Competitive Algorithms for Generalized k-Server in Uniform Metrics

Nikhil Bansal, Marek Eliáš, Grigorios Koumoutsos, Jesper Nederlof

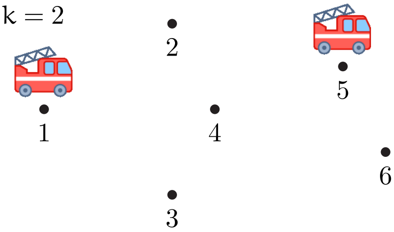
TU Eindhoven

The k-server problem [Manasse, McGeoch, Sleator '90]

- one of the central problems in Online Optimization
- studied intensively for several decades
- its study contributed many techniques to Online Algorithms

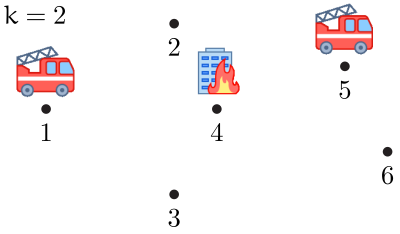
The k-server problem [Manasse, McGeoch, Sleator '90]

- k servers in given metric space
- sequence of requests received online
- target: minimize the distance travelled by the servers



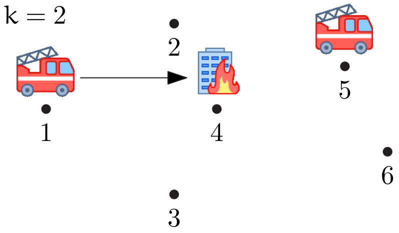
$\sigma = (\quad ?, \quad ?, \quad ?, \quad ?, \quad \ldots)$

- k servers in given metric space
- sequence of requests received online
- target: minimize the distance travelled by the servers



$$\sigma = (\quad 4, \quad ?, \quad ?, \quad ?, \quad \ldots)$$

- k servers in given metric space
- sequence of requests received online
- target: minimize the distance travelled by the servers

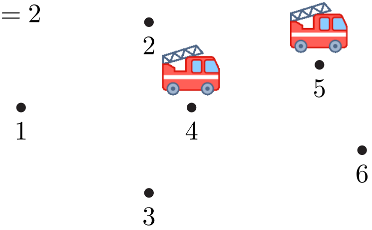


$$\sigma = (\ 4,\ ?,\ ?,\ ?,\ \ldots)$$

The k-server problem [Manasse, McGeoch, Sleator '90]

- k servers in given metric space
- sequence of requests received online
- target: minimize the distance travelled by the servers

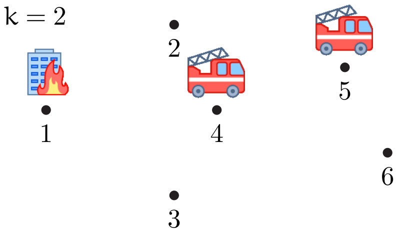
$k = 2$



$$\sigma = (\quad 4, \quad ?, \quad ?, \quad ?, \quad \dots)$$

The k-server problem [Manasse, McGeoch, Sleator '90]

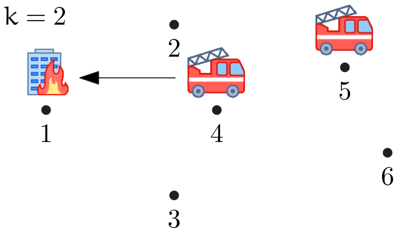
- k servers in given metric space
- sequence of requests received online
- target: minimize the distance travelled by the servers



$$\sigma = (\quad 4, \quad 1, \quad ?, \quad ?, \quad \ldots)$$

The k-server problem [Manasse, McGeoch, Sleator '90]

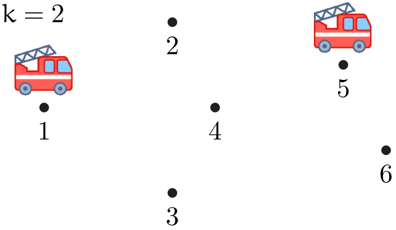
- k servers in given metric space
- sequence of requests received online
- target: minimize the distance travelled by the servers



$$\sigma = (\quad 4, \quad 1, \quad ?, \quad ?, \quad \ldots)$$

The k-server problem [Manasse, McGeoch, Sleator '90]

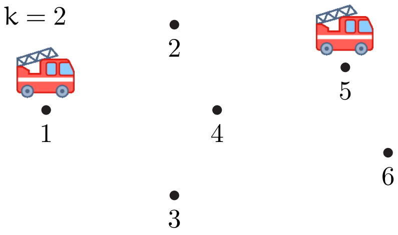
- k servers in given metric space
- sequence of requests received online
- target: minimize the distance travelled by the servers



$$\sigma = (\quad 4, \quad 1, \quad ?, \quad ?, \quad \ldots)$$

The k-server problem [Manasse, McGeoch, Sleator '90]

