COMP 7720 - Online Algorithms

Paging and $k$-Server Problem

Shahin Kamali

Lecture 11 - Oct. 16, 2017

University of Manitoba
Review & Plan
Today’s objectives

- $k$-server problem
  - Paths & trees
  - Balancing algorithms
  - Offline algorithms
  - Work-function algorithm
$k$-Server Problem
Introduction

$k$-sever problem

We have a metric space of size $m$. Let $k$ < $m$ servers in the graph.

A sequence of $n$ requests to the vertices of the graph.

Each request should be served by a server.

Minimize the total distance moved by servers.

$\sigma = < S, M, K, A, D, B, D, B, D >$

costs = 2 0 2 1 1 1 1 1 1
Introduction

$k$-server problem

- We have a metric space of size $m$
  - $k < m$ servers in the graph

\[ \sigma = \langle S, M, K, A, D, B, D, B, D \rangle \]

\[ \text{costs} = 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \]
Introduction

\( k \)-sever problem

- We have a metric space of size \( m \)
  - \( k < m \) servers in the graph
- A sequence of \( n \) requests to the vertices of the graph
  - Each request should be served by a server

\[
\sigma = < S \ M \ K \ A \ D \ B \ D \ B \ D >
\]

\[
\text{costs} = 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1
\]
We have a metric space of size $m$
- $k < m$ servers in the graph
- A sequence of $n$ requests to the vertices of the graph
  - Each request should be served by a server
- Minimize the total distance moved by servers
We have a metric space of size $m$
- $k < m$ servers in the graph
- A sequence of $n$ requests to the vertices of the graph
  - Each request should be served by a server
- Minimize the total distance moved by servers

\[
\sigma = <S, M, K, A, D, B, D, B, D, G, F, E, Q, O, P, I, T, C, B, D, B, D, >
\]
\[
\text{costs} = 2, 0, 2, 1, 1, 1, 1, 1, 1, 1
\]
Introduction

\textit{k-sever problem}

- We have a metric space of size $m$
  - $k < m$ servers in the graph
- A sequence of \textit{n requests} to the vertices of the graph
  - Each request should be served by a server
- Minimize the total distance moved by servers

$\sigma = < S \ M \ K \ A \ D \ B \ D \ B \ D >$

\text{costs} = 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1
Introduction

\textbf{k-sever problem}

- We have a metric space of size \( m \)
  - \( k < m \) servers in the graph
- A sequence of \( n \) requests to the vertices of the graph
  - Each request should be served by a server
- Minimize the total distance moved by servers

\begin{align*}
\sigma &= <S\ M\ K\ A\ D\ B\ D\ B\ D> \\
\text{costs} &= 2\ 0\ 2\ 1\ 1\ 1\ 1\ 1\ 1
\end{align*}
We have a metric space of size $m$
- $k < m$ servers in the graph

A sequence of $n$ requests to the vertices of the graph
- Each request should be served by a server

Minimize the total distance moved by servers

$$\sigma = \langle S, M, K, A, D, B, D, B, D \rangle$$
$$\text{costs} = 2, 0, 2, 1, 1, 1, 1, 1, 1$$
**Introduction**

**$k$-server problem**

- We have a metric space of size $m$
  - $k < m$ servers in the graph
- A sequence of $n$ requests to the vertices of the graph
  - Each request should be served by a server
- Minimize the total distance moved by servers

$$
\sigma = < S \ M \ K \ A \ D \ B \ D \ B \ D >
$$

$$
costs = 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1
$$
**Introduction**

**k-sever problem**

- We have a metric space of size $m$
  - $k < m$ servers in the graph
- A sequence of $n$ requests to the vertices of the graph
  - Each request should be served by a server
- Minimize the total distance moved by servers

\[\sigma = < S, M, K, A, D, B, D, B, D >\]

\[\text{costs} = 2, 0, 2, 1, 1, 1, 1, 1, 1\]
Introduction

$k$-sever problem

- We have a metric space of size $m$
  - $k < m$ servers in the graph
- A sequence of $n$ requests to the vertices of the graph
  - Each request should be served by a server
- Minimize the total distance moved by servers

$\sigma = < S \ M \ K \ A \ D \ B \ D \ B \ D >$

$\text{costs} = 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1$
Introduction

