COMP 7720 - Online Algorithms

$k$-Server Problem & Advice

Shahin Kamali

Lecture 13 - Oct. 18, 2017

University of Manitoba
Review & Plan
Today’s objectives

- Review of work-function algorithm
- Randomized algorithms for $k$-server
- Techniques for advice lower bounds
Work-function Review
Consider a fixed initial configuration $C_0$, input sequence $\sigma$.

For a given configuration $X$ and time $t$ work-function is defined as the cost of Opt for serving the first $t$ requests of $\sigma$ and ending up at configuration $X$

$$w_t(X) = \min_{Z} \{w_{t-1}(Z) + d(X, Z)\} \text{ so that } x_t \in Z;$$

$$w_0(X) = d(X, C_0)$$

Work-function can be computer in an online manner using dynamic programming.
Work Function Algorithm

Assume we want to serve the $t'$th request and we are at configuration $C$.

- Values of $w_{t-1}(X)$ are computed when serving previous request

Step I: compute the values of $w_t(X)$ using the recursive formula

Step II: for any configuration $X$, find $w_t(X) + d(C, X)$.

Step III: move servers to form the configuration which minimizes this value
Assume $\sigma = \langle B, A, B, A, B, A, \ldots \rangle$

- Current config.: $(A, D)$, Current request: $B$

<table>
<thead>
<tr>
<th>confs\input</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(A, B)$</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(A, C)$</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$(A, D)$</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(B, C)$</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$(B, D)$</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$(C, D)$</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Assume $\sigma = \langle B \ A \ B \ A \ B \ A \ldots \rangle$

- Current config.: $(A, D)$, Current request: $B$

Step I: calculate new work-function values (recall $w_t(X) = \min_{Z} \{w_{t-1}(Z) + d(X, Z)\}$ so that $x_t \in Z$)

Step II: find config. with $\min_{X} \{w_1(X) + d(C, X)\}$

Step III: Move servers to config. $(B, D)$!
Assume $\sigma = \langle B \ A \ B \ A \ B \ A \ldots \rangle$

- Current config.: $(A, D)$, Current request: $B$

Step I: calculate new work-function values (recall $w_t(X) = \min_Z \{w_{t-1}(Z) + d(X, Z)\}$ so that $x_t \in Z$)

- e.g., $w(A, B) = \min_{Z \in \{(A, B), (B, C), (B, D)\}} \langle w_0(Z) + d((A, B), Z) \rangle$

  $$\begin{array}{c}
  \text{confs} \\
  \text{\input{}} \\
  0 \ 1 \ 2 \ 3 \ 4 \\
  (A, B) \ 2 \ 2 \\
  (A, C) \ 1 \\
  (A, D) \ 0 \\
  (B, C) \ 2 \\
  (B, D) \ 1 \\
  (C, D) \ 2 \\
  \end{array}$$

Step II: find config. with $\min X \in \{(A, B), (A, C), (A, D), (B, C), (B, D), (C, D)\} w_1(X) + d((A, D), X)$

$$\begin{array}{c}
\min \{2 + 0, 2 + 2, 1 + 3\} = 2
\end{array}$$

Step III: Move servers to config. $(B, D)$!
Assume $\sigma = \langle B \ A \ B \ A \ B \ A \ldots \rangle$

- Current config.: $(A, D)$, Current request: $B$

Step I: calculate new work-function values (recall $w_t(X) = \min_Z \{w_{t-1}(Z) + d(X, Z)\}$ so that $x_t \in Z$)

- e.g., $w(A, B) = \min_{Z \in \{(A,B),(B,C),(B,D)\}} \langle w_0(Z) + d((A, B), Z) \rangle$
  
  
  $= \min \{2 + 0, 2 + 2, 1 + 3\} = 2$

<table>
<thead>
<tr>
<th>confs \ input</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(A, B)$</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(A, C)$</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(A, D)$</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(B, C)$</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(B, D)$</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(C, D)$</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assume $\sigma = \langle B \ A \ B \ A \ B \ A \ldots \rangle$

