Online Algorithms with Advice

Joan Boyar, U. of Southern Denmark

April/May 2019

Overview

Section 1

The advice model

Standard online algorithm model

Input $I = \langle x_1, \ldots, x_n \rangle$ → Online algorithm → Output $O = \langle y_1, \ldots, y_n \rangle$

Adversary

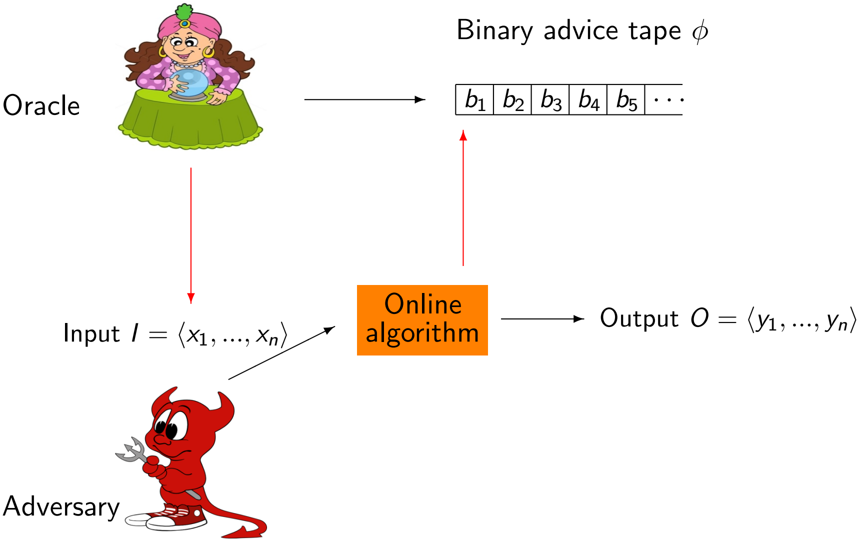
Competitive Analysis

Compare the performance of an online algorithm, ALG, with an optimal **offline** algorithm, OPT:

- OPT knows the whole sequence in the beginning.

Competitive ratio of ALG is the maximum ratio between the cost of ALG and OPT for serving the same sequence (minimization problems).

Advice from an Oracle



Advice from an Oracle

[Böckenhauer,Komm,Královič,Královič,Mömke, 2009]

Algorithms get $b(n)$ bits of **advice** for sequences of length n:

Advice from an Oracle

[Böckenhauer,Komm,Královič,Královič,Mömke, 2009]

Algorithms get $b(n)$ bits of **advice** for sequences of length n:

The advice is generated by an offline **oracle**.

Advice from an Oracle

[Böckenhauer,Komm,Královič,Královič,Mömke, 2009]

Algorithms get $b(n)$ bits of **advice** for sequences of length n:

The advice is generated by an offline **oracle**.

The advice is written on a tape and can be accessed by the online algorithm at any time.

Advice from an Oracle

[Böckenhauer,Komm,Královič,Královič,Mömke, 2009]

Algorithms get $b(n)$ bits of **advice** for sequences of length n:

The advice is generated by an offline **oracle**.

The advice is written on a tape and can be accessed by the online algorithm at any time.

It can be seen as **nondeterministically** specifying one of $2^{b(n)}$ algorithms.

Advice from an Oracle

[Böckenhauer,Komm,Královič,Královič,Mömke, 2009]

Algorithms get $b(n)$ bits of **advice** for sequences of length n:

The advice is generated by an offline **oracle**.

The advice is written on a tape and can be accessed by the online algorithm at any time.

It can be seen as **nondeterministically** specifying one of $2^{b(n)}$ algorithms.

- There are other advice models for bin packing
  - Original: [Dobrev, Královič, Markou, 2009]
  - Advice with request: [Fraigniaud,Korman,Rosén, 2011]

What is advice complexity?

Advice complexity is:

- a measure of how much knowledge of the future an online algorithm needs to achieve a certain competitive ratio

What is advice complexity?

Advice complexity is:

- a measure of how much knowledge of the future an online algorithm needs to achieve a certain competitive ratio
- a problem-independent and quantitative approach

What is advice complexity?

Advice complexity is:

- a measure of how much knowledge of the future an online algorithm needs to achieve a certain competitive ratio
- a problem-independent and quantitative approach
- a means of modelling keeping parallel solutions, multi-solution model

What is advice complexity?

Advice complexity is:

- a measure of how much knowledge of the future an online algorithm needs to achieve a certain competitive ratio
- a problem-independent and quantitative approach
- a means of modelling keeping parallel solutions, multi-solution model
- sometimes useful in practice

Relevant Questions

For a sequence of fixed length,

- How many bits of advice are required (sufficient) to achieve an optimal solution?

Relevant Questions

For a sequence of fixed length,

- How many bits of advice are required (sufficient) to achieve an optimal solution?
- How many bits of advice are sufficient to outperform all online algorithms?

Relevant Questions

For a sequence of fixed length,

- How many bits of advice are required (sufficient) to achieve an optimal solution?
- How many bits of advice are sufficient to outperform all online algorithms?
- How good can the competitive ratio be with advice of linear/sublinear size?

Relevant Questions

For a sequence of fixed length,

- How many bits of advice are required (sufficient) to achieve an optimal solution?
- How many bits of advice are sufficient to outperform all online algorithms?
- How good can the competitive ratio be with advice of linear/sublinear size?

Is there useful advice one could reasonably get (without knowing $\textsc{Opt}$)?

Optimality example: Ski rental

Rent ski equipment for one day: cost \$ 1.
Buying ski equipment: cost \$ d.
Buying once is sufficient.

Optimality example: Ski rental

Rent ski equipment for one day: cost \$ 1.
Buying ski equipment: cost \$ d.
Buying once is sufficient.

Input: List, L, of dates when you go skiing.
Output: Whether you rent, buy, or do nothing.
"Do nothing" is only possible if you have previously bought.
Goal: Minimize cost

Optimality example: Ski rental

Rent ski equipment for one day: cost \$ 1.
Buying ski equipment: cost \$ d.
Buying once is sufficient.

Input: List, L, of dates when you go skiing.
Output: Whether you rent, buy, or do nothing.
"Do nothing" is only possible if you have previously bought.
Goal: Minimize cost

Optimal algorithm: Rent until the dth date, if that comes. Then buy.
Competitive ratio $= 2 - 1/d$.

Optimality example: Ski rental

Rent ski equipment for one day: cost \$ 1.
Buying ski equipment: cost \$ d.
Buying once is sufficient.

Input: List, L, of dates when you go skiing.
Output: Whether you rent, buy, or do nothing.
"Do nothing" is only possible if you have previously bought.
Goal: Minimize cost

Optimal algorithm: Rent until the dth date, if that comes. Then buy.
Competitive ratio $= 2 - 1/d$.

[Dobrev,Královič,Pardubská, 2009]
The advice complexity for optimality is 1 bit:

$$b = \begin{cases} 0 & \text{if } |L| < d \\ 1 & \text{if } |L| \geq d \end{cases}$$

Optimality example: Paging

Cache: k pages
Slow memory: $N > k$ pages

Optimality example: Paging

Cache: k pages
Slow memory: $N > k$ pages

Request sequence: Sequence of page numbers $\langle r_1, ..., r_n \rangle$
Fault: Page requested not in cache
Cost: 1 per fault to bring page into cache
Goal: Minimize cost

Optimality example: Paging

Cache: k pages
Slow memory: $N > k$ pages

Request sequence: Sequence of page numbers $\langle r_1, ..., r_n \rangle$
Fault: Page requested not in cache
Cost: 1 per fault to bring page into cache
Goal: Minimize cost

Fact: No deterministic algorithm is better than k-competitive.

Optimality example: Paging

Cache: k pages
Slow memory: $N > k$ pages

Request sequence: Sequence of page numbers $\langle r_1, \ldots, r_n \rangle$
Fault: Page requested not in cache
Cost: 1 per fault to bring page into cache
Goal: Minimize cost

Fact: No deterministic algorithm is better than k-competitive.

[Dobrev,Královič,Pardubská, 2009]
[Böckenhauer,Komm,Královič,Královič,Mömke, 2009]
Asymptotically, for large k, the advice complexity for optimality is 1 bit per request:

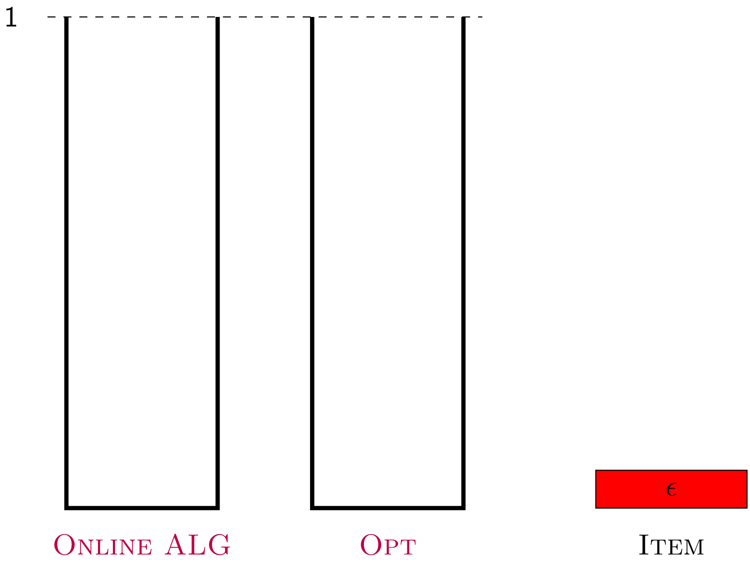
$$b_i = \begin{cases} 0 & \text{if } \textsc{Opt} \text{ would have } r_i \text{ in cache next time it is requested} \\ 1 & \text{otherwise} \end{cases}$$

Competitive ratio example: Simple knapsack problem

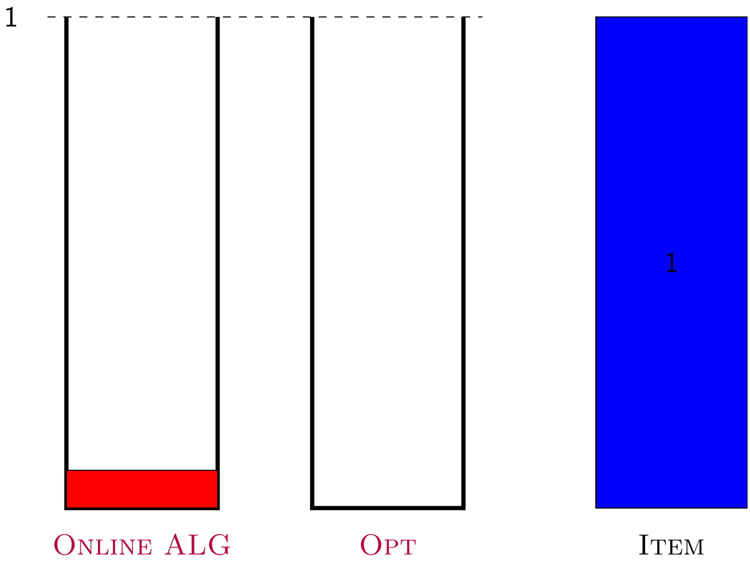
Simple knapsack problem:

- Knapsack of size 1
- Items of size $\in (0, 1]$ arrive online
- An item must be accepted or rejected
- Goal maximize total size accepted (total size ≤ 1)

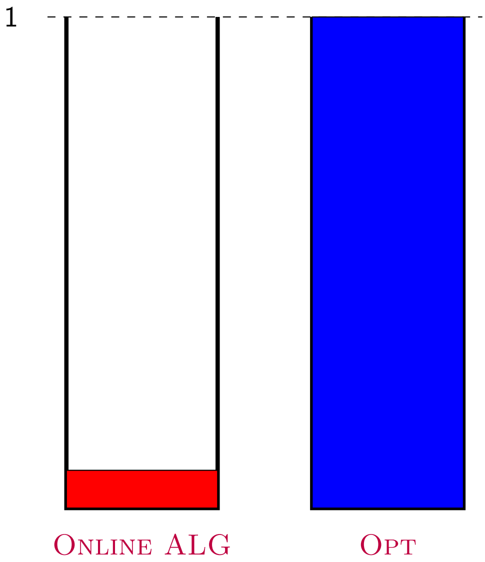
Competitive ratio example: Simple knapsack problem



