COMP 4060/7720 - Online Algorithms

Paging and $k$-Server Problem

Shahin Kamali

$k$-server Problem

University of Manitoba
**k-sever problem**

- A metric is a set of points with a **distance** between each of pairs so that \( d(x, y) \leq d(x, z) + d(z, y) \).
  - E.g., a connected, undirected graph or a set of points in plane
- 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
  - Requests appear in an online manner
- Minimize the total distance moved by servers

\[
\sigma = < S \ M \ K \ A \ D \ B \ D \ B \ D > \\
\text{costs} = 2 \ 0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1
\]
The $k$-server Problem

- What happens if we have a complete graph (a **uniform** metric)?
  - If there is a request to a vertex at which a server is located → there is no cost; otherwise, there is a cost of 1 to move a server to requested vertex.
    - Think of vertices as pages; vertices with servers on them are pages in the cache → caching problem.

- Recall that for caching problem, we have:

  **Theorem**

  No deterministic algorithm can achieve a competitive ratio better than $k$, and LRU and FIFO achieve this ratio.
  No randomized algorithm can achieve a competitive ratio that is asymptotically better than $\Theta(\log k)$ and Mark algorithm achieves this.

- $k$-server problem has the **right level of difficulty** compared to paging (which is ‘too easy’) and Metrical Task Systems (another problem which is ‘too hard’).
Greedy Algorithm

- Move the closest server to serve each request.
- Is Greedy a good algorithm?
  - what about the input $\sigma = \langle B \ R \ B \ R \ldots \rangle$?
    - For $n$ requests, greedy incurs a cost of $n$
    - OPT moves another server from $M$ to $T$ at a cost of 3 and incurs no cost.
    - Competitive ratio will be at least $\frac{n}{3}$ for this graph!
Greedy Algorithm

**Theorem**

*For any graph of diameter \( d \), the competitive ratio of greedy is at least \( \frac{n}{2d} \).*

- It holds for any graph, even a path!
- Consider two vertices \( A \) and \( B \) which are close to one server and further from other servers.
  - Greedy servers sequence \( \langle A \ B \ A \ B \ \ldots \rangle \) by one server
Lower Bound for Deterministic Algs

### Theorem

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

### proof:

- Consider any connected subgraph of $G$ on $k + 1$ vertices.
- Create an adversarial sequence $\sigma$ where the adversary always asks for the node at which Alg has no server.
  - Assume the cost of Alg is $C$ for $\sigma$
- We show there are $k$ offline algorithms so that the summations of the costs of all these algorithms is roughly $C$
  - At least one of them has cost $C/k$
  - The competitive ratio of Alg will be at least $\frac{C}{C/k} = k$. 

COMP 4060/7720 - Online Algorithms    Paging and $k$-Server Problem
Lower Bound for Deterministic Algs

- Alg and the \( k \) offline algorithms always have different configurations.
- On each request, Alg has to move a server, while all offline algorithms have a ‘hit’.
- when Alg makes a move, exactly one offline algorithm uses the reverse move (to maintain different configurations).

\[ \sigma : \quad A \ D \ F \ G \ C \]

Algorithm:

\[
\begin{align*}
D & \rightarrow A \\
F & \rightarrow D \\
G & \rightarrow F \\
C & \rightarrow G \\
A & \rightarrow C
\end{align*}
\]

Offline algorithms:

\[
\begin{align*}
\text{off3:} & \quad A \rightarrow D \\
\text{off2:} & \quad D \rightarrow F \\
\text{off1:} & \quad F \rightarrow G \\
\text{off4:} & \quad G \rightarrow C
\end{align*}
\]
Lower Bound for Deterministic Algs

- At each given time, exactly one offline algorithm moves a server.
- The offline algorithm that moves a server, uses the reverse move so that all algorithms maintain different configurations.
- The cost of all offline algorithms is roughly equal to Alg
  - They pay a bit more to move servers to form the initial configuration at the beginning.
- Offline algorithms fail the online algorithm by emulating it!
**k-server 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 when $k = 2$, $m = k + 1$, $m = k + 2$, 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: $4 + 2 + 1 + 2$
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 use the potential function method :)}
Potential Function Method Review

1. Define the potential as a function of the states of $A$ and $OPT$ at time $t$ (before serving the $t$'th request).

2. Define the amortized cost at time $t$ as the summation of the actual cost and the difference in potential before and after serving the $t$'th request, i.e., $amortized\_cost(t) = actual\_cost(t) + \Phi(t+1) - \Phi(t)$.

