Metrical Task System Problem and K-Server Problem

Xiaoxi Zhang
Department of Computer Science
zxx1121xiaoxi@gmail.com

Motivation

- Introduce some delicate details in Work Function Algorithm
- Introduce some good ideas in the randomized algorithm of k-server problem
- Hope you give me some inspirations on my problem

Outline

- Metrical Task System Problem
  - The MTS Work Function Algorithm
- The k-Server Problem
  - Deterministic: The K-server Work Function Algorithm
  - Randomized:
    STOC: Randomized k-Server on Hierarchical Binary Trees
- Our Model
  - Third level
    - Fourth level
      - Fifth level

The Metric Space

Definition: A metric space $M = (V, d)$ consists of a set of points V with a distance function $d: V \mapsto R$ satisfying the following properties:

$d(u,v) \geq 0$ for all $u, v \in V$.

$d(u,v) = 0$ iff $u = v$.

$d(u,v) = d(v,u)$ for all $u, v \in V$.

$d(u,v) + d(v,w) \geq d(u,w)$ for all $u, v, w \in V$.

Metrical Task System Problem

- A task r is defined as an N-array vector of costs r=<r(1),r(2),r(3),…,r(N)>, r(i) is the cost of "processing" the task while **in state i**
- Transition cost between two states is d(i,j)
- Upon arrival of the task r, the player(or algorithm) change the state to any desired state and then processes the task in the state.

Metrical Task System Problem

- ALG[i] denotes the state in which the ith task is processed by an algorithm ALG.
- The total cost by ALG for task sequence σ is

$$\text{ALG}(\sigma) = \sum_{i=1}^{n} d(ALG[i-1], ALG[i]) + \sum_{i=1}^{n} r_i(ALG[i])$$

Outline

- Metrical Task System Problem
  - The MTS Work Function Algorithm
- The k-Server Problem
  - Deterministic: The K-server Work Function Algorithm
  - Randomized:
    STOC: Randomized k-Server on Hierarchical Binary Trees
- Our Model
  - Third level
    - Fourth level
      - Fifth level

The MTS Work Function Algorithm

- Let (S,d) be any MTS and let s_0 be an initial state
- Fix any task sequence $\sigma=r_1r_2,\ldots,r_n$
- Let $\sigma_i=r_1r_2,\ldots,r_i$ be the prefix of σ
- For each state $s \in S$, define $w_i(s)$ to be the minimum (offline) cost to process σ_i starting from s_0 and ending in state s

The MTS Work Function Algorithm

- For each state $s \in S$, define $w_i(s)$ to be the minimum (offline) cost to process σ_i starting from s_0 and ending in state s
- **Optimal offline cost** $OPT(\sigma) = \min_{x \in S} w_n(x)$
- To compute $w_n(s)$, we have:

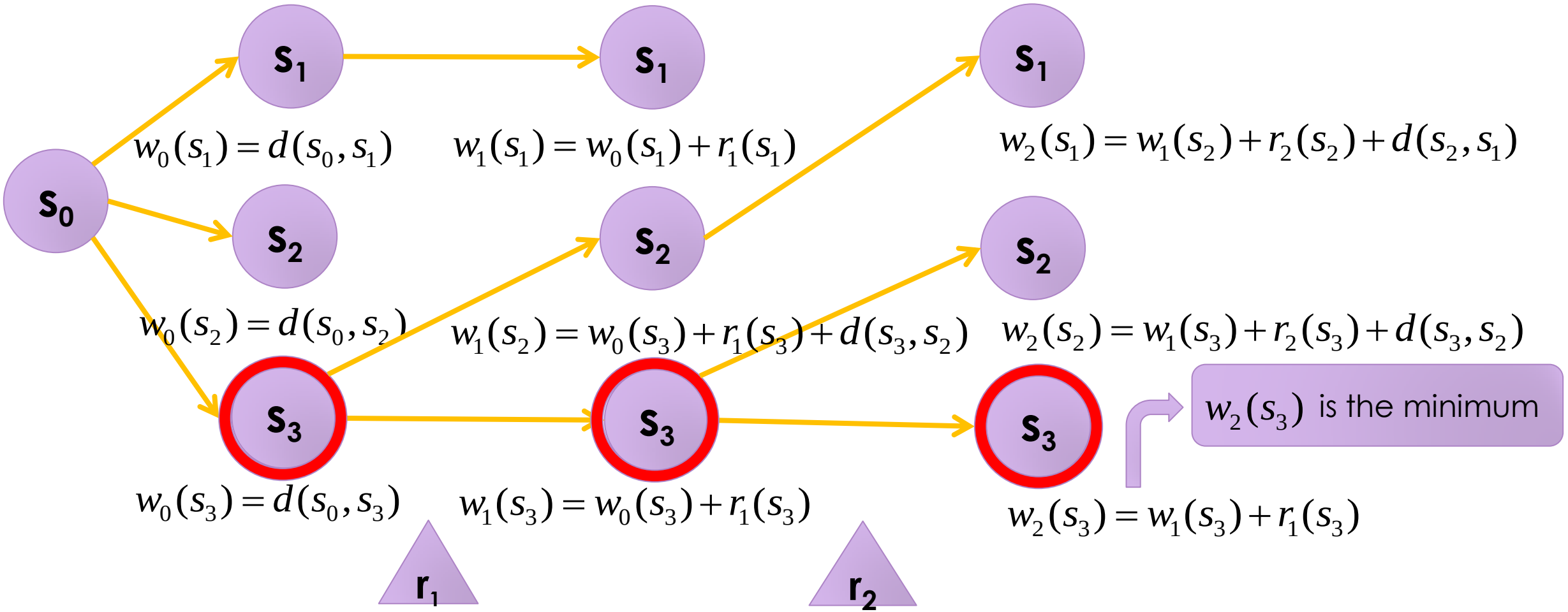
$$w_{i+1}(s) = \min_{x \in S} \{w_i(x) + r_{i+1}(x) + d(x,s)\},$$

$$w_0(s) = d(s_0, s)$$

Dynamic programming

The MTS Work Function Algorithm

$$w_{i+1}(s) = \min_{x \in S} \{w_i(x) + r_{i+1}(x) + d(x,s)\}, w_0(s) = d(s_0, s)$$



The MTS Work Function Algorithm

- Online:

  Algorithm WFA: Suppose that the algorithm is in state s_i after processing the ith tasks in σ_i. Then to process r_{i+1} ,the algorithm moves to a state

$$\begin{cases} s_{i+1} = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}, \\ \quad w_{i+1}(s_{i+1}) = w_i(s_{i+1}) + r_{i+1}(s_{i+1}) \end{cases}$$

WFA Can Always Choose An Appropriate State s_{i+1}

Offline: $w_{i+1}(s) = \min_{x \in S} \{w_i(x) + r_{i+1}(x) + d(x,s)\}, w_0(s) = d(s_0, s)$

Algorithm WFA:

$$\begin{cases} s_{i+1} = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}, & (1) \\ w_{i+1}(s_{i+1}) = w_i(s_{i+1}) + r_{i+1}(s_{i+1}) & (2) \end{cases}$$

Proof: **Let A be the set of the states satisfying (1)and (2).**

We firstly define a set A' satisfying (1). Clearly, A' isn't empty.

Then we prove that there is an element of A' that satisfies (2).

WFA Can Always Choose An Appropriate State s_{i+1}

Offline: $w_{i+1}(s) = \min_{x \in S} \{w_i(x) + r_{i+1}(x) + d(x,s)\}, w_0(s) = d(s_0, s)$

Algorithm WFA:
$$\begin{cases} s_{i+1} = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}, \\ w_{i+1}(s_{i+1}) = w_i(s_{i+1}) + r_{i+1}(s_{i+1}) \end{cases}$$

→ A'

→ Choose the same state

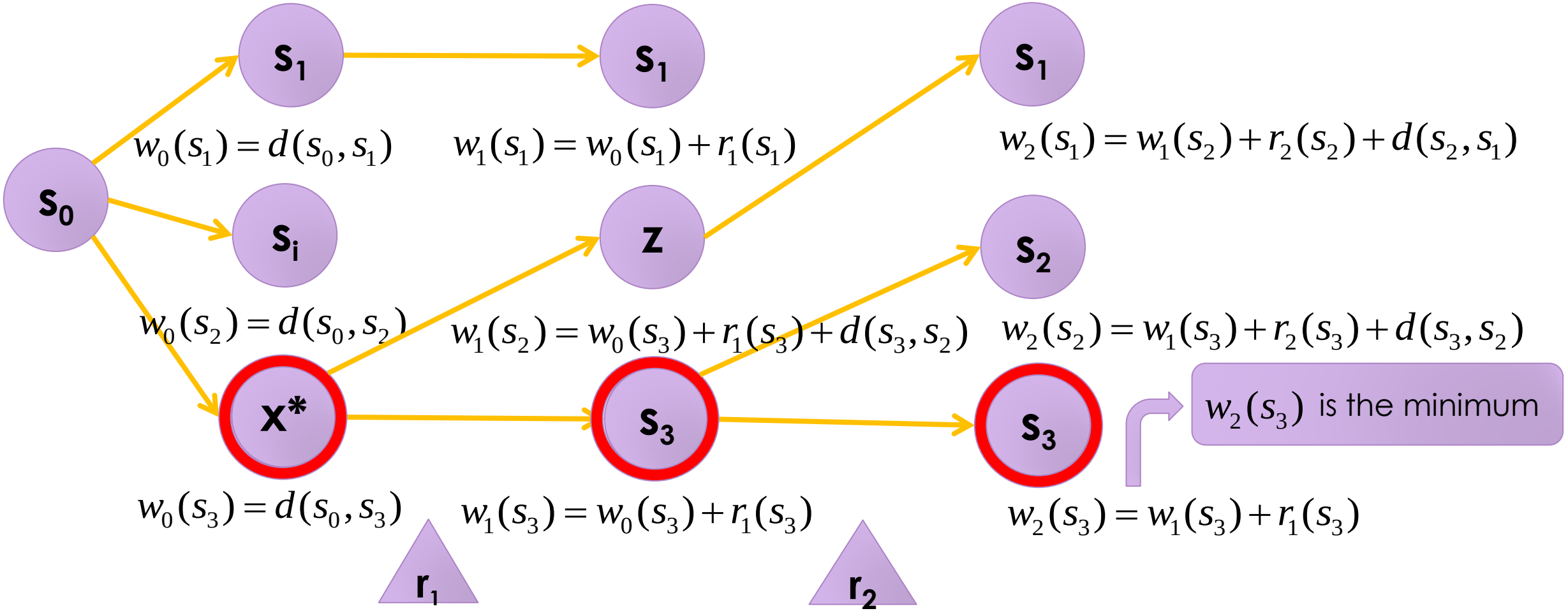
Proof: 1. $\forall x \in S$, we have $w_{i+1}(x) \le w_i(x) + r_{i+1}(x)$ → Staying in the same state may be not optimal

2. Then we have $w_{i+1}(x) + d(x, s_i) \le w_i(x) + r_{i+1}(x) + d(x, s_i)$

3. **Let z be any element of A'**
   and let x* be a state for which the minimum in equation.
   Then we have $w_{i+1}(z) = w_i(x^*) + r_{i+1}(x^*) + d(x^*, z)$ → Offline or Definition of w

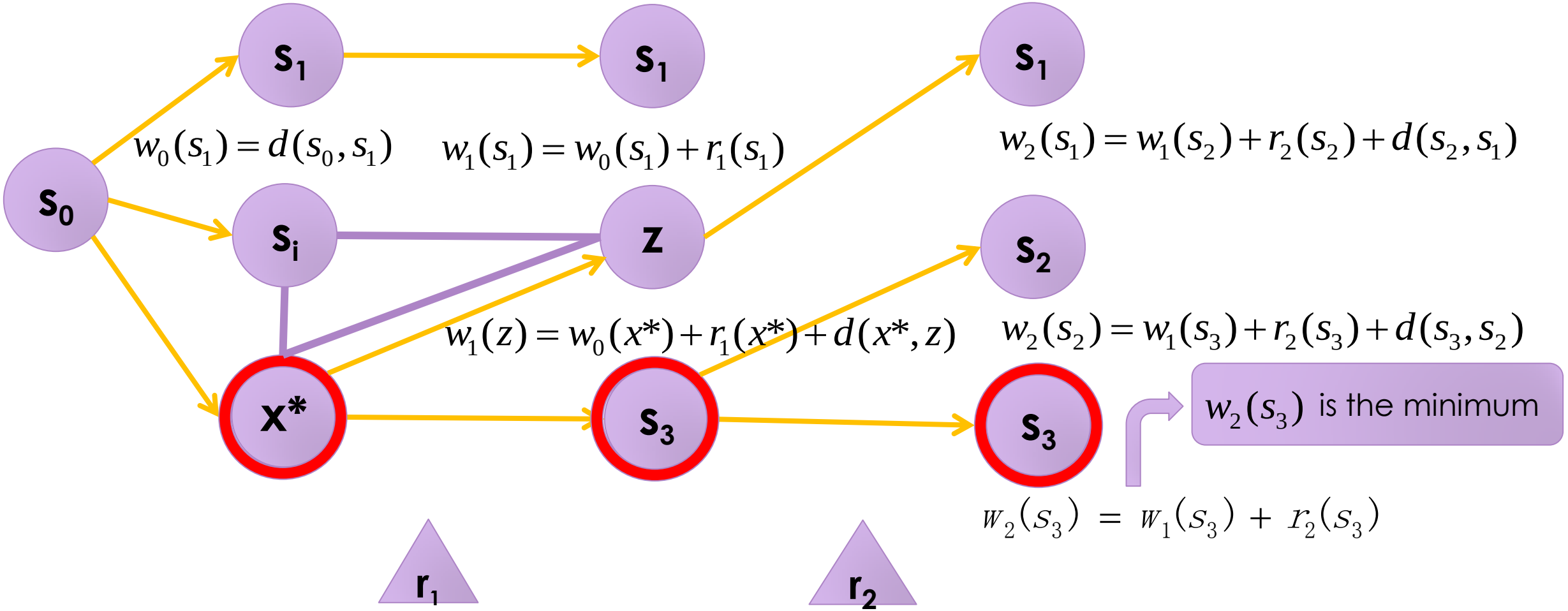
$$w_{i+1}(s) = \min_{x \in S}\{w_i(x) + r_{i+1}(x) + d(x,s)\}, w_0(s) = d(s_0, s)$$

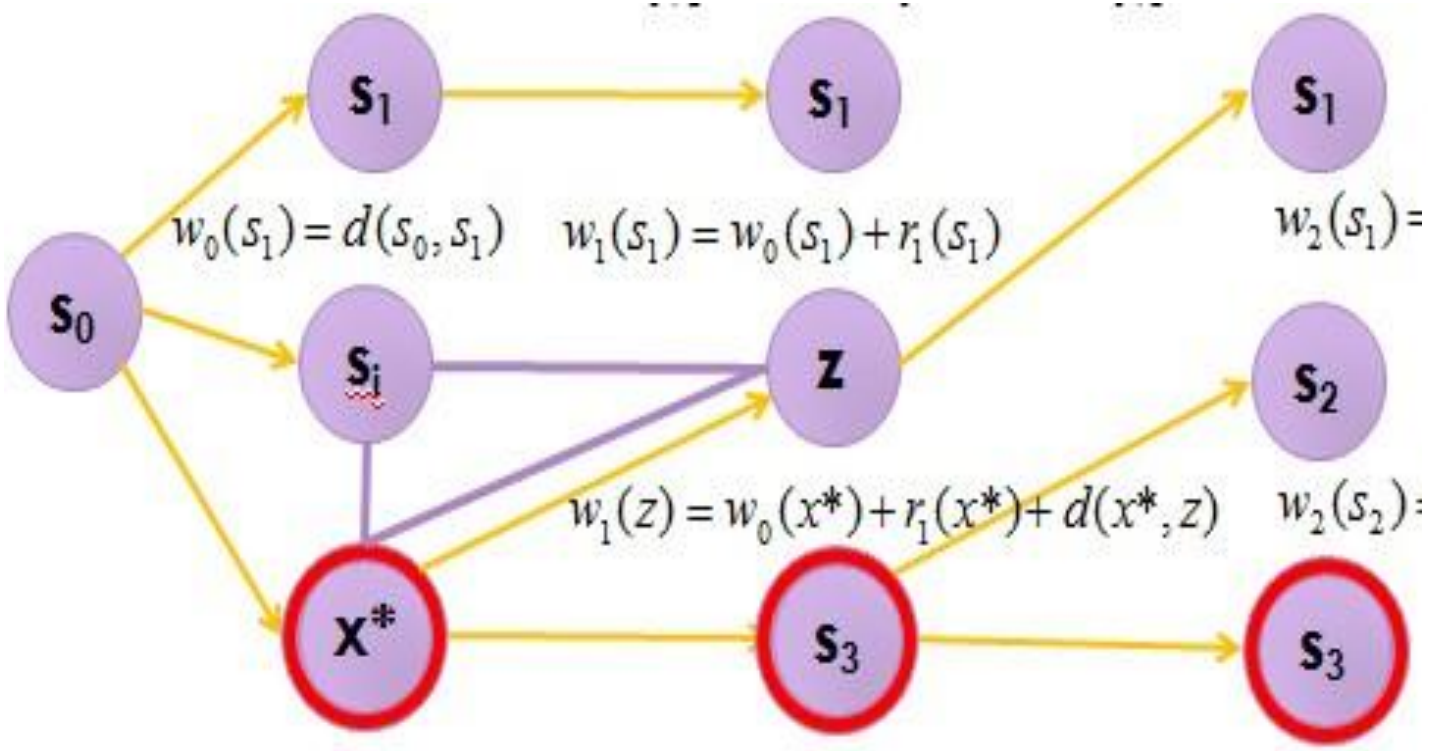
3. Let z be any element of A'
   and let x* be a state for which the minimum in equation.
   Then we have $w_{i+1}(z) = w_i(x^*) + r_{i+1}(x^*) + d(x^*, z)$



$$w_{i+1}(s) = \min_{x \in S}\{w_i(x) + r_{i+1}(x) + d(x,s)\}, w_0(s) = d(s_0, s)$$