- Current config.: $(A, D)$, Current request: $B$

Step II: find config. with $\min w_t(X) + d(C, X)$

<table>
<thead>
<tr>
<th>confs \ input</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>0 1 2 3 4</td>
</tr>
<tr>
<td>(A,B)</td>
</tr>
<tr>
<td>2 2</td>
</tr>
<tr>
<td>(A,C)</td>
</tr>
<tr>
<td>1 3</td>
</tr>
<tr>
<td>(A,D)</td>
</tr>
<tr>
<td>0 2</td>
</tr>
<tr>
<td>(B,C)</td>
</tr>
<tr>
<td>2 2</td>
</tr>
<tr>
<td>(B,D)</td>
</tr>
<tr>
<td>1 1</td>
</tr>
<tr>
<td>(C,D)</td>
</tr>
<tr>
<td>2 2</td>
</tr>
</tbody>
</table>
Work Function Algorithm Examples

Assume $\sigma = \langle B \ A \ B \ A \ B \ A \ldots \rangle$

- Current config.: $(A, D)$, Current request: $B$

Step II: find config. with $\min w_t(X) + d(C, X)$

$$\min_{X \in \{(A,B),(A,C),(A,D),(B,C),(B,D),(C,D)\}} w_1(X) + d((A,D),X)$$

$$= \arg \min \{2 + 2, 3 + 1, 2 + 0, 2 + 2, 1 + 1, 2 + 2\} \rightarrow (B, D)$$

<table>
<thead>
<tr>
<th>configs</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(A,B)$</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(A,C)$</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(A,D)$</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(B,C)$</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(B,D)$</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(C,D)$</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assume $\sigma = \langle B, A, B, A, B, A \ldots \rangle$

- Current config.: $(A, D)$, Current request: $B$

Step II: find config. with min $w_t(X) + d(C, X)$

$\min_{X \in \{(A,B),(A,C),(A,D),(B,C),(B,D),(C,D)\}} w_1(X) + d((A, D), X) = \arg\min\{2 + 2, 3 + 1, 2 + 0, 2 + 2, 1 + 1, 2 + 2\} \rightarrow (B, D)$

Step III: Move servers to config. $(B, D)$!
Assume $\sigma = \langle B \ A \ B \ A \ B \ A \ldots \rangle$

- Current config.: $(B, D)$, Current request: $A$

**Step I:** calculate new work-function values ($\text{recall } w_t(X) = \min Z \{ w_t - 1(Z) + d(X, Z) \}$ so that $x_t \in Z$)

$w_t(A, B) = \min Z \{ 2 + 0, 3 + 1, 2 + 2 \} = 2$

**Step II:** find config. with $\min w_t(X) + d(C, X)$

$\min X \{ 2 + 3, 3 + 2, 2 + 1, 4 + 1, 3 + 0, 4 + 1 \} \rightarrow (A, D)$

**Step III:** Move servers to config. $(A, D)$!
Work Function Algorithm Examples

Assume \( \sigma = \langle B \ A \ B \ A \ B \ A \ldots \rangle \)

- Current config.: \((B, D)\), Current request: \(A\)

Step I: calculate new work-function values (recall \( w_t(X) = \min_Z \{ w_{t-1}(Z) + d(X, Z) \} \) so that \( x_t \in Z \))

\[
\begin{align*}
\text{confs} \times \text{input} & \quad 0 & 1 & 2 & 3 & 4 \\
(A,B) & 2 & 2 & 2 & & \\
(A,C) & 1 & 3 & 3 & & \\
(A,D) & 0 & 2 & 2 & & \\
(B,C) & 2 & 2 & 4 & & \\
(B,D) & 1 & 1 & 3 & & \\
(C,D) & 2 & 2 & 4 & & 
\end{align*}
\]
Review & Plan

Work Function Algorithm Examples

Assume $\sigma = \langle B\ A\ B\ A\ B\ A\ \ldots \rangle$

- Current config.: $(B, D)$, Current request: $A$

Step I: calculate new work-function values (recall $w_t(X) = \min_Z \{w_{t-1}(Z) + d(X, Z)\}$ so that $x_t \in Z$)

- e.g., $w(A, B) = \min_{Z \in \{(A, B), (A, C), (A, D)\}} \langle w_1(Z) + d((A, B), Z) \rangle$

$$= \min \{2 + 0, 3 + 1, 2 + 2\} = 2$$

<table>
<thead>
<tr>
<th>confs\input</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\uparrow$ (A, B)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\uparrow$ (A, C)</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\uparrow$ (A, D)</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B, C)</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B, D)</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(C, D)</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assume $\sigma = \langle B \ A \ B \ A \ B \ A \ldots \rangle$

