COT 6936: Topics in Algorithms

Giri Narasimhan

ECS 254A / EC 2474; Phone x3748; Email: giri@cs.fiu.edu
HOMEPAGE: http://www.cs.fiu.edu/~giri
https://moodle.cis.fiu.edu/v2.1/course/view.php?id=612

Feb 18, 2014

Presentation Outline

How to Analyze (Randomized) Online Algorithms?

- Online Algorithm A is α-competitive if there exists constant b such that for every sequence of inputs σ:

$$COST_A(\sigma) \leq \alpha \cdot COST_{OPT}(\sigma) + b$$

- Randomized Online Algorithm A is α-competitive if there exists constant b such that for every sequence of inputs σ:

$$E[COST_A(\sigma)] \leq \alpha COST_{OPT}(\sigma) + b$$

Use Expected cost instead

Presentation Outline

Randomized Algorithm

RANDOM Algorithm

- On a miss: evict a random item from cache.
- RANDOM is k-competitive

Randomized Algorithm

RANDOM Algorithm

- On a miss: evict a random item from cache.
- RANDOM is k-competitive

Randomized MARKER Algorithm

- Each of k pages in cache has a MARKer bit
- In each phase
  - If start of phase: UNMARK all pages
  - If request is a hit: MARK the page
  - If request is a miss:
    1. Replace random UNMARKed page; MARK new page;
    2. If all pages MARKed, start new round; UNMARK all pgs;
- MARKER algorithm is $2H_k$-competitive.

Randomized Algorithm

RANDOM Algorithm

- On a miss: evict a random item from cache.
- RANDOM is k-competitive

Randomized MARKER Algorithm

- Each of k pages in cache has a MARKer bit
- In each phase
  - If start of phase: UNMARK all pages
  - If request is a hit: MARK the page
  - If request is a miss:
    1. Replace random UNMARKed page; MARK new page;
    2. If all pages MARKed, start new round; UNMARK all pgs;
- MARKER algorithm is $2H_k$-competitive.

Lower Bound for Randomized Algorithm $= H_k$

Adversary Models

- In analyzing randomized online algs, we have 3 choices:
  - **Oblivious adversary**: adversary generates request sequence at start. I.e., cannot see action of algorithm or random choices. Adversary serves offline.
  - **Adaptive online adversary**: adversary generates request sequence adaptively (online). I.e., can see action, but not random choices. Adversary serves online
  - **Adaptive oine adversary**: adversary generates request sequence adaptively, and knows the result of the coin tosses. Adversary serves offline

Presentation Outline

Load Balancing

- We have a stream of m jobs to be assigned to one of n processors as they arrive.

Load Balancing

- We have a stream of m jobs to be assigned to one of n processors as they arrive.
- Centralized Algorithm, e.g., Round-Robin, can ensure that each processor gets m/n jobs.

Load Balancing

- We have a stream of m jobs to be assigned to one of n processors as they arrive.
- Centralized Algorithm, e.g., Round-Robin, can ensure that each processor gets m/n jobs.
- What if centralization is not possible?

Load Balancing

- We have a stream of m jobs to be assigned to one of n processors as they arrive.
- Centralized Algorithm, e.g., Round-Robin, can ensure that each processor gets m/n jobs.
- What if centralization is not possible? Randomization

Load Balancing

- We have a stream of m jobs to be assigned to one of n processors as they arrive.
- Centralized Algorithm, e.g., Round-Robin, can ensure that each processor gets m/n jobs.
- What if centralization is not possible? Randomization
  - Assign jobs uniformly at random to processors

Randomized Load Balancing: Analysis

- Y_{ij} = indicator random variable for the event: [job j is assigned to processor i].

Randomized Load Balancing: Analysis

- Y_{ij} = indicator random variable for the event: [job j is assigned to processor i].
- $E[Y_{ij}] = 1/n$.

