the weighted majority algorithm in a simple and concrete setting, which will naturally lead to our generalized meta-algorithm. This is known as the *Prediction from Expert Advice* problem.

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, and if it's correct, we lose nothing.

The stock movements can be *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 they 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.

Theorem 1.1. *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:*

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

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

Weighted majority algorithm

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

1. Make the prediction that is the weighted majority of the experts' predictions based on the weights $w_1^{(t)}, \ldots, w_n^{(t)}$. That is, predict "up" or "down" depending on which prediction has a higher total weight of experts advising it (breaking ties arbitrarily).

2. For every expert i who predicts wrongly, decrease his weight for the next round by multiplying it by a factor of $(1-\eta)$:

$$w_i^{(t+1)} = (1-\eta) w_i^{(t)} \qquad \text{(update rule)}. \tag{1.1}$$

Remark When $m_i^{(T)} \gg (2/\eta)\ln n$ we see that the number of mistakes made by the algorithm is bounded from above by roughly $2(1+\eta)m_i^{(T)}$, i. e., approximately twice the number of mistakes made by the best expert. This is tight for any deterministic algorithm. However, 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 upper bound holds for its *expectation* (see Section 2 for more details).

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

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

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

$$-\ln(1-\eta) \leq \eta + \eta^2$$

since $\eta < 1/2$. □

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 3.2.

2 The Multiplicative Weights algorithm

In the general setting, we have a set of *n decisions* and in each round, we are required to select one decision from the set. In each round, each decision incurs a certain *cost*, determined by nature or an adversary. All the costs are revealed *after* we choose our decision, and we incur the cost of the decision we chose. For example, in the prediction from expert advice problem, each decision corresponds to a choice of an expert, and the cost of an expert is 1 if the expert makes a mistake, and 0 otherwise.

To motivate the Multiplicative Weights (MW) algorithm, consider the naïve strategy that, in each iteration, simply picks a decision at random. The expected penalty will be that of the "average" decision. Suppose now that a few decisions are clearly better in the long run. This is easy to spot as the costs are revealed over time, 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 decisions at the outset, we select them uniformly at random. This maximum entropy starting rule reflects our ignorance. As we learn which ones are the good decisions and which ones are bad, we lower the entropy to reflect our increased knowledge. The multiplicative weight update is our means of skewing the distribution.

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

The expected cost to the algorithm for sampling a decision i from the distribution $\mathbf{p}^{(t)}$ is

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

The total expected cost over all rounds is therefore $\sum_{t=1}^{T}\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)}$. Just as before, our goal is to design an algorithm which achieves a total expected cost not too much more than the cost of the best decision in hindsight, viz. $\min_i \sum_{t=1}^{T} m_i{}^{(t)}$. Consider the following algorithm, which we call the Multiplicative Weights Algorithm. This algorithm has been studied before as the `prod` algorithm of Cesa-Bianchi, Mansour, and Stoltz [17], and Theorem 2.1 can be seen to follow from Lemma 2 in [17].

The following theorem—completely analogous to Theorem 1.1—bounds the total expected cost of the Multiplicative Weights algorithm (given in Figure 1) in terms of the total cost of the best decision:

Theorem 2.1. *Assume that all costs $m_i{}^{(t)} \in [-1,1]$ and $\eta \le 1/2$. Then the Multiplicative Weights algorithm guarantees that after T rounds, for any decision i, we have*

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

Multiplicative Weights algorithm

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

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

$$ {w_i}^{(t+1)} = {w_i}^{(t)}(1 - \eta {m_i}^{(t)}) \tag{2.1} $$

Figure 1: The Multiplicative Weights algorithm.

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

$$ \begin{aligned} \Phi^{(t+1)} &= \sum_i {w_i}^{(t+1)} \\ &= \sum_i {w_i}^{(t)}(1 - \eta {m_i}^{(t)}) \\ &= \Phi^{(t)} - \eta \Phi^{(t)} \sum_i {m_i}^{(t)} {p_i}^{(t)} \\ &= \Phi^{(t)}(1 - \eta \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}) \\ &\leq \Phi^{(t)} \exp(-\eta \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}) \,. \end{aligned} $$

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

$$ \Phi^{(T+1)} \leq \Phi^{(1)} \exp\left(-\eta \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}\right) = n \cdot \exp\left(-\eta \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}\right). \tag{2.2} $$

Next we use the following facts, which follow immediately from the convexity of the exponential function:

$$ \begin{aligned} (1-\eta)^x &\leq (1 - \eta x) \quad \text{if } x \in [0,1]\,, \\ (1+\eta)^{-x} &\leq (1 - \eta x) \quad \text{if } x \in [-1,0]\,. \end{aligned} \tag{2.3} $$

Since ${m_i}^{(t)} \in [-1,1]$, we have for every decision i,

$$ \Phi^{(T+1)} \geq {w_i}^{(T+1)} = \prod_{t \leq T} (1 - \eta {m_i}^{(t)}) \geq (1-\eta)^{\sum_{\geq 0} {m_i}^{(t)}} \cdot (1+\eta)^{-\sum_{<0} {m_i}^{(t)}}, \tag{2.4} $$

where the subscripts "≥ 0" and "< 0" in the summations refer to the rounds t where ${m_i}^{(t)}$ is ≥ 0 and < 0 respectively. Taking logarithms in equations (2.2) and (2.4) we get:

$$\ln n - \eta \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \geq \sum_{\geq 0} {m_i}^{(t)} \ln(1-\eta) - \sum_{<0} {m_i}^{(t)} \ln(1+\eta)\,.$$

Negating, rearranging, and scaling by $1/\eta$:

$$\begin{aligned}
\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} &\leq \frac{\ln n}{\eta} + \frac{1}{\eta}\sum_{\geq 0} {m_i}^{(t)} \ln \frac{1}{1-\eta} + \frac{1}{\eta}\sum_{<0} {m_i}^{(t)} \ln(1+\eta) \\
&\leq \frac{\ln n}{\eta} + \frac{1}{\eta}\sum_{\geq 0} {m_i}^{(t)} (\eta+\eta^2) + \frac{1}{\eta}\sum_{<0} {m_i}^{(t)} (\eta-\eta^2) \\
&= \frac{\ln n}{\eta} + \sum_{t=1}^{T} {m_i}^{(t)} + \eta \sum_{\geq 0} {m_i}^{(t)} - \eta \sum_{<0} {m_i}^{(t)} \\
&= \frac{\ln n}{\eta} + \sum_{t=1}^{T} {m_i}^{(t)} + \eta \sum_{t=1}^{T} |{m_i}^{(t)}|\,.
\end{aligned}$$

In the second inequality we used the facts that

$$\ln\left(\frac{1}{1-\eta}\right) \leq \eta + \eta^2 \quad \text{and} \quad \ln(1+\eta) \geq \eta - \eta^2 \tag{2.5}$$

for $\eta \leq 1/2$. □

Corollary 2.2. *The Multiplicative Weights algorithm also guarantees that after T rounds, for any distribution* $\mathbf{p}$ *on the decisions,*

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

where $|\mathbf{m}^{(t)}|$ is the vector obtained by the taking the coordinate-wise absolute value of $\mathbf{m}^{(t)}$.

Proof. This corollary follows immediately from Theorem 2.1, by taking a convex combination of the inequalities for all decisions i with the distribution $\mathbf{p}$. □

2.1 Updating with exponential factors: the Hedge algorithm

In our description of the MW algorithm, the update rule uses multiplication by a linear function of the cost (specifically, $(1-\eta {m_i}^{(t)})$ for expert i). In several other incarnations of MW algorithm, notably the Hedge algorithm of Freund and Schapire [26], an exponential factor is used instead. This update rule is the following:

$$ {w_i}^{(t+1)} = {w_i}^{(t)} \cdot \exp(-\eta {m_i}^{(t)})\,. \tag{2.6}$$

As can be seen from the analysis of the MW algorithm, Hedge is not very different. The bound we obtain for Hedge is slightly different however. While most of the applications we present in the rest of the paper can be derived using Hedge as well with a little extra calculation, some applications, such as the ones in Sections 3.3 and 3.5, explicitly need the MW algorithm rather than Hedge to obtain better bounds. Here, we state the bound obtained for Hedge without proof—the analysis is on the same lines as before. The only difference is that instead of the inequalities (2.3), we use the inequality

$$\exp(-\eta x) \leq 1 - \eta x + \eta^2 x^2$$

if $|\eta x| \leq 1$.

Theorem 2.3. *Assume that all costs $m_i^{(t)} \in [-1,1]$ and $\eta \leq 1$. Then the Hedge algorithm guarantees that after T rounds, for any decision i, we have*

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \leq \sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t=1}^{T} (\mathbf{m}^{(t)})^2 \cdot \mathbf{p}^{(t)} + \frac{\ln n}{\eta}.$$

Here, $(\mathbf{m}^{(t)})^2$ is the vector obtained by taking coordinate-wise square of $\mathbf{m}^{(t)}$.

This guarantee is very similar to the one in Theorem 2.1, with one important difference: the term multiplying η is a loss which depends on the algorithm's distribution. In Theorem 2.1, this additional term depends on the loss of the best decision in hindsight. For some applications the latter guarantee is stronger (see Section 3.3).

2.2 Proof via KL-divergence

In this section, we give an alternative proof of Theorem 2.1 based on the Kullback-Leibler (KL) divergence, or relative entropy. While this proof is somewhat more complicated, it gives a good insight into why the MW algorithm works: the reason is that it tends to reduce the KL-divergence to the optimal solution. Another reason for giving this proof is that it yields a more nuanced form of the MW algorithm that is useful in some applications (such as the construction of hard-core sets, see Section 3.7). Readers may skip this section without loss of continuity.

For two distributions $\mathbf{p}$ and $\mathbf{q}$ on the decision set, the relative entropy between them is

$$\mathbf{RE}(\mathbf{p} \parallel \mathbf{q}) = \sum_i p_i \ln \frac{p_i}{q_i}.$$

where the term $p_i \ln(p_i/q_i)$ is defined to be zero if $p_i = 0$ and infinite if $p_i \neq 0$, $q_i = 0$.

Consider the following twist on the basic decision-making problem from Section 2. Fix a convex subset of distributions over decisions, $\mathcal{P}$ (note: the basic setting is recovered when $\mathcal{P}$ is the set of *all* distributions). In each round t, the decision-maker is required to produce a distribution $\mathbf{p}^{(t)} \in \mathcal{P}$. At that point, the cost vector $\mathbf{m}^{(t)}$ is revealed and the decision-maker suffers cost $\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$. Since we make the restriction that $\mathbf{p}^{(t)} \in \mathcal{P}$, we now want to compare the total cost of the decision-maker to the cost of the best fixed distribution in $\mathcal{P}$. Consider the algorithm in Figure 2.

Note that in the special case when $\mathcal{P}$ is the set of all distributions on the decisions, this algorithm is exactly the basic MW algorithm presented in Figure 1. The relative entropy projection step ensures that

Multiplicative Weights Update algorithm with Restricted Distributions

Initialization: Fix a $\eta \leq \frac{1}{2}$. Set $\mathbf{p}^{(1)}$ to be be an arbitrary distribution in $\mathcal{P}$ initialized to 1.
For $t = 1, 2, \ldots, T$:

1. Choose decision i by sampling from $\mathbf{p}^{(t)}$.

2. Observe the costs of the decisions $\mathbf{m}^{(t)}$.

3. Compute the probability vector $\hat{\mathbf{p}}^{(t+1)}$ using the usual multiplicative update rule: for every expert i,
$$p_i^{(t+1)} \;=\; p_i^{(t)}(1-\eta m_i^{(t)})/\Phi^{(t)} \tag{2.7}$$
where $\Phi^{(t)}$ is the normalization factor to make $\hat{\mathbf{p}}^{(t+1)}$ a distribution.

4. Set $\mathbf{p}^{(t+1)}$ to be the relative entropy projection of $\hat{\mathbf{p}}^{(t)}$ on the set $\mathcal{P}$, i. e.,
$$\mathbf{p}^{(t+1)} \;=\; \arg\min_{\mathbf{p}\in\mathcal{P}} \mathbf{RE}(\mathbf{p} \parallel \hat{\mathbf{p}}^{(t)}).$$

Figure 2: The Multiplicative Weights algorithm with Restricted Distributions.

we always choose a distribution in $\mathcal{P}$. This projection is a convex program since relative entropy is convex and $\mathcal{P}$ is a convex set, and hence can be computed using standard convex programming techniques.

We now prove a bound on the total cost of the algorithm (compare to Corollary 2.2). Note that in the basic setting when $\mathcal{P}$ is the set of all distributions, the bound given below is tighter than the one in Theorem 2.1.

Theorem 2.4. *Assume that all costs $m_i^{(t)} \in [-1,1]$ and $\eta \leq 1/2$. Then the Multiplicative Weights algorithm with Restricted Distributions guarantees that after T rounds, for any $\mathbf{p} \in \mathcal{P}$, we have*
$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;\leq\; \sum_{t=1}^{T} (\mathbf{m}^{(t)} + \eta|\mathbf{m}^{(t)}|) \cdot \mathbf{p} + \frac{\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(1)})}{\eta},$$
where $|\mathbf{m}^{(t)}|$ is the vector obtained by taking the coordinate-wise absolute value of $\mathbf{m}^{(t)}$.