- k servers in given metric space
- sequence of requests received online
- target: minimize the distance travelled by the servers



$$\sigma = (\quad 4, \quad 1, \quad ?, \quad ?, \quad \ldots)$$

- Competitive ratio: $\frac{\text{ALG}}{\text{OPT}}$

The k-server problem [Manasse, McGeoch, Sleator '90]

Competitive ratio for k-server problem:

	upper bound	lower bound
deterministic:	$2k-1$	k
randomized:	$\log^6 k$	$\frac{\log k}{\log\log k}$

The k-server problem [Manasse, McGeoch, Sleator '90]

Competitive ratio for k-server problem:

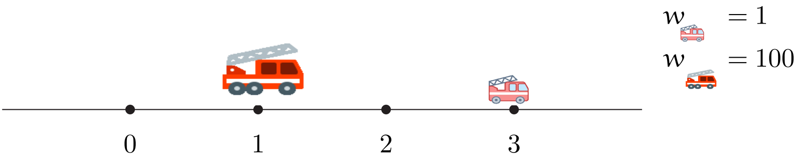
	upper bound	lower bound
deterministic:	$2k-1$	k
randomized:	$\log^6 k$	$\frac{\log k}{\log\log k}$

Natural variants not yet understood:

- e.g. weighted k-server, CNN, generalized k-server
- existing proofs for k-server do not extend
- several successful k-server algorithms not competitive

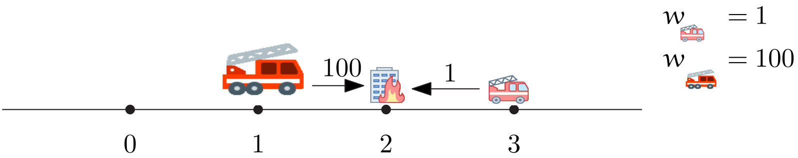
Example 1: weighted k-server [Fiat, Ricklin '94]

- servers have weights: $w_1, w_2, \ldots, w_k$
- target: minimize the weighted distance travelled
  - if server i moves by distance D, we pay $D \cdot w_i$



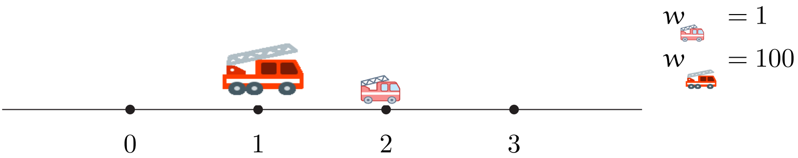
Example 1: weighted k-server [Fiat, Ricklin '94]

- servers have weights: $w_1, w_2, \ldots, w_k$
- target: minimize the weighted distance travelled
  - if server i moves by distance D, we pay $D \cdot w_i$



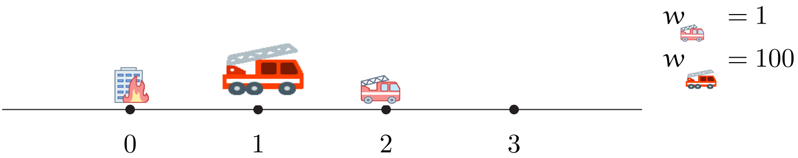
Example 1: weighted k-server [Fiat, Ricklin '94]

- servers have weights: $w_1, w_2, \ldots, w_k$
- target: minimize the weighted distance travelled
  - if server i moves by distance D, we pay $D \cdot w_i$



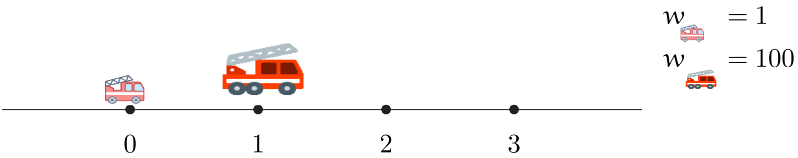
Example 1: weighted k-server [Fiat, Ricklin '94]

- servers have weights: $w_1, w_2, \ldots, w_k$
- target: minimize the weighted distance travelled
  - if server i moves by distance D, we pay $D \cdot w_i$



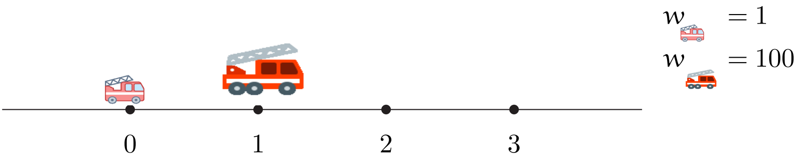
Example 1: weighted k-server [Fiat, Ricklin '94]

- servers have weights: $w_1, w_2, \ldots, w_k$
- target: minimize the weighted distance travelled
  - if server i moves by distance D, we pay $D \cdot w_i$



Example 1: weighted k-server [Fiat, Ricklin '94]

- servers have weights: $w_1, w_2, \ldots, w_k$
- target: minimize the weighted distance travelled
  - if server i moves by distance D, we pay $D \cdot w_i$



