Machine Learning Theory
Lecture 17

Nicholas Harvey

November 7, 2018

1 Bandits

Bandits are basically like the "experts" setup of last time, except that the algorithm only receives **partial feedback**. In round i, the algorithm does not receive the entire vector c_i; if it chooses expert j, it only receives the cost $c_{i,j}$.

1.1 The bandit model

There is an agent who must make a sequence of online decisions. There are a sequence of discrete time steps; in each time step, the agent must make a decision and incur the cost of that decision. The agent has limited or no forehand knowledge of what that cost will be. Of course, the goal is to incur as little cost as possible. At the end of each time step, the agent is told the cost of the choice he actually made, but not the other choices he could have made.

A typical setup is that there are n ***actions***. The costs at time i are specified by a vector $c_i = (c_{i,1}, ..., c_{i,n}) \in \mathbb{R}^n$, where $c_{i,j}$ is the cost of performing action j at time i. The agent must (without knowing c_i) pick an action j to perform, then incur the cost $c_{i,j}$. The cost vector c_i is **not** revealed to the agent; only the **single** cost $c_{i,j}$ is revealed.

There are many different models of bandits. Two of the most basic are ***stochastic bandits***, which we will discuss next week, and ***adversarial bandits*** (or ***non-stochastic bandits***), which we discuss today.

"Adversarial" Bandits. This setting is most similar to the model of learning with experts, which we discussed last week. Let us now consider the power of the adversary. As mentioned last time, if c_i depends on the action chosen at time i, the results would be uninteresting: the lower bound discussed last week implies a regret of $\Omega(t)$ after t rounds.

So suppose we only allowed the adversary to determine c_i based upon the algorithm's decisions in rounds 1 through $i-1$. The adversary does not know any random bits used by the algorithm. This is called an ***adaptive online adversary***. Important results have been proven in this setting, even in the bandit model, but there is also a sentiment[1] that regret bounds against adaptive online adversaries are "not meaningful".

[1] See Arora, Dekel, Tewari for further discussion.

Today, we will consider adversarial bandits with an ***oblivious adversary***: the adversary knows the algorithm (but not its random bits), and all cost functions c_i are chosen *before* the algorithm begins execution. Already this is quite interesting because it is a substantial generalization of stochastic bandits.

As before, let $A(t)$ be the cost incurred by the algorithm up to round t. We now define

$$\text{Regret}(t) \;=\; \max_{j\in[n]} \text{E}\left[\, A(t) - \sum_{i=1}^{t} c_{i,j} \,\right] \;=\; \text{E}\,[\,A(t)\,] - \min_{j\in[n]} \sum_{i=1}^{t} c_{i,j}.$$

Usually, the goal is to get $\text{Regret}(t) = o(t)$.

2 Brainstorming

In this section we try some ideas for bandit algorithms against oblivious adversaries.

Idea 1. Since the bandit model only gives us a single cost per round, why not use n rounds to build up a "simulated" cost function, then run RWM on those simulated cost functions.

For example, we could use build the simulated cost function $\hat{c}_p$ during rounds $n(p-1)+1, ..., np$. Simply do action j during round $n(p-1)+j$, and set

$$\hat{c}_{p,j} \;=\; c_{n(p-1)+j,\,j}.$$

Then we can run RWM on the cost functions $\hat{c}_1, \hat{c}_2, ...$

Bad Example 1. The trouble is that the adversary, though oblivious, knows our algorithm so it can just ensure that $c_{n(p-1)+j,\,j} = 1$ and all other costs are zero. Then the algorithm's total cost is t, but every action's total cost is t/n, so the regret is $(1-1/n)t$.

Idea 2. The natural way to defeat the adversary is to randomize our decisions so that he cannot predict which costs we will examine.

For example, we could pick random permutations $\pi_1, \pi_2, ...$, where $\pi_i : [n] \to [n]$. Then we could build the simulated cost function $\hat{c}$ by setting

$$\hat{c}_{p,j} \;=\; c_{n(p-1)+\pi(j),\,j}.$$

So we use the $\pi(j)^{\text{th}}$ round of phase p to learn the cost of action j.