3. Let z be any element of A'
   and let x* be a state for which the minimum in equation.
   Then we have $w_{i+1}(z) = w_i(x^*) + r_{i+1}(x^*) + d(x^*, z)$





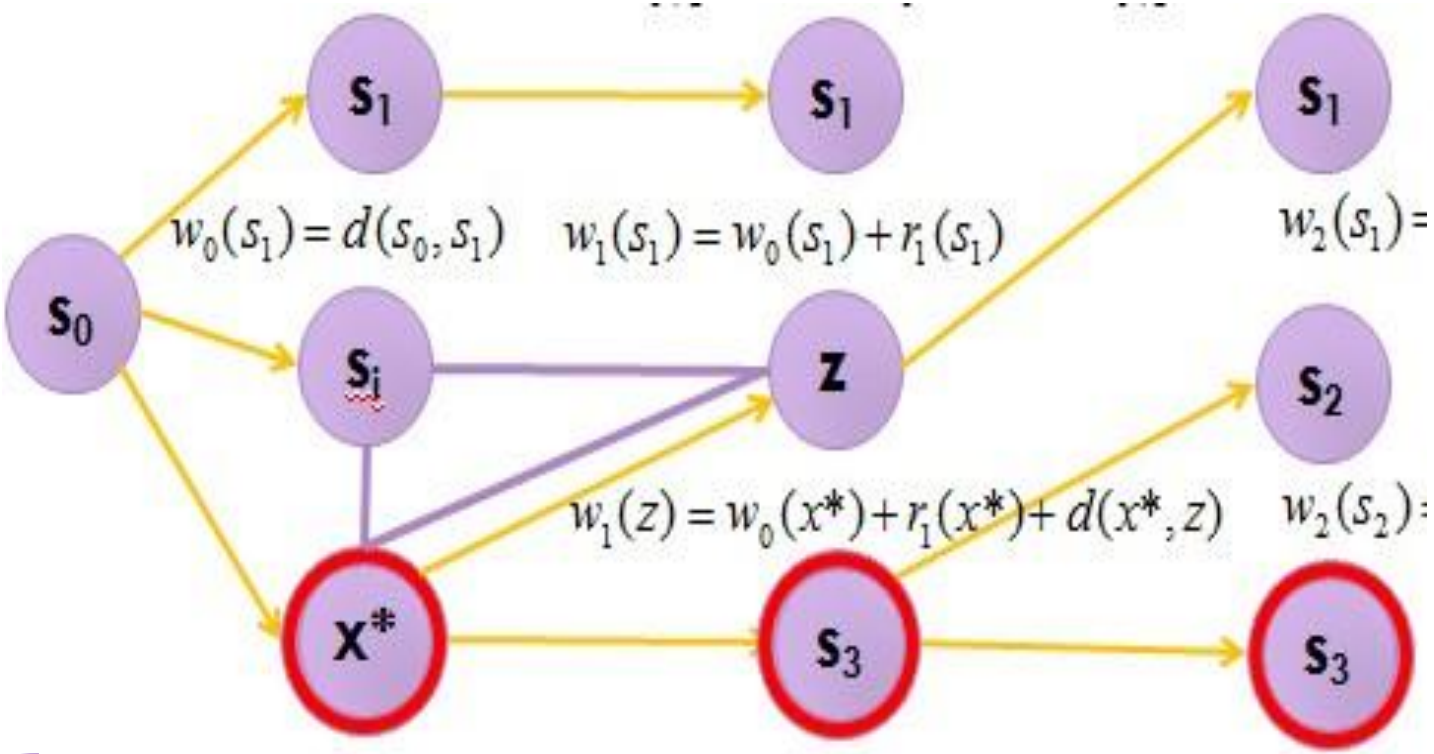
Add d(x*,si) to both sides

3. $w_{i+1}(z) = w_i(x^*) + r_{i+1}(x^*) + d(x^*, z)$

Then $w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) - d(x^*, z) + d(x^*, s_i)$

4. Using the triangle inequality

$$d(x^*, s_i) - d(x^*, z) \le d(z, s_i)$$



$$w_{i+1}(x^*) + d(x^*, s_i) \le w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i)$$ → Staying may be not optimal

$$w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) - d(x^*, z) + d(x^*, s_i)$$ → Offline or Definition of w

$$d(x^*, s_i) - d(x^*, z) \le d(z, s_i)$$ → Triangle Inequality

→ $$w_{i+1}(x^*) + d(x^*, s_i) \le w_{i+1}(z) + d(z, s_i)$$

$$w_{i+1}(x^*)+d(x^*,s_i)\le w_i(x^*)+r_{i+1}(x^*)+d(x^*,s_i)$$ → Staying may not be optimal

$$w_i(x^*)+r_{i+1}(x^*)+d(x^*,s_i)=w_{i+1}(z)-d(x^*,z)+d(x^*,s_i)$$ → Offline or Definition of w

$$d(x^*,s_i)-d(x^*,z)\le d(z,s_i)$$ → Triangle Inequality

→ $$w_{i+1}(x^*)+d(x^*,s_i)\le w_{i+1}(z)+d(z,s_i)$$

Algorithm WFA:

$$\begin{cases} s_{i+1}=\arg\min_{x\in S}\{w_{i+1}(x)+d(s_i,x)\}, \\ w_{i+1}(s_{i+1})=w_i(s_{i+1})+r_{i+1}(s_{i+1}) \end{cases}$$ → A'

Since z ∈ A', we have $z=\arg\min_{x\in S}\{w_{i+1}(x)+d(s_i,x)\}$

$$w_{i+1}(x^*)+d(x^*,s_i)\le w_i(x^*)+r_{i+1}(x^*)+d(x^*,s_i)$$ → Staying may not be optimal

$$w_i(x^*)+r_{i+1}(x^*)+d(x^*,s_i)=w_{i+1}(z)-d(x^*,z)+d(x^*,s_i)$$ → Offline or Definition of w

$$d(x^*,s_i)-d(x^*,z)\le d(z,s_i)$$ → Triangle Inequality

$$w_{i+1}(x^*)+d(x^*,s_i)\le w_{i+1}(z)+d(z,s_i)$$

$w_{i+1}(z)+d(z,s_i)$ Is the minimum

Algorithm WFA:

$$\begin{cases} s_{i+1}=\arg\min_{x\in S}\{w_{i+1}(x)+d(s_i,x)\}, \\ w_{i+1}(s_{i+1})=w_i(s_{i+1})+r_{i+1}(s_{i+1}) \end{cases}$$ → A'

Since z ∈ A', we have $z=\arg\min_{x\in S}\{w_{i+1}(x)+d(s_i,x)\}$

$$w_{i+1}(x^*)+d(x^*,s_i)\le w_i(x^*)+r_{i+1}(x^*)+d(x^*,s_i)$$ → Staying may not be optimal

$$w_i(x^*)+r_{i+1}(x^*)+d(x^*,s_i)=w_{i+1}(z)-d(x^*,z)+d(x^*,s_i)$$ → Offline or Definition of w

$$d(x^*,s_i)-d(x^*,z)\le d(z,s_i)$$ → Triangle Inequality

→ $$w_{i+1}(x^*)+d(x^*,s_i)\le w_{i+1}(z)+d(z,s_i)$$

$w_{i+1}(z)+d(z,s_i)$ Is the minimum

$$w_{i+1}(x^*)+d(x^*,s_i)=w_{i+1}(z)+d(z,s_i)$$

Algorithm WFA:

$$\begin{cases} s_{i+1}=\arg\min_{x\in S}\{w_{i+1}(x)+d(s_i,x)\}, \\ w_{i+1}(s_{i+1})=w_i(s_{i+1})+r_{i+1}(s_{i+1}) \end{cases}$$ → A'

Since z ∈ A', we have $z=\arg\min_{x\in S}\{w_{i+1}(x)+d(s_i,x)\}$

$$w_{i+1}(x^*)+d(x^*,s_i)\le w_i(x^*)+r_{i+1}(x^*)+d(x^*,s_i)$$ → Staying may not be optimal

$$w_i(x^*)+r_{i+1}(x^*)+d(x^*,s_i)=w_{i+1}(z)-d(x^*,z)+d(x^*,s_i)$$ → Offline or Definition of w

$$d(x^*,s_i)-d(x^*,z)\le d(z,s_i)$$ → Triangle Inequality

→ $$w_{i+1}(x^*)+d(x^*,s_i)\le w_{i+1}(z)+d(z,s_i)$$

$w_{i+1}(z)+d(z,s_i)$ Is the minimum

So $w_{i+1}(x^*)+d(x^*,s_i)=w_{i+1}(z)+d(z,s_i)$

$x^*=\arg\min_{x\in S}\{w_{i+1}(x)+d(s_i,x)\}$ → So $x^* \in$ A'

Algorithm WFA:

$$\begin{cases} s_{i+1}=\arg\min_{x\in S}\{w_{i+1}(x)+d(s_i,x)\}, \\ w_{i+1}(s_{i+1})=w_i(s_{i+1})+r_{i+1}(s_{i+1}) \end{cases}$$ → A'

Since z ∈ A', we have $z=\arg\min_{x\in S}\{w_{i+1}(x)+d(s_i,x)\}$

$$x^* = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}$$ → So $x^* \in A'$

$$w_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) + d(z, s_i)$$ → **Important !**

$$w_{i+1}(x^*) + d(x^*, s_i) \le w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i)$$ → Staying may not be optimal

$$w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) - d(x^*, z) + d(x^*, s_i)$$ → Offline or Definition of w

$$d(x^*, s_i) - d(x^*, z) \le d(z, s_i)$$ → Triangle Inequality

$$\begin{aligned} w_{i+1}(x^*) + d(x^*, s_i) &\le w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) \\ &= w_{i+1}(z) - d(x^*, z) + d(x^*, s_i) \\ &\le w_{i+1}(z) + d(z, s_i) \end{aligned}$$

$$x^* = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}$$ → So $x^* \in A'$

$$w_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) + d(z, s_i)$$ → **Important !**

$$w_{i+1}(x^*) + d(x^*, s_i) \le w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i)$$ → Staying may not be optimal

$$w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) - d(x^*, z) + d(x^*, s_i)$$ → Offline or Definition of w

$$d(x^*, s_i) - d(x^*, z) \le d(z, s_i)$$ → Triangle Inequality

$$\begin{aligned} w_{i+1}(x^*) + d(x^*, s_i) &\le w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) \\ &= w_{i+1}(z) - d(x^*, z) + d(x^*, s_i) \\ &= w_{i+1}(z) + d(z, s_i) \end{aligned}$$

$$x^* = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}$$ → So $x^* \in A'$

$$w_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) + d(z, s_i)$$ → **Important !**

$$w_{i+1}(x^*) + d(x^*, s_i) \le w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i)$$ → Staying may not be optimal

$$w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) - d(x^*, z) + d(x^*, s_i)$$ → Offline or Definition of w

$$d(x^*, s_i) - d(x^*, z) \le d(z, s_i)$$ → Triangle Inequality

$$\begin{aligned} w_{i+1}(x^*) + d(x^*, s_i) &= w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) \\ &= w_{i+1}(z) - d(x^*, z) + d(x^*, s_i) \\ &= w_{i+1}(z) + d(z, s_i) \end{aligned}$$

$$x^* = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}$$ → So $x^* \in A'$

$$w_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) + d(z, s_i)$$ → **Important !**

$$w_{i+1}(x^*) + d(x^*, s_i) = w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i)$$ → Staying may not be optimal

$$w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) - d(x^*, z) + d(x^*, s_i)$$ → Offline or Definition of w

$$d(x^*, s_i) - d(x^*, z) \le d(z, s_i)$$ → Triangle Inequality

$$\begin{aligned} w_{i+1}(x^*) + d(x^*, s_i) &= w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) \\ &= w_{i+1}(z) - d(x^*, z) + d(x^*, s_i) \\ &= w_{i+1}(z) + d(z, s_i) \end{aligned}$$

$$x^* = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}$$ → So $x^* \in A'$

$$w_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) + d(z, s_i)$$ → **Important !**

$$w_{i+1}(x^*) + d(x^*, s_i) = w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i)$$ → Staying may not be optimal

$$w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) - d(x^*, z) + d(x^*, s_i)$$ → Offline or Definition of w

$$d(x^*, s_i) - d(x^*, z) = d(z, s_i)$$ → Triangle Inequality

$$\begin{aligned} w_{i+1}(x^*) + d(x^*, s_i) &= w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) \\ &= w_{i+1}(z) - d(x^*, z) + d(x^*, s_i) \\ &= w_{i+1}(z) + d(z, s_i) \end{aligned}$$

$$x^* = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}$$

So $x^* \in A'$

$$w_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) + d(z, s_i)$$

Important !

$$w_{i+1}(x^*) + d(x^*, s_i) = w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i)$$

Staying may not be optimal

$$w_i(x^*) + r_{i+1}(x^*) + d(x^*, s_i) = w_{i+1}(z) - d(x^*, z) + d(x^*, s_i)$$

Offline or Definition of w

$$d(x^*, s_i) - d(x^*, z) = d(z, s_i)$$

Triangle Inequality

$$w_{i+1}(x^*) = w_i(x^*) + r_{i+1}(x^*)$$

x* is an element of A

Algorithm WFA:

$$\begin{cases} s_{i+1} = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}, \\ w_{i+1}(s_{i+1}) = w_i(s_{i+1}) + r_{i+1}(s_{i+1}) \end{cases}$$

Online WFA Illustrated

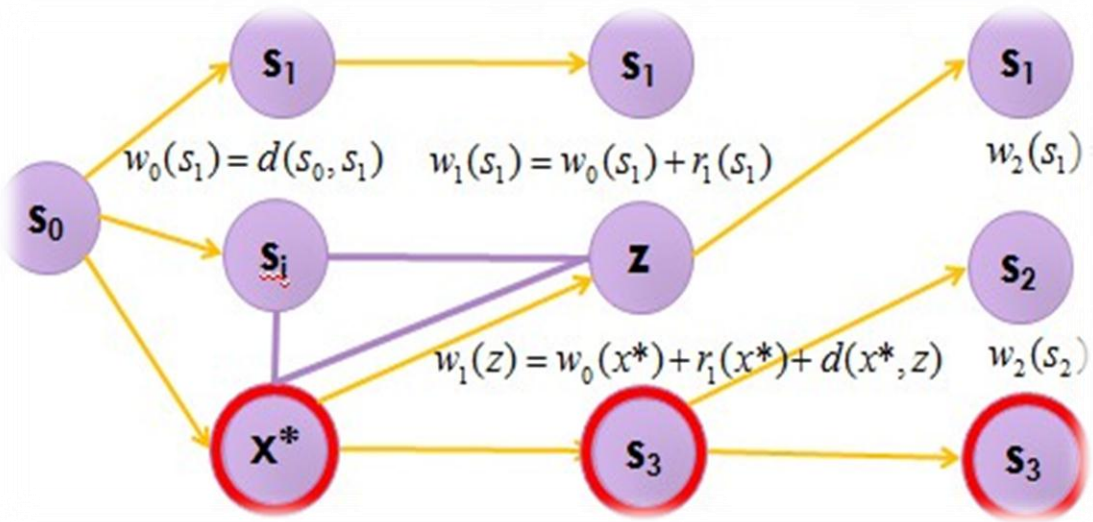
Definition (Offline): $w_{i+1}(s) = \min_{x \in S} \{w_i(x) + r_{i+1}(x) + d(x,s)\}, w_0(s) = d(s_0, s)$