Proof. We use the relative entropy between $\mathbf{p}$ and $\mathbf{p}^{(t)}$, $\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(t)}) := \sum_i p_i \ln(p_i/p_i^{(t)})$ as a “potential” function. We have
$$\begin{aligned}
\mathbf{RE}(\mathbf{p} \parallel \hat{\mathbf{p}}^{t+1}) - \mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(t)}) \;&=\; \sum_i p_i \ln \frac{p_i^{(t)}}{\hat{p}_i^{(t+1)}} \\
&=\; \sum_i p_i \ln \frac{\Phi^{(t)}}{(1-\eta m_i^{(t)})} \\
&\leq\; \ln \frac{1}{1-\eta} \sum_{\geq 0} p_i m_i^{(t)} + \ln(1+\eta) \sum_{<0} p_i m_i^{(t)} + \ln \Phi^{(t)} \\
&\leq\; \eta(\mathbf{m}^{(t)} + \eta|\mathbf{m}^{(t)}|) \cdot \mathbf{p} + \ln \Phi^{(t)}.
\end{aligned}$$

The first inequality above follows from (2.3), and the second follows from (2.5). Next, we have

$$\ln \Phi^{(t)} \;=\; \ln\left[\sum_i p_i{}^{(t)}(1-\eta m_i{}^{(t)})\right] \;=\; \ln[1-\eta \mathbf{m}^{(t)}\cdot \mathbf{p}^{(t)}] \;\le\; -\eta \mathbf{m}^{(t)}\cdot \mathbf{p}^{(t)}$$

since $\ln(1-x)\le -x$ for $x<1$. Thus, we get

$$\mathbf{RE}(\mathbf{p}\,\|\,\hat{\mathbf{p}}^{t+1}) - \mathbf{RE}(\mathbf{p}\,\|\,\mathbf{p}^{(t)}) \;\le\; \eta(\mathbf{m}^{(t)}+\eta|\mathbf{m}^{(t)}|)\cdot\mathbf{p} - \eta(\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)})\,.$$

This inequality essentially says that if the cost of the algorithm in round t, $\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)}$, is significantly larger than the cost of the comparator, $\mathbf{m}^{(t)}\cdot\mathbf{p}$, then $\hat{\mathbf{p}}^{t+1}$ moves closer to $\mathbf{p}$ (in the relative entropy distance) than $\mathbf{p}^{(t)}$.

Now, projection on the set $\mathcal{P}$ using the relative entropy as a distance function is a Bregman projection, and thus it satisfies the following Generalized Pythagorean inequality (see, e. g., [39]), for any $\mathbf{p}\in\mathcal{P}$:

$$\mathbf{RE}(\mathbf{p}\,\|\,\mathbf{p}^{(t+1)}) + \mathbf{RE}(\mathbf{p}^{(t+1)}\,\|\,\hat{\mathbf{p}}^{(t+1)}) \;\le\; \mathbf{RE}(\mathbf{p}\,\|\,\hat{\mathbf{p}}^{(t+1)})\,.$$

I. e., the projection step only brings the distribution closer to $\mathbf{p}$. Since relative entropy is always non-negative, we have $\mathbf{RE}(\mathbf{p}^{(t+1)}\,\|\,\hat{\mathbf{p}}^{(t+1)})\ge 0$ and so

$$\mathbf{RE}(\mathbf{p}\,\|\,\mathbf{p}^{(t+1)}) - \mathbf{RE}(\mathbf{p}\,\|\,\mathbf{p}^{(t)}) \;\le\; \eta(\mathbf{m}^{(t)}+\eta|\mathbf{m}^{(t)}|)\cdot\mathbf{p} - \eta(\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)})\,.$$

Summing up from $t=1$ to T, dividing by η, and simplifying using the fact that $\mathbf{RE}(\mathbf{p}\,\|\,\mathbf{p}^{(T+1)})$ is non-negative, we get the stated bound. □

2.3 Gains instead of losses

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

The resulting algorithm is identical, and the following theorem follows directly from Theorem 2.1 by simply negating the quantities:

Theorem 2.5. *Assume that all gains $m_i{}^{(t)}\in[-1,1]$ and $\eta\le 1$. Then the Multiplicative Weights algorithm (for gains) guarantees that after T rounds, for any expert i, we have*

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

We also have the following immediate corollary, corresponding to Corollary 2.2:

Corollary 2.6. *The Multiplicative Weights algorithm also guarantees that after T rounds, for any distribution $\mathbf{p}$ on the decisions,*

$$\sum_{t=1}^{T}\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)} \;\ge\; \sum_{t=1}^{T}(\mathbf{m}^{(t)}-\eta|\mathbf{m}^{(t)}|)\cdot\mathbf{p} - \frac{\ln n}{\eta}\,,$$

where $|\mathbf{m}^{(t)}|$ is the vector obtained by the taking the coordinate-wise absolute value of $\mathbf{m}^{(t)}$.

3 Applications

Typically, the Multiplicative Weights method is applied in the following manner. A prototypical example is to solve a constrained optimization problem. We then let a decision represent each constraint in the problem, with costs specified by the points in the domain of interest. For a given point, the cost of a decision is made proportional to *how well* the corresponding constraint is satisfied on the point. This might seem counterintuitive, but recall that we *reduce* a decision's weight depending on its penalty, and if a constraint is well satisfied on points so far we would like its weight to be smaller, so that the algorithm focuses on constraints that are poorly satisfied.

In many applications (though not all) the choice of point is also under our control. Typically we will need to generate the *maximally adversarial* point, i. e., the point that maximizes the expected cost. Then the overall algorithm consists of two subprocedures: an "oracle" for generating the maximally adversarial point at each step, and the MW algorithm for updating the weights of the decisions. With this intuition, we can describe the following applications.

3.1 Learning a linear classifier: the Winnow algorithm

To the best of our knowledge, the first time multiplicative weight updates were used was in the Winnow algorithm of Littlestone [50]. This is an algorithmic technique used in machine learning to learn linear classifiers. Equivalently, this can also be seen as solving a linear program.

The setup is as follows. We are given m labeled examples, $(\mathbf{a}_1,\ell_1),(\mathbf{a}_2,\ell_2),\ldots,(\mathbf{a}_m,\ell_m)$ where $\mathbf{a}_j \in \mathbb{R}^n$ are *feature vectors*, and $\ell_j \in \{-1,1\}$ are their labels. Our goal is to find non-negative weights such that for any example, the sign of the weighted combination of the features matches its labels, i. e., find $\mathbf{x} \in \mathbb{R}^n$ with $x_i \geq 0$ such that for all $j = 1,2,\ldots,m$, we have $\operatorname{sgn}(\mathbf{a}_j \cdot \mathbf{x}) = \ell_j$. Equivalently, we require that $\ell_j \mathbf{a}_j \cdot \mathbf{x} \geq 0$ for all j. Without loss of generality, we may assume that the weights sum to 1 so that they form a distribution, i. e., $\mathbf{1} \cdot \mathbf{x} = 1$, where $\mathbf{1}$ is the all 1's vector.

Thus, for notational convenience, if we redefine $\mathbf{a}_j$ to be $\ell_j \mathbf{a}_j$, then the problem reduces to finding a solution to the following LP:

$$\begin{aligned} \forall j = 1,2,\ldots,m:\ \mathbf{a}_j \cdot \mathbf{x} &\geq 0\,, \\ \mathbf{1} \cdot \mathbf{x} &= 1\,, \\ \forall i:\ x_i &\geq 0\,. \end{aligned} \tag{3.1}$$

Note this is a quite general form of LP, and many commonly seen LPs can be reduced to this form.

Now suppose there is a *large-margin* solution to this problem. I. e., there is an $\varepsilon > 0$ and a distribution $\mathbf{x}^\star$ so that for all j, we have $\mathbf{a}_j \cdot \mathbf{x}^\star \geq \varepsilon$. We now give an algorithm based on MW to solve the LP above. Define $\rho = \max_j \|\mathbf{a}_j\|_\infty$.

We run the MW algorithm in the gain form (see Section 2.3) with $\eta = \varepsilon/(2\rho)$. The decisions are given by the n features, and gains are specified by the m examples. The gain for feature i for example j is a_{ij}/ρ. Note that these gains lie in the range $[-1,1]$ as required.

In each round t, let $\mathbf{x}$ to be the distribution $\mathbf{p}^{(t)}$ generated by the MW algorithm. Now, we look for a misclassified example, i. e., an example j such that $\mathbf{a}_j \cdot \mathbf{x} < 0$. If no such constraint exists, we are done and we can stop. Otherwise, if j is a misclassified example, then it specifies the gains for round t. Note that the gain in round t is $\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} = (1/\rho)\mathbf{a}_j \cdot \mathbf{x} < 0$, whereas for the solution $\mathbf{x}^\star$, we have

$$\mathbf{m}^{(t)} \cdot \mathbf{x}^\star = \frac{1}{\rho}\mathbf{a}_j \cdot \mathbf{x}^\star \geq \frac{\varepsilon}{\rho}\,.$$

We keep running the MW algorithm until we find a good solution (i. e., one that classifies all examples correctly).

To get a bound on the number of iterations until we find a good solution, we apply Corollary 2.6 with $\mathbf{p} = \mathbf{x}^\star$. Using the trivial bound $\eta|\mathbf{m}^{(t)}| \cdot \mathbf{p} \leq \eta$, we have

$$0 > \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;\geq\; \sum_{t=1}^{T} (\mathbf{m}^{(t)} - \eta|\mathbf{m}^{(t)}|) \cdot \mathbf{p} - \frac{\ln n}{\eta} \;\geq\; \frac{\varepsilon}{2\rho}T - \frac{2\rho \ln n}{\varepsilon}\,,$$

which implies that $T < 4\rho^2 \ln(n)/\varepsilon^2$. Thus, in at most $\lceil 4\rho^2 \ln(n)/\varepsilon^2 \rceil$ iterations, we find a good solution.

3.2 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 [27], 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 [34].)

Let $\mathbf{A}$ be the payoff matrix of a finite 2-player zero-sum game, with n rows (the number of columns will play no role). When the row player plays strategy i and the column player plays strategy j, then the payoff to the column player is $A(i, j) := A_{ij}$. We assume that $A(i, j) \in [0, 1]$. If the row player chooses his strategy i from a distribution $\mathbf{p}$ over the rows, then the expected payoff to the column player for choosing a strategy j is $A(\mathbf{p}, j) := \mathbf{E}_{i\in\mathbf{p}}[A(i, j)]$. Thus, the best response for the column player is the strategy j which maximizes this payoff. Similarly, if the column player chooses his strategy j from a distribution $\mathbf{q}$ over the columns, then the expected payoff he gets if the row player chooses the strategy i is $A(i, \mathbf{q}) := \mathbf{E}_{j\in\mathbf{q}}[A(i, j)]$. Thus, the best response for the row player is the strategy i which minimizes this payoff. John von Neumann's min-max theorem says that if each of the players chooses a distribution over their strategies to optimize their worst case payoff (or payout), then the value they obtain is the same:

$$\min_{\mathbf{p}} \max_{j} A(\mathbf{p}, j) \;=\; \max_{\mathbf{q}} \min_{i} A(i, \mathbf{q}) \tag{3.2}$$

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

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

$$\lambda^* - \varepsilon \;\leq\; \min_{i} A(i, \tilde{\mathbf{q}})\,, \tag{3.3}$$

$$\max_{j} A(\tilde{\mathbf{p}}, j) \;\leq\; \lambda^* + \varepsilon\,. \tag{3.4}$$

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

Theorem 3.1. *Given an error parameter $\varepsilon > 0$, there is an algorithm which solves the zero-sum game up to an additive factor of ε using $O(\log(n)/\varepsilon^2)$ calls to* ORACLE*, with an additional processing time of $O(n)$ per call.*

Proof. We map our general algorithm from Section 2 to this setting by considering (3.3) as specifying n linear constraints on the probability vector $\tilde{\mathbf{q}}$: viz., for all rows i, $A(i, \tilde{\mathbf{q}}) \geq \lambda^* - \varepsilon$. Now, following the intuition given in the beginning of this section, we make our decisions to correspond to pure strategies of the row player. Thus a distribution on the decisions corresponds to a mixed row strategy. Costs of the decisions are specified by pure strategies of the column player. The cost paid by a decision i when column player chooses strategy j is $A(i, j)$.

