Online Algorithm and Online Optimization

Henry Vu

Online Algorithm - An Example

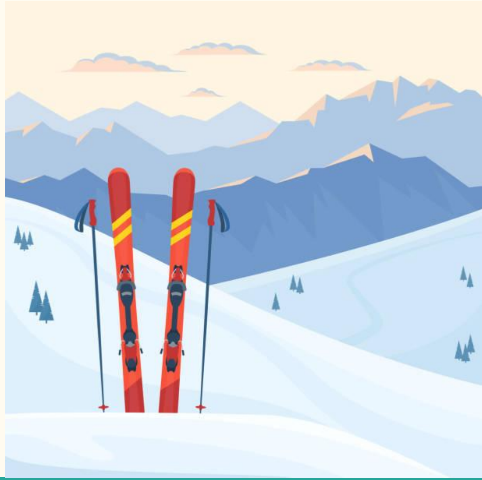
- **Ski rental problem**:

  At each day, player can decide to **rent** at $1, or **buy** at $B

  If ski is bought, no need to rent (i.e. no future cost)

- **On day d, should you buy or rent?**

  Need to know when the player stops skiing



Ski Rental

- If stop date **D** is known beforehand

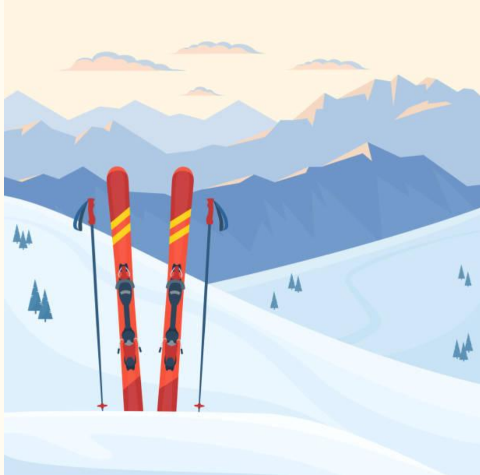
  D ≥ **B**: buy on day 1

  D < **B**: rent every day

  Cost = min{D, B}

- **But what if D is unknown?**

  Need to design an **online** algorithm that minimizes cost.



Competitive Online Algorithms

- Algorithm design where problem parameters σ's are revealed sequentially

- **How to evaluate performance?** Compare how the online algorithm (ALG) compete vs its offline counterpart (OPT) that knows everything

Competitive ratio: $$\alpha = \frac{Cost(ALG)}{Cost(OPT)}$$

- **Goal:** minimize α in the worst case (also called worst-case competitive analysis)

Competitive Ratios

- A **deterministic online algorithm** is called α-competitive if:

$$\alpha = \max_{\sigma} \frac{ALG(\sigma)}{OPT(\sigma)}$$

- A **randomized online algorithm** (a set of deterministic algorithms over some probability distribution) is called α-competitive if:

$$\alpha = \max_{\sigma} \frac{\mathbb{E}[ALG(\sigma)]}{OPT(\sigma)}$$

Back to Ski Rental

- Problem parameter σ = D

- **Offline algorithm:** OPT(D) = min{D, B}

- **Deterministic online algorithm:** decides to buy on some day **d** (renting up to day d-1)

  if d ≤ D i.e. buy ski before actually stop

  ALG(D) = d-1 + B

  if d > D i.e. rent until stop

  ALG(D) = D

Competitive Analysis

- In the worst case scenario, the player is forced to stop right after buying i.e. d = D

$$\alpha = \max_D \frac{ALG(D)}{OPT(D)} = \max_D \frac{d-1+B}{\min(D,B)} = \frac{d-1+B}{\min(d,B)}$$

- The solution d* that minimizes α is d* = B

  ALG(D) = B - 1 + B = 2B - 1

- The best possible **deterministic online algorithm** has almost **twice the cost!**

Can we do better?

