Lecture 20: Online Algorithms & k-server

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

July 20, 2021

Overview

- Online Algorithms: Randomized Lower Bounds
- k-server on a line
- Conclusion
- Acknowledgements

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s
- Let $C_A(s)$ be the cost of your online algorithm on input s

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \ldots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s
- Let $C_A(s)$ be the cost of your online algorithm on input s

Definition (Deterministic Competitive Ratio)

A deterministic online algorithm A has *competitive ratio* k (aka k-competitive) if for all inputs s, we have:

$$C_A(s) \leq k \cdot C_{opt}(s) + O(1)$$

Competitive Analysis

- Input is given as a sequence $s = s_1, s_2, \dots, s_n$ of events.
- Let $C_{opt}(s)$ be the *minimum cost* that *any algorithm* (even one that could look at the *entire input* beforehand) could achieve for input s
- Let $C_A(s)$ be the cost of your online algorithm on input s

Definition (Deterministic Competitive Ratio)

A deterministic online algorithm A has *competitive ratio* k (aka k-competitive) if for all inputs s, we have:

$$C_A(s) \leq k \cdot C_{opt}(s) + O(1)$$

Definition (Randomized Competitive Ratio)

A randomized online algorithm A has *competitive ratio* k (aka k-competitive) if for all inputs s, we have:

worst-case

expectation over the randomness of your algorithm

$$\mathbb{E}[C_A(s)] \leq k \cdot C_{opt}(s).$$

Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).

Randomized Online Algorithms & Game Theory

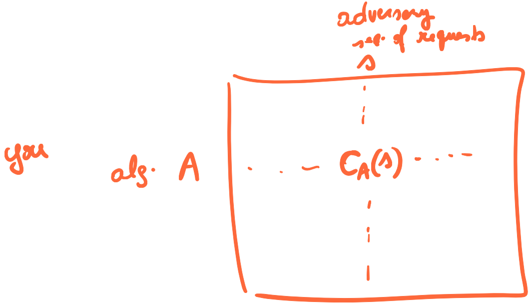
- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*

Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests

Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests
- Entry (A, s) of payoff matrix: $C_A(s)$



Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests
- Entry (A, s) of payoff matrix: $C_A(s)$
- Algorithm wants to minimize cost $-C_A(s)$
- Adversary wants to maximize it $C_A(s)$

Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests

pure strategies

- Entry (A, s) of payoff matrix: $C_A(s)$
- Algorithm wants to minimize cost
- Adversary wants to maximize it
    - Randomized algorithm $\Leftrightarrow$ mixed strategies!

Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests
- Entry (A, s) of payoff matrix: $C_A(s)$
- Algorithm wants to minimize cost
- Adversary wants to maximize it
  - Randomized algorithm $\Leftrightarrow$ mixed strategies!
- As we showed in lecture 12, if one player is using *mixed strategy*, the other player has as best response a *pure strategy*

Randomized Online Algorithms & Game Theory

- Think of online algorithms as being a zero-sum, two-player game between you (the algorithm) and an adversary (the entity choosing the sequence of requests).
- Each of your strategies is a different *deterministic algorithm*
- Each of adversary's strategies is a sequence of requests
- Entry (A, s) of payoff matrix: $C_A(s)$
- Algorithm wants to minimize cost
- Adversary wants to maximize it
  - Randomized algorithm $\Leftrightarrow$ mixed strategies!
- As we showed in lecture 12, if one player is using *mixed strategy*, the other player has as best response a *pure strategy*

LP duality applied to games

Theorem (Yao's minimax principle)

If for some input distribution, no deterministic algorithm is k-competitive, then no randomized algorithm is k-competitive!

randomized algorithm $\leftrightarrow$ mixed strategy for you

probability distribution over the possible sequences of requests $\leftrightarrow$ mixed strategy for adversary

(how to choose sequence of requests at random)

From Lect. 12

if you use $\underbrace{\text{randomized algorithm}}_{\text{mixed strat.}}$ then adversary can (and will) use $\underbrace{\text{pure strategy}}_{\text{worst possible req.}}$ as best response

competitiveness for rand. alg.