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

Since all $A(i, j) \in [0, 1]$, we can apply Corollary 2.2 to get that after T rounds, for any distribution on the rows $\mathbf{p}$, we have

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

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

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

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

$$\lambda^* \;\leq\; \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}^{(t)}, j^{(t)}) \;\leq\; \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}, j^{(t)}) + \varepsilon \;\leq\; \lambda^* + \varepsilon\,. \tag{3.5}$$

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

Let $\tilde{t}$ be the round t with the minimum value of $A(\mathbf{p}^{(t)}, j^{(t)})$. We have, from (3.5),

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

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

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

$$A(\mathbf{p}^*, j^*) = \frac{1}{T} \sum_t A(\mathbf{p}^{(t)}, j^*) \leq \frac{1}{T} \sum_t A(\mathbf{p}^{(t)}, j^{(t)}) \leq \lambda^* + \varepsilon\,.$$

We set $\mathbf{q}^*$ to be the distribution which assigns to column j the probability

$$\frac{|\{t:\ j^{(t)}=j\}|}{T}.$$

From (3.5), for any row strategy i, by setting $\mathbf{p}$ to be concentrated on the pure strategy i, we have

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

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

3.3 Plotkin, Shmoys, Tardos framework for packing/covering LPs

Plotkin, Shmoys, and Tardos [56] generalized some known flow algorithms to a framework for approximately solving *fractional packing and covering* problems, which are a special case of linear programming formally defined below. Their algorithm is a quantitative version of the classical *Lagrangean 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. 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 convex program:

$$\exists ?\mathbf{x}\ \in\ \mathcal{P}:\quad \mathbf{Ax}\ \ge\ \mathbf{b}, \tag{3.6}$$

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

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

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

$$\exists ?\mathbf{x}\in\mathcal{P}:\quad \mathbf{p}^\top\mathbf{Ax}\ \ge\ \mathbf{p}^\top\mathbf{b}. \tag{3.7}$$

One way to implement this procedure is by maximizing $\mathbf{p}^\top\mathbf{Ax}$ over $\mathbf{x}\in\mathcal{P}$. It is reasonable to expect such an optimization procedure to exist (indeed, such is the case for many applications) since we only need to check the feasibility of one constraint rather than m. If the feasibility problem (3.6) has a solution $\mathbf{x}^*$, then the same solution also satisfies (3.7) for *any* probability vector $\mathbf{p}$ over the constraints. Thus, if there is a probability vector $\mathbf{p}$ over the constraints such that no $\mathbf{x}\in\mathcal{P}$ satisfies (3.7), then it is proof that the original problem is infeasible.

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

Definition 3.2. An (ℓ,ρ)**-bounded** ORACLE, for parameters $0 \le \ell \le \rho$, is an algorithm which given a probability vector $\mathbf{p}$ over the constraints, solves the feasibility problem (3.7). Furthermore, there is a fixed subset $I \subseteq [m]$ of constraints such that whenever the ORACLE manages to find a point $\mathbf{x} \in \mathcal{P}$ satisfying (3.7), the following holds:

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

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

In previous work, such as [56], only (ρ,ρ)-bounded ORACLEs are considered. We separate out the upper and lower bounds in order to obtain tighter guarantees on the running time. The results of [56] can be recovered simply by setting $\ell = \rho$.

Theorem 3.3. *Let $\varepsilon > 0$ be a given error parameter. Suppose there exists an (ℓ,ρ)-bounded* ORACLE *for the feasibility problem (3.7). Assume that $\ell \ge \varepsilon/2$. Then there is an algorithm which either finds an $\mathbf{x}$ such that $\forall i$. $\mathbf{A}_i\mathbf{x} \ge b_i - \varepsilon$, or correctly concludes that the system is infeasible. The algorithm makes only $O(\ell\rho\log(m)/\varepsilon^2)$ calls to the* ORACLE*, with an additional processing time of $O(m)$ per call.*

Proof. The condition $\ell \ge \varepsilon/2$ is only technical, and if it is not met we can just redefine ℓ to be $\varepsilon/2$. To map our general framework to this situation, we have a decision representing each of the m constraints. Costs are determined by points $\mathbf{x} \in \mathcal{P}$. The cost of constraint i for point $\mathbf{x}$ is $(1/\rho)[\mathbf{A}_i\mathbf{x} - b_i]$ (so that the costs lie in the range $[-1,1]$).

In each round t, given a distribution over the decisions (i. e., the constraints) $\mathbf{p}^{(t)}$, we run the ORACLE with $\mathbf{p}^{(t)}$. If the ORACLE declares that there is no $\mathbf{x} \in \mathcal{P}$ such that ${\mathbf{p}^{(t)}}^\top\mathbf{A}\mathbf{x} \ge {\mathbf{p}^{(t)}}^\top\mathbf{b}$, then we stop, because now $\mathbf{p}^{(t)}$ is proof that the problem (3.6) is infeasible.

So let us assume that this doesn't happen, i. e., in all rounds t, the ORACLE manages to find a solution $\mathbf{x}^{(t)}$ such ${\mathbf{p}^{(t)}}^\top\mathbf{A}\mathbf{x} \ge {\mathbf{p}^{(t)}}^\top\mathbf{b}$. Since the cost vector to the Multiplicative Weights algorithm is specified to be $\mathbf{m}^{(t)} := (1/\rho)[\mathbf{A}\mathbf{x}^{(t)} - \mathbf{b}]$, we conclude that the expected cost in each round is non-negative:

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

Let $i \in I$. Then Theorem 2.1 tells us that after T rounds,

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

Here, the subscript "< 0" refers to the rounds t when $\mathbf{A}_i\mathbf{x}^{(t)} - b_i < 0$. The last inequality follows because if $\mathbf{A}_i\mathbf{x}^{(t)} - b_i < 0$, then $|\mathbf{A}_i\mathbf{x}^{(t)} - b_i| \le \ell$. Dividing by T, multiplying by ρ, and letting $\bar{\mathbf{x}} = (1/T)\sum_{t=1}^{T}\mathbf{x}^{(t)}$

(note that $\bar{\mathbf{x}} \in \mathcal{P}$ since $\mathcal{P}$ is a convex set), we get that

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

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

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

Reasoning similarly for $i \notin I$, we get the same inequality. Putting both together, we conclude that $\bar{\mathbf{x}}$ satisfies the feasibility problem (3.6) up to an additive ε factor, as desired. □

3.3.1 Concave constraints

The algorithm of Section 3.3 works not just for linear constraints over a convex domain, but also for concave constraints. Imagine that we have the following feasibility problem:

$$\exists ?\mathbf{x} \in \mathcal{P}: \quad \forall i \in [m]: \; f_i(\mathbf{x}) \geq 0 \tag{3.8}$$

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

$$\exists ?\mathbf{x} \in \mathcal{P}: \quad \sum_i p_i f_i(\mathbf{x}) \geq 0\,. \tag{3.9}$$

An ORACLE would be called (ℓ,ρ)-bounded there is a fixed subset of constraints $I \subseteq [m]$ such that whenever it returns a feasible solution $\mathbf{x}$ to (3.9), all constraints $i \in I$ take values in the range $[-\ell,\rho]$ on the point $\mathbf{x}$, and all the rest take values in $[-\rho,\ell]$.

Theorem 3.4. *Let $\varepsilon > 0$ be a given error parameter. Suppose there exists an (ℓ,ρ)-bounded* ORACLE *for the feasibility problem (3.8). Assume that $\ell \geq \varepsilon/2$. Then there is an algorithm which either solves the problem up to an additive error of ε, or correctly concludes that the system is infeasible, making only $O(\ell\rho\log(m)/\varepsilon^2)$ calls to the* ORACLE*, with an additional processing time of $O(m)$ per call.*

Proof. Just as before, we have a decision for every constraint, and costs are specified by points $\mathbf{x} \in \mathcal{P}$. The cost of constraint i for point $\mathbf{x}$ is $(1/\rho)f_i(\mathbf{x})$.

Now we run the Multiplicative Weights algorithm with this setup. Again, if at any point the ORACLE declares that (3.9) is infeasible, we immediately halt and declare the system (3.8) infeasible. So assume this never happens. Then as before, the expected cost in each round is $\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)} \geq 0$. Now, applying Theorem 2.1 as before, we conclude that for any $i \in I$, we have

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

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

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

since

$$\frac{1}{T}\sum_{t=1}^{T} f_i(\mathbf{x}^{(t)}) \leq f_i\left(\frac{1}{T}\sum_{t=1}^{T}\mathbf{x}^{(t)}\right),$$

by Jensen's inequality, since all the f_i are concave.

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

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

Reasoning similarly for $i \notin I$, we get the same inequality. Putting both together, we conclude that $\bar{\mathbf{x}}$ satisfies the feasibility problem (3.8) up to an additive ε factor, as desired. □

3.3.2 Approximate ORACLEs

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

Define a ε-approximate ORACLE for the feasibility problem (3.6) to be one that solves the feasibility problem (3.7) up to an additive error of ε. That is, given a probability vector $\mathbf{p}$ on the constraints, either it finds an $\mathbf{x} \in \mathcal{P}$ such that $\mathbf{p}^\top\mathbf{A}\mathbf{x} \geq \mathbf{p}^\top\mathbf{b} - \varepsilon$, or it declares correctly that (3.7) is infeasible.

Theorem 3.5. *Let $\varepsilon > 0$ be a given error parameter. Suppose there exists an (ℓ,ρ)-bounded $\varepsilon/3$-approximate* ORACLE *for the feasibility problem (3.6). Assume that $\ell \geq \varepsilon/3$. Then there is an algorithm which either solves the problem up to an additive error of ε, or correctly concludes that the system is infeasible, making only $O(\ell\rho\log(m)/\varepsilon^2)$ calls to the* ORACLE*, with an additional processing time of $O(m)$ per call.*

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

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

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

3.3.3 Fractional Covering Problems

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

We assume without loss of generality (by appropriately scaling the inequalities) that $b_i = 1$ for all rows, so that now we desire to find an $\mathbf{x} \in \mathcal{P}$ which satisfies the system within an additive ε factor. Since for all $\mathbf{x} \in \mathcal{P}$, we have $\mathbf{A}\mathbf{x} \geq 0$, and since all $b_i = 1$, we conclude that for any i, $\mathbf{A}_i\mathbf{x} - b_i \geq -1$. Thus, we assume that there is a $(1,\rho)$-bounded ORACLE for this problem. Now, applying Theorem 3.3, we get the following theorem.

Theorem 3.6. *Suppose there exists a $(1,\rho)$-bounded* ORACLE *for the program $\mathbf{Ax} \geq \mathbf{1}$ with $\mathbf{x} \in \mathcal{P}$. Given an error parameter $\varepsilon > 0$, there is an algorithm which computes a ε-approximate solution to the program, or correctly concludes that it is infeasible, using $O(\rho \log(m)/\varepsilon^2)$ calls to the* ORACLE*, plus an additional processing time of $O(m)$ per call.*

3.3.4 Fractional Packing Problems

A fractional packing problem can be written as

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

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

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

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

we cast it in our general framework, and a solution $\mathbf{x} \in \mathcal{P}$ which satisfies this up to an additive ε is a ε-approximate solution to the original system. Since for all $\mathbf{x} \in \mathcal{P}$, we have $\mathbf{Ax} \geq 0$, and since all $b_i = 1$, we conclude that for any i, $-\mathbf{A}_i\mathbf{x} + b_i \leq 1$. Thus, we assume that there is a $(1,\rho)$-bounded ORACLE for this problem. Now, applying Theorem 3.3, we get the following:

Theorem 3.7. *Suppose there exists a $(1,\rho)$-bounded* ORACLE *for the program $-\mathbf{Ax} \geq -\mathbf{1}$ with $\mathbf{x} \in \mathcal{P}$. Given an error parameter $\varepsilon > 0$, there is an algorithm which computes a ε-approximate solution to the program, or correctly concludes that it is infeasible, using $O(\rho \log(m)/\varepsilon^2)$ calls to the* ORACLE*, plus an additional processing time of $O(m)$ per call.*

3.4 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-time because its running time is bounded by a polynomial function of the edge capacities (as opposed to the logarithm of the capacities, which is the number of bits needed to represent them). Garg and Könemann [29, 30] 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 [29, 30] (the termination condition is slightly different) and neither is the proof; the running time bound is the same however.

For illustrative purposes we focus on the *maximum multicommodity flow problem*. In this problem, we are given a graph $G = (V,E)$ with capacities c_e on edges, and set of k source-sink pairs of nodes. Let P be the set of all paths between the source-sink pairs. The objective is to maximize the total flow between these pairs. The LP formulation is as follows:

$$\begin{aligned} &\max \sum_{p\in P} f_p\,, \\ \forall e\in E:\quad &\sum_{p\ni e} f_p \;\le\; c_e\,, \\ \forall p\in P:\; &f_p \;\ge\; 0\,. \end{aligned} \tag{3.10}$$

Here, the variable f_p represents the flow on path p.

Before presenting the Garg-Könemann idea we first present the algorithm one would obtain by applying our packing-covering framework (Section 3.3) in the obvious way.

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^{opt} of the total flow in the optimum solution. Then we want to solve the following feasibility problem:

$$\exists ?\in \mathcal{P}:\ \forall e\in E:\ \sum_{p\ni e} f_p \le c_e\,,$$

where

$$\mathcal{P}\ :=\ \left\{\mathbf{f}:\ \forall p\in P:\ f_p\ge 0,\ \sum_{p\in P} f_p = F^{\text{opt}}\right\}.$$

In this form, the feasibility problem given above is a packing LP, thus, we can apply the Multiplicative Weights algorithm of Section 3.3.4.