- Try **randomization**: at the start, choose algorithm ALG_d with probability p_d where ALG_d: rent for d-1 days then buy on day d

  Assume α-competitiveness:

$$\mathbb{E}[ALG(\sigma)] \leq \alpha.OPT(\sigma)$$

$$\Rightarrow \sum_{i=1}^{D}(i-1+B).p_i + \sum_{i>D} D.p_i \leq \alpha.\min(D,B)$$

An example: B = $4

- We have the following LP

$$
\begin{aligned}
4p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + \ldots &\leq 1.\alpha \quad \text{for D} = 1 \\
4p_1 + 5p_2 + 2p_3 + 2p_4 + 2p_5 + 2p_6 + \ldots &\leq 2.\alpha \quad \text{for D} = 2 \\
4p_1 + 5p_2 + 6p_3 + 3p_4 + 3p_5 + 3p_6 + \ldots &\leq 3.\alpha \quad \text{for D} = 3 \\
4p_1 + 5p_2 + 6p_3 + 7p_4 + 4p_5 + 4p_6 + \ldots &\leq 4.\alpha \quad \text{for D} = 4 \\
4p_1 + 5p_2 + 6p_3 + 7p_4 + 8p_5 + 5p_6 + \ldots &\leq 4.\alpha \quad \text{for D} = 5 \\
4p_1 + 5p_2 + 6p_3 + 7p_4 + 8p_5 + 9p_6 + \ldots &\leq 4.\alpha \quad \text{for D} = 6 \\
&\ldots
\end{aligned}
$$

An example: B = $4

- Table of **competitive ratios**

	D = 1	D = 2	D = 3	D ≥ 4
ALG_1	4/1	4/2	4/3	4/4
ALG_2	1/1	5/2	5/3	5/4
ALG_3	1/1	2/2	6/3	6/4
ALG_4	1/1	2/2	3/3	7/4
ALG_5	1/1	2/2	3/3	8/4

An example: B = $4

- End up with:

$$4p_1 + p_2 + p_3 + p_4 \leq 1.\alpha$$
$$4p_1 + 5p_2 + 2p_3 + 2p_4 \leq 2.\alpha$$
$$4p_1 + 5p_2 + 6p_3 + 3p_4 \leq 3.\alpha$$
$$4p_1 + 5p_2 + 6p_3 + 7p_4 \leq 4.\alpha$$

- α is minimized when all equalities hold

- In general, solving for p_d then α based on the assumption that $\Sigma p_d = 1$, we obtain:

$$p_i = \left(\frac{B-1}{B}\right)^{B-i} \cdot \frac{\alpha}{B}$$
$$\alpha \simeq \frac{e}{e-1} = 1.582$$

Better than the optimal deterministic algorithm for large B

Does randomization always improve competitiveness?

- How to establish a **lower bound** for the competitive ratio of randomized algorithms?

- How to prove that the competitive ratio of some randomized algorithm is the **best possible** for all randomized algorithms

Yao’s Principle

Yao's Principle

Statement:

The expected cost of a **randomized algorithm** on the **worst-case input** is no better than the expected cost for a **worst-case probability distribution on the inputs** of the **deterministic algorithm** that performs best against that distribution.

Yao’s Principle: Proof

- The principle states that: Let $\mathscr{X}$ and A be the set of all inputs and deterministic algorithms respectively.

- For any input x $\in \mathscr{X}$ and algorithm a $\in$ A, let c(a, x) ≥ 0 be the cost of running algorithm a on input x.

- Let X $\in \mathscr{X}$ be a random input chosen according to some probability distribution p, and A_r be a random algorithm chosen according to some probability distribution q.

$$\max_{x \in \mathcal{X}} \mathbb{E}[c(A_r, x)] \geq \min_{a \in A} \mathbb{E}[c(a, X)]$$

Yao's Principle: Proof (cont.)