Randomized Load Balancing: Analysis

- Y_{ij} = indicator random variable for the event: [job j is assigned to processor i].
- $E[Y_{ij}] = 1/n$.
- X_i = random variable for: [number of jobs assigned to processor i]

Randomized Load Balancing: Analysis

- Y_{ij} = indicator random variable for the event: [job j is assigned to processor i].
- $E[Y_{ij}] = 1/n$.
- X_i = random variable for: [number of jobs assigned to processor i]
- $E[X_i] = \sum_j E[Y_{ij}] = m/n$

Randomized Load Balancing: Analysis

- Y_{ij} = indicator random variable for the event: [job j is assigned to processor i].
- $E[Y_{ij}] = 1/n$.
- X_i = random variable for: [number of jobs assigned to processor i]
- $E[X_i] = \sum_j E[Y_{ij}] = m/n$
- $Prob[X_i > c] < e^{c-1}/e^c$ using Chernoff Bounds

Randomized Load Balancing: Analysis (2)

- **Case** $m = n$: With high probability (at least $1 - 1/n$), no processor is assigned more than $\Theta(\log n / \log \log n)$ jobs

Randomized Load Balancing: Analysis (2)

- **Case** $m = n$: With high probability (at least $1 - 1/n$), no processor is assigned more than $\Theta(\log n / \log\log n)$ jobs
- **Case** $m = \Omega(n \log n)$: With high probability (at least $1 - (1/n^2)$), every processor gets assigned between $m/(2n)$ and $2n/m$ jobs

Randomized Load Balancing: Analysis (2)

- **Case** $m = n$: With high probability (at least $1 - 1/n$), no processor is assigned more than $\Theta(\log n / \log \log n)$ jobs
- **Case** $m = \Omega(n \log n)$: With high probability (at least $1 - (1/n^2)$), every processor gets assigned between $m/(2n)$ and $2n/m$ jobs
- As m increases, imbalance diminishes

Presentation Outline

k-Server Problem

- Given a matric space $M = (V, d)$ on $|V| = n$ points, where the distance between any two vertices is given by the distance function $d(\cdot, \cdot)$.

k-Server Problem

- Given a matric space $M = (V, d)$ on $|V| = n$ points, where the distance between any two vertices is given by the distance function $d(\cdot, \cdot)$. This is equivalent to a weighted complete graph where weights satisfy triangle inequality.

k-Server Problem

- Given a matric space $M = (V, d)$ on $|V| = n$ points, where the distance between any two vertices is given by the distance function $d(\cdot, \cdot)$. This is equivalent to a weighted complete graph where weights satisfy triangle inequality.
- Given k servers that reside on k vertices from V.

k-Server Problem

- Given a matric space $M = (V, d)$ on $|V| = n$ points, where the distance between any two vertices is given by the distance function $d(\cdot, \cdot)$. This is equivalent to a weighted complete graph where weights satisfy triangle inequality.
- Given k servers that reside on k vertices from V.
- Given a request sequence $\sigma(t), t = 1, \ldots$ where each request specifies the vertex where the request is being made.

k-Server Problem

- Given a matric space $M = (V, d)$ on $|V| = n$ points, where the distance between any two vertices is given by the distance function $d(\cdot, \cdot)$. This is equivalent to a weighted complete graph where weights satisfy triangle inequality.
- Given k servers that reside on k vertices from V.
- Given a request sequence $\sigma(t), t = 1, \ldots$ where each request specifies the vertex where the request is being made.
- In order to serve a request at vertex $v \in V$, if there is server on v, then that server serves.

k-Server Problem

- Given a matric space $M = (V, d)$ on $|V| = n$ points, where the distance between any two vertices is given by the distance function $d(\cdot, \cdot)$. This is equivalent to a weighted complete graph where weights satisfy triangle inequality.
- Given k servers that reside on k vertices from V.
- Given a request sequence $\sigma(t), t = 1, \ldots$ where each request specifies the vertex where the request is being made.
- In order to serve a request at vertex $v \in V$, if there is server on v, then that server serves. A HIT

k-Server Problem

- Given a matric space $M = (V, d)$ on $|V| = n$ points, where the distance between any two vertices is given by the distance function $d(\cdot, \cdot)$. This is equivalent to a weighted complete graph where weights satisfy triangle inequality.
- Given k servers that reside on k vertices from V.
- Given a request sequence $\sigma(t), t = 1, \ldots$ where each request specifies the vertex where the request is being made.
- In order to serve a request at vertex $v \in V$, if there is server on v, then that server serves. A HIT
- Otherwise, one of the k servers is moved to that vertex and the request is served.