As outlined in Section 3.3, the obvious algorithm would maintain at each step t a weight ${w_e}^{(t)}$ for each edge e. The ORACLE can be implemented by finding the flow in $\mathcal{P}$ which minimizes

$$\sum_e {w_e}^{(t)} \sum_{p\ni e} f_p/c_e = \sum_p f_p \sum_{e\in p} {w_e}^{(t)}/c_e\,.$$

The optimal flow is supported on a single path, namely, the path $p^{(t)}\in P$ that has minimum length, when every edge $e\in E$ is given length ${w_e}^{(t)}/c_e$. Thus in every round we find this path $p^{(t)}$ and pass a flow F^{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^{opt}. Costs for the edges are defined as in Section 3.3.

Unfortunately the width parameter is

$$\rho = \max_{\mathbf{f}\in\mathcal{P}} \max_e \sum_{p\ni e} f_p/c_e = F^{\text{opt}}/c_{\min}$$

where $c_{\min}$ is the capacity of the minimum capacity edge in the graph. The algorithm requires $T = \rho\ln(n)/\varepsilon^2$ iterations to get an $(1-\varepsilon)$-approximation to the optimal flow. The overall running time is $\tilde{O}(F^{\text{opt}}T_{\text{sp}}/c_{\min})$ where $T_{\text{sp}} = \tilde{O}(mk)$ is the time needed to compute k shortest paths. As already mentioned, this is not polynomial-time since it depends upon $1/c_{\min}$ rather than the logarithm of this value.

Now we describe the Garg-Könemann modification. We continue to maintain weights ${w_e}^{(t)}$ for every edge e, where initially, ${w_e}^{(1)} = 1$ for all e. The costs are determined by flows as before, however, we consider a larger set of flows, viz.,

$$\mathcal{P}' := \{\mathbf{f} : \forall p \in P : f_p \geq 0\}.$$

Note that we no longer need to use the value F^{opt}. Again, in each round we choose a flow $\mathbf{f} \in \mathcal{P}'$ that is supported on the shortest path $p^{(t)} \in P$ with edge lengths w_e/c_e.

The main idea behind width reduction is the following: instead of routing the same flow F^{opt} at each time step, we route only as much flow as is allowed by the minimum capacity edge on the path. In other words, at time t we route a flow of value $c^{(t)}$ on path $p^{(t)}$, where $c^{(t)}$ is the minimum capacity of an edge on the path $p^{(t)}$. The cost incurred by edge e is ${m_e}^{(t)} = c^{(t)}/c_e$. (In other words, a cost of $1/c_e$ per unit of flow passing through e.) *The width is therefore automatically upper bounded by* 1.

We run the MW algorithm with $\eta = \varepsilon/2$. The 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:

$$\forall e \in p^{(t)} : \quad {w_e}^{(t+1)} = {w_e}^{(t)} \left(1 + \eta \cdot \frac{c^{(t)}}{c_e}\right).$$

The *termination rule* for the algorithm is to stop when as soon as for some edge e, the *congestion* $f_e/c_e \geq \ln(m)/\eta^2$, where f_e is the total amount of flow routed by the algorithm so far on edge e.

3.4.1 Analysis

We apply Theorem 2.5. Since we have ${m_e}^{(t)} \in [0, 1]$ for all edges e and rounds t, we conclude that for any edge e, we have

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \geq (1 - \eta) \sum_{t=1}^{T} {m_e}^{(t)} - \frac{\ln(m)}{\eta}. \tag{3.11}$$

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

$$\sum_{t=1}^{T} {\mathbf{m}_e}^{(t)} = \frac{f_e}{c_e}, \tag{3.12}$$

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

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} = \sum_{t=1}^{T} \frac{\sum_{e \in p^{(t)}} \frac{c^{(t)}}{c_e} \cdot {w_e}^{(t)}}{\sum_e {w_e}^{(t)}} = \sum_{t=1}^{T} c^{(t)} \cdot \frac{\sum_{e \in p^{(t)}} \cdot \frac{{w_e}^{(t)}}{c_e}}{\sum_e {w_e}^{(t)}}. \tag{3.13}$$

Now, suppose the optimum flow assigns f_p^{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 $p \in P$ with these edge lengths 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 $\sum_{p' \ni e} f_{p'}^{\mathrm{opt}} \le c_e$. 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 (3.13), we get that

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \ \le\ \sum_{t=1}^{T} \frac{c^{(t)}}{F^{\mathrm{opt}}} \ =\ \frac{F}{F^{\mathrm{opt}}}\,, \tag{3.14}$$

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

Plugging (3.12) and (3.14) into (3.11), we get that

$$\frac{F}{F^{\mathrm{opt}}} \ \ge\ (1-\eta)\max_{e}\left\{\frac{f_e}{c_e}\right\} - \frac{\ln(m)}{\eta}\,.$$

We stop the algorithm as soon as there we have an edge e with congestion $f_e/c_e \ge \ln(m)/\eta^2$, so when the algorithm terminates we have

$$\frac{F}{F^{\mathrm{opt}}} \ \ge\ (1-2\eta)\max_{e}\left\{\frac{f_e}{c_e}\right\}.$$

Now, $C := \max_e \{f_e/c_e\}$ 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

$$\frac{F}{C} \ \ge\ (1-2\eta)F^{\mathrm{opt}}\,,$$

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

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

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

3.5 $O(\log n)$-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 [58] randomized rounding. This yields a randomized algorithm; to obtain a deterministic algorithm, derandomize it using Raghavan's [57] method of *pessimistic estimators*.

Young [65] has given an especially clear framework for understanding these algorithms which as a bonus also yields faster, combinatorial algorithms for approximating integer packing/covering programs. 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 the analysis of randomized rounding uses the Chernoff-Hoeffding bounds. These bounds show that the sum of bounded independent random variables $X_1, X_2, \ldots, X_n$ (which should be thought of as the random variables generated in the randomized rounding algorithm) is sharply concentrated about its mean, and are proved by applying Markov's inequality to the variable $e^{\eta(\sum_i X_i)}$ for some parameter η. The key observation now is that the aforementioned application of Markov's inequality bounds the probability of failure (of the randomized rounding algorithm) away from 0 in terms of $\mathbf{E}[e^{\eta(\sum_i X_i)}]$. Thus, one can treat $\mathbf{E}[e^{\eta(\sum_i X_i)}]$ as a pessimistic estimator (up to scaling by a constant) of the failure probability, and derandomization can be achieved by greedily (and hence, deterministically) choosing the X_i sequentially to decrease this pessimistic estimator. The resulting algorithm is essentially the MW algorithm: in each round t, the deterministic part of the pessimistic estimator, viz. $e^{\eta \sum_{\tau<t} X_\tau}$, plays the role of the weight.

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 detail 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 up to constant factors (assuming reasonable complexity-theoretic conjectures [23]).

In the SET COVER problem, we are given a universe of n elements, say $U = \{1,2,3,\ldots,n\}$ 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. Each element of the universe represents a constraint that the union of sets picked by the algorithm must cover it. Following the guidelines given in the beginning of Section 3, we cast the problem in our framework by letting decisions correspond to elements in the universe, and costs determined by sets $C_j \in \mathcal{C}$. The cost of the constraint corresponding to element i for a given set C_j is 1 if $i \in C_j$ and 0 otherwise.

To translate the greedy algorithm to our framework, suppose we run the Multiplicative Weights Update algorithm with this setup with $\eta = 1$. Since the analysis in the proof of Theorem 2.1 technically requires $\eta \le 1/2$, in the following we repeat the same potential function analysis for the current setting for $\eta = 1$. For $\eta = 1$, the update rule $w_i^{(t+1)} = w_i^{(t)}(1-\eta m_i^{(t)})$ implies that elements that have been covered so far have weight 0 while all the rest have weight 1. Thus, the distribution $\mathbf{p}^{(t)}$ is simply the uniform distribution on the uncovered elements until time t. Since cost of an element for a given set is 1 if it is in the set and 0 otherwise, the set that maximizes the expected cost under $\mathbf{p}^{(t)}$ is the one that maximizes the number of uncovered elements. The resulting algorithm is the Greedy Set Cover algorithm.

Since OPT sets cover all the elements, for any distribution $p_1, p_2, \ldots, p_n$ on the elements, one set must cover at least 1/OPT fraction of elements. This implies that if we choose set C in round t that maximizes the number of uncovered elements, we have

$$\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} = \max_C \sum_{i \in C} p_i \ge 1/\mathrm{OPT}\,.$$

Following the analysis of Theorem 2.1, the change in potential for each round is:

$$\Phi^{(t+1)} = \Phi^{(t)}(1-\eta\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)}) \ < \ \Phi^{(t)}e^{-\eta/\mathrm{OPT}} \ = \ \Phi^{(t)}e^{-1/\mathrm{OPT}}.$$

The strict inequality holds because $\mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)} > 0$ as long as there are uncovered elements. Thus, the potential drops by a factor of $e^{-1/\mathrm{OPT}}$ every time.

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

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

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

3.6 Learning theory and boosting

Boosting [60]—the process of combining several moderately accurate rules-of-thumb into a single highly accurate prediction rule—is a central idea in Machine Learning today. Freund and Schapire's *AdaBoost* [26] uses the Multiplicative Weights Update Rule and fits in our framework. Here we explain the main idea using some simplifying assumptions.

Let $\mathcal{X}$ be some set (*domain*) and suppose we are trying to learn an unknown function (*concept*) $c : \mathcal{X} \to \{0,1\}$ 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 $\mathcal{X}$, the learning algorithm is required to output a *hypothesis* $h : \mathcal{X} \to \{0,1\}$. 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 $\mathcal{D}$, given $\varepsilon, \delta > 0$ and access to random examples drawn from $\mathcal{D}$, outputs with probability at least $1-\delta$ a hypothesis whose error is at most ε. Furthermore, the running time is required to be polynomial in $1/\varepsilon$, $1/\delta$ and othe relevant parameters. A γ*-weak learning algorithm*, for some given $\gamma > 0$, is an algorithm satisfying the same conditions but the error can be as high as $1/2-\gamma$. Boosting shows that if a γ-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 γ.)

We prove this result in the so-called *boosting by sampling framework*, which uses a fixed training set S of N examples drawn from the distribution $\mathcal{D}$. The goal is to make sure that the final hypothesis erroneously classifies at most ε fraction of this training set. Using VC-dimension theory (see [26]) one can then show that if the weak learner produces hypotheses from a class $\mathcal{H}$ of bounded VC-dimension, and if N is chosen large enough (in terms of the error and confidence parameters, and the VC-dimension of the hypothesis class $\mathcal{H}$), then with probability at least $1-\delta$ over the choice of the sample set, the error of the hypothesis over the entire domain $\mathcal{X}$ (under distribution $\mathcal{D}$) is at most 2ε.

The idea in boosting is to repeatedly run the weak learning algorithm on different distributions defined on the fixed training set S. The final hypothesis has error ε under the *uniform* distribution on S. We run the MW algorithm with $\eta = \gamma$ for $T = \lceil (2/\gamma^2)\ln(1/\varepsilon) \rceil$ rounds. The decisions correspond to samples in S and costs are specified by a hypothesis generated by the weak learning algorithm, in the following way. If hypothesis h is generated, the cost for decision point x is 1 or 0 depending on whether $h(x) = c(x)$ or not. In other words, the cost vector $\mathbf{m}$ (indexed by x rather than i) is specified by $m_x = 1 - |h(x) - c(x)|$. Intuitively, we want the weight of an example to *increase* if the hypothesis labels it incorrectly.

In each iteration, the algorithm presents the current distribution $\mathbf{p}^{(t)}$ on the examples to the weak learning algorithm, and in return obtains a hypothesis $h^{(t)}$ whose error with respect to the distribution $\mathbf{p}^{(t)}$ is not more than $1/2 - \gamma$, in other words, the expected cost in each iteration satisfies

$$\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;\geq\; \frac{1}{2} + \gamma .$$

The algorithm is run for T rounds, where T will be specified shortly. The final hypothesis, h_{final}, labels $x \in \mathcal{X}$ according to the majority vote among $h^{(1)}(x), h^{(2)}(x), \ldots, h^{(T)}(x)$.

Let E be the set of $x \in S$ incorrectly labeled by h_{final}. The total cost of each $x \in E$,

$$\sum_t \mathbf{m}_x{}^{(t)} \;=\; \sum_t 1 - |h^{(t)}(x) - c(x)| \;\leq\; \frac{T}{2}$$

since the majority vote gives an incorrect label for it. Instead of applying Theorem 2.1, we apply Theorem 2.4 which gives a more nuanced bound. The set $\mathcal{P}$ in this theorem is simply the set of all possible distributions on S. Choosing $\mathbf{p}$ to be the uniform distribution on E, we get

$$\left(\frac{1}{2} + \gamma\right) T \leq \sum_t \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \leq (1+\eta) \sum_t \mathbf{m}^{(t)} \cdot \mathbf{p} + \frac{\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(1)})}{\eta} \leq (1+\eta)\frac{T}{2} + \frac{\log(n/|E|)}{\eta} .$$

Since $\eta = \gamma$ and $T = \lceil (2/\gamma^2) \ln(1/\varepsilon) \rceil$, the above inequality implies that the fraction of errors, $|E|/n$, is at most ε as desired.