3. Assuming the potential is defined properly, we should be able to show $amortized\_cost(t) \leq c \cdot OPT(t)$,
Potential Function Method Steps

- Define the potential
- Find the followings for any timestep $t$:
  - The cost of Opt $\Rightarrow cost(opt)$ (usually a variable $j$).
  - How the actions of Opt changes potential $\Rightarrow \Delta_{opt} \Phi$
  - The actual cost of the algorithm
  - How the actions of alg changes potential $\Rightarrow \Delta_{alg} \Phi$

- The amortized cost is:
  $amortized\_cost = actual\_cost + \Delta \Phi = actual\_cost + \Delta_{opt} \Phi + \Delta_{alg} \Phi$

- For a competitive ratio of $c$ we should have:
  $actual\_cost + \Delta_{opt} \Phi + \Delta_{alg} \Phi \leq c \cdot cost(opt)$

- For example, for MTF (list update) we had $actual\_cost = i$, $cost_{opt} = j + k$, $\Delta_{Opt} \Phi \leq k$, $\Delta_{MTF} \Phi \leq -i + 2j$.
  - Amortize$_{cost} \leq i + k + (-i + 2j) = 2j + k$, which is no more than $2 \cdot (j + k)$
Potential for DCA

- At any given time, for any serve $s_i$, let $p(s_i)$ be the distance between the location of $s_i$ in DCA configuration and the location of $s_i$ in OPT’s configuration.
- Define $P = k \times (p(s_1) + p(s_2) + \ldots + p(s_k))$
  - We consider $P$ as the first component in potential

\[
p(1) = 0,\; p(2) = 1,\; p(3) = 2,\; P(4) = 2,\; p(5) = 2
\]

\[
P = 5 \times (0 + 1 + 2 + 2 + 2) = 35
\]
For the second component, for any pair \( s_i, s_j \) of servers, let \( q(s_i, s_j) \) be the distance between the location of \( s_i \) and \( s_j \) in DCA configuration.

Define \( Q = \sum_{i \neq j} q(s_i, s_j) \)

- the closer the servers are \( \rightarrow \) the lower the potential

\[
q(1, 2) = 2, \ q(1, 3) = 7, \ q(1, 4) = 8, \ q(1, 5) = 10, \ q(2, 3) = 5 \\
q(2, 4) = 6, \ q(2, 5) = 8, \ q(3, 4) = 1, \ q(3, 5) = 3, \ q(4, 5) = 2
\]

\[
Q = 2 + 7 + 8 + 10 + 5 + 6 + 8 + 1 + 3 + 2 = 52
\]
The potential at time $t$ is $\Phi = P + Q$, the summation of the two components.

- So, the potential is now defined :)
Potential function method for DCA

- Assume Opt moves a server $s$ for $j$ units for a request.
  - The cost of opt is $j$

- How does the first component $P$ of the potential change?
  - The distance between the location of $s$ in configuration of Opt and DCA is increased by at most $j$
  - For other servers $P$ does not change!
  - So, $P$ increases by at most $k \cdot j$

- How does the second component $Q$ change?
  - It does not change! ($Q$ depends on configuration of DCA)

\[
\text{cost}_{\text{opt}} = j \\
\Delta_{\text{Opt}} \Phi \leq k \cdot j
\]
Case 1: DCA moves only one server \( w \) a distance \( d \).
- Happens when there is no server on left (or right)
- The actual cost of DCA is \( d \).

How does the first component \( P \) of the potential change?
- The distance between the location of \( w \) in configuration of Opt and DCA is decreased by exactly \( d \).
- \( P \) is increased by \( -kd \)

How does the second component \( Q \) change?
- Distance of \( w \) with any other vertex is increased by \( d \)
- \( Q \) is increased by \((k - 1) \cdot d\)

\[
\text{cost}_{\text{opt}} = j\]
\[
\Delta_{\text{Opt}} \Phi \leq k \cdot j
\]
\[
\text{actual\_cost} = ?
\]
\[
\text{actual\_cost} = d
\]
\[
\Delta_{\text{DCA}} \Phi = ?
\]
\[
\Delta_{\text{DCA}} \Phi = -kd + (k - 1)d = -d
\]

amortized\_cost = \[
d + (kj - d) = kj = k \cdot \text{cost}_{\text{opt}}
\]
Case 2: DCA moves two server \( L \) and \( R \) a distance \( d \).