Competitive ratio example: Simple knapsack problem

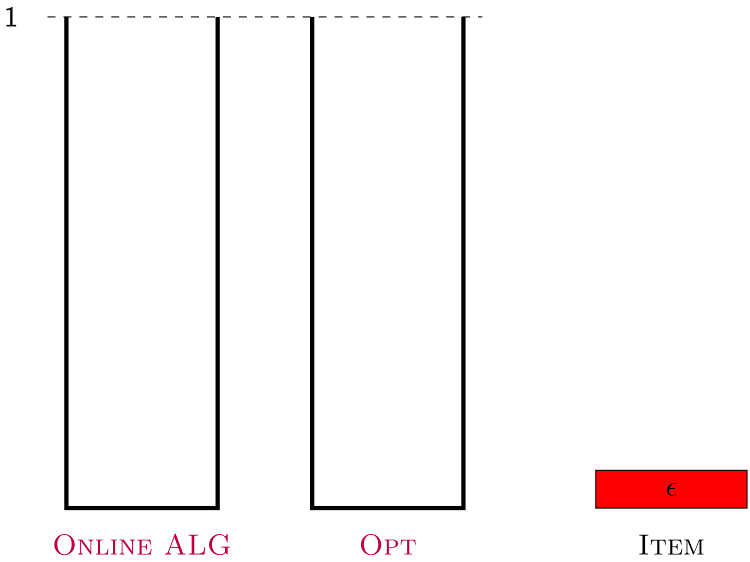


Competitive ratio example: Simple knapsack problem

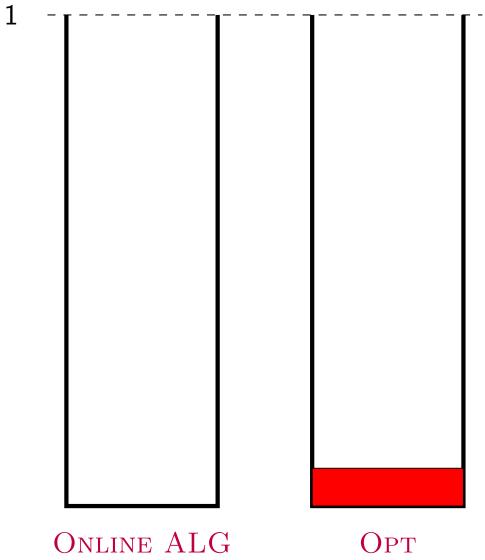


$\frac{\text{Opt}(I)}{\text{Alg}(I)}$ unbounded

Competitive ratio example: Simple knapsack problem

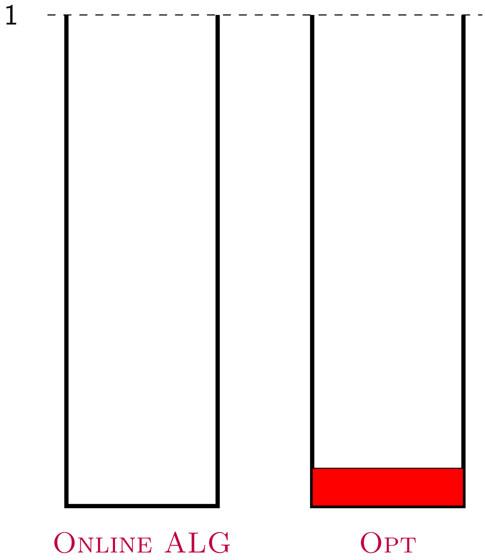


Competitive ratio example: Simple knapsack problem



No more items

Competitive ratio example: Simple knapsack problem



$\frac{\textsc{Opt}(I)}{\textsc{Alg}(I)}$ unbounded

Competitive ratio example: Simple knapsack problem

Without advice: unbounded competitive ratio.
[Marchetti-Spaccamela,Vercellis, 1995]

Competitive ratio example: Simple knapsack problem

Without advice: unbounded competitive ratio.
[Marchetti-Spaccamela,Vercellis, 1995]

With advice: 2-competitive algorithm with 1 advice bit.
[Böckenhauer,Komm,Královič,Rossmanith, 2014]

Competitive ratio example: Simple knapsack problem

Without advice: unbounded competitive ratio.
[Marchetti-Spaccamela,Vercellis, 1995]

With advice: 2-competitive algorithm with 1 advice bit.
[Böckenhauer,Komm,Královič,Rossmanith, 2014]

Algorithm:

- If $b = 0$, use Greedy.
- If $b = 1$, wait for item of size $> \frac{1}{2}$.

Competitive ratio example: Simple knapsack problem

Without advice: unbounded competitive ratio.
[Marchetti-Spaccamela,Vercellis, 1995]

With advice: 2-competitive algorithm with 1 advice bit.
[Böckenhauer,Komm,Královič,Rossmanith, 2014]

Algorithm:

- If $b = 0$, use Greedy.
- If $b = 1$, wait for item of size $> \frac{1}{2}$.

Fact: If there is no item of size $> \frac{1}{2}$, Greedy will accept at least

$$\min\{\mathrm{OPT}(I), 1/2\}$$

(in order to reject anything, it must have already accepted $\frac{1}{2}$).

Section 2

The bin packing problem

Bin Packing Problem

Input: items of various sizes $\in (0, 1]$

Output: packing of all items into unit size bins

Goal: use minimum number of bins

Bin Packing Problem

Input: items of various sizes $\in (0, 1]$

Output: packing of all items into unit size bins

Goal: use minimum number of bins

Applications: storage, cutting stock...

Offline Bin Packing Problem

The problem is NP-hard: Reduce from 2-PARTITION.

First-Fit-Decreasing has an approximation ratio of $11/9 \approx 1.22$ [Johnson,Demers,Ullman,Garey,Graham, 1974]

There is an asymptotic PTAS for the problem [de la Vega,Lueker, 1981]

Online Bin Packing Problem

Request sequence is revealed in a sequential, online manner.

Examples:

- Next-Fit
- First-Fit
- Best-Fit
- Harmonic, Harmonic++
- Heydrich, van Stee

First-Fit vs. Next-Fit — Online

First-Fit

- Find the first open bin with enough space, and place the item there
- If such a bin does not exist, open a new bin

First-Fit vs. Next-Fit — Online

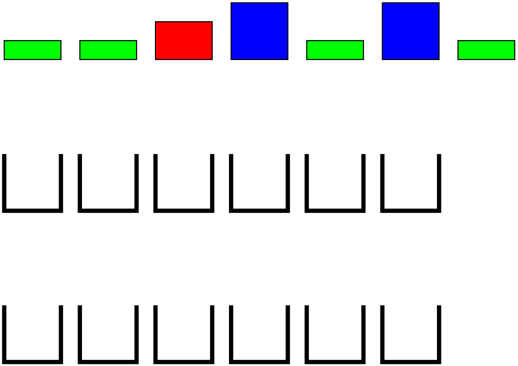
First-Fit

- Find the first open bin with enough space, and place the item there
- If such a bin does not exist, open a new bin

Next-Fit

- Put item in current open bit, if it fits
- Otherwise, close that bin and open a new current bin

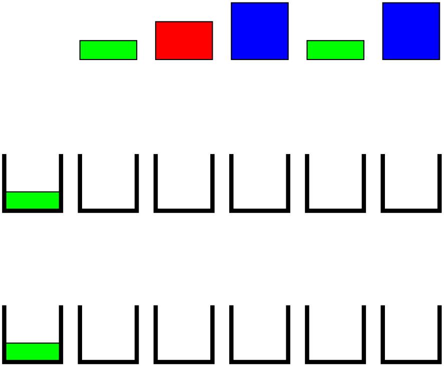
First-Fit vs. Next-Fit — Online



First-Fit

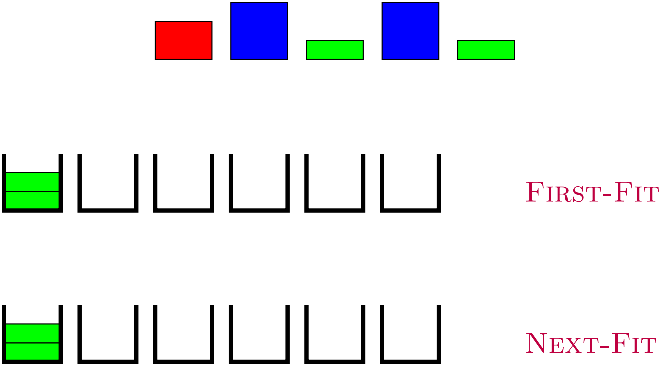
Next-Fit

First-Fit vs. Next-Fit — Online



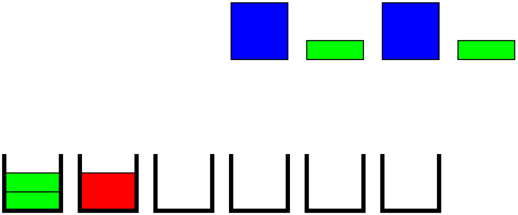
FIRST-FIT

NEXT-FIT



FIRST-FIT
NEXT-FIT

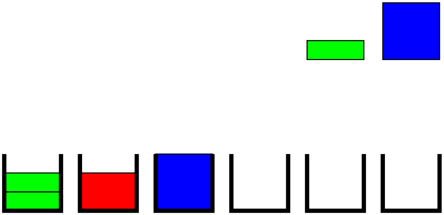
First-Fit vs. Next-Fit — Online



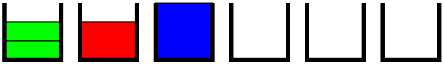
FIRST-FIT

NEXT-FIT

First-Fit vs. Next-Fit — Online

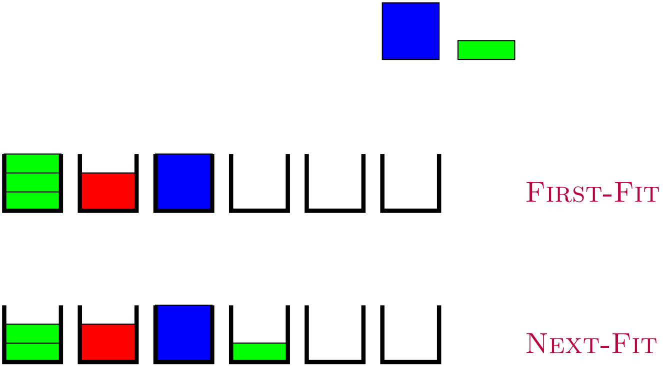


First-Fit

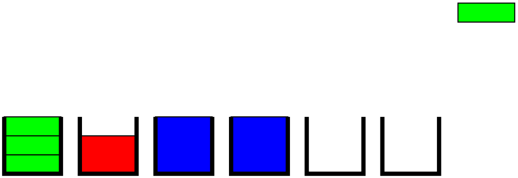


Next-Fit

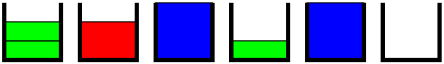
First-Fit vs. Next-Fit — Online



First-Fit vs. Next-Fit — Online

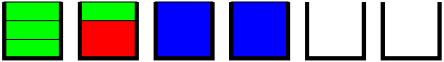


First-Fit

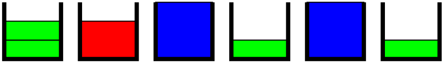


Next-Fit

First-Fit vs. Next-Fit — Online

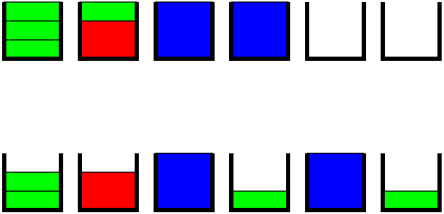


FIRST-FIT



NEXT-FIT

First-Fit vs. Next-Fit — Online



First-Fit Result: 4

Next-Fit Result: 6

Competitive Analysis

Next-Fit has competitive ratio 2
[Johnson, 1974]

Best-Fit and First-Fit have competitive ratio 1.7
[Johnson,Demers,Ullman,Garey,Graham, 1974]

Best known online algorithm has competitive ratio 1.57829
[Balogh,Békési,Dósa,Epstein,Levin, 2018]

Competitive Analysis

Next-Fit has competitive ratio 2
[Johnson, 1974]

Best-Fit and First-Fit have competitive ratio 1.7
[Johnson,Demers,Ullman,Garey,Graham, 1974]

Best known online algorithm has competitive ratio 1.57829
[Balogh,Békési,Dósa,Epstein,Levin, 2018]

No online algorithm has a competitive ratio less than 1.54278
[Balogh,Békési,Dósa,Epstein,Levin, 2018]

Competitive Analysis

Next-Fit has competitive ratio 2
[Johnson, 1974]

Best-Fit and First-Fit have competitive ratio 1.7
[Johnson,Demers,Ullman,Garey,Graham, 1974]

Best known online algorithm has competitive ratio 1.57829
[Balogh,Békési,Dósa,Epstein,Levin, 2018]

No online algorithm has a competitive ratio less than 1.54278
[Balogh,Békési,Dósa,Epstein,Levin, 2018]

Recall that offline First-Fit-Decreasing has approximation ratio ≈ 1.22.

- A big gap between quality of online and offline solutions.
- What about an "almost online" algorithm? What about advice?

Relevant Questions

For a sequence of fixed length

- How many bits of advice are required (sufficient) to achieve an optimal solution?
- How many bits of advice are sufficient to outperform all online algorithms?
- How good can the competitive ratio be with advice of linear/sublinear size?

Relevant Questions

For a sequence of fixed length

- How many bits of advice are required (sufficient) to achieve an optimal solution?
- How many bits of advice are sufficient to outperform all online algorithms?
- How good can the competitive ratio be with advice of linear/sublinear size?

Is there useful advice one could reasonably get (without knowing $\textsc{Opt}$)?

Relevant Questions

For a sequence of fixed length

- How many bits of advice are required (sufficient) to achieve an optimal solution?
- How many bits of advice are sufficient to outperform all online algorithms?
- How good can the competitive ratio be with advice of linear/sublinear size?

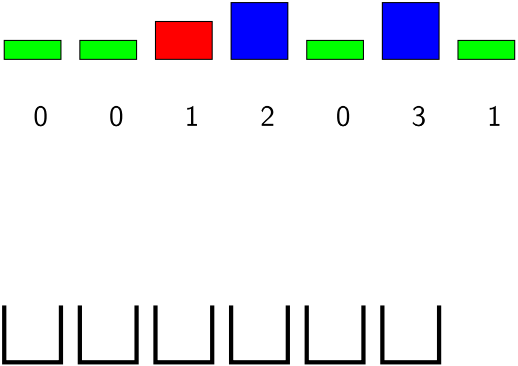
Is there useful advice one could reasonably get (without knowing $\textsc{Opt}$)?

Most advice results are from [B.,Kamali,Larsen,López-Ortiz, 2016]

Optimal Solution with Advice

How many bits of advice are sufficient to achieve an optimal solution?