3.7 Hard-core sets and the XOR Lemma

A boolean function $f : X \to \{0,1\}$, where X is a finite domain, is γ-*strongly hard*, for circuits of size S if for every circuit C of size at most S,

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

Here $x \in X$ is drawn uniformly at random, and $\gamma < 1/2$ is a parameter. For some parameter $\varepsilon > 0$, it is ε-*weakly hard* for circuits of size S if for every circuit C of size at most S, we have

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

Now given $f : \{0,1\}^n \to \{0,1\}$, define $f^{\oplus k} : \{0,1\}^{nk} \to \{0,1\}$ to be the boolean function obtained by dividing up the input nk-bit string into k blocks of n bits each in the natural way, applying f to each block in turn, and taking the XOR of the k outputs. Yao's XOR Lemma [64] shows that if f is ε-weakly hard against circuits of size S then $f^{\oplus k}$ is $\gamma + (1-\varepsilon)^k$-strongly hard for circuits of size $S\varepsilon^2\gamma^2/8$.

The original proofs were difficult but Impagliazzo [40] suggested a simpler proof that as a byproduct proves an interesting fact about weakly hard functions: there is a reasonably large subset (at least ε fraction of X) of inputs on which the function behaves like a strongly hard function, for somewhat smaller circuit size S'. This subset is called a *hard-core set*. For technical reasons, to prove such a result, it suffices to exhibit a "smooth" distribution $\mathbf{p}$ on X (precise definition given momentarily), such for any circuit C of size at most S', we have

$$\Pr_{x\sim\mathbf{p}}[C(x) = f(x)] \leq \frac{1}{2} + \gamma.$$

An ε-smooth distribution $\mathbf{p}$ (the same ε from the weak hardness assumption on f) is one that doesn't assign too much probability to any single input: $p_x \leq 1/(\varepsilon|X|)$ for any $x \in X$. Such a distribution be decomposed as a convex combination of probability distributions over subsets of size at least $\varepsilon|X|$.

Klivans and Servedio [49] observed that Impagliazzo's proof is an application of a boosting algorithm. The argument is as follows. We are given a boolean function f that is ε-weakly hard for circuits of size S. Assume for the sake of contradiction f is not γ-strongly hard on any smooth distribution for circuits of some size $S' < S$. Then for any smooth distribution, we can find a small circuit of size S' that calculates f correctly with probability better than $1/2+\gamma$ when inputs are drawn from the distribution. Treat this as a weak learning algorithm, and apply boosting. Boosting combines the small circuits of size S' found by the weak learning algorithm into a larger circuit that calculates f correctly with probability at least $1-\varepsilon$ on the uniform distribution on X, contradicting the fact f is ε-weakly hard, if we ensure that the size of the larger circuit is smaller than S. This can be done if the S' is set to $O(S/T)$, where T is the number of boosting rounds.

With this insight, the problem now boils down to designing boosting algorithms that (a) are able to deal with smooth distributions on inputs and (b) have a small number of boosting rounds. The lower the number of boosting rounds, the better circuit size bound we get for showing γ-strong hardness.

The third author [45] has shown how to construct such a boosting algorithm using the MW algorithm for restricted distributions (see Section 2.2). This boosting algorithm obtains the best known parameters in hard-core set constructions directly without having to resort to composing two different boosting algorithms as in [49]. This technique was extended in [8] to obtain *uniform* constructions of hard-core sets with the best known parameters.

We describe the boosting algorithm of [45] now. The main observation is that the set of all ε-smooth distributions is convex. Call this set $\mathcal{P}$. Then, exactly as in Section 3.6, the boosting algorithm simply runs the MW algorithm, with the only difference being that the distributions it generates are restricted to be in $\mathcal{P}$ using relative entropy projections, as in the algorithm of Section 2.2. We can now apply the same analysis as in Section 3.6. Following this analysis, let $E \subseteq X$ be the set of inputs on which the final majority hypotheses incorrectly computes f. Now we claim that $|E| < \varepsilon|X|$: otherwise, since the uniform distribution on E is ε-smooth, we obtain a contradiction for $T = \lceil (2/\gamma^2)\ln(1/\varepsilon)\rceil$ as before. Thus, the final majority hypothesis computes f correctly on at least a $1-\varepsilon$ fraction of inputs from X.

This immediately implies the following hard-core set existence result, which has the best known parameters to date:

Theorem 3.8. *Given a function $f : X \to \{0,1\}$ that is ε-weakly hard for circuits of size S, there is a subset of X of size at least $\varepsilon|X|$ on which f is γ-strongly hard for circuits of size*

$$S' = O\left(\frac{\gamma^2}{\log(1/\varepsilon)} S\right).$$

3.8 Hannan's algorithm and multiplicative weights

Perhaps the earliest decision-making algorithm which attains bounds similar to the MW algorithm is Hannan's algorithm [34], dubbed "follow the perturbed leader" by Kalai and Vempala [43]. This algorithm is given below.

Initialization: Pick random numbers $r_1, r_2, \ldots, r_n$, one for each decision.
For $t = 1, 2, \ldots, T$:

1. Choose the decision which minimizes the total cost including random initial cost:
$$i^{(t)} = \arg\min_i \left\{ L_i{}^{(t)} + r_i \right\} \tag{3.15}$$
where $L_i{}^{(t)} = \sum_{\tau < t} m_i{}^{(\tau)}$ is the total cost so far for the ith decision.
2. Observe the costs of the decisions $\mathbf{m}^{(t)}$.

Figure 3: Hannan's algorithm.

In this section we show that for a particular choice of random initialization numbers $\{r_i\}$, the algorithm above exactly reproduces the multiplicative weights algorithm, or more precisely the Hedge algorithm as in Section 2.1. This observation is due to Adam Kalai [42].

Theorem 3.9 ([42]). *Let $u_1, \ldots, u_n$ be n independent random numbers chosen uniformly from $[0,1]$, and consider the algorithm above with $r_i = \frac{1}{\eta} \ln \ln \frac{1}{u_i}$. Then for any decision j, we have*

$$\Pr\left[i^{(t)} = j\right] = \frac{e^{-\eta L_j{}^{(t)}}}{\sum_i e^{-\eta L_i{}^{(t)}}} .$$

Proof. By monotonicity of the exponential function, we have:

$$\begin{aligned}
\arg\min_i \left\{ L_i{}^{(t)} + \frac{1}{\eta} \ln \ln \frac{1}{u_i} \right\} &= \arg\min_i \left\{ \eta L_i{}^{(t)} + \ln \ln \frac{1}{u_i} \right\} \\
&= \arg\max_i \left\{ \frac{e^{-\eta L_i{}^{(t)}}}{\ln \frac{1}{u_i}} \right\} \\
&= \arg\max_i \left\{ \frac{w_i{}^{(t)}}{s_i} \right\} .
\end{aligned}$$

Where $w_i{}^{(t)} = e^{-\eta L_i{}^{(t)}}$ and $s_i = \ln \frac{1}{u_i}$ are independent exponentially distributed random variables with mean 1. The result now follows from Lemma 3.10 below. $\square$

Lemma 3.10. *Let $w_1,\dots,w_n$ be arbitrary non-negative real numbers, and let $s_1,\dots,s_n$ be independent exponential random variables with mean* 1*. Then*

$$\Pr\left[\arg\max_i \frac{w_i}{s_i} = j\right] \;=\; \frac{w_j}{\sum_i w_i}\,.$$

Proof. The probability density function of s_i is e^{-s_i}. Conditioning on the value of s_j, for a particular $i \neq j$ we have

$$\Pr\left[\frac{w_i}{s_i} \;\leq\; \frac{w_j}{s_j} \;\middle|\; s_j\right] \;=\; \int_{s_i=\frac{w_i s_j}{w_j}}^{\infty} e^{-s_i} ds_i \;=\; e^{-\frac{w_i s_j}{w_j}}\,.$$

Since all variables $s_1,s_2,\dots,s_n$ are independent, we have

$$\Pr\left[\forall i \neq j:\; \frac{w_i}{s_i} \leq \frac{w_j}{s_j} \;\middle|\; s_j\right] \;=\; e^{-\sum_{i\neq j}\frac{w_i s_j}{w_j}}\,.$$

Integrating to remove the conditioning on s_j, we have

$$\Pr\left[\forall i \neq j:\; \frac{w_i}{s_i} \leq \frac{w_j}{s_j}\right] \;=\; \int_{s_j=0}^{\infty} e^{-\sum_{i\neq j}\frac{w_i s_j}{w_j}} e^{-s_j} ds_j \;=\; \int_{s_j=0}^{\infty} e^{-s_j\frac{\sum_i w_i}{w_j}} ds_j \;=\; \frac{w_j}{\sum_i w_i}\,.$$

□

3.9 Online convex optimization

Online convex optimization is a very general framework that can be applied to many of the problems discussed in the applications section 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. For a much more detailed treatment of online learning techniques see [16].

In online convex optimization, we move from a discrete decision set to a continuous one. Specifically, the set of decisions is a convex, compact set $K \subseteq \mathbb{R}^n$. In each round $t = 1,2,\dots$, the online algorithm is required to choose a decision, i. e., point $\mathbf{p}^{(t)} \in K$. A convex loss function $f^{(t)}$ is presented, and the decision maker incurs a loss of $f^{(t)}(\mathbf{p}^{(t)})$. 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.

$$\mathcal{R}_T(\mathcal{A}) \;=\; \sum_{t=1}^{T} f^{(t)}(\mathbf{p}^{(t)}) - \min_{\mathbf{p}\in K}\sum_{t=1}^{T} f^{(t)}(\mathbf{p})\,.$$

The basic decision-making problem described in Section 2 with n discrete decisions is recovered as a special case of online convex optimization as follows. The convex set K is the n-dimensional simplex corresponding to the set of all distributions over the n decisions, and the payoff functions $f^{(t)}$ are defined as $f^{(t)}(\mathbf{p}^{(t)}) = \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$ given the cost vector $\mathbf{m}^{(t)}$. It also generalizes other online learning problems such as the online portfolio selection problem and online routing (see [35] for more discussion on applications). Zinkevich [66] gives algorithms for the general online convex optimization problem. More efficient algorithms for online convex optimization based on strong convexity of the loss functions have also been developed [37].

We now describe how to use the Multiplicative Weights algorithm to the online convex optimization problem for the special case where K is the n dimensional simplex of distributions on the coordinates. The advantage is that this algorithm has much better dependance on the dimension n than Zinkevich's algorithm (see [35] for more details). This algorithm has several applications such as in online portfolio selection [38]. Here is the algorithm. First, define

$$\rho \ := \ \max_{\mathbf{p}\in K}\max_{t} \|\nabla f^{(t)}(\mathbf{p})\|_\infty ,$$

where $\nabla f^{(t)}(\mathbf{p})$ is the (sub)gradient of the function $f^{(t)}$ at point $\mathbf{p}$. The parameter ρ is called the *width* of the problem. Then, run the standard Multiplicative Weights algorithm with $\eta = \sqrt{\ln(n)/T}$ and the costs defined as

$$\mathbf{m}^{(t)} \ := \ \frac{1}{\rho}\nabla f^{(t)}(\mathbf{p}^{(t)})$$

where $\mathbf{p}^{(t)}$ is the point played in round t. Note that for all t and all i, $|{m_i}^{(t)}| \leq 1$ as required by the MW algorithm.

Theorem 3.11. *After T rounds of applying the Multiplicative Weights algorithm to the online convex optimization framework, for any $\mathbf{p} \in K$ we have*

$$\sum_{t=1}^{T} f^{(t)}(\mathbf{p}^{(t)}) - \sum_{t=1}^{T} f^{(t)}(\mathbf{p}) \ \leq \ 2\rho\sqrt{\ln(n)T}\,.$$

Proof. If $f : K \to \mathbb{R}$ is a differentiable convex function, then for any two points $\mathbf{p},\mathbf{q} \in K$ we have

$$f(\mathbf{q}) \ \geq \ f(\mathbf{p}) + \nabla f(\mathbf{p})\cdot(\mathbf{q}-\mathbf{p})\,,$$

where $\nabla f(\mathbf{p})$ is the gradient of f at $\mathbf{p}$. Rearranging we get

$$f(\mathbf{p}) - f(\mathbf{q}) \ \leq \ \nabla f(\mathbf{p})\cdot(\mathbf{p}-\mathbf{q}). \tag{3.16}$$

Applying Corollary 2.2, we get that for any $\mathbf{p} \in K$,

$$\sum_{t=1}^{T} \mathbf{m}^{(t)}\cdot\mathbf{p}^{(t)} \ \leq \ \sum_{t=1}^{T} (\mathbf{m}^{(t)} + \eta|\mathbf{m}^{(t)}|)\cdot\mathbf{p} + \frac{\ln n}{\eta} \ \leq \ \sum_{t=1}^{T} \mathbf{m}^{(t)}\cdot\mathbf{p} + \eta + \frac{\ln n}{\eta}, \tag{3.17}$$

since $\|\mathbf{m}^{(t)}\|_\infty \leq 1$. Now we have

$$\begin{aligned}
\sum_t [f^{(t)}(\mathbf{p}^{(t)}) - f^{(t)}(\mathbf{p}) \ &\leq \ \sum_{t=1}^{T} \nabla f^{(t)}(\mathbf{p}^{(t)})\cdot(\mathbf{p}^{(t)}-\mathbf{p}) && \text{(from (3.16))}\\
&= \ \sum_{t=1}^{T} \rho\mathbf{m}^{(t)}\cdot(\mathbf{p}^{(t)}-\mathbf{p})\\
&\leq \ \eta\rho T + \frac{\rho\ln n}{\eta}. && \text{(from (3.17))}
\end{aligned}$$

Substituting $\eta = \sqrt{\ln n/T}$ completes the proof. □

3.10 Other applications

3.10.1 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 [31], 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 [47] use the PST framework (Section 3.3) to derive a more efficient 0.878-approximation algorithm for MAX-CUT than the original SDP-based method in Goemans-Williamson [31]. 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 ρ (see 3.3) is too high for certain SDPs of interest.

