There is always uncertainty in future. How to cope with uncertain future?
Online Algorithm is like Playing Chess with Adversary
Online Decision Problems

**Online decision problems:**
- Problems are *not* always solved in one shot, but progressively and continually
- Consider this decision problem
  - Input is revealed gradually as time evolves
  - A decision has to be made from time to time, given partial input
  - But an optimal decision depends on all future input (so cannot make optimal decision)
  - Decisions made cannot be retracted
- Examples:
  - When should we sell/buy in stock markets?
  - How much should you earn for retirement?
  - How to find your true love?
  - How much should a student learn to pass an exam?

**Online algorithms:**
- Solve online decision problems without knowing the entire input from the start to the end
- Motto: *Always prepare for the worst-case scenario; if it is the best you’ll win anyway*
Definition (Online Algorithms & Competitive Ratio)

- There is a sequence of input $I = (x_1, x_2, x_3, \ldots, x_T)$
- Online algorithm $\mathcal{A}$ makes a sequence of decisions $O_\mathcal{A} = (o_1, o_2, o_3, \ldots, o_T)$
  - Each decision $o_t$ only depends on $(x_1, \ldots, x_t)$
- Let Opt be an offline optimal solution
  - Each decision $o_t$ can depend on whole $(x_1, x_2, x_3, \ldots, x_T)$
- Let $\text{Cost}_\mathcal{A}(I, O)$ be the cost of decisions $O$ and input $I$
- Let the competitive ratio be $\alpha(\mathcal{A}) \triangleq \max_I \frac{\text{Cost}(I, O_\mathcal{A})}{\text{Cost}(I, \text{Opt})}$
- Online algorithm $\mathcal{A}$ is called $c$-competitive if $\alpha(\mathcal{A}) \leq c$

- Let the competitive ratio be $\alpha(\mathcal{A}) \triangleq \max_I \frac{\text{Value}(I, \text{Opt})}{\text{Value}(I, O_\mathcal{A})}$, if we maximize value, instead
- Competitive ratio is like approximation ratio – competitive ratio benchmarks online algorithms with partial information vs. offline algorithms with full information
Playing a Zero-sum Game with Adversary

- **Game Tree:**
  - Player and Adversary follow sequential actions with alternate turns

- **Zero-sum Game:**
  - Player wants to minimize the final value, while Adversary wants to maximize the final value
Game Theoretical Perspective

- One can think that online algorithm \( A \) is playing a zero-sum game against \textit{Adversary}
  - Adversary can observe \( A \)'s decision \( o_i \), and then make \( x_{i+1} \) to incur maximal cost
  - Let \( O_i \) be the feasible decision set at the \( i \)-th step
  - Let \( I_i \) be the feasible input set at the \( i \)-th step
  - One can construct a game tree by \( O_i \) and \( I_i \) at each step
  - Let \( I = (x_1, x_2, x_3, \ldots, x_T) \) and \( O = (o_1, o_2, o_3, \ldots, o_T) \)

- One way to design a competitive online algorithm is based on \textit{subgame perfect equilibrium}

\[
\min_{o_T \in O_T} \max_{x_T \in I_T} \ldots \min_{o_1 \in O_1} \max_{x_1 \in I_1} \frac{\text{Cost}(I, O)}{\text{Cost}(I, \text{Opt})}
\]

- Online algorithm \( A \) controls decision \( o_t \) given input \( (x_1, \ldots, x_t) \)
- Adversary controls input \( x_t \) given decisions \( (o_1, \ldots, o_{t-1}) \)
- Subgame perfect equilibrium can be computed by backward induction on the game tree
Buy-or-Rent Problem (aka Ski-Rental Problem)

Example (Buy-or-Rent Problem)

- Say you are new to skiing
- You can either rent skis for $1 per day or buy once for all at cost $B
- On each day, you have to decide whether you will continue to rent for one more day or buy skis
- But you don’t know the number of days for skiing ($D$), except on the $D$-th day (you will not continue skiing)
- Offline optimal strategy: If $D \geq B$ then buy, else rent
- How do you decide without knowing $D$?
Buy-or-Rent Problem