- Current config.: $(B, D)$, Current request: $A$

Step I: calculate new work-function values (recall $w_t(X) = \min_Z \{w_{t-1}(Z) + d(X, Z)\}$ so that $x_t \in Z$)

- e.g., $w(A, B) = \min_{Z \in \{(A, B), (A, C), (A, D)\}} \langle w_1(Z) + d((A, B), Z) \rangle$
  
  $= \min \{2 + 0, 3 + 1, 2 + 2\} = 2$

### Table

<table>
<thead>
<tr>
<th>confs</th>
<th>input</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A,B)</td>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(A,C)</td>
<td></td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(A,D)</td>
<td></td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B,C)</td>
<td></td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B,D)</td>
<td></td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(C,D)</td>
<td></td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assume $\sigma = \langle B\ A\ B\ A\ B\ A\ \ldots \rangle$

- Current config.: $(B, D)$, Current request: $A$

Step II: find config. with $\min w_t(X) + d(C, X)$

$$\min_{X \in \{(A,B),(A,C),(A,D),(B,C),(B,D),(C,D)\}} \langle w_2(X) + d((B, D), X) \rangle$$

$$= \min \{2 + 3, 3 + 2, 2 + 1, 4 + 1, 3 + 0, 4 + 1\} \rightarrow (A, D)$$
Assume $\sigma = \langle B \ A \ B \ A \ B \ A \ ... \rangle$

- Current config.: $(B, D)$, Current request: $A$

Step II: find config. with $\min w_t(X) + d(C, X)$

- $\min_{X \in \{(A, B), (A, C), (A, D), (B, C), (B, D), (C, D)\}} \langle w_2(X) + d((B, D), X) \rangle$
  
  $= \min\{2 + 3, 3 + 2, 2 + 1, 4 + 1, 3 + 0, 4 + 1\} \rightarrow (A, D)$

Step III: Move servers to config. $(A, D)$!
Assume $\sigma = \langle B, A, B, A, B, A, \ldots \rangle$

- Current config.: $(B, D)$, Current request: $A$

Step II: find config. with $\min w_t(X) + d(C, X)$

- $\min_{X \in \{(A,B), (A,C), (A,D), (B,C), (B,D), (C,D)\}} \langle w_2(X) + d((B, D), X) \rangle$
  $= \min\{2 + 3, 3 + 2, 2 + 1, 4 + 1, 3 + 0, 4 + 1\} \rightarrow (A, D)$

Step III: Move servers to config. $(A, D)$!
General graphs

- Work-function algorithm:
General graphs

- Work-function algorithm:
  - has a competitive ratio of $2k - 1$ competitive for general metrics.
Review & Plan

General graphs

- Work-function algorithm:
  - has a competitive ratio of $2k - 1$ competitive for general metrics.
  - $k$-competitive for line, star, and graphs with $m \leq k + 2$.
  - Trees and general graphs?
General graphs

- **Work-function algorithm:**
  - has a competitive ratio of $2k - 1$ competitive for general metrics.
  - $k$-competitive for line, star, and graphs with $m \leq k + 2$.
  - Trees and general graphs?

- Work-function algorithm is conjectured to be $k$-competitive for any metric.
General graphs

- **Work-function algorithm:**
  - has a competitive ratio of $2k - 1$ competitive for general metrics.
  - $k$-competitive for line, star, and graphs with $m \leq k + 2$.
  - Trees and general graphs?

- **Work-function algorithm is conjectured to be $k$-competitive for any metric**
  - It might answer the $k$-server conjecture in affirmative (but we are not sure)
Work-function Framework

- Define a ‘configuration’ as the state of an algorithm
  - locations of servers or state of the linked-list (list update), etc.
Work-function Framework

- Define a ‘configuration’ as the state of an algorithm
  - locations of servers or state of the linked-list (list update), etc.
- Define the ‘distance’ between two configurations based on the cost model
  - distance moved by servers or number of paid exchanges to change the state of the list from one config. to another

Define the work function $w_t(X)$ as the cost of Opt for serving $t$ requests and ending up at config. $X$.

Maintain the work-function in an online manner.

Work-function algorithm: assume we are at configuration $C$; switch to a configuration $Y$ that minimizes $w_t(Y) + d(C, Y)$.
**Work-function Framework**

- Define a ‘configuration’ as the state of an algorithm
  - locations of servers or state of the linked-list (list update), etc.
- Define the ‘distance’ between two configurations based on the cost model
  - distance moved by servers or number of paid exchanges to change the state of the list from one config. to another
- Define the work function $w_t(X)$ as the cost of OPT for serving $t$ requests and ending up at config. $X$
  - Maintain the work-function in an online manner.
Define a ‘configuration’ as the state of an algorithm
- locations of servers or state of the linked-list (list update), etc.