To be more precise, an SDP feasibility problem is given by:

$$\begin{aligned} \forall j = 1, 2, \ldots, m: \ \mathbf{A}_j \bullet \mathbf{X} \ &\geq \ b_j\,, \\ \mathbf{X} &\in \mathcal{P}\,. \end{aligned}$$

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 $\mathbb{R}^{n^2}$. The set $\mathcal{P} = \{\mathbf{X} \in \mathbb{R}^{n \times n} \mid \mathbf{X} \succeq 0,\ \mathbf{Tr}(\mathbf{X}) \leq 1\}$ is the set of all positive semidefinite matrices with trace bounded by 1.

The Plotkin-Shmoys-Tardos framework (see Section 3.3) is suitable for approximating SDPs since all constraints are linear, and the oracle given in (3.7) can be implemented efficiently by an eigenvector computation. To see this, note that the oracle needs to decide, given a probability distribution $\mathbf{p}$ on the constraints, if there exists an $\mathbf{X} \in \mathcal{P}$ such that $\sum_j p_j \mathbf{A}_j \bullet \mathbf{X} \geq \sum_j p_j b_j$. This can be implemented by solving the following optimization problem:

$$\max_{\mathbf{x} \in \mathcal{P}} : \quad \sum_j p_i \mathbf{A}_j \bullet \mathbf{X}\,.$$

It is easily checked that an optimal solution to the above optimization problem is given by the matrix $\mathbf{X} = \mathbf{v}\mathbf{v}^\top$ where $\mathbf{v}$ is unit eigenvector corresponding to the largest eigenvalue of the matrix $\sum_j p_j \mathbf{A}_j$.

The Klein-Lu approach was of limited uses in many cases because it does not do too well when the additive error ε 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 ε 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 Section 3.3, 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 [3]. For several families of SDPs we obtain the best running time known.

More recently Arora and Kale [5] 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.

Yet another approach for approximately solving SDPs is by reducing the SDP into a maximization problem of a single concave function over the PSD cone. The latter problem can be approximated efficiently via an iterative greedy method. The resulting algorithm is extremely similar to the MW-based algorithm of [3], however its analysis is very different, see [36] for more details.

3.10.2 Approximating graph separators

A recent application of the multiplicative weights method is a combinatorial algorithm for approximating the SPARSEST CUT of graphs [4]. This is a fundamental graph partitioning problem. Given a graph $G = (V,E)$, the expansion of a cut $(S,\bar{S})$ where $S \subseteq V$ and $\bar{S} = V \setminus S$, is defined to be

$$\frac{|E(S,\bar{S})|}{\min\{|S|,|\bar{S}|\}}.$$

Here, $E(S,\bar{S})$ is the set of edges with one end point in S and the other in $\bar{S}$. The SPARSEST CUT problem is to find the cut in the input graph of minimum expansion. 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 [7] 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 [4], 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 3.3. The oracle for this purpose is implemented using a variety of techniques: the multicommodity flow algorithm of Garg and Könemann [29, 30] (and its subsequent improvement by Fleischer [24]), eigenvalue computations, and graph sparsification algorithms of Benczúr and Karger [9] based on random sampling.

3.10.3 Multiplicative weight algorithms in geometry

The multiplicative weight idea has been used several times in computational geometry. Chazelle [19] (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 3.5.

The geometric applications consist of derandomizing a natural 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 3.5. Clarkson used this idea to give a deterministic algorithm for Linear Programming [20, 21]. Following Clarkson, Brönnimann and Goodrich use similar methods to find Set Covers for hypergraphs with small VC dimension [11].

The MW algorithm was also used in the context of geometric embeddings of finite metric spaces, specifically, embedding negative-type metrics (i. e., a set of points in Euclidean space such that the squared Euclidean distance between them also forms a metric) into ℓ_1. Such embeddings are important for approximating the important non-uniform SPARSEST CUT PROBLEM.

The approximation algorithm for SPARSEST CUT in Arora et al. [7] involves a "Structure Theorem." This structure theorem was interpreted by Chawla et al. [18] as saying that any n-point negative-type metric with maximum distance $O(1)$ and average distance $\Omega(1)$ can be embedded into ℓ_1 such that *average* ℓ_1 distance is $\Omega(1/\sqrt{\log n})$. Then they used the MW algorithm to construct an embedding into ℓ_1 in which *every* pair of points that have negative-type metric distance $\Omega(1)$ have ℓ_1 distance that is off by at most an $O(\sqrt{\log n})$ factor of the original. Using a similar idea for other distances and combining the resulting embeddings they obtained an embedding of the negative-type metric into ℓ_1 in which every distance distorts by at most a factor $O(\log^{3/4} n)$. Arora et al. [6] gave a more complicated construction to improve the distortion bound to $O(\sqrt{\log(n)}\log\log n)$, leading to a $O(\sqrt{\log(n)}\log\log n)$-approximation for non-uniform sparsest cut.

3.11 Design of competitive online algorithms

Starting with the work of Alon, Awerbuch, Azar, Buchbinder and Naor [1], a number of competitive online algorithms have been developed using an elegant primal-dual approach which involves multiplicative weight updates. While the analysis of their algorithms seems to be beyond our general framework, we briefly mention this work without going into many details. We refer the readers to the survey by Buchbinder and Naor [14] for an extensive discussion of the topic.

Several online problems such as the ski rental problem, caching, load balancing, ad auctions, etc. can be cast (in their fractional form) as a linear program with non-negative coefficients in the constraints and cost, where either the constraints or variables arrive online one by one. The online problem is to maintain a feasible solution at all times with a bounded competitive ratio, i. e., ensuring that the cost of the solution maintained is bounded in terms of the cost of the optimal solution in each round. The main difficulty comes from the requirement that the solution maintained is monotonic in some sense (for example, the variables are never allowed to decrease).

Buchbinder and Naor [15] give a algorithm based on the primal-dual method that obtains good competitive ratios in this scenario. At the heart is a multiplicative weight update rule. Imagine we have a covering LP, i. e., all constraints are of the form $\mathbf{a}\cdot\mathbf{x} \geq 1$, where $\mathbf{a}$ has non-negative coefficients, and $\mathbf{x}$ is the vector of variables. The cost function has non-negative coefficients as well. Constraints arrive one at a time in each round and must be satisfied by the current solution. The requirement is that no variable can decrease from round to round.

In every round, the algorithm increases primal variables in the new constraint using multiplicative updates and the corresponding dual variable additively (so essentially, each primal variable is the exponential of the dual constraint that it corresponds to, much like the Plotkin-Shmoys-Tardos algorithms of Section 3.3). This is done until the constraint gets satisfied. Clearly, we maintain a feasible solution in each round, and the variables never decrease. The analysis goes by bounding the increase in the primal cost in terms of the dual cost (this is an easy consequence of the multiplicative update, in fact, the multiplicative update can be derived using this requirement). The competitive ratio is obtained by showing that the dual solution generated simultaneously, while infeasible, is not far from being feasible, i. e., scaling down the variables by a certain factor makes it feasible. This gives us a bound on the competitive ratio via weak duality.

4 Lower bound

Can our analysis of the Multiplicative Weights algorithm be improved? This section shows that the answer is no, not only for the MW algorithm itself, but for any algorithm which operates in the online setting we have described. A similar lower bound was obtained by Klein and Young [48].

Technically, we prove the following:

Theorem 4.1. *For any online decision-making algorithm with $n \geq 2$ decisions, there exists a sequence of cost vectors $\mathbf{m}^{(1)}, \mathbf{m}^{(2)}, \ldots, \mathbf{m}^{(T)} \in \{0,1\}^n$ such that $\min_i \sum_t {m_i}^{(t)} = \Omega(T)$, and if the sequence of distributions over decisions produced by the algorithm are $\mathbf{p}^{(1)}, \mathbf{p}^{(2)}, \ldots, \mathbf{p}^{(T)}$, then we have*

$$\sum_t \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;\geq\; \min_i \sum_{t=1}^{T} {m_i}^{(t)} + \Omega(\sqrt{T \ln n})\,.$$

Since in the theorem above we have $\min_i \sum_{t=1}^{T} {m_i}^{(t)} = \Omega(T)$, Theorem 2.1 implies that by choosing the optimal value of η, viz. $\eta = \Theta\big(\sqrt{\log n / T}\big)$ we have

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;\leq\; \min_i \sum_{t=1}^{T} {m_i}^{(t)} + O(\sqrt{T \ln n})\,.$$

Hence our analysis is tight up to constants in the additive error term. Moreover, the above lower bound applies to *any* algorithm, efficient or not.

Proof. The proof is via the probabilistic method. We construct a distribution over the costs so that the required bound is obtained in expectation. Interestingly, the distribution is independent of the actual algorithm used.

We now specify the costs of the decisions. The cost of decision 1 is set to $1/2$ for all rounds t. For any decision $i > 1$, we construct its cost via the following random process: in each iteration t, choose its cost to be either one or zero uniformly at random, i. e., ${m_i}^{(t)} \in \{0,1\}$ with probability of each outcome being $1/2$.

The expected cost of each decision is $1/2$. Hence, the expected cost of the chosen decision is also $1/2$ irrespective of the algorithm's distribution $\mathbf{p}^{(t)}$, and hence:

$$\mathbf{E}\left[\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}\right] \;=\; \frac{T}{2}. \tag{4.1}$$

For every decision i, define $X_i = \sum_{t=1}^{T} {m_i}^{(t)}$. Note that $X_1 = T/2$, and for $i > 1$, we have $X_i \sim B(T, 1/2)$, where $B(T, 1/2)$ is the binomial distribution with T trials and both outcomes equally likely.

We use the following standard concentration lemma (see Proposition 7.3.2 in [54]):

Lemma 4.2. *Let $X \sim B(T, 1/2)$ be a binomial random variable with T trials and probability $1/2$. Then for $t \in [0, T/8]$:*

$$\Pr\left[X \le \frac{T}{2} - t\right] \;\ge\; \frac{1}{15} e^{-16t^2/T}.$$

We claim that the expectation of the cost of the best decision, viz., $\mathbf{E}[\min_i X_i]$, in our construction is $T/2 - \Omega(\sqrt{T \log n})$. Let $t = (1/4)\sqrt{T \ln(n-1)}$. We have

$$\Pr\left[\min_i X_i > \frac{T}{2} - t\right] \;=\; \prod_{i=2}^{n} \Pr\left[X_i > \frac{T}{2} - t\right] \;\le\; \left(1 - \frac{1}{15} e^{-16t^2/T}\right)^{n-1} \;\le\; e^{-1/15} \;<\; 0.95\,.$$

The first equality above is by the independence of the X_i, and the first inequality is by Lemma 4.2. Thus, with probability at least $1 - 0.95 = 0.05$, $\min_i X_i \le (T/2) - t$. Since $\min_i X_i$ is always at least $T/2$, we get

$$\mathbf{E}[\min_i X_i] \;\ge\; 0.95 \cdot \frac{T}{2} + 0.05 \cdot \left(\frac{T}{2} + t\right) \;=\; \frac{T}{2} + \frac{1}{80}\sqrt{T \ln(n-1)}\,.$$

It follows that

$$\mathbf{E}\left[\min_i \sum_{t=1}^{T} {m_i}^{(t)}\right] \;\ge\; \frac{T}{2} - \frac{1}{80}\sqrt{T \ln(n-1)}. \tag{4.2}$$

From (4.1) and (4.2), we have

$$\mathbf{E}\left[\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} - \min_i \sum_{t=1}^{T} {m_i}^{(t)}\right] \;\ge\; \frac{1}{80}\sqrt{T \ln(n-1)}\,.$$

Since $\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} - \min_i \sum_{t=1}^{T} {m_i}^{(t)} \le T$, by Markov's inequality we conclude that

$$\Pr\left[\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} - \min_i \sum_{t=1}^{T} {m_i}^{(t)} > \frac{1}{160}\sqrt{T \ln(n-1)}\right] \ge \frac{1}{160}\sqrt{\frac{\ln(n-1)}{T}}\,.$$

By the Hoeffding bound [2], for any decision $i > 1$, we have

$$\Pr\left[\sum_{t=1}^{T} {m_i}^{(t)} < \frac{T}{4}\right] \;\le\; \exp(-T/32)\,.$$

By the union bound, we have

$$\Pr\left[\exists i : \sum_{t=1}^{T} {m_i}^{(t)} < \frac{T}{4}\right] \;\le\; n \exp(-T/32) \;<\; \frac{1}{160}\sqrt{\frac{\ln(n-1)}{T}}\,,$$

for large enough T. This implies that there exists a sequence $\mathbf{m}^{(1)}, \mathbf{m}^{(2)}, \ldots, \mathbf{m}^{(T)}$ for which

$$\min_i \sum_{t=1}^{T} m_i{}^{(t)} \geq T/4$$

and

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} - \min_i \sum_{t=1}^{T} m_i{}^{(t)} \;\geq\; \frac{1}{160}\sqrt{T \ln(n-1)}\,.$$

□

5 The Matrix Multiplicative Weights algorithm

In the preceding sections, we considered an online decision making problem. We refer to that setting as the *basic* or *scalar* case. Now we briefly consider a different online decision making problem, 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 cost vectors to cost 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 [5], derandomizing constructions of expander graphs, and obtaining bounds on the sample complexity for a learning problem in quantum computing. The celebrated result of Jain et al. [41] showing that **QIP =PSPACE** relied on the Matrix MW algorithm. Here **QIP** is the set of all languages which have quantum interactive proofs. These applications are unfortunately beyond the scope of this survey; please see the third author's Ph. D. thesis [44] and Jain et al. [41] for details. The algorithm given here is from a paper of Arora and Kale [5]. A very similar algorithm was discovered independently slightly earlier by Warmuth and Kuzmin [63], and is based on the even earlier work of Tsuda, Rätsch, and Warmuth [62].

