Advanced Algorithms April 24, 2023

Lecture Solving LPs using Multiplicative Weights

Notes by Ola Svensson[1]

In this lecture we do the following:

- We describe the Multiplicative Weight Update (actually Hedge) method.
- We then use this method to solve covering (Set Cover) LPs.
- This is a very fast and simple (i.e., very attractive) method for solving these LPs approximately.

These lecture notes are partly based on an updated version of "Lecture 11 of Topics in TCS, 2015" that were written by Vincent Eggerling and Simon Rodriguez and on the lecture notes by Shiva Kaul that we used in the last lecture.

1 Deterministic Strategies

This section is basically taken verbatim from the excellent lecture notes[2] by Anupam Gupta available here (together with a lot of other interesting information): *https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/*

Multiplicative weights is a retronym for the simple iterative rule that underlies modular, iterative algorithms for solving LPs and SDPs (semidefinite programs)[3]. The Multiplicative Weights Algorithm is known by diferent names in the various fields where it was (re)discovered. Check out the survey by Arora, Hazan and Kale [1]; our discussion will be based on their treatment. Due to its broad appeal, we will consider multiplicative weights in more generality than is needed for solving LPs and SDPs. In this lecture, we'll introduce some strategies for playing a prediction game.

1.1 Warmup: Prediction with Expert Advice

The following sequential game is played between an omniscient Adversary and an Aggregator who is advised by N experts. Special cases of this game include predicting if it will rain tomorrow, or if the stock market will go up or down.

Strategy 1 Prediction with Expert Advice

```
1: for $t = 1 \dots T$ do
2:     Each expert $i \in [N]$ advises either YES or NO
3:     Aggregator predicts either YES or NO
4:     Adversary, with knowledge of the expert advice and Aggregator's prediction, decides the YES/NO
       outcome.
5:     Aggregator observes the outcome and suffers if his prediction was incorrect.
6: end for
```

Naturally, Aggregator wants to make as few mistakes as possible. Since the experts may be unhelpful and the outcomes may be capricious, Aggregator can hope only for a relative performance guarantee. In particular, Aggregator hopes to do as well as the best single expert in hindsight[4]. In order to do so, Aggregator must track which experts are helpful. We will consider a few tracking strategies. Almost

[1] **Disclaimer:** These notes were written as notes for the lecturer. They have not been peer-reviewed and may contain inconsistent notation, typos, and omit citations of relevant works.
[2] This section corresponds to Lecture 16 in the course, which was scribed by Shiva Kaul
[3] See Lecture 17 in the course linked earlier
[4] The excess number of mistakes is called (external) regret

every other aspect of the game - that advise is aggregated into a single value, that this value is binary, and even that the game is sequential - is not relevant; we will generalize or eliminate these aspects.

If there is a perfect expert, then an obvious strategy is to dismiss experts who are not perfect. With the remaining experts, take a majority vote. Whenever the Aggregator makes a mistake, at least half of the remaining experts are dismissed, so Aggregator makes at most $\log N$ mistakes[5]. We can use the same strategy even when there isn't a perfect expert, if we restart after every expert has been eliminated. If the best expert has made M mistakes by time T, then Aggregator has restarted at most $M+1$ times, so it has made at most $(M+1)\cdot\log N$ mistakes. This bound is rather poor since it depends multiplicatively on M.

1.2 Fewer Mistakes with Weighted Majority

We may obtain an additive mistake bound by softening our strategy: instead of dismissing experts who erred, discount their advice. This leads to the *Weighted Majority Algorithm* of Littlestone and Warmuth [2]. Assign each expert i a weight $w_i^{(1)}$ initialized to 1. Then for every t, predict YES/NO based on the weighted majority vote and halve the mistaken experts' weights after observing the outcome. The game strategy then becomes:

Strategy 2 Prediction with Weighted Majority

```
1: Initialize $\mathbf{w}^{(1)} = (w_1^{(1)}, \dots, w_N^{(1)})$ to be a vector of 1's
2: for $t = 1 \dots T$ do
3:     Each expert $i \in [N]$ advises either YES or NO
4:     Aggregator predicts either YES or NO based on a weighted majority vote using $\mathbf{w}^{(t)}$
5:     Adversary, with knowledge of the expert advice and Aggregator's prediction, decides the YES/NO
       outcome.
6:     Aggregator observes the outcome and for every mistaken expert $i$, set $w_i^{(t+1)} \leftarrow w_i^{(t)}/2$
7: end for
```

Claim 1 *For any sequence of outcomes, duration T, let M_{WM} be the number of mistakes that the Weighted Majority strategy makes, and M_i be the number of mistakes that expert i makes. Then*

$$M_{WM} \le 2.41 \cdot (M_i + \log N)$$

Proof Let

$$\Phi^{(t)} = \sum_{i \in [N]} w_i^{(t)}$$

be a 'potential' function. Observe that:

- By definition, we have $\Phi^{(1)} = N$
- Also by definition, $\left(\frac{1}{2}\right)^{M_i} \le \Phi^{(T+1)}$
- At any time τ when WM errs, at least half of the weight gets halved:

  $$\Phi^{(\tau+1)} \le \frac{3}{4}\Phi^{(\tau)}$$

  This implies

  $$\Phi^{(T+1)} \le \left(\frac{3}{4}\right)^{M_{WM}} \Phi^{(1)}$$

[5]We will use log to denote the binary logarithm

Combining these facts yields

$$\left(\frac{1}{2}\right)^{M_i} \leq \left(\frac{3}{4}\right)^{M_{WM}} N$$

Taking the base-2 logarithm of both sides,

$$-M_i \leq \log N + \log\left(\frac{3}{4}\right) M_{WM}$$

so

$$M_{WM} \leq \underbrace{\frac{1}{\log(4/3)}}_{\approx 2.41}(M_i + \log N)$$

■

The leading constant is a consequence of our arbitrary choice to halve the weights. We can optimize ϵ in the update rule

$$w_i^{(t+1)} \leftarrow w_i^{(t)}/(1+\epsilon)$$

then using the WM strategy we may achieve

$$M_{WM} \leq 2(1+\epsilon)(M_i + O(\log N/\epsilon))$$

We analyzed the case when $\epsilon = 1/2$. The same proof gives the following

Theorem 2 *For any sequence of outcomes, duration T, and expert $i \in [N]$,*

$$\textit{\# of WM mistakes} \leq 2(1+\epsilon)\cdot(\textit{\# of i's mistakes}) + O(\log(N)/\epsilon)\,.$$

Proof [Sketch] The proof was done by defining a potential function: for each $t = 1, \ldots, T+1$, let

$$\Phi^{(t)} = \sum_{i\in[N]} w_i^{(t)}\,.$$

We now lower bound the "final" potential $\Phi^{(T+1)}$ using the number of mistakes of i. We then upper bound it in terms of our number of mistakes.

Lower bound: The weight of expert i goes down by a factor $(1-\epsilon)$ for each mistake i does. As the initial weight of i is 1,

$$\Phi^{(T+1)} = \sum_{j\in[N]} w_j^{(T+1)} \geq w_i^{(T+1)} = (1-\epsilon)^{\text{\# of } i\text{'s mistakes}}\,.$$

Upper bound: Every time WM errs, at least half the weight of the experts was wrong (since weighted majority was wrong). These weights are then decreased by $(1-\epsilon)$. It follows that the potential goes down by at least a factor $(1-\epsilon/2)$ every time WM errs. And so

$$\Phi^{(T+1)} \leq \Phi(1)\cdot(1-\epsilon/2)^{\text{\# of WM mistakes}} = N\cdot(1-\epsilon/2)^{\text{\# of WM mistakes}},$$

where for the equality we used that $\Phi(1) = N$ since each expert was initialized with a weight of 1.

The above bounds give us

$$(1-\epsilon)^{\text{\# of } i\text{'s mistakes}} \leq \Phi^{(T+1)} \leq N\cdot(1-\epsilon/2)^{\text{\# of WM mistakes}}\,.$$

Taking logs on both sides, simplifying and rearranging then gives us the statement.