Algorithm WFA:

$$\begin{cases} s_{i+1} = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}, & (1) \\ w_{i+1}(s_{i+1}) = w_i(s_{i+1}) + r_{i+1}(s_{i+1}) & (2) \end{cases}$$

$$w_{i+1}(z) = w_i(x^*) + r_{i+1}(x^*) + d(x^*, z)$$

- For any si, we can find a si+1 (or z)----O(N)
- In the figure, we can regard x* as the **procedure** of z
- We have proved that $w_{i+1}(x^*) = w_i(x^*) + r_{i+1}(x^*)$

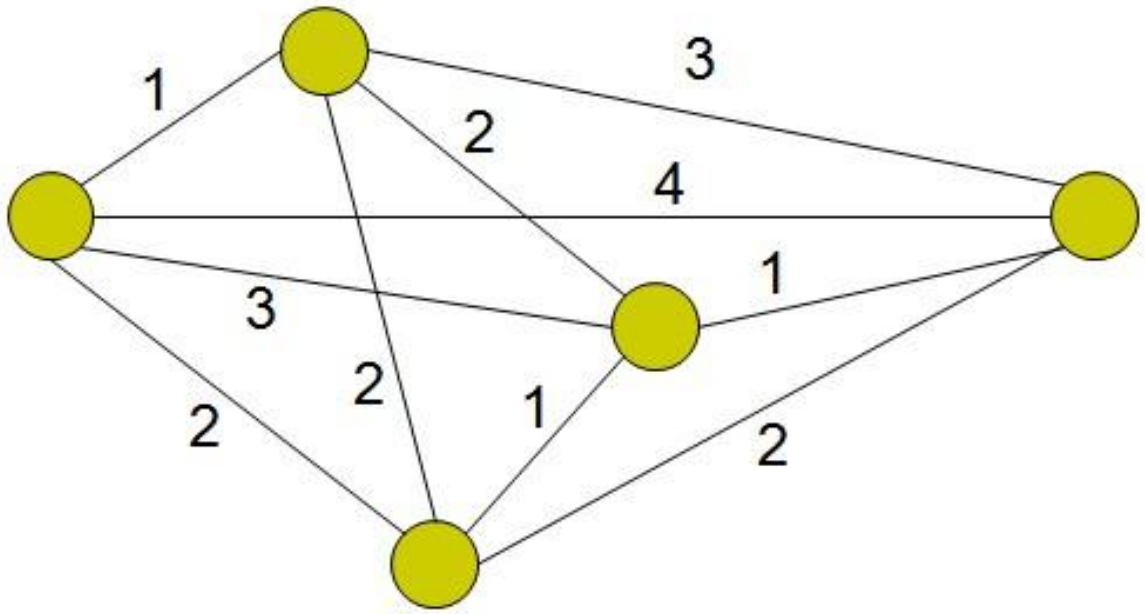


Outline

- Metrical Task System Problem
  - The MTS Work Function Algorithm
- The k-Server Problem
  - Deterministic: The K-server Work Function Algorithm
  - Randomized:
    STOC: Randomized k-Server on Hierarchical Binary Trees
- Our Model
  - Third level
    - Fourth level
      - Fifth level

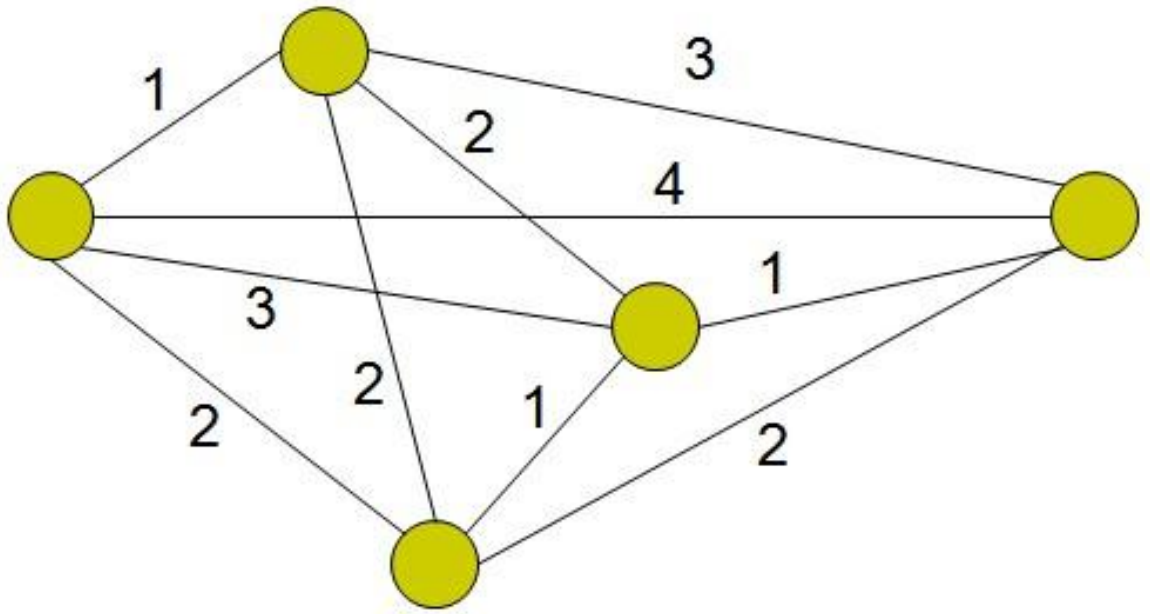
Introduction of K-Server Problem

- Think of it as a complete weighted graph
- Weight corresponds to distance between points

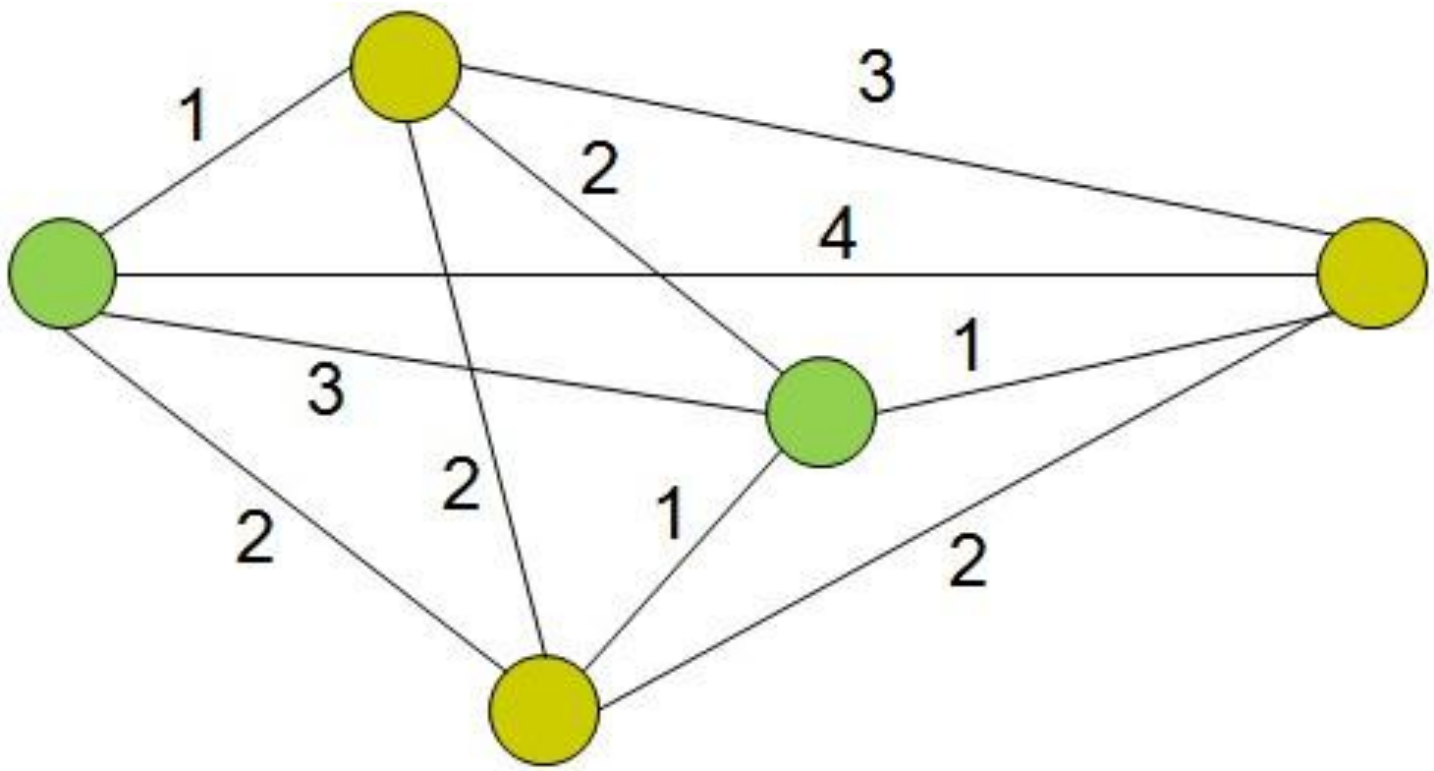


Introduction of The Model

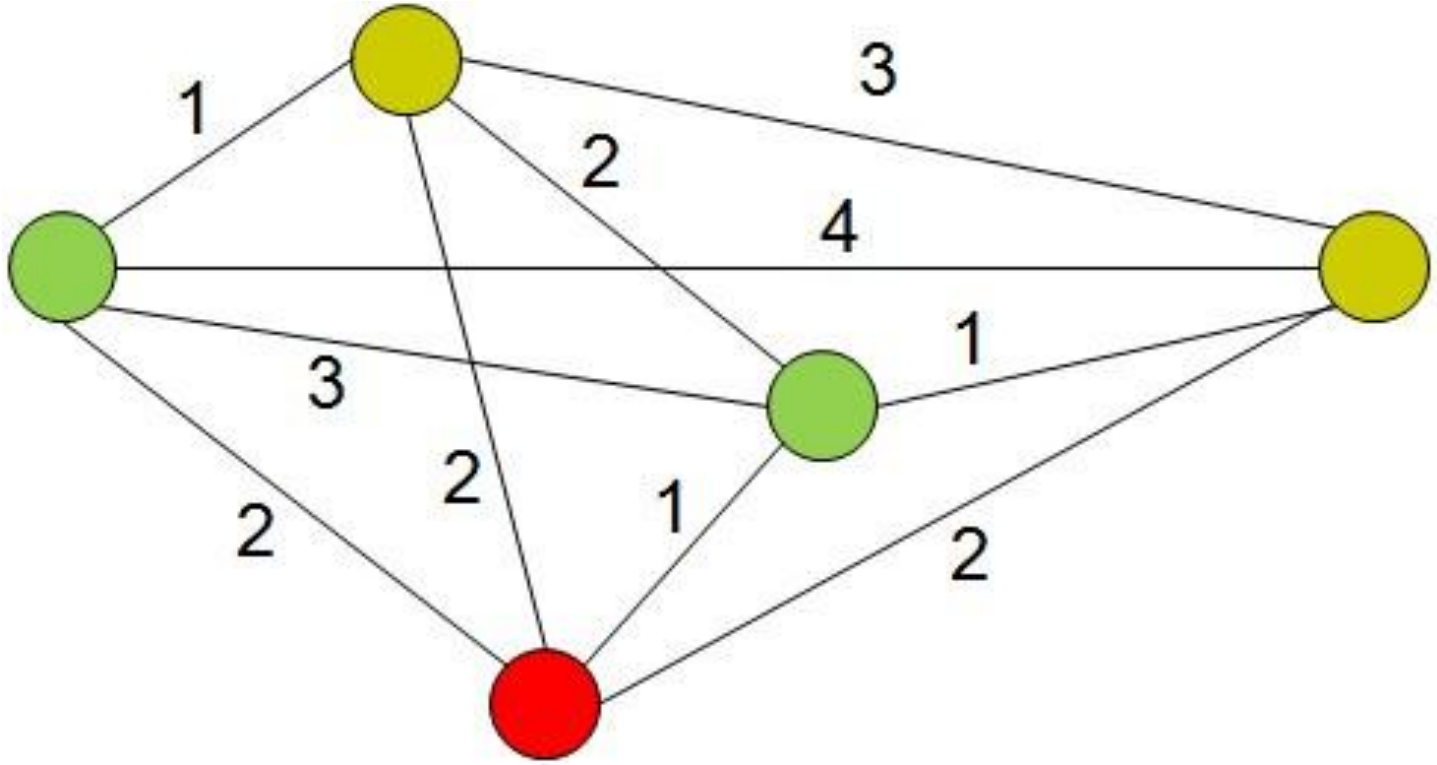
- k servers in the metric space
  - Located at particular points
- Request of service
  - Happens at the points
  - To serve the request: move a server to the point of request
  - A request sequence, where is a point in M, is a finite sequence of requests