= if adversary uses $\underbrace{\text{prob. distribution over sequences}}_{\text{mixed strat.}}$ then you can use det. algorithm

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests

[1] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k + 1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*

[1] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen

[1] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm
   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*

[1] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm
   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*
   - Expected number of requests per fault: $k+1$

Overall $\boxed{\mathop{E}_{s}\left[C_A(s)\right] = \frac{n}{k+1}}$

[1] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm
   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*
   - Expected number of requests per fault: $k+1$
5. Offline Algorithm (OPT)
   - OPT can see the whole input beforehand (still use Farthest in Future)

[1] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm
   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*
   - Expected number of requests per fault: $k+1$
5. Offline Algorithm (OPT)
   - OPT can see the whole input beforehand (still use Farthest in Future)
   - Farthest in Future faults only after $k+1$ distinct pages seen

[1] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution*
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm
   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*
   - Expected number of requests per fault: $k+1$
5. Offline Algorithm (OPT)
   - OPT can see the whole input beforehand (still use Farthest in Future)
   - Farthest in Future faults only after $k+1$ distinct pages seen
   - Expected number of requests per fault:[1] $\Theta(k \log k)$ (see reference)

$$\mathbb{E}_s\left[C_{opt}(s)\right] = \frac{n}{\Theta(k \log k)}$$

[1] Here expectation is over the choice of input.

Lower Bound - Randomized Paging Algorithms

1. Setting: $k+1$ distinct pages, cache of size k, n requests
2. Distribution of inputs: *uniform distribution* $\frac{\mathbb{E}(C_A)}{\mathbb{E}[C_{opt}]}$ = competitive ratio
3. Equivalently: each page has probability $\frac{1}{k+1}$ of being chosen
4. Online Algorithm
   - No matter what our (*fixed*) deterministic algorithm A does, only k pages in cache, with probability $\frac{1}{k+1}$ requested page *not in memory*
   - Expected number of requests per fault: $k+1$
5. Offline Algorithm (OPT)
   - OPT can see the whole input beforehand (still use Farthest in Future)
   - Farthest in Future faults only after $k+1$ distinct pages seen
   - Expected number of requests per fault:[1] $\Theta(k \log k)$ (see reference)

Theorem

Any randomized algorithm for paging with k pages is $\Omega(\log k)$-competitive!

[1] Here expectation is over the choice of input.

- Online Algorithms: Randomized Lower Bounds
- k-server on a line
- Conclusion
- Acknowledgements

k-server Problem

- **Setup:** we are given a metric space (X, d).

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.
- Main question: which server to move?

k- server problem

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.
- Main question: which server to move?
- Cost function: *total distance travelled*

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.
- Main question: which server to move?
- Cost function: *total distance travelled*
- Goal: minimize distance travelled

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.
- Main question: which server to move?
- Cost function: *total distance travelled*
- Goal: minimize distance travelled
- Paging is special case of this problem (points of simplex)

Practice problem: prove this!

k-server Problem

- **Setup:** we are given a metric space (X, d).
- Online algorithm manages k mobile servers, each server is located at a point in X
- A request specifies a point in X, to which a server *must be moved*, unless we already have a server there.
- Main question: which server to move?
- Cost function: *total distance travelled*
- Goal: minimize distance travelled
- Paging is special case of this problem (points of simplex)
- Today's Simplification: assume X is a *line*. Think $X = \mathbb{R}$

Attempt 1: Greedy

1. Strategy: just move the server which is closest to the request to it

Attempt 1: Greedy

1. Strategy: just move the server which is closest to the request to it
2. Not competitive.

Attempt 1: Greedy

1. Strategy: just move the server which is closest to the request to it
2. Not competitive.
3. Scenario: two servers A and B, initially located at 0 and 1 respectively