$k$-sever problem

- We have a metric space of size $m$
  - $k < m$ servers in the graph
- A sequence of $n$ requests to the vertices of the graph
  - Each request should be served by a server
- Minimize the total distance moved by servers

$\sigma = < S \ M \ K \ A \ D \ B \ D \ B \ D >$

$\text{costs} = 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1$

COMP 7720 - Online Algorithms
Paging and $k$-Server Problem
We have a metric space of size $m$
- $k < m$ servers in the graph
- A sequence of $n$ requests to the vertices of the graph
  - Each request should be served by a server
- Minimize the total distance moved by servers

$$\sigma = \langle S \ M \ K \ A \ D \ B \ D \ B \ D \rangle$$
$$\text{costs} = 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1$$
Theorem

For any metric $G$, no deterministic $k$-server algorithm $Alg$ can have a competitive ratio smaller than $k$. 
Introduction

Major Results

Theorem

For any metric $G$, no deterministic $k$-server algorithm Alg can have a competitive ratio smaller than $k$.

Conjecture

Conjecture: for any metric space, there is a deterministic algorithm with competitive ratio of $k$.

$k$-server conjecture is one of the big open problems in the context of online algorithms.
Theorem

For any metric $G$, no deterministic $k$-server algorithm $\text{Alg}$ can have a competitive ratio smaller than $k$.

Conjecture

Conjecture: for any metric space, there is a deterministic algorithm with competitive ratio of $k$.

- $k$-server conjecture is one of the big open problems in the context of online algorithms.
  - Verified for $k = 2$, $m = k + 1$, $m = k + 2$, paths and trees.
Double Coverage Algorithm (DCA) for Paths

On a request to $x$:

- Move the closest server on left and closest server on right at the same ‘speed’ toward $x$ until one meets $x$.
  - If the closest server is at distance $d$, the algorithm incurs a cost of $2d$.
  - If there is no server on left (or right), just move the closest server!

Cost:
Double Coverage Algorithm (DCA) for Paths

On a request to $x$:
- Move the closest server on left and closest server on right at the same ‘speed’ toward $x$ until one meets $x$.
  - If the closest server is at distance $d$, the algorithm incurs a cost of $2d$.
- If there is no server on left (or right), just move the closest server!

Cost: 4
Double Coverage Algorithm (DCA) for Paths

On a request to \(x\):

- Move the closest server on left and closest server on right at the same ‘speed’ toward \(x\) until one meets \(x\).
  - If the closest server is at distance \(d\), the algorithm incurs a cost of \(2d\).
  - If there is no server on left (or right), just move the closest server!

Cost: 4
Double Coverage Algorithm (DCA) for Paths

On a request to $x$:

- Move the closest server on left and closest server on right at the same ‘speed’ toward $x$ until one meets $x$.
  - If the closest server is at distance $d$, the algorithm incurs a cost of $2d$.
  - If there is no server on left (or right), just move the closest server!

Cost: $4 + 2$
Introduction

Double Coverage Algorithm (DCA) for Paths

On a request to $x$:

- Move the closest server on left and closest server on right at the same ‘speed’ toward $x$ until one meets $x$.
  - If the closest server is at distance $d$, the algorithm incurs a cost of $2d$.
  - If there is no server on left (or right), just move the closest server!

Cost: $4 + 2$
Double Coverage Algorithm (DCA) for Paths

On a request to $x$:
- Move the closest server on left and closest server on right at the same ‘speed’ toward $x$ until one meets $x$.
  - If the closest server is at distance $d$, the algorithm incurs a cost of $2d$.
- If there is no server on left (or right), just move the closest server!

Cost: $4 + 2 + 1$
Double Coverage Algorithm (DCA) for Paths

- On a request to \( x \):
  - Move the closest server on left and closest server on right at the same ‘speed’ toward \( x \) until one meets \( x \).
    - If the closest server is at distance \( d \), the algorithm incurs a cost of \( 2d \).
  - If there is no server on left (or right), just move the closest server!

Cost: \( 4 + 2 + 1 \)
Double Coverage Algorithm (DCA) for Paths

- On a request to $x$:
  - Move the closest server on left and closest server on right at the same ‘speed’ toward $x$ until one meets $x$.
    - If the closest server is at distance $d$, the algorithm incurs a cost of $2d$.
  - If there is no server on left (or right), just move the closest server!