- There is one server on each side of the request
- The actual cost of DCA is \( 2d \).

How does the first component \( P \) of the potential change?

- The distance between the location \( R \) in configuration of Opt and DCA is decreased by \( d \) while the distance of the \( L \) is increased by at most \( d \).
- \( P \) is increased by at most \( 0 \).

How does the second component \( Q \) change?

- For any server \( z \), \( q(z, L) + q(z, R) \) is unchanged.
- \( q(L, R) \) is decreased by \( 2d \) \( \Rightarrow \) \( Q \) is increased by \(-2d\)

\[
\begin{align*}
\text{actual\_cost} &= 2d \\
\Delta_{DCA} \Phi &= 0 + (-2d) = -2d \\
\text{amortized\_cost} &\leq 2d + (kj - 2d) = kj = k \cdot \text{cost\_opt}
\end{align*}
\]
Potential function method for DCA

Theorem

*The Double Coverage Algorithm has a competitive ratio of* $k$ *for paths.*

- And this is the best deterministic algorithm for paths (why?)
- The algorithm can be extended to trees!
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.
**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

**Theorem**

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

- Similar potential & proof as in paths!
- The $k$-server conjecture is true (via DCA) for paths & trees
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)
  - 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!
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 a ‘red spider’ that embeds shortest distances of servers and request.
- Apply DCA using the red spider (move servers on the star edges).
- In reality, we cannot move on the star (since it is not a part of graph).
  - Use a lazy variant; star positions are virtual positions; in reality only one server is moved.
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.

![Diagram of triangle and square with labels and calculations](image)
Double Coverage Algorithm (DCA) Summary

- 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, $\text{cost}(\text{Balance}) = n \cdot d$
  - $\text{cost}(\text{OPT}) = d + n$ (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 $k = 2$.

- 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
- 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 $k$
Randomized Algorithm CIRC for Cycle

- Select a point $P$, uniformly at random, from the cycle of length $C$.
  - Think of $P$ as a ‘road-block’ and apply DCA for the resulting segment $L$
  - This selection of $P$ is equivalent to deletion of a random edge from the cycle

**Theorem**

*CIRC is a 2k-competitive algorithm for cycle*

- Observation: $P$ appears in the shortest path between $(A, B)$ with probability $d(A, B)/C$. 

![Diagram of cycle with points A, B, and P]
Let OPT-Line be the optimal offline algorithm when restricted to $L$.

We have $Cost(CIRC) \leq k \cdot Cost(OPT-Line)$ (double-coverage algorithm on line).

$Cost(OPT-Line) \leq 2Cost(OPT)$

1. Assume OPT makes moves of lengths $d_1, d_2, \ldots, d_n$
   
   $cost(opt) = d_1 + d_2 + \ldots + d_n$

2. Apply the same moves as OPT; with additional penalty of at most $C$ if a server passes $P$ (the penalty means you go all the way through other side).
   
   The chance of passing $P$ on a move of length $d_i$ is $d_i / C$.

   - The whole penalty is expected to be at most $d_1 / C \cdot C + d_2 / C \cdot C + \ldots + d_n / C \cdot C = Cost(OPT)$.

   - The expected cost of OPT-Line is at most $d_1 + d_2 + \ldots + d_n + Cost(OPT) = 2Cost(OPT)$.
Randomized Algorithm CIRC for Cycle

- In summary, we have \( \text{cost}(\text{CIRC}) \leq k \cdot \text{cost}(\text{OPT-Line}) \) and \( \text{cost}(\text{OPT-Line}) \leq 2\text{cost}(\text{OPT}) \).

**Theorem**

\textit{CIRC is a 2k-competitive algorithm for cycle}

- Is it good?
  - Yes (it is the best existing algorithm) and No (we hope to get something around \( \log k \)).
  - Deterministic k-server conjecture is still open for cycles.

- Here, we reduced a cycle to a line segment

- This type of reduction is the main tool for analysis of randomized k-server
  - Reduce an arbitrary graph to a ‘hierarchical binary tree’