**Online algorithm** $A_{\text{BoR}}(\theta)$

- Chooses a day $\theta$, rents for up to $\theta - 1$ days
- Buy on the $\theta$-th day, if continue skiing

The cost of $A_{\text{BoR}}(\theta)$ is

$$A_{\text{BoR}}(\theta) = \begin{cases} 
D, & \text{if } D < \theta \\
\theta - 1 + B, & \text{if } D \geq \theta 
\end{cases}$$

Let ratio of $A_{\text{BoR}}(\theta)$ vs Opt: $R(D, \theta) = \begin{cases} 
\frac{D}{\min\{D,B\}}, & \text{if } D < \theta \\
\frac{\theta - 1 + B}{\min\{\theta, B\}}, & \text{if } D \geq \theta 
\end{cases}$

The competitive ratio $A_{\text{BoR}}$ is obtained by optimizing $\theta$ and assuming Adversary maximizes $R(D, \theta)$ by choosing $D = \theta$:

$$\alpha(A_{\text{BoR}}) = \min\theta \max D R(D, \theta) = \min\theta \frac{\theta - 1 + B}{\min\{\theta, B\}} = 2 - \frac{1}{B}$$

where the optimal value is $\theta = B$
Buy-or-Rent Problem

Theorem

\( A_{\text{BoR}}(B) \) has the best possible competitive ratio for any deterministic online algorithms

Proof:

- Every deterministic online algorithm can be expressed as a strategy played on a game tree against Adversary
- Suppose the maximum number of days that a deterministic online algorithm stops renting is \( \theta \)
- Every deterministic online algorithm can be expressed by \( A_{\text{BoR}}(\theta) \)
- Then the competitive ratio of every deterministic online algorithm is lower bounded by:

\[
\min_{\theta} \max_{D} R(D, \theta) \geq \min_{\theta} \alpha(A_{\text{BoR}}(\theta))
\]
Variant Buy-or-Rent Problem

Example (Elevator or Stairs)
- You can either use the elevator and wait, or take the stairs
- It takes time $E$ to get to your floor by elevator (once it comes)
- But the waiting time for elevator is unknown
- It takes time $S$ by stairs

Break-even Waiting Strategy
- Wait $S - E$ time, and then take stairs
- If elevator comes in $< S - E$, then it is optimal
- In general, competitive ratio for break-even waiting is $\frac{S - E + S}{S} = 2 - \frac{E}{S}$
- This is similar to rent-or-buy where stairs = buying; think of buying has the cost $S - E$
Definition (Randomized Online Algorithms)

- Consider a randomized online algorithm $\mathcal{A}$ that can make a randomized decision $o_t$ at each $t$-th step.
- Let the sequence of random decisions by randomized online algorithm $\mathcal{A}$ be $O_A$.
- **Oblivious Adversary** knows the online algorithm, but does not know what the random decisions are made by $\mathcal{A}$.
  - We assume that oblivious Adversary prepares all the input in advance without knowing which random decisions will be made by the online algorithm.
  - Oblivious Adversary is sufficient to model the adversarial nature in practice.
- The expected competitive ratio with respect to $\mathcal{A}$’s random decisions, considering oblivious Adversary, is $\alpha(\mathcal{A}) \triangleq \max_{I} \frac{E[\text{Cost}(I,O_A)]}{\text{Cost}(I,\text{Opt})}$.
Consider randomized algorithm \( A_{rBoR}(p) \) for buy-or-rent problem in continuous time

- Choose \( \theta \) according to a probabilistic distribution \( p(\theta) \) such that \( \int_0^B p(\theta) d\theta = 1 \)
- Adversary knows \( p(\theta) \) but does not know \( \theta \) in each instance (not omniscient Adversary)
- Suppose \( D \leq B \). The expected cost of randomized algorithm \( A_{rBoR}(p) \) is

\[
\text{Cost}[A_{rBoR}] = \int_0^D (B + \theta)p(\theta) d\theta + D \int_D^B p(\theta) d\theta
\]