Attempt 1: Greedy

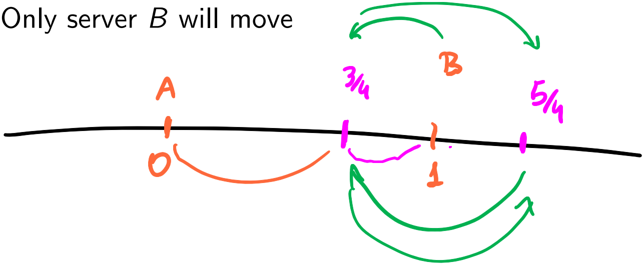
1. Strategy: just move the server which is closest to the request to it
2. Not competitive.
3. Scenario: two servers A and B, initially located at 0 and 1 respectively
4. **Requests**: sequence given by $s_{2k-1} = 3/4$, $s_{2k} = 5/4$, for $k \geq 1$

$s = \frac{3}{4}, \frac{5}{4}, \frac{3}{4}, \frac{5}{4}, \frac{3}{4}, \frac{5}{4}, \ldots$

Attempt 1: Greedy

Over n requests $\quad C_{greedy}(s) = \frac{1}{4} + \frac{1}{2}(n-1)$

1. Strategy: just move the server which is closest to the request to it
2. Not competitive.
3. Scenario: two servers A and B, initially located at 0 and 1 respectively
4. **Requests**: sequence given by $s_{2k-1} = 3/4$, $s_{2k} = 5/4$, for $k \geq 1$
5. Only server B will move



Attempt 1: Greedy

1. Strategy: just move the server which is closest to the request to it
2. Not competitive.
3. Scenario: two servers A and B, initially located at 0 and 1 respectively
4. **Requests**: sequence given by $s_{2k-1} = 3/4$, $s_{2k} = 5/4$, for $k \geq 1$
5. Only server B will move
6. Best strategy: put A on $3/4$, B on $5/4$

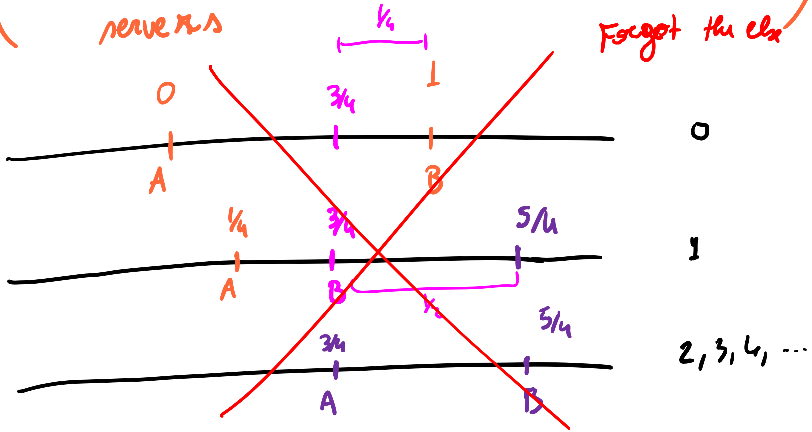
$C_{opt}(s) = \frac{3}{4} + \frac{1}{4} = 1$

Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it

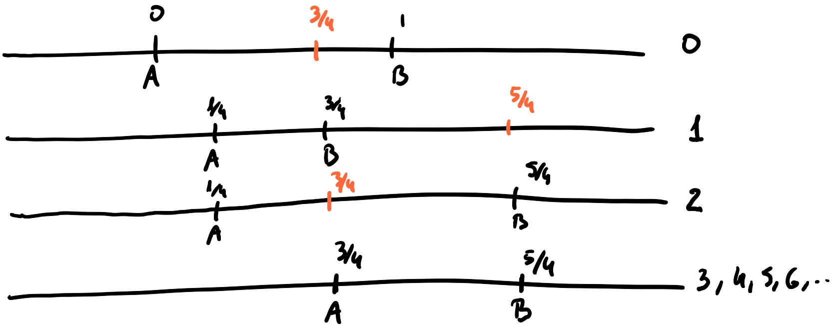
(Simplification: never query midpoint between servers)

Forget the els



Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.



Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

1. How to model OPT (offline algorithm)?

Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

1. How to model OPT (offline algorithm)?
2. Will assume that OPT algorithm moves *exactly one server at a time*.

Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

1. How to model OPT (offline algorithm)?
2. Will assume that OPT algorithm moves *exactly one server at a time*.
3. This is w.l.o.g., because can convert *any offline strategy* into a strategy that moves one server per request, by deferring moves to the future

Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

1. How to model OPT (offline algorithm)?
2. Will assume that OPT algorithm moves *exactly one server at a time*.
3. This is w.l.o.g., because can convert *any offline strategy* into a strategy that moves one server per request, by deferring moves to the future
4. How to analyze competitiveness?

Attempt 2: Double Coverage (DC)

- If request falls between two servers, move both towards request at same rate until one reaches it
- Else, just move the closest server to the request.

Theorem

For k servers, Double Coverage is k-competitive.

1. How to model OPT (offline algorithm)?
2. Will assume that OPT algorithm moves *exactly one server at a time*.
3. This is w.l.o.g., because can convert *any offline strategy* into a strategy that moves one server per request, by deferring moves to the future
4. How to analyze competitiveness?
5. Potential Function:
   - match each server from DC to a server of OPT
   - track changes as requests come

Potential Method - Recap

- In potential method, we have a potential function Φ_t for each time t

Potential Method - Recap

- In potential method, we have a potential function Φ_t for each time t
- Real cost of operation: c_t

Potential Method - Recap

- In potential method, we have a potential function Φ_t for each time t
- Real cost of operation: c_t
- Ammortized cost at time t:

$$\gamma_t = c_t + \underbrace{\Phi_t - \Phi_{t-1}}_{\text{change in potential}}$$

Potential Method - Recap

- In potential method, we have a potential function Φ_t for each time t
- Real cost of operation: c_t
- Ammortized cost at time t:

$$\gamma_t = c_t + \Phi_t - \Phi_{t-1}$$

- Total ammortized cost:

$$\begin{aligned} \sum_{t=1}^{n} \gamma_t &= \sum_{t=1}^{n} \left(c_t + \Phi_t - \Phi_{t-1} \right) \\ &= \underbrace{\Phi_n}_{\text{final potential}} - \underbrace{\Phi_0}_{\text{initial potential}} + \underbrace{\sum_{t=1}^{n} c_t}_{\text{total true cost}} \end{aligned}$$

Potential Method - Recap

- In potential method, we have a potential function Φ_t for each time t
- Real cost of operation: c_t
- Ammortized cost at time t:

$$\gamma_t = c_t + \Phi_t - \Phi_{t-1}$$

- Total ammortized cost:

$$\sum_{t=1}^{n} \gamma_t = \sum_{t=1}^{n} c_t + \Phi_t - \Phi_{t-1}$$

$$= \cancel{\Phi_n} - \Phi_0 + \sum_{t=1}^{n} c_t \geq -\Phi_0 + \sum_{t=1}^{n} c_t$$

- If potential function is always *non-negative* $\Phi_t \geq 0 \quad \forall t$

and if $\Phi_0 = 0$

$$\sum_{t=1}^{n} c_t \leq \cancel{\Phi_0} + \sum_{t=1}^{n} \gamma_t$$

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

if $\gamma_t = \alpha \cdot C_{opt}(\text{time } t)$

$e_t = C_A(\text{time } t)$

$$\sum c_t = C_A(\sigma) \leq \alpha \cdot C_{opt}$$

$$\alpha \cdot C_{opt} = \sum \gamma_t$$

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t

assume they start at the same state

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t
- Let M_t be cost of minimum cost matching between DC's servers and OPT servers
- Let S_t be sum of pairwise distances of DC's servers

initially have all of them at same pt

$M_0 = 0$, $S_0 = 0$

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t
- Let M_t be cost of minimum cost matching between DC's servers and OPT servers
- Let S_t be sum of pairwise distances of DC's servers
- Our potential function will be

$$\Phi_t = \underbrace{k \cdot M_t}_{\text{min cost matching between servers of DC and servers of OPT}} + \underbrace{S_t}_{\text{sum of pairwise distances of DC's servers}}$$