Define the ‘distance’ between two configurations based on the cost model
- distance moved by servers or number of paid exchanges to change the state of the list from one config. to another

Define the work function $w_t(X)$ as the cost of $\text{OPT}$ for serving $t$ requests and ending up at config. $X$
- Maintain the work-function in an online manner.

Work-function algorithm: assume we are at configuration $C$; switch to a configuration $Y$ that minimizes $w_t(Y) + d(C, Y)$
k-Server & Randomization
Randomized algorithms

Theorem

*For any metric space, no algorithm can be better than $\log k$ competitive*
Randomized algorithms

Theorem

*For any metric space, no algorithm can be better than \( \log k \) competitive*

- Randomized \( k \)-server conjecture: For any metric space there is a randomized \( \log k \)-competitive algorithm
**Randomized algorithms**

**Theorem**

*For any metric space, no algorithm can be better than* \( \log k \) *competitive*

- Randomized \( k \)-server conjecture: For any metric space there is a randomized \( \log k \)-competitive algorithm
- Only verified for a class of ‘hierarchical binary trees’
Randomized algorithms

**Theorem**

*For any metric space, no algorithm can be better than* $\log k$ *competitive*

- Randomized $k$-server conjecture: For any metric space there is a randomized $\log k$-competitive algorithm.
- Only verified for a class of ‘hierarchical binary trees’.
- For general graphs, there is a $O(\log^3 m \log^2 k)$-competitive algorithm [the best result Until October 2018].
- For general graphs, there is a $O(\log^c(k))$-competitive algorithm [FOCS 2018] ($c$ is constant, i.e., the algorithm is poly-log-competitive).
$k$-Server & Advice
How many bits of advice are sufficient to achieve an optimal solution for $k$-server on a sequence of length $n$?
How many bits of advice are sufficient to achieve an optimal solution for $k$-server on a sequence of length $n$?

What is advice? For each request $x$ encode the server that a lazy optimal algorithm uses to serve $x$. The size of advice is $O(\log k)$ bits; so it is $n \cdot O(\log k) = O(n \log k)$. How does the algorithm work? For each request, it uses the server indicated by the advice for that request. Why is the algorithm optimal? It works exactly like the lazy optimal algorithm.
How many bits of advice are sufficient to achieve an optimal solution for $k$-server on a sequence of length $n$?

What is advice? For each request $x$ encode the server that a lazy optimal algorithm uses to serve $x$.

What is the size of advice? You can indicate a server using $O(\log k)$ bits; so it is $n \cdot O(\log k) = O(n \log k)$.
How many bits of advice are sufficient to achieve an optimal solution for $k$-server on a sequence of length $n$?

What is advice? For each request $x$, encode the server that a lazy optimal algorithm uses to serve $x$.

What is the size of advice? You can indicate a server using $O(\log k)$ bits; so it is $n \cdot O(\log k) = O(n \log k)$.

How does the algorithm work? For each request, it uses the server indicated by the advice for that request.
How many bits of advice are sufficient to achieve an optimal solution for $k$-server on a sequence of length $n$?

What is advice? For each request $x$ encode the server that a lazy optimal algorithm uses to serve $x$.

What is the size of advice? You can indicate a server using $O(\log k)$ bits; so it is $n \cdot O(\log k) = O(n \log k)$.

How does the algorithm work? For each request, it uses the server indicated by the advice for that request.

Why the algorithm is optimal? It works exactly like the lazy optimal algorithm.
How many bits of advice are sufficient to achieve an optimal solution when the input is a path?
How many bits of advice are sufficient to achieve an optimal solution when the input is a path?

What is advice? For each request, a lazy optimal uses the left or right server.

How does the algorithm work? For each request, it uses the server on the side indicated by the advice for that request.

Why is the algorithm optimal? It works exactly like the lazy optimal algorithm.
k-server & advice

- How many bits of advice are sufficient to achieve an optimal solution when the input is a path?
- What is advice? For each request $x$, a lazy optimal uses the left or right server.
- What is the size of advice? For each request, you can indicate the left/right server using 1 bit; so it is $n \cdot 1 = n$ bits.
How many bits of advice are sufficient to achieve an optimal solution when the input is a path?

