Online Learning tools for Online Algorithms

(6 years later)

Avrim Blum

Carnegie Mellon University

[Dagstuhl workshop on Online Algorithms, June-July 2002]

Basic tool: Combining expert advice (game-theoretic setting)

There are n "experts" / "strategies" / "states".

Loss version:

1. You (probabilistically) select an expert.
2. Each expert experiences a "loss" between 0 and ℓ_{max}. (You can see what they all are)
3. You experience the loss of your expert. Repeat.

→ Goal: expected loss not much worse than loss of best expert.

Assume adversary in charge of 2 only sees probability distribution from 1. I.e., repeated game playing.

Gain version: gains between 0 and g_{max}. You want expected gain almost as much as best expert.

Basic result: can do amazingly well using simple RWM/Hedge algorithm. [LW][FS][CFHHSW]...

Basic result [LW][FS][CFHHSW]...

For gains version:

$$\mathbf{E}[gain] \;\geq\; (1 - \varepsilon/2)G_{opt} - \frac{g_{max}}{\varepsilon}\ln n$$

(G_{opt} = total gain of best expert)

For losses version:

$$\mathbf{E}[loss] \;\lesssim\; (1 + \varepsilon/2)L_{opt} + \frac{\ell_{max}}{\varepsilon}\ln n$$

(Using $1 + \varepsilon/2$ as approx to $-\ln(1-\varepsilon)/\varepsilon$)

What if we add in a movement cost C? [BB]

→ Just need to multiply additive term by C and replace $\varepsilon/2$ by $3\varepsilon/2$.

→ Can save ℓ_{max} factor if you move second as in MTS (Swap steps 1 and 2)

→ Must assume oblivious adversary. Not repeated game setting.

Let's see some easy applications.

App: Picking a winner

- n buckets. [Think of as different companies]
- You are standing in one of them.
- At each time step, ball falls into some bucket. If it's *your* bucket, you get \$1.
- Then you can choose to move if you want.
- Game ends when fullest bucket has D balls.

This is hopeless if an opponent is tossing the balls based on knowing where you are(n't), so we assume oblivious adversary.

[AAFL]: Get $\Omega(D/\log n)$ if can only switch once. Get $\Omega(D)$ if can switch $O(\log n)$ times.

What do we get with RWM/Hedge?

$$\begin{aligned}\mathbf{E}[gain] &\geq (1-\varepsilon/2)D - \frac{1}{\varepsilon}\ln n \\ &= D - \sqrt{2D\ln n} \quad (\text{using } \varepsilon = \sqrt{\tfrac{\ln n}{2D}})\end{aligned}$$

App: Self-adjusting search trees

- Have n items in a search tree.
- Get a request: "5"
- Pay \$1 for each pointer followed. Can also perform rotations for cost of \$1 each.

Splay trees are $O(1)$-competitive w.r.t. best static tree. Can we get $(1+\varepsilon)$? Let's use experts alg:

- One expert per bst. (Roughly 2^n)
- Access causes each expert to incur loss based on depth.
- Movement cost between experts at most $4n$.

Guarantee:

$$\mathbf{E}[\text{cost}] \lesssim (1+3\varepsilon/2)[\text{best tree}] + O(n^2/\varepsilon).$$

Of course, computation time is horrendous... But [Kalai-Vempala'02] give efficient version.

Can this help at all for dynamic optimality conjecture?

Combining online algorithms online

Can view experts-with-movement-costs as a MTS where the goal is just to perform nearly as well as the best *state*. If space has diameter C, then get:

$$\mathbf{E}[cost] \leq \mathrm{OPT}(1+\varepsilon) + O\left(\frac{C}{\varepsilon}\ln n\right).$$

Or, can view in the context of combining online algorithms online:

- We have n online algorithms for some problem. Want to do nearly as well as the best *on the given request sequence.*
- [Azar-Broder-Manasse]: $O(\log n)$ factor worse. This is optimal in general.
- Online learning: if the state space has *bounded diameter* then you only need to lose a $1+\varepsilon$ factor.

"Bandit version": can do this *even if you only see the loss of your current state*. (But additive term gets worse). [Auer Cesa-Bianchi Freund Schapire] [Burch]

Algorithm and Analysis

Let's do "gains" version. Randomized Weighted-Majority / Hedge Algorithm then looks like this:

- Initialize weights $w_1, \ldots, w_n$ on the n experts to 1. Let $W = \sum_i w_i$.
- Pick expert i with probability $p_i = w_i / W$.
- Given a vector of gains $g_1, \ldots, g_n$, we update using the rule $w_i \leftarrow w_i(1+\varepsilon)^{\widehat{g}_i}$, where $\widehat{g}_i = \frac{g_i}{g_{max}}$.