- Note that the offline optimal cost of is \( D \)

**Theorem**

The expected competitive ratio of randomized online algorithm \( A_{rBoR}(p) \) is \( \frac{e}{e-1} \approx 1.58 \)
Proof:

- Let $R$ be the expected competitive ratio of $A_{rBoR}(p)$. Choose $p(\theta)$, such that for $D \leq B$

$$\int_0^D (B + \theta)p(\theta)d\theta + D\int_D^B p(\theta)d\theta = R \cdot D$$

- Differentiating the above equation with respect to $D$, we have

$$(B + D)p(D) + \int_D^B p(\theta)d\theta + D(-p(D)) = R$$

- Differentiating with respect to $D$ again, we have

$$p(D) + (B + D)p'(D) - p(D) - Dp'(D) - p(D) = 0 \Rightarrow \frac{p'(D)}{p(D)} = \frac{1}{B} \Rightarrow p(\theta) = K e^{\frac{\theta}{B}}$$

- We obtain $K = \frac{1}{B(e-1)}$ (since $\int_0^B p(\theta)d\theta = 1$) and the expected competitive ratio is $\frac{e}{e-1}$
How to reduce energy consumption?
- Shutting down energy consumption, as soon as it is not needed

Simple idea but not so simple decision
- Instantaneous shut-down is not always optimal
- Performance latency when re-starting-up if needed
- Additional energy consumption when re-starting-up

How do we decide shut-down/starting-up without sacrificing performance?
- Offline solution is only possible future knowledge of energy demands
- Online algorithm is needed without knowing future energy demands

Power management problem:
- Power management for computing processors
- Power management for cloud computing/HPCs
- Power management for power generators in power grid
Definition (Multi-state Processor Power Management)

- Processor has \( k + 1 \) power states \( \{s_0, s_1, \ldots, s_k\} \), where \( s_0 \) is fully active state.
- Energy consumption at the \( i \)-th state is \( \alpha_i \), \( \alpha_i > \alpha_j \) if \( i < j \).
- Always active is not efficient, since \( \alpha_0 > \alpha_1 > \ldots > \alpha_k \).
- Transition cost from \( s_i \) to active state \( s_0 \) is \( \beta_i \), \( \beta_i < \beta_j \) if \( i < j \).
- Instantaneous shut-down is not efficient, since \( \beta_0 < \beta_1 < \ldots < \beta_k \).
- The total Idle period is unknown – it may be idle forever, or become busy instantaneously.

<table>
<thead>
<tr>
<th>State</th>
<th>Power (W)</th>
<th>Starting-up Energy (J)</th>
<th>Transition Time to Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep ( s_3 )</td>
<td>0 (( \alpha_3 ))</td>
<td>4.75 (( \beta_3 ))</td>
<td>5 s</td>
</tr>
<tr>
<td>Stand-by ( s_2 )</td>
<td>0.2 (( \alpha_2 ))</td>
<td>1.575 (( \beta_2 ))</td>
<td>1.5 s</td>
</tr>
<tr>
<td>Idle ( s_1 )</td>
<td>0.9 (( \alpha_1 ))</td>
<td>0.56 (( \beta_1 ))</td>
<td>40 ms</td>
</tr>
<tr>
<td>Active ( s_0 )</td>
<td>1.9 (( \alpha_0 ))</td>
<td>0 (( \beta_0 ))</td>
<td>0</td>
</tr>
</tbody>
</table>
Online Algorithm for Multi-state Processor Power Management

- Assume that it is active without workload at the beginning $t = 1$
- The total idle period without workload is from $t = 1$ to $t = T$ (workload at $t = T + 1$)
- Then the offline optimal state with known $T$ is $s_i^*$, where

\[ i^* = \arg \min_{i=0,\ldots,k} \left( \alpha_i T + \beta_i \right) \]

- How about online algorithm with unknown $T$?

