Foundations for Algorithms with Predictions: Online Algorithms

CS 263

Michael T. Goodrich

University of California, Irvine

Online Algorithms

- The study of algorithms with predictions is a fast-growing field, with considerable amount of recent work.
- Many such algorithms are designed as **online algorithms**; hence, it is useful to understand the framework of online algorithms, so as to understand the goals of algorithms with predictions.

Measuring Quality

- How do we know if an online algorithm is good?
- For optimization problems (find the min), let ALG be what your algorithm finds, let OPT be the true best answer knowing the input that's coming.
- We achieve a **competitive ratio** of α if:

$$ALG \leq \alpha \cdot OPT$$

  (for find the max, switch the inequality direction)
- The goal is for our algorithm to "learn" or "adapt" as the input is coming.

Ski Rental

- You and a group of friends have decided to take up skiing.
- Every time you go skiing, you can either:
- Rent skis for the day (say it costs **$1**, for simplicity of calculation)
- Buy skis, cost **$B** and can use them forever (don't need to rent again).
- Every time you go skiing, you have the option of rent or buy.
- It's unclear how long you and your friends will keep up this hobby. How many times should you rent before you buy?

What Information Do We Get?

- Every time your friends call and ask “wanna go skiing this weekend?” you decide whether to run out and buy your own skis or to rent another time.
- You aren’t going to get a good sense of how often this will happen. At any time your friends might give up the hobby. You can’t estimate the chances of there being another trip.
- One goal is to decide on a plan in advance, how many times should you rent before deciding to buy. Your goal is to minimize the competitive ratio, i.e., minimize the factor of money you overpay.

Deterministic Algorithm

- Suppose you decide you'll rent for the first k days, and buy if you are invited for trip $k+1$.
- And let D be the true number of days you're invited.
  - What is OPT?
  - What is ALG?
- What's the worst case ratio? i.e. for any particular k what's the worst-case ratio?
- What value of k should you choose?

Deterministic Algorithm

- Suppose you decide you'll rent for the first k days, and buy if you are invited for trip $k+1$.
- And let D be the true number of days you're invited.

$$OPT = \begin{cases} D & \text{if } D < B \\ B & \text{otherwise} \end{cases}$$

If $D < B$ then only rent, if $D \geq B$ buy right away.
OPT has foresight (it's psychic) it knows $D \geq B$ and it's better to buy right away.

- What's the worst case ratio? i.e. for any particular k what's the worst-case ratio?
- What value of k should you choose?

Deterministic Algorithm

- Suppose you decide you'll rent for the first k days, and buy if you are invited for trip $k+1$.
- And let D be the true number of days you're invited.

If $D \leq k$ we rent all D days.
Otherwise we rent the first k days then buy

$$ALG = \begin{cases} D & \text{if } D \leq k \\ B + k & \text{otherwise} \end{cases}$$

- What's the worst case ratio? i.e. for any particular k what's the worst-case ratio?
- What value of k should you choose?

Finding the competitive ratio

- $OPT = \begin{cases} D & \text{if } D < B \\ B & \text{otherwise} \end{cases}$ $ALG = \begin{cases} D & \text{if } D \leq k \\ B + k & \text{otherwise} \end{cases}$
- We get to choose k
- If we choose $k = B - 1$, what is the worst D?

- If we choose $k = B$ what is the worst D?

- If we choose $k \leq B - 2$ what is the worst D?

Finding the competitive ratio

- $OPT = \begin{cases} D & \text{if } D < B \\ B & \text{otherwise} \end{cases}$ $ALG = \begin{cases} D & \text{if } D \leq k \\ B + k & \text{otherwise} \end{cases}$
- We get to choose k
- If we choose $k = B - 1$, what is the worst D?
- Worst D is B, ratio is $\frac{B-1+B}{B} = 2 - \frac{1}{B}$
- If we choose $k \geq B$ what is the worst D?
- Worst D is $\mathrm{k} + 1$, ratio is $\frac{k+B}{B} \geq 2$ as k gets bigger, we add a small amount to the ratio each time.
- If we choose $k \leq B - 2$ what is the worst D?
- Worst $D = k + 1$ ratio is $\frac{k+B}{k+1} = 1 + \frac{B-1}{k+1} \geq 2$

A slightly better algorithm

- So our deterministic algorithm can achieve a competitive ratio of just less than 2 in the worst case.
- We might be worrying a bit too much about the absolute worst case. Let's try to "spread out" the badness a little.
- For example, let's say that B is 10.
- The last algorithm has a competitive ratio of 1.9.

- Here's a slightly better idea: flip a coin: if it's heads, choose $k = 7$, otherwise choose $k = 9$.