Analysis:

- Say we are currently at time step t. Our expected gain $E_t = \frac{1}{W}\sum_i g_i w_i$. Let $\widehat{E}_t = E_t / g_{max}$.
- If W is the sum of weights before updating and W' is the sum after, then:

$$
\begin{aligned}
W' = \sum_i w_i(1+\varepsilon)^{\widehat{g}_i} \quad &\leq \quad \sum_i w_i(1+\varepsilon\widehat{g}_i) \\
&= \quad W + \varepsilon \sum_i w_i \widehat{g}_i \\
&= \quad W(1+\varepsilon\widehat{E}_t)
\end{aligned}
$$

Algorithm and Analysis (cont)

Recap: $W' \leq W(1+\varepsilon \widehat{E}_t)$, $\widehat{E}_t = \frac{1}{W}\sum_i \widehat{g}_i w_i$.

Since $W = n$ at the start of the game, at the end we have:

$$W_{final} \leq n \prod_t (1+\varepsilon \widehat{E}_t).$$

On the other hand, the best expert has weight $(1+\varepsilon)^{\widehat{G}_{opt}}$ and W_{final} is surely at least that.

Taking logs:

$$\widehat{G}_{opt} \ln(1+\varepsilon) \quad \leq \quad \ln n + \sum_t \ln(1+\varepsilon E_t).$$

Since $\ln(1+x) \in [x - x^2/2, x]$ for $x \in [0,1]$:

$$\widehat{G}_{opt}(\varepsilon - \varepsilon^2/2) \leq \ln n + \varepsilon \sum_t \widehat{E}_t.$$

So,

$$\sum_t \widehat{E}_t \geq \widehat{G}_{opt}(1-\varepsilon/2) - \frac{1}{\varepsilon}\ln n.$$

$$\mathbf{E}[gain] \geq G_{opt}(1-\varepsilon/2) - \frac{g_{max}}{\varepsilon}\ln n.$$

More applications: online auctions

Auctioning digital goods to online sequence of buyers.

Model of [Bar-Yossef Hildrum Wu]:

- Selling some good we have in infinite supply.
- Buyers come in one at a time. Each has valuation between 1 and h. Buyers never return.
- Want incentive-compatibility. E.g., set current price based only on previous bids.
- Goal: competitive wrt best fixed-price auction in hindsight. [BHW] get CR roughly $O(\log\log h)$.

Let's use experts algorithm. One expert per price level. Note that this is IC.

$\mathbf{E}[gain] \geq (1 - \varepsilon/2)G_{opt} - \frac{h}{\varepsilon}\ln n$.

Can use $n = h$ or $n = \log_{1+\varepsilon} h$. This is constant-competitive for $G_{opt} \geq h \ln h$.

What if there is a movement cost C?

- E.g., startup-company setup was unrealistic. You can't just switch jobs every day. (Vesting, etc.)

- To apply the same algorithm, we shouldn't think of picking *independently* from the new distrib at each time step.

  E.g., winner-picking problem with $\varepsilon = 1$:

w_j :	1	2	1	2	3	1
p_j :	1/10	1/5	1/10	1/5	3/10	1/10
w'_j :	1	4	1	2	3	1
p'_j :	1/12	1/3	1/12	1/6	1/4	1/12

Can view as: "if we were at position 2 then stay there. Else, move to position 2 with probability 1/6."

Movement cost analysis

Let's just do case of [000001000] gain-vectors.

In this case, the rule is: "if ball falls in location i then: if we were at position i, stay there. Else move to i with probability $\frac{\varepsilon p_i}{1+\varepsilon p_i}$."

How much does this hurt us?

- Movement probability $< \varepsilon p_i$, so expected movement cost $< C\varepsilon p_i$.
- But, remember our expected gain was p_i. So, we are really getting $(1 - C\varepsilon)$ times as much as we thought.
- So, decrease ε by factor of C to get same multiplicative factor, but additive term C times larger.

E.g., in winner-picking, the guarantee becomes

$$D - \sqrt{(4C + 2)2D \ln n},$$

which is $D(1 - o(1))$ for $C = o(D/\log n)$.

What can we use this for?

We saw for online data-structures (assuming computation time is free).

Also, online exchanges [BSZ]

- Buy and sell bids for some stock arrive and depart over time.
- Goal is to match bids to maximize profit.
- Optimal C.R. of $\ln(p_{max} - p_{min}) + 1$ achievable by picking profit threshold from exponential and then using a particular matching algorithm.
- Using experts algorithm, can do nearly as well as best threshold in hindsight...
  - if there is a bound on number of bids in play at any one time.
  - Different "experts" have different internal state, which can be simulated by movement cost.