We stick with our basic decision-making scenario but decisions now correspond to unit vectors $\mathbf{v}$ in $\mathbb{S}^{n-1}$, the unit sphere in $\mathbb{R}^n$. As in the basic case, in every round, our task is to pick a decision $\mathbf{v} \in \mathbb{S}^{n-1}$. At this point, the costs of all decisions are revealed by nature. These costs are not arbitrary, but they are correlated in the following way. A *cost matrix* $\mathbf{M} \in \mathbb{R}^{n \times n}$ is revealed, and the cost of a decision $\mathbf{v}$ is then $\mathbf{v}^\top \mathbf{M} \mathbf{v}$. We assume that the costs of all decisions lie in $[-1,1]$. 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 the range $[-1,1]$.

This game is repeated over a number of rounds. Let $t = 1, 2, \ldots, T$ denote the current round. In each round t, we select a distribution $\mathcal{D}^{(t)}$ over the set of decisions $\mathbb{S}^{n-1}$, and select a decision $\mathbf{v}$ randomly from it (and use his advised course of action). At this point, the costs of all decisions are revealed by nature via the cost matrix $\mathbf{M}^{(t)}$. The expected cost to the algorithm for choosing the distribution $\mathcal{D}^{(t)}$ is

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

Recall that we are using 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 $\mathbf{Tr}(\mathbf{A})$ denote the trace of a matrix $\mathbf{A}$. Then, $\mathbf{Tr}(\mathbf{P}^{(t)}) = 1$, again because for all $\mathbf{v}$, we have $\mathbf{Tr}(\mathbf{v}\mathbf{v}^\top) = \|\mathbf{v}\|^2 = 1$. 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 cost to the algorithm, and all the information required for computing the expected cost 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 $\mathbf{P}^{(t)}$. The eigendecomposition of $\mathbf{P}^{(t)} = \sum_{i=1}^n \lambda_i \mathbf{v}_i \mathbf{v}_i^\top$, where λ_i is an eigenvalue corresponding to the unit eigenvector $\mathbf{v}_i$, gives one such distribution (a discrete one): vector $\mathbf{v}_i$ gets probability λ_i. We then observe the cost matrix $\mathbf{M}^{(t)}$ revealed by nature, and suffer the expected cost $\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}$. After T rounds, the total expected cost is $\sum_{t=1}^T \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}$, while the best fixed decision 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 cost is exactly the minimum eigenvalue of $\sum_{t=1}^T \mathbf{M}^{(t)}$, denoted by $\lambda_n(\sum_{t=1}^T \mathbf{M}^{(t)})$. Our goal is to design an online algorithm whose total expected cost over the T rounds is not much more than the cost of the best decision.

Consider the following generalization of the basic MW algorithm, called the Matrix Multiplicative Weights algorithm (Matrix MW). It uses the notion of matrix exponential, $\exp(\mathbf{A}) := \sum_{i=0}^\infty \mathbf{A}^i / i!$. The key point here is that regardless of the matrix $\mathbf{A}$, its exponential $\exp(\mathbf{A})$ is always positive definite. Note that in case all matrices involved are diagonal, the Matrix Multiplicative Weights algorithm exactly reduces to the Hedge algorithm.

Matrix Multiplicative Weights algorithm

Initialization: Fix an $\eta \leq \frac{1}{2}$. Initialize the weight matrix $\mathbf{W}^{(1)} = \mathbf{I}_n$.
For $t = 1, 2, \ldots, T$:

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

$$\mathbf{W}^{(t+1)} = \exp\left(-\eta \textstyle\sum_{\tau=1}^t \mathbf{M}^{(\tau)}\right).$$

Figure 4: The Matrix Multiplicative Weights algorithm.

The following theorem bounds the total expected cost of the Matrix Multiplicative Weights algorithm (given in Figure 4) in terms of the cost of the best fixed decision. This theorem is completely analogous to Theorem 2.3, and in fact, Theorem 2.3 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 2.3, and uses $\Phi^{(t)}$ as a potential function. The analysis is based on the matrix analogue of the real number inequality $\exp(-\eta x) \le 1 - \eta x + \eta^2 x^2$ for $|\eta x| \le 1$: if $\mathbf{X}$ is a matrix such that $\|\eta \mathbf{X}\| \le 1$, then

$$-\exp(-\eta \mathbf{X}) \preceq \mathbf{I} - \eta \mathbf{X} + \eta^2 \mathbf{X}^2 .$$

Also, we need an additional inequality from statistical mechanics, the Golden-Thompson inequality [32, 61], which states that for two matrices $\mathbf{A}$ and $\mathbf{B}$, we have

$$\mathbf{Tr}(\exp(\mathbf{A}+\mathbf{B})) \le \mathbf{Tr}(\exp(\mathbf{A})\exp(\mathbf{B})) .$$

See [44, 5] for further details.

Theorem 5.1. *In the given setup, the Matrix Multiplicative Weights algorithm guarantees that after T rounds, for any decision $\mathbf{v}$, we have*

$$\sum_{t=1}^{T} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \le \sum_{t=1}^{T} \mathbf{v}^\top \mathbf{M}^{(t)} \mathbf{v} + \eta \sum_{t=1}^{T} (\mathbf{M}^{(t)})^2 \bullet \mathbf{P}^{(t)} + \frac{\ln n}{\eta}. \tag{5.1}$$

5.1 Applications in solving SDPs

In this section, we mention the application of the Matrix MW algorithm to solving SDPs without going into any details. We refer the interested reader to [44, 5] for further details and other applications.

Consider the following quite general form of feasibility problem using SDP:

$$\begin{aligned} \forall j = 1,2,\ldots,m: \ \mathbf{A}_j \bullet \mathbf{X} &\ge 0 , \\ \mathbf{Tr}(\mathbf{X}) &= 1 , \\ \mathbf{X} &\succeq \mathbf{0} . \end{aligned} \tag{5.2}$$

Note if all $\mathbf{A}_j$ matrices and $\mathbf{X}$ were diagonal, then this exactly reduces to the LP (3.1).

Now suppose, as in Section 3.1, that there exists a large-margin solution, i. e., a density matrix $\mathbf{X}^\star$ such that for all j, we have $\mathbf{A}_j \bullet \mathbf{X}^\star \ge \varepsilon$. Then just as in Section 3.1, we can use the Matrix MW algorithm to find a good solution, i. e., a density matrix $\mathbf{X}$ such that for all j we have $\mathbf{A}_j \bullet \mathbf{X} \ge 0$. We now need to define $\rho = \max_j \|\mathbf{A}_j\|$.

We run the Matrix MW algorithm (in the gain form). In each round, we set $\mathbf{X}$ to be the current density matrix $\mathbf{P}^{(t)}$, and check if it is a good solution. If not, and there is a constraint j such that $\mathbf{A}_j \bullet \mathbf{X} < 0$, then we set $\mathbf{M}^{(t)} = (1/\rho)\mathbf{A}_j$. Note that in case all the matrices involved in the SDP are diagonal, then this algorithm reduces to the Winnow algorithm of Section 3.1.[2]

The analysis is exactly the same as in Section 3.1. This analysis (using the gain form of Theorem 5.1 corresponding to Theorem 2.5), implies that in at most $\lceil 4\rho^2 \ln n/\varepsilon^2 \rceil$ iterations, we find a good solution.

This technique was used in [5] to obtain significantly faster algorithms than previously known for approximating several combinatorial optimization problems such as SPARSEST CUT, BALANCED SEPARATOR (both in directed and undirected graphs), MIN UNCUT, and MIN 2CNF DELETION. This also gives the first near-linear time algorithm for solving the MAX CUT SDP of [31] to any constant approximation factor.

[2] A minor difference being that we get the version of Winnow based on the Hedge algorithm, rather than the MW algorithm.

Acknowledgments

This work benefitted greatly from discussions with Avrim Blum, Bernard Chazelle, Russell Impagliazzo, Satish Rao, Robert E. Schapire, and Manfred K. Warmuth. We also want to thank the editors and the anonymous referees at *Theory of Computing* for very carefully reading our manuscript and coming up with numerous helpful suggestions to improve the presentation. In addition, since a preprint of this paper was available on the Internet for several years prior to publication, a number of people have read it and suggested improvements; we cannot list them all but we thank them for their suggestions.

References

[1] NOGA ALON, BARUCH AWERBUCH, YOSSI AZAR, NIV BUCHBINDER, AND JOSEPH (SEFFI) NAOR: The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009. Preliminary version in STOC'03. [doi:10.1137/060661946] 152

[2] NOGA ALON AND JOEL H. SPENCER: *The Probabilistic Method*. John Wiley, 1992. [doi:10.1002/9780470277331] 154

[3] SANJEEV ARORA, ELAD HAZAN, AND SATYEN KALE: Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *Proc. 46th FOCS*, pp. 339–348. IEEE Comp. Soc. Press, 2005. [doi:10.1109/SFCS.2005.35] 151

[4] SANJEEV ARORA, ELAD HAZAN, AND SATYEN KALE: $O(\sqrt{\log n})$ approximation to SPARSEST CUT in $\tilde{O}(n^2)$ time. *SIAM J. Comput.*, 39:1748–1771, 2010. Preliminary version in FOCS'04. [doi:10.1137/080731049] 151

[5] SANJEEV ARORA AND SATYEN KALE: A combinatorial, primal-dual approach to semidefinite programs. In *Proc. 39th STOC*, pp. 227–236. ACM Press, 2007. [doi:10.1145/1250790.1250823] 151, 155, 157

[6] SANJEEV ARORA, JAMES R. LEE, AND ASSAF NAOR: Euclidean distortion and the sparsest cut. *J. Amer. Math. Soc.*, 21:1–21, 2008. Preliminary version in STOC'05. [doi:10.1090/S0894-0347-07-00573-5] 152

[7] SANJEEV ARORA, SATISH RAO, AND UMESH VAZIRANI: Expander flows, geometric embeddings, and graph partitioning. *J. ACM*, 56(2):5:1–5:37, 2009. Preliminary version in 36th STOC (2004). [doi:10.1145/1502793.1502794] 151, 152

[8] BOAZ BARAK, MORITZ HARDT, AND SATYEN KALE: The uniform hardcore lemma via approximate Bregman projections. In *Proc. 20th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'09)*, pp. 1193–1200. ACM Press, 2009. [ACM DL] [SIAM PDF]. 146

[9] ANDRÁS A. BENCZÚR AND DAVID R. KARGER: Approximating $s-t$ minimum cuts in $\tilde{O}(n^2)$ time. In *Proc. 28th STOC*, pp. 47–55. ACM Press, 1996. [doi:10.1145/237814.237827] 151

[10] AVRIM BLUM: On-line algorithms in machine learning. In A. FIAT AND G. WOEGINGER, editors, *Online Algorithms: The State of the Art*, pp. 306–325. LNCS 1442, 1998. [doi:10.1007/BFb0029575] 122, 123

[11] H. BRÖNNIMANN AND M.T. GOODRICH: Almost optimal set covers in finite VC-dimension. *Discrete Comput. Geom.*, 14:463–479, 1995. Preliminary version in 10th Ann. Symp. Comp. Geom. (SCG'94). [doi:10.1007/BF02570718] 123, 152