■

2 Changing the game: allowing for randomized strategies

One can prove that there are instances for which weighted majority does twice as many mistakes as the best expert! This is undesirable. To overcome this limitation, we will allow the aggregator to play a random strategy instead of always making a deterministic prediction (that the adversary then uses to create bad instances). A side note is that this is often a general strategy: randomization is often very good to limit the effect of adversaries.

Allowing for randomized strategies leads to the following game with T days and N experts:

For $t = 1, \dots, T$:
1. Each expert $i \in [N]$ gives some advice.
2. Allocator picks some distribution $\vec{p}^{(t)} = \left(p_1^{(t)}, \dots, p_N^{(t)}\right)$ over the experts.
3. Adversary, with knowledge of the expert advice and $\vec{p}^{(t)}$, determines a cost vector $\vec{m}^{(t)} = (m_1^{(t)}, \dots, m_N^{(t)}) \in [-1, 1]^N$.
4. Allocator observes the cost vector and suffers $\vec{p}^{(t)} \cdot \vec{m}^{(t)} = \sum_{i \in [N]} p_i^{(t)} m_i^{(t)}$.

Note that in the above game, we have abstracted away the given advice and in fact Step 1 is not important. At each day t, the goal is to find a distribution $\vec{p}^{(t)}$ over the experts so as to minimize the suffering (the cost). For each day t, the adversary decides that the cost of following the i'th expert's advice is $m_i^{(t)}$. Here, we have generalized the cost to be anything in $[-1, 1]$ instead of only counting the number of mistakes. (A negative number means that it was profitable to follow that expert's advice.) As we play a randomized strategy, the expected cost incurred at day t is thus

$$\sum_{i \in [N]} \Pr[\text{aggregator follows expert } i \text{ at day } t] \cdot m_i^{(t)} = \sum_{i \in [N]} p_i^{(t)} m_i^{(t)} =: \vec{p}^{(t)} \cdot \vec{m}^{(t)} .$$

The ultimate goal is to design a strategy that does as well as the best expert. That is, we wish to to find a strategy such that for every $i \in [N]$

$$\underbrace{\sum_{t=1}^{T} \vec{p}^{(t)} \cdot \vec{m}^{(t)}}_{\text{our loss}} \leq \underbrace{\sum_{t=1}^{T} \vec{m}_i^{(t)}}_{\text{loss of best expert}} + (\text{small error terms}) .$$

The Hedge strategy that we explain in the next section accomplishes this.

Example 1 *A natural example of the above situation is when you investment money. You have the option to choose between say 4 different funds provided by Postfinance, UBS, Credit Suisse, and Banque cantonal. The probability $\vec{p}^{(t)}$ vector that you choose at day t then corresponds to what fraction of your investment goes to the different funds. The goal is to update these fractions so as to make as much money as the best of the 4 options over the long run. Here the penalties (cost or profit) $\vec{m}^{(t)}$ that you observe reflects the performance of the different investments.*

It is quite remarkable that you can do (almost) as good as the best possible option by doing rebalancing. HOWEVER, the caveat "almost" assumes that you live a very long life. For a small T, the guarantees get slightly worse.

3 The Hedge strategy

We now explain and analyze the Hedge strategy for the setting introduced in the last section. It is parameterized by the "learning parameter" $\epsilon > 0$:

- Initially, assign each expert i a weight $w_i^{(1)}$ of 1. (All experts are equally trustworthy in the beginning.)

At each time t:

- Pick the distribution $p_i^{(t)} = w_i^{(t)}/\Phi^{(t)}$ where $\Phi^{(t)} = \sum_{i\in[N]} w_i^{(t)}$.
- After observing the cost vector, set $w_i^{(t+1)} = w_i^{(t)} \cdot e^{-\varepsilon \cdot m_i^{(t)}}$.