Online Algorithm for Multi-state Processor $A_{\text{fev}}$ (Follow-the-envelope)

- Suppose there is no workload from $t = 1$ to $t = \tau$
- At time $\tau$, pick state $s_{\hat{i}}$, such that

\[ \hat{i} = \arg \min_{i=0,\ldots,k} \left( \alpha_i \tau + \beta_i \right) \]
Define envelope $f(t) \triangleq \min_{i=0,\ldots,k} \left( \alpha_i t + \beta_i \right)$

Let $t_i$ be the solution that $\alpha_i t_i + \beta_i = \alpha_{i-1} t_{i-1} + \beta_{i-1}$
Theorem

The competitive ratio for online algorithm $A_{\text{fev}}$ is 2

Proof:

- Suppose the total idle period is $t_{i^*} + \gamma$, where $t_{i^*} + \gamma < t_{i^*+1}$. The offline optimal cost is
  
  $$a_{i^*}(t_{i^*} + \gamma) + b_{i^*}$$

- But the cost for $A_{\text{fev}}$ is
  
  $$\sum_{l=0}^{i^*-1} a_l(t_{l+1} - t_l) + a_{i^*}\gamma + b_{i^*}$$

- The competitive ratio is
  
  $$\frac{\sum_{l=0}^{i^*-1} a_l(t_{l+1} - t_l) + a_{i^*}\gamma + b_{i^*}}{a_{i^*}(t_{i^*} + \gamma) + b_{i^*}} \leq \frac{\sum_{l=0}^{i^*-1} a_l(t_{l+1} - t_l) + b_{i^*}}{a_{i^*}t_{i^*} + b_{i^*}}$$
Online Algorithm for Multi-state Processor Power Management

Proof (Cont.):

• The competitive ratio is upper bounded by

\[
\sum_{l=0}^{i^*-1} \frac{\alpha_l(t_{l+1} - t_l)}{\alpha_i t_i^* + \beta_i^*} + \beta_i^* = 1 + \sum_{l=0}^{i^*-1} \frac{\alpha_l(t_{l+1} - t_l) - \alpha_i t_i^*}{\alpha_i t_i^* + \beta_i^*}
\]

• By the definition of \( t_l \),

\[
\sum_{l=0}^{i^*-1} \alpha_l(t_{l+1} - t_l) = \sum_{l=1}^{i^*} (\alpha_l - \alpha_{l-1}) t_l = \sum_{l=1}^{i^*} (\beta_l - \beta_{l-1}) = \beta_i^* - \beta_0
\]

• The competitive ratio is upper bounded by

\[
1 + \frac{\beta_i^* - \beta_0 - \alpha_i t_i^*}{\alpha_i t_i^* + \beta_i^*} \leq 2
\]
Randomized Online Algorithm for Multi-state Processor $A_{rfev}$

- Suppose processor is in state $s_i$ starting at time $\tau_0$
- Randomly picks threshold $\tau$ from probability distribution $p(\tau)$
- Continue to stay at $s_i$ until time $\tau_0 + \tau$
- After time $\tau_0 + \tau$, transit to $s_{i+1}$ if idling continues

- Decide the optimal $p(\theta)$ and competitive ratio
- But require $\int_{0}^{\tau_{i+1} - \tau_0} p(\tau) d\tau = 1$
- Basic idea: mapping to buy-or-rent problem
  - Focus on two states $s_i$ and $s_{i+1}$
  - Transition from $s_i$ to $s_{i+1}$ needs to pay $\beta_{i+1} - \beta_i$ (i.e. buying)
  - Staying at $s_i$ for a duration $t$ needs to pay $\alpha_i t$ (i.e. renting)
Randomized Online Algorithm for Multi-state Processor

**Theorem**

The expected competitive ratio for randomized online algorithm $A_{rfev}$ is $\frac{e}{e - 1}$