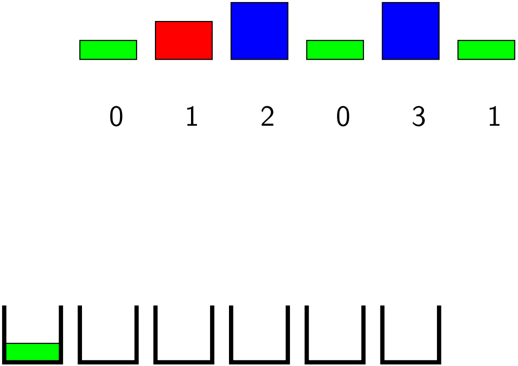
- Advice for each item: index of target bin in $\textsc{Opt}$'s packing.
- $n\lceil \log \textsc{Opt}(\sigma) \rceil$ bits of advice are sufficient



Optimal Solution with Advice

How many bits of advice are sufficient to achieve an optimal solution?

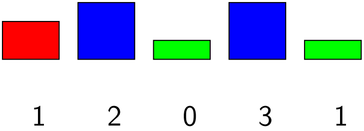
- Advice for each item: index of target bin in $\textsc{Opt}$'s packing.
- $n\lceil \log \textsc{Opt}(\sigma) \rceil$ bits of advice are sufficient



Optimal Solution with Advice

How many bits of advice are sufficient to achieve an optimal solution?

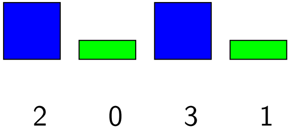
- Advice for each item: index of target bin in $\textsc{Opt}$'s packing.
- $n\lceil \log \textsc{Opt}(\sigma) \rceil$ bits of advice are sufficient



Optimal Solution with Advice

How many bits of advice are sufficient to achieve an optimal solution?

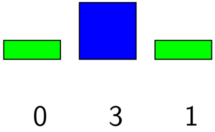
- Advice for each item: index of target bin in $\textsc{Opt}$'s packing.
- $n\lceil\log \textsc{Opt}(\sigma)\rceil$ bits of advice are sufficient

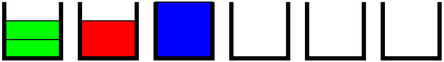


Optimal Solution with Advice

How many bits of advice are sufficient to achieve an optimal solution?

- Advice for each item: index of target bin in $\textsc{Opt}$'s packing.
- $n\lceil \log \textsc{Opt}(\sigma) \rceil$ bits of advice are sufficient

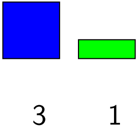


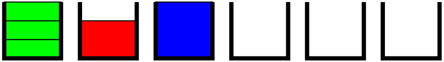


Optimal Solution with Advice

How many bits of advice are sufficient to achieve an optimal solution?

- Advice for each item: index of target bin in OPT's packing.
- $n\lceil \log \text{OPT}(\sigma) \rceil$ bits of advice are sufficient

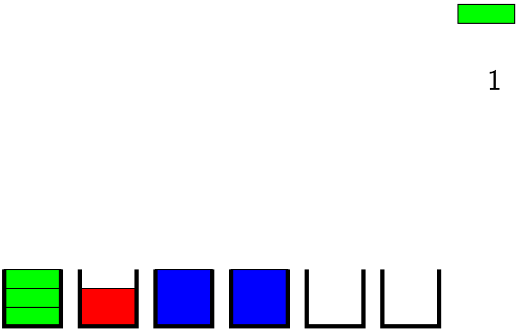




Optimal Solution with Advice

How many bits of advice are sufficient to achieve an optimal solution?

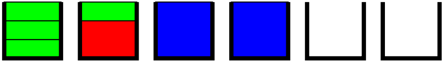
- Advice for each item: index of target bin in $\textsc{Opt}$'s packing.
- $n\lceil \log \textsc{Opt}(\sigma) \rceil$ bits of advice are sufficient



Optimal Solution with Advice

How many bits of advice are sufficient to achieve an optimal solution?

- Advice for each item: index of target bin in $\textsc{Opt}$'s packing.
- $n\lceil \log \textsc{Opt}(\sigma) \rceil$ bits of advice are sufficient



Optimal Solution with Advice

In fact, $(n - 2\,\mathbf{Opt}(\sigma)) \log \mathbf{Opt}(\sigma)$ bits of advice are required to achieve an optimal packing.

Optimal Solution with Advice

In fact, $(n - 2\,\mathbf{Opt}(\sigma))\log \mathbf{Opt}(\sigma)$ bits of advice are required to achieve an optimal packing.

Comparison:
$n\lceil \log \mathbf{Opt}(\sigma) \rceil$ bits of advice are sufficient for optimality.
$(n - 2\,\mathbf{Opt}(\sigma))\log \mathbf{Opt}(\sigma)$ bits of advice are required to guarantee optimality.

Breaking the Lower Bound — Effectively

Recall: All online algorithms have a competitive ratio of at least 1.54.

Breaking the Lower Bound — Effectively

Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Breaking the Lower Bound — Effectively

Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Advice: The number of items in range $(1/2, 2/3]$.

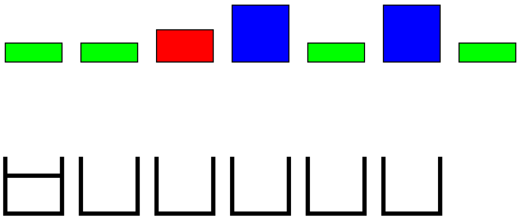
Breaking the Lower Bound — Effectively

Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Advice: The number of items in range $(1/2, 2/3]$.
Algorithm: Reserve a space of size $2/3$ for each of them
Apply First-Fit for the other items.

Advice: 1

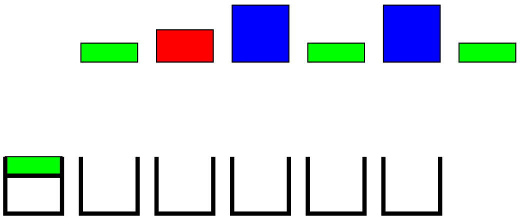


Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Advice: The number of items in range $(1/2, 2/3]$.
Algorithm: Reserve a space of size $2/3$ for each of them
Apply First-Fit for the other items.

Advice: 1

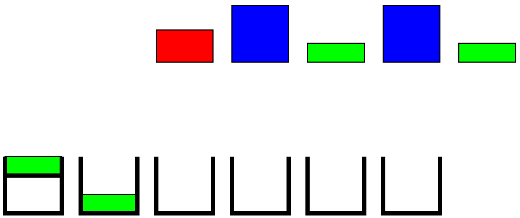


Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Advice: The number of items in range $(1/2, 2/3]$.
Algorithm: Reserve a space of size $2/3$ for each of them
Apply First-Fit for the other items.

Advice: 1



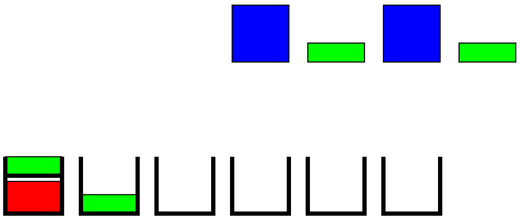
Breaking the Lower Bound — Effectively

Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Advice: The number of items in range $(1/2, 2/3]$.
Algorithm: Reserve a space of size $2/3$ for each of them
Apply First-Fit for the other items.

Advice: 1

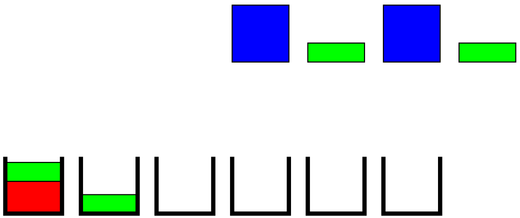


Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Advice: The number of items in range $(1/2, 2/3]$.
Algorithm: Reserve a space of size $2/3$ for each of them
Apply First-Fit for the other items.

Advice: 1



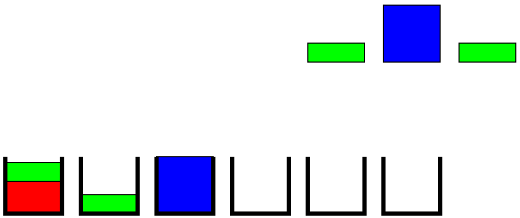
Breaking the Lower Bound — Effectively

Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Advice: The number of items in range $(1/2, 2/3]$.
Algorithm: Reserve a space of size $2/3$ for each of them
Apply First-Fit for the other items.

Advice: 1



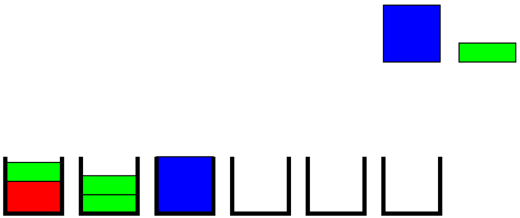
Breaking the Lower Bound — Effectively

Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Advice: The number of items in range $(1/2, 2/3]$.
Algorithm: Reserve a space of size $2/3$ for each of them
Apply First-Fit for the other items.

Advice: 1



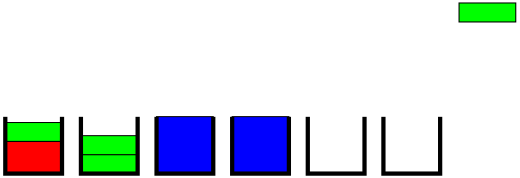
Breaking the Lower Bound — Effectively

Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Advice: The number of items in range $(1/2, 2/3]$.
Algorithm: Reserve a space of size $2/3$ for each of them
Apply First-Fit for the other items.

Advice: 1



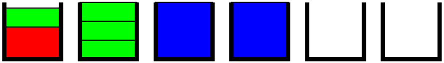
Breaking the Lower Bound — Effectively

Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Advice: The number of items in range $(1/2, 2/3]$.
Algorithm: Reserve a space of size $2/3$ for each of them
Apply First-Fit for the other items.

Advice: 1



Breaking the Lower Bound — With Less Advice

Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Breaking the Lower Bound — With Less Advice

Recall: All online algorithms have a competitive ratio of at least 1.54.

Advice of size $\lceil \log(n+1) \rceil$ is sufficient to achieve a competitive ratio of 1.5.

Newer result: [Angelopoulos,Dürr,Kamali,Renault,Rosén, 2018]

With constant advice, one can get a competitive ratio arbitrarily close to 1.47012.

Unfortunately, the advice depends on $\textsc{Opt}$.

Advice of Linear Size

An online algorithm which receives 2 bits of advice per request (plus an additive lower order term).

Advice of Linear Size

An online algorithm which receives 2 bits of advice per request (plus an additive lower order term).

Achieves a competitive ratio of $4/3 + \varepsilon$, for any positive value of ε.

Advice of Linear Size

An online algorithm which receives 2 bits of advice per request (plus an additive lower order term).

Achieves a competitive ratio of $4/3 + \varepsilon$, for any positive value of ε.

A variety of bin packing techniques are used in the proof.

Advice of Linear Size

An online algorithm which receives 2 bits of advice per request (plus an additive lower order term).

Achieves a competitive ratio of $4/3 + \varepsilon$, for any positive value of ε.

A variety of bin packing techniques are used in the proof.

Advice depends on $\textsc{Opt}$'s packing.

Advice of Linear Size

One can obtain results similar to PTAS results:

[Renault,Rósen,van Stee, 2015]

Theorem

There is an online bin packing algorithm which is $(1+3\delta)$-competitive (or asymptotically $(1+2\delta)$-competitive), using $s = \frac{1}{\delta}\log\frac{2}{\delta^2} + \log\frac{2}{\delta^2} + 3$ bits of advice per request.

A Lower Bound

A linear amount of advice is required to achieve a competitive ratio better than $9/8$.

Get a trade-off — better ratio requires more advice

A Lower Bound

A linear amount of advice is required to achieve a competitive ratio better than 9/8.

Get a trade-off — better ratio requires more advice

Reduction order:

Binary string guessing problem $\longrightarrow$ Binary separation problem

Binary separation problem $\longrightarrow$ Bin packing problem

Binary string guessing problem (with known history): 2-SGKH
[Emek,Fraigniaud,Korman,Rosén, 2011]
[Böckenhauer,Hromkovic,Komm,Krug,Smula,Sprock, 2014]

Binary String Guessing Problem

Binary string guessing problem (with known history): 2-SGKH
[Emek,Fraigniaud,Korman,Rosén, 2011]
[Böckenhauer,Hromkovic,Komm,Krug,Smula,Sprock, 2014]

- Guess the next bit in a bit string revealed in an online manner

Binary String Guessing Problem

Binary string guessing problem (with known history): 2-SGKH
[Emek,Fraigniaud,Korman,Rosén, 2011]
[Böckenhauer,Hromkovic,Komm,Krug,Smula,Sprock, 2014]

- Guess the next bit in a bit string revealed in an online manner
- $\langle 0,\ 1,\ 0,\ ?\rangle$

Binary String Guessing Problem

Binary string guessing problem (with known history): 2-SGKH
[Emek,Fraigniaud,Korman,Rosén, 2011]
[Böckenhauer,Hromkovic,Komm,Krug,Smula,Sprock, 2014]

- Guess the next bit in a bit string revealed in an online manner
- $\langle 0,\ 1,\ 0,\ ?\rangle$
- A linear amount advice is required to correctly guess more than half of the bits.

Binary String Guessing Problem

Binary string guessing problem (with known history): 2-SGKH
[Emek,Fraigniaud,Korman,Rosén, 2011]
[Böckenhauer,Hromkovic,Komm,Krug,Smula,Sprock, 2014]

- Guess the next bit in a bit string revealed in an online manner
- $\langle 0,\ 1,\ 0,\ ?\rangle$
- A linear amount advice is required to correctly guess more than half of the bits.

Theorem

On inputs of length n, any deterministic algorithm for 2-SGKH that is guaranteed to guess correctly on more than αn bits, for $1/2 \leq \alpha < 1$, needs to read at least $(1 + (1-\alpha)\log(1-\alpha) + \alpha\log(\alpha))n$ bits of advice.