The difference between Hedge and the so-called Multiplicative Weight Update method is the weight-update $w_i^{(t+1)} = w_i^{(t)} \cdot e^{-\varepsilon \cdot m_i^{(t)}}$ instead of $w_i^{(t+1)} = w_i^{(t)} \cdot (1 - \varepsilon \cdot m_i^{(t)})$. Both methods have similar guarantees:

Theorem 3 *Suppose $\epsilon \leq 1$ and for $t \in [T]$, $\vec{p}^{(t)}$ is picked by Hedge. Then for any expert i,*

$$\sum_{t=1}^{T} \vec{p}^{(t)} \cdot \vec{m}^{(t)} \leq \sum_{t=1}^{T} m_i^{(t)} + \frac{\ln N}{\epsilon} + \epsilon T\,.$$

Note that in the above theorem, i can be chosen to be the best expert and thus Hedge does as well as the best expert plus some small error terms.
Proof Similar to the analysis of the weighted majority strategy, the proof goes by analyzing the potential $\Phi^t = \sum_{i\in[N]} w_i^{(t)}$.

Lower bound on $\Phi^{(T+1)}$: We lower bound the final potential as a function of i's performance:

$$\Phi^{(T+1)} = \sum_{j\in[N]} w_j^{(T+1)} \geq w_i^{(T+1)} = \exp(-\epsilon \sum_{t=1}^{T} m_i^{(t)})\,,$$

where the last equality follows from that the initial weight of i was one and every day t his weight was updated by $e^{-\epsilon m_i^{(t)}}$.

Upper bound on $\Phi^{(T+1)}$: We upper bound the final potential in terms of the total cost Hedge suffered. By definition,

$$\Phi^{(t+1)} = \sum_{j\in[N]} w_j^{(t+1)} = \sum_{j\in[N]} w_j^{(t)} \cdot \exp(-\epsilon m_j^{(t)})\,.$$

The exponentiated term is in $[-1,1]$. Since $e^x \leq 1 + x + x^2$ for $x \in [-1,1]$ (do a Taylor expansion),

$$\begin{aligned}
\sum_{j\in[N]} w_j^{(t)} \cdot \exp(-\epsilon m_j^{(t)}) &\leq \sum_{j\in[N]} w_j^{(t)} \left((1 - \epsilon m_j^{(t)} + \epsilon^2 \left(m_j^{(t)}\right)^2\right) \\
&\leq \sum_{j\in[N]} w_j^{(t)} \left((1 - \epsilon m_j^{(t)} + \epsilon^2\right) \qquad \left(\text{since } \left(m_j^{(t)}\right)^2 \leq 1\right) \\
&= \sum_{j\in[N]} w_j^{(t)}(1+\epsilon^2) - \sum_{j\in[N]} \epsilon w_j^{(t)} m_j^{(t)} \\
&= \Phi^{(t)}(1+\epsilon^2) - \epsilon \sum_{j\in[N]} \Phi^{(t)} p_j^{(t)} m_j^{(t)} \qquad (\text{since } \Phi^{(t)} p_j^{(t)} = w_j^{(t)}) \\
&= \Phi^{(t)} \left(1 + \epsilon^2 - \epsilon \vec{p}^{(t)} \vec{m}^{(t)}\right) \\
&\leq \Phi^{(t)} \cdot \exp\left(\epsilon^2 - \epsilon \vec{p}^{(t)} \vec{m}^{(t)}\right) \qquad (\text{since } 1 + x \leq e^x)\,.
\end{aligned}$$

We therefore have

$$\begin{aligned}
\Phi^{(T+1)} &\leq \Phi^{(T)} \cdot \exp\left(\epsilon^2 - \epsilon \vec{p}^{(T)} \vec{m}^{(T)}\right) \\
&\leq \Phi^{(T-1)} \cdot \exp\left(\epsilon^2 - \epsilon \vec{p}^{(T-1)} \vec{m}^{(T-1)}\right) \cdot \exp\left(\epsilon^2 - \epsilon \vec{p}^{(T)} \vec{m}^{(T)}\right) \\
&\quad \vdots \\
&\leq \Phi^{(1)} \cdot \exp\left(\epsilon^2 T - \epsilon \sum_{t=1}^{T} \vec{p}^{(t)} \vec{m}^{(t)}\right) \\
&= N \cdot \exp\left(\epsilon^2 T - \epsilon \sum_{t=1}^{T} \vec{p}^{(t)} \vec{m}^{(t)}\right),
\end{aligned}$$

where for the equality we used that $\Phi(1) = N$ since each expert was initialized with a weight of 1.