Analysis

- How much do we pay for D days of skiing?
- If $D \leq 7$,
- We always pay D; exactly what OPT pays
- If $8 \leq D \leq 9$
- Half the time, pay $7 + 10$, half the time pay D
- On average pay $\frac{17}{2} + \frac{D}{2}$, OPT pays D, ratio $\frac{1}{2} + \frac{9}{D}$ worst D is 9, gives $\frac{3}{2}$
- If $D \geq 10$
- Half the time, pay $7 + 10$, half the time pay $9 + 10$; OPT pays 10. ratio is $\frac{8+10}{10} = 1.8$ worst ratio is 1.8. That's better than 1.9 from before!

How did Randomization Help?

- Randomness makes it harder to find a "worst-case"
- When $k = 8$ or $k = 10$ individually, the worst D is fixed. When we're choosing randomly, no value of D is always as bad as could be! There's at least a 50% chance the D that happens isn't the worst one!

- So we "spread out" the badness and make our worst-case result a little better.

The Best Strategy

- You should choose the day k to buy with probability:

- $$p_k = \begin{cases} \left(\frac{B-1}{B}\right)^{B-k} \cdot \frac{1}{B\left(1-\left(1-\left(\frac{1}{B}\right)\right)^{B}\right)} & \text{if } k \leq B \\ 0 & \text{otherwise} \end{cases}$$

- Don't get scared – the second term is just normalizing (the probabilities have to sum to 1, ignore that part). As k increases, the probability we buy increases. Once we hit B, we're guaranteed to buy.

Why?

- A lot of math…
- What's the ratio? As B gets very big, it approaches $\frac{e}{e-1} \approx 1.58$.
- Quite a bit better than $2 - \frac{1}{B}$.
- Suppose next we have a machine-learning prediction for when to buy…

Skiing with Predictions

- Instead of using the pure e/(e-1) distribution, we create a hybrid strategy. We pick a confidence parameter λ in the interval [0, 1]. In our case, if the prediction is, say, 80% reliable, we might set $\lambda = 0.8$.
- The algorithm follows two logical paths:
- **With probability λ:** Follow the "Advice-Trusting" strategy (Buy slightly before the predicted day h).
- **With probability 1-λ:** Follow the "Robust" strategy (The randomized 1.58 distribution).

Competitive Ratio Analysis

- Let us analyze the algorithm with predictions in the consistency-robustness framework.
- Consistency: what is the performance if the prediction is right?
- Robustness: what is the performance if the prediction is wrong?

A. Consistency (When the 80% is right)

If the prediction h is accurate, the competitive ratio C is:

$$C \approx 1 + \frac{1-\lambda}{\lambda}$$

- If we trust the prediction fully ($\lambda = 1$), our cost is $1.0 \times Opt$ (Perfect).
- With $\lambda = 0.8$, our consistency is $1 + 0.2/0.8 = \mathbf{1.25}$.
- **Result:** You perform significantly better than the 1.58 randomized limit when the prediction is right.

B. Robustness (When the 20% is wrong)

If the prediction is a "malicious" lie (e.g., it says ski for 100 days but the season ends on day 1), the algorithm must still be safe. The robustness R is capped at:

$$R \approx \frac{1}{1-\lambda} \cdot \left(\frac{e}{e-1} \right)$$

- With $\lambda = 0.8$, the robustness is roughly $5 \times 1.58 = \mathbf{7.9}$.
- **The Trade-off:** By trying to be super efficient 80% of the time, you risk a much higher cost in the 20% worst-case scenario.

Online List Maintenance: Move to Front

slides adapted from slides by Charles Leiserson and Eric Demaine

Self-organizing lists

- List L of n elements
- The operation ACCESS(x) costs

$$rank_L(x) = \text{distance of } x \text{ from the head of } L.$$

- L can be reordered by swapping adjacent elements at a cost of 1
- Goal: access to a sequence of n items with minimal cost

List access algorithms

- Off-line Algorithm: if the sequence of access S is known in advance, one can design an optimal algorithm to rearrange the list based on how often items are accessed
- On-line Algorithm: if the sequence is not known in advance, one can design an algorithm based on some heuristics.

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

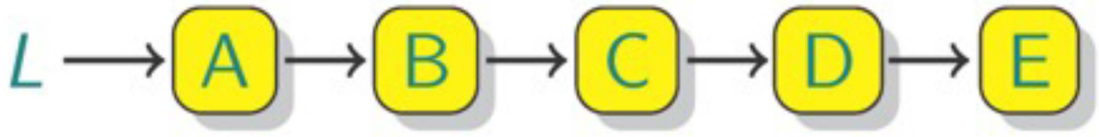
$$cost = 2 \cdot rank_L(x)$$

Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Access item D:

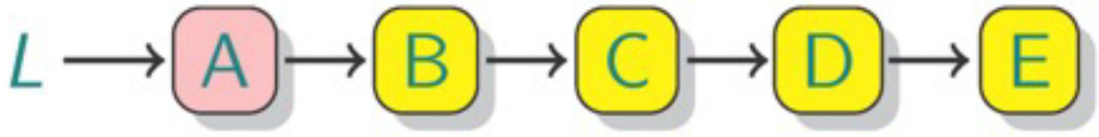


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Access item D:

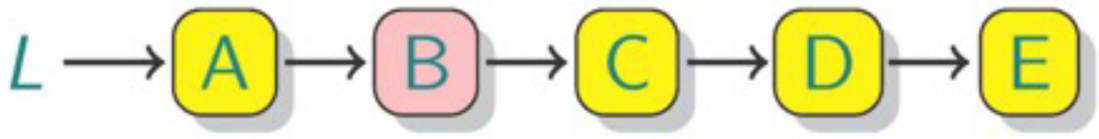


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Access item D:

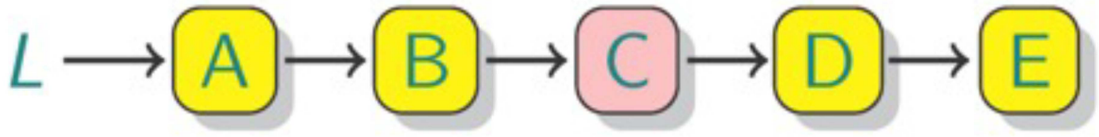


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Access item D:

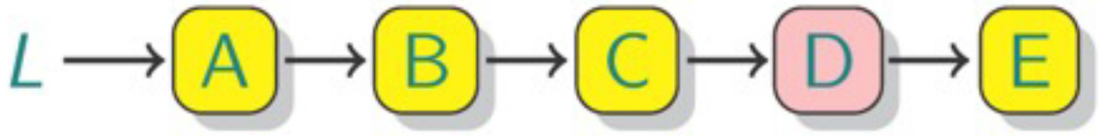


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Access item D:

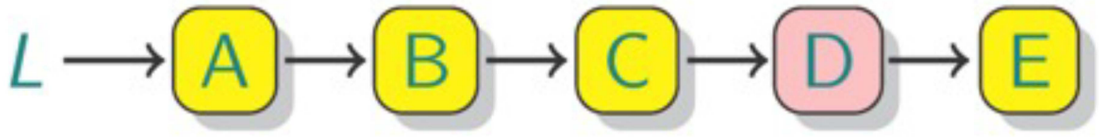


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

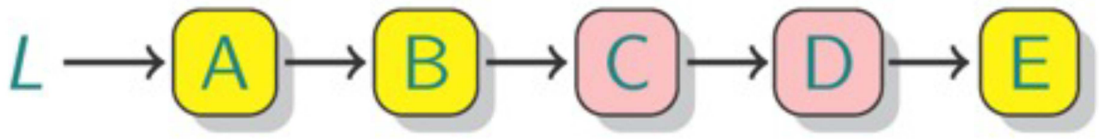


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

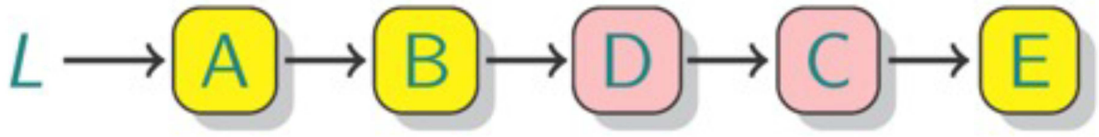


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

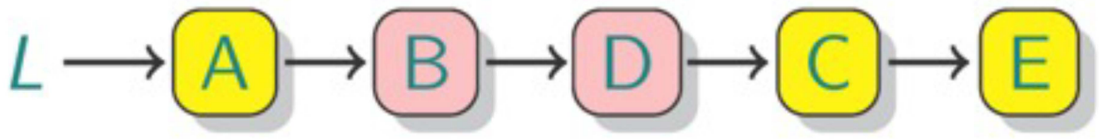


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

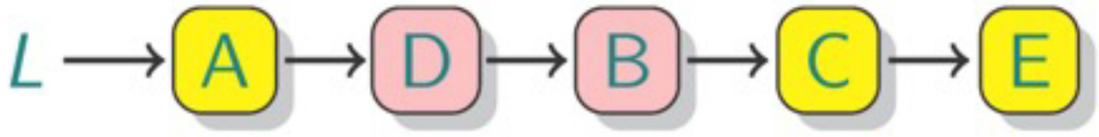


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

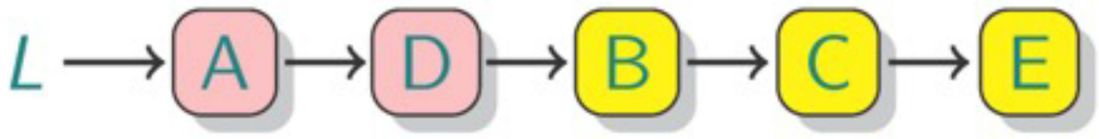


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

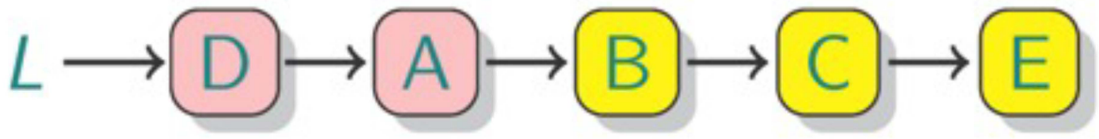


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

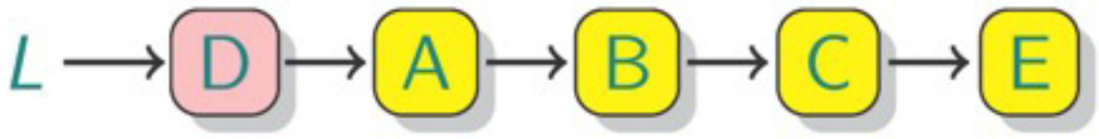


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:

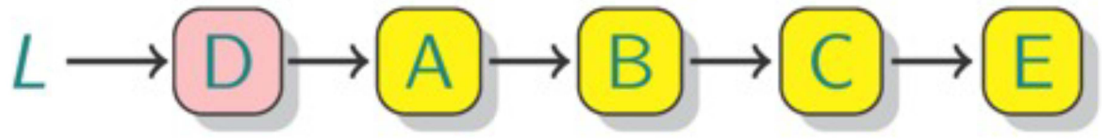


Move-to-front algorithm

- Algorithm: After accessing x, move x to the head of L using swaps.

$$cost = 2 \cdot rank_L(x)$$

- Move D to front:



- Heuristic: if x is accessed at time t, it is likely to be accessed again soon after time t.
- Cost: MTF always performs within a factor of 4 of the optimal algorithm.

Amortized analysis of MTF

Theorem: $C_{MTF}(S) \leq 4C_{OPT}(S)$
Proof: Let L_i be MTF's list after the ith access, and let L_i^* be OPT's list after the ith access. Let

$$\begin{aligned} c_i &= \text{ MTF's cost for the } i\text{th operation} \\ &= 2 \cdot rank_{L_{i-1}}(x) \text{ if it accesses } x; \\ c_i^* &= \text{ OPT's cost for the } i\text{th operation} \\ &= rank_{L_{i-1}^*}(x) + t_i, \end{aligned}$$

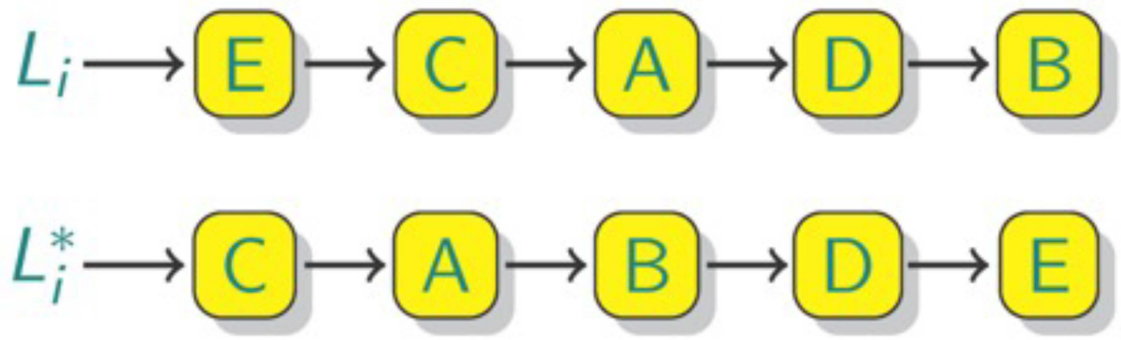
where t_i is the number of swaps that OPT performs.

Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \#\text{ inversions}\end{aligned}$$

Example:

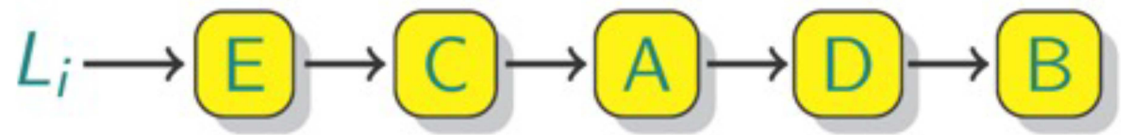


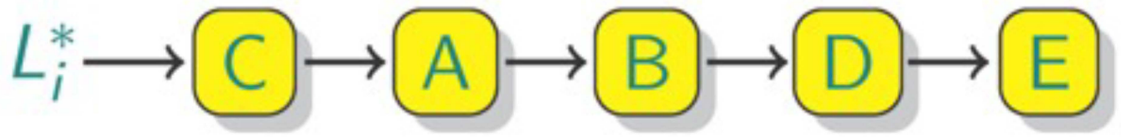
Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \#\text{ inversions}\end{aligned}$$

Example:





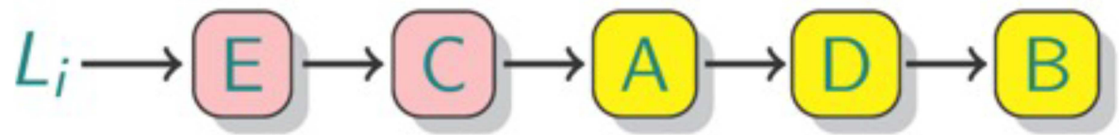
$\Phi(L_i) = 2 \cdot |\{\cdots\}|$

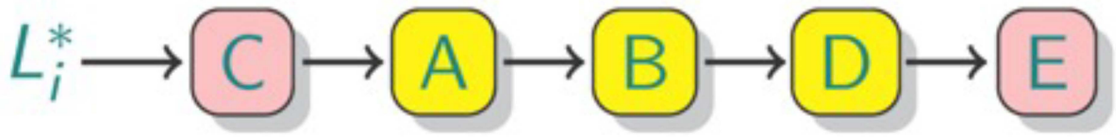
Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \#\text{ inversions}\end{aligned}$$

Example:





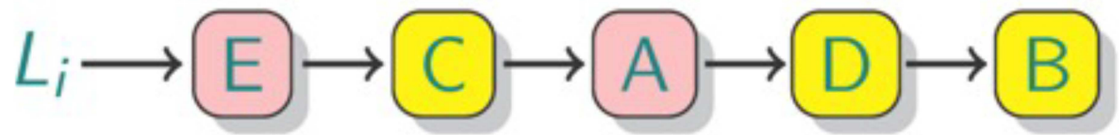
$\Phi(L_i) = 2 \cdot |\{(E, C), \cdots\}|$

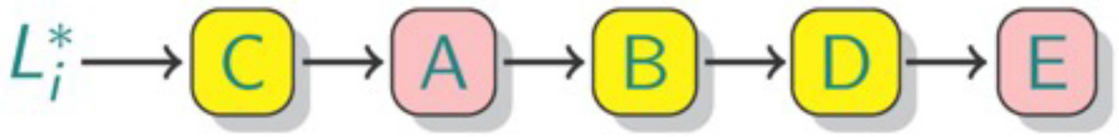
Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \#\text{ inversions}\end{aligned}$$

Example:





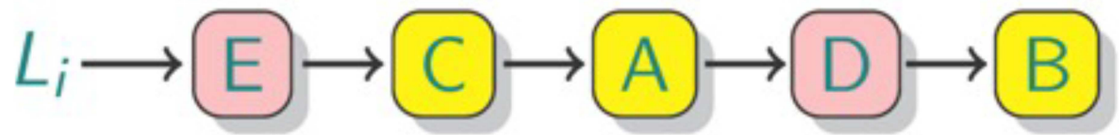
$$\Phi(L_i) = 2 \cdot |\{(E, C), (E, A), \cdots\}|$$

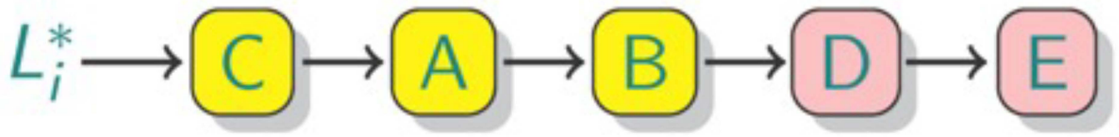
Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \#\text{ inversions}\end{aligned}$$

Example:





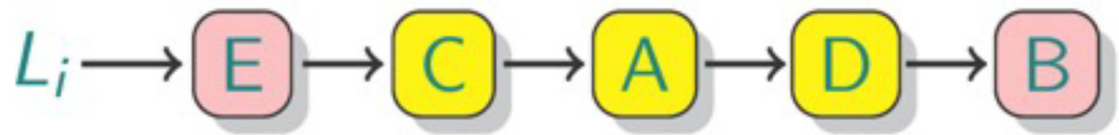
$$\Phi(L_i) = 2 \cdot |\{(E, C), (E, A), (E, D), \cdots\}|$$