**Proof:**

- Suppose idling stops at $\tau_0 + d$, where $\tau_0 + d < t_{i+1}$
- $\text{Cost}(A_{rfev}) = \sum_{i=1}^{k} \text{SubCost}_i(A_{rfev})$, where $\text{SubCost}_i(A_{rfev})$ is the expected sub-cost during the processor in state $s_i$
- $\text{SubCost}_i(A_{rfev}) = \int_{0}^{d} (\alpha_i \tau + \beta_{i+1} - \beta_i) p(\tau) d\tau + (\alpha_i d) \int_{d}^{t_{i+1} - \tau_0} p(\tau) d\tau$
- Let the offline optimal sub-cost for each interval $[t_j, t_{j+1})$ be $\text{Cost}_i(Opt)$
- The expected competitive ratio is

$$\frac{\text{Cost}(A_{rfev})}{\text{Cost}(Opt)} = \sum_{i=1}^{k} \frac{\text{SubCost}_i(A_{rfev})}{\text{Cost}(Opt)} \leq \max_{i=1,\ldots,k} \frac{\text{SubCost}_i(A_{rfev})}{\text{Cost}_i(Opt)} \leq \frac{e}{e - 1}$$

- $p(\tau)$ follows the same randomized online algorithm for buy-or-rent problem
Can Online Algorithms be applied to Algorithmic Trading?
When should you sell/buy in stock markets without knowing the future of market prices?

Trading in financial market is an online decision problem
  - Goal: Want to sell at the highest price, or buy at the lowest price

Decisions need to be made without complete future information

How do we know if the current price is the highest/lowest?

Probabilistic analysis
  - Need to model risk and uncertainty of future price fluctuations

Competitive analysis
  - Risk-less, guaranteeing the worst case performance
1-Max Search

**Definition (1-Max Search)**

- Give a sequence of prices \( (p_1, p_2, \ldots, p_T) \) over time
- Goal: Decide whether to sell at price \( p_t \) at current time \( t \),
or wait for the next time at an unknown price

**Unknown:**
- Assume no knowledge of future price \( p_t \)

**Known:**
- Price range \( m \leq p_t \leq M \) for all \( t = 1, \ldots, T \)
- Deadline: If not sold before \( T \), then will be forced to sell at \( p_T \)

**Offline optimal solution:**
- Pick the time with the highest price: \( t_{\text{max}} = \arg \max \{ p_t : t = 1, \ldots, T \} \)

**Online algorithm:** How?
1-Max Search

Threshold Selling Algorithm $A_{1\text{max}}(\hat{p})$

- Repeat
  - If the current price $p_t \geq \hat{p}$, then sell and exit
  - Else wait for the next price $p_{t+1}$
- Until $t = T$ then sell at $p_T$

Lemma

Setting $\hat{p} = \sqrt{Mm}$ in $A_{1\text{max}}(\hat{p})$ achieves the best competitive ratio $= \sqrt{\frac{M}{m}}$

Proof:

- Knowing $\hat{p}$, Adversary has two options:
  1. Case 1: Make $A_{1\text{max}}$ sell at $\hat{p}$
  2. Case 2: Make $A_{1\text{max}}$ sell at $p_T$
1-Max Search

Proof:

- Knowing $\hat{p}$, Adversary has two options:
  1. Case 1: Make $A_{1\text{max}}$ sell at $\hat{p}$
  2. Case 2: Make $A_{1\text{max}}$ sell at $p_T$

- The competitive ratio is
  1. Case 1: $\frac{M}{\hat{p}}$ by price sequence $(\hat{p}, M, ...)$
  2. Case 2: $\frac{\hat{p}}{m}$ by price sequence $(\hat{p} - \epsilon, ..., m)$

- We optimize $\hat{p}$ by minimizing the following value:

$$\min \frac{M}{\hat{p}}, \max \frac{\hat{p}}{m}$$

- Note that $\frac{M}{\hat{p}}$ is decreasing in $\hat{p}$; $\frac{\hat{p}}{m}$ is increasing in $\hat{p}$