k-Server Problem

- Given a matric space $M = (V, d)$ on $|V| = n$ points, where the distance between any two vertices is given by the distance function $d(\cdot, \cdot)$. This is equivalent to a weighted complete graph where weights satisfy triangle inequality.
- Given k servers that reside on k vertices from V.
- Given a request sequence $\sigma(t), t = 1, \ldots$ where each request specifies the vertex where the request is being made.
- In order to serve a request at vertex $v \in V$, if there is server on v, then that server serves. A HIT
- Otherwise, one of the k servers is moved to that vertex and the request is served. A MISS

k-Server Problem

- Given a matric space $M = (V, d)$ on $|V| = n$ points, where the distance between any two vertices is given by the distance function $d(\cdot, \cdot)$. This is equivalent to a weighted complete graph where weights satisfy triangle inequality.
- Given k servers that reside on k vertices from V.
- Given a request sequence $\sigma(t), t = 1, \ldots$ where each request specifies the vertex where the request is being made.
- In order to serve a request at vertex $v \in V$, if there is server on v, then that server serves. A HIT
- Otherwise, one of the k servers is moved to that vertex and the request is served. A MISS
- Need an online algorithm to decide which of k servers will serve request, while minimizing total distance traveled by servers.

k-Server Problem: Applications

- Paging / Caching

k-Server Problem: Applications

- Paging / Caching
- Weighted caching (e.g., fonts on a printer)

k-Server Problem: Applications

- Paging / Caching
- Weighted caching (e.g., fonts on a printer)
- Two-headed disk drives

k-Server Problem: Results

- Lower Bound on competitiveness of k applies from before

k-Server Problem: Results

- Lower Bound on competitiveness of k applies from before
- Conjecture: Upper bound for competitiveness is k [MMS, 1990]

Presentation Outline

k-Server Problem: Greedy Algorithm

k-Server Problem: Greedy Algorithm

- Let the nearest server serve the request

k-Server Problem: Greedy Algorithm

- Let the nearest server serve the request
    - It minimizes the cost of each individual request

How competitive is this algorithm?

k-Server Problem: Greedy Algorithm

- Let the nearest server serve the request
  - It minimizes the cost of each individual request

How competitive is this algorithm?

What is the worst case scenario?

k-Server Problem: Greedy Algorithm

- Let the nearest server serve the request
  - It minimizes the cost of each individual request

  How competitive is this algorithm?
  What is the worst case scenario?
- Conjecture: Upper bound for competitiveness is k [MMS, 1990]

Balance Algorithm

Balance Algorithm

- Choose a server that would have moved the minimum total distance of any server

Balance Algorithm

- Choose a server that would have moved the minimum total distance of any server
  - Takes care of previous bad example since eventually the second server would be employed

Balance Algorithm

- Choose a server that would have moved the minimum total distance of any server
  - Takes care of previous bad example since eventually the second server would be employed
  - Tends to use all servers equally

Balance Algorithm

- Choose a server that would have moved the minimum total distance of any server
  - Takes care of previous bad example since eventually the second server would be employed
  - Tends to use all servers equally
  - Can be shown to be k-competitive if $k = n - 1$

Balance Algorithm

- Choose a server that would have moved the minimum total distance of any server
  - Takes care of previous bad example since eventually the second server would be employed
  - Tends to use all servers equally
  - Can be shown to be k-competitive if $k = n - 1$
  - Can do poorly in other situations

Balance Algorithm

- Choose a server that would have moved the minimum total distance of any server
  - Takes care of previous bad example since eventually the second server would be employed
  - Tends to use all servers equally
  - Can be shown to be k-competitive if $k = n - 1$
  - Can do poorly in other situations Meaning what?