$$\max_{x \in \mathcal{X}} \mathbb{E}\underbrace{[c(A_r, x)]}_{\sum_a q_a c(a, x)} \geq \min_{a \in A} \mathbb{E}\underbrace{[c(a, X)]}_{\sum_x p_x c(a, x)}$$

$$\begin{aligned}
&\max_{x \in \mathcal{X}} \mathbb{E}[c(A_r, x)] \\
&= \max_{x \in \mathcal{X}} \sum_a q_a c(a, x) \geq \sum_x p_x \sum_a q_a c(a, x) = \sum_a q_a \sum_x p_x c(a, x) = \sum_a q_a \mathbb{E}[c(a, X)] \\
&\geq \min_{a \in A} \mathbb{E}[c(a, X)]
\end{aligned}$$

Another example: One-way trading

- When to trade your assets? Trade now or wait for better rates?

- **Example:** Consider a trader who wants to trade 1 CAD to USD.
  At time t (t = 1, 2,..., T), rate v_t is revealed. Should the trader **wait** or **buy now**? If buy, how much to buy?

  Objective: maximize the amount of USD

Problem Formulation

- At each step t, trade x_t CAD for USD at rate v_t
- Online Optimization Problem: input v_t's are revealed sequentially

$$\begin{aligned} \underset{x_t}{\text{minimize}} \quad & \sum_{t=1}^{T} v_t x_t \\ \text{subject to} \quad & \sum_{t=1}^{T} x_t \leq 1, \\ & x_t \geq 0, \quad \forall t \end{aligned}$$

Competitive Analysis

- **Assumptions:**
  - Exchange rates (prices) are bounded: $v_t \in [m, M]$ where $0 < m \leq M$
  - M and m are known to the online player
  - Can only sell 1 unit at each step

- The optimal deterministic algorithm is called ***reservation price policy*** (RPP):
  Accept the first price greater or equal to some *reservation price* p*

- How to choose p* that maximizes return?

How to choose p*?

- Consider 2 cases:

  1. The search encounters some price ≥ p*. The worst case return is p*

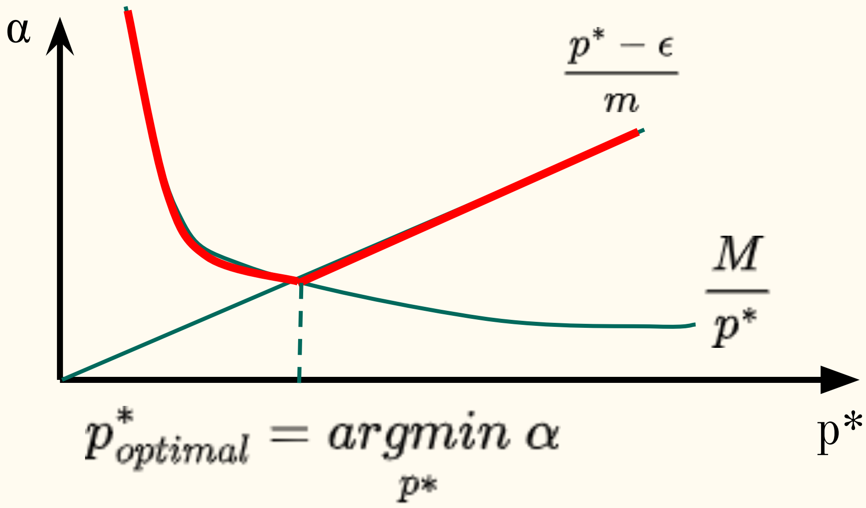
$$\alpha = \max_{\sigma} \frac{OPT(\sigma)}{ALG(\sigma)} = \frac{M}{p*}$$

  2. The search doesn't encounter any price ≥ p*, thus, chooses the last price. The worst case return is m

$$\alpha = \max_{\sigma} \frac{OPT(\sigma)}{ALG(\sigma)} = \frac{p * - \epsilon}{m} \quad \text{for } \epsilon \to 0^+$$

How to choose p*? (cont.)