- The optimal setting: $\frac{M}{\hat{p}} = \frac{\hat{p}}{m} \Rightarrow \hat{p} = \sqrt{Mm}$
1-Min Search

- Need to buy at as low price as possible

Threshold Buying Algorithm $\mathcal{A}_{1\min}(\hat{p})$

- Repeat
  - If the current price $p_t \leq \hat{p}$, then buy and exit
  - Else wait for the next price $p_{t+1}$
- Until $t = T$ then buy at $p_T$

Lemma

Setting $\hat{p} = \sqrt{Mm}$ in $\mathcal{A}_{1\min}(\hat{p})$ achieves the best competitive ratio $= \sqrt{\frac{M}{m}}$
Definition ($k$-Max Search)

- **Goal:** Need to sell $k$ items, only one item sold at each time
- **Known:**
  - Price range $m \leq p_t \leq M$
  - Deadline: If $i$ items unsold before $T - i + 1$, then will be forced to sell all at $(p_{T-i+1}, \ldots, p_T)$
- **Offline optimal:**
  - Pick the $k$ highest prices
- **Online:** How?
  - We extend from $A_{1\max}$ to $A_{k\max}$ with $k$ threshold selling prices
Let $\hat{p}_i$ be the threshold selling price of the $i$-th item

- $\hat{p}_1 \leq \hat{p}_2 \leq \ldots \leq \hat{p}_k$ (i.e., selling the first item should be easier than the second)

**Theorem**

If we set $\hat{p}_i = m \left(1 + (\hat{\tau} - 1)(1 + \frac{\hat{\tau}}{k})^{i-1}\right)$, then $A_{k_{\text{max}}}$ achieves the best competitive ratio $\hat{\tau}$, where $\hat{\tau}$ is the unique solution to $\frac{M}{m} - 1 - \hat{\tau} - 1 = (1 + \frac{\hat{\tau}}{k})^k$. Note that $\hat{\tau} \to O\left(\ln\left(\frac{M}{m}\right)\right)$, when $k \to \infty$

**Proof (Sketch):**

- Adversary has $k + 1$ options:
  - Case 1: Make $A_{k_{\text{max}}}$ sell at $\hat{p}_1$ and $(p_{T-k+2}, \ldots, p_T)$
  - Case 2: Make $A_{k_{\text{max}}}$ sell at $(\hat{p}_1, \hat{p}_2)$ and $(p_{T-k+3}, \ldots, p_T)$
  - Case $k$: Make $A_{k_{\text{max}}}$ sell at $(\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_k)$
  - Case $k + 1$: Make $A_{k_{\text{max}}}$ sell at $(p_{T-k+1}, \ldots, p_T)$

- We analyze the competitive ratio in each case
Proof (Cont.):

- **Competitive ratios**
  - Case 1: \( \frac{k \hat{p}_2}{\hat{p}_1 + (k-1)m} \) by price sequence \((\hat{p}_1, \hat{p}_2 - \epsilon, \ldots, \hat{p}_2 - \epsilon, m, \ldots, m)\)
  - Case 2: \( \frac{k \hat{p}_3}{\hat{p}_1 + \hat{p}_2 + (k-2)m} \) by price sequence \((\hat{p}_1, \hat{p}_2, \hat{p}_3 - \epsilon, \ldots, \hat{p}_3 - \epsilon, m, \ldots, m)\)
  - Case \( k \): \( \frac{kM}{\sum_{i=1}^{k} \hat{p}_i} \) by price sequence \((\hat{p}_1, \ldots, \hat{p}_k, M, \ldots, M)\)
  - Case \( k + 1 \): \( \frac{k \hat{p}_1}{km} \) by price sequence \((\hat{p}_1 - \epsilon, \ldots, \hat{p}_1 - \epsilon, m, \ldots, m)\)

- Equating the competitive ratios for all cases to solve \( \hat{r} \) and \((\hat{p}_i)_{i=1}^{k}\)
**Definition (k-Min Search)**