Balance Algorithm

- Choose a server that would have moved the minimum total distance of any server
  - Takes care of previous bad example since eventually the second server would be employed
  - Tends to use all servers equally
  - Can be shown to be k-competitive if $k = n - 1$
  - Can do poorly in other situations Meaning what?
  - Not 2-competitive for $k = 2$

Residues Algorithm

Residues Algorithm

- Define **Residues** as

$$R_c(\sigma, S) = c \cdot C_{OPT}(sigma, S) - C_A(\sigma)$$

Residues Algorithm

- Define **Residues** as

$$R_c(\sigma, S) = c \cdot C_{OPT}(sigma, S) - C_A(\sigma)$$

- Choose a server that has the least residues of any server.

Residues Algorithm

- Define **Residues** as

$$R_c(\sigma, S) = c \cdot C_{OPT}(sigma, S) - C_A(\sigma)$$

- Choose a server that has the least residues of any server.
- RESIDUES is 2-competitive for $k = 2$.

HARMONIC Algorithm

- Natural, memoryless, randomized algorithm

HARMONIC Algorithm

- Natural, memoryless, randomized algorithm
- Choose a server with probability inversely proportional to its distance from request location

HARMONIC Algorithm

- Natural, memoryless, randomized algorithm
- Choose a server with probability inversely proportional to its distance from request location
- Expected to be α-competitive

HARMONIC Algorithm

- Natural, memoryless, randomized algorithm
- Choose a server with probability inversely proportional to its distance from request location
- Expected to be α-competitive
  - $k = 3$: $\alpha = 3^{17000}$

HARMONIC Algorithm

- Natural, memoryless, randomized algorithm
- Choose a server with probability inversely proportional to its distance from request location
- Expected to be α-competitive
  - $k = 3$: $\alpha = 3^{17000}$
  - **General** k: $\alpha = O(k2^k)$

WORK FUNCTION Algorithm

- Assume that a new request $r = \sigma(t)$ arrives

WORK FUNCTION Algorithm

- Assume that a new request $r = \sigma(t)$ arrives
- Let S be the current configuration of the servers.

WORK FUNCTION Algorithm

- Assume that a new request $r = \sigma(t)$ arrives
- Let S be the current configuration of the servers.
- Let x_i be the location of server s_i

WORK FUNCTION Algorithm

- Assume that a new request $r = \sigma(t)$ arrives
- Let S be the current configuration of the servers.
- Let x_i be the location of server s_i
- Serve the request by moving the server s_i that minimizes

$$w(X_i) + d(x_i, r),$$

WORK FUNCTION Algorithm

- Assume that a new request $r = \sigma(t)$ arrives
- Let S be the current configuration of the servers.
- Let x_i be the location of server s_i
- Serve the request by moving the server s_i that minimizes

$$w(X_i) + d(x_i, r),$$

where $w(X_i)$ is the minimal cost to serve a request and end in confirguration X_i, and $X_i = X - x_i + r$

WORK FUNCTION Algorithm

- Assume that a new request $r = \sigma(t)$ arrives
- Let S be the current configuration of the servers.
- Let x_i be the location of server s_i
- Serve the request by moving the server s_i that minimizes

$$w(X_i) + d(x_i, r),$$

  where $w(X_i)$ is the minimal cost to serve a request and end in confirguration X_i, and $X_i = X - x_i + r$
- WORK FUNCTION is $(2k-1)$-competitive

Simpler Metric Spaces

- Points on a line,

Simpler Metric Spaces

- Points on a line, circle,

Simpler Metric Spaces

- Points on a line, circle, , tree

Simpler Metric Spaces

- Points on a line, circle, , tree
- $(2n - 1)$-competitive algorithms exist

Presentation Outline

Notation

- Metric space $M = (V, d)$ with n-point vertex set V and distance function $d(\cdot, \cdot)$.