- 2 servers in a line: already non-trivial
  no memoryless algorithm competitive [Chrobak, Sgall '04]
  - for standard k-server, harmonic algorithm works

Example 1: weighted k-server [Fiat, Ricklin '94]

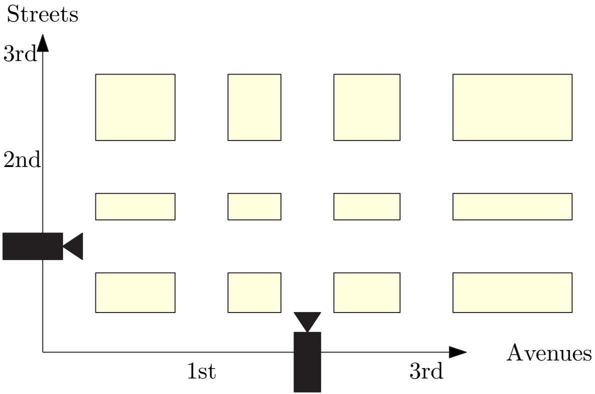
Known results

- upper bounds only for special cases
  - for uniform metrics by Fiat and Ricklin '94
  - for $k = 2$ by Sitters and Stougie '06
- lower bound: $2^{2^{\Omega(k)}}$ by Bansal et al. FOCS'17

	upper bound	lower bound
uniform metrics:	$2^{2^{O(k)}}$	$2^{2^{\Omega(k)}}$
$k = 2$:	$O(1)$	$\Omega(1)$
$k > 2$:	??	$2^{2^{\Omega(k)}}$

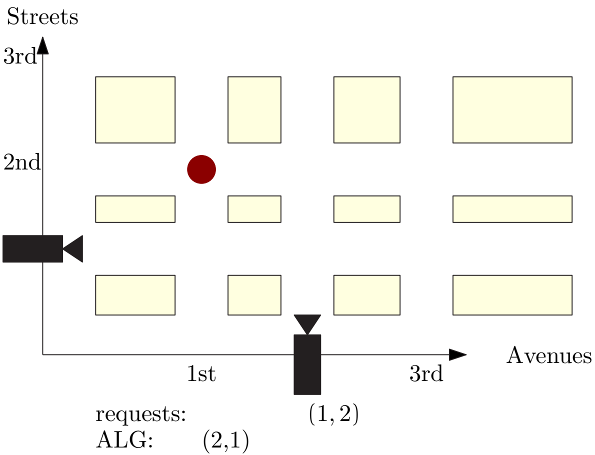
Example 2: CNN problem [Koutsoupias, Taylor '04]

- k servers, each moving in its own line metric
- for $k = 2$: moving live-broadcast vehicles in a city
- target: minimize the distance travelled by the servers



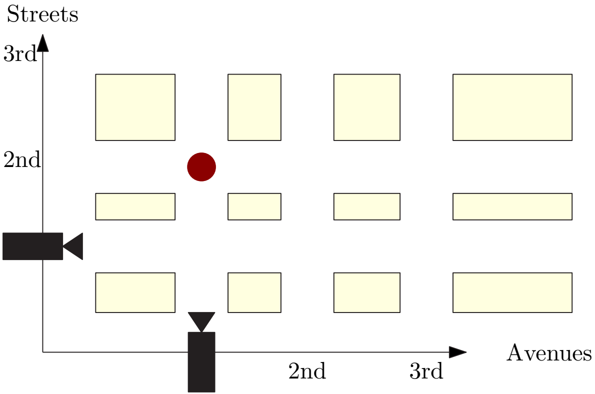
requests:
ALG: (2,1)

- k servers, each moving in its own line metric
- for $k = 2$: moving live-broadcast vehicles in a city
- target: minimize the distance travelled by the servers



Example 2: CNN problem [Koutsoupias, Taylor '04]

- k servers, each moving in its own line metric
- for $k = 2$: moving live-broadcast vehicles in a city
- target: minimize the distance travelled by the servers



requests: $(1,2)$
ALG: (2,1) $(1,1)$

Example 2: CNN problem [Koutsoupias, Taylor '04]

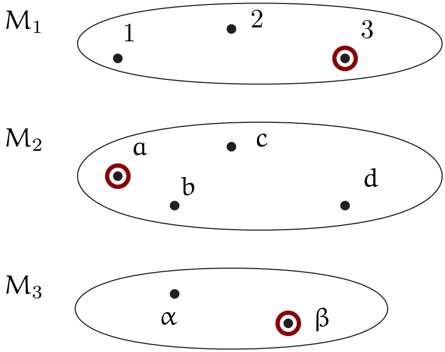
Known results:

- upper bound for $k = 2$ by Sitters and Stougie '06
- doubly-exponential lower bound by Bansal et al. '17

	upper bound	lower bound
$k = 2$:	$O(1)$	$\Omega(1)$
$k > 2$:	??	$2^{2^{\Omega(k)}}$

