Online Learning with Global Cost Functions

Shie Mannor

Technion EE

January 2010

Based on joint work with Eyal Even-dar (Google), Robert Kleinberg (Cornell), Yishay Mansour (TAU), John Tsitsiklis (MIT), and Jia Yuan Yu (McGill)

Table of contents

Introduction

- Many decisions to be made (high decision rate)
- Each decision is from a small or *structured* set
- A notion of "state" is weak: low temporal effect - each decision is "small"
- Regret = how much I have in my pocket - how much I could have had with hindsight
- An algorithm is **no-regret** if regret is "small"

Introduction

- Many decisions to be made (high decision rate)
- Each decision is from a small or *structured* set
- A notion of "state" is weak: low temporal effect - each decision is "small"
- Regret $=$ how much I have in my pocket - how much I could have had with hindsight
- An algorithm is **no-regret** if regret is "small"
- Lack of good model for the environment

Examples for Regret Minimization

- Routing in communication & ad-hoc networks
- Meta-classification: choosing between "experts" for online classification
- Load balancing
- Power management
- Paging

Standard Regret Minimization

Model: A actions, each with immediate loss: $\ell_t = \ell(a_t)$.
Cost of interest = total loss = $\sum_t \ell_t = \sum_t (\ell_t(a_t))$
Regret = $\sum_t \ell_t - \quad \min_a \sum_t (\ell_t(a))$
Regret = Actual cost - cost in hindsight.

Standard Regret Minimization

Model: A actions, each with immediate loss: $\ell_t = \ell(a_t)$.
Cost of interest = total loss $= \sum_t \ell_t = \sum_t(\ell_t(a_t))$
Regret $= \sum_t \ell_t - \quad \min_a \sum_t(\ell_t(a))$
Regret = Actual cost - cost in hindsight.

The good news

- N experts (full and partial information)
- Action can be in a convex set (works for convex loss)
- Can compare to richer classes of strategy
- Very simple algorithms

Standard Regret Minimization

Model: A actions, each with immediate loss: $\ell_t = \ell(a_t)$.
Cost of interest = total loss = $\sum_t \ell_t = \sum_t(\ell_t(a_t))$
Regret = $\sum_t \ell_t - \quad \min_a \sum_t(\ell_t(a))$
Regret = Actual cost - cost in hindsight.

The good news

- N experts (full and partial information)
- Action can be in a convex set (works for convex loss)
- Can compare to richer classes of strategy
- Very simple algorithms

The bad news

- There is no state
- Losses are assumed to be additive across time
- Most algorithms are essentially greedy

Regret Minimization with State

- Routing [AK]
- Control problems [EKM, YMS]
- Paging [BBK]
- Data structures [BCK]
- Power management [MTY] (later in the talk)
- Load balancing [EKMM] (focus of this talk)

Model (Load Balancing)

- N alternatives (machines)
- Algorithm chooses a load distribution $\bar{p}_t$ over the alternatives and then observes loss vector $\bar{\ell}_t$.
- Algorithm accumulated loss: $\bar{L}_t^A = \sum_{\tau=1}^t \bar{\ell}_\tau \cdot \bar{p}_\tau$
- Overall loss: $\bar{L}_t = \sum_{\tau=1}^t \bar{\ell}_\tau$
- Algorithm cost: $C(\bar{L}_t^A)$, where C is a **global cost function**.
- Optimal cost: $C^*(\bar{L}_t) = \min_{\alpha \in \Delta(N)} C(\alpha \cdot \bar{L}_t)$.
- Regret: $C(\bar{L}_t^A) - C^*(\bar{L}_t)$.

Model - load balancing with makespan

Assume makespan: $C = \| \cdot \|_\infty$.

Time	loss	Dist.	Alg Accu.	$C(Alg)$	Total loss	C^*
1	(1,1)	(.5,.5)	(.5,.5)	.5	(1,1)	.5

Model - load balancing with makespan

Assume makespan: $C = \|\cdot\|_\infty$.

Time	loss	Dist.	Alg Accu.	$C(Alg)$	Total loss	C^*
1	(1,1)	(.5,.5)	(.5,.5)	.5	(1,1)	.5
2	(1,0)	(.5,.5)	(1,.5)	1	(2,1)	.66

Model - load balancing with makespan

Assume makespan: $C = \| \cdot \|_\infty$.