Bad Example 2. The trouble is that, even if most of actions are completely useless, we will spend most our time repeatedly learning that they are completely useless. For example, suppose that the first action always has cost 0, and the other actions always have cost 1. Then the algorithm's total cost is $(1-1/n)t$, so the regret is again $(1-1/n)t$.

Idea 3. The trouble with the previous algorithm is that we spent all our time learning the actions' costs, and not enough time using what we had learned. Too much exploration, not enough exploitation.

Perhaps instead we could alternate phases of exploring and exploiting?

- **Explore.** Spend the first n rounds building $\hat{c}_1$. Then run one iteration of RWM, obtaining a random action X_1 as its output.
- **Exploit.** For the next m iterations, just perform the same action X_1.
- **Explore.** Spend the next n rounds building $\hat{c}_2$. Then run another iteration of RWM, obtaining a random action X_2 as its output.
- **Exploit.** For the next m iterations, just perform the same action X_2.
- ...

Bad Example 3. The adversary can just modify the bad example for Idea 2 by negating the costs during the Exploit rounds. Then the algorithm's cost is $n - 1 + m = t - 1$, the first action's total cost is m, the second action's total cost is n, so the regret is $(t-1) - \min m, n \geq t/2 - 1$.

Idea 4. The trouble with the previous algorithm is that the adversary knew in advance when we were Exploring and when we were Exploiting.

Perhaps we could randomly mix the exploration and exploitation phases?

- **Pick a good action.** Arbitrarily pick some action X_1.
- **Phase 1.** Randomly partition the first $n + m$ rounds into n rounds of Exploration and m rounds of Exploitation. Build $\hat{c}_1$ by querying each action exactly once during a randomly chosen Exploration round.

  During each Exploitation round, we don't yet have enough information to do anything useful, so we just query the action X_1.
- **Pick a good action.** Now that Phase 1 has finished, we have built the cost function $\hat{c}_1$. We run one iteration of RWM, and it gives us a (random) action X_2 that it believes to be good.
- **Phase 1.** Randomly partition the next $n + m$ rounds into n rounds of Exploration and m rounds of Exploitation. Build $\hat{c}_2$ by querying each action exactly once during a randomly chosen Exploration round. During each Exploitation round, we query the action X_2.
- ...

Now there doesn't seem to be a bad example. Indeed, the algorithm we have developed is exactly the PSim algorithm.

3 PSim: Phased Simulation

In this section we develop a fairly simple algorithm that achieves regret $O\big(t^{2/3}(n \log n)^{1/3}\big)$ against oblivious adversaries. I am not sure from where this algorithm originates; I learned it from here `http://www.cs.cornell.edu/courses/cs683/2007sp/`.

3.1 Overview of PSim

Main ideas of PSim

- Simulate Randomized Weighted Majority (RWM)
- Each bandit round only gives us cost of a single action, so we will use multiple rounds (a "phase") to learn the cost of all the action.
- Since it takes us multiple rounds to learn these different values, we will learn the actions' values on *different* cost functions.
- This is not a big problem because we will randomize things, and the costs that we learn will be close to their average values. (This strategy relies on our assumption that the costs are determined by an *oblivious* adversary.)

More formally:

- **Phases.** Partition the t rounds into k contiguous phases, where each phase has length $L > n$. (So $t = kL$.) The time steps in phase p are:

  $$\Pi_p \;=\; \{(p-1) \cdot L + 1, ..., p \cdot L\}\,.$$

  In each phase we will use n of the rounds to "learn" the cost functions (**exploring**) and use the remaining rounds to repeatedly perform the good actions (**exploiting**). (Hence the need that $L > n$.)

- **Exploring.** In each phase, we will learn each action's cost exactly once in a randomly chosen round. (Randomly chosen so that the adversary cannot give that action a garbage cost in that round.) These rounds must all be distinct, because the bandit model only allows us to learn a single action's cost per round. (Hence $L > n$.) Formally, we let τ_p be a random one-to-one (i.e., injective) map from $[n]$ to Π_j, and use $\tau_p(j)$ as the round in phase p during which we learn action j's cost. In most rounds we will not be learning costs (recall $L > n$), so τ_p is not onto (i.e., not surjective).