The above bounds give us

$$\exp(-\epsilon \sum_{t=1}^{T} m_i^{(t)}) \leq \Phi^{(T+1)} \leq N \cdot \exp\left(\epsilon^2 T - \epsilon \sum_{t=1}^{T} \vec{p}^{(t)} \vec{m}^{(t)}\right).$$

Taking (natural) logarithms,

$$-\epsilon \sum_{t=1}^{T} m_i^{(t)} \leq \ln(N) + \epsilon^2 T - \epsilon \sum_{t=1}^{T} \vec{p}^{(t)} \vec{m}^{(t)}$$

and the final result follows by dividing by ϵ and rearranging the terms. ■

Remark The above proof may seem messy with all the inequalities. However, it follows a standard "framework": upper and lower bound the potential function. These Multiplicative Weight Update type of algorithms are most often analyzed using this potential argument. Once you know that cool argument, the analysis is almost automatic modulo some rewriting.

For the application of solving covering LPs, it will be convenient to consider the *average cost* incurred per day. We also generalize so that the cost vector can take values in $[-\rho, \rho]^N$ where ρ is called the "width". The proof is the same by scaling and we get the following corollary:

Corollary 4 *Suppose $\epsilon \leq 1$ and for $t \in [T]$, $\vec{p}^{(t)}$ is picked by Hedge and cost vectors satisfy $\vec{m}^{(t)} \in [-\rho, \rho]^N$. If $T \geq (4\rho^2 \ln N)/\epsilon^2$, then for any expert i:*

$$\frac{1}{T} \sum_{t=1}^{T} \vec{p}^{(t)} \cdot \vec{m}^{(t)} \leq \frac{1}{T} \sum_{t=1}^{T} m_i^{(t)} + 2\epsilon.$$

4 Hedging for LPs

We now turn to a nice application of the Hedge method to that of solving *covering LPs*. We start by defining these LPs.

4.1 Covering LPs

Definition 5 *A linear program of the form :*

$$\begin{aligned} \textbf{\textit{minimize}} \quad & \sum_{j=1}^{n} c_j x_j \\ \textbf{\textit{subject to}} \quad & Ax \geq b \\ & 1 \geq x_j \geq 0 \quad \forall j \end{aligned}$$

is a ***covering linear program*** *if*

$$A \in \mathbb{R}_{+}^{m \times n} \text{ , } b \in \mathbb{R}_{+}^{m} \text{ and } c \in \mathbb{R}_{+}^{n}$$

This definition simply ensures that all the coefficients of the constraints and of the objective function are non-negative. Notice that both the set cover relaxation and the vertex cover relaxation that we saw in class were covering LPs.

Let us now introduce an example of a covering LP that we will reuse later:

$$\begin{aligned} \textbf{minimize} \quad & x_1 + 2x_2 \\ \textbf{subject to} \quad & x_1 + 3x_2 \geq 2 \\ & 2x_1 + x_2 \geq 1 \\ & 1 \geq x_1, x_2 \geq 0 \end{aligned} \tag{1}$$

4.2 General idea

The idea of using the Hedge method for linear programming is to associate an expert with each constraint of the LP. In other words, the Hedge method will maintain a weight distribution over the set of constraints of a linear problem to solve, and to iteratively update those weights in a multiplicative manner based on the cost function at each step/day. Of course we need to define the cost function in a smart way later but let us first define the notion of an oracle and then give a small instructive example.