- The competitive ratio is the worst case of of the 2: $\alpha = \max\{\frac{M}{p^*}, \frac{p^* - \epsilon}{m}\}$



So the optimal choice of p* is p* = $\sqrt{Mm}$

Online Primal Dual Approach

- Let p(σ) be the cost of the primal problem and d(σ) be the cost of its dual counterpart

- To achieve α-competitiveness, we need $\alpha = \max_{\sigma} \frac{p^*(\sigma)}{p_T(\sigma)}$ or $p_T(\sigma) \geq \frac{1}{\alpha} p^*(\sigma)$

- By weak duality, $p_t(\sigma) \geq \frac{1}{\alpha} d_t(\sigma) \geq \frac{1}{\alpha} d^*(\sigma) \geq \frac{1}{\alpha} p^*(\sigma)$

$$\sum_{t=1}^{T} (p_t(\sigma) - p_{t-1}(\sigma)) + p_0(\sigma)$$

$$\sum_{t=1}^{T} (d_t(\sigma) - d_{t-1}(\sigma)) + d_0(\sigma)$$

Online Primal Dual Approach (cont.)

- So to ensure α-competitiveness, we need to design a **trajectory** of feasible decision such that: $\alpha.\Delta_p \geq \Delta_d$

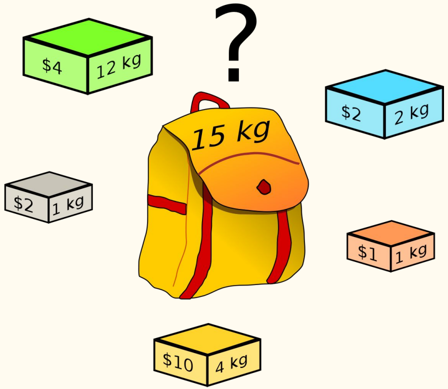
- **Challenges:**

  How to design such trajectory?
  How to minimize α?
  How to satisfy the boundary conditions at each step t in the design?

Online Knapsack Problem

- Classic (Offline) Knapsack Problem: Given a set of items, each with a weight and a value, choose some items such that their total weights is less than the total capacity of the knapsack while their total value is as large as possible.

- Online Knapsack Problem: The value and weight of each item is revealed **sequentially**. At each iteration t, determine whether or not to take the item.



Online Knapsack: Assumptions

- **Assumption 1**: each item's weight is much smaller than the capacity of the knapsack, i.e. $\exists \varepsilon$ close to 0 s.t $w_t/B \leq \varepsilon$.

- **Assumption 2**: the value-to-weight ratio of each item are bounded. i.e. $\exists$ L, U ≥ 0 s.t.

$$L \leq \frac{v_t}{w_t} \leq U$$

A simple deterministic algorithm

- Use **threshold**: If the item is "good enough", i.e. its value-to-ratio is greater than some amount, pick that item.

- The algorithm becomes increasingly selective as knapsack capacity becomes scarcer.

For t = 1, 2, ..., T

$$If\ \frac{v_t}{w_t} \geq p_t,\ \text{accept} \longrightarrow x_t = 1$$

$$\text{Else if } \frac{v_t}{w_t} < p_t,\ \text{reject} \longrightarrow x_t = 0$$

$$p_t = \varphi(y_{t-1}) \quad \text{where } y_t = \sum_{t=1}^{T} w_t x_t$$

is an increasing function to reflect pickiness as more space is filled

Intuitions about threshold price p_t

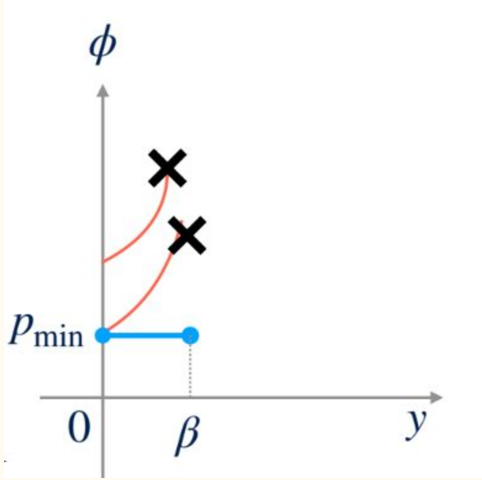
- At the beginning when the knapsack is empty ($y \in [0, \beta]$), it could be that $p_t \leq L \leq \frac{v_t}{w_t} \quad \forall t$, i.e. we **accept all items** until the knapsack is filled to a certain point.