  The input to the RWM algorithm at the end of phase p is the "simulated" cost functions learned during phase p's exploration rounds. Formally, the simulated cost function learned during phase p is $\hat{c} : [n] \to [0, 1]$ defined by

  $$\hat{c}_{p,j} = c_{\tau_p(j),j}.$$

- **Exploiting.** The choice of action to be used during exploitation rounds is determined by the output of the RWM algorithm. At the start of phase p, we have already learned the simulated cost functions $\hat{c}_1, ..., \hat{c}_{p-1}$. We then run RWM on those cost functions, giving us the distribution x_{p-1} on actions to be used for exploitation during phase p. Randomly draw a "good action" X_{p-1} from the distribution x_{p-1}; this is the action we will use repeatedly for exploitation in round p.

- Formally, define

  - $\mathsf{EXPLORE}_j = \{\ \tau_p(j)\ :\ p \geq 1\ \}$ to be the set of all rounds in which learn the j'th action's cost,
  - $\mathsf{EXPLORE} = \bigcup_{j=1}^{n} \mathsf{EXPLORE}_j$ to be the set of all rounds in which we learn *some* action's cost, and
  - $\mathsf{EXPLOIT} = [t] \setminus \mathsf{EXPLORE}$ to be all other rounds.

  The action to be used in round i is:

$$a(i) \ = \ \begin{cases} j & (\text{if } i \in \mathsf{EXPLORE}_j) \\ X_{p-1} & (\text{if } i \in \mathsf{EXPLOIT} \cap \Pi_p) \end{cases}.$$

  That is, we set $a(i) = j$ if we are supposed to learn action j's cost in round i; otherwise, we are exploiting in round i, so we set $a(i) = X_{p-1}$, the (randomly chosen) "good action" that we learned during phase $p - 1$.

Algorithm 1 The PSim algorithm.

procedure PSIM(n, t)
  Let $\epsilon = (n \ln(n)/t)^{1/3}$
  Let $k = \ln(n)/\epsilon^2$
  ▷ Assume t is a multiple of k
  Let $L = t/k$
  Let $X_0 = 0$
  for $p \leftarrow 1, ..., k$ **do**
    Pick the random injective map $\tau_p : [n] \to \Pi_p$
    Initialize $\hat{c}_p$ to be the zero vector
    for $i \in \Pi_p$ **do**
      Perform the action $a(i)$, receiving its cost γ
      If $i \in \mathsf{EXPLORE}_j$, record the simulated cost $\hat{c}_{p,j} = \gamma$
    Continue running the Randomized Weighted Majority algorithm, now using the simulated cost vector $\hat{c}_p$, and receiving the distribution x_p.
    Randomly sample the good action X_p according to the distribution x_p.

Analysis idea. Recall that the simulated costs are determined from the actual costs in randomly sampled rounds (i.e., $\hat{c}_{p,j} = c_{\tau_p(j),j}$). So the average of the simulated costs should be close to the

average of the actual costs. That is, the cost vectors satisfy

$$\underbrace{\frac{1}{k}\sum_{p=1}^{k}\hat{c}_p}_{\text{average simulated cost}} \quad \approx \quad \underbrace{\frac{1}{t}\sum_{i=1}^{t}c_i}_{\text{average real cost}} \quad . \tag{3.1}$$

Since RWM gives us a strategy that is nearly-optimal with respect to the $\hat{c}$ simulated costs, it should also be nearly-optimal with respect to the real costs. All that remains is to make those ideas precise.

Review of RWM's regret. Recall from our analysis last week that the expected total cost of Randomized Weighted Majority at time t is

$$A(t) \;\le\; (1+\epsilon)\sum_{i=1}^{t} c_{i,j^*} + \frac{\ln n}{\epsilon} \;\le\; \sum_{i=1}^{t} c_{i,j^*} + \epsilon t + \frac{\ln n}{\epsilon},$$

using that the costs are in $[0,1]$, and where j^* is the action of minimum total cost. Thus,

$$\text{Regret}(t) \;\le\; \epsilon t + \frac{\ln n}{\epsilon}. \tag{3.2}$$

3.2 Formal analysis

Theorem 3.1. With appropriately chosen parameters, the PSim algorithm achieves regret $3t^{2/3}(n\log n)^{1/3}$ after t iterations.