Initially, the Hedge method will give a weight $w_i^{(1)} = 1$ for every constraint/expert $i = 1, \ldots, m$ (the number m of constraints now equals the number of experts). And at each day t, it will maintain a convex combination $\vec{p}^{(t)}$ of the constraints (that is defined in terms of the weights). Using such a convex combination $\vec{p}$, a natural easier LP with a single constraint is obtained by summing up all the constraints according to $\vec{p}$. Any optimal solution of the original LP is also a solution of this reduced problem, so the new problem will have at most the same cost as the previous one. We define an oracle for solving this reduced problem:

Definition 6 *An oracle that, given $\vec{p} = (p_1, \ldots, p_m) \geq \mathbf{0}$ such that $\sum_{i=1}^{m} p_i = 1$, outputs an optimal solution x^* to the following reduced linear problem:*

$$\begin{aligned} \textbf{\textit{minimize}} \quad & \sum_{j=1}^{n} c_j x_j \\ \textbf{\textit{subject to}} \quad & \left(\sum_{i=1}^{m} p_i A_i \right) \cdot x \geq \sum_{i=1}^{m} p_i b_i \\ & 1 \geq x \geq 0 \end{aligned}$$

This linear problem is in practice much easier to solve than the original one and we can design a simple greedy algorithm for the oracle.

For concreteness, let us consider a small example. If we apply the method we described to our example (1), we have two initials weights $w_1 = w_2 = 1$ and thus $p_1 = p_2 = \frac{1}{2}$, and we sum all the constraints:

$$\begin{gathered} p_1(x_1 + 3x_2) + p_2(2x_1 + x_2) \geq p_1 \cdot 2 + p_2 \cdot 1 \Leftrightarrow \\ 1.5x_1 + 2x_2 \geq 1.5 \\ 1 \geq x_1, x_2 \geq 0 \end{gathered}$$

By using the oracle, an optimal solution to this reduced problem is $x_1 = 1, x_2 = 0$ of cost 1. But is this a feasible solution to our original problem? By checking the constraints of the original LP:

$$\begin{array}{ll} 2x_1 + x_2 = 2 \geq 1 & \text{OK} \\ x_1 + 3x_2 = 1 < 2 & \text{not OK} \end{array}$$

We will need to go back to the original problem and *increase the weights of the unsatisfied constraints* to give them more importance and *decrease the weights of the satisfied constraint.* We will perform these updates (as done by Hedge) multiplicatively that we describe in detail in Section 4.4. We first describe how to implement the oracle in general.

4.3 Implementation of the oracle

The oracle is given an objective function **minimize** $\sum_{i=1}^n c_i x_i$ and only one constraint that we can rewrite as $\sum_{i=1}^n d_i x_i \geq b$ (which is the weighted sum of all constraints). We also have $1 \geq x_i \geq 0\ \forall i$ and $c_i, d_i \geq 0\ \forall i$ since it is a covering problem.

The idea is to assign the maximum value (namely 1) to the variables that have the highest ratio *constraint coefficient/objective coefficient.* That way we will satisfy the constraint as fast as possible while maintaining a small objective function. Then assign zero to the other variables and possibly an intermediate value for the variable that is at the limit to make the constraint satisfied. This is very similar to the most "bang-for-the-buck" greedy and it is the solution to fractional knapsack. Formally:

- Sort and relabel all variables x_i so that $d_1/c_1 \geq d_2/c_2 \geq \ldots d_n/c_n$.
- Let $k = \min\{j : \sum_{i=1}^j d_i \geq b\}$ and $\ell = b - \sum_{i=1}^{k-1} d_i$.
- Set $x_i := 1$ for $i = 1, ..., k-1$.
- Set $x_k := \ell/d_k$.
- Set $x_i := 0$ for $i = k+1, ..., n$.

We have,

$$\sum_{i=1}^n d_i x_i = \sum_{i=1}^n d_i x_i = \sum_{i=1}^{k-1} d_i x_i + d_k x_{\sigma(k)} + \sum_{i=k+1}^n d_i x_i = \sum_{i=1}^{k-1} d_i + \ell = b$$

Therefore the constraint is exactly satisfied. By the ordering of the variables, this ensures that we minimize the objective function as required. Having implemented the oracle, we proceed to formally define the Hedge algorithm for linear programming.

4.4 Hedge algorithm for covering LPs

As already explained, we associate an expert to each constraint of the covering LP. In addition, as hinted above, we wish to increase the weight of unsatisfied constraints and decrease the weight of satisfied constraints (in a smooth manner depending on the size of the violation or the slack). The Hedge algorithm for covering LPs thus becomes:

- Assign each constraint i a weight $w_i^{(1)}$ initialized to 1.