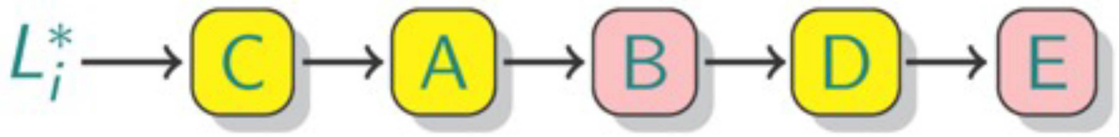
Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \#\text{ inversions}\end{aligned}$$

Example:





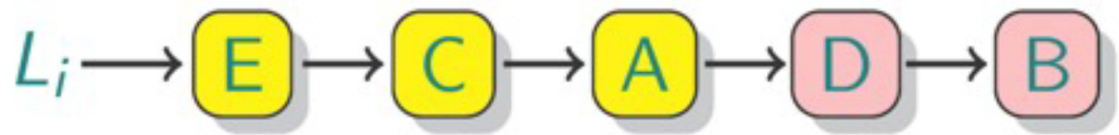
$$\Phi(L_i) = 2 \cdot |\{(E, C), (E, A), (E, D), (E, B), \cdots\}|$$

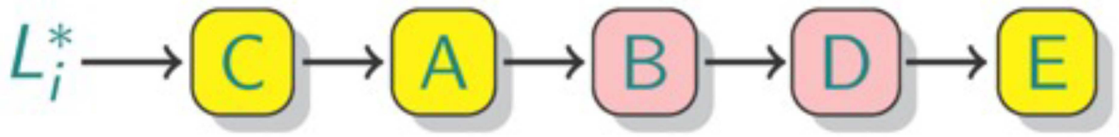
Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \#\text{ inversions}\end{aligned}$$

Example:





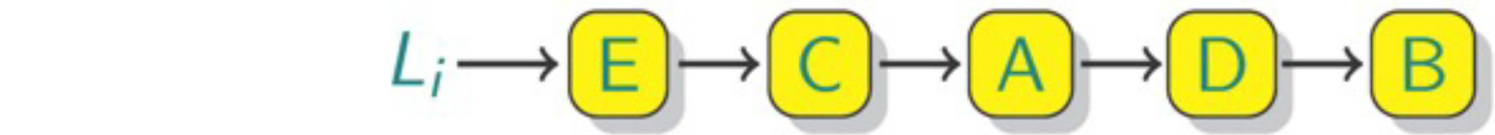
$$\Phi(L_i) = 2 \cdot |\{(E, C), (E, A), (E, D), (E, B), (D, B)\}|$$

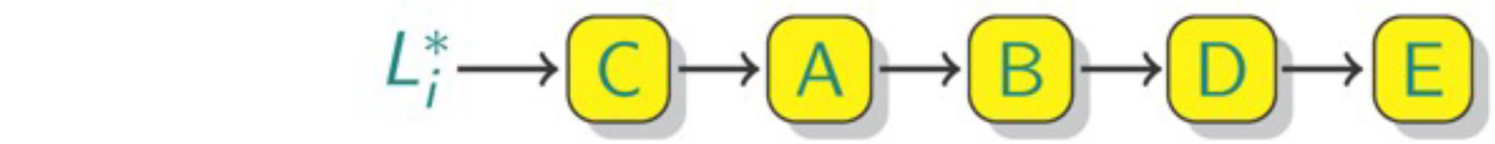
Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \#\text{ inversions}\end{aligned}$$

Example:





$$\Phi(L_i) = 2 \cdot |\{(E, C), (E, A), (E, D), (E, B), (D, B)\}| = 10$$

Potential function

Define the potential function $\Phi : L_i \to \mathcal{R}$ by

$$\begin{aligned}\Phi(L_i) =& 2 \cdot |\{(x, y) : x \prec_{L_i} y \text{ and } y \prec_{L_i^*} x\}| \\ =& 2 \cdot \# \text{ inversions}\end{aligned}$$

Note that:

- $\Phi(L_i) \geq 0$ for $i = 0, 1, ...$
- $\Phi(L_0) = 0$ if MTF and OPT start with the same list.

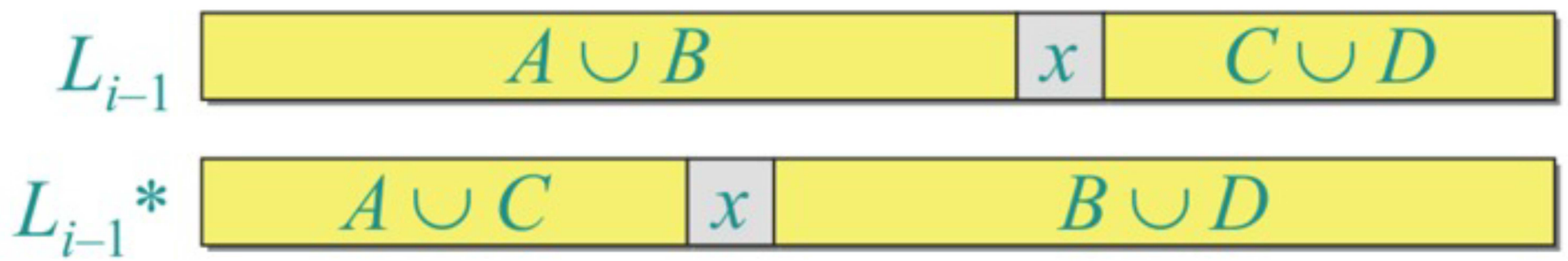
How much does Φ change from one swap?