- If p_t increases too rapidly, and/or starts at some value ≥ L, then

OPT makes use of entire knapsack

$$\alpha = \max_{\sigma} \frac{OPT(\sigma)}{ALG(\sigma)} \geq \frac{OPT(p_{min})}{ALG(p_{min})} = \frac{p_{min}.1}{p_{min}.\,w_1} \longrightarrow \infty$$

ALG only able to accept first item

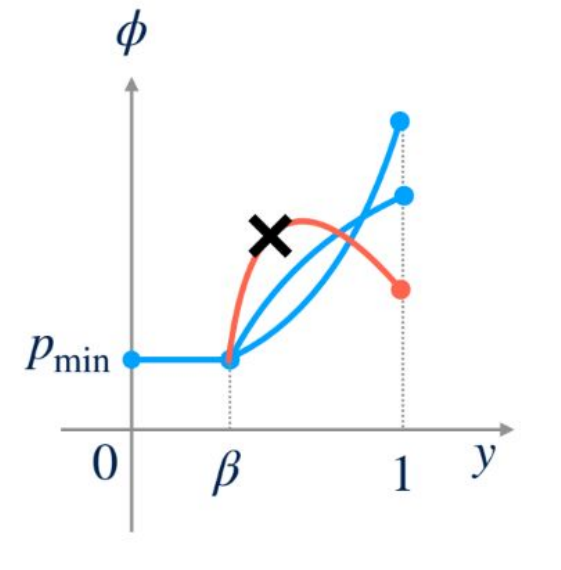


Intuitions about p_t (cont.)

- We can henceforth assume **p_t = L for $y \in [0, \beta]$**

- p_t must be **non-decreasing for $y \in [\beta, 1]$** to reflect scarcity

$\boxed{\text{How to design } \beta \text{ and } p_t\text{?}}$



Online Knapsack: LP Formulation

- Knapsack capacity is normalized to be 1. Item $_t$ (t = 1, 2, ..., T) has weight w_t and value v_t.

$$\begin{aligned} \underset{x_t}{\text{maximize}} \quad & \sum_{t=1}^{T} v_t x_t \\ \text{subject to} \quad & \sum_{t=1}^{T} w_t x_t \leq 1, \\ & x_t \in \{0,1\} \quad \forall t \end{aligned}$$

Primal Problem P(σ)

$$\begin{aligned} \underset{x_t}{\text{maximize}} \quad & \sum_{t=1}^{T} v_t x_t \\ \text{subject to} \quad & \sum_{t=1}^{T} w_t x_t \leq 1, \\ & 0 \leq x_t \leq 1 \quad \forall t \end{aligned}$$

Relaxed Primal Problem $P_r(\sigma)$

$$\begin{aligned} \underset{\lambda,\, \mu_t}{\text{minimize}} \quad & \lambda + \sum_{t=1}^{T} \mu_t \\ \text{subject to} \quad & v_t - \lambda w_t \leq \mu_t \quad \forall t, \\ & \lambda \geq 0, \\ & \mu_t \geq 0 \quad \forall t \end{aligned}$$

Dual Problem D(σ)

How to design? OPD Approach

From the complementary slackness condition:

$$\mu_t > 0 \Rightarrow x_t = 1$$

$$\text{and } v_t - \lambda w_t - \mu_t < 0 \Rightarrow x_t = 0$$

$$\frac{v_t}{w_t} > \lambda \Leftrightarrow v_t - \lambda w_t \geq 0$$

From the inequality constraint of the dual problem

$$\mu_t \geq v_t - \lambda w_t \geq 0 \Rightarrow x_t = 1$$

$$\frac{v_t}{w_t} < \lambda \Leftrightarrow v_t - \lambda w_t < 0$$

From the inequality constraint of the dual problem

$$\mu_t \geq 0 \Rightarrow v_t - \lambda w_t - \mu_t < 0 \Rightarrow x_t = 0$$

How to design? OPD Approach

- We design λ_t = p_t = $\varphi(y_t)$ where y_t captures resource utilization after step t