$$\sum_{i \neq j} |x_i - x_j|$$

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t
- Let M_t be cost of minimum cost matching between DC's servers and OPT servers
- Let S_t be sum of pairwise distances of DC's servers
- Our potential function will be

$$\Phi_t = k \cdot M_t + S_t$$

- Note that $\Phi_t \geq 0$ at all times

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t
- Let M_t be cost of minimum cost matching between DC's servers and OPT servers
- Let S_t be sum of pairwise distances of DC's servers
- Our potential function will be

$$\Phi_t = k \cdot M_t + S_t$$

- Note that $\Phi_t \geq 0$ at all times
- Use Amortized Analysis to compute amortized cost of DC

DC Analysis - Potential Function

Main idea: have the *ammortized cost* per request be (a multiple of) the cost of OPT, while the actual cost is the cost of DC.

- Consider the state of DC and of OPT at time t
- Let M_t be cost of minimum cost matching between DC's servers and OPT servers
- Let S_t be sum of pairwise distances of DC's servers
- Our potential function will be

$$\Phi_t = k \cdot M_t + S_t \Bigg\} \text{ Changes if OPT changes or if DC changes}$$

- Note that $\Phi_t \geq 0$ at all times
- Use Amortized Analysis to compute amortized cost of DC
- Break requests into two parts:
  - First account for OPT move
  - Then account for DC move

DC Analysis - Potential Function

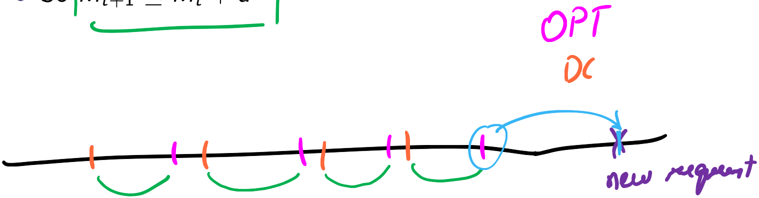
1. OPT moves

DC Analysis - Potential Function

1. OPT moves
   - If OPT moves a distance d, the distance from the moved server to the matched DC's server increases by d

DC Analysis - Potential Function

1. OPT moves
   - If OPT moves a distance d, the distance from the moved server to the matched DC's server increases by d
   - So $\boxed{M_{t+1} \leq M_t + d}$



$M_{t+1} \le$ old matching now has cost $\le \underbrace{\text{old cost}}_{M_t} + d$

$\Rightarrow$ there exists a matching of cost $\le M_t + d$

M_{t+1} is a minimum cost matching

DC Analysis - Potential Function

1. OPT moves
   - If OPT moves a distance d, the distance from the moved server to the matched DC's server increases by d
   - So $M_{t+1} \leq M_t + d$
   - Thus potential increased (so far) by $\Phi_{t+1} - \Phi_t \leq k \cdot d$

$$\Phi_{t+1} - \Phi_t = k(M_{t+1} - M_t) \leq k \cdot d$$

(we still have not accounted for DC moves)

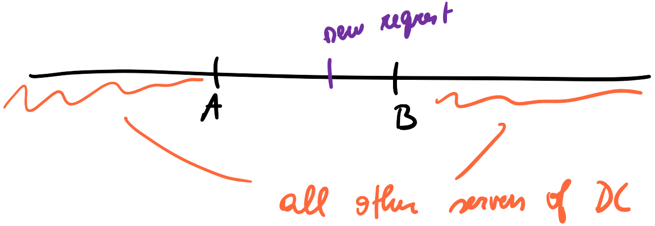
DC Analysis - Potential Function

1. OPT moves
   - If OPT moves a distance d, the distance from the moved server to the matched DC's server increases by d
   - So $M_{t+1} \leq M_t + d$
   - Thus potential increased (so far) by $\Phi_{t+1} - \Phi_t \leq k \cdot d$
   - Real cost incurred by DC: $c_{t+1} = 0$