Uniform-metric MTS

Which is better for uniform-metric MTS, Marking or RWM?

Bad case for Marking:

$$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ . & . & . & \end{array}$$

Bad case for RWM:

$$\begin{array}{cccc} 1 & 1 & 0 & 0 \\ . & . & . & \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ . & . & . & \\ 0 & 0 & 1 & 1 \end{array}$$

Experts-tracking gives nice “best of both”.

Best of both worlds

- [Herbster-Warmuth]: Weight-sharing version of RWM. [BB] analyze in game-theoretic setting with movement costs.

- Result is that for all k,

$$\mathbf{E}[\text{Alg loss}] \lesssim (1+\varepsilon)L_k + (\frac{1}{\varepsilon}\log n)(k+1)$$

  where L_k is loss of best strategy in hindsight that switches k times. (Assuming losses in $[0,1]$).

- In uniform MTS setting, can view as:

$$\mathbf{E}[\text{Alg cost}] \leq O(1) \times \text{OPT}_{local} + O(\log n) \times \text{OPT}_{move}.$$

  where

  – OPT_{local} = local cost of OPT.

  – OPT_{move} = movement cost of OPT.

- This means the alg behaves well in the unfair-MTS game of [Seiden].

Making efficient for online datastructures

Consider the list-update problem.

- COMB algorithm [Albers, von Stengel, Werchner] gets competitive ratio of 1.6.
- RWM gets *static* ratio of $(1+\varepsilon)$. But computation time is insane.
- Can we get good static ratio efficiently?
- Can we get both simultaneously?

Efficient static ratio for list-update

[BCK02]:

- Give each element i a random number $r_i \in [0, 1]$ subject to $r_1 + \ldots + r_n = 1$. Also give each a weight w_i initialized to 1.
- Order by $r_i w_i$.
- When element i is accessed, let $w_i \leftarrow w_i(1+\varepsilon)$. Then move it up in list if necessary to preserve order.

Then argue that for each pair, this is identical to 2-expert RWM. This implies desired result because costs can be decomposed pairwise.

Can actually use simpler algorithm, since we don't care if "$\log n$" term becomes $O(n)$ when $n = 2$.

[KV02] use more sophisticated trick to get efficient $(1 + \varepsilon)$ static optimality for trees.

Simultaneous static / dynamic optimal

Idea: just combine the two online algorithms online.

Bandit version allows you to not spend computation time on one while you are running the other.

(A few tricky issues)

Open problems

- Can we use these tools for some of the big problems: dynamic optimality in search trees? Randomized weighted caching?
- Can we handle non-uniform metric spaces better? Is there a natural extension of RWM when distance costs are non-uniform that gives better results?
- Other ways of using techniques to compete against dynamic offline.

References

[ASW95] Susanne Albers, Bernhard von Stengel, and Ralph Werchner. A combined bit and timestamp algorithm for the list update problem. *Information Processing Letters*, 56:135–139, 1995.

[ACBFS95] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. Gambling in a rigged casino: The adversarial multiarmed bandit problem In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 322-331, November 1995.

[BB00] A. Blum and C. Burch. On-line learning and the metrical task system problem. *Machine Learning*, 39(1):35–58, April 2000.

[BSZ02] A. Blum, T. Sandholm, and M. Zinkevich. Online Algorithms for Market Clearing. SODA '02, pages 971-980.

[BCK02] A. Blum, S. Chawla, and A. Kalai. Static Optimality and Dynamic Search-Optimality in Lists and Trees. SODA '02, pages 1-8.

[Bur00] C. Burch. Machine learning in metrical task systems and other on-line problems. *CMU Tech Report* CMU-CS-00-135, May 2000.

[CFHHSW97] N. Cesa-Bianchi, Y. Freund, D.P. Helmbold, D. Haussler, R.E. Schapire, and M.K. Warmuth. How to use Expert Advice. *Journal of the ACM*, 44(3):427–485, 1997.

[FS96] Yoav Freund and Robert Schapire. Game theory, on-line prediction and boosting. In *Proc. 9th Conf. on Computational Learning Theory*, pages 325–332, 1996.

[HW98] M. Herbster and M. Warmuth. Tracking the best expert. *Machine Learning*, *32*(2), 286–294, 1998.

[KV02] Adam Kalai and Santosh Vempala. Follow the Leader for Online Algorithms. Unpublished Manuscript.

[LW94] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.