- Since the primal optimal solution largely depends on the dual optimal solution, we rely on λ_{t-1} to design the online primal decision x_t

- $x_t = \begin{cases} 1, & \frac{v_t}{w_t} \geq \lambda_{t-1} = \phi(y_{t-1}) \\ 0, & \frac{v_t}{w_t} < \lambda_{t-1} = \phi(y_{t-1}) \end{cases}$

- In addition, we design $\mu_t = \max\{0, v_t - \lambda_{t-1} w_t\}$

How to design? OPD Approach

- Now what we have left is to ensure **incremental inequality** holds $\alpha.\Delta_p \geq \Delta_d$

- To achieve this, we need to design φ such that

$$\phi(y_t) \leq \alpha\phi(y) \quad \text{for } y \in [0, 1]$$

Design Principles

- There is a critical threshold β such that:

  $\alpha \geq 1/\beta$

  $\varphi(y) =$ (or $\leq$) L for $y \in [0, \beta]$

- For the increasing segment $y \in [\beta, 1]$:

  $\varphi'(y) \leq \alpha\varphi(y)$

- $\varphi(1) \geq U$

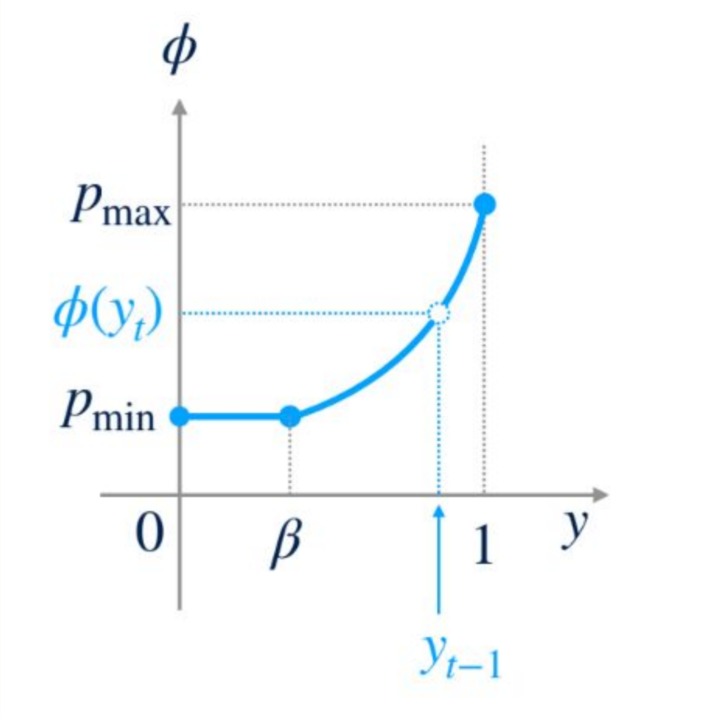
Optimal Threshold Design

$$\phi(y) = (\frac{Ue}{L})^y(\frac{L}{E})$$

$$\beta = \frac{1}{1 + ln(\frac{U}{L})}$$

$$\alpha = \frac{1}{\beta} = 1 + ln(\frac{U}{L})$$

Threshold ALG achieves constant competitive ratio



Best possible among all online algorithm!