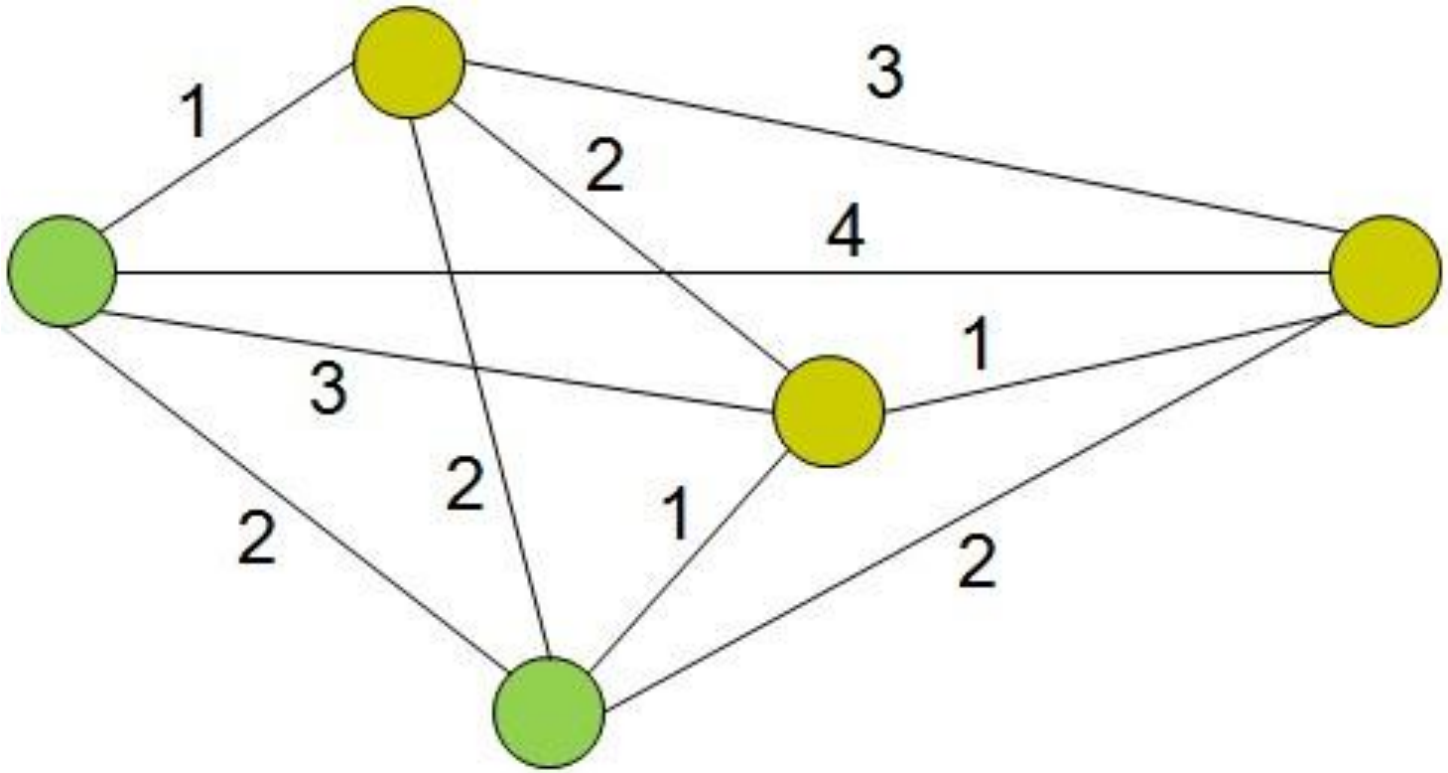
Introduction of The Model



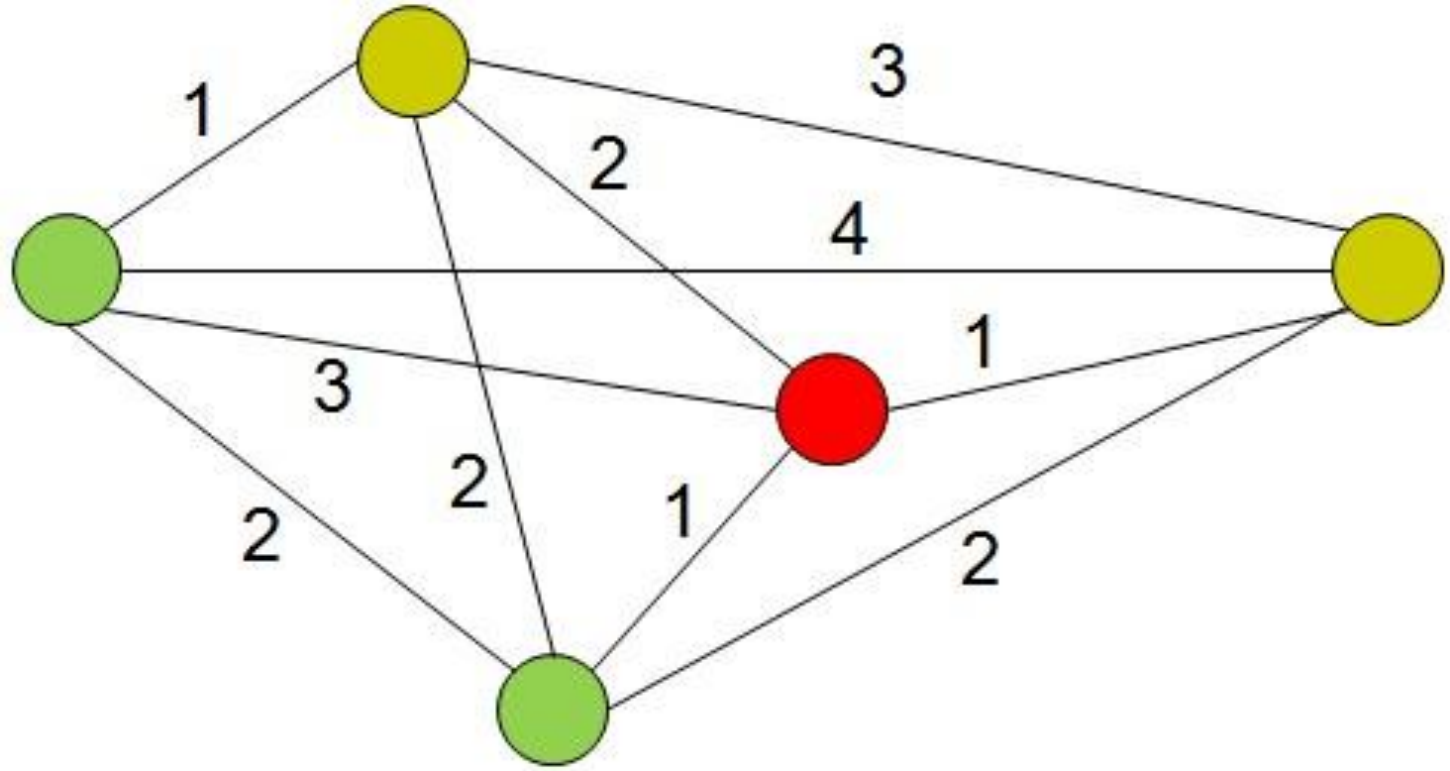
Introduction of The Model



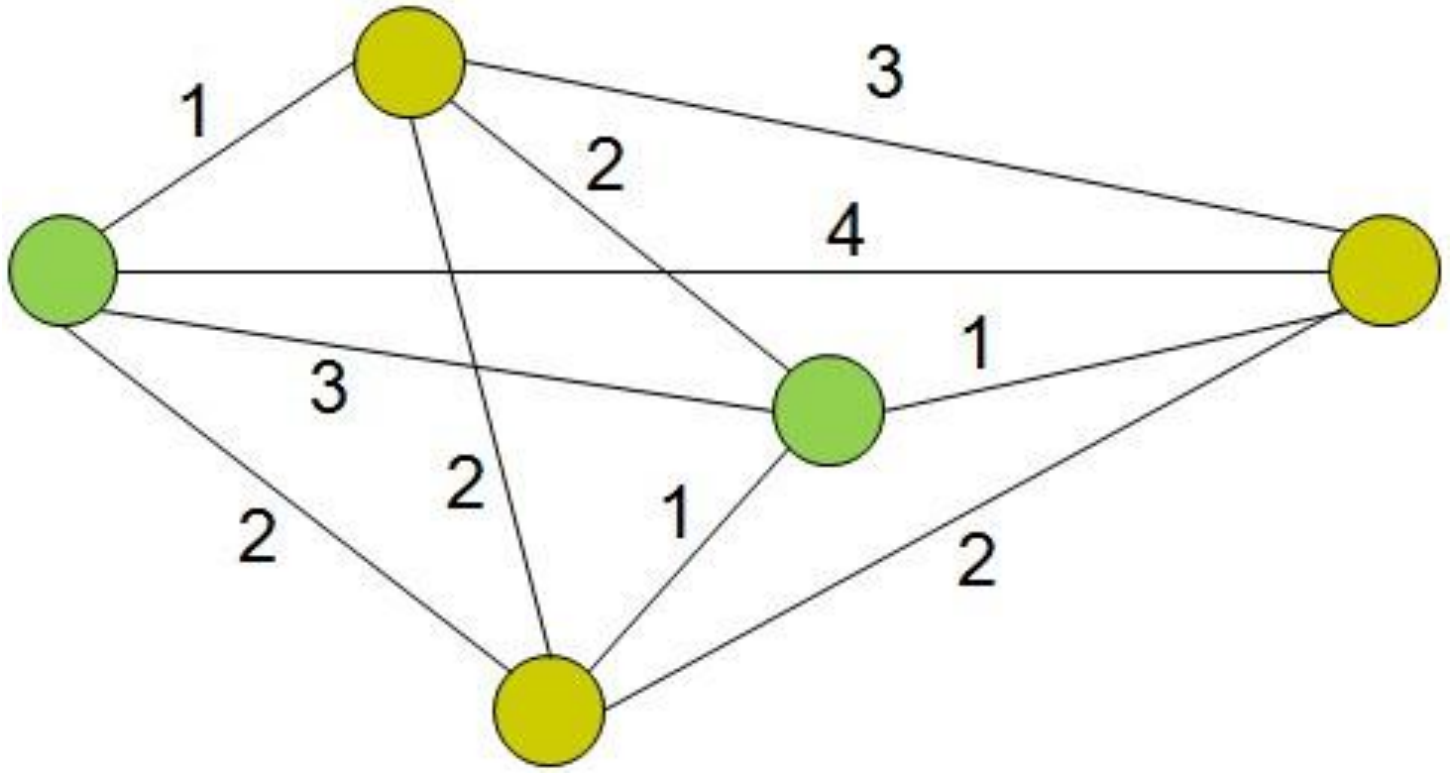
Introduction of The Model



Introduction of The Model



Introduction of The Model



Outline

- Metrical Task Problem
  - The MTS Work Function Algorithm
- The k-Server Problem
  - Deterministic: The K-server Work Function Algorithm
  - Randomized:
    STOC: Randomized k-Server on Hierarchical Binary Trees
- Our Model
  - Third level
    - Fourth level
      - Fifth level

The k-Server Work Function Algorithm

Configuration C:

A set of **k points** representing the locations of the servers.

Configuration Distance D(X,Y):

For every two configurations X and Y, D(X,Y) is the value of the minimum weight matching between X and Y.

The k-Server Work Function Algorithm

Configuration C:

A set of **k points** representing the locations of the servers.

Configuration Distance D(X,Y):

For every two configurations X and Y, D(X,Y) is the value of the minimum weight matching between X and Y.

The K-server work function $w_{\sigma_i r}(C) = w_{\sigma_i r}(C_0, C)$ is defined as the optimal offline cost of serving all the requests in σ_i

The k-Server Work Function Algorithm

The K-server work function $w_{\sigma_i r}(C) = w_{\sigma_i r}(C_0, C)$ is defined as the optimal offline cost of serving all the requests in σ_i

Given the next request r=r_i and a configuration C, the value of $w_{\sigma_i r}(C)$ is computed as follows:

- If r ∈ C, $w_{\sigma_i r}(C) = w_{\sigma_i}(C), \qquad w_0 = D(C_0, C)$
- Otherwise,

$$w_{\sigma_i r}(C) = \min_{x \in C}\{w_{\sigma_i r}(C - x + r) + d(r, x)\}$$

$$= \min_{x \in C}\{w_{\sigma_i}(C - x + r) + d(r, x)\}$$

The k-Server Work Function Algorithm

- Offline
- If r ∈ C, $w_{\sigma_i r}(C) = w_{\sigma_i}(C), \qquad w_0 = D(C_0, C)$
- Otherwise,

$$w_{\sigma_i r}(C) = \min_{x \in C}\{w_{\sigma_i r}(C - x + r) + d(r, x)\}$$

$$= \min_{x \in C}\{w_{\sigma_i}(C - x + r) + d(r, x)\}$$

- MTS: $w_{i+1}(s) = \min_{x \in S}\{w_i(x) + r_{i+1}(x) + d(x, s)\}, w_0(s) = d(s_0, s)$

The k-Server Work Funcion Algorithm

- Online
- Algorithm WFA: Let σ_i be the request sequence thus far and let C be the configuration of WFA after serving σ_i. Then, given the next request $r=r_{i+1}$, **WFA serves r with a server s ϵ C satisfying**
- $$s = \arg\min_{x \in C} \{w(C - x + r) + d(x, r)\}$$
- MTS:
$$\begin{cases} s_{i+1} = \arg\min_{x \in S} \{w_{i+1}(x) + d(s_i, x)\}, \\ w_{i+1}(s_{i+1}) = w_i(s_{i+1}) + r_{i+1}(s_{i+1}) \end{cases}$$

Outline

- Metrical Task Problem
  - The MTS Work Function Algorithm
- The k-Server Problem
  - Deterministic: The K-server Work Function Algorithm
  - Randomized:
    STOC: Randomized k-Server on Hierarchical Binary Trees
- Our Model
  - Third level
    - Fourth level
      - Fifth level

Randomized K-Server on Hierarchical Binary Trees

Aaron Cote, Adam Meyerson, and Laura poplawski

STOC '08

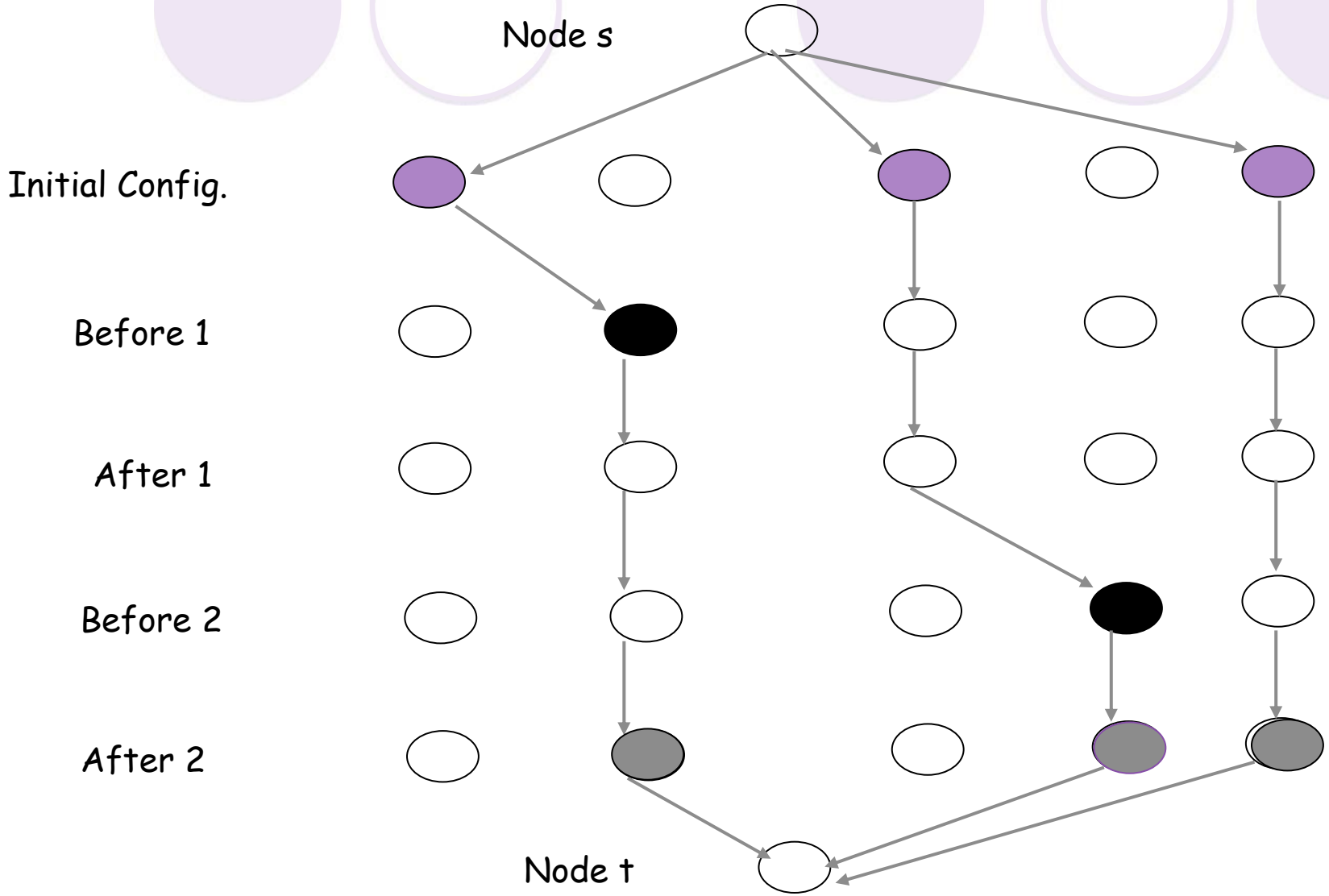
How to measure competitive ratio of randomized algorithms?

- Adaptive Adversary
- Oblivious Adversary
- Expected Competitive Ratio=

  $\max_{\text{Instances I}}$ E[cost of algorithm on I]/[cost of optimum on I]

Offline Solution

K-server →minimum cost flow


Node s
Initial Config.
Before 1
After 1
Before 2
After 2
Node t

Quasi-convexity

Let c(A) be the cost of the optimum k-server solution to some particular instance which ends with servers at locations A.

The **quasi-convexity theorem** states:

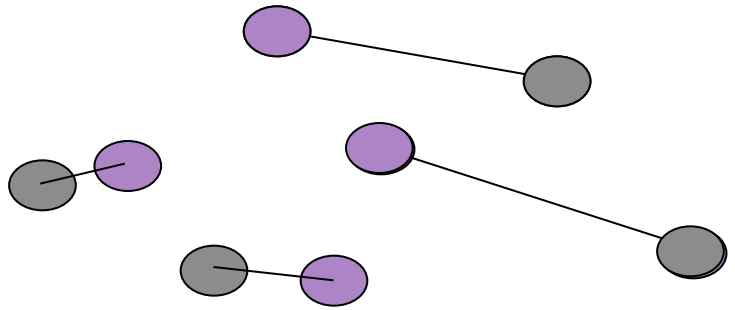
For any configurations A, B there exists a matching $\mu: A \rightarrow B$ such that for any partitioning of A into A_1, A_2 we have:

$$c(A_1 \cup \mu(A_2)) + c(\mu(A_1) \cup A_2) \leq c(A) + c(B)$$

Further, if $x \in A \cap B$ then we can guarantee $\mu(x) = x$.

Quasi-Convexity Picture

Solution A has this final server location set.



Solution B has this final server location set.

There exists a matching μ between these sets. $c(A_1 \cup \mu(A_2)) + c(\mu(A_1) \cup A_2) \leq c(A) + c(B)$

If we swap any subset of server locations according to μ.
And find the best solutions A' and B' with servers at these new locations.

The sum of $c(A') + c(B') \leq c(A) + c(B)$

New Extensions/Applications of Quasi-Convexity

- Comparing the cost of **one extra request**:
- Let $c(\rho, k)$ be the optimum k-server cost for request sequence ρ.
- Let $c(\rho r, k)$ be the optimum k-server cost with request r added.
- Obviously $c(\rho r, k) - c(\rho, k) \geq 0$.
- What happens if we have one extra server?
- **$c(\rho r, k) - c(\rho, k) \geq c(\rho r, k+1) - c(\rho, k+1)$**
- **The increase in cost is less!**
- This is intuitive, and can be proven using Quasi-Convexity.