- **Goal:** Need to buy $k$ items, only one item bought at each time
- We extend from $A_{1\text{min}}$ to $A_{k\text{min}}$ with $k$ threshold buying prices
- Let $\hat{p}_i$ be the threshold buying price of the $i$-th item
- $\hat{p}_1 \geq \hat{p}_2 \geq \ldots \geq \hat{p}_k$

**Theorem**

If we set $\hat{p}_i = M \left(1 - \left(1 - \frac{1}{\hat{s}}\right)\left(1 + \frac{1}{k\hat{s}}\right)^{i-1}\right)$, then $A_{k\text{min}}$ achieves the best competitive ratio $\hat{s}$, where $\hat{s}$ is the unique solution to $\frac{1 - m}{1 - \frac{1}{\hat{s}}} = (1 + \frac{1}{k\hat{s}})^{k}$. Note that $\hat{s} \rightarrow O\left(\sqrt{\frac{M}{m}}\right)$, when $k \rightarrow \infty$
Definition (One-way Trading)

- You want to convert, $D_0$, an amount of one currency to another currency over time (e.g. from USD to EUR)
- Give a sequence of exchange rates $(p_1, p_2, ..., p_T)$ over time
- Goal: Decide how much amount, $D_i$, to be converted at the $i$-th step
- Unknown:
  - Assume no knowledge of future rate $p_t$
- Known:
  - Rate range $0 \leq m \leq p_t \leq M \leq 1$
  - Deadline: If not all $D_T$ is converted by $T$, then will be forced to convert at rate $p_T$
Lemma (One-way Trading = Randomized 1-Max Search)

For any deterministic online algorithm for one-way trading, there exists a randomized online algorithm for 1-max search that achieves the same expected competitive ratio.

Proof:

- Consider that a deterministic online algorithm for one-way trading $A_1$ that converts a fraction $p$ of currency when given a rate sequence.
- We construct a randomized online algorithm $A_2$ for 1-max search that sells with probability $p$ before the deadline when given the same past rate sequence.
- The expected selling price of $A_2$ should be the same as $A_1$ for any given rate sequence.
One-way Trading

Lemma (Randomized One-way Trading)

For any randomized online algorithm for 1-max search, there exists a deterministic online algorithm for one-way trading that achieves the same expected competitive ratio.

Proof:

- Given a set of deterministic online algorithms \( \mathcal{A} \), a randomized algorithm assigns a probability \( p(\mathcal{A}) \) to each \( \mathcal{A} \in \mathcal{A} \).
- For a given rate sequence \( (p_1, p_2, ..., p_T) \), let \( s(i, \mathcal{A}) \) be the value of \( \mathcal{A} \) at the \( i \)-th step.
  - \( s(i, \mathcal{A}) = p_i \), if \( \mathcal{A} \) sells at the \( i \)-th step; otherwise, \( s(i, \mathcal{A}) = 0 \).
- Consider a randomized algorithm \( \mathcal{A}' \) that picks a deterministic algorithm in \( \mathcal{A} \) at random:

\[
\mathbb{E}[\text{Value}(\mathcal{A}')] = \sum_{\mathcal{A} \in \mathcal{A}} p(\mathcal{A}) \sum_{i} s(i, \mathcal{A}) = \sum_{i} \sum_{\mathcal{A} \in \mathcal{A}} p(\mathcal{A}) s(i, \mathcal{A}) = \text{Value}(\mathcal{A}'')
\]

where \( \mathcal{A}'' \) is another deterministic algorithm that converts a fraction of \( \sum_{\mathcal{A} \in \mathcal{A}} p(\mathcal{A})s(i, \mathcal{A}) \) at the \( i \)-th step.
By the two lemmas, randomization in one-way trading cannot improve competitive ratio, because randomization does not help any (already) randomized 1-max search algorithm.

A simple deterministic algorithm for one-way trading is use $k$-max search with a large $k$.

Let $D_0$ be the initial amount of USD, and let $D_i$ and $U_i$ to be the amount of remaining USD and converted EUR at the $i$-th step.