Proof: Yao's Principle

Online Convex Optimization

- **Online Convex Optimization (OCO)**: rich theoretical area with a wide range of important applications.
  In an OCO problem, an agent interacts with the environment in sequence.
  At each time t (t = 1, 2, …, T):
  i, The agent chooses action x_t
  ii, The environment reveals cost function c_t
  iii, The agent experiences cost $c_t(x_t)$
  Objective: Minimizes the *cumulative cost* over T (T is large) rounds

- Example: The well-known k-experts problem

K-Experts Problem

- At each time t, the agent has to choose action A or B (e.g. to buy or sell stocks). There are N experts that offer advice. After a decision is made, the correct action is revealed and the agent accumulates some loss (let's say 0 if the agent chose the correct decision and 1 otherwise)

- **Observation:** Let $L \geq T/2$ be the number of mistakes the best expert makes. Then there does not exist a deterministic algorithm that can guarantee less than $2L$ mistakes.

K-Experts Problem with Deterministic Algorithm

- **Proof**: Assume that there are 2 experts and one always chooses A and the other always chooses B. Since our algorithm is deterministic, the adversary can always choose a sequence such that our total mistakes at the end is T. In this settings, the best expert makes no more than T/2 mistakes
$\Rightarrow$ A deterministic algorithm makes at least 2L mistakes

- This observation motivates the design of a **randomized algorithm** for this problem.

The Weighted Majority Algorithm

- **Intuition:** Each expert is initially assigned with some weight. At iteration t, if the total weights of all experts that choose A ($W_t(A)$) is ≥ the total weights of all experts that choose A ($W_t(B)$), the agent chooses A (or B if $W_t(B) \geq W_t(A)$) If an expert is wrong at iteration t, they have less influence on the agent's decision in the future (t+1, t+2, ..., T)

- Specifically, for expert i, $w_{t+1}(i) = \begin{cases} w_t(i), & \text{if i was right} \\ (1-\epsilon)w_t(i), & \text{if i was wrong} \end{cases}$

- $x_t = \begin{cases} A, & \text{if } W_t(A) \geq W_t(B) \\ B, & \text{otherwise} \end{cases}$

$$where\ W_t(A) = \sum_{\text{i that chose A}} w_t(i)$$

$$and\ W_t(B) = \sum_{\text{i that chose B}} w_t(i)$$

Performance Bounds

- **Theorem 1**: Supposed $\varepsilon \in [0, ½]$ and the best expert makes L mistakes and the algorithm makes M mistakes. Then:

  1. There is an efficient **deterministic algorithm** that can guarantee

     $M \leq 2(1+\varepsilon)L + 2\log N/\varepsilon$

  2. There is an efficient **randomized algorithm** that can guarantee

     $E[M] \leq (1+\varepsilon)L + \log N/\varepsilon$

  This randomized algorithm is called **Randomized Weighted Majority** (RWM). In RWM, the agent chooses expert i at time t with probability

$$p_t(i) = \frac{w_t(i)}{\sum_{j=1}^{N} w_t(j)}$$

Online Convex Optimization

- General framework:

Input: set of possible decisions (convex set C)

For t = 1, 2, ..., T:

. Make decision xt $\in$ C

. Loss is revealed after decision is made $f_t(x_t)$

Objective: Minimize the cumulative loss:

$$\min_{x_t \in C} \sum_{t=1}^{T} f_t(x_t)$$

Two Communities, Two Metrics

- The goal of the OCO community is most typically to develop algorithms that perform well with respect to **regret**: the difference between the cost of the online algorithm and the cost of the offline optimal fixed (static solution).

$$Regret = \sum_{t=1}^{T} f_t(x_t) - \min_{u \in C} \sum_{t=1}^{T} f_t(u)$$

- The online algorithm community focuses on **competitive ratio,** which is the worst-case ratio between the cost of the online algorithm and the cost of the optimal offline solution (dynamic solution).

$$\alpha = \max_{\sigma} \frac{ALG(\sigma)}{OPT(\sigma)}$$

Two Communities, Two Metrics

- The variation of OCO that the Online Algorithm community focuses on differs in that the cost $c_t(x_t)$ is revealed to the agent at iteration t before the agent makes a decision. This is the main difference between the two communities
  optimal dynamic solution (CR) vs optimal static solution (Regret)

- It is desirable for an algorithm to learn both dynamic and static concepts