To start, we will aim to show something more refined than (3.1). The average cost during phase p will be denoted

$$\bar{c}_p \;=\; \frac{1}{L}\sum_{i\in\Pi_p} c_i.$$

Since $\hat{c}_p$ is obtained by random sampling from Π_p, clearly $\hat{c}_p$ equals $\bar{c}_p$ in expectation (over τ_p).

We can write that statement more formally as follows. Let $\mathcal{F}_{p-1}$ be the sigma-field generated by all random variables up to the end of phase $p-1$, namely $X_1, ..., X_{p-1}$ and $\tau_1, ..., \tau_{p-1}$. Recall that τ_p is independent of the past, i.e., generated with fresh randomness. Formally, τ_p is independent of $\mathcal{F}_{p-1}$. Thus, we have

$$\mathrm{E}\left[\,\hat{c}_p \;\middle|\; \mathcal{F}_{p-1}\,\right] \;=\; \bar{c}_p. \tag{3.3}$$

This says that, no matter what happened in earlier phases, $\hat{c}_p$ is an unbiased estimate of $\bar{c}_p$.

Remark. We started this section saying that the adversary should be oblivious. Now we can point out how the analysis uses this assumption. Suppose that the adversary could adaptively change the costs based on past actions. Then, for $i \in \Pi_p$, the cost vector c_i would not be $\mathcal{F}_{p-1}$-measurable (except for the first round of phase p). Consequently $\bar{c}_p$ would not be $\mathcal{F}_{p-1}$-measurable and (3.3) would make no sense.

It turns out that we must make a slightly more precise statement than (3.3). Ultimately, we will want to compare $\hat{c}_p$ and $\bar{c}_p$ at the *random* index X_{p-1}, because that is the "good action" used in phase p. That would be problematic if X_{p-1} depended on τ_p, but fortunately it does not. Formally we say that X_{p-1} is $\mathcal{F}_{p-1}$-measureable.

Lemma 3.2. Then $\mathrm{E}\left[\,\hat{c}_{p,J} - \bar{c}_{p,J} \mid \mathcal{F}_{p-1}\,\right] = 0$ for any index J that is $\mathcal{F}_{p-1}$-measurable.

Proof.

$$\begin{aligned}
\mathrm{E}\left[\,\hat{c}_{p,J} \mid \mathcal{F}_{p-1}\,\right] &= \sum_{i\in\Pi_p} \Pr\left[\,\tau_p(J) = i \mid \mathcal{F}_{p-1}\,\right] \cdot c_{i,J} \\
&= \sum_{i\in\Pi_p} \frac{1}{L} \cdot c_{i,J} \quad \text{(since } \tau_p \text{ is independent of } \mathcal{F}_{p-1} \text{ and } J \text{ is } \mathcal{F}_{p-1}\text{-measureable)} \\
&= \bar{c}_{p,J}.
\end{aligned}$$

Subtracting yields the statement of the lemma. ■

The next lemma directly bounds PSim's expected regret.

Lemma 3.3. For all $j \in [n]$,

$$\mathrm{E}\left[\sum_{i=1}^{t}(c_{i,a(i)} - c_{i,j})\right] \;\le\; \epsilon t + \frac{L\ln n}{\epsilon} + nk.$$

Proof. The first step is to relate the regret with respect to the real costs c_i to the regret with respect to the simulated costs $\hat{c}_p$.