DC Analysis - Potential Function

1. OPT moves
   - If OPT moves a distance d, the distance from the moved server to the matched DC's server increases by d
   - So $M_{t+1} \leq M_t + d$
   - Thus potential increased (so far) by $\Phi_{t+1} - \Phi_t \leq k \cdot d$
   - Real cost incurred by DC: $c_{t+1} = 0$
   - Ammortized cost of DC: $\gamma_{t+1} \leq k \cdot d$

DC Analysis - Potential Function

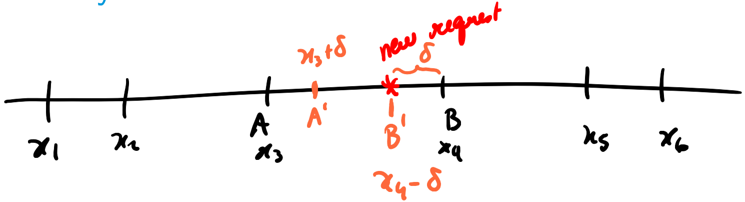


2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.

DC Analysis - Potential Function

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.

DC Analysis - Potential Function



2. DC moves
    1. The request falls between two servers A and B. Say that B is taken to the location requested.
        - Both servers move a distance δ.
        - Thus pairwise distances decrease by 2δ (because they are in a line)

$$\sum \text{pairwise distances} = \sum_i \left(\text{distances from } x_i\right)$$

$$d(A', B') = d(A, B) - 2\delta$$

this does not get cancelled

DC Analysis - Potential Function

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)

DC Analysis - Potential Function

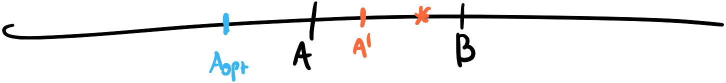
2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ

$$S_{t+1} = S_t - 2\delta$$

DC Analysis - Potential Function

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ
      - B has $\underbrace{\text{match}}_{\text{a server from OPT}}$ at destination (problem constraint)

DC Analysis - Potential Function



2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ
      - B has match at destination (problem constraint)
      - A may be further from its match, but balanced by B's move

DC Analysis - Potential Function

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ
      - B has match at destination (problem constraint)
      - A may be further from its match, but balanced by B's move
      - $M_{t+1} \leq M_t$

DC Analysis - Potential Function

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ
      - B has match at destination (problem constraint)
      - A may be further from its match, but balanced by B's move
      - $M_{t+1} \leq M_t$
      - Potential Change: $\Phi_{t+1} - \Phi_t \leq k \cdot 0 - 2 \cdot \delta = -2 \cdot \delta$

$$\leq k \underbrace{(M_{t+1} - M_t)}_{\leq 0} + \underbrace{(S_{t+1} - S_t)}_{= -2\delta}$$

DC Analysis - Potential Function

2. DC moves
    1. The request falls between two servers A and B. Say that B is taken to the location requested.
        - Both servers move a distance δ.
        - Thus pairwise distances decrease by 2δ (because they are in a line)
        - Changes in other pairwise distances cancel out (because line)
        - Thus S decreases by 2δ
        - B has match at destination (problem constraint)
        - A may be further from its match, but balanced by B's move
        - $M_{t+1} \leq M_t$
        - Potential Change: $\Phi_{t+1} - \Phi_t \leq k \cdot 0 - 2 \cdot \delta = -2 \cdot \delta$
        - Real cost incurred by DC: $c_{t+1} = 2\delta$

DC Analysis - Potential Function

$$\gamma_{t+1} = c_{t+1} + \Delta\Phi$$
$$= -2\delta + \Delta\Phi \leq -2\delta$$

2. DC moves
   1. The request falls between two servers A and B. Say that B is taken to the location requested.
      - Both servers move a distance δ.
      - Thus pairwise distances decrease by 2δ (because they are in a line)
      - Changes in other pairwise distances cancel out (because line)
      - Thus S decreases by 2δ
      - B has match at destination (problem constraint)
      - A may be further from its match, but balanced by B's move
      - $M_{t+1} \leq M_t$
      - Potential Change: $\Phi_{t+1} - \Phi_t \leq k \cdot 0 - 2 \cdot \delta = -2 \cdot \delta$
      - Real cost incurred by DC: $c_{t+1} = 2\delta$
      - Ammortized cost of DC: $\gamma_{t+1} \leq 2\delta - 2\delta = 0$