Note: If we assume the number, n_0, of 0s is given, we need at least $(1 + (1-\alpha)\log(1-\alpha) + \alpha\log(\alpha))n - e(n_0)$ bits of advice, where $e(n_0) = \lceil\log(n_0+1)\rceil + 2\lceil\log(\lceil\log(n_0+1)\rceil + 1)\rceil$ (self-delimiting code).

Binary Separation Problem

Binary separation problem:

- For a sequence of $n_1 + n_2$ items decide whether an item belongs to the n_1 smaller items or n_2 larger items.

Binary Separation Problem

Binary separation problem:

- For a sequence of $n_1 + n_2$ items decide whether an item belongs to the n_1 smaller items or n_2 larger items.
- $\left\langle \frac{1}{2}\ (s),\ \frac{3}{4}\ (l),\ \frac{5}{8}\ (s),\ \frac{11}{16}\ (?) \right\rangle$

Binary Separation Problem

Binary separation problem:

- For a sequence of $n_1 + n_2$ items decide whether an item belongs to the n_1 smaller items or n_2 larger items.
- $\left\langle \frac{1}{2}\ (s),\ \frac{3}{4}\ (l),\ \frac{5}{8}\ (s),\ \frac{11}{16}\ (?) \right\rangle$
- Don't have to choose in $[0, 1]$.

Binary Separation Problem

Binary separation problem:

- For a sequence of $n_1 + n_2$ items decide whether an item belongs to the n_1 smaller items or n_2 larger items.
- $\left\langle \frac{1}{2}\ (s),\ \frac{3}{4}\ (l),\ \frac{5}{8}\ (s),\ \frac{11}{16}\ (?) \right\rangle$
- Don't have to choose in $[0, 1]$.
- Don't have to choose the exact middle value.

Reduction from 2-SGKH to Binary separation

```
small = 0; large = 1
repeat
   mid = (large + small) / 2
   class_guess = SeparationAlgorithm.ClassifyThis(mid)
   if class_guess = "large" then
      bit_guess = 0
   else
      bit_guess = 1
   actual_bit = Guess(bit_guess)
   {The actual value is received after guessing (2-SGKH).}
   if actual_bit = 0 then
      large = mid {We let "large" be the correct decision.}
   else
      small = mid {We let "small" be the correct decision.}
until end of sequence
```

Reduction from Binary separation to Bin packing

Idea: Create small and large items, so ALG has to decide which is which.

Reduction from Binary separation to Bin packing

Idea: Create small and large items, so $\textsc{Alg}$ has to decide which is which.

Give n_2 items of size $\frac{1}{2} + \epsilon$ — begin items, B.

Reduction from Binary separation to Bin packing

Idea: Create small and large items, so ALG has to decide which is which.

Give n_2 items of size $\frac{1}{2} + \epsilon$ — begin items, B.
ALG (and OPT) must put them in separate bins.

Reduction from Binary separation to Bin packing

Idea: Create small and large items, so ALG has to decide which is which.

Give n_2 items of size $\frac{1}{2} + \epsilon$ — begin items, B.
ALG (and OPT) must put them in separate bins.
Give large items, L and small items, S:

- OPT places large items with begin items.
- OPT places small items, one per bin.
- ALG much choose.

Reduction from Binary separation to Bin packing

Idea: Create small and large items, so ALG has to decide which is which.

Give n_2 items of size $\frac{1}{2} + \epsilon$ — begin items, B.
ALG (and OPT) must put them in separate bins.
Give large items, L and small items, S:

- OPT places large items with begin items.
- OPT places small items, one per bin.
- ALG much choose.

For each small item of size $\frac{1}{2} - \epsilon_i$,
give an item of size $\frac{1}{2} + \epsilon_i$ — matching items, M.

Reduction from Binary separation to Bin packing

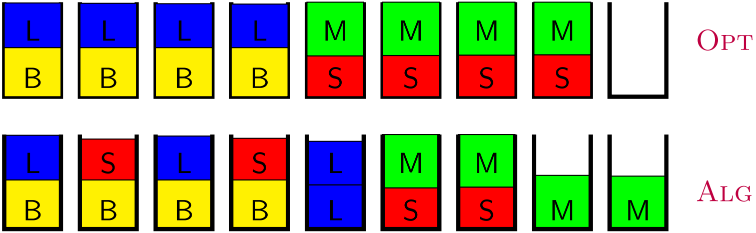
Idea: Create small and large items, so ALG has to decide which is which.

Give n_2 items of size $\frac{1}{2} + \epsilon$ — begin items, B.
ALG (and OPT) must put them in separate bins.
Give large items, L and small items, S:

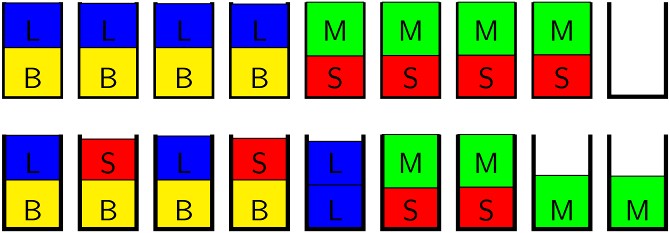
- OPT places large items with begin items.
- OPT places small items, one per bin.
- ALG much choose.

For each small item of size $\frac{1}{2} - \epsilon_i$,
give an item of size $\frac{1}{2} + \epsilon_i$ — matching items, M.
OPT packs matching items with small items, using $n_1 + n_2$ bins.

Reduction from Binary separation to Bin packing



Reduction from Binary separation to Bin packing



OPT Result: 8

ALG Result: 9

Reduction from Binary separation to Bin packing

Large item + matching item > 1.

Reduction from Binary separation to Bin packing

Large item + matching item $>$ 1.

Suppose large item is not with a begin item. Why?

- bad guess for that item
- bad guess for small item — no space

Reduction from Binary separation to Bin packing

Large item $+$ matching item > 1.

Suppose large item is not with a begin item. Why?

- bad guess for that item
- bad guess for small item — no space

Let
$x = \max\{\text{number bad guesses for small}, \text{number bad guesses for large}\}$

Reduction from Binary separation to Bin packing

Large item $+$ matching item > 1.

Suppose large item is not with a begin item. Why?

- bad guess for that item
- bad guess for small item — no space

Let
$x = \max\{\text{number bad guesses for small}, \text{number bad guesses for large}\}$
x large items not paired with begin items.

Reduction from Binary separation to Bin packing

Large item $+$ matching item $>$ 1.

Suppose large item is not with a begin item. Why?

- bad guess for that item
- bad guess for small item — no space

Let
$x = \max\{\text{number bad guesses for small}, \text{number bad guesses for large}\}$
x large items not paired with begin items.
At most 2 fit in a bin together.

Reduction from Binary separation to Bin packing

Large item $+$ matching item > 1.

Suppose large item is not with a begin item. Why?

- bad guess for that item
- bad guess for small item — no space

Let
$x = \max\{\text{number bad guesses for small}, \text{number bad guesses for large}\}$
x large items not paired with begin items.
At most 2 fit in a bin together.

4 errors in binary separation $\Rightarrow \geq 1$ more bin

Lower bound result for bin packing

Theorem

On inputs of length n, to achieve a competitive ratio of c ($1 < c < 9/8$), an online algorithm must get at least $(1 + (4c-4)\log(4c-4) + (5-4c)\log(5-4c))n - e(n)$ bits of advice.

Recall that $e(n) = \lceil \log(n+1) \rceil + 2\lceil \log(\lceil \log(n+1) \rceil + 1) \rceil$.

Lower bound result for bin packing

Theorem

On inputs of length n, to achieve a competitive ratio of c $(1 < c < 9/8)$*, an online algorithm must get at least*
$(1 + (4c-4)\log(4c-4) + (5-4c)\log(5-4c))n - e(n)$ *bits of advice.*

Recall that $e(n) = \lceil \log(n+1) \rceil + 2\lceil \log(\lceil \log(n+1) \rceil + 1) \rceil$.

Newer result: [Angelopoulos,Dürr,Kamali,Renault,Rosén, 2016]
Can improve analysis from 4 mistakes causing at least 1 extra bin to 3 mistakes causing at least 1 extra bin.

Theorem

On inputs of length n, to achieve a competitive ratio of c $(1 < c < 7/6)$*, an online algorithm must get at least*
$(1 + (3c-3)\log(3c-3) + (4-3c)\log(4-3c))n - e(n)$ *bits of advice.*

Recall that $e(n) = \lceil \log(n+1) \rceil + 2\lceil \log(\lceil \log(n+1) \rceil + 1) \rceil$.

Lower bound result for bin packing

Even newer result: [Mikkelsen, 2016]
Can improve analysis using above result and using weighted binary string guessing.

Theorem

On inputs of length n, to achieve a competitive ratio of c ($1 < c < 4 - 2\sqrt{2}$), a randomized c-competitive bin packing algorithm must read at least $\Omega(n)$ bits of advice.

$9/8 = 1.125$, $7/6 \approx 1.1667$, $4 - 2\sqrt{2} \approx 1.1716$

Lower bound result for bin packing

[Mikkelsen, 2016]

Theorem

On inputs of length n, to achieve a competitive ratio of c ($1 < c < 4 - 2\sqrt{2}$), a randomized c-competitive bin packing algorithm must read at least $\Omega(n)$ bits of advice.

Lower bound result for bin packing

[Mikkelsen, 2016]

Theorem

On inputs of length n, to achieve a competitive ratio of c ($1 < c < 4 - 2\sqrt{2}$), a randomized c-competitive bin packing algorithm must read at least $\Omega(n)$ bits of advice.

What does this say?

- One cannot get a competitive ratio better than 1.17 by giving information such as the number of bins that OPT uses.
- One cannot beat 1.17 by keeping $2^{o(n)}$ solutions in the multi-solution model.

Lower bound result for bin packing

[Mikkelsen, 2016]

Theorem

On inputs of length n, to achieve a competitive ratio of c ($1 < c < 4 - 2\sqrt{2}$), a randomized c-competitive bin packing algorithm must read at least $\Omega(n)$ bits of advice.

[Renault, Rosén, van Stee, 2015] For a fixed competitive ratio, there exists an online algorithm which only needs linear advice:
They present an algorithm for online bin packing which is $(1 + 3\delta)$-competitive (or asymptotically $(1 + 2\delta)$-competitive), using $s = \frac{1}{\delta} \log \frac{2}{\delta^2} + \log \frac{2}{\delta^2} + 3$ bits of advice per request.

Open Problems

- Linear advice is needed to be c-competitive, $c < 1.17$.
  Linear advice is sufficient for any fixed c. There is a huge gap, though.
- $(2 + o(1))n$ advice is sufficient to be $(4/3 + \epsilon)$-competitive.
  Can one get a better ratio with so few bits?
- Counting the number of items of size in $(1/2, 2/3]$
  ($O(\log n)$ bits of advice) is sufficient to be $3/2$-competitive.
  Are there other algorithms (based on advice) which could be practical?

Section 3

An advice complexity class, AOC

B.,Favrholdt,Kudahl,Mikkelsen, 2017

- Introduce an advice complexity class, Asymmetric Online Covering (AOC).
- Prove upper bound on advice complexity for all problems in AOC; advice complexity for competitive ratio c:

$$\frac{0.53n}{c} \leq \log_2\left(1 + \frac{(c-1)^{c-1}}{c^c}\right) n \leq \frac{n}{c}$$

- Many problems, including Vertex Cover, Independent Set, Set Cover, and Dominating Set are AOC-Complete.

Vertex Cover

Vertex Cover problem:

- Vertices of graph $G = (V, E)$ arrive online, with edges to previous vertices.
- Vertices must be accepted or rejected.
- Accepted vertices, $C \subseteq V$, must be a vertex cover, i.e. $\forall (u, v) \in E$, either u or $v \in C$.
- Goal: Minimize $|C|$.

Vertex Cover

Vertex Cover problem:

- Vertices of graph $G = (V, E)$ arrive online, with edges to previous vertices.
- Vertices must be accepted or rejected.
- Accepted vertices, $C \subseteq V$, must be a vertex cover, i.e. $\forall(u, v) \in E$, either u or $v \in C$.
- Goal: Minimize $|C|$.

Note: Due to the vertex-arrival model, the 2-approximation algorithm which takes both endpoints of an uncovered edge, cannot be used.

Vertex Cover

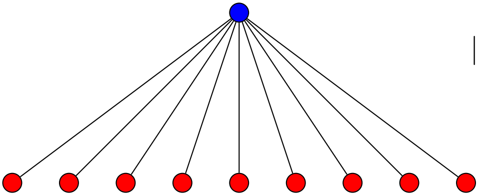
ALG

OPT

ALG must reject vertex; otherwise it will be the last.

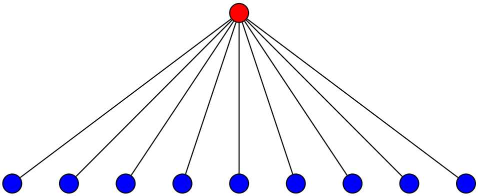
Vertex Cover

ALG



$|C| = n - 1$

OPT



$|C| = 1$

Vertex Cover

The competitive ratio for Vertex Cover is unbounded.

Vertex Cover

The competitive ratio for Vertex Cover is unbounded.

Can achieve optimality with n advice bits.

Vertex Cover

The competitive ratio for Vertex Cover is unbounded.

Can achieve optimality with n advice bits.

Can we achieve constant competitive ratio with small advice?

Vertex Cover

The competitive ratio for Vertex Cover is unbounded.

Can achieve optimality with n advice bits.

Can we achieve constant competitive ratio with small advice?

No.

Vertex Cover

The competitive ratio for Vertex Cover is unbounded.

Can achieve optimality with n advice bits.

Can we achieve constant competitive ratio with small advice?

No.

How many bits do we need to be c-competitive?

Advice complexity of Vertex Cover

Let V_{OPT} be an optimal vertex cover.
Let $k = |V_{\mathrm{OPT}}|$.