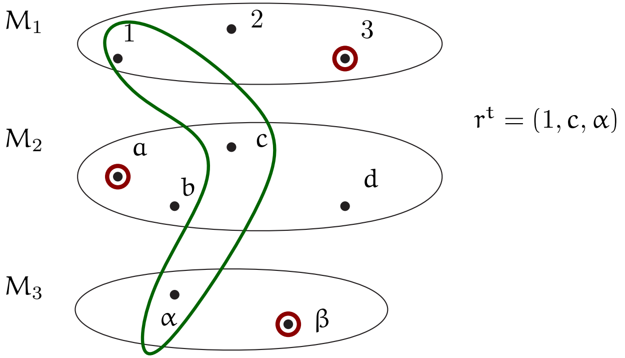
The generalized k-server problem [Koutsoupias, Taylor '04]

- each server moves in its own metric: $(M_1, d_1), \dots, (M_k, d_k)$
- at time t: we receive request $(r_1^t, \dots, r_k^t)$, $r_i^t \in M_i$
  - ALG has to select some server i and move it to r_i^t.
- target: minimize the distance travelled by the servers



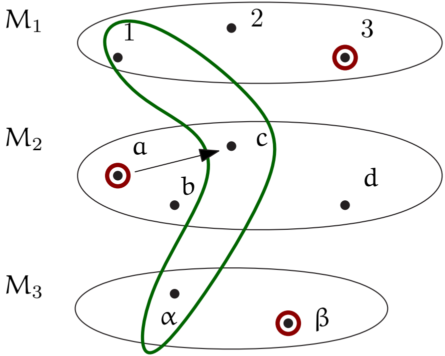
The generalized k-server problem [Koutsoupias, Taylor '04]

- each server moves in its own metric: $(M_1, d_1), \ldots, (M_k, d_k)$
- at time t: we receive request $(r_1^t, \ldots, r_k^t)$, $r_i^t \in M_i$
  - ALG has to select some server i and move it to r_i^t.
- target: minimize the distance travelled by the servers



The generalized k-server problem [Koutsoupias, Taylor '04]

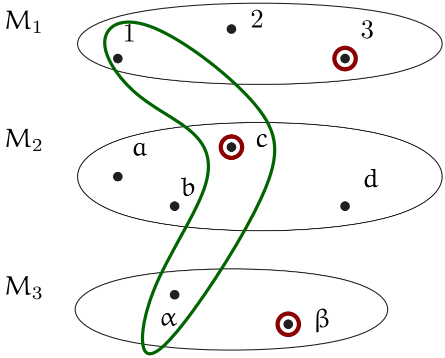
- each server moves in its own metric: $(M_1, d_1), \dots, (M_k, d_k)$
- at time t: we receive request $(r_1^t, \dots, r_k^t)$, $r_i^t \in M_i$
  - ALG has to select some server i and move it to r_i^t.
- target: minimize the distance travelled by the servers



$r^t = (1, c, \alpha)$

The generalized k-server problem [Koutsoupias, Taylor '04]

- each server moves in its own metric: $(M_1, d_1), \dots, (M_k, d_k)$
- at time t: we receive request $(r_1^t, \dots, r_k^t)$, $r_i^t \in M_i$
  - ALG has to select some server i and move it to r_i^t.
- target: minimize the distance travelled by the servers



$r^t = (1, c, \alpha)$

cost: $d_2(a, c)$

Generalized k-server: special cases

The k-server problem

- all metrics are the same: $M_1 = \cdots = M_k$
- each request has all coordinates equal: $r^t = (\sigma^t, \ldots, \sigma^t)$
  - σ^t is then the request for the k-server instance

Generalized k-server: special cases

The k-server problem

- all metrics are the same: $M_1 = \cdots = M_k$
- each request has all coordinates equal: $r^t = (\sigma^t, \ldots, \sigma^t)$
  - σ^t is then the request for the k-server instance

The weighted k-server problem

- for each i, $M_i = w_i M$ (a scaled copy of some fixed M)
- each request has all coordinates equal: $r^t = (\sigma^t, \ldots, \sigma^t)$

Generalized k-server: special cases

The k-server problem

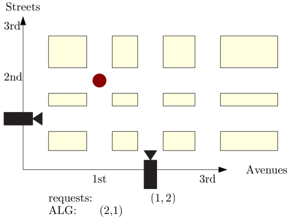
- all metrics are the same: $M_1 = \cdots = M_k$
- each request has all coordinates equal: $r^t = (\sigma^t, \ldots, \sigma^t)$
  - σ^t is then the request for the k-server instance

The weighted k-server problem

- for each i, $M_i = w_i M$ (a scaled copy of some fixed M)
- each request has all coordinates equal: $r^t = (\sigma^t, \ldots, \sigma^t)$

The CNN problem

- each M_i is a real line



Generalized k-server: special cases

The k-server problem

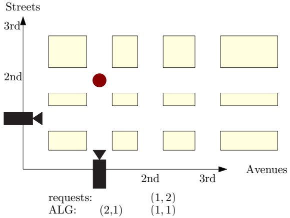
- all metrics are the same: $M_1 = \cdots = M_k$
- each request has all coordinates equal: $r^t = (\sigma^t, \ldots, \sigma^t)$
  - σ^t is then the request for the k-server instance

The weighted k-server problem

- for each i, $M_i = w_i M$ (a scaled copy of some fixed M)
- each request has all coordinates equal: $r^t = (\sigma^t, \ldots, \sigma^t)$

The CNN problem

- each M_i is a real line



Generalized k-server: simple lower bound

Theorem (Koutsoupias, Taylor '04). No deterministic algorithm can be better than $\frac{2^k-1}{k}$-competitive, even if each metric M_i contains only two points.

