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

1. Introduction
2. The Easy Case: No State
3. Cost Model
4. Learning with Global Cost
5. Power Management
6. Summary & outlook
Introduction

- Many decisions to be made (high decision rate)
- Each decision is from a small or \textit{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 \textbf{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 = $\sum_t \ell_t - \min_a \sum_t (\ell_t(a))$

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 = \( \sum_t \ell_t - \min_a \sum_t (\ell_t(a)) \)

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
Standard Regret Minimization

Model: A actions, each with immediate loss: $l_t = l(a_t)$.  
Cost of interest = total loss = $\sum_t l_t = \sum_t (l_t(a_t))$  
Regret = $\sum_t l_t - \min_a \sum_t (l_t(a))$  
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 $\tilde{\mathbf{p}}_t$ over the alternatives and then observes loss vector $\tilde{\mathbf{\ell}}_t$.
- Algorithm accumulated loss: $\tilde{L}^A_t = \sum_{\tau=1}^{t} \tilde{\ell}_\tau \cdot \tilde{p}_\tau$
- Overall loss: $\tilde{L}_t = \sum_{\tau=1}^{t} \tilde{\ell}_\tau$
- Algorithm cost: $C(\tilde{L}^A_t)$, where $C$ is a global cost function.
- Optimal cost: $C^*(\tilde{L}_t) = \min_{\alpha \in \Delta(N)} C(\alpha \cdot \tilde{L}_t)$.
- Regret: $C(\tilde{L}^A_t) - C^*(\tilde{L}_t)$. 
Assume makespan: $C = \| \cdot \|_\infty$.

<table>
<thead>
<tr>
<th>Time</th>
<th>loss</th>
<th>Dist.</th>
<th>Alg Accu.</th>
<th>$C(Alg)$</th>
<th>Total loss</th>
<th>$C^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1,1)</td>
<td>(.5,.5)</td>
<td>(.5,.5)</td>
<td>.5</td>
<td>(1,1)</td>
<td>.5</td>
</tr>
</tbody>
</table>
Assume makespan: $C = \| \cdot \|_{\infty}$.

<table>
<thead>
<tr>
<th>Time</th>
<th>loss</th>
<th>Dist.</th>
<th>Alg Accu.</th>
<th>$C(Alg)$</th>
<th>Total loss</th>
<th>$C^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1,1)</td>
<td>(.5,.5)</td>
<td>(.5,.5)</td>
<td>.5</td>
<td>(1,1)</td>
<td>.5</td>
</tr>
<tr>
<td>2</td>
<td>(1,0)</td>
<td>(.5,.5)</td>
<td>(1,.5)</td>
<td>1</td>
<td>(2,1)</td>
<td>.66</td>
</tr>
</tbody>
</table>
Assume makespan: \( C = \| \cdot \|_\infty \).

<table>
<thead>
<tr>
<th>Time</th>
<th>loss</th>
<th>Dist.</th>
<th>Alg Accu.</th>
<th>( C(Alg) )</th>
<th>Total loss</th>
<th>( C^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1,1)</td>
<td>(.5,.5)</td>
<td>(.5,.5)</td>
<td>.5</td>
<td>(1,1)</td>
<td>.5</td>
</tr>
<tr>
<td>2</td>
<td>(1,0)</td>
<td>(.5,.5)</td>
<td>(1,.5)</td>
<td>1</td>
<td>(2,1)</td>
<td>.66</td>
</tr>
<tr>
<td>3</td>
<td>(1,0)</td>
<td>(.33,.66)</td>
<td>(1.33,.5)</td>
<td>1.33</td>
<td>(3,1)</td>
<td>.75</td>
</tr>
</tbody>
</table>
Model - load balancing with makespan

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

<table>
<thead>
<tr>
<th>Time</th>
<th>loss</th>
<th>Dist.</th>
<th>Alg Accu.</th>
<th>$C(Alg)$</th>
<th>Total loss</th>
<th>$C^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1,1)</td>
<td>(.5,.5)</td>
<td>(.5,.5)</td>
<td>.5</td>
<td>(1,1)</td>
<td>.5</td>
</tr>
<tr>
<td>2</td>
<td>(1,0)</td>
<td>(.5,.5)</td>
<td>(1,.5)</td>
<td>1</td>
<td>(2,1)</td>
<td>.66</td>
</tr>
<tr>
<td>3</td>
<td>(1,0)</td>
<td>(.33,.66)</td>
<td>(1.33,.5)</td>
<td>1.33</td>
<td>(3,1)</td>
<td>.75</td>
</tr>
<tr>
<td>4</td>
<td>(0,1)</td>
<td>(.25,.75)</td>
<td>(1.33,1.25)</td>
<td>1.33</td>
<td>(3,2)</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Assume makespan: \( C = \| \cdot \|_{\infty} \).