Advice complexity of Vertex Cover

Let V_{OPT} be an optimal vertex cover.
Let $k = |V_{\text{OPT}}|$.

ALG selects vertices V_{ALG} s.t.

- $|V_{\text{ALG}}| \leq ck$
- $V_{\text{OPT}} \subseteq V_{\text{ALG}}$

Advice complexity of Vertex Cover

Let V_{OPT} be an optimal vertex cover.
Let $k = |V_{\text{OPT}}|$.

ALG selects vertices V_{ALG} s.t.

- $|V_{\text{ALG}}| \leq ck$
- $V_{\text{OPT}} \subseteq V_{\text{ALG}}$

Oracle will specify which vertices to accept.
How many ck-subsets are needed to cover all k-subsets?

Advice complexity of Vertex Cover

Suppose $n = |V| = 6$, $c = 2$, and $|V_{\mathrm{OPT}}| = 2$.
There are $\binom{6}{2} = 15$ possibilities for V_{OPT}.

Advice complexity of Vertex Cover

Suppose $n = |V| = 6$, $c = 2$, and $|V_{\mathrm{OPT}}| = 2$.
There are $\binom{6}{2} = 15$ possibilities for V_{OPT}.

1 2 3 4

1 2 5 6

3 4 5 6

These 3 subsets of size $ck = 4$ cover all 2-subsets of $\{1, 2, 3, 4, 5, 6\}$.

Advice complexity of Vertex Cover

Suppose $n = |V| = 6$, $c = 2$, and $|V_{\mathrm{OPT}}| = 2$.
There are $\binom{6}{2} = 15$ possibilities for V_{OPT}.

1 2 3 4
1 2 5 6
3 4 5 6

These 3 subsets of size $ck = 4$ cover all 2-subsets of $\{1, 2, 3, 4, 5, 6\}$.

2 advice bits are sufficient to specify one which covers V_{OPT}.

Covering designs

Given a universe of size n, a collection of ck-subsets covering all k-subsets is an (n, ck, k)-covering design.

Covering designs

Given a universe of size n, a collection of ck-subsets covering all k-subsets is an (n, ck, k)-covering design.
The minimum size of an (n, ck, k)-covering design is the covering number, $C(n, ck, k)$.

Covering designs

Given a universe of size n, a collection of ck-subsets covering all k-subsets is an (n, ck, k)-covering design.
The minimum size of an (n, ck, k)-covering design is the covering number, $C(n, ck, k)$.

By the Pigeonhole Principle,

$$\frac{\binom{n}{k}}{\binom{ck}{k}} \leq C(n, ck, k)$$

Covering designs

Given a universe of size n, a collection of ck-subsets covering all k-subsets is an (n, ck, k)-covering design.
The minimum size of an (n, ck, k)-covering design is the covering number, $C(n, ck, k)$.

Covering designs

Given a universe of size n, a collection of ck-subsets covering all k-subsets is an (n, ck, k)-covering design.
The minimum size of an (n, ck, k)-covering design is the covering number, $C(n, ck, k)$.

[Erdös,Spencer, 1974]

$$\frac{\binom{n}{k}}{\binom{ck}{k}} \leq C(n, ck, k) \leq \frac{\binom{n}{k}}{\binom{ck}{k}} \left(1 + \ln \binom{ck}{k}\right)$$

Algorithm with advice for Vertex Cover

Oracle writes:

- values of n and k — $O(\log n)$ bits
- index of a ck-subset, C, in an (n, ck, k)-covering design, s.t. $V_{\mathrm{OPT}} \subseteq C$.

Algorithm with advice for Vertex Cover

Oracle writes:

- values of n and k — $O(\log n)$ bits
- index of a ck-subset, C, in an (n, ck, k)-covering design, s.t. $V_{\text{OPT}} \subseteq C$.

ALG uses $C = \langle b_1, b_2, ..., b_n \rangle$:

- if $b_i = 1$, accept v_i
- if $b_i = 0$, reject v_i

Algorithm with advice for Vertex Cover

Oracle writes:

- values of n and k — $O(\log n)$ bits
- index of a ck-subset, C, in an (n, ck, k)-covering design, s.t. $V_{\text{OPT}} \subseteq C$.

ALG uses $C = \langle b_1, b_2, ..., b_n \rangle$:

- if $b_i = 1$, accept v_i
- if $b_i = 0$, reject v_i

ALG is c-competitive. It reads at most $B = \log \max_{k:ck<n} C(n, ck, k) + O(\log n)$ advice bits.

$$B \leq \log \left(1 + \frac{(c-1)^{c-1}}{c^c}\right) n + O(\log n) \leq \frac{n}{c} + O(\log n)$$

A class of online problems

Generalizing from the properties used for Vertex Cover, we define a class of online problems.

A class of online problems

Generalizing from the properties used for Vertex Cover, we define a class of online problems.

It applies to *accept/reject* online problems: The online algorithm only accepts or rejects each request. Let the set of accepted requests be Y.

A class of online problems

Generalizing from the properties used for Vertex Cover, we define a class of online problems.

It applies to *accept/reject* online problems: The online algorithm only accepts or rejects each request. Let the set of accepted requests be Y.

Definition

An accept/reject minimization problem is in *Asymmetric Online Covering (AOC)* if

1. If Y is feasible, $\text{cost}(Y) = |Y|$. Otherwise $\text{cost}(Y) = \infty$.
2. A superset of a feasible solution is feasible.

Example problems in AOC:
Vertex Cover, Dominating Set, Set Cover, Cycle Finding.

A class of online problems

Generalizing from the properties used for Vertex Cover, we define a class of online problems.

It applies to *accept/reject* online problems: The online algorithm only accepts or rejects each request. Let the set of accepted requests be Y.

Definition

An accept/reject maximization problem is in *Asymmetric Online Covering (AOC)*, if

1. If Y is feasible, $\mathrm{cost}(Y) = |Y|$. Otherwise $\mathrm{cost}(Y) = -\infty$.
2. A subset of a feasible solution is feasible.

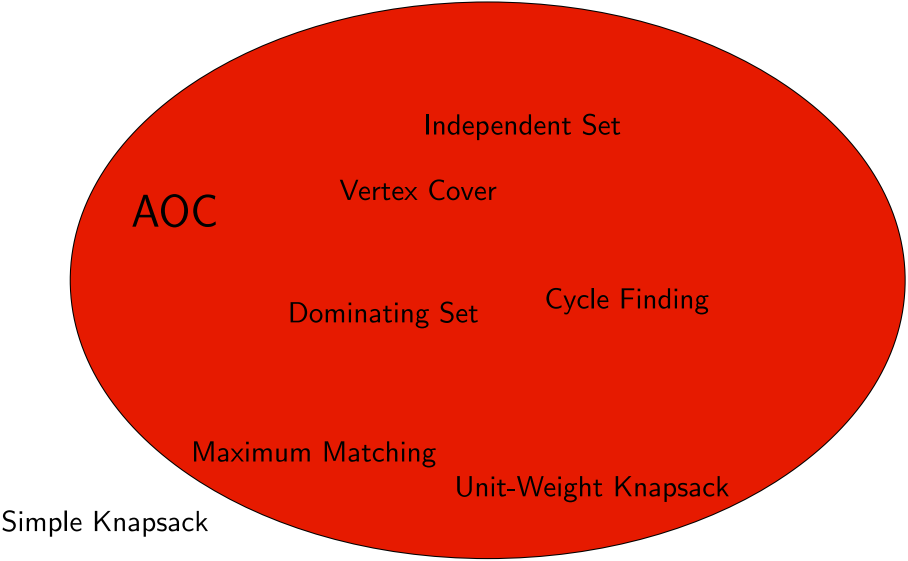
Example problems in AOC:
Independent Set, Disjoint Path Allocation.

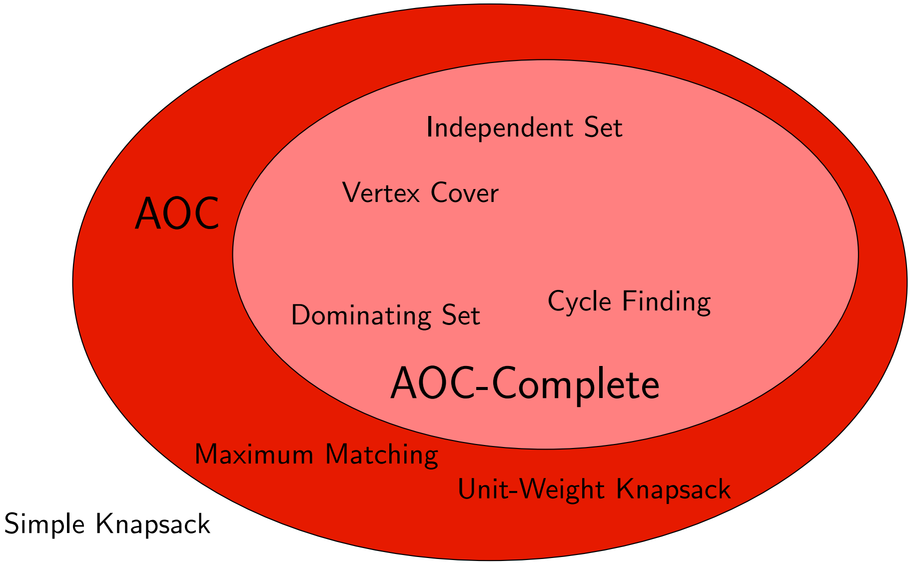
AOC-Complete

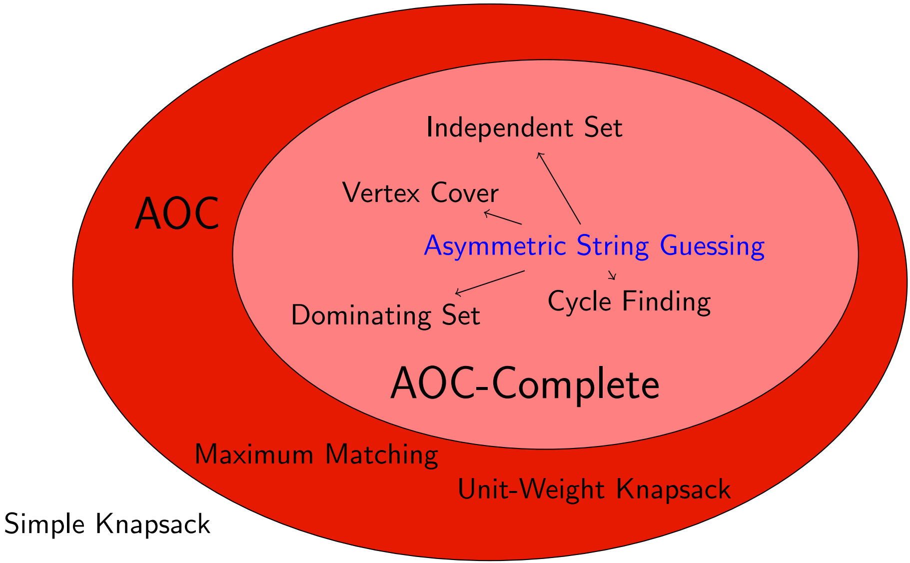
Definition

A problem in AOC is *AOC-Complete* if $\log\left(1+\frac{(c-1)^{c-1}}{c^c}\right)n - O(\log n)$ advice bits are necessary to achieve a competitive ratio of c.

Example AOC-Complete problems:
Vertex Cover, Dominating Set, Cycle Finding.


Independent Set
Vertex Cover
AOC
Dominating Set
Cycle Finding
Maximum Matching
Unit-Weight Knapsack
Simple Knapsack


Independent Set
Vertex Cover
AOC
Dominating Set
Cycle Finding
AOC-Complete
Maximum Matching
Unit-Weight Knapsack
Simple Knapsack


Independent Set
Vertex Cover
AOC
Asymmetric String Guessing
Cycle Finding
Dominating Set
AOC-Complete
Maximum Matching
Unit-Weight Knapsack
Simple Knapsack

Asymmetric string guessing

Similar to string guessing.

Asymmetric string guessing

Similar to string guessing.

Definition

The online problem, MINASG, is as follows:

- The input is a (secret) string $x \in \{0,1\}^n$.
- In round i, the algorithm answers $a_i \in \{0,1\}$.
- The correct answer, x_i is then revealed (known history).
- If $x_i = 1$ and $a_i = 0$, the algorithm loses (cost ∞).
- The cost of a feasible solution is $\sum_{i=1}^{n} a_i$.
- The goal is to minimize this cost.

Asymmetric string guessing

Results:

- MINASG is in AOC.
- MINASG is AOC-Complete:

$$\log\left(1+\frac{(c-1)^{c-1}}{c^c}\right)n - O(\log n)$$

  advice bits are necessary to achieve a competitive ratio of c.
- MINASG can be reduced to other problems.
  - Vertex Cover
  - Dominating Set
  - Set Cover
  - Cycle Finding

Asymmetric string guessing is in AOC

Recall:

Definition

An accept/reject minimization problem is in *(AOC)*, *Asymmetric Online Covering*, if, for an online solution, Y:

1. If Y is feasible, $\text{cost}(Y) = |Y|$. Otherwise $\text{cost}(Y) = \infty$.
2. A superset of a feasible solution is feasible.

Asymmetric string guessing is in AOC

Recall:

Definition

An accept/reject minimization problem is in *(AOC), Asymmetric Online Covering*, if, for an online solution, Y:

1. If Y is feasible, $\text{cost}(Y) = |Y|$. Otherwise $\text{cost}(Y) = \infty$.
2. A superset of a feasible solution is feasible.

Interpret guessing $a_i = 1$ as accept and $a_i = 0$ as reject.
A solution is the set of indices where $a_i = 1$.