$$\begin{aligned}
&\sum_{i=1}^{t} \mathrm{E}\left[\,c_{i,a(i)} - c_{i,j}\,\right] \\
&= \sum_{p=1}^{k} \sum_{i\in\Pi_p} \mathrm{E}\left[\,c_{i,a(i)} - c_{i,j}\,\right] \quad \text{(partitioning the sum into phases)} \\
&= \sum_{p=1}^{k} \left(\sum_{i\in\Pi_p} \mathrm{E}\left[\,c_{i,X_{p-1}} - c_{i,j}\,\right] - \sum_{i\in\Pi_p} \mathrm{E}\left[\,c_{i,X_{p-1}} - c_{i,a(i)}\,\right] \right) \quad \text{(adding and subtracting } c_{i,X_{p-1}}\text{)} \\
&= \sum_{p=1}^{k} \left(\sum_{i\in\Pi_p} \mathrm{E}\left[\,c_{i,X_{p-1}} - c_{i,j}\,\right] - \sum_{i\in\Pi_p\cap\textsf{EXPLORE}} \mathrm{E}\left[\,c_{i,X_{p-1}} - c_{i,a(i)}\,\right] \right) \quad (X_{p-1} = a(i) \text{ for } i \in \Pi_p \cap \textsf{EXPLOIT}) \\
&= \sum_{p=1}^{k} \left(L \cdot \mathrm{E}\left[\,\bar{c}_{p,X_{p-1}} - \bar{c}_{p,j}\,\right] - \sum_{i\in\Pi_p\cap\textsf{EXPLORE}} \mathrm{E}\left[\,c_{i,X_{p-1}} - c_{i,a(i)}\,\right] \right) \quad \text{(since } \bar{c}_p = \sum_{i\in\Pi_p} c_i/L\text{)} \\
&\le L\sum_{p=1}^{k} \mathrm{E}\left[\,\bar{c}_{p,X_{p-1}} - \bar{c}_{p,j}\,\right] + \sum_{p=1}^{k} \sum_{i\in\Pi_p\cap\textsf{EXPLORE}} 1 \quad \text{(since } c_i \in [0,1]^n\text{)} \\
&= L\sum_{p=1}^{k} \mathrm{E}\left[\,\bar{c}_{p,X_{p-1}} - \bar{c}_{p,j}\,\right] + nk \quad \text{(each phase has } n \text{ exploration rounds)}
\end{aligned}$$

$$= L\sum_{p=1}^{k} \mathrm{E}\left[\,\hat{c}_{p,X_{p-1}} - \hat{c}_{p,j}\,\right] \;+\; nk, \tag{3.4}$$

applying Lemma 3.2 with $J = X_{p-1}$ and again with $J = j$, then taking the (unconditional) expectation.

Next, the regret with respect to the simulated costs is easy to control using our RWM analysis. Recall that we ran RWM for $p = 1, ..., k$ on inputs $\hat{c}_p$, during which it randomly chose the action X_p. So, by our regret bound on RWM (see (3.2)), we obtain the upper bound

$$\sum_{p=1}^{k} \mathrm{E}\left[\,\hat{c}_{p,X_{p-1}} - \hat{c}_{p,j}\,\right] \;\leq\; \epsilon k + \frac{\ln n}{\epsilon}.$$

Multiplying by L, then combining with (3.4) completes the proof. ■

Proof (of Theorem 3.1). All that remains is to plug the appropriate ϵ and k into Lemma 3.3. Setting $\epsilon = \left(\frac{n \log n}{t}\right)^{1/3}$ and $k = \frac{\ln n}{\epsilon^2}$, we obtain the regret bound

$$\epsilon t + \frac{L \ln n}{\epsilon} + nk \;=\; \Big(\epsilon + \frac{\ln n}{\epsilon k} + \frac{n \ln n}{t\epsilon^2}\Big)t \;=\; (\epsilon + \epsilon + \epsilon)t.$$

■

4 Exp3

The previous section presented the PSim algorithm for the bandits problem, which explicitly "explores" the cost of each action at random times, then uses RWM to choose an good action to "exploit" for most other times. This algorithm achieves regret bound $O\big(t^{2/3}(n \log n)^{1/3}\big)$, which is at least $o(T)$, but it is not optimal.

In this section we consider the following idea: is the randomness inherent in RWM already enough to achieve a good exploration/exploitation tradeoff? For example, initially all actions have equal weight, so RWM will explore all the actions in some random order. Eventually, some actions will be determined to have high cost, but they will be given small weight, so won't be explored very often. The remaining actions of small cost will all have large weights, and so will be exploited most of the time.