- for standard k-server, the competitive ratio is $O(k)$

Generalized k-server: simple lower bound

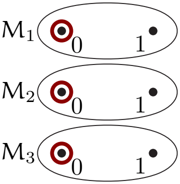
Theorem (Koutsoupias, Taylor '04). No deterministic algorithm can be better than $\frac{2^k-1}{k}$-competitive, even if each metric M_i contains only two points.

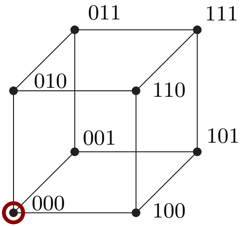
- for standard k-server, the competitive ratio is $O(k)$

Theorem (Koutsoupias, Taylor '04). No deterministic algorithm can be better than $\frac{2^k-1}{k}$-competitive, even if each metric M_i contains only two points.

Useful definition

- state $q = (q_1, \dots, q_k)$: server i is located at $q_i \in M_i$
- 2^k possible states



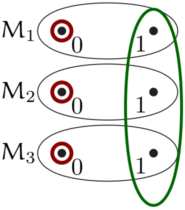


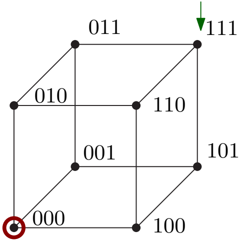
Generalized k-server: simple lower bound

Theorem (Koutsoupias, Taylor '04). No deterministic algorithm can be better than $\frac{2^k-1}{k}$-competitive, even if each metric M_i contains only two points.

Useful definition

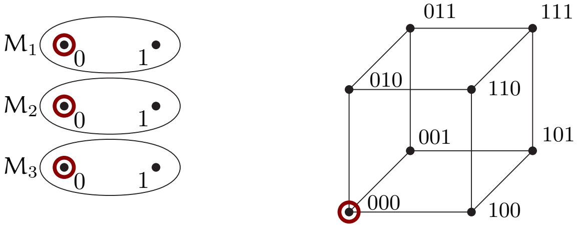
- state $q = (q_1, \ldots, q_k)$: server i is located at $q_i \in M_i$
- 2^k possible states
- q is feasible w.r.t. r^t: $q_i = r_i^t$ for some i





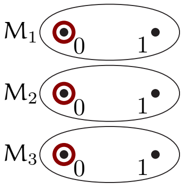
Generalized k-server: simple lower bound

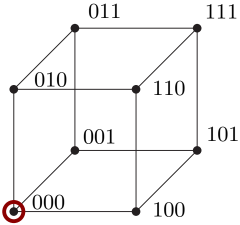
- ALG moves to $2^k - 1$ different states
- one state remains feasible during the whole sequence



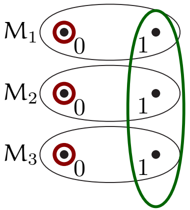
Generalized k-server: simple lower bound

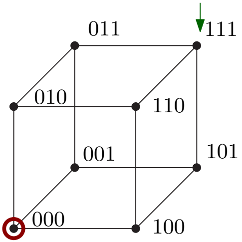
- ALG moves to $2^k - 1$ different states
- one state remains feasible during the whole sequence





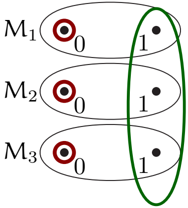
- ALG moves to $2^k - 1$ different states
- one state remains feasible during the whole sequence

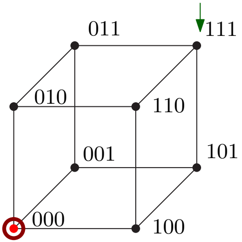




Generalized k-server: simple lower bound

- ALG moves to $2^k - 1$ different states
- one state remains feasible during the whole sequence

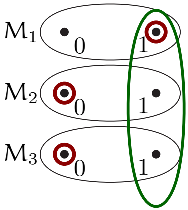


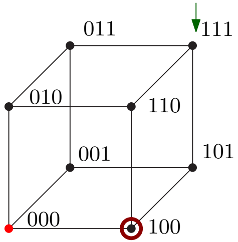


- red state: unfeasible — OPT cannot stay there
- black state: feasible for all the requests so far