**Cost:** $4 + 2 + 1 + 2$
Introduction

Double Coverage Algorithm (DCA) for Paths

- On a request to $x$:
  - Move the closest server on left and closest server on right at the same ‘speed’ toward $x$ until one meets $x$.
    - If the closest server is at distance $d$, the algorithm incurs a cost of $2d$.
  - If there is no server on left (or right), just move the closest server!

Cost: $4 + 2 + 1 + 2$
Double Coverage Algorithm for Paths (cntd.)

Theorem

The double coverage algorithm (DCA) has a competitive ratio of $k$ for paths.

- So, it is the optimal deterministic algorithm for paths.
- For the proof, we used the potential function method 😊
An algorithm is called **lazy** if it moves at most one server to serve each request.

Is DCA a lazy algorithm?
Lazy Algorithms

An algorithm is called lazy if it moves at most one server to serve each request.

Is DCA a lazy algorithm?

- No, it might move two servers.
Lazy Algorithms

**Theorem**

*Any non-lazy algorithm $A$ can be converted to a lazy algorithm $A'$ without increasing its cost.*

- In $A'$, for each server, maintain a real position and a virtual position.
- Virtual positions are maintained similar to $A$.
- When $A$ moves $p$ servers for a request to node $x$:
  - Only update the real position of one server that arrives to $x$.
  - We ‘delay’ moving other servers.
Introduction

Lazy Algorithms

**Theorem**

Any non-lazy algorithm $A$ can be converted to a lazy algorithm $A'$ without increasing its cost.

- In $A'$, for each server, maintain a real position and a virtual position.
- Virtual positions are maintained similar to $A$.
- When $A$ moves $p$ servers for a request to node $x$:
  - Only update the real position of one server that arrives to $x$.
  - We ‘delay’ moving other servers.
Lazy Algorithms

Theorem

Any non-lazy algorithm $A$ can be converted to a lazy algorithm $A'$ without increasing its cost.

- In $A'$, for each server, maintain a real position and a virtual position.
- Virtual positions are maintained similar to $A$.
- When $A$ moves $p$ servers for a request to node $x$:
  - Only update the real position of one server that arrives to $x$.
  - We ‘delay’ moving other servers.
Lazy Algorithms

Theorem

Any non-lazy algorithm $A$ can be converted to a lazy algorithm $A'$ without increasing its cost.

- In $A'$, for each server, maintain a real position and a virtual position.
- Virtual positions are maintained similar to $A$.
- When $A$ moves $p$ servers for a request to node $x$:
  - Only update the real position of one server that arrives to $x$.
  - We ‘delay’ moving other servers.
Lazy Algorithms

**Theorem**

Any non-lazy algorithm $A$ can be converted to a lazy algorithm $A'$ without increasing its cost.

- In $A'$, for each server, maintain a real position and a virtual position.
- Virtual positions are maintained similar to $A$.
- When $A$ moves $p$ servers for a request to node $x$:
  - Only update the real position of one server that arrives to $x$.
  - We ‘delay’ moving other servers.
Introduction

Lazy Algorithms

Theorem

*Any non-lazy algorithm $A$ can be converted to a lazy algorithm $A'$ without increasing its cost.*

- In $A'$, for each server, maintain a real position and a virtual position.
- Virtual positions are maintained similar to $A$.
- When $A$ moves $p$ servers for a request to node $x$:
  - Only update the real position of one server that arrives to $x$.
  - We ‘delay’ moving other servers.
Any non-lazy algorithm $A$ can be converted to a lazy algorithm $A'$ without increasing its cost.

In $A'$, for each server, maintain a real position and a virtual position.

Virtual positions are maintained similar to $A$.

When $A$ moves $p$ servers for a request to node $x$:

- Only update the real position of one server that arrives to $x$.
- We ‘delay’ moving other servers.
**Theorem**

*Any non-lazy algorithm A can be converted to a lazy algorithm A’ without increasing its cost.*

In A’, for each server, maintain a real position and a virtual position.

Virtual positions are maintained similar to A.

When A moves p servers for a request to node x:

- Only update the real position of one server that arrives to x.
- We ‘delay’ moving other servers.