- a swap creates/destroys 1 inversion
- $\Delta\Phi = \pm 2$

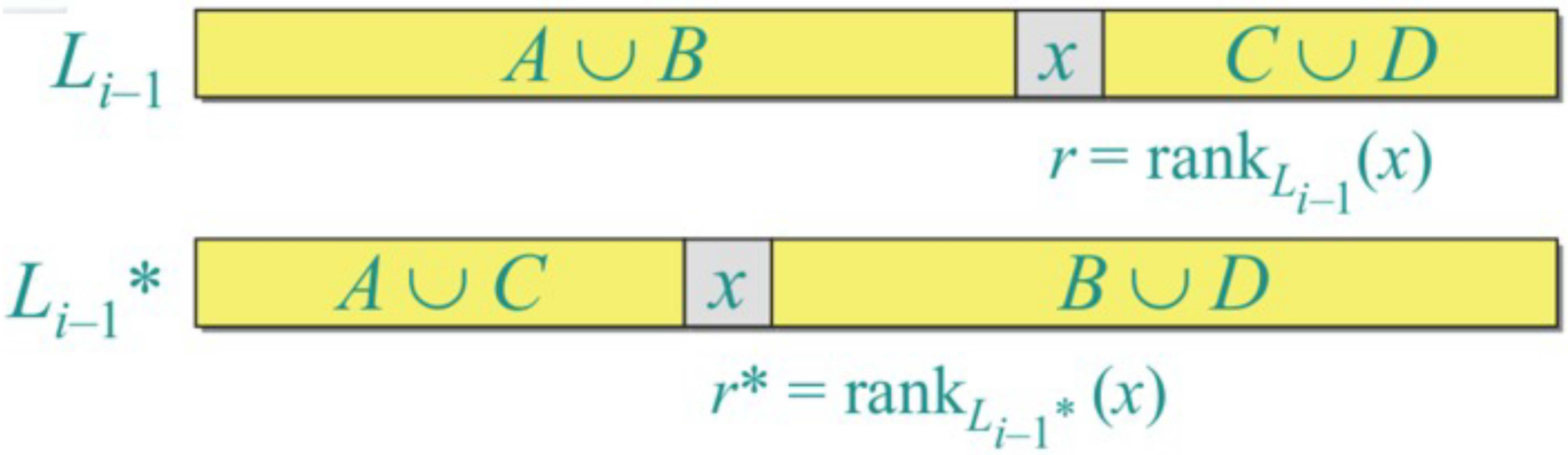
What happens on access?

Suppose that operation i access item x, and define

$$A = \{y \in L_{i-1} : y \prec_{L_{i-1}} x \text{ and } y \prec_{L^*_{i-1}} x\},$$
$$B = \{y \in L_{i-1} : y \prec_{L_{i-1}} x \text{ and } y \succ_{L^*_{i-1}} x\},$$
$$C = \{y \in L_{i-1} : y \succ_{L_{i-1}} x \text{ and } y \prec_{L^*_{i-1}} x\},$$
$$D = \{y \in L_{i-1} : y \succ_{L_{i-1}} x \text{ and } y \succ_{L^*_{i-1}} x\},$$



What happens on access?



We have $r = |A| + |B| + 1$ and $r^* = |A| + |C| + 1$.
When MTF moves x to the front, it creates $|A|$ inversions and destroys $|B|$ inversions. Each swap by OPT creates ≤ 1 inversion.
Thus, we have