A Hierarchical Algorithm

Divide and Conquer on a Binary Tree

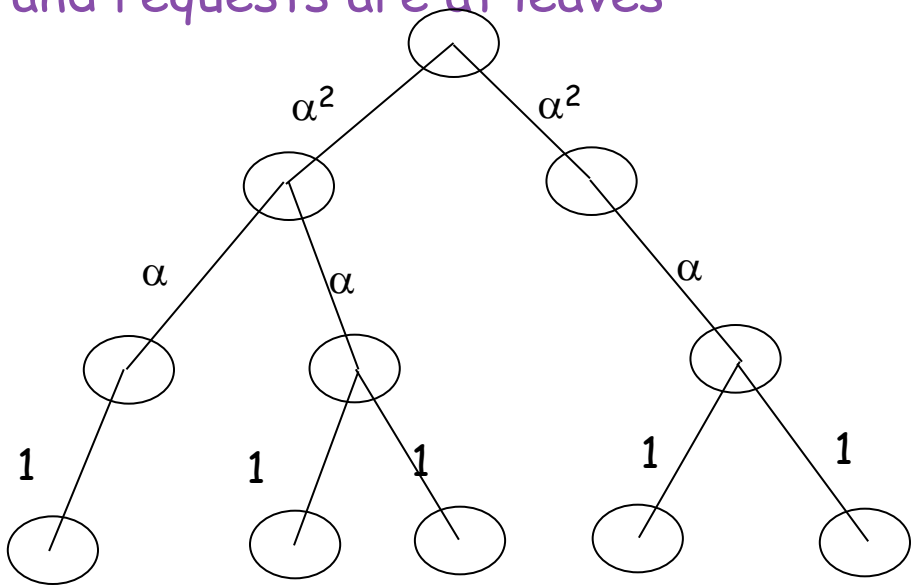
Hierarchical Binary Trees

We will consider weighted binary trees with the following properties:

(1) All leaves are at the same depth

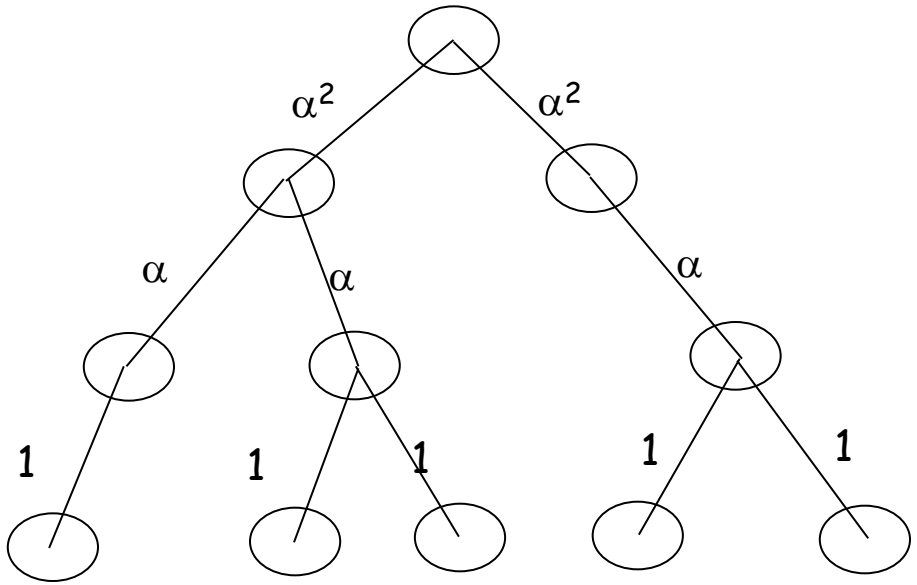
(2) Edge weights decrease geometrically by factor α

(3) All initial server locations and requests are at leaves



Metric Embedding

1.
2.

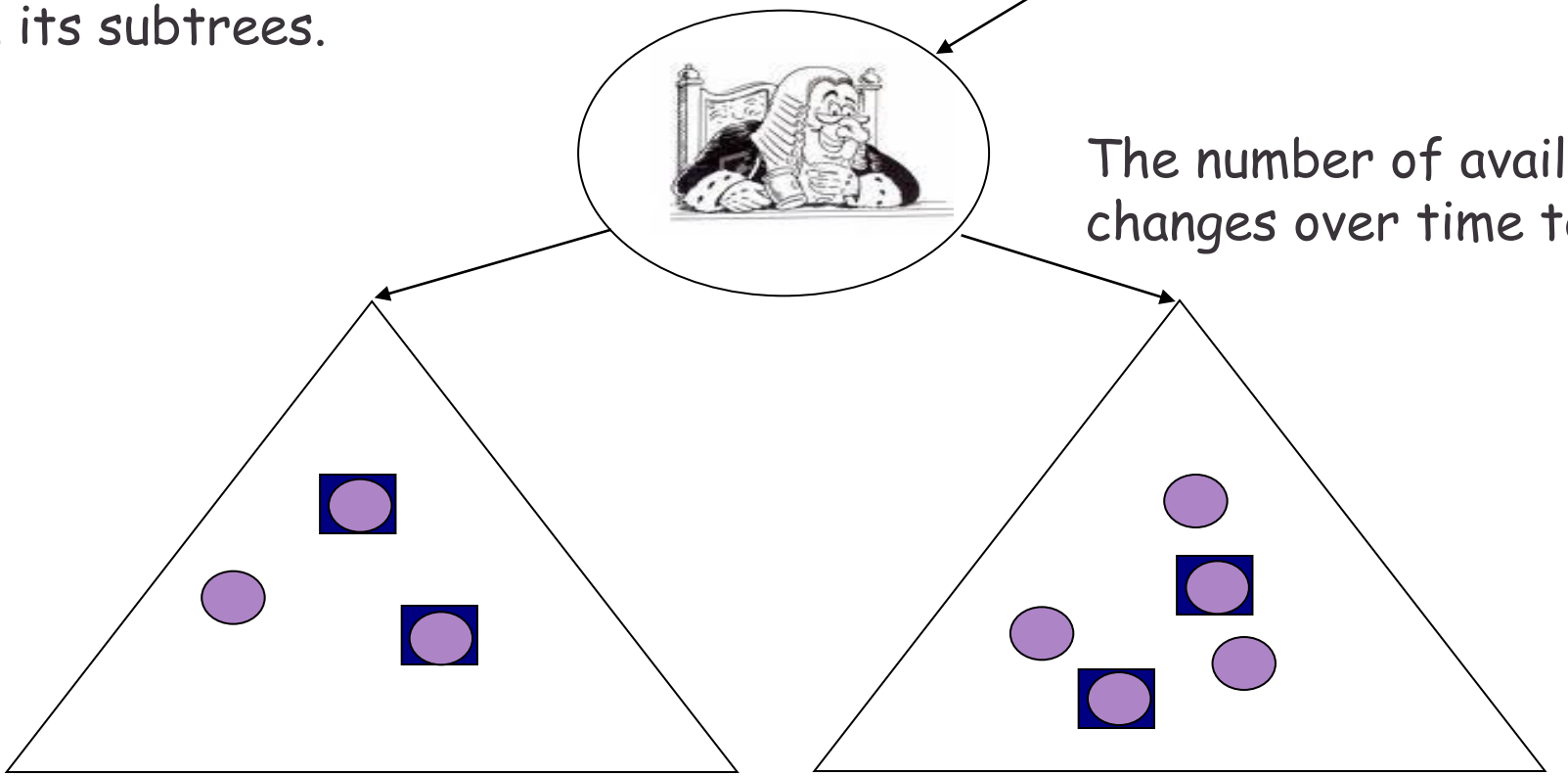


Multiple Decision-Makers

At each node we run an online algorithm.
The node "sees" requests in its subtrees.

The node must partition servers between its subtrees.

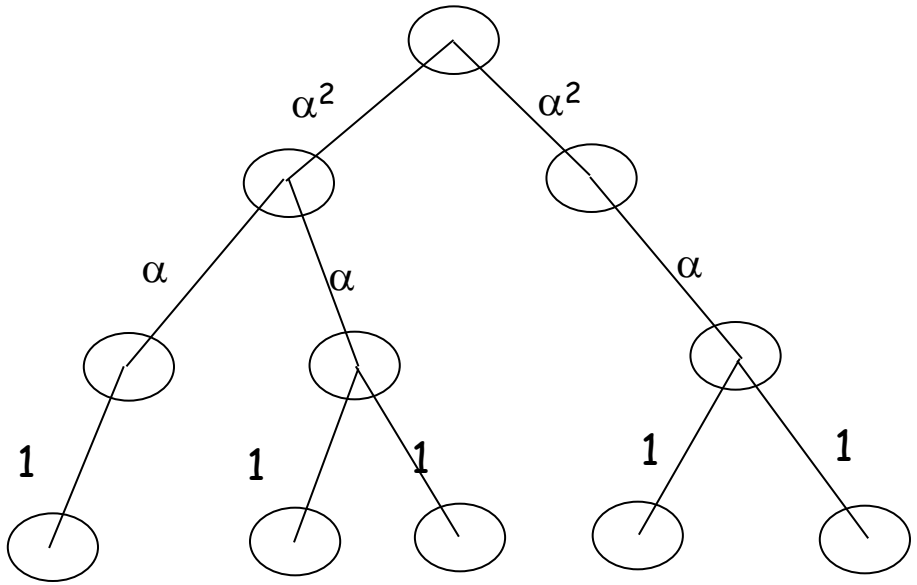
5 Servers available.
4 Servers available.



The number of available servers changes over time too!

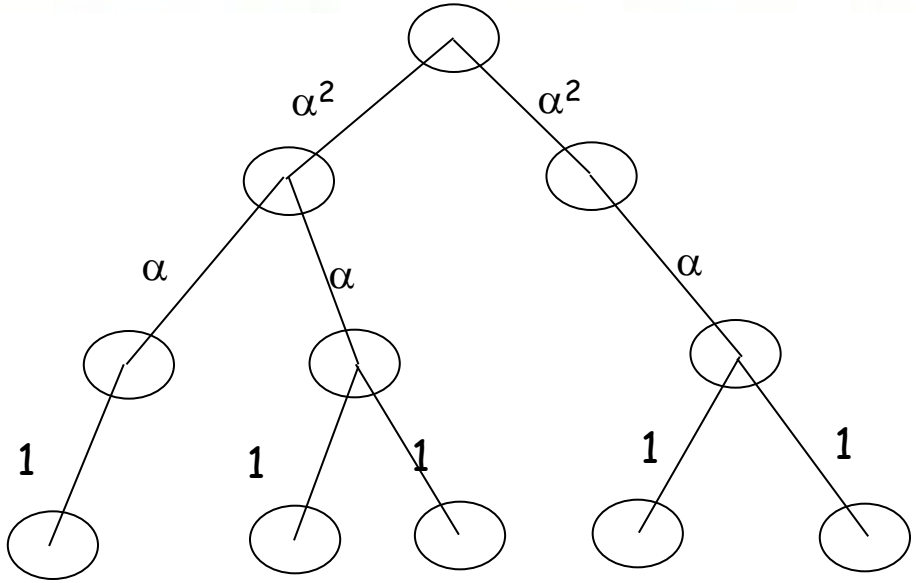
The Cost of The Solution

$c(T, d_T, C[0], \rho, \kappa) =$
$\Sigma_{t=1}^{m} d_T(C[t], C[t+1]) + \delta g(\kappa)$ where $g(\kappa) = \Sigma_t |\kappa[t] - \kappa[t+1]|$.



Divide and Conquer

$c(T, d_T, C[0], \rho, \kappa) =$
$\Sigma_{t=1}^{m} d_T(C[t], C[t+1]) + \delta g(\kappa)$ where $g(\kappa) = \Sigma_t |\kappa[t] - \kappa[t+1]|$.
$\Sigma_{i=1}^{2} c(T_i, d_T, C[0] \cap T_i, \rho \cap T_i, \kappa_i) + (\delta - \delta') \Sigma_{i=1}^{2} g(\kappa_i)$



Two Types of Cost

DEFINITION 1. *The* **Move Cost** *of vectors* κ_i *for instance* $(T, d_T, C[0], \rho, \kappa)$ *is*

$$MC = \delta \Sigma_{i=1}^{2} g(\kappa_i)$$

DEFINITION 2. *For ease of notation, we define* ρ^t *as the sequence of requests* $\{\rho[0], \rho[1], ..., \rho[t]\}$.

The **Hit Cost** *of a set of vectors* κ_i *for instance* $(T, d_T, C[0], \rho, \kappa)$ *is the sum over child trees* T_i *of the sum over all times* t *of:*

$$c(T_i, d_T, C[0] \cap T_i, \rho^t \cap T_i, \kappa_i[t])$$
$$-c(T_i, d_T, C[0] \cap T_i, \rho^{t-1} \cap T_i, \kappa_i[t])$$

DEFINITION 1. *The* **Move Cost** *of vectors* κ_i *for instance* $(T, d_T, C[0], \rho, \kappa)$ *is*

$$MC = \delta \Sigma_{i=1}^{2} g(\kappa_i)$$

DEFINITION 2. *For ease of notation, we define* ρ^t *as the sequence of requests* $\{\rho[0], \rho[1], ..., \rho[t]\}$.

The **Hit Cost** *of a set of vectors* κ_i *for instance* $(T, d_T, C[0], \rho, \kappa)$ *is the sum over child trees* T_i *of the sum over all times* t *of:*

$$c(T_i, d_T, C[0] \cap T_i, \rho^t \cap T_i, \kappa_i[t])$$
$$-c(T_i, d_T, C[0] \cap T_i, \rho^{t-1} \cap T_i, \kappa_i[t])$$

Move Cost. This is the cost of moving servers from one subtree to the other, or moving servers out of the entire tree. Because of hierarchical structure, this does not depend upon actual locations of the servers.

Hit Cost. This is the cost of (optimally) satisfying the requests given the partitionings selected by the decision-maker. We would like the change in this cost to depend only on the current partition and request, not the entire history.

Two Types of Cost

DEFINITION 1. *The* **Move Cost** *of vectors* κ_i *for instance* $(T, d_T, C[0], \rho, \kappa)$ *is*

$$MC = \delta \Sigma_{i=1}^{2} g(\kappa_i)$$

DEFINITION 2. *For ease of notation, we define* ρ^t *as the sequence of requests* $\{\rho[0], \rho[1], ..., \rho[t]\}$.

The **Hit Cost** *of a set of vectors* κ_i *for instance* $(T, d_T, C[0], \rho, \kappa)$ *is the sum over child trees* T_i *of the sum over all times* t *of:*

$$c(T_i, d_T, C[0] \cap T_i, \rho^t \cap T_i, \kappa_i[t])$$
$$-c(T_i, d_T, C[0] \cap T_i, \rho^{t-1} \cap T_i, \kappa_i[t])$$

Theorem 4. There exists a set of vectors Ki with total move cost plus hit cost $MC^*+HC^* \leq (\alpha+2)/(\alpha-1) \times c(T,dT, C[0], \rho, K)$

Theorem 5. There is a randomized online algorithm which can construct a set of vectors Ki such that, if there exists a set of vectors with move cost MC^* and hit cost HC^*, we guarantee $E[MC] \leq MC^*+HC^*$ and $E[HC] \leq MC^*+HC^*$

Theorem 6. Suppose we compute a set of vectors Ki with hit cost HC and move cost MC. Then these vectors guarantee $\sum i=1,2\ c(Ti, dT, C[0] \cap Ti, Ki) \leq HC+(1/\alpha)MC$.