Generalized k-server: simple lower bound

- ALG moves to $2^k - 1$ different states
- one state remains feasible during the whole sequence

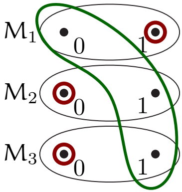


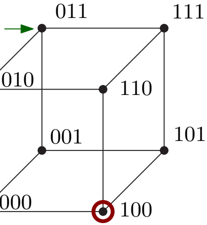


- red state: unfeasible — OPT cannot stay there
- black state: feasible for all the requests so far

Generalized k-server: simple lower bound

- ALG moves to $2^k - 1$ different states
- one state remains feasible during the whole sequence

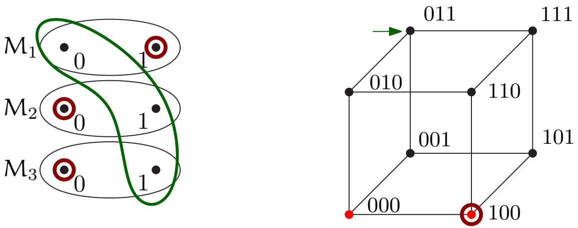




- red state: unfeasible — OPT cannot stay there
- black state: feasible for all the requests so far

Generalized k-server: simple lower bound

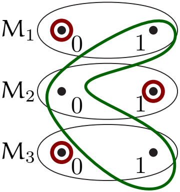
- ALG moves to $2^k - 1$ different states
- one state remains feasible during the whole sequence

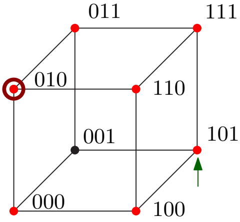


- red state: unfeasible — OPT cannot stay there
- black state: feasible for all the requests so far

Generalized k-server: simple lower bound

- ALG moves to $2^k - 1$ different states
- one state remains feasible during the whole sequence

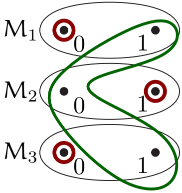


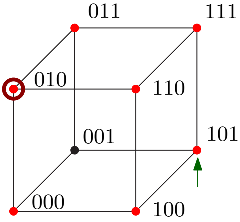


- red state: unfeasible — OPT cannot stay there
- black state: feasible for all the requests so far

Generalized k-server: simple lower bound

- ALG moves to $2^k - 1$ different states
- one state remains feasible during the whole sequence





- red state: unfeasible — OPT cannot stay there
- black state: feasible for all the requests so far
- solution for OPT:
  in the beginning, move to $(0, 0, 1)$ and stay there

Known results

Lower bounds

- $2^k - 1$ by Koutsoupias and Taylor '04

Known results

Lower bounds

- $2^k - 1$ by Koutsoupias and Taylor '04
- $2^{2^{\Omega(k)}}$ from weighted k-server [Bansal et al. '17]

Known results

Lower bounds

- $2^k - 1$ by Koutsoupias and Taylor '04
- $2^{2^{\Omega(k)}}$ from weighted k-server [Bansal et al. '17]

Upper bounds

- $k = 2$: $O(1)$ by Sitters and Stougie '06
- $k > 2$: no upper bound known
  - not even for a special class of metrics

Our results

New algorithms for the uniform metrics

- each M_i, for $i \in [k]$, is uniform, i.e., $d_i(x, y) = 1$

	upper bound	lower bound
deterministic:	$k2^k$	$2^k - 1$
randomized:	$k^3 \log k$	$k/\log^2 k$

Our results

New algorithms for the uniform metrics

- each M_i, for $i \in [k]$, is uniform, i.e., $d_i(x, y) = 1$

	upper bound	lower bound
deterministic:	$k2^k$	$2^k - 1$
randomized:	$k^3 \log k$	$k/\log^2 k$

Case of weighted (or scaled) uniform metrics

- each M_i is a scaled uniform metric, i.e., $d_i(x, y) = w_i$
- $2^{2^{k+3}}$-competitive algorithm

Our results

New algorithms for the uniform metrics

- each M_i, for $i \in [k]$, is uniform, i.e., $d_i(x, y) = 1$

	upper bound	lower bound
deterministic:	$k2^k$	$2^k - 1$
randomized:	$k^3 \log k$	$k/\log^2 k$

Case of weighted (or scaled) uniform metrics

- each M_i is a scaled uniform metric, i.e., $d_i(x, y) = w_i$
- $2^{2^{k+3}}$-competitive algorithm
- extension of algorithm by Fiat and Ricklin '94
- tight result: LB of $2^{2^{\Omega(k)}}$ [Bansal et al. '17]

Our results

New algorithms for the uniform metrics

- each M_i, for $i \in [k]$, is uniform, i.e., $d_i(x, y) = 1$

This talk

	upper bound	lower bound
deterministic:	$k2^k$	$2^k - 1$
randomized:	$k^3 \log k$	$k/\log^2 k$