Time	loss	Dist.	Alg Accu.	$C(Alg)$	Total loss	C^*
1	(1,1)	(.5,.5)	(.5,.5)	.5	(1,1)	.5
2	(1,0)	(.5,.5)	(1,.5)	1	(2,1)	.66
3	(1,0)	(.33,.66)	(1.33,.5)	1.33	(3,1)	.75

Assume makespan: $C = \| \cdot \|_\infty$.

Time	loss	Dist.	Alg Accu.	$C(Alg)$	Total loss	C^*
1	(1,1)	(.5,.5)	(.5,.5)	.5	(1,1)	.5
2	(1,0)	(.5,.5)	(1,.5)	1	(2,1)	.66
3	(1,0)	(.33,.66)	(1.33,.5)	1.33	(3,1)	.75
4	(0,1)	(.25,.75)	(1.33,1.25)	1.33	(3,2)	1.2

Model - load balancing with makespan

Assume makespan: $C = \| \cdot \|_\infty$.

Time	loss	Dist.	Alg Accu.	$C(Alg)$	Total loss	C^*
1	(1,1)	(.5,.5)	(.5,.5)	.5	(1,1)	.5
2	(1,0)	(.5,.5)	(1,.5)	1	(2,1)	.66
3	(1,0)	(.33,.66)	(1.33,.5)	1.33	(3,1)	.75
4	(0,1)	(.25,.75)	(1.33,1.25)	1.33	(3,2)	1.2

Note that minimizing the sum of losses does not minimize C^* and vice versa

Model - load balancing with makespan

Optimal policy in hindsight the load vector $\bar{L}$ is

$$p_i = \frac{1/L_i}{\sum_{j=1}^{N} 1/L_j}$$

Cost of the optimal policy is

$$C^*(\bar{L}) = \frac{1}{\sum_{j=1}^{N} 1/L_j} = \frac{\prod_{j=1}^{N} L_j}{\sum_{j=1}^{N} \prod_{i \neq j} L_i}$$

The adversarial case

No assumption on how the sequence is generated.

The adversarial case

No assumption on how the sequence is generated.

- Very complex model
- Model is plain wrong
- Varying loads

Main Theorem

Theorem

If C, the global cost function, is convex and C^* is concave, then we can construct an algorithm with no regret.

Main Theorem

Theorem

If C, the global cost function, is convex and C^* is concave, then we can construct an algorithm with no regret.

- Algorithm is not greedy
- Main observation: can optimize locally
- Requires solving an optimization problem at every step

(Approximate) Greedy is failing

Load balancing for makepsan metric

- Assume two machines, and global cost function is L_∞.
- If the total loads are L_1, L_2 then the makespan is $\frac{L_1 L_2}{L_1 + L_2}$

(Approximate) Greedy is failing

Load balancing for makepsan metric

- Assume two machines, and global cost function is L_∞.
- If the total loads are L_1, L_2 then the makespan is $\frac{L_1 L_2}{L_1+L_2}$
- First T steps loss vector is $(1,1)$.
- Last T steps loss vector is $(1,0)$.

(Approximate) Greedy is failing

Load balancing for makepsan metric

- Assume two machines, and global cost function is L_∞.
- If the total loads are L_1, L_2 then the makespan is $\frac{L_1 L_2}{L_1+L_2}$
- First T steps loss vector is $(1, 1)$.
- Last T steps loss vector is $(1, 0)$.

Consider an "improved" greedy algorithm:

Time	Alg. Dist	Alg load	OPT load
$[0, T]$	$(.5, .5)$	$(T/2, T/2)$	$(T/2, T/2)$

(Approximate) Greedy is failing

Load balancing for makepsan metric

- Assume two machines, and global cost function is L_∞.
- If the total loads are L_1, L_2 then the makespan is $\frac{L_1 L_2}{L_1+L_2}$
- First T steps loss vector is $(1, 1)$.
- Last T steps loss vector is $(1, 0)$.

Consider an "improved" greedy algorithm:

Time	Alg. Dist	Alg load	OPT load
$[0, T]$	$(.5, .5)$	$(T/2, T/2)$	$(T/2, T/2)$
$[T, 3T/2]$	$(.4, .6)$	$(7T/10, T/2)$	$(3T/5, 3T/5)$

(Approximate) Greedy is failing

Load balancing for makepsan metric

- Assume two machines, and global cost function is L_∞.
- If the total loads are L_1, L_2 then the makespan is $\frac{L_1 L_2}{L_1+L_2}$
- First T steps loss vector is $(1,1)$.
- Last T steps loss vector is $(1,0)$.