Asymmetric string guessing is in AOC

Recall:

Definition

An accept/reject minimization problem is in *(AOC), Asymmetric Online Covering*, if, for an online solution, Y:

1. If Y is feasible, $\text{cost}(Y) = |Y|$. Otherwise $\text{cost}(Y) = \infty$.
2. A superset of a feasible solution is feasible.

Interpret guessing $a_i = 1$ as accept and $a_i = 0$ as reject.
A solution is the set of indices where $a_i = 1$.

If a set is feasible, $(x_i = 1) \Rightarrow (a_i = 1)$, so any superset is feasible.
Clearly, ASG is in AOC.

Asymmetric string guessing is AOC-Complete

Theorem

A c-competitive algorithm, ALG*, for* MIN*ASG must read at least b bits advice, where*

$$b \geq \log_2 \left(\max_{t:\lfloor ct \rfloor < n} \frac{\binom{n}{t}}{\binom{\lfloor ct \rfloor}{t}} \right) - O(\log_2 n) = \log \left(1 + \frac{(c-1)^{c-1}}{c^c} \right) n - O(\log n)$$

Pf sketch (by contradiction)

Let $b =$ max number of advice bits read, given n, c.

Suppose $\exists t$, s.t. $\lfloor ct \rfloor < n$, $b < \log_2 \left(\frac{\binom{n}{t}}{\binom{\lfloor ct \rfloor}{t}} \right)$.

Asymmetric string guessing is AOC-Complete

Theorem

A c-competitive algorithm, ALG*, for* MIN*ASG must read at least b bits advice, where*

$$b \geq \log_2 \left(\max_{t:\lfloor ct \rfloor < n} \frac{\binom{n}{t}}{\binom{\lfloor ct \rfloor}{t}} \right) - O(\log_2 n) = \log \left(1 + \frac{(c-1)^{c-1}}{c^c} \right) n - O(\log n)$$

Pf sketch (by contradiction)

Let $b =$ max number of advice bits read, given n, c.

Suppose $\exists t$, s.t. $\lfloor ct \rfloor < n$, $b < \log_2 \left(\frac{\binom{n}{t}}{\binom{\lfloor ct \rfloor}{t}} \right)$.

$I_{n,t} = \{\bar{x} \in \{0,1\}^n \mid t = \sum_{i=1}^{n} x_i\}$
$|I_{n,t}| = \binom{n}{t}$
Let the set of strings where the Oracle gives advice ϕ be $I_{n,t}^{\phi} \subseteq I_{n,t}$.

ASG is AOC-Complete, cont.

Since $b < \log_2 \left(\frac{\binom{n}{t}}{\binom{\lfloor ct \rfloor}{t}} \right)$, number of different advice strings $< \frac{\binom{n}{t}}{\binom{\lfloor ct \rfloor}{t}}$.
By Pigeonhole Principle and $|I_{n,t}| = \binom{n}{t}$,

$$\exists \phi \;\; \text{s.t.} \;\; |I^{\phi}_{n,t}| > \binom{\lfloor ct \rfloor}{t}$$

ASG is AOC-Complete, cont.

Since $b < \log_2\left(\frac{\binom{n}{t}}{\binom{\lfloor ct \rfloor}{t}}\right)$, number of different advice strings $< \frac{\binom{n}{t}}{\binom{\lfloor ct \rfloor}{t}}$.
By Pigeonhole Principle and $|I_{n,t}| = \binom{n}{t}$,

$$\exists \phi \ \text{ s.t. } \ |I^{\phi}_{n,t}| > \binom{\lfloor ct \rfloor}{t}$$

Claim $\exists \bar{x} \in I^{\phi}_{n,t}$ where $\textsc{Alg}$ answers 1 at least $\lfloor ct \rfloor + 1$ times.

ASG is AOC-Complete, cont.

Since $b < \log_2\left(\frac{\binom{n}{t}}{\binom{\lfloor ct \rfloor}{t}}\right)$, number of different advice strings $< \frac{\binom{n}{t}}{\binom{\lfloor ct \rfloor}{t}}$.

By Pigeonhole Principle and $|I_{n,t}| = \binom{n}{t}$,

$$\exists \phi \ \text{ s.t. } \ |I^{\phi}_{n,t}| > \binom{\lfloor ct \rfloor}{t}$$

Claim $\exists \bar{x} \in I^{\phi}_{n,t}$ where $\textsc{Alg}$ answers 1 at least $\lfloor ct \rfloor + 1$ times.

Given $\textsc{Alg}$ and $I^{\phi}_{n,t}$, create an Adversary which forces this.

Consider strings in $I^{\phi}_{n,t}$ which are possible after Adversary has revealed some bits.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 001 & {\color{red}0}01110 \\ 001 & {\color{red}0}01101 \\ 001 & {\color{red}0}01011 \\ 001 & {\color{red}0}00111 \\ 001 & {\color{red}0}10011 \end{array}$$

Note: $n = 9$, $t = 4$. $\langle x_1, x_2, x_3 \rangle = \langle 0, 0, 1 \rangle$ already known.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 001 & 001110 \\ 001 & 001101 \\ 001 & 001011 \\ 001 & 000111 \\ 001 & 010011 \end{array}$$

Note: $n = 9$, $t = 4$. $\langle x_1, x_2, x_3 \rangle = \langle 0, 0, 1 \rangle$ already known.

ALG is guessing x_4. It can answer $a_4 = 0$.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 001 & 001110 \\ 001 & 001101 \\ 001 & 001011 \\ 001 & 000111 \\ 001 & 010011 \end{array}$$

Note: $n = 9$, $t = 4$. $\langle x_1, x_2, x_3 \rangle = \langle 0, 0, 1 \rangle$ already known.

ALG is guessing x_4. It can answer $a_4 = 0$.

Adversary has to reveal $x_4 = 0$.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 0010 & 01110 \\ 0010 & 01101 \\ 0010 & 01011 \\ 0010 & 00111 \\ 0010 & 10011 \end{array}$$

Note: $n = 9$, $t = 4$. $\langle x_1, x_2, x_3, x_4 \rangle = \langle 0, 0, 1, 0 \rangle$ already known.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 0010 & 01110 \\ 0010 & 01101 \\ 0010 & 01011 \\ 0010 & 00111 \\ 0010 & 10011 \end{array}$$

Note: $n = 9$, $t = 4$. $\langle x_1, x_2, x_3, x_4 \rangle = \langle 0, 0, 1, 0 \rangle$ already known.

ALG is guessing x_5. It has to answer $a_5 = 1$.
Otherwise Adversary chooses last string 001010011, and ALG has lost.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 0010 & 01110 \\ 0010 & 01101 \\ 0010 & 01011 \\ 0010 & 00111 \\ 0010 & 10011 \end{array}$$

Note: $n = 9$, $t = 4$. $\langle x_1, x_2, x_3, x_4 \rangle = \langle 0, 0, 1, 0 \rangle$ already known.

ALG is guessing x_5. It has to answer $a_5 = 1$.
Otherwise Adversary chooses last string 001010011, and ALG has lost.

Suppose Adversary reveals $x_5 = 1$.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 0010 & 01110 \\ 0010 & 01101 \\ 0010 & 01011 \\ 0010 & 00111 \\ 0010 & 10011 \end{array}$$

Note: $n = 9,\ t = 4.\ \langle x_1, x_2, x_3, x_4 \rangle = \langle 0, 0, 1, 0 \rangle$ already known.

ALG is guessing x_5. It has to answer $a_5 = 1$.
Otherwise Adversary chooses last string 001010011, and ALG has lost.

Suppose Adversary reveals $x_5 = 1$.
Then the new $I_\phi = \{001010011\}$. ALG makes no more errors.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 0010 & 01110 \\ 0010 & 01101 \\ 0010 & 01011 \\ 0010 & 00111 \\ 0010 & 10011 \end{array}$$

Note: $n = 9$, $t = 4$. $\langle x_1, x_2, x_3, x_4 \rangle = \langle 0, 0, 1, 0 \rangle$ already known.

ALG is guessing x_5. It has to answer $a_5 = 1$.
Otherwise Adversary chooses last string 001010011, and ALG has lost.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 0010 & 01110 \\ 0010 & 01101 \\ 0010 & 01011 \\ 0010 & 00111 \\ 0010 & 10011 \end{array}$$

Note: $n = 9$, $t = 4$. $\langle x_1, x_2, x_3, x_4 \rangle = \langle 0, 0, 1, 0 \rangle$ already known.

ALG is guessing x_5. It has to answer $a_5 = 1$.
Otherwise Adversary chooses last string 001010011, and ALG has lost.

Suppose Adversary reveals $x_5 = 0$.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 0010 & 01110 \\ 0010 & 01101 \\ 0010 & 01011 \\ 0010 & 00111 \\ 0010 & 10011 \end{array}$$

Note: $n = 9$, $t = 4$. $\langle x_1, x_2, x_3, x_4 \rangle = \langle 0, 0, 1, 0 \rangle$ already known.

ALG is guessing x_5. It has to answer $a_5 = 1$.
Otherwise Adversary chooses last string 001010011, and ALG has lost.

Suppose Adversary reveals $x_5 = 0$.
Then the new I_ϕ contains 4 strings. ALG must answer 1 for remaining bits.

ASG is AOC-Complete, cont.

Round i:

Let $m = |I_\phi|$, $h =$ number of 1's remaining in each string.
Let $m_0 =$ number of strings in I_ϕ where $x_i = 0$. $m_1 = m - m_0$.

ASG is AOC-Complete, cont.

Round i:

Let $m = |I_\phi|$, $h =$ number of 1's remaining in each string.
Let $m_0 =$ number of strings in I_ϕ where $x_i = 0$. $m_1 = m - m_0$.

If $m_0 = m$, Adversary answers $x_i = 0$.
If $m_0 < m$, look at minimum number of columns with ones:

- Let $d_1 = \min\{d' \mid m_1 \leq \binom{d'}{h-1}\}$
- Let $d = \min\{d' \mid m \leq \binom{d'}{h}\}$
- If $d_1 + 1 \geq d$ then

  Adversary answers $x_i = 1$.

  Otherwise

  Adversary answers $x_i = 0$.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 0010 & 01110 \\ 0010 & 01101 \\ 0010 & 01011 \\ 0010 & 00111 \\ 0010 & 10011 \end{array}$$

Note: $n = 9$, $t = 4$. $\langle x_1, x_2, x_3, x_4 \rangle = \langle 0, 0, 1, 0 \rangle$ already known.

ALG is guessing x_5. It has to answer $a_5 = 1$.

ASG is AOC-Complete, cont.

$$I_\phi = \begin{array}{c|c} 0010 & 01110 \\ 0010 & 01101 \\ 0010 & 01011 \\ 0010 & 00111 \\ 0010 & 10011 \end{array}$$

Note: $n = 9$, $t = 4$. $\langle x_1, x_2, x_3, x_4 \rangle = \langle 0, 0, 1, 0 \rangle$ already known.

ALG is guessing x_5. It has to answer $a_5 = 1$.

Note: $m = 5$, $h = 3$, $m_0 = 4$, $m_1 = 1$.
$d_1 = \min\{d' \mid m_1 \leq \binom{d'}{h-1}\} = \min\{d' \mid 1 \leq \binom{d'}{2}\} = 2$
Let $d = \min\{d' \mid m \leq \binom{d'}{h}\} = \min\{d' \mid 5 \leq \binom{d'}{3}\} = 5$

These are the number of remaining columns where ALG is forced to answer $a_j = 1$ (for d_1, not counting the current column).

ASG is AOC-Complete, cont.

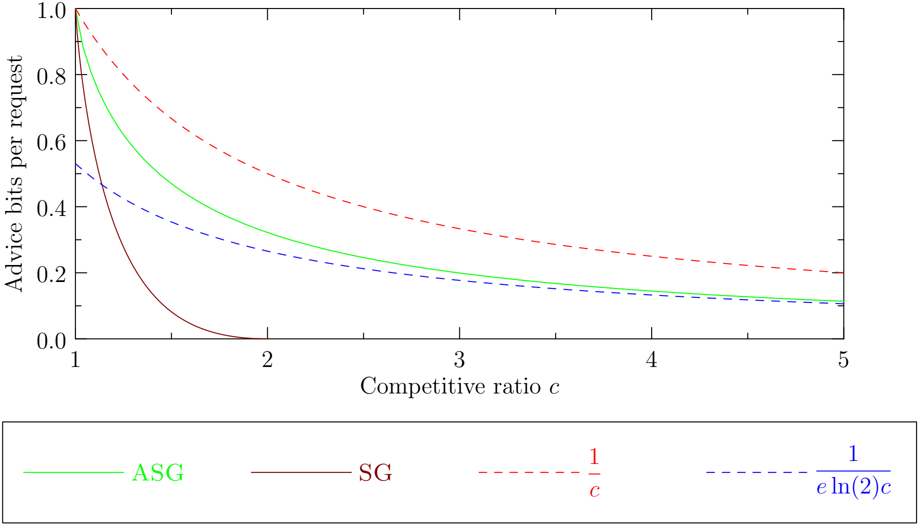
Using this adversary, the total number of columns (indices) where $\textsc{Alg}$ needs to answer 1 does not decrease.

ASG is AOC-Complete, cont.

Using this adversary, the total number of columns (indices) where $\textsc{Alg}$ needs to answer 1 does not decrease.