What is advice? For each request $x$, a lazy optimal uses the left or right server.

What is the size of advice? For each request you can indicate the left/right server using 1 bits, so it is $n \cdot 1 = n$ bits.

How does the algorithm work? For each request, it uses the server on the side indicated by the advice for that request.
How many bits of advice are sufficient to achieve an optimal solution when the input is a path?

What is advice? For each request $x$ a lazy optimal uses the left or right server.

What is the size of advice? For each request you can indicate the left/right server using 1 bits; so it is $n \cdot 1 = n$ bits.

How does the algorithm work? For each request, it uses the server on the side indicated by the advice for that request.

Why the algorithm is optimal? It works exactly like the lazy optimal algorithm.
Can we achieve an optimal solution for paths using asymptotically less than $O(n)$ bits?

The answer is NO! To prove it, we use a general framework for devising lower bounds.
Can we achieve an optimal solution for paths using asymptotically less than $O(n)$ bits?

- The answer is NO!

To prove it, we use a general framework for devising lower bounds!
Binary Guessing Problem

- Assume an online sequence of binary bits such as \( \sigma = \langle 0100101 \ldots \rangle \)
- Before the next bit is revealed, an online algorithm ‘guesses’ whether it is ‘0’ or ‘1’.

Lemma

On an input of length \( m \), any deterministic algorithm that guesses correctly on more than \( \alpha m \) bits, for \( \frac{1}{2} < \alpha < 1 \), requires at least 
\[
(1 + (1 - \alpha) \log(1 - \alpha) + \alpha \log \alpha) \cdot m
\]
bits of advice.
Binary Guessing Problem

- Assume an online sequence of binary bits such as \( \sigma = \langle 0100101 \ldots \rangle \)
- Before the next bit is revealed, an online algorithm ‘guesses’ whether it is ‘0’ or ‘1’.
- If no advice, adversary makes algorithm wrongly guess all the time.
Binary Guessing Problem

- Assume an online sequence of binary bits such as $\sigma = \langle 0100101 \ldots \rangle$
- Before the next bit is revealed, an online algorithm ‘guesses’ whether it is ‘0’ or ‘1’.
- If no advice, adversary makes algorithm wrongly guess all the time.
- If there is one bit of advice, an algorithm can guess at least half of all bits correctly.
  - The bit indicates whether ‘0’s are more than ‘1’s; the algorithm always guesses the more frequent bit to be the next bit.
Binary Guessing Problem

- Assume an online sequence of binary bits such as $\sigma = \langle 0100101 \ldots \rangle$
- Before the next bit is revealed, an online algorithm ‘guesses’ whether it is ‘0’ or ‘1’.
- If no advice, adversary makes algorithm wrongly guess all the time.
- If there is one bit of advice, an algorithm can guess at least half of all bits correctly.
  - The bit indicates whether ‘0’s are more than ‘1’s; the algorithm always guesses the more frequent bit to be the next bit.
- To guess more than half correctly, $\Omega(n)$ bits of advice are required.

Lemma

On an input of length $m$, any deterministic algorithm that guesses correctly on more than $\alpha m$ bits, for $1/2 < \alpha < 1$, requires at least $(1 + (1 - \alpha) \log(1 - \alpha) + \alpha \log \alpha) \cdot m$ bits of advice
The binary guessing problem can be reduced to many online problems.

I.e., a linear number of bits of advice are required to achieve close-to-optimal solutions.

Let’s see an example in terms of k-server on line metric.
Consider a line graph formed by 5 nodes and assume $k = 2$ servers are initially located at nodes 2 and 4.
Consider a line graph formed by 5 nodes and assume $k = 2$ servers are initially located at nodes 2 and 4.

Assume the input is formed by phases of two types:

- Type 0: (3, 1, 3, 2, 4, 2, 4)
- Type 1: (3, 5, 3, 2, 4, 2, 4)

An optimal algorithm incurs a cost of 4 for each phase.

The online algorithm should 'guess' the type of the phase:
- A cost of 4 for a right guess.
- A cost of at least 6 for a wrong guess.
Consider a line graph formed by 5 nodes and assume \( k = 2 \) servers are initially located at nodes 2 and 4.