A’ saved a distance of 2 on moves of server 3!
Double Coverage Algorithm for Trees

- Move servers that have no other serve between them and the request
  - Move servers with equal speed to the requested sequence
  - Stop when any server arrives to the requested vertex
Double Coverage Algorithm for Trees

- Move servers that have no other serve between them and the request
  - Move servers with equal speed to the requested sequence
  - Stop when any server arrives to the requested vertex
Double Coverage Algorithm for Trees

- Move servers that have no other serve between them and the request
  - Move servers with equal speed to the requested sequence
  - Stop when any server arrives to the requested vertex
Double Coverage Algorithm for Trees

- Move servers that have no other serve between them and the request
  - Move servers with equal speed to the requested sequence
  - Stop when any server arrives to the requested vertex
Double Coverage Algorithm for Trees

- Move servers that have no other serve between them and the request
  - Move servers with equal speed to the requested sequence
  - Stop when any server arrives to the requested vertex
Double Coverage Algorithm for Trees

- Move servers that have no other serve between them and the request
  - Move servers with equal speed to the requested sequence
  - Stop when any server arrives to the requested vertex

Theorem

Double-Coverage algorithm (DCA) has a competitive ratio of $k$ for trees.

- Same potential & proof as in paths!
- The $k$-server conjecture is true (via DCA) for paths & trees
Recall that $k$-server becomes equal to caching problem when the metric is **uniform**. When distance between vertices associated with pages (yellow vertices) is the same.
Recall that $k$-server becomes equal to caching problem when the metric is **uniform**

- When distance between vertices associated with pages (yellow vertices) is the same.

We can **embed** a complete graph into a **star tree**

- So that the distances remain the same between pages (yellow vertices)
Revisiting Paging

- Recall that $k$-server becomes equal to caching problem when the metric is uniform. When distance between vertices associated with pages (yellow vertices) is the same.
- We can embed a complete graph into a star tree. So that the distances remain the same between pages (yellow vertices).
- What is the double-coverage algorithm for star? (paging)
Recall that $k$-server becomes equal to caching problem when the metric is \textit{uniform}.

- When distance between vertices associated with pages (yellow vertices) is the same.

We can \textit{embed} a complete graph into a \textbf{star tree}.

- So that the distances remain the same between pages (yellow vertices).

What is the double-coverage algorithm for star? (paging)
Recall that $k$-server becomes equal to caching problem when the metric is **uniform**

- When distance between vertices associated with pages (yellow vertices) is the same.

We can **embed** a complete graph into a **star tree**

- So that the distances remain the same between pages (yellow vertices)

What is the double-coverage algorithm for star? (paging)
Recall that $k$-server becomes equal to caching problem when the metric is uniform when distance between vertices associated with pages (yellow vertices) is the same.

We can embed a complete graph into a star tree so that the distances remain the same between pages (yellow vertices).

What is the double-coverage algorithm for star? (paging)
Recall that $k$-server becomes equal to caching problem when the metric is \textit{uniform}.

- When distance between vertices associated with pages (yellow vertices) is the same.

We can \textit{embed} a complete graph into a \textit{star tree}.

- So that the distances remain the same between pages (yellow vertices).

What is the double-coverage algorithm for star? (paging)

- It will be Flash-When-Full (FWF).
- Another proof that FWF has competitive ratio $k$.
- Note that FWF can be implemented in a lazy fashion!
Introduction

**Double Coverage Algorithm for** \( k = 2 \)

- When we have \( k = 2 \), we can use a version of double-coverage algorithm.
- On a request to \( x \), consider the shortest paths between the servers and \( x \).
  - Select shortest paths with maximum shared edges.
- When both servers move, they should get closer [for potential to work] (why)?
  
  Move servers at the same ‘speed’ on the selected paths
  - In case server \( s_1 \) ‘blocks’ \( s_2 \), stop moving \( s_2 \).
Double Coverage Algorithm for $k = 2$

- When we have $k = 2$, we can use a version of double-coverage algorithm.

- On a request to $x$, consider the shortest paths between the servers and $x$.

- Move servers at the same ‘speed’ on the selected paths.
  - In case server $s_1$ ‘blocks’ $s_2$, stop moving $s_2$. 
When we have $k = 2$, we can use a version of double-coverage algorithm.

On a request to $x$, consider the shortest paths between the servers and $x$.