DC Analysis - Potential Function

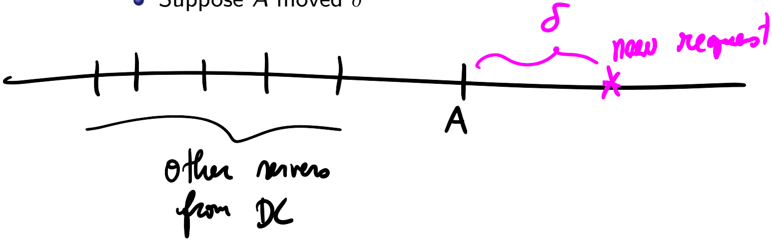
2. DC moves
    1. The request falls between two servers A and B. Say that B is taken to the location requested.
        - Both servers move a distance δ.
        - Thus pairwise distances decrease by 2δ (because they are in a line)
        - Changes in other pairwise distances cancel out (because line)
        - Thus S decreases by 2δ
        - B has match at destination (problem constraint)
        - A may be further from its match, but balanced by B's move
        - $M_{t+1} \leq M_t$
        - Potential Change: $\Phi_{t+1} - \Phi_t \leq k \cdot 0 - 2 \cdot \delta = -2 \cdot \delta$
        - Real cost incurred by DC: $c_{t+1} = 2\delta$
        - Ammortized cost of DC: $\gamma_{t+1} \leq 2\delta - 2\delta = 0$
    2. Only one server moves (request outside the border)

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ



DC Analysis - Potential Function

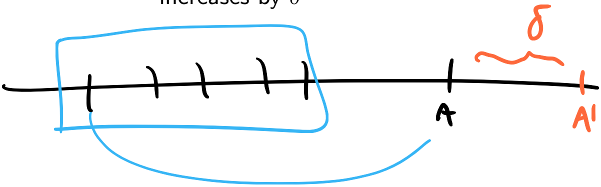
1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination
      - $M_{t+1} \leq M_t - \delta$

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination
      - $M_{t+1} \leq M_t - \delta$
      - Each pairwise distance (A, B) (where B is another of DC's servers) increases by δ



DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination
      - $M_{t+1} \leq M_t - \delta$
      - Each pairwise distance (A, B) (where B is another of DC's servers) increases by δ
      - Total distance increased: $S_{t+1} - S_t \leq (k-1) \cdot \delta$

DC Analysis - Potential Function

1. OPT moves distance d
    - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
    1. The request falls between two servers.
        - Ammortized cost of DC: $\gamma_t \leq 0$
    2. Only one server, say A, moves (request outside the border)
        - Suppose A moved δ
        - A has its match (from OPT's server) at destination
        - $M_{t+1} \leq M_t - \delta$
        - Each pairwise distance (A, B) (where B is another of DC's servers) increases by δ
        - Total distance increased: $S_{t+1} - S_t \leq (k-1) \cdot \delta$
        - Change in potential:

$$\Delta\Phi \leq \underline{-k \cdot \delta} + \underline{(k-1) \cdot \delta} = -\delta$$

$$= k(M_{t+1} - M_t) + (S_{t+1} - S_t)$$

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination
      - $M_{t+1} \leq M_t - \delta$
      - Each pairwise distance (A, B) (where B is another of DC's servers) increases by δ
      - Total distance increased: $S_{t+1} - S_t \leq (k-1) \cdot \delta$
      - Change in potential:

        $$\Delta\Phi \leq -k \cdot \delta + (k-1) \cdot \delta = -\delta$$

      - Real cost incurred by DC: $c_{t+1} = \delta$

DC Analysis - Potential Function

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server, say A, moves (request outside the border)
      - Suppose A moved δ
      - A has its match (from OPT's server) at destination
      - $M_{t+1} \leq M_t - \delta$
      - Each pairwise distance (A, B) (where B is another of DC's servers) increases by δ
      - Total distance increased: $S_{t+1} - S_t \leq (k-1) \cdot \delta$
      - Change in potential:

$$\Delta\Phi \leq -k \cdot \delta + (k-1) \cdot \delta = -\delta$$

$c_t \geq \Delta\Phi$

      - Real cost incurred by DC: $c_{t+1} = \delta$
      - Ammortized cost at this step: $\gamma_{t+1} = \leq \delta - \delta = 0$