Assume the input is formed by phases of two types

- Type 0: \((3, 1, 3, 2, 4, 2, 4)\)
- Type 1: \((3, 5, 3, 2, 4, 2, 4)\)
Lower Bound for 2-server with Advice

Consider a line graph formed by 5 nodes and assume $k = 2$ servers are initially located at nodes 2 and 4.

Assume the input is formed by phases of two types

- Type 0: $(3, 1, 3, 2, 4, 2, 4)$
- Type 1: $(3, 5, 3, 2, 4, 2, 4)$

At the beginning of each phase servers are at 2 and 4

The last requests to 2, 4 in previous phase ensure that any reasonable algorithm has servers at 2, 4.
Consider a line graph formed by 5 nodes and assume $k = 2$ servers are initially located at nodes 2 and 4.

Assume the input is formed by phases of two types

- Type 0: $(3, 1, 3, 2, 4, 2, 4)$
- Type 1: $(3, 5, 3, 2, 4, 2, 4)$

An optimal algorithm incurs a cost of 4 for each phase.
Consider a line graph formed by 5 nodes and assume \( k = 2 \) servers are initially located at nodes 2 and 4.

Assume the input is formed by phases of two types

- Type 0: \((3,1,3,2,4,2,4)\)
- Type 1: \((3,5,3,2,4,2,4)\)

An optimal algorithm incurs a cost of 4 for each phase.

The online algorithm should ‘guess’ the type of the phase

- A cost of 4 for a right guess
- A cost of at least 6 for a wrong guess
Consider an input of $n = 7m$ requests formed by $m$ phases

- The cost of $\text{OPT}$ is at most $4m$

If the algorithm guesses half of phases correctly, its cost will be at least $\frac{m}{2} \cdot 4 + \frac{m}{2} \cdot 6 = 5m$. Guessing half items correctly → competitive ratio of at least $\frac{5}{4}$. Use the $k$-server algorithm for binary guessing. For each phase, if the algorithm moves server at 2 (resp. 4) for serving the first request at 3, guess the next bit (phase type) to be 0 (resp. 1).

We know that no binary guessing can guess more than half of phases correctly with $O(m) = O(n)$ advice → no $k$-server algorithm can be better than $\frac{5}{4}$ competitive.
Consider an input of $n = 7m$ requests formed by $m$ phases.

- The cost of $\text{Opt}$ is at most $4m$.
- If the algorithm guesses half of phases correctly, its cost will be at least $m/2 \cdot 4 + m/2 \cdot 6 = 5m$.
- Guessing half items correctly $\rightarrow$ competitive ratio of at least $5/4$.
Consider an input of $n = 7m$ requests formed by $m$ phases

- The cost of $\text{Opt}$ is at most $4m$

If the algorithm guesses half of phases correctly, its cost will be at least $m/2 \cdot 4 + m/2 \cdot 6 = 5m$.

- Guessing half items correctly $\rightarrow$ competitive ratio of at least $5/4$

Use the $k$-server algorithm for binary guessing

- For each phase, if the algorithm moves server at 2 (resp. 4) for serving the first request at 3, guess the next bit (phase type) to be 0 (resp. 1).
Consider an input of $n = 7m$ requests formed by $m$ phases.

- The cost of $\text{Opt}$ is at most $4m$.

- If the algorithm guesses half of phases correctly, its cost will be at least $m/2 \cdot 4 + m/2 \cdot 6 = 5m$.

- Guessing half items correctly $\rightarrow$ competitive ratio of at least $5/4$.

Use the $k$-server algorithm for binary guessing.

- For each phase, if the algorithm moves server at 2 (resp. 4) for serving the first request at 3, guess the next bit (phase type) to be 0 (resp. 1).

We know that no binary guessing can guess more than half of phases correctly with $O(m) = O(n)$ advice $\rightarrow$ no $k$-server algorithm can be better than $5/4$ competitive.
Theorem

In order to achieve any algorithm with competitive ratio less than $5/4$, $\Omega(n)$ bits of advice are required.
Theorem

In order to achieve any algorithm with competitive ratio less than $5/4$, $\Omega(n)$ bits of advice are required.

Recall that $O(n)$ bits of advice were sufficient to achieve an optimal solution!
Theorem

*In order to achieve any algorithm with competitive ratio less than 5/4, \( \Omega(n) \) bits of advice are required.*

- Recall that \( O(n) \) bits of advice were sufficient to achieve an optimal solution!
- We will see another example of this type of lower bound in the context of list updated