<table>
<thead>
<tr>
<th>Time</th>
<th>loss</th>
<th>Dist.</th>
<th>Alg Accu.</th>
<th>( C(Alg) )</th>
<th>Total loss</th>
<th>( C^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1,1)</td>
<td>(.5,.5)</td>
<td>(.5,.5)</td>
<td>.5</td>
<td>(1,1)</td>
<td>.5</td>
</tr>
<tr>
<td>2</td>
<td>(1,0)</td>
<td>(.5,.5)</td>
<td>(1,.5)</td>
<td>1</td>
<td>(2,1)</td>
<td>.66</td>
</tr>
<tr>
<td>3</td>
<td>(1,0)</td>
<td>(.33,.66)</td>
<td>(1.33,.5)</td>
<td>1.33</td>
<td>(3,1)</td>
<td>.75</td>
</tr>
<tr>
<td>4</td>
<td>(0,1)</td>
<td>(.25,.75)</td>
<td>(1.33,1.25)</td>
<td>1.33</td>
<td>(3,2)</td>
<td>1.2</td>
</tr>
</tbody>
</table>

Note that minimizing the sum of losses does not minimize \( C^* \) and vice versa.
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 $L_\infty$.
- 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 makespan metric

- Assume two machines, and global cost function is $L_\infty$.
- 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 makespan metric

- Assume two machines, and global cost function is $L_\infty$.
- 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:

<table>
<thead>
<tr>
<th>Time</th>
<th>Alg. Dist</th>
<th>Alg load</th>
<th>OPT load</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[0, T]$</td>
<td>(.5, .5)</td>
<td>$(T/2, T/2)$</td>
<td>$(T/2, T/2)$</td>
</tr>
</tbody>
</table>
Assume two machines, and global cost function is $L_\infty$.

If the total loads are $L_1, L_2$ then the makespan is \(\frac{L_1L_2}{L_1+L_2}\).

First $T$ steps loss vector is $(1, 1)$.

Last $T$ steps loss vector is $(1, 0)$.

Consider an “improved” greedy algorithm:

<table>
<thead>
<tr>
<th>Time</th>
<th>Alg. Dist</th>
<th>Alg load</th>
<th>OPT load</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, $T$]</td>
<td>(.5, .5)</td>
<td>($T/2$, $T/2$)</td>
<td>($T/2$, $T/2$)</td>
</tr>
<tr>
<td>[$T$, $3T/2$]</td>
<td>(.4, .6)</td>
<td>($7T/10$, $T/2$)</td>
<td>($3T/5$, $3T/5$)</td>
</tr>
</tbody>
</table>
Load balancing for makespan metric

- Assume two machines, and global cost function is $L_{\infty}$.
- 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:

<table>
<thead>
<tr>
<th>Time</th>
<th>Alg. Dist</th>
<th>Alg load</th>
<th>OPT load</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[0, T]$</td>
<td>(.5, .5)</td>
<td>$(T/2, T/2)$</td>
<td>$(T/2, T/2)$</td>
</tr>
<tr>
<td>$[T, 3T/2]$</td>
<td>(.4, .6)</td>
<td>$(7T/10, T/2)$</td>
<td>$(3T/5, 3T/5)$</td>
</tr>
<tr>
<td>$[3T/2, 2T]$</td>
<td>(.33, .66)</td>
<td>$(13T/15, T/2)$</td>
<td>$(2T/3, 2T/3)$</td>
</tr>
</tbody>
</table>
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 $\epsilon$.
- 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 \( t = 1 \): \( p_1(1) = p_1(2) = 1/2 \)
- \( p_{t+1}(1) = p_t(1) + \frac{p_t(2)l_t(2) - p_t(1)l_t(1)}{\sqrt{T}} \)

Analysis ideas

- Partition \([0, 1]\) into intervals of size \( \epsilon \).
- 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 $O\left(\frac{\log |K|}{\sqrt{T}}\right)$, i.e.,

$$ C_\infty(L_{T}^{A_{2^r}}) - C_\infty^*(\ell) \leq 242 \frac{r}{\sqrt{T}}. $$
A real-world application (with Intel Research):

- Turn on/off voltage in CPUs
- Decision every ≈ 16msec:
  1. User don’t need much CPU power
  2. User does need extra CPU power ⇒ hiccup if voltage throttled down
- Objective: Maximize power saving
A real-world application (with Intel Research):

- Turn on/off voltage in CPUs
- Decision every \( \approx 16 \text{msec} \):
  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

And ... it works!

Shie Mannor (Technion EE)

Online Learning with Global Cost Functions
January 2010
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!