Let $L(m,h)$ denote the minimum cost Adversary can force.
$L(m,h) \geq \min\{d \mid m \leq \binom{d}{h}\}$.
Initially, $m > \binom{\lfloor ct \rfloor}{t}$ and $h = t$, so the minimum cost is at least $\lfloor ct \rfloor + 1$.
Contradiction

Advice bits needed to obtain competitive ratio c



Vertex Cover is AOC-Complete

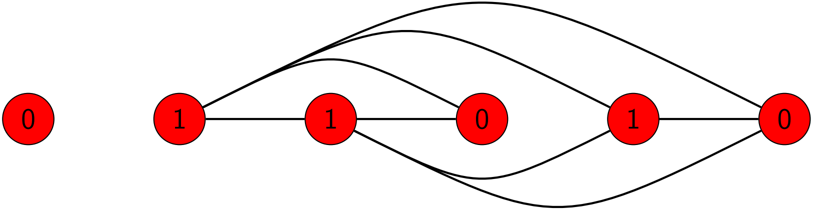
Recall that Vertex Cover is in AOC.
To show completeness, we reduce from Asymmetric String Guessing:

$$x = \langle x_1, x_2, \ldots, x_n \rangle \longrightarrow \begin{array}{rcl} V & = & \{v_1, v_2, \ldots, v_n\}, \\ E & = & \{(v_i, v_j) \mid x_i = 1 \text{ and } i < j\}. \end{array}$$

Vertex Cover is AOC-Complete

Recall that Vertex Cover is in AOC.
To show completeness, we reduce from Asymmetric String Guessing:

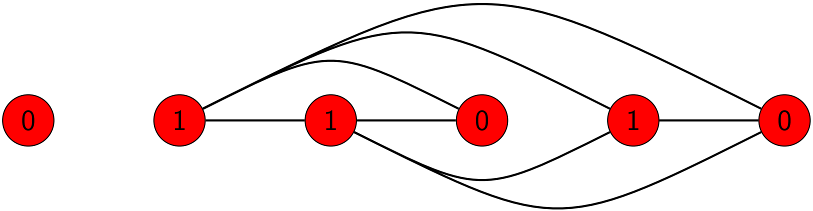
$$x = \langle x_1, x_2, \dots, x_n \rangle \longrightarrow \begin{array}{rcl} V & = & \{v_1, v_2, \dots, v_n\}, \\ E & = & \{(v_i, v_j) \mid x_i = 1 \text{ and } i < j\}. \end{array}$$



Vertex Cover is AOC-Complete, cont.

- Suppose we have a c-competitive algorithm, ALG, and an oracle, O, for Vertex Cover.
- Construct a c-competitive algorithm, ALG', and an oracle, O', for ASG.

$$x = \langle x_1, x_2, \ldots, x_n \rangle \longrightarrow \begin{array}{rcl} V & = & \{v_1, v_2, \ldots, v_n\}, \\ E & = & \{(v_i, v_j) \mid x_i = 1 \text{ and } i < j\}. \end{array}$$

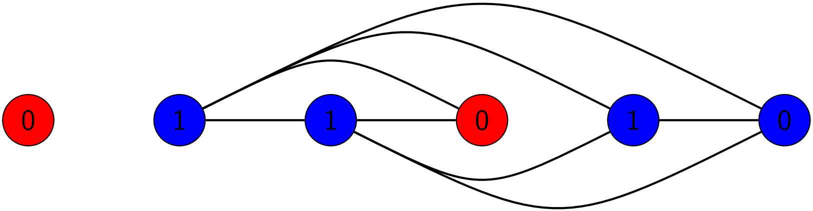


Vertex Cover is AOC-Complete, cont.

If ALG accepts all 1-vertices, ALG' answers 1 iff ALG accepts.

Since ALG is c-competitive, ALG' is too.

$$x = \langle x_1, x_2, \ldots, x_n \rangle \longrightarrow \begin{array}{rcl} V & = & \{v_1, v_2, \ldots, v_n\}, \\ E & = & \{(v_i, v_j) \mid x_i = 1 \text{ and } i < j\}. \end{array}$$



ALG' answers $\langle a_1, a_2, a_3, a_4, a_5, a_6 \rangle = \langle 0, 1, 1, 0, 1, 1 \rangle$.

Vertex Cover is AOC-Complete, cont.

If ALG rejects some 1-vertex, let v_i be the first it rejects.

Vertex Cover is AOC-Complete, cont.

If ALG rejects some 1-vertex, let v_i be the first it rejects.

Oracle, O', specifies index i and index j of some 0-vertex accepted.

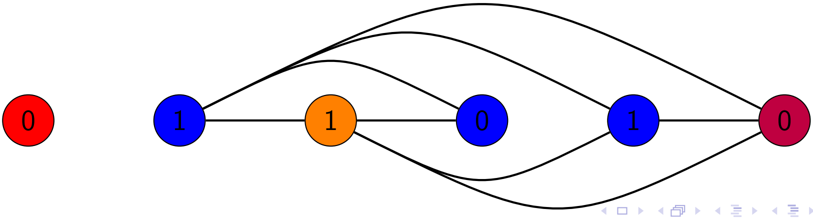
Vertex Cover is AOC-Complete, cont.

If ALG rejects some 1-vertex, let v_i be the first it rejects.

Oracle, O', specifies index i and index j of some 0-vertex accepted.

For index i, ALG' answers 1.
For index j, ALG' answers 0.
For all other indices, ALG' answers 1 iff ALG accepts.

ALG' answers 1 as many times as ALG accepts.
Since ALG is c-competitive, ALG' is too.



AOC-Complete problems

- Examples of AOC-Complete problems:
  - Vertex Cover
  - Dominating Set
  - Independent Set
  - Disjoint Path Allocation
  - Set Cover
- There are problems in AOC, which are not complete:
  - Unit-Weight Knapsack
  - Maximum Matching

Comparison with previous results

- Independent Set
  [Halldórsson,Iwama,Miyazaki,Taketomi, 2009]
  - Upper bound n/c
  - Lower bound $n/2c$
- Disjoint Path Allocation
  [Böckenhauer,Komm,Královič,Královič,Mömke, 2009]
  - Upper bound of $O\left(\frac{n \log_2 c}{c}\right)$
  - Lower bound of $n/2c$
- All other problems: No previous results
- All ASG and AOC results
  [B.,Favrholdt,Kudahl,Mikkelsen, 2017]

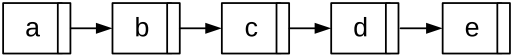
Section 4

Randomization and list access

List Accessing Problem

- Requests: for items in a list.
- Cost of accessing an item in index *i* is *i*.

$$< \mathbf{d\ b\ b\ d\ c\ a\ c} >$$

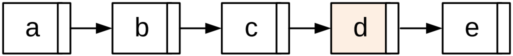


List Accessing Problem

- Requests: for items in a list.
- Cost of accessing an item in index i is i.

$$< \mathbf{d}\ \mathbf{b}\ \mathbf{b}\ \mathbf{d}\ \mathbf{c}\ \mathbf{a}\ \mathbf{c} >$$

cost: 4

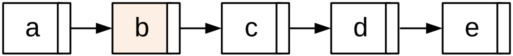


List Accessing Problem

- Requests: for items in a list.
- Cost of accessing an item in index i is i.

$$< d\ b\ b\ d\ c\ a\ c >$$

cost: $4 + 2$

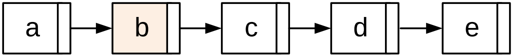


List Accessing Problem

- Requests: for items in a list.
- Cost of accessing an item in index i is i.

$< \mathbf{d\ b\ b\ d\ c\ a\ c} >$

cost: 4+2+2

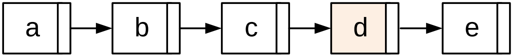


List Accessing Problem

- Requests: for items in a list.
- Cost of accessing an item in index *i* is *i*.

< d b b d c a c >

cost: 4+2+2+4

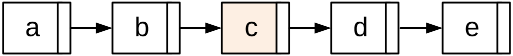


List Accessing Problem

- Requests: for items in a list.
- Cost of accessing an item in index *i* is *i*.

$< \mathbf{d}\ \mathbf{b}\ \mathbf{b}\ \mathbf{d}\ \mathbf{c}\ \mathbf{a}\ \mathbf{c} >$

cost: 4+2+2+4+3

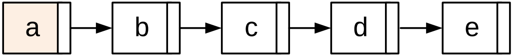


List Accessing Problem

- Requests: for items in a list.
- Cost of accessing an item in index *i* is *i*.

$$< \mathbf{d}\ \mathbf{b}\ \mathbf{b}\ \mathbf{d}\ \mathbf{c}\ \mathbf{a}\ \mathbf{c} >$$

cost: $4+2+2+4+3+1$

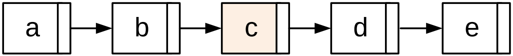


List Accessing Problem

- Requests: for items in a list.
- Cost of accessing an item in index i is i.

$$< \mathbf{d\ b\ b\ d\ c\ a\ c} >$$

$$\text{cost:} \quad 4{+}2{+}2{+}4{+}3{+}1{+}3 = 19$$



Self-Adjusting Lists

- Update the list to adjust it to the patterns in the list.

Self-Adjusting Lists

- Update the list to adjust it to the patterns in the list.
  - Free exchanges: Move a requested item closer to the front. No cost.

Self-Adjusting Lists

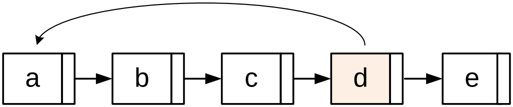
- Update the list to adjust it to the patterns in the list.
  - Free exchanges: Move a requested item closer to the front. No cost.
  - Paid exchanges: Swap positions of two consecutive items. Cost 1.

Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

$$< d\ b\ b\ d\ c\ a\ c >$$

cost: 4

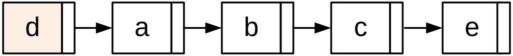


Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

$$< \color{red}{d}\ b\ b\ d\ c\ a\ c >$$

cost: 4

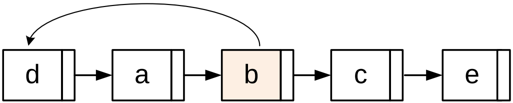


Move-To-Front (MTF)

- After each access, move the requested item to the front.
    - It only uses free exchanges.

$<$ d **b** b d c a c $>$

cost: 4+3

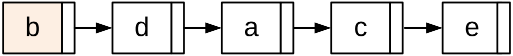


Move-To-Front (MTF)

- After each access, move the requested item to the front.
    - It only uses free exchanges.

$<$ d b b d c a c $>$

cost: 4+3

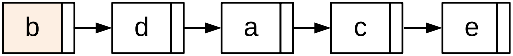


Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

< d b b d c a c >

cost: 4+3+1

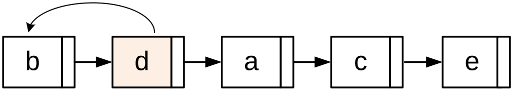


Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

$$< d\ b\ b\ d\ c\ a\ c >$$

cost: 4+3+1+2

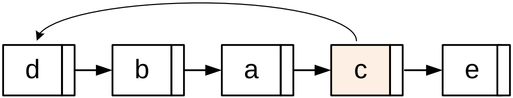


Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

$<$ d b b d c a c $>$

cost: 4+3+1+2+4

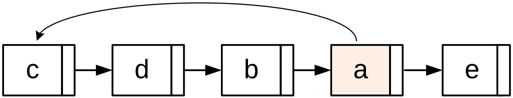


Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

$$< \mathbf{d}\ \mathbf{b}\ \mathbf{b}\ \mathbf{d}\ \mathbf{c}\ \mathbf{a}\ \mathbf{c} >$$

cost: 4+3+1+2+4+4

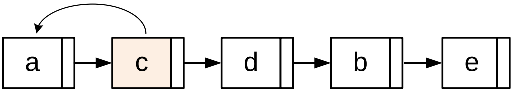


Move-To-Front (MTF)

- After each access, move the requested item to the front.
  - It only uses free exchanges.

< d b b d c a c >

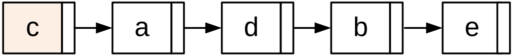
cost: 4+3+1+2+4+4+2



- After each access, move the requested item to the front.
  - It only uses free exchanges.

$$\langle\ d\ b\ b\ d\ c\ a\ c\ \rangle$$

cost: 4+3+1+2+4+4+2



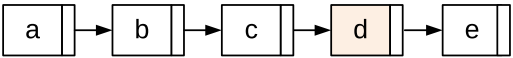
- Total cost: 20.

TIMESTAMP (TS)

- After an access to x,
  move x to the front of the first item y
  which has been requested at most once since the last access to x.
  - Do nothing if such an item y does not exist.
  - It only uses free exchanges.

$< d\ b\ b\ d\ c\ a\ c >$

cost: 4

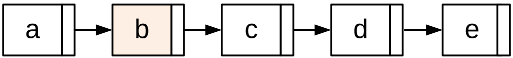


TIMESTAMP (TS)

- After an access to x,
  move x to the front of the first item y
  which has been requested at most once since the last access to x.
    - Do nothing if such an item y does not exist.
    - It only uses free exchanges.

< d **b** b d c a c >

cost: 4+2

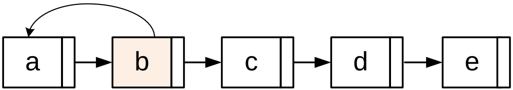


TIMESTAMP (TS)

- After an access to x,
  move x to the front of the first item y
  which has been requested at most once since the last access to x.
  - Do nothing if such an item y does not exist.
  - It only uses free exchanges.

$< \mathrm{d\ b\ b\ d\ c\ a\ c} >$

cost: 4+2+2

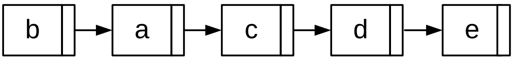


TIMESTAMP (TS)

- After an access to x,
  move x to the front of the first item y
  which has been requested at most once since the last access to x.
  - Do nothing if such an item y does not exist.
  - It only uses free exchanges.