Notation

- Metric space $M = (V, d)$ with n-point vertex set V and distance function $d(\cdot, \cdot)$.
- Configuration $S \subseteq V$ with k vertices indicating location of servers

Notation

- Metric space $M = (V, d)$ with n-point vertex set V and distance function $d(\cdot, \cdot)$.
- Configuration $S \subseteq V$ with k vertices indicating location of servers
- Request Sequence: $\sigma = \{r_1, \ldots, \}$ where $r_i \in V$

Notation

- Metric space $M = (V, d)$ with n-point vertex set V and distance function $d(\cdot, \cdot)$.
- Configuration $S \subseteq V$ with k vertices indicating location of servers
- Request Sequence: $\sigma = \{r_1, \ldots, \}$ where $r_i \in V$
- Solutions: Sequence of configurations $S_0, S_1, \ldots,$

Notation

- Metric space $M = (V, d)$ with n-point vertex set V and distance function $d(\cdot, \cdot)$.
- Configuration $S \subseteq V$ with k vertices indicating location of servers
- Request Sequence: $\sigma = \{r_1, \ldots, \}$ where $r_i \in V$
- Solutions: Sequence of configurations $S_0, S_1, \ldots,$
- Cost of Algorithm A: $D_A(S_0, \sigma) = \sum_{t=1} D(S_{t-1}, S_t)$, where the distance between configurations is given by the cost of a minimum weight matching between the configurations.

Notation

- Metric space $M = (V, d)$ with n-point vertex set V and distance function $d(\cdot, \cdot)$.
- Configuration $S \subseteq V$ with k vertices indicating location of servers
- Request Sequence: $\sigma = \{r_1, \ldots, \}$ where $r_i \in V$
- Solutions: Sequence of configurations $S_0, S_1, \ldots,$
- Cost of Algorithm A: $D_A(S_0, \sigma) = \sum_{t=1} D(S_{t-1}, S_t)$, where the distance between configurations is given by the cost of a minimum weight matching between the configurations.
- Analysis: Performance ratio is ρ if

$$D_A(S_0, \sigma) \leq \rho \cdot D_{OPT}(S_0, \sigma) + f(S_0)$$

Notation

- Metric space $M = (V, d)$ with n-point vertex set V and distance function $d(\cdot, \cdot)$.
- Configuration $S \subseteq V$ with k vertices indicating location of servers
- Request Sequence: $\sigma = \{r_1, \ldots, \}$ where $r_i \in V$
- Solutions: Sequence of configurations $S_0, S_1, \ldots,$
- Cost of Algorithm A: $D_A(S_0, \sigma) = \sum_{t=1} D(S_{t-1}, S_t)$, where the distance between configurations is given by the cost of a minimum weight matching between the configurations.
- Analysis: Performance ratio is ρ if

$$D_A(S_0, \sigma) \leq \rho \cdot D_{OPT}(S_0, \sigma) + f(S_0)$$

OPT: Offline Algorithm

- We only consider lazy moves, i.e., no unprovoked moves are made.

OPT: Offline Algorithm

- We only consider lazy moves, i.e., no unprovoked moves are made.
- Use dynamic programming

$$C_{OPT}(\epsilon, S) = \begin{cases} 0 & \text{if } S = S_0 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

OPT: Offline Algorithm

- We only consider lazy moves, i.e., no unprovoked moves are made.
- Use dynamic programming

$$C_{OPT}(\epsilon, S) = \begin{cases} 0 & \text{if } S = S_0 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

$$C_{OPT}(\sigma v, S) = \begin{cases} \min_T C_{OPT}(\sigma, T) \\ +D(T, S), & \text{if } v \text{ is covered in } S \\ \\ \textit{undefined} & \text{otherwise} \end{cases}$$

Open Problems

Open Problems

- ***k*-Server Conjecture**: For every metric space, there exists an algorithm for the k-server problem with competitive ratio of k.

Open Problems

- ***k*-Server Conjecture**: For every metric space, there exists an algorithm for the k-server problem with competitive ratio of k.
- **Randomized *k*-Server Conjecture**: For every metric space, there exists a randomized algorithm for the k-server problem with competitive ratio of $\log k$.