$$\Phi(L_i) - \Phi_{L_{i-1}} \leq 2(|A| - |B| + t_i).$$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$\hat{c}_i = c_i + \Phi(L_i) - \Phi(L_{i-1})$$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\ &\leq 2r + 2(|A| - |B| + t_i)\end{aligned}$$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
&\leq 2r + 2(|A| - |B| + t_i) \\
&= 2r + 2(|A| - (r - 1 - |A|) + t_i) \\
&\text{(since } r = |A| + |B| + 1)
\end{aligned}$$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
&\leq 2r + 2(|A| - |B| + t_i) \\
&= 2r + 2(|A| - (r - 1 - |A|) + t_i) \\
&\quad (\text{ since } r = |A| + |B| + 1) \\
&= 2r + 4|A| - 2r + 2 + 2t_i
\end{aligned}$$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
&\leq 2r + 2(|A| - |B| + t_i) \\
&= 2r + 2(|A| - (r - 1 - |A|) + t_i) \\
&\quad (\text{ since } r = |A| + |B| + 1) \\
&= 2r + 4|A| - 2r + 2 + 2t_i \\
&= 4|A| + 2 + 2t_i
\end{aligned}$$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
&\leq 2r + 2(|A| - |B| + t_i) \\
&= 2r + 2(|A| - (r - 1 - |A|) + t_i) \\
&\quad (\text{ since } r = |A| + |B| + 1) \\
&= 2r + 4|A| - 2r + 2 + 2t_i \\
&= 4|A| + 2 + 2t_i \\
&\leq 4(r^* + t_i) \\
&\quad (\text{ since } r^* = |A| + |C| + 1 \geq |A| + 1)
\end{aligned}$$

Amortized cost

The amortized cost for the ith operation of MTF with respect to Φ is

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi(L_i) - \Phi(L_{i-1}) \\
&\leq 2r + 2(|A| - |B| + t_i) \\
&= 2r + 2(|A| - (r - 1 - |A|) + t_i) \\
&\quad (\text{ since } r = |A| + |B| + 1) \\
&= 2r + 4|A| - 2r + 2 + 2t_i \\
&= 4|A| + 2 + 2t_i \\
&\leq 4(r^* + t_i) \\
&\quad (\text{ since } r^* = |A| + |C| + 1 \geq |A| + 1) \\
&= 4c_i^*
\end{aligned}$$

The grand finale

Thus, we have

$$C_{MTF}(S) = \sum_{i=1}^{|S|} c_i$$

The grand finale

Thus, we have

$$C_{MTF}(S) = \sum_{i=1}^{|S|} c_i$$

$$= \sum_{i=1}^{|S|} (\hat{c}_i + \Phi(L_{i-1}) - \Phi(L_i))$$

The grand finale

Thus, we have

$$
\begin{aligned}
C_{MTF}(S) &= \sum_{i=1}^{|S|} c_i \\
&= \sum_{i=1}^{|S|} (\hat{c}_i + \Phi(L_{i-1}) - \Phi(L_i)) \\
&\leq (\sum_{i=1}^{|S|} 4c_i^*) + \Phi(L_0) - \Phi(L_{|S|})
\end{aligned}
$$

The grand finale

Thus, we have

$$
\begin{aligned}
C_{MTF}(S) &= \sum_{i=1}^{|S|} c_i \\
&= \sum_{i=1}^{|S|} (\hat{c}_i + \Phi(L_{i-1}) - \Phi(L_i)) \\
&\leq (\sum_{i=1}^{|S|} 4c_i^*) + \Phi(L_0) - \Phi(L_{|S|}) \\
&\leq 4C_{OPT}(s) \\
&\text{(since } \Phi(L_0) = 0 \text{ and } \Phi(L_{|S|}) \geq 0)
\end{aligned}
$$

Adding Learning to MTF

- To create a predictive version of Move-to-Front (MTF) that improves average performance while guaranteeing the classic worst-case performance of the **4-competitive** algorithm, we can use a combination of “advice” and a “ripcord.”

1. The Prediction: What is the "Hint"?

In the ski rental problem, the prediction was the length of the season. For a list, the prediction is usually the **Optimal Static Ordering**—an ML model analyzes past traffic and provides a predicted list L_{pred} sorted from "most frequently accessed" to "least frequently accessed."

2. The Simple Strategy: The "Ripcord" Method

The easiest way to build a learning-augmented algorithm that guarantees 4-competitiveness is to use a **Virtual Tracking** framework. It runs two algorithms at the same time: one in reality, and one in its "imagination."

- **The Primary (Prediction):** By default, the algorithm fully trusts the ML model. It leaves the list in the exact order of L_{pred} and does *not* move items to the front when accessed. (If the prediction is perfect, this yields an optimal cost of $1 \times \mathrm{OPT}$).
- **The Simulation (MTF):** In the background, the algorithm secretly simulates a virtual list using the standard, blind MTF rules. It keeps a running tally of what the cost *would* have been if it had just used MTF.

The “Ripcord”

- **The Ripcord (Robustness):** The algorithm constantly compares its actual cost against the virtual MTF cost. If the prediction is horribly wrong (e.g., the ML model predicted an item was useless, but it's being spammed repeatedly), the actual cost will spike. The moment $\mathrm{Cost}(\mathrm{Prediction}) \geq \mathrm{Cost}(\mathrm{MTF})$, the algorithm pulls the ripcord. It pays a small, one-time penalty to physically rearrange the real list to match the virtual MTF list, and runs standard MTF for the rest of time.