< d b b d c a c >

cost: 4+2+2

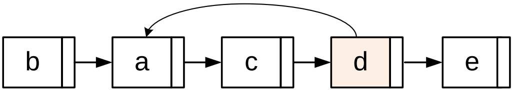


TIMESTAMP (TS)

- After an access to x,
  move x to the front of the first item y
  which has been requested at most once since the last access to x.
  - Do nothing if such an item y does not exist.
  - It only uses free exchanges.

$$< \mathrm{d\ b\ b\ } \color{red}{\mathrm{d}} \mathrm{\ c\ a\ c} >$$

cost: 4+2+2+4

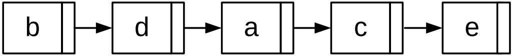


TIMESTAMP (TS)

- After an access to x,
  move x to the front of the first item y
  which has been requested at most once since the last access to x.
  - Do nothing if such an item y does not exist.
  - It only uses free exchanges.

< d b b d c a c >

cost: 4+2+2+4

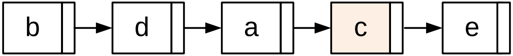


TIMESTAMP (TS)

- After an access to x,
  move x to the front of the first item y
  which has been requested at most once since the last access to x.
  - Do nothing if such an item y does not exist.
  - It only uses free exchanges.

< d b b d c a c >

cost: 4+2+2+4+4

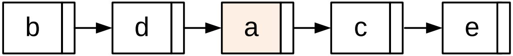


TIMESTAMP (TS)

- After an access to x,
  move x to the front of the first item y
  which has been requested at most once since the last access to x.
    - Do nothing if such an item y does not exist.
    - It only uses free exchanges.

$< \mathbf{d\ b\ b\ d\ c\ a\ c} >$

cost: 4+2+2+4+4+3

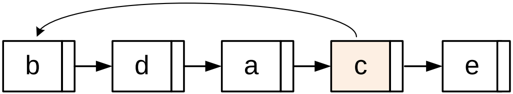


TIMESTAMP (TS)

- After an access to x,
  move x to the front of the first item y
  which has been requested at most once since the last access to x.
  - Do nothing if such an item y does not exist.
  - It only uses free exchanges.

$< \mathrm{d\ b\ b\ d\ c\ a\ c} >$

cost: 4+2+2+4+4+3+4

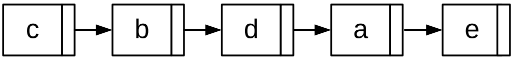


TIMESTAMP (TS)

- After an access to x,
  move x to the front of the first item y
  which has been requested at most once since the last access to x.
  - Do nothing if such an item y does not exist.
  - It only uses free exchanges.

$$< \mathbf{d\ b\ b\ d\ c\ a\ c} >$$

cost: 4+2+2+4+4+3+4



- Total cost: 23.

Competitive Analysis

- Competitive ratio of MTF for a list of length l is $2 - \frac{2}{l+1}$ [Sleator, Tarjan, 1985; Irani, 1991].
- Competitive ratio of TS is $2 - \frac{1}{l}$ [Albers, 1994].
- No deterministic online algorithm can have a competitive ratio better than $2 - \frac{2}{l+1}$ [Irani, 1991].

Breaking the lower bound

[B.,Kamali,Larsen,López-Ortiz, 2016]

- How many bits of advice are sufficient to perform strictly better than any online algorithm?
  - We show 2 bits of advice are sufficient.

Breaking the lower bound

[B.,Kamali,Larsen,López-Ortiz, 2016]

- How many bits of advice are sufficient to perform strictly better than any online algorithm?
  - We show 2 bits of advice are sufficient.
- We consider three classical algorithms and show that for any sequence, at least one of them has a performance ratio of at most $1.\bar{6}$:
  - TS (TIMESTAMP)
  - MTFE (Move-To-Front on Even accesses)
  - MTFO (Move-To-Front on Odd accesses)

MTF-every-other-access

MTFE and MTFO are the MTF-every-other-access (MTF2) algorithms.

MTF-every-other-access

MTFE and MTFO are the MTF-every-other-access (MTF2) algorithms.

What is the competitive ratio of MTF2?

MTF-every-other-access

MTFE and MTFO are the MTF-every-other-access (MTF2) algorithms.

What is the competitive ratio of MTF2?

Theorem

MTF2 *algorithms are 2.5-competitive.*

In an unpublished manuscript, 2003, Ansgar Grüne proves a lower bound of $\frac{7}{3}$ and claims an upper bound of 2.5.
We confirm that claim and raise the lower bound.

MTF-every-other-access

MTFE and MTFO are the MTF-every-other-access (MTF2) algorithms.

What is the competitive ratio of MTF2?

Theorem

MTF2 *algorithms are 2.5-competitive.*

In an unpublished manuscript, 2003, Ansgar Grüne proves a lower bound of $\frac{7}{3}$ and claims an upper bound of 2.5.
We confirm that claim and raise the lower bound.

Consider a list of ℓ items, initially ordered as $[a_1, a_2, \ldots, a_\ell]$.
Consider the following sequence of requests:

$$\sigma_m = \left\langle (a_1, a_2, \ldots, a_\ell, a_1^3, a_2^3, \ldots, a_\ell^3, a_\ell, a_{\ell-1}, \ldots, a_1, a_\ell^3, a_{\ell-1}^3, \ldots, a_1^3)^m \right\rangle .$$

We show that asymptotically, $\lim_{m\to\infty} \text{MTFO}(\sigma_m) = 2.5\text{OPT}(\sigma_m)$.

Breaking the lower bound

- A variety of techniques are used.

Breaking the lower bound

- A variety of techniques are used.
- Partial cost model:
  - The cost of accessing position i is $i - 1$.
  - The upper bounds hold in the full cost model.

Breaking the lower bound

- A variety of techniques are used.
- Partial cost model:
  - The cost of accessing position i is $i - 1$.
  - The upper bounds hold in the full cost model.
- These algorithms have pair-wise property:
  - The relative order of two items in the lists maintained by the algorithms only depends on the requests to those items.
  - To prove an upper-bound for competitive ratio, it is sufficient to study the algorithms for lists of length 2.

Breaking the lower bound

- A variety of techniques are used.
- Partial cost model:
  - The cost of accessing position i is $i - 1$.
  - The upper bounds hold in the full cost model.
- These algorithms have pair-wise property:
  - The relative order of two items in the lists maintained by the algorithms only depends on the requests to those items.
  - To prove an upper-bound for competitive ratio, it is sufficient to study the algorithms for lists of length 2.
- Phase-partitioning technique:
  - Compare the costs of the algorithms with $\textsc{Opt}$ for each projected sequence in *phases*.
  - Each phase ends with two consecutive requests to the same item.
  - We need to have the same phases for all algorithms.

Phase-partitioning technique:

- Compare the costs of the algorithms with OPT for each projected sequence in *phases*.
- Each phase ends with two consecutive requests to the same item.

Breaking the lower bound

Phase-partitioning technique:

- Compare the costs of the algorithms with $\textsc{Opt}$ for each projected sequence in *phases*.
- Each phase ends with two consecutive requests to the same item.

Type 1 phase: At start: $L = [x, y]$.
$\textsc{AlgMin}(\sigma) = \min\{\textsc{Mtfo}(\sigma), \textsc{Mtfe})(\sigma)\}$
$\textsc{AlgMax}(\sigma) = \max\{\textsc{Mtfo}(\sigma), \textsc{Mtfe})(\sigma)\}$
The costs are in the partial cost model.

Phase	$\textsc{AlgMin}$	$\textsc{AlgMax}$	TS	Sum ($\textsc{AlgMin}$ + $\textsc{AlgMax}$ + TS)	$\textsc{Opt}'$	$\frac{\text{Sum}}{\textsc{Opt}'}$
$x^j yy$	1	2	2	5	1	5
$x^j(yx)^{2i}yy$	$\leq 3i+1$	$\leq 3i+2$	$2 \times 2i = 4i$	$\leq 10i+3$	$2i+1$	< 5
$x^j(yx)^{2i-2}yxyy$	$\leq 3(i-1)+1$ $+\textsc{AlgMin}(\langle xyy\rangle)$	$\leq 3(i-1)+1$ $+\textsc{AlgMax}(\langle xyy\rangle)$	$2 \times (2i-1)$ $= 4i-2$	$\leq 6(i-1)+2+4$ $+(4i-2) = 10i-2$	$2i$	< 5
$x^j(yx)^{2i}x$	$\leq 3i$	$\leq 3i+1$	$2 \times 2i - 1$ $= 4i-1$	$\leq (6i+1)+(4i-1)$ $= 10i$	$2i$	≤ 5
$x^j(yx)^{2i-2}yxx$	$\leq 3(i-1)$ $+\textsc{AlgMin}(\langle yxx\rangle)$	$\leq 3(i-1)$ $+\textsc{AlgMax}(\langle yxx\rangle)$	$2 \times (2i-1)-1$ $= 4i-3$	$\leq 6(i-1)+4$ $+(4i-3) = 10i-5$	$2i-1$	≤ 5

Breaking the lower bound

Theorem

For any sequence σ *we have* $\mathrm{TS}(\sigma) + \textsc{Mtfo}(\sigma) + \textsc{Mtfe}(\sigma) \leq 5\,\textsc{Opt}(\sigma)$

Breaking the lower bound

Theorem

For any sequence σ we have $\mathrm{TS}(\sigma) + \textsc{Mtfo}(\sigma) + \textsc{Mtfe}(\sigma) \leq 5\,\textsc{Opt}(\sigma)$

Since at least one must do as well as the average:

Breaking the lower bound

Theorem

For any sequence σ we have $\mathrm{TS}(\sigma) + \textsc{Mtfo}(\sigma) + \textsc{Mtfe}(\sigma) \leq 5\,\textsc{Opt}(\sigma)$

Since at least one must do as well as the average:

Theorem

There is an algorithm that receives two bits of advice and achieves a competitive ratio of $1.\overline{6}$.

Breaking the lower bound

Theorem

For any sequence σ we have $\mathrm{TS}(\sigma) + \mathrm{MTFO}(\sigma) + \mathrm{MTFE}(\sigma) \leq 5\,\mathrm{OPT}(\sigma)$

Since at least one must do as well as the average:

Theorem

There is an algorithm that receives two bits of advice and achieves a competitive ratio of $1.\overline{6}$.

- This can be regarded as the best existing (deterministic) approximation algorithm for the offline problem.
- If list access is used for file compression, adding two bits at the beginning of the file can guarantee better compression.

Randomization and advice

- The randomized algorithm which chooses each of TS, MTFE, and MTFO with probability $1/3$ is $1.\bar{6}$-competitive.
- If the number of algorithms was 2^k for some k,
  there would be a randomized algorithm using only k bits of randomness.
- A c-competitive randomized algorithm using $b(n)$ bits of randomness implies a c-competitive algorithm using at most $b(n)$ bits of advice.
- If there is provably no algorithm which is c-competitive using only $b(n)$ bits of advice, then there is no c-competitive randomized algorithm using only $b(n)$ bits of randomness.

Randomization and advice

- A c-competitive randomized algorithm using $b(n)$ bits of randomness implies a c-competitive algorithm using at most $b(n)$ bits of advice.
- [Böckenhauer,Komm,Královic,Královic, 2011]
  A c-competitive randomized algorithm
  implies a $(c+\epsilon)$-competitive algorithm using at most

$$b(n) = \lceil \log n \rceil + 2\lceil \log \lceil \log n \rceil \rceil + \log \left(\left\lfloor \frac{\log(m(n))}{\log(1+\epsilon)} \right\rfloor + 1 \right)$$

  bits of advice.

Randomization and advice

- A c-competitive randomized algorithm using $b(n)$ bits of randomness implies a c-competitive algorithm using at most $b(n)$ bits of advice.
- [Böckenhauer,Komm,Královic,Královic, 2011]
  A c-competitive randomized algorithm
  implies a $(c+\epsilon)$-competitive algorithm using at most

  $$b(n) = \lceil \log n \rceil + 2\lceil \log \lceil \log n \rceil \rceil + \log \left(\left\lfloor \frac{\log(m(n))}{\log(1+\epsilon)} \right\rfloor + 1 \right)$$

  bits of advice.
  The number of different inputs of length n is $m(n)$.
  Holds for any $\epsilon > 0$.

Randomization and advice

- A c-competitive randomized algorithm using $b(n)$ bits of randomness implies a c-competitive algorithm using at most $b(n)$ bits of advice.
- [Böckenhauer,Komm,Královic,Královic, 2011]
  A c-competitive randomized algorithm
  implies a $(c+\epsilon)$-competitive algorithm using at most

  $$b(n) = \lceil \log n \rceil + 2\lceil \log \lceil \log n \rceil \rceil + \log \left(\left\lfloor \frac{\log(m(n))}{\log(1+\epsilon)} \right\rfloor + 1 \right)$$

  bits of advice.
  The number of different inputs of length n is $m(n)$.
  Holds for any $\epsilon > 0$.
- Now proving that any algorithm which is c-competitive requires enough advice, shows there is no c'-competitive randomized algorithm.

Randomization and advice

The randomized k-server conjecture claims there is a randomized algorithm for the k-server problem which is $\Theta(\log k)$-competitive.

Randomization and advice

The randomized k-server conjecture claims there is a randomized algorithm for the k-server problem which is $\Theta(\log k)$-competitive.

[Böckenhauer,Komm,Královic,Královic, 2011]
If every online algorithm with advice for the k-server problem needs to use at least $\omega(\log n)$ advice bits to be $O(\log k)$-competitive, the randomized k-server conjecture does not hold.

Section 5

Concluding remarks

Concluding remarks

- Lower bounds on advice complexity can:
  - Rule out possibilities for certain semi-online approaches.
  - Rule out possibilities for randomized approaches.

Concluding remarks

- Lower bounds on advice complexity can:
  - Rule out possibilities for certain semi-online approaches.
  - Rule out possibilities for randomized approaches.
- Upper bounds can be either practical or purely theoretical.

Concluding remarks

- Lower bounds on advice complexity can:
  - Rule out possibilities for certain semi-online approaches.
  - Rule out possibilities for randomized approaches.
- Upper bounds can be either practical or purely theoretical.
- There is a start of a complexity theory, based on string guessing problems.

Thank you for your attention.