Theorem 7. We can solve k-server on a binary tree with the properties described before with an expected competitive ratio $\Theta(\log\Delta)$ provided $\alpha=\Omega(\log \Delta)$

Online Algorithm

(local sub-problem)

A special metrical task system, randomized greedy

A Special Metrical Task System

- **State x** : the number of servers on the **left** subtree
- If the request p[t]ϵ T_1, then we have:

$$c_x[t] = c(T_1, d_T, C[0] \cap T_1, \rho^t \cap T_1, x) - c(T_1, d_T, C[0] \cap T_1, \rho^{t-1} \cap T_1, x)$$

- If the request p[t]ϵ T_2, then we have:

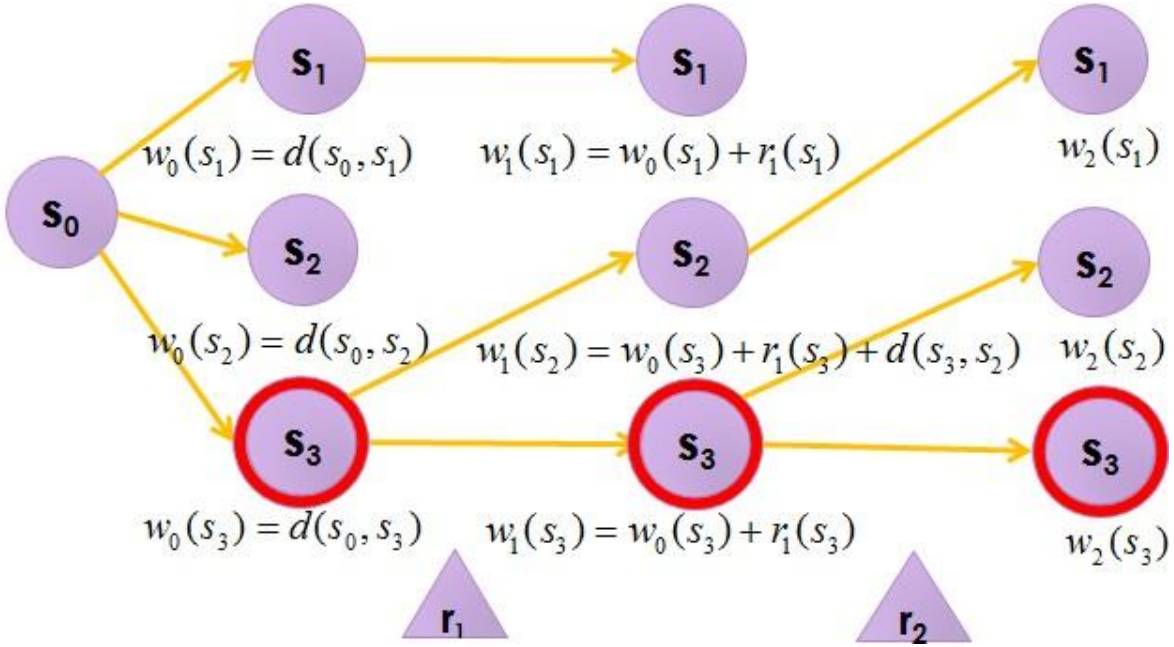
$$c_x[t] = c(T_1, d_T, C[0] \cap T_1, \rho^t \cap T_1, x) - c(T_1, d_T, C[0] \cap T_1, \rho^{t-1} \cap T_1, x)$$

Work Function Value

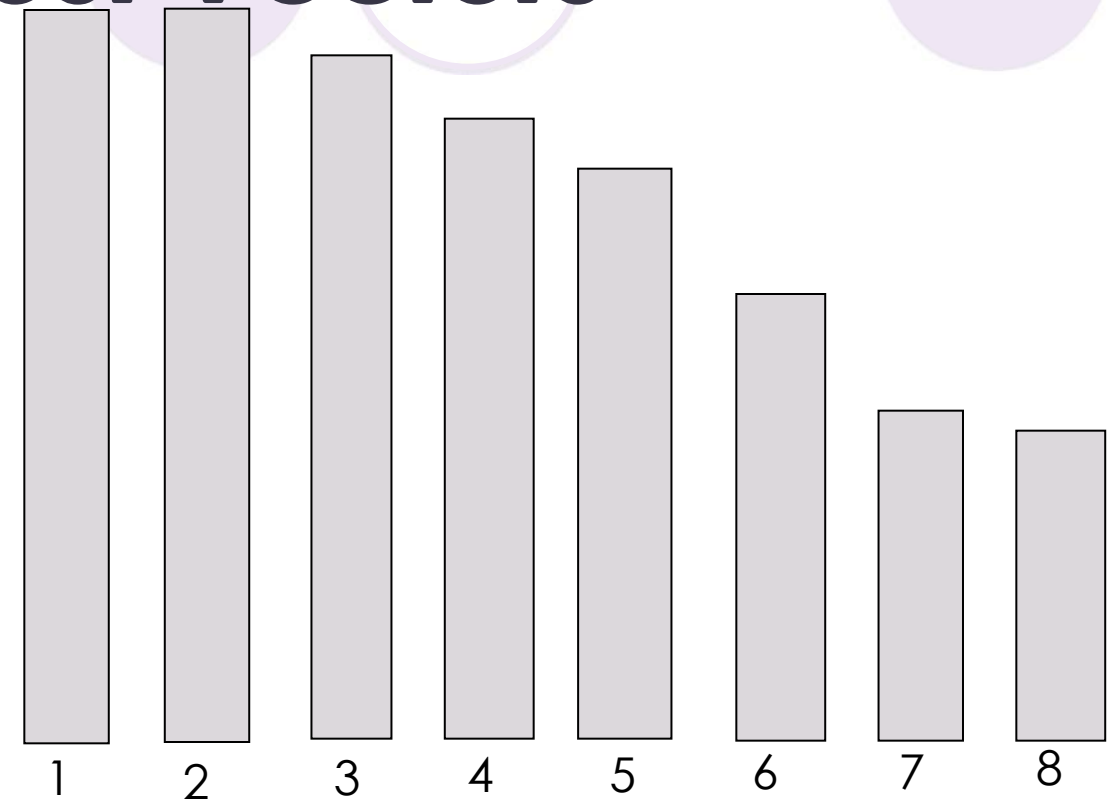
- $W_x[t]=\min\{w_x[t-1]+c_x[t], w_{x-1}[t]+2\delta, w_{x+1}[t]+2\delta\}$

- Compare:

$$w_{i+1}(s)=\min_{x\in S}\{w_i(x)+r_{i+1}(x)+d(x,s)\}, w_0(s)=d(s_0,s)$$



Cost Vectors



A cost is applied to each **state**.
Here we have a request on the left subtree.
Note the **higher** cost for **fewer** servers.

Cost Vector Property

- The cost on a particular state x at request i is given by:
- $c_x[i] = c_T(\rho[i], x) - c_T(\rho[i-1], x)$
- Note that this is just the added cost of one more request in an x-server problem.
- We again apply Quasi-Convexity to see that more servers implies a smaller change in cost!
- This implies that the cost vector will be either non-increasing (request on left subtree) or non-decreasing (request on right subtree).

Online Algorithm

- $W_x[t]=\min\{w_x[t-1]+c_x[t], w_{x-1}[t]+2\delta, w_{x+1}[t]+2\delta\}$

Algorithm 1 SUBPROBLEM(distance 2δ, initial state s_i): outputs a sequence of states as a solution to the above problem

1: generate a random number r between -1 and 1.
2: output initial state $x[0]$ corresponding to $C[0]$
3: **for** each timestep t **do**
4: current number of servers, κ_t arrives
5: cost vector $c_x[t]=(c_0[t], c_1[t], ..., c_{\kappa_t}[t])$ arrives
6: calculate the work function value for each state $w_x[t]$
7: find the state $x[t]$ that minimizes $X_x[t] = w_x[t] - xr(2\delta)$.
8: output the current state $x[t]$

Online Algorithm

- $W_x[t]=\min\{w_x[t-1]+c_x[t], w_{x-1}[t]+2\delta, w_{x+1}[t]+2\delta\}$
- $X_x[t]=w_x[t]-2rx\delta$
- Theorem 8. Algorithm 1 pays expected hit cost ≤ the total cost of the optimum offline solution to the metrical task system instance.
  - Proof. $HC \le OPT+2r\delta(x[0]-x[p])$
- Theorem 9. Algorithm 1 pays expected move cost ≤ the total cost of the optimum offline solution to the metrical task system instance.
  - Proof. $\Pr[\text{move x to y}] \le \varepsilon/4(x-y)\delta$. If we do move x to y, we pay exactly $2\delta(x-y)$

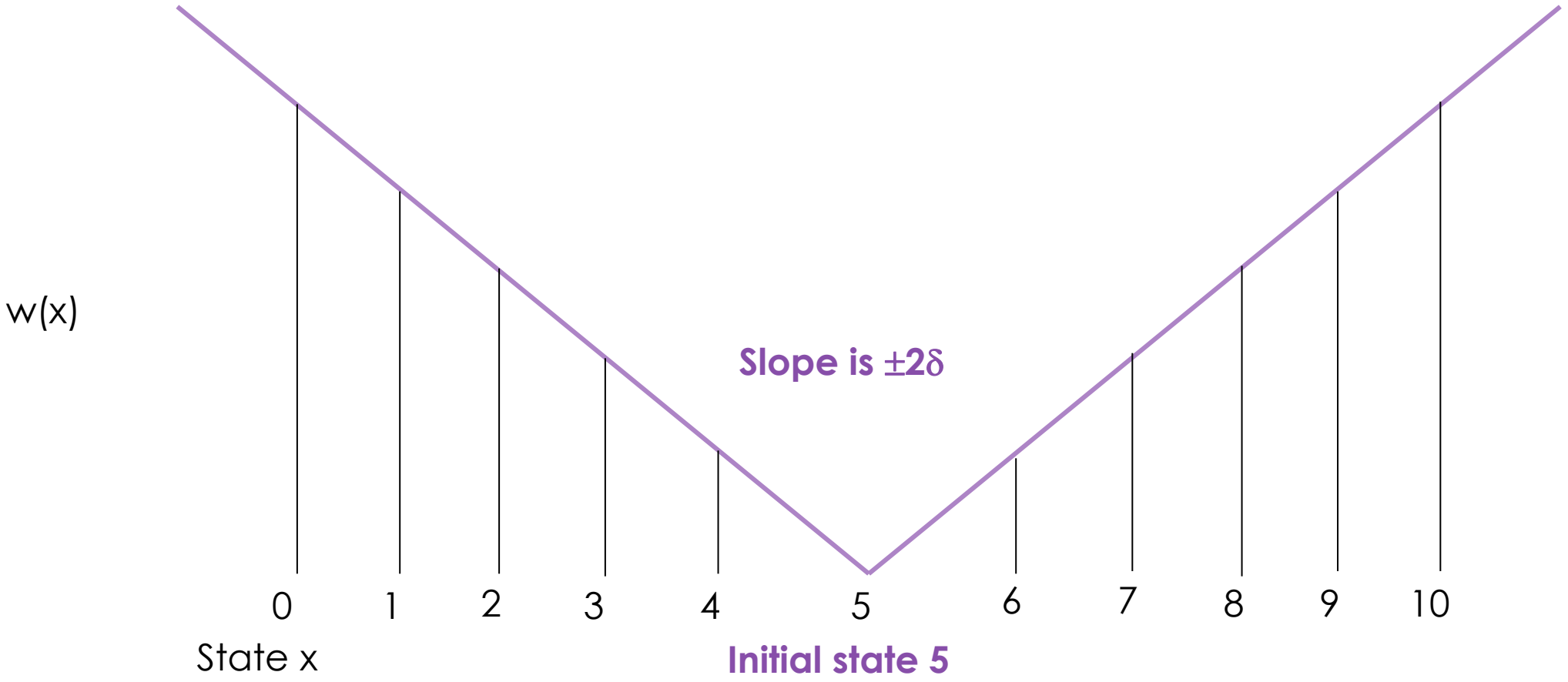
Algorithm Intuition

- Say we were to just always stay in the state with minimum work function value.

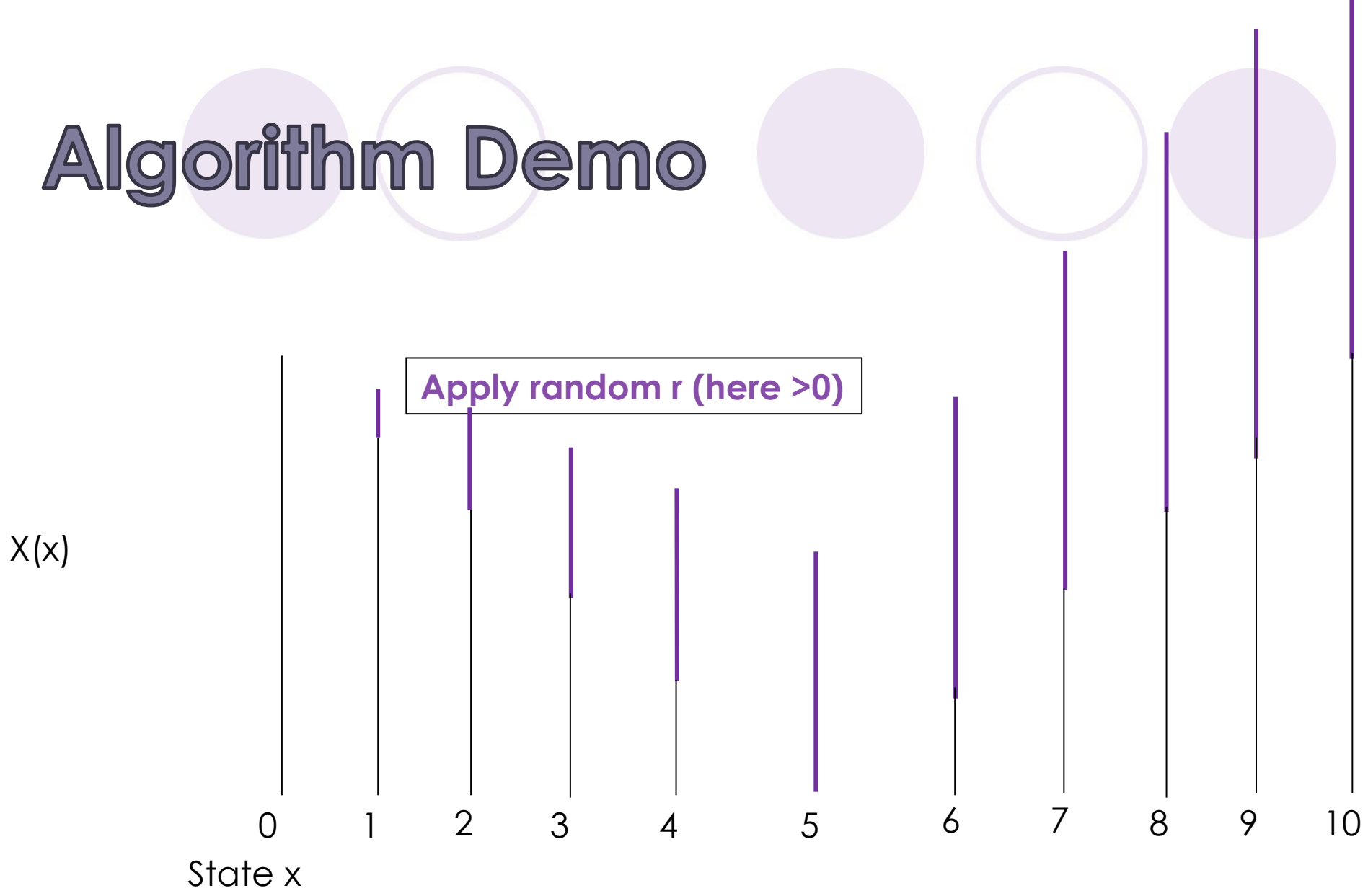
- Then our hit cost paid at each step will be the change in work function, which means our total hit cost paid is **bounded** by the work function value of the state we're in.
- **So we would have $HC \leq HC^* + MC^*$.**

- The problem is that the **move cost** can blow up without bound if we don't use randomization. The request sequence could be such that we keep switching which state has the minimum work function value by having very low-cost requests.