This idea is realized by the Exp3 algorithm, which achieves regret bound $O(\sqrt{tn \log n})$. That bound is nearly optimal in the adversarial bandits setting. The algorithm's name stands for "Exponential-weight algorithm for Exploration and Exploitation". It was original presented by Auer, Cesa-Bianchi, Freund and Schapire in the FOCS 1995 conference [2, 1].

Overview of ideas. In more modern terminology, Exp3 can be explained as follows:

Gradient Descent	is to	Random Coordinate Descent
	as	
Randomized Weighted Majority	is to	Exp3

The main idea is extremely simple. Recall that we generalized gradient descent to stochastic gradient descent by introducing a randomized gradient oracle; in expectation, the analysis did not change at all. Then, on assignment 3, we saw random *coordinate* descent, in which each random subgradient has only a single non-zero coordinate. Roughly, Exp3 is what results from applying the same idea to Randomized Weighted Majority.

Here's a bit more detail: the stochastic gradient descent analysis applies to *any* oracle returning a random vector that, in expectation, is a subgradient. The analogous idea for Exp3 is to construct a random vector of "simulated costs" that, in expectation, equal the true costs. Actually, there is an additional key requirement on the simulated costs: since the bandit model only reveals the cost of a single action, simulated cost vector will only have a single non-zero coordinate.

Algorithm 2 The Exp3 algorithm.

procedure EXP3(η)
  Let $y_1 = (1, ..., 1)$
  for $i \leftarrow 1, 2, ...$ **do**
    Let $x_i = y_i / \|y_i\|_1$ (normalize the weights).
    Use fresh randomness to pick action $X_i \in [n]$ according to distribution x_i.
    Perform the action X_i
    ▷ Expected cost incurred is $\sum_{j=1}^{n} c_{i,j} x_{i,j} = c_i^{\mathsf{T}} x_i$
    ▷ Receive cost c_{i,X_i}
    Construct the simulated cost vector $\hat{c}_i \in \mathbb{R}^n$ with

$$\hat{c}_{i,j} \;=\; \begin{cases} c_{i,j}/x_{i,j} & \text{(for } j = X_i\text{)} \\ 0 & \text{(otherwise)} \end{cases}$$

    for $j \leftarrow 1, ..., n$ **do**
      $y_{i+1,j} = y_{i,j} \exp(-\eta \hat{c}_{i,j})$ (decrease weight according to action j's simulated cost)

Let $c_i \in [0,1]^n$ denote the true cost of the actions in round i. Let $\mathcal{F}_i$ denote the sigma-field generated by $\hat{c}_1, ..., \hat{c}_i$.

The following claim says that, no matter what the algorithm has learned so far, in round i the simulated costs equal the actual costs in expectation. (This is quite similar to Lemma 3.2.)

Claim 4.1. Fix $i \in [T]$. For any index J that is $\mathcal{F}_{i-1}$-measurable, $\mathrm{E}[\,\hat{c}_{i,J} \mid \mathcal{F}_{i-1}\,] = c_{i,J}$.

Proof.

$$\begin{aligned} \mathrm{E}[\,\hat{c}_{i,J} \mid \mathcal{F}_{i-1}\,] &= \Pr[\,X_i = J \mid \mathcal{F}_{i-1}\,] \cdot \frac{c_{i,J}}{x_{i,J}} + \Pr[\,X_i \neq J \mid \mathcal{F}_{i-1}\,] \cdot 0 \\ &= x_{i,J} \cdot \frac{c_{i,j}}{x_{i,j}} \qquad \text{(since } J \text{ is } \mathcal{F}_{i-1}\text{-measurable)} \\ &= c_{i,j}. \end{aligned}$$

■

We will apply the Randomized Weighted Majority analysis of Theorem 2.1 in Lecture 16, with the simulated cost $\hat{c}_i$ rather than the actual costs c_i. That will introduce a quadratic term in the costs, which we first show how to deal with.