$= c_t + \Delta\Phi$

DC Analysis - Wrapping Up

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server moves (request outside the border)
      - Ammortized cost at this step: $\gamma_t \leq \delta - \delta = 0$

$\boxed{\gamma_t \leq 0}$

Total ammortized cost per request is the sum of
the ammortized cost from 1 (OPT)
plus " " from 2

$\therefore$ at each request $\gamma_t^{final} \leq k \cdot d_{OPT,t}$

DC Analysis - Wrapping Up

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server moves (request outside the border)
      - Ammortized cost at this step: $\gamma_t \leq \delta - \delta = 0$

- By our potential function inequality, we have:

$$\underbrace{\sum_{t=1}^{n} c_t}_{C_{DC}} \leq \cancelto{0}{\Phi_0} + \underbrace{\sum_{t=1}^{n} \gamma_t}_{\text{total ammortized cost}} \leq \sum_{t=1}^{n} k \cdot d_{OPT,t}$$

$$k \cdot C_{opt} = \sum_{t=1}^{n} k \cdot d_{OPT,t}$$

DC Analysis - Wrapping Up

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server moves (request outside the border)
      - Ammortized cost at this step: $\gamma_t \leq \delta - \delta = 0$

- By our potential function inequality, we have:

$$\sum_{t=1}^{n} c_t \leq \Phi_0 + \sum_{t=1}^{n} \gamma_t$$

- Since $\gamma_t \leq k \cdot d$ whenever OPT moves d, and $\gamma_t \leq 0$ when OPT doesn't move, we have that $\sum_t \gamma_t \leq k \cdot C_{opt}$

DC Analysis - Wrapping Up

DC is k-competitive

1. OPT moves distance d
   - Ammortized cost of DC: $\gamma_t \leq k \cdot d$
2. DC moves
   1. The request falls between two servers.
      - Ammortized cost of DC: $\gamma_t \leq 0$
   2. Only one server moves (request outside the border)
      - Ammortized cost at this step: $\gamma_t \leq \delta - \delta = 0$

- By our potential function inequality, we have:

$$\sum_{t=1}^{n} c_t \leq \Phi_0 + \sum_{t=1}^{n} \gamma_t$$

- Since $\gamma_t \leq k \cdot d$ whenever OPT moves d, and $\gamma_t \leq 0$ when OPT doesn't move, we have that $\sum_t \gamma_t \leq k \cdot C_{opt}$
- Since Φ_0 is the initial state, we can regard it as constant (even 0, if require that servers start at a certain place)

Conclusion

- Online algorithms are important for many applications, when we need to make decisions right when we receive the information.
- Applications in
  - Stock Market
  - Dating
  - Skiing
  - Caching
  - Machine Learning (regret minimization)
  - many more...
- *Competitive Analysis*: measures performance of our algorithm against best algorithm that could *see into the future*
- Saw how to use *minimax theorem* in *Yao's principle* to prove lower bounds for randomized online algorithms.
- combined ammortized analysis in the online setting to solve k-server

Acknowledgement

- Lecture based largely on:
  - Lectures 18 & 20 of Karger's 6.854 Fall 2004 algorithms course
  - [Motwani & Raghavan 2007, Chapter 13]
- See Karger's Lecture 18 notes at

  `http://courses.csail.mit.edu/6.854/06/scribe/s23-onlineRandomLb.pdf`
- See Karger's Lecture 20 notes at

  `http://courses.csail.mit.edu/6.854/06/scribe/s24-paging.pdf`

References I

Motwani, Rajeev and Raghavan, Prabhakar (2007)
Randomized Algorithms