[12] GEORGE W. BROWN: Iterative solution of games by fictitious play. In T. C. KOOPMANS, editor, *Analysis of Production and Allocation*, pp. 374–376. Wiley, 1951. 123

[13] GEORGE W. BROWN AND JOHN VON NEUMANN: Solutions of games by differential equations. *Annals of Mathematics Studies*, 24:73–79, 1950. 123

[14] NIV BUCHBINDER AND JOSEPH (SEFFI) NAOR: The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009. [doi:10.1561/0400000024] 152

[15] NIV BUCHBINDER AND JOSEPH (SEFFI) NAOR: Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009. [doi:10.1287/moor.1080.0363] 152

[16] NICOLÒ CESA-BIANCHI AND GÁBOR LUGOSI: *Prediction, Learning, and Games*. Cambridge University Press, 2006. [doi:10.1017/CBO9780511546921] 148

[17] NICOLÒ CESA-BIANCHI, YISHAY MANSOUR, AND GILLES STOLTZ: Improved second-order bounds for prediction with expert advice. *Machine Learning*, 66(2-3):321–352, 2007. [doi:10.1007/s10994-006-5001-7] 126

[18] SUCHI CHAWLA, ANUPAM GUPTA, AND HARALD RÄCKE: Embeddings of negative-type metrics and an improved approximation to generalized sparsest cut. *ACM Trans. Algorithms*, 4:22:1–22:18, 2008. Preliminary version in SODA'05. [doi:10.1145/1361192.1361199] 152

[19] BERNARD CHAZELLE: *The Discrepancy Method—Randomness and Complexity*. Cambridge University Press, Cambridge, 2000. [doi:10.1007/3-540-49381-6_1] 151

[20] KENNETH L. CLARKSON: A Las Vegas algorithm for linear programming when the dimension is small. In *Proc. 29th FOCS*, pp. 452–456. IEEE Comp. Soc. Press, 1988. [doi:10.1109/SFCS.1988.21961] 123, 152

[21] KENNETH L. CLARKSON: A Las Vegas algorithm for linear and integer programming when the dimension is small. *J. ACM*, 42:488–499, 1995. [doi:10.1145/201019.201036] 123, 152

[22] THOMAS M. COVER: Universal data compression and portfolio selection. In *Proc. 37th FOCS*, pp. 534–538. IEEE Comp. Soc. Press, 1996. [doi:10.1109/SFCS.1996.548512] 123

[23] URIEL FEIGE: A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, July 1998. [doi:10.1145/285055.285059] 143

[24] LISA K. FLEISCHER: Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal of Discrete Mathematics*, 13(4):505–520, 2000. Preliminary version in FOCS'99. [doi:10.1137/S0895480199355754] 122, 151

[25] DEAN P. FOSTER AND RAKESH VOHRA: Regret in the on-line decision problem. *Games and Economic Behaviour*, 29:7–35, 1999. [doi:10.1006/game.1999.0740] 123

[26] YOAV FREUND AND ROBERT E. SCHAPIRE: A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. System Sci.*, 55(1):119–139, 1997. [doi:10.1006/jcss.1997.1504] 122, 123, 128, 144

[27] YOAV FREUND AND ROBERT E. SCHAPIRE: Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999. [doi:10.1006/game.1999.0738] 124, 133

[28] NAVEEN GARG AND ROHIT KHANDEKAR: Fractional covering with upper bounds on the variables: Solving LPs with negative entries. In *Proc. 12th Ann. Europ. Symp. on Algorithms (ESA'04)*, pp. 371–382, 2004. [doi:10.1007/978-3-540-30140-0_34] 123

[29] NAVEEN GARG AND JOCHEN KÖNEMANN: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proc. 39th FOCS*, pp. 300–309. IEEE Comp. Soc. Press, 1998. [doi:10.1109/SFCS.1998.743463] 122, 139, 151

[30] NAVEEN GARG AND JOCHEN KÖNEMANN: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37:630–652, 2007. [doi:10.1137/S0097539704446232] 122, 139, 151

[31] 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. [doi:10.1145/227683.227684] 150, 157

[32] SIDNEY GOLDEN: Lower Bounds for the Helmholtz Function. *Physical Review*, 137:1127–1128, February 1965. [doi:10.1103/PhysRev.137.B1127] 157

[33] MICHAEL D. GRIGORIADIS AND LEONID G. KHACHIYAN: A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18:53–58, 1995. [doi:10.1016/0167-6377(95)00032-0] 123

[34] JAMES HANNAN: Approximation to Bayes risk in repeated plays. In M. DRESHER, A. W. TUCKER, AND P. WOLFE, editors, *Contributions to the Theory of Games*, volume 3, pp. 97–139. Princeton University Press, 1957. 123, 133, 147

[35] ELAD HAZAN: *Efficient Algorithms for Online Convex Optimization and their Applications*. PhD thesis, Princeton University, 2006. Technical Report TR-766-06. 148, 149

[36] ELAD HAZAN: Sparse approximate solutions to semidefinite programs. In *Proc. 8th Latin Amer. Symp. on Theoretical Informatics (LATIN'08)*, pp. 306–316, 2008. [doi:10.1007/978-3-540-78773-0_27] 151

[37] ELAD HAZAN, A. AGARWAL, AND SATYEN KALE: Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007. [doi:10.1007/s10994-007-5016-8] 149

[38] DAVID P. HELMBOLD, ROBERT E. SCHAPIRE, YORAM SINGER, AND MANFRED K. WARMUTH: On-line portfolio selection using multiplicative updates. *Mathematical Finance*, 8:325–347, 1998. Preliminary version in 13th Int. Conf. on Machine Learning (ICML'96). [doi:10.1111/1467-9965.00058] 149

[39] MARK HERBSTER AND MANFRED K. WARMUTH: Tracking the best expert. *Machine Learning*, 32(2):151–178, 1998. [doi:10.1023/A:1007424614876] 131

[40] RUSSELL IMPAGLIAZZO: Hard-core distributions for somewhat hard problems. In *Proc. 36th FOCS*, pp. 538–545. IEEE Comp. Soc. Press, 1995. [doi:10.1109/SFCS.1995.492584] 122, 145

[41] RAHUL JAIN, ZHENGFENG JI, SARVAGYA UPADHYAY, AND JOHN WATROUS: QIP = PSPACE. *J. ACM*, 58:30:1–30:27, 2011. Preliminary version in STOC'10. [doi:10.1145/2049697.2049704] 155

[42] ADAM KALAI: Personal communication with Manfred Warmuth, 2009. 147

[43] ADAM KALAI AND SANTOSH VEMPALA: Efficient algorithms for online decision problems. *J. Comput. System Sci.*, 71:291–307, 2005. Preliminary version in 16th COLT (2003). [doi:10.1016/j.jcss.2004.10.016] 123, 147

[44] SATYEN KALE: *Efficient Algorithms Using The Multiplicative Weights Update Method*. PhD thesis, Princeton University, 2006. Technical Report TR-804-07. 155, 157

[45] SATYEN KALE: Boosting and hard-core set constructions: A simplified approach. Technical Report TR07-131, Electron. Colloq. on Comput. Complexity (ECCC), 2007. [ECCC:TR07-131] 146

[46] ROHIT KHANDEKAR: *Lagrangian Relaxation based Algorithms for Convex Programming Problems*. PhD thesis, Indian Institute of Technology, Delhi, 2004. 122

[47] PHILIP N. KLEIN AND HSUEH-I LU: Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *Proc. 28th STOC*, pp. 338–347. ACM Press, 1996. [doi:10.1145/237814.237980] 150

[48] PHILIP N. KLEIN AND NEAL YOUNG: On the number of iterations for Dantzig-Wolfe optimization and packing-covering approximation algorithms. In *Proc. 7th Conf. on Integer Programming and Combinatorial Optimization (IPCO'99)*, LNCS 1610, pp. 320–327. Springer, 1999. [ACM DL]. 153

[49] ADAM R. KLIVANS AND ROCCO A. SERVEDIO: Boosting and hard-core set construction. *Machine Learning*, 51(3):217–238, 2003. [doi:10.1023/A:1022949332276] 123, 146

[50] NICK LITTLESTONE: Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1987. Preliminary version in FOCS'87. [doi:10.1007/BF00116827] 123, 132

[51] NICK LITTLESTONE: *Mistake bounds and logarithmic linear-threshold learning algorithms*. PhD thesis, University of California at Santa Cruz, Santa Cruz, CA, USA, 1989. 123

[52] NICK LITTLESTONE AND MANFRED K. WARMUTH: The weighted majority algorithm. *Inform. and Comput.*, 108(2):212–261, 1994. [doi:10.1006/inco.1994.1009] 123

[53] MICHAEL LUBY AND NOAM NISAN: A parallel approximation algorithm for positive linear programming. In *Proc. 25th STOC*, pp. 448–457. ACM Press, 1993. [doi:10.1145/167088.167211] 122

[54] JIŘI MATOUŠEK AND JAN VONDRAK: The probabilistic method, 2008. Lecture notes at Charles University. 154

[55] M. MINSKY AND S. PAPERT: *Perceptrons*. MIT Press, 1969. 123

[56] SERGE A. PLOTKIN, DAVID B. SHMOYS, AND ÉVA TARDOS: Fast approximation algorithm for fractional packing and covering problems. *Math. Oper. Res.*, 20:257–301, 1995. Preliminary version in FOCS'91. [doi:10.1287/moor.20.2.257] 122, 135, 136

[57] PRABHAKAR RAGHAVAN: Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *J. Comput. System Sci.*, 37:130–143, 1988. Preliminary version in FOCS'86. [doi:10.1016/0022-0000(88)90003-7] 142

[58] PRABHAKAR RAGHAVAN AND CLARK D. THOMPSON: Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987. [doi:10.1007/BF02579324] 142

[59] JULIA ROBINSON: An iterative method of solving a game. *Annals of Mathematics*, 54:296–301, 1951. [doi:10.2307/1969530] 123

[60] ROBERT E. SCHAPIRE: The strength of weak learnability. *Machine Learning*, 5:197–227, 1990. [doi:10.1007/BF00116037] 144

[61] COLIN J. THOMPSON: Inequality with applications in statistical mechanics. *J. Math. Phys.*, 6(11):1812–1823, 1965. [doi:10.1063/1.1704727] 157

[62] KOJI TSUDA, GUNNAR RÄTSCH, AND MANFRED K. WARMUTH: Matrix exponentiated gradient updates for on-line learning and Bregman projection. *J. Machine Learning Research (JMLR)*, 6:995–1018, 2005. [JMLR PDF] . 155

[63] MANFRED K. WARMUTH AND DIMA KUZMIN: Online variance minimization. *Machine Learning*, 87:1–32, 2012. Preliminary version in COLT'06. [doi:10.1007/s10994-011-5269-0] 155

[64] ANDREW C. YAO: Theory and applications of trapdoor functions (extended abstract). In *Proc. 23rd FOCS*, pp. 80–91. IEEE Comp. Soc. Press, 1982. [doi:10.1109/SFCS.1982.88] 145

[65] NEAL E. YOUNG: Randomized rounding without solving the linear program. In *Proc. 6th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'95)*, pp. 170–178. ACM Press, 1995. [ACM DL]. 122, 123, 124, 142

[66] MARTIN ZINKEVICH: Online convex programming and generalized infinitesimal gradient ascent. In *Proc. 20th Int. Conf. on Machine Learning (ICML'03)*, pp. 928–936, 2003. 148

AUTHORS

Sanjeev Arora
Professor
Princeton University, NJ
arora@princeton.edu
`http://www.cs.princeton.edu/~arora`

Elad Hazan
Assistant Professor
Technion, Israel Institute of Technology
ehazan@ie.technion.ac.il
`http://ie.technion.ac.il/~ehazan`

Satyen Kale
Research Staff Member
IBM Watson Research Center, NY
sckale@us.ibm.com
`http://www.satyenkale.com`

ABOUT THE AUTHORS

SANJEEV ARORA is Charles C. Fitzmorris Professor of computer science at Princeton University. He was the founding director and lead PI of the Center for Computational Intractability. He has received the Gödel prize twice (as cowinner), for his works on Probabilistically Checkable Proofs (PCP Theorem) and Approximation Schemes for Geometric problems such as Euclidean TSP.

ELAD HAZAN completed undergraduate studies in Tel Aviv University and received a Ph. D. from Princeton University in 2006, under the supervision of Sanjeev Arora. From 2006 to 2010 he was a research staff member of the Theory Group at the IBM Almaden Research Center. Since 2010, he has been on the faculty at the Technion - Israel Institute of Technology. His research interests focus on machine learning and convex optimization.

SATYEN KALE received a B. Tech. degree from IIT Bombay and a Ph. D. from Princeton University in 2007. His Ph. D. advisor was Sanjeev Arora. His thesis focused on the topic of this paper: the Multiplicative Weights Update algorithm and its applications. After postdocs at Microsoft Research and Yahoo! Research, he continued his conquest of industrial research labs joining IBM Research in 2011. His current research is the design of efficient and practical algorithms for fundamental problems in Machine Learning and Optimization.