Claim 4.2. Fix $i \in [T]$. $\mathrm{E}\left[\sum_{j=1}^{n} x_{i,j}\hat{c}_{i,j}^2 \mid \mathcal{F}_{i-1}\right] \le n$.

Proof.

$$\begin{aligned}\mathrm{E}\left[\sum_{j=1}^{n} x_{i,j}\hat{c}_{i,j}^2 \mid \mathcal{F}_{i-1}\right] &= \sum_{\ell=1}^{n} \Pr[\, X_i = \ell \mid \mathcal{F}_{i-1}\,]\, x_{i,\ell}\Big(\frac{c_{i,\ell}}{x_{i,\ell}}\Big)^2 \\ &= \sum_{\ell=1}^{n} c_{i,\ell}^2 \;\le\; n\end{aligned}$$

The first equality is explained as follows: recall that, on the event $X_i = \ell$, we have $\hat{c}_{i,\ell} = c_{i,\ell}/x_{i,\ell}$ and $\hat{c}_{i,j} = 0$ for all $j \neq \ell$. Thus, after expanding the expectation, there is no longer any need to sum over j. The second equality holds because $\Pr[\, X_i = \ell \mid \mathcal{F}_{i-1}\,] = x_{i,\ell}$. The inequality holds because every real cost $c_{i,j}$ is at most 1. ■

Theorem 4.3. Let j^* be the action of minimum total cost. The expected cost of the algorithm satisfies

$$\mathrm{E}\left[\sum_{i=1}^{t} c_i^{\mathsf{T}} x_i\right] - \sum_{i=1}^{t} c_{i,j^*} \;\le\; \sqrt{2tn\ln n}.$$

References. See [3, Lemma 6.2].

Proof. We will apply Theorem 2.1 in Lecture 16 to the simulated cost vectors $\hat{c}_i$ with $z = e_{j^*}$. These cost vectors $\hat{c}_i$ are randomly chosen, but that irrelevant because RWM works for *all* cost vectors, even if they are *adversarially* chosen. Thus, (with probability 1)

$$\sum_{i=1}^{t}(\hat{c}_i^{\mathsf{T}} x_i - \hat{c}_i^{\mathsf{T}} z) \;\le\; \frac{\ln n}{\eta} + \frac{\eta}{2}\sum_{i=1}^{t}\sum_{j=1}^{n} x_{i,j}\hat{c}_{i,j}^2. \tag{4.1}$$

Consider the i^{th} term on the left-hand side. Observe that z is non-random and x_i is $\mathcal{F}_{i-1}$-measurable. So, by Claim 4.1,

$$c_i^{\mathsf{T}} x_i - c_i^{\mathsf{T}} z \;=\; \mathrm{E}\left[\hat{c}_i^{\mathsf{T}} x_i - \hat{c}_i^{\mathsf{T}} z \mid \mathcal{F}_{i-1}\right].$$

Now take the *unconditional* expectation of (4.1). The left-hand side becomes:

$$\mathrm{E}\left[\sum_{i=1}^{t} c_i^{\mathsf{T}} x_i - \sum_{i=1}^{t} c_i^{\mathsf{T}} z\right] \;=\; \mathrm{E}\left[\sum_{i=1}^{t} c_i^{\mathsf{T}} x_i\right] - \sum_{i=1}^{t} c_{i,j^*}. \tag{4.2}$$

Regarding the i^{th} term on the right-hand side of (4.1), Claim 4.2 gives the the bound

$$\mathrm{E}\left[\sum_{j=1}^{n} x_{i,j}\hat{c}_{i,j}^2 \mid \mathcal{F}_{i-1}\right] \le n.$$

Thus, taking the unconditional expectation, the right-hand side of (4.1) is at most

$$\frac{\ln n}{\eta} + \frac{\eta t n}{2}. \tag{4.3}$$

Combining (4.2) and (4.3) and setting $\eta = \sqrt{2\ln(n)/tn}$ proves the theorem. ■

References

[1] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[2] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. Gambling in a rigged casino: The adversarial multi-arm bandit problem. In *36th Annual Symposium on Foundations of Computer Science*, pages 322–331, 1995.

[3] Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3–4), 2015.