**Online Algorithm for One-way Trading $A_{1\text{way}}$**

- At the $i$-th step, only convert when $p_i$ is the highest seen before.
- Let $s_i \triangleq D_{i-1} - D_i$ be the amount of conversion from USD to EUR at the $i$-th step.
- Convert $s_i = \frac{p_i - c^*(U_{i-1} + mD_{i-1})}{c^*(p_i - m)}$ from USD to EUR, where $c^*$ the solution to

$$c^* = T \left(1 - \left(\frac{m(c^* - 1)}{M - m}\right)^{\frac{1}{T}}\right)$$
Theorem

The competitive ratio for online algorithm $A_{1\text{way}}$ is $c^*$. Note that $c^* \rightarrow O\left(\ln\left(\frac{M}{m}\right)\right)$, when $T \rightarrow \infty$

Proof (Sketch):

- Suppose $A_{1\text{way}}$ converts at $p_i$ (when $p_i$ is the highest so far)
- Then Adversary makes $A_{1\text{way}}$ to convert $mD_i$ at the end
- But the offline optimal can be $p_iD_0$, because $p_i$ is the highest
- Assume $D_0 = 1$ and note that $s_i \triangleq D_{i-1} - D_i$
- Then the competitive ratio is $c^* = \frac{p_iD_0}{U_i+mD_i} = \frac{p_i}{(U_{i-1}+s_ip_i)+m(D_{i-1}-s_i)}$

$$c^* = \frac{p_i}{(U_{i-1} + s_ip_i) + m(D_{i-1} - s_i)} \Rightarrow s_i = \frac{p_i - c^*(U_{i-1} + mD_{i-1})}{c^*(p_i - m)}$$
One-way Trading

Proof (Cont.):

- Suppose we convert from USD to EUR at each $i$-th step by $s_i = \frac{p_i - c^* (U_{i-1} + mD_{i-1})}{c^*(p_i - m)}$
- Let us consider increasing rates $p_1 \leq p_2 \leq \ldots \leq p_{i-1} \leq p_i$ (otherwise, just ignore decreasing rates, as they will not trigger conversions), we have

\[
U_1 + mD_1 = \frac{p_1}{c^*}, \ldots, U_{i-1} + mD_{i-1} = \frac{p_{i-1}}{c^*}, U_i + mD_i = \frac{p_i}{c^*}
\]

- Substituting $U_{i-1} + mD_{i-1} = \frac{p_{i-1}}{c^*}$ into $s_i = \frac{p_i - c^*(U_{i-1} + mD_{i-1})}{c^*(p_i - m)}$, we obtain $s_i = \frac{p_i - p_{i-1}}{c^*(p_i - m)}$

- Since $\sum_{i=1}^{T} s_i = D_0 = 1$, we obtain $1 = \frac{p_1 - mc^*}{c^*(p_1 - m)} + \sum_{i=2}^{T} \frac{p_i - p_{i-1}}{c^*(p_i - m)}$, where $s_1 = \frac{p_1 - mc^*}{c^*(p_1 - m)}$

- Finally, Adversary controls the rate sequence $(p_i)_{i=1}^{T}$ to maximize the competitive ratio, which will be the solution to $c^* = T \left(1 - \left(\frac{m(c^* - 1)}{M - m}\right)^{\frac{1}{T}}\right)$

▶ See the full proof in *Optimal Search and One-Way Trading Online Algorithms* (El-Yaniv, Fiat, Karp, Turpin)
Reference Materials

- Online Computation and Competitive Analysis (Borodin, El-Yaniv), Cambridge Uni. Press
  - Chapters 2, 8, 14

Recommended Materials

- Online Strategies for Dynamic Power Management in Systems with Multiple Power-Saving States (Irani, Shukla, Gupta), ACM Trans. on Embedded Computer Sys., 2003
- Optimal Search and One-Way Trading Online Algorithms (El-Yaniv, Fiat, Karp, Turpin), Algorithmica, 2001