Consider an "improved" greedy algorithm:

Time	Alg. Dist	Alg load	OPT load
$[0, T]$	$(.5, .5)$	$(T/2, T/2)$	$(T/2, T/2)$
$[T, 3T/2]$	$(.4, .6)$	$(7T/10, T/2)$	$(3T/5, 3T/5)$
$[3T/2, 2T]$	$(.33, .66)$	$(13\text{T}/15, \text{T}/2)$	$(2T/3, 2T/3)$

Least loaded algorithm is failing

Counterexample

- At time 0 $(\epsilon, 0)$
- At odd times $(0, 1)$
- At even times $(1, 0)$

Least loaded algorithm is failing

Counterexample

- At time 0 $(\epsilon, 0)$
- At odd times $(0, 1)$
- At even times $(1, 0)$

- Optimal static allocation, $(1/2, 1/2)$, has cost of $T/4$
- Least loaded has a cost of $T/2$

An efficient algorithm for makespan - two machines

Algorithm

- At time $t = 1$: $p_1(1) = p_1(2) = 1/2$
- $p_{t+1}(1) = p_t(1) + \frac{p_t(2)\ell_t(2) - p_t(1)\ell_t(1)}{\sqrt{T}}$

An efficient algorithm for makespan - two machines

Algorithm

- At time $t = 1$: $p_1(1) = p_1(2) = 1/2$
- $p_{t+1}(1) = p_t(1) + \frac{p_t(2)\ell_t(2) - p_t(1)\ell_t(1)}{\sqrt{T}}$

Analysis ideas

- Partition $[0, 1]$ into intervals of size ϵ.
- Let T_i be the timesteps in which $p_1(t)$ was at interval i.
- Show that at T_i the algorithm is "calibrated", i.e. probability average is $i\epsilon$

An efficient algorithm for makespan - two machines

Algorithm

- At time $t = 1$: $p_1(1) = p_1(2) = 1/2$
- $p_{t+1}(1) = p_t(1) + \frac{p_t(2)\ell_t(2) - p_t(1)\ell_t(1)}{\sqrt{T}}$

Analysis ideas

- Partition $[0, 1]$ into intervals of size ϵ.
- Let T_i be the timesteps in which $p_1(t)$ was at interval i.
- Show that at T_i the algorithm is “calibrated”, i.e. probability average is $i\epsilon$

Convergence rate

Theorem

For any loss sequence, L, the regret is bounded by $O(\sqrt{T})$

Efficient algorithm for makespan - K-machines

Algorithm

Build a full binary tree where in each node we use the two machines algorithm as a black box.

Theorem

Suppose the global cost function is makespan. For any set K of 2^r alternatives, and for any loss sequence $\ell_1, \ldots, \ell_T$, algorithm A_{2^r} will have regret at most $O(\frac{\log |K|}{\sqrt{T}})$, i.e.,

$$C_\infty(L_T^{A_{2^r}}) - C_\infty^*(\ell) \leq 242 \frac{r}{\sqrt{T}}.$$

A Power Management Application

A real-world application (with Intel Research):

- Turn on/off voltage in CPUs
- Decision every $\approx$ 16msec:
  1. User don't need much CPU power
  2. User does need extra CPU power $\Rightarrow$ hiccup if voltage throttled down
- Objective: Maximize power saving

A Power Management Application

A real-world application (with Intel Research):

- Turn on/off voltage in CPUs
- Decision every $\approx$ 16msec:
  1. User don't need much CPU power
  2. User does need extra CPU power $\Rightarrow$ hiccup if voltage throttled down
- Objective: Maximize power saving
- But: Can't have too many hiccups.

This is inherently a problem with a state!

Solving the Power Management Problem

Modeling the power management problem:

Maximize power saving
Subject to: average hiccup rate $\leq$ threshold

- Problem can be solved by modifying the cost functions: penalize hiccups
- But cost functions depend on history
- Not so simple algorithms needed to solve the problem

Implemented as part of a low-level hardware solution

Summary & Outlook

- When to use online learning:
  - A good model is not available
  - Many frequent decisions, none too crucial
  - Care mostly about average performance
- Some very simple algorithms.
- Strong theoretical guarantees
- Adapted to different objectives: cumulative/global/constrained

Summary & Outlook

- When to use online learning:
  - A good model is not available
  - Many frequent decisions, none too crucial
  - Care mostly about average performance
- Some very simple algorithms.
- Strong theoretical guarantees
- Adapted to different objectives: cumulative/global/constrained

And .... it works!