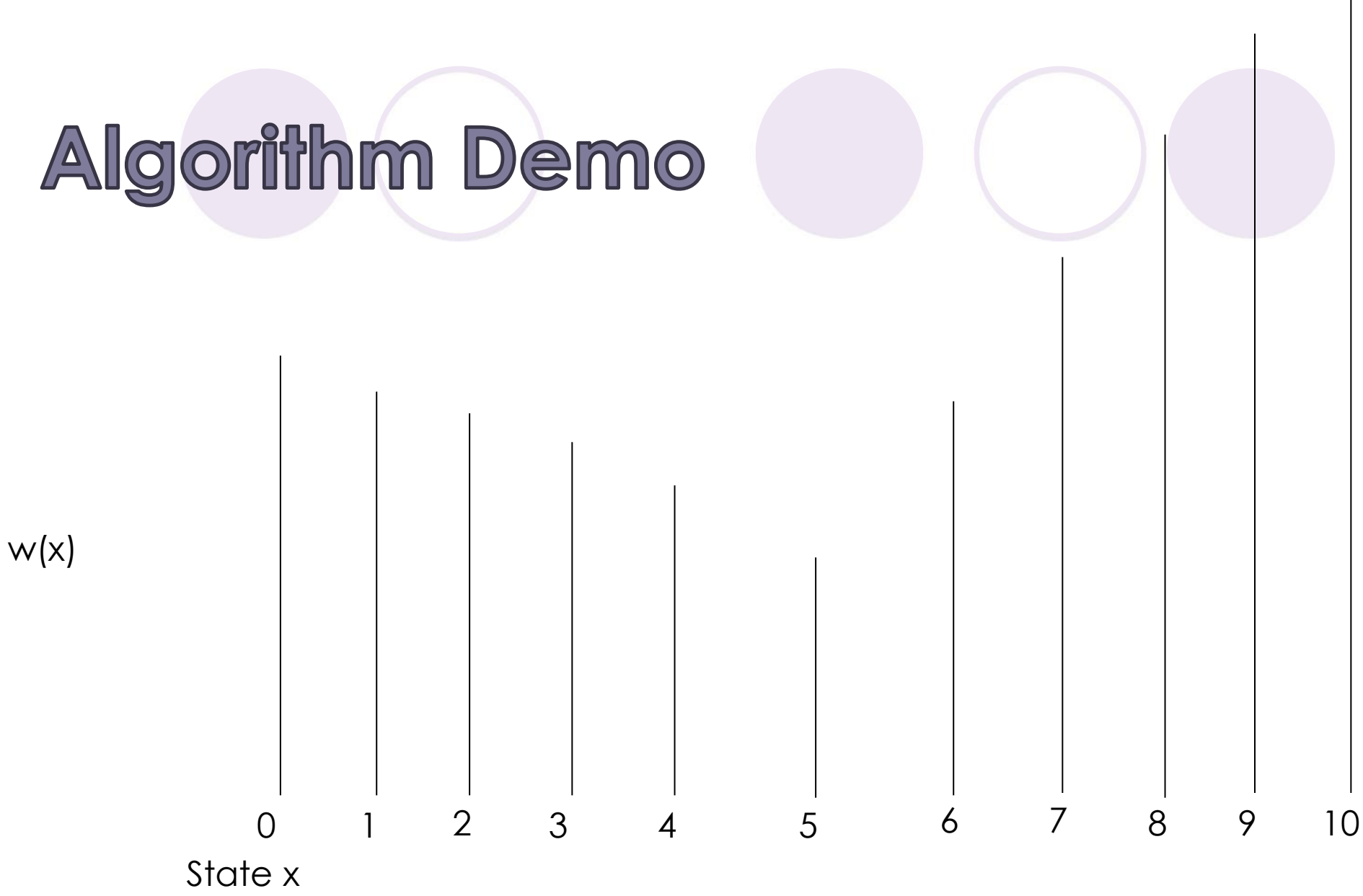
Algorithm Demo



Algorithm Demo

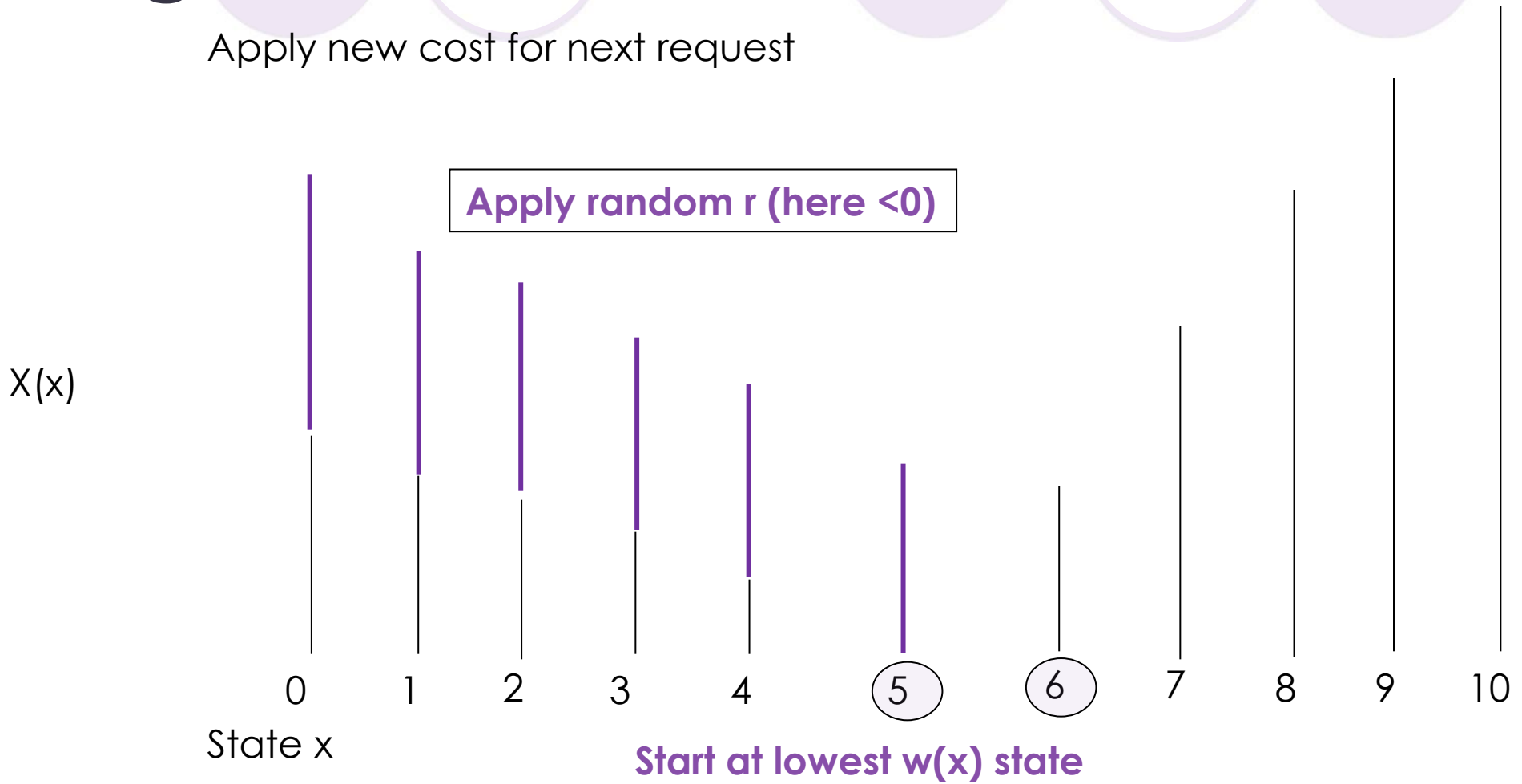


Algorithm Demo



Algorithm Demo

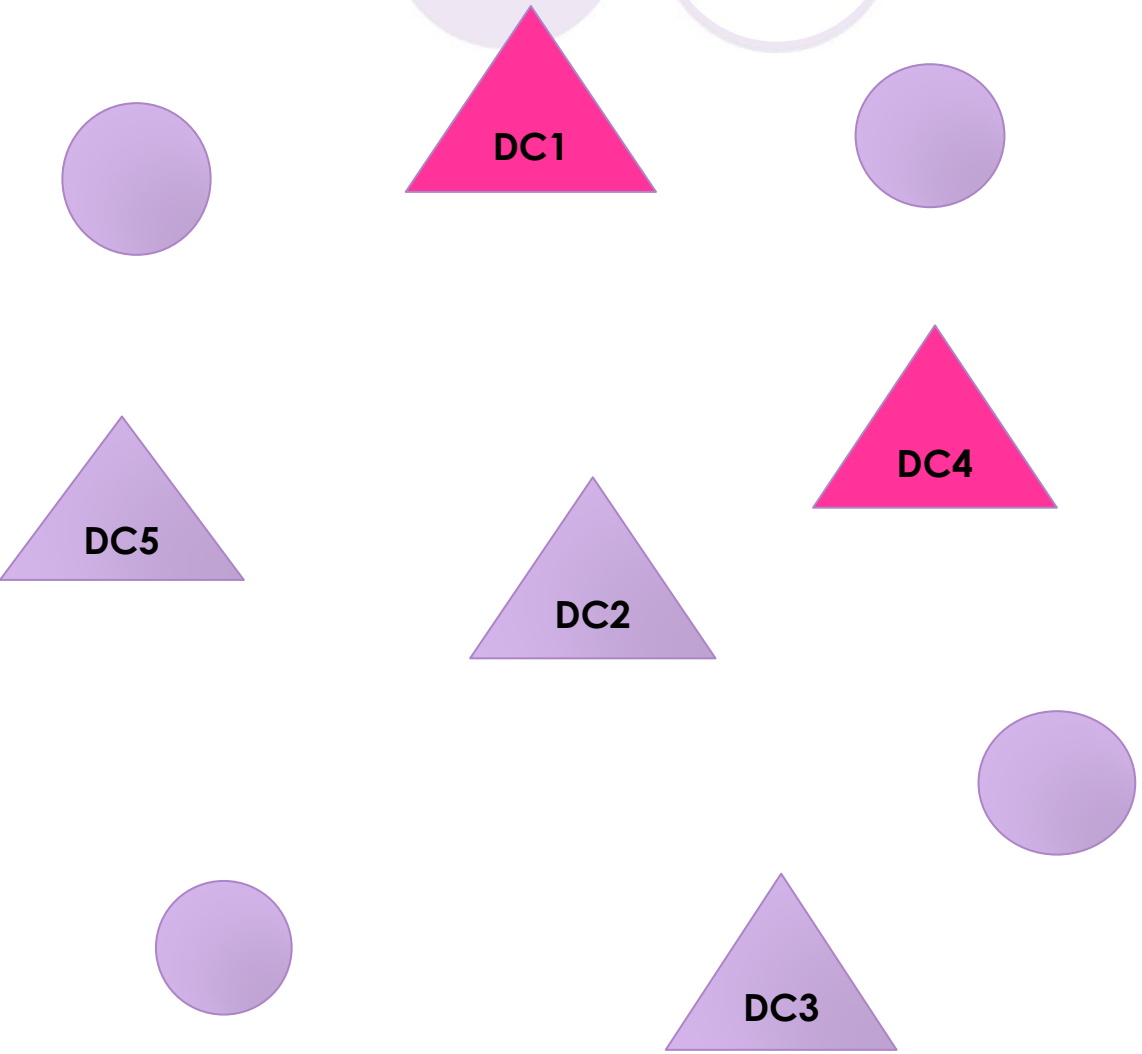
Apply new cost for next request



Conclusion of This Paper

- K-Server Definitions and History
  - Examples, applications, competitive ratio, known results
- Quasi-convexity: Solving K-Server Offline
  - Offline solution, relation to flow, useful theorems
- A Hierarchical Problem
  - Defining binary hierarchical trees, divide and conquer
- The Local Sub-problem
  - A special metrical task system, randomized greedy
- Future work: Towards a General Randomized Algorithm
  - Non-binary hierarchical trees extend to finite metrics
  - Can we make this approach work for non-binary trees?
  - Infinite metrics and "online embeddings"
  - K-Server with random requests

A New Problem



Request Point

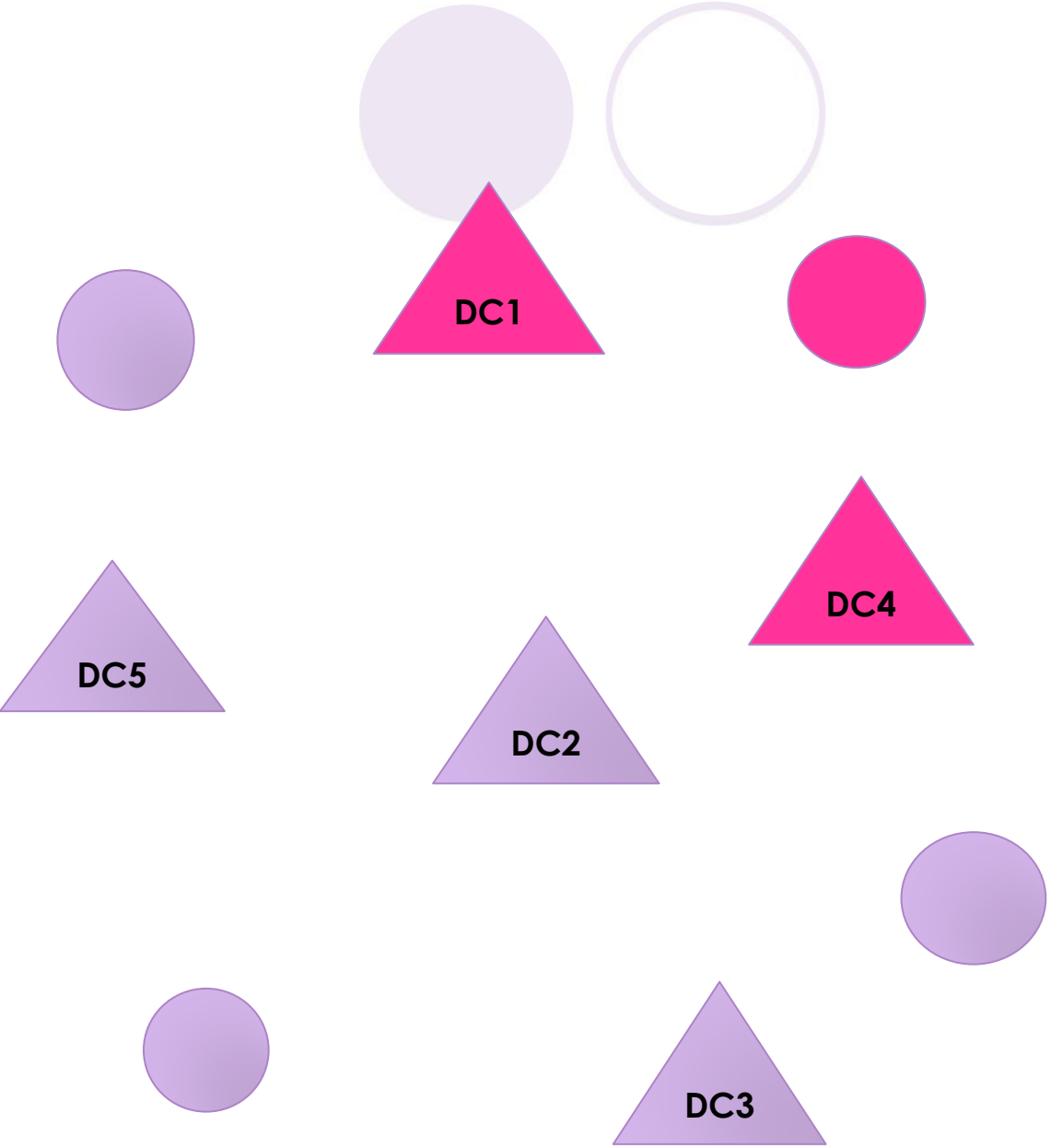
Data Center

Request for application arouses one by one at the locations of request points

We only consider one application c

We assume that at any time there are exactly 2 copies of c storage in the data centers (k=2)

No capacity limitation of each data center



Request Point

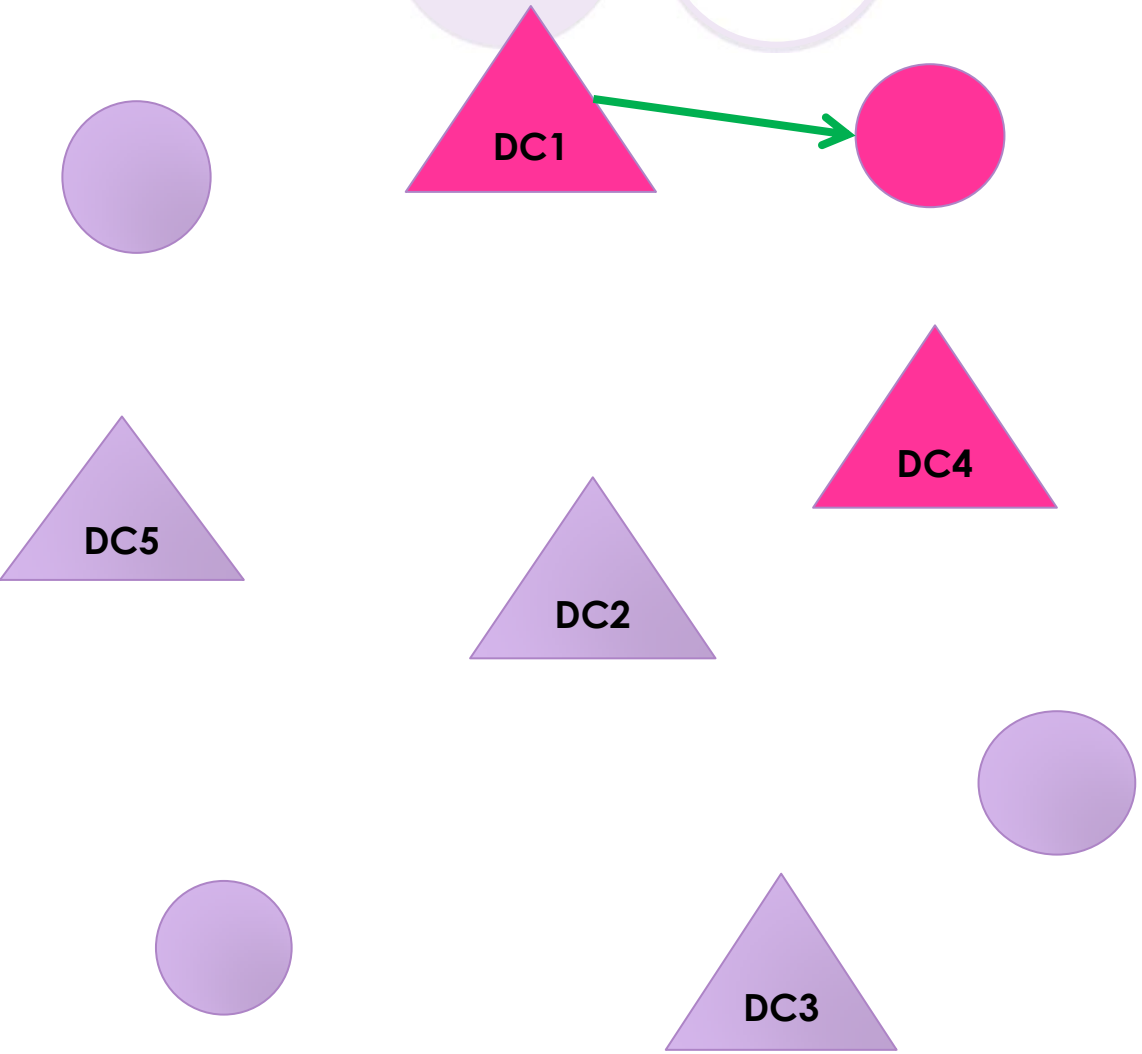
Data Center

Request for application arouses one by one at the locations of request points

We only consider one application c

We assume that at any time there are exactly 2 copies of c storage in the data centers