Move servers at the same ‘speed’ on the selected paths.

In case server $s_1$ ‘blocks’ $s_2$, stop moving $s_2$. 
Double Coverage Algorithm for $k = 2$

- When we have $k = 2$, we can use a version of double-coverage algorithm.
- On a request to $x$, consider the shortest paths between the servers and $x$.
- Move servers at the same ‘speed’ on the selected paths.
  - In case server $s_1$ ‘blocks’ $s_2$, stop moving $s_2$. 
Introduction

Double Coverage Algorithm for $k = 2$

- When we have $k = 2$, we can use a version of double-coverage algorithm.
- On a request to $x$, consider the shortest paths between the servers and $x$.

- Move servers at the same ‘speed’ on the selected paths.
  - In case server $s_1$ ‘blocks’ $s_2$, stop moving $s_2$. 
When we have $k = 2$, we can use a version of double-coverage algorithm.

On a request to $x$, consider the shortest paths between the servers and $x$.

Move servers at the same ‘speed’ on the selected paths.

In case server $s_1$ ‘blocks’ $s_2$, stop moving $s_2$. 
When we have \( k = 2 \), we can use a version of double-coverage algorithm.

On a request to \( x \), consider the shortest paths between the servers and \( x \).

Move servers at the same ‘speed’ on the selected paths.

- In case server \( s_1 \) ‘blocks’ \( s_2 \), stop moving \( s_2 \).
When we have \( k = 2 \), we can use a version of double-coverage algorithm.

On a request to \( x \), consider the shortest paths between the servers and \( x \):

- Selects shortest paths with maximum shared edges!

Move servers at the same ‘speed’ on the selected paths:

- In case server \( s_1 \) ‘blocks’ \( s_2 \), stop moving \( s_2 \).
When we have $k = 2$, we can use a version of double-coverage algorithm.

On a request to $x$, consider the shortest paths between the servers and $x$
- Selects shortest paths with maximum shared edges!

Move servers at the same ‘speed’ on the selected paths
- In case server $s_1$ ‘blocks’ $s_2$, stop moving $s_2$. 
Introduction

Double Coverage Algorithm for $k = 2$

- When we have $k = 2$, we can use a version of double-coverage algorithm.

- On a request to $x$, consider the shortest paths between the servers and $x$
  - Selects shortest paths with maximum shared edges!

- Move servers at the same ‘speed’ on the selected paths
  - In case server $s_1$ ‘blocks’ $s_2$, stop moving $s_2$. 
When we have $k = 2$, we can use a version of double-coverage algorithm.

On a request to $x$, consider the shortest paths between the servers and $x$:

- Selects shortest paths with maximum shared edges!
- When both servers move, they should get closer [for potential to work] (why)?

Move servers at the same ‘speed’ on the selected paths:

- In case server $s_1$ ‘blocks’ $s_2$, stop moving $s_2$. 
Introduction

Double Coverage Algorithm for $k = 2$

- When we have $k = 2$, we can use a version of double-coverage algorithm.

- On a request to $x$, consider the shortest paths between the servers and $x$
  - Selects shortest paths with maximum shared edges!
  - When both servers move, they should get closer [for potential to work] (why)?

- Move servers at the same ‘speed’ on the selected paths
  - In case server $s_1$ ‘blocks’ $s_2$, stop moving $s_2$.

**Theorem**

*DCA has a competitive ratio of $k$ when $k = 2$.*

Similar proof & potential (exercise)
Introduction

**Double Coverage Algorithm for** $k = 3$?

**Theorem**

When $k = 3$, double coverage algorithm is not $k$-competitive even for cycles.

- For $\sigma = (B\ D\ E)^n$, we have $\text{cost}(\text{DCA}) > n$.

\[\sigma = (B\ D\ E)^n\]
Introduction

Double Coverage Algorithm for $k = 3$?

Theorem

When $k = 3$, double coverage algorithm is not $k$-competitive even for cycles.

- For $\sigma = \sigma = (B \ D \ E)^n$, we have $\text{cost}(DCA) > n$. 

\[ \sigma = (B \ D \ E)^n \]
Double Coverage Algorithm for $k = 3$?

**Theorem**

When $k = 3$, double coverage algorithm is not $k$-competitive even for cycles.