At each time t:

- Pick the distribution $p_i^{(t)} = w_i^{(t)}/\Phi^{(t)}$ where $\Phi^{(t)} = \sum_{i\in[N]} w_i^{(t)}$.
- Now *we define the cost vector instead of the adversary* as follows:
  - Let $x^{(t)}$ be the solution returned by the oracle on the LP obtained by using the convex combination $\vec{p}^{(t)}$ of constraints. Notice that cost of $x^{(t)}$, i.e., $c^\top x^{(t)}$, is at most the cost of an optimal solution to the original LP.
  - Define the cost of constraint i as

    $$m_i^{(t)} = \sum_{j=1}^{n} A_{ij}x_j - b_i = A_i x - b_i.$$

    Notice that we have a positive cost if the constraint is satisfied (so the weight will be decreased by Hedge) and a negative cost if it is violated (so the weight will be increased by Hedge).
- After observing the cost vector, set $w_i^{(t+1)} = w_i^{(t)} \cdot e^{-\varepsilon\cdot m_i^{(t)}}$.

Output: the average $\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x^{(t)}$ of the constructed solutions.

Analysis. Since the analysis for the Hedge algorithm works with an adversarial construction of the cost vectors, it definitely holds for the cost vectors we constructed. Let

$$\rho = \max_{1\le i\le m}\left\{\max(b_i, A_i\mathbf{1} - b_i)\right\},$$

be (an upper bound on) the width of our constructed cost vectors. By Corollary 4, we thus have for $\epsilon \in [0,1], T \ge (4\rho^2 \ln m)/\epsilon^2$ and any constraint i that

$$\frac{1}{T}\sum_{t=1}^{T} \vec{p}^{(t)} \cdot \vec{m}^{(t)} \le \frac{1}{T}\sum_{t=1}^{T} m_i^{(t)} + 2\epsilon\,. \tag{2}$$

Let us first consider the left-hand-side of the above expression. We have that

$$\begin{aligned}\sum_{t=1}^{T} \vec{p}^{(t)} \cdot \vec{m}^{(t)} &= \sum_{t=1}^{T}\left(\sum_i p_i^{(t)} \cdot m_i^{(t)}\right)\\ &= \sum_{t=1}^{T}\left(\underbrace{\sum_i p_i^{(t)} \cdot (A_i x^{(t)} - b_i)}_{(*)}\right)\end{aligned}$$

which is non-negative because $(*)$ is non-negative for every t since the oracle outputs a feasible solution. Using that the left-hand-side of (2) is non-negative, we get

$$-2\epsilon \le \frac{1}{T}\sum_{t=1}^{T} m_i^{(t)} = \frac{1}{T}\sum_{t=1}^{T}\left(A_i x^{(t)} - b_i\right) = A_i\bar{x} - b_i$$

which implies that

$$A_i\bar{x} \ge b_i - 2\epsilon\,,$$

for every constraint i. So our solution $\bar{x}$ is **almost feasible**. Moreover the cost $c^\top\bar{x} = c^\top(\frac{1}{T}\sum_{t\in[T]} x^{(t)})$ is **at most the cost of an optimal solution** to the original LP since each $x^{(t)}$ has no higher cost than an optimal solution (by the properties of the oracle).

Setting the parameters for solving the set cover relaxation. For set cover we have that $\rho \le n$ and therefore, it is sufficient to set $T = (4n^2 \ln m)/\epsilon^2$ (this can in fact be improved by a better analysis to $\approx n\ln m/\epsilon^2$). This gives a solution $\bar{x}$ that satisfies $\sum_{e\in S} x_e \ge 1-2\epsilon$ for every set S and the cost of $\bar{x}$ is at most that of an optimal LP solution. We can obtain a feasible (approximately optimal) solution by simply defining $x^* = \frac{1}{1-2\epsilon}\bar{x}$.

References

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. Technical report, Princeton University, 2005. 16

[2] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In FOCS, pages 256–261, 1989. 16.1.1