No capacity limitation of each data center



Request Point

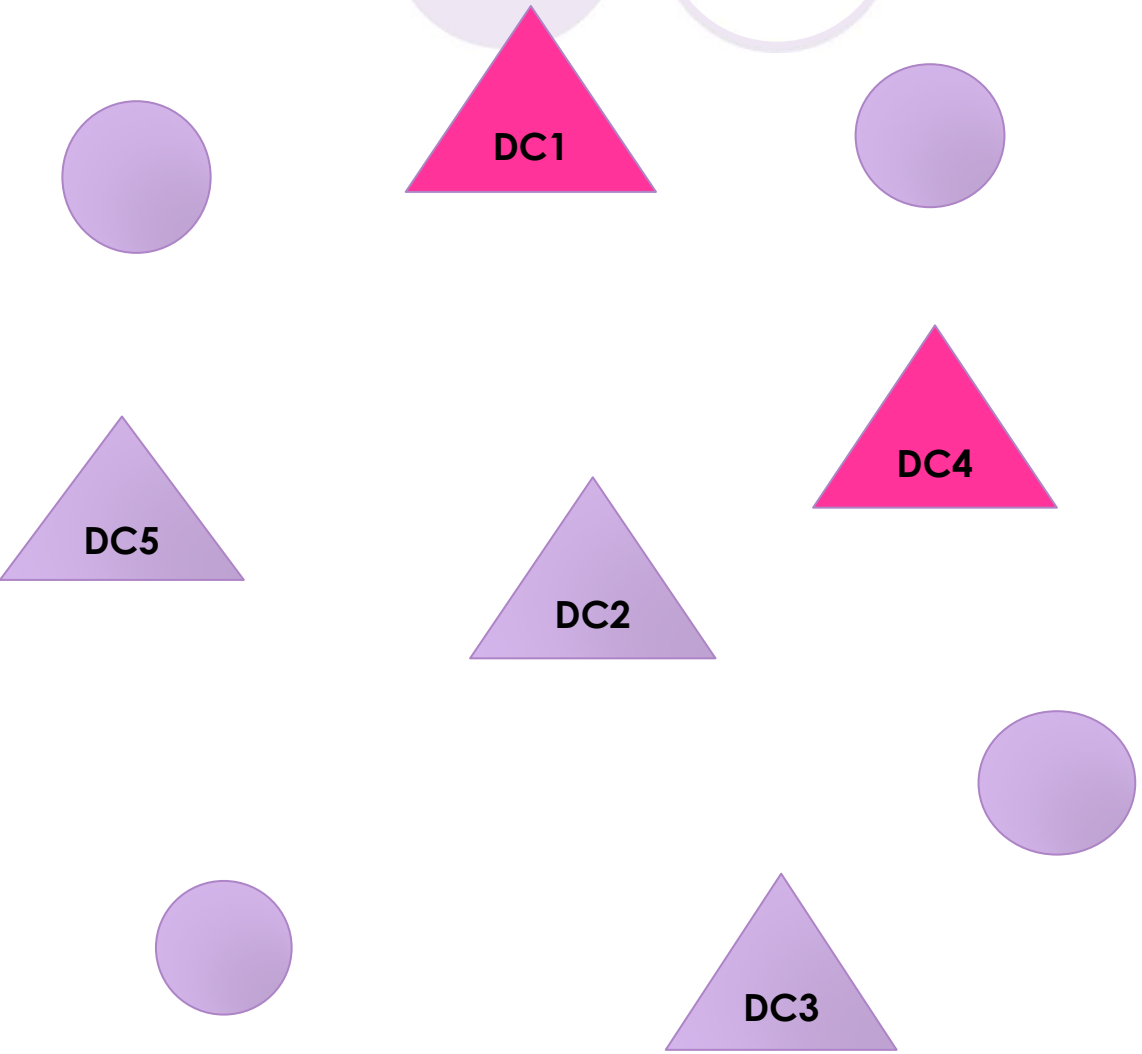
Data Center

Request for application arouses one by one at the locations of request points

We only consider one application c

We assume that at any time there are exactly 2 copies of c storage in the data centers

No capacity limitation of each data center



Request Point

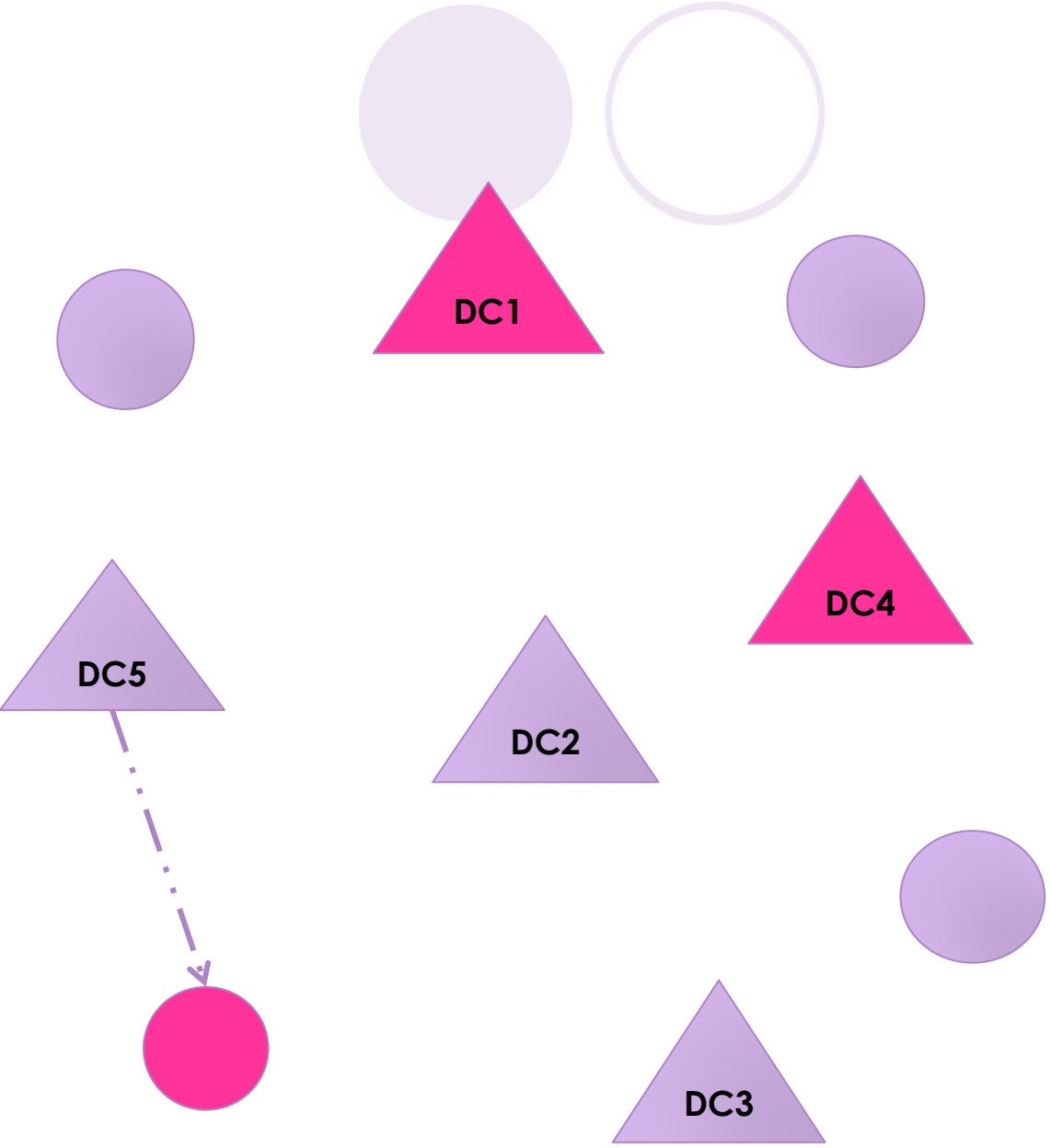
Data Center

Request for application arouses one by one at the locations of request points

We only consider one application c

We assume that at any time there are exactly 2 copies of c storage in the data centers

No capacity limitation of each data center



Request Point

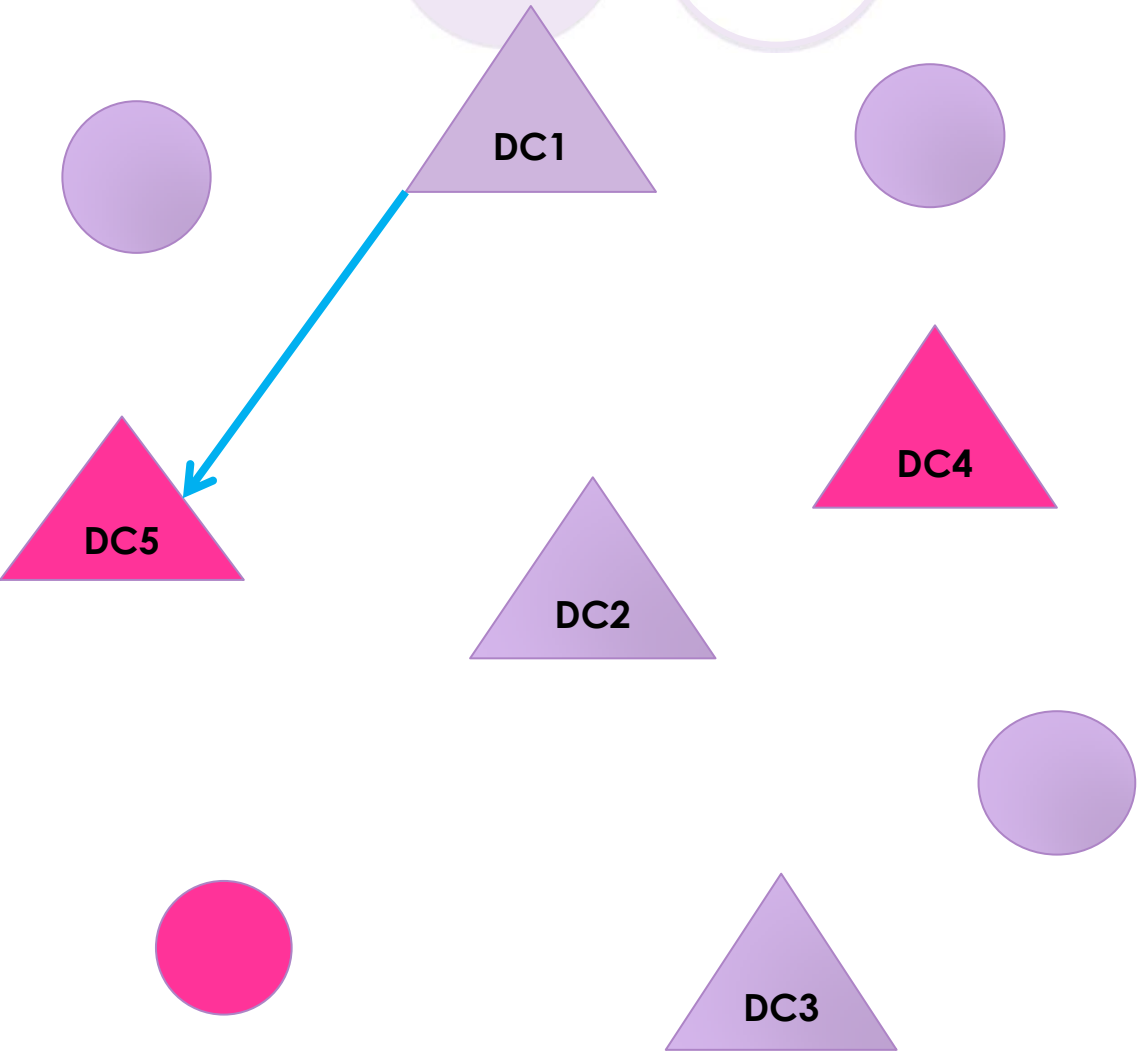
Data Center

Request for application arouses one by one at the locations of request points

We only consider one application c

We assume that at any time there are exactly 2 copies of c storage in the data centers

No capacity limitation of each data center



Request Point

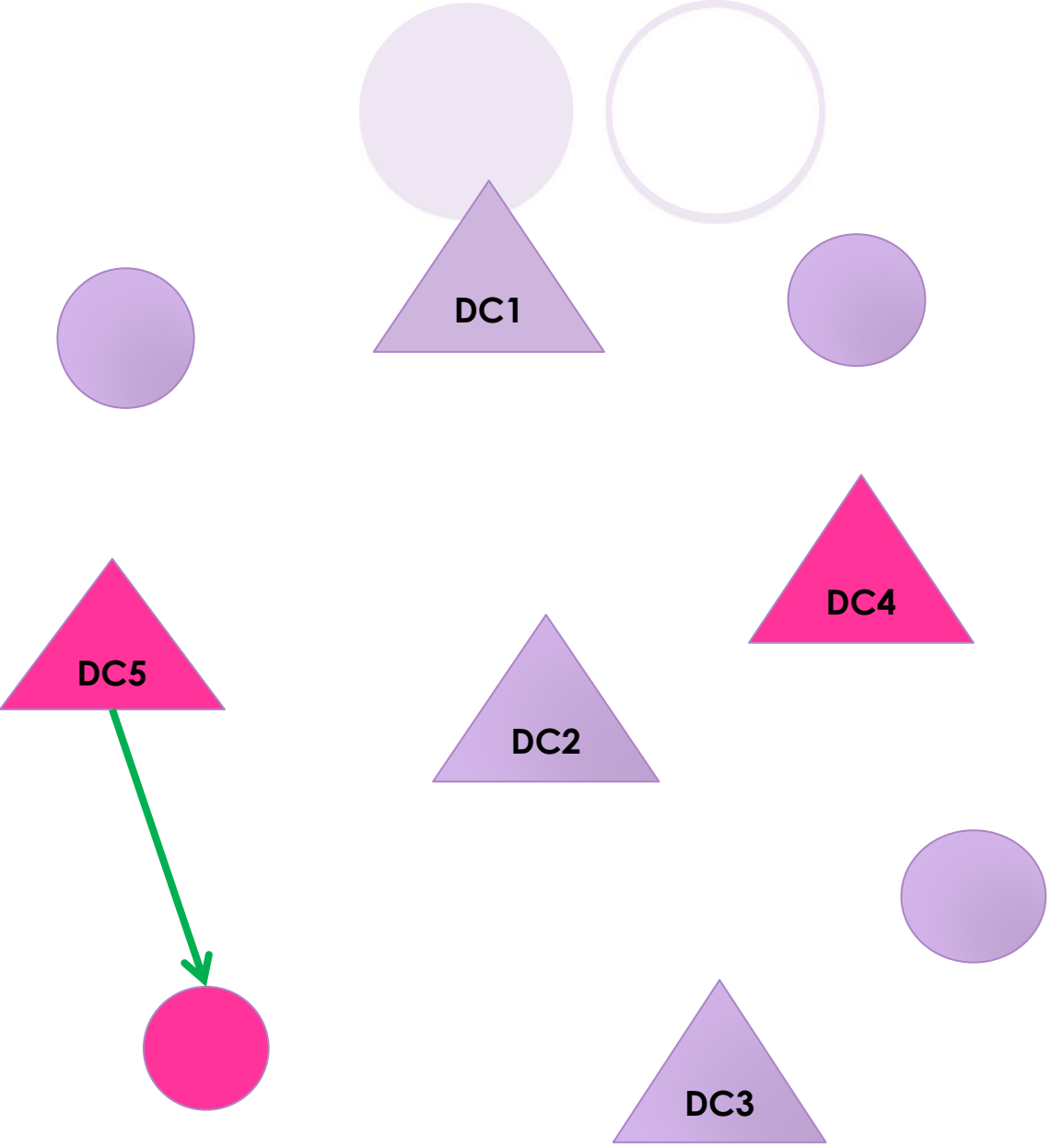
Data Center

Request for application arouses one by one at the locations of request points

We only consider one application c

We assume that at any time there are exactly 2 copies of c storage in the data centers

No capacity limitation of each data center



Request Point

Data Center

Request for application arouses one by one at the locations of request points

We only consider one application c

We assume that at any time there are exactly 2 copies of c storage in the data centers

No capacity limitation of each data center

Thank you!
Q&A