For $\sigma = \sigma = (B \ D \ E)^n$, we have $\text{cost}(DCA) > n$. 

\[ \sigma = (B \ D \ E)^n \]
Theorem

When $k = 3$, the double coverage algorithm is not $k$-competitive even for cycles.

For $\sigma = (B \ D \ E)^n$, we have $\text{cost}(DCA) > n$. 

\[ \sigma = (B \ D \ E)^n \]
Introduction

Double Coverage Algorithm for $k = 3$?

Theorem

*When $k = 3$, double coverage algorithm is not $k$-competitive even for cycles.*

For $\sigma = (B \ D \ E)^n$, we have $\text{cost}(DCA) > n$. 

$\sigma = (B \ D \ E)^n$
Theorem

When \( k = 3 \), double coverage algorithm is not \( k \)-competitive even for cycles.

For \( \sigma = (B \ D \ E)^n \), we have \( \text{cost}(DCA) > n \).
Theorem

When \( k = 3 \), double coverage algorithm is not \( k \)-competitive even for cycles.

For \( \sigma = (B \ D \ E)^n \), we have

\[
\text{cost}(DCA) > n.
\]
Double Coverage Algorithm for \( k = 3 \)?

**Theorem**

When \( k = 3 \), double coverage algorithm is not \( k \)-competitive even for cycles.

- For \( \sigma = (B \ D \ E)^n \), we have \( \text{cost}(DCA) > n \).
- Cost of \( \text{OPT} \) for \( \sigma \) is 2.
  - \( \text{OPT} \) moves server 3 to \( B \) and makes no further move.

\[ \sigma = (B \ D \ E)^n \]
Double Coverage Algorithm for $k = 3$?

**Theorem**

When $k = 3$, double coverage algorithm is not $k$-competitive even for cycles.

- For $\sigma = \sigma = (B \ D \ E)^n$, we have $\text{cost}(DCA) > n$.
- Cost of $\text{OPT}$ for $\sigma$ is 2.
  - $\text{OPT}$ moves server 3 to $B$ and makes no further move.
- The competitive ratio of DCA when $k = 3$ is more than $\frac{n}{2}$ for a cycle graph.

$$\sigma = (B \ D \ E)^n$$
When $k = 3$, double coverage algorithm is not $k$-competitive even for cycles.

- For $\sigma = (B \ D \ E)^n$, we have $\text{cost}(\text{DCA}) > n$.

- Cost of $\text{OPT}$ for $\sigma$ is 2.
  - $\text{OPT}$ moves server 3 to $B$ and makes no further move.

- The competitive ratio of DCA when $k = 3$ is more than $\frac{n}{2}$ for a cycle graph.
  - This is much worse than $k$ (why?)
Introduction

Double Coverage Algorithm (DCA) for $k = 2 \& k = 3$

- Why DCA has a competitive ratio of $k$ when $k = 2$ and unbounded competitive ratio for $k = 3$? (intuition)
Introduction

Double Coverage Algorithm (DCA) for \( k = 2 \) & \( k = 3 \)

- Why DCA has a competitive ratio of \( k \) when \( k = 2 \) and unbounded competitive ratio for \( k = 3 \)? (intuition)

- When \( k = 2 \), the triangle formed by the two servers & the requested node can be embedded into a tree.

\[
\begin{align*}
\text{When } k = 2, \quad & \text{the triangle formed by the two servers & the requested node can be embedded into a tree.} \\
& \quad \text{where } a, b, c \text{ are the sides of the triangle.}
\end{align*}
\]
Introduction

Double Coverage Algorithm (DCA) for $k = 2$ & $k = 3$

- Why DCA has a competitive ratio of $k$ when $k = 2$ and unbounded competitive ratio for $k = 3$? (intuition)
- When $k = 2$, the triangle formed by the two servers & the requested node can be embedded into a tree.
- When $k = 3$, the graph formed by the three vertices & the requested node cannot be necessarily embedded into a tree.
  - E.g., a cycle cannot be embedded into a tree
DCA is $k$-competitive (optimal) for paths, trees, and any metric that can be embedded in trees (e.g., complete graph).
DCA is $k$-competitive (optimal) for paths, trees, and any metric that can be embedded in trees (e.g., complete graph).

DCA is $k$-competitive (optimal) for $k = 2$.

DCA is not useful for $k \geq 3$ even if the metric is a cycle.
Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?

For \( n \) requests, cost (Balance) = \( n \cdot d \)

\[ \text{cost} (\text{Opt}) = d + n \text{ (why?)} \]

The competitive ratio of the Balance algorithm is at least \( \frac{nd}{n+d} \approx d \), which is much more than the optimal ratio of \( k = 2 \).
Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?

For \( n \) requests,

\[
\text{cost (Balance)} = n \cdot d
\]

\[
\text{cost (Opt)} = d + n \approx d \quad \text{(why?)}
\]

The competitive ratio of the Balance algorithm is at least

\[
\frac{nd}{n + d} \approx d
\]

which is much more than the optimal ratio of \( k = 2 \).

Balance is \( k \)-competitive for metrics with \( k + 1 \) nodes.

\[
\sigma = (D \ C \ B \ A)^n
\]
Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?

For \( n \) requests, cost (Balance) = \( n \cdot d \)

\[ \text{cost (Opt)} = d + n \] (why?)

The competitive ratio of the Balance algorithm is at least \( nd \), which is much more than the optimal ratio of \( k = 2 \).

Balance is \( k \)-competitive for metrics with \( k + 1 \) nodes.

\[ \sigma = (D \ C \ B \ A)^n \]
Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?

\[ \sigma = (D \ C \ B \ A)^n \]
Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?

For $n$ requests, cost (Balance) = $n \cdot d$

Cost (Opt) = $d + n$ (why?)

The competitive ratio of the Balance algorithm is at least $\frac{nd}{n+d} \approx d$, which is much more than the optimal ratio of $k = 2$.

Balance is $k$-competitive for metrics with $k+1$ nodes.

$\sigma = (D \ C \ B \ A)^n$
**Introduction**

**Balancing Algorithms**

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?

\[ \sigma = (D \ C \ B \ A)^n \]
Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?

\[ \sigma = (D \ C \ B \ A)^n \]
Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?

\[ \sigma = (D \ C \ B \ A)^n \]
Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?

\[ \sigma = (D \ C \ B \ A)^n \]
Introduction

Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?

For $n$ requests, cost (Balance) = $n \cdot d$

$\text{cost (Opt)} = d + n$ (why?)

The competitive ratio of the Balance algorithm is at least $n/d \approx d/n$, which is much more than the optimal ratio of $k = 2$.

Balance is $k$-competitive for metrics with $k + 1$ nodes.

$\sigma = (D \ C \ B \ A)^n$
Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?
  - For $n$ requests, $\text{cost}(\text{Balance}) = n \cdot d$

$$\sigma = (D \ C \ B \ A)^n$$
Introduction

Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?
  - For $n$ requests, $\text{cost}(\text{Balance}) = n \cdot d$
  - $\text{cost}(\text{OPT}) = d + n$ (why?)
  - The competitive ratio of the Balance algorithm is at least $\frac{nd}{n+d} \approx d$, which is much more than the optimal ratio of $k = 2$.

\[ \sigma = (D \ C \ B \ A)^n \]
Balancing Algorithms

- Move the server which after (potentially) serving the request, has moved less than other servers
- Is it a good algorithm?
  - For $n$ requests, $cost(Balance) = n \cdot d$
  - $cost(OPT) = d + n$ (why?)
  - The competitive ratio of the Balance algorithm is at least $\frac{nd}{n+d} \approx d$, which is much more than the optimal ratio of $k = 2$.

**Balance is $k$-competitive for metrics with $k + 1$ nodes**

$$\sigma = (D \ C \ B \ A)^n$$
Randomized algorithms

- Compare against *oblivious adversary*
  - For any metric space, no algorithm can be better than $\log k$ competitive
Introduction

Randomized algorithms

- Compare against *oblivious adversary*
  - 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

- Compare against *oblivious adversary*
  - 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
- Verified for hierarchical binary trees [?]
Randomized algorithms

- Compare against *oblivious adversary*
  - 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
- Verified for hierarchical binary trees [?]
- For general graphs, there is a $O(\log^3 m \log^2 k)$-competitive graph
  - Better than $2k - 1$ when $m$ is sub-exponential of $k$ [?].
Assignment I
If your mark is under or close to 75%, you should reconsider your approach to this course.