Case of weighted (or scaled) uniform metrics

- each M_i is a scaled uniform metric, i.e., $d_i(x, y) = w_i$
- $2^{2^{k+3}}$-competitive algorithm
- extension of algorithm by Fiat and Ricklin '94
- tight result: LB of $2^{2^{\Omega(k)}}$ [Bansal et al. '17]

Deterministic algorithm for uniform metrics

- ALG works in phases
- in each phase: maintains set F of feasible states
- always moves to some $q \in F$

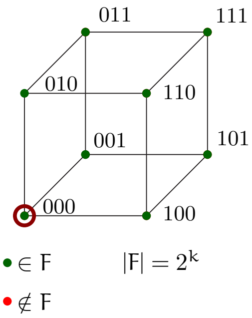
Deterministic algorithm for uniform metrics

- ALG works in phases
- in each phase: maintains set F of feasible states
- always moves to some $q \in F$

Beginning of the phase:

- set $F = M_1 \times M_2 \times \cdots \times M_k$ (all possible states)

Example: $|M_i| = 2 \; \forall i$



Deterministic algorithm for uniform metrics

- ALG works in phases
- in each phase: maintains set F of feasible states
- always moves to some $q \in F$

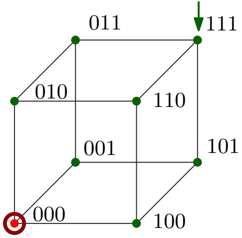
Beginning of the phase:

- set $F = M_1 \times M_2 \times \cdots \times M_k$ (all possible states)

To server request r^t:

- remove from F all states unfeasible w.r.t. r^t
- move to an arbitrary $q \in F$

Example: $|M_i| = 2 \ \forall i$



$\bullet \in F$ $\quad |F| = 2^k - 1$

$\bullet \notin F$

Deterministic algorithm for uniform metrics

- ALG works in phases
- in each phase: maintains set F of feasible states
- always moves to some $q \in F$

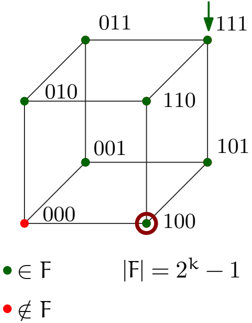
Beginning of the phase:

- set $F = M_1 \times M_2 \times \cdots \times M_k$ (all possible states)

To server request r^t:

- remove from F all states unfeasible w.r.t. r^t
- move to an arbitrary $q \in F$

Example: $|M_i| = 2 \ \forall i$



Deterministic algorithm for uniform metrics

- ALG works in phases
- in each phase: maintains set F of feasible states
- always moves to some $q \in F$

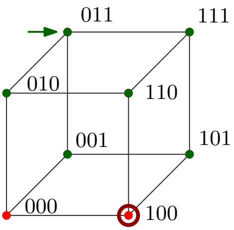
Beginning of the phase:

- set $F = M_1 \times M_2 \times \cdots \times M_k$ (all possible states)

To server request r^t:

- remove from F all states unfeasible w.r.t. r^t
- move to an arbitrary $q \in F$

Example: $|M_i| = 2\ \forall i$



$\bullet \in F$ $\quad |F| = 2^k - 2$

$\bullet \notin F$

Deterministic algorithm for uniform metrics

- ALG works in phases
- in each phase: maintains set F of feasible states
- always moves to some $q \in F$

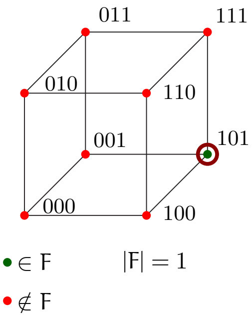
Beginning of the phase:

- set $F = M_1 \times M_2 \times \cdots \times M_k$ (all possible states)

To server request r^t:

- remove from F all states unfeasible w.r.t. r^t
- move to an arbitrary $q \in F$

Example: $|M_i| = 2 \; \forall i$



Deterministic algorithm for uniform metrics

- ALG works in phases
- in each phase: maintains set F of feasible states
- always moves to some $q \in F$

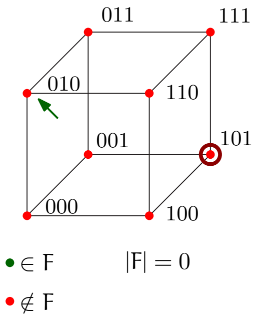
Beginning of the phase:

- set $F = M_1 \times M_2 \times \cdots \times M_k$ (all possible states)

To server request r^t:

- remove from F all states unfeasible w.r.t. r^t
- move to an arbitrary $q \in F$

Example: $|M_i| = 2 \; \forall i$



Deterministic algorithm for uniform metrics

- ALG works in phases
- in each phase: maintains set F of feasible states
- always moves to some $q \in F$

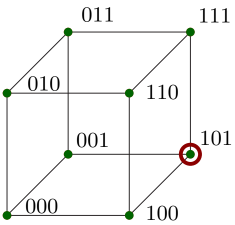
Beginning of the phase:

- set $F = M_1 \times M_2 \times \cdots \times M_k$ (all possible states)

To server request r^t:

- remove from F all states unfeasible w.r.t. r^t
- move to an arbitrary $q \in F$
- once $F = \emptyset$, start a new phase

Example: $|M_i| = 2 \; \forall i$



$\bullet \in F$ $\quad |F| = 2^k$

$\bullet \notin F$

Deterministic algorithm for uniform metrics

- ALG works in phases
- in each phase: maintains set F of feasible states
- always moves to some $q \in F$

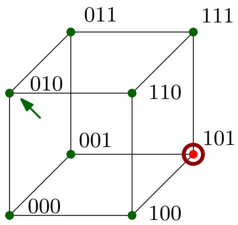
Beginning of the phase:

- set $F = M_1 \times M_2 \times \cdots \times M_k$ (all possible states)

To server request r^t:

- remove from F all states unfeasible w.r.t. r^t
- move to an arbitrary $q \in F$
- once $F = \emptyset$, start a new phase

Example: $|M_i| = 2 \; \forall i$



$\bullet \in F$ $\quad |F| = 2^k - 1$

$\bullet \notin F$

Bounding the length of the phase

Lemma. During any phase, $F = \emptyset$ after 2^k moves by ALG.

- even if $|M_i| \gg 2$

Bounding the length of the phase

Lemma. During any phase, $F = \emptyset$ after 2^k moves by ALG.

- even if $|M_i| \gg 2$
- proof using polynomial method

Bounding the length of the phase

Lemma. During any phase, $F = \emptyset$ after 2^k moves by ALG.

- even if $|M_i| \gg 2$
- proof using polynomial method

- assume $M_i = \{1, 2, \ldots, n\}$ for each i
- we define a feasibility polynomial of $2k$ variables
  - state $q = (q_1, \ldots, q_k)$
  - request $r^t = (r_1^t, \ldots, r_k^t)$

$$p(q, r^t) = \prod_{i=1}^{k} (q_i - r_i^t)$$

Bounding the length of the phase

Lemma. During any phase, $F = \emptyset$ after 2^k moves by ALG.

- even if $|M_i| \gg 2$
- proof using polynomial method

- assume $M_i = \{1, 2, \ldots, n\}$ for each i
- we define a feasibility polynomial of $2k$ variables
  - state $q = (q_1, \ldots, q_k)$
  - request $r^t = (r_1^t, \ldots, r_k^t)$

$$p(q, r^t) = \prod_{i=1}^{k} (q_i - r_i^t)$$

- $p(q, r^t) = 0$ if q is feasible w.r.t. r^t, i.e., $q_i = r_i^t$ for some i
- $p(q, r^t) \neq 0$ otherwise

Bounding the length of the phase

During a fixed phase:

- $r^1, r^2, \ldots, r^\ell$ — requests
- $q^1, q^2, \ldots, q^\ell$ — states of the algorithm

Bounding the length of the phase

During a fixed phase:

- $r^1, r^2, \dots, r^\ell$ — requests
- $q^1, q^2, \dots, q^\ell$ — states of the algorithm
- matrix $M \in \mathbb{R}^{\ell\times\ell}$: $M[t,t'] = p(q^t, r^{t'})$

$$M = \begin{array}{c} \\ r^1 \\ r^2 \\ r^3 \\ \vdots \\ r^\ell \end{array} \begin{array}{c} \begin{array}{ccccc} q^1 & q^2 & q^3 & \dots & q^\ell \end{array} \\ \left(\begin{array}{ccccc} \times & 0 & 0 & \cdots & 0 \\ . & \times & 0 & \cdots & 0 \\ . & . & \times & \ddots & 0 \\ . & . & . & \ddots & 0 \\ . & . & . & . & \times \end{array}\right) \text{requests} \\ \text{configurations} \end{array}$$

Bounding the length of the phase

During a fixed phase:

- $r^1, r^2, \ldots, r^\ell$ — requests
- $q^1, q^2, \ldots, q^\ell$ — states of the algorithm
- matrix $M \in \mathbb{R}^{\ell \times \ell}$: $M[t, t'] = p(q^t, r^{t'})$

$$
M = \begin{array}{c} \\ r^1 \\ r^2 \\ r^3 \\ \vdots \\ r^\ell \end{array}
\begin{array}{c}
\begin{array}{ccccc} q^1 & q^2 & q^3 & \ldots & q^\ell \end{array} \\
\left(\begin{array}{ccccc}
\times & 0 & 0 & \cdots & 0 \\
. & \times & 0 & \cdots & 0 \\
. & . & \times & \ddots & 0 \\
. & . & . & \ddots & 0 \\
. & . & . & . & \times
\end{array}\right) \text{requests} \\
\text{configurations}
\end{array}
$$

to server request r^t:

- remove from F all states unfeasible w.r.t. r^t
- move to some $q^{t+1} \in F$

Bounding the length of the phase

During a fixed phase:

- $r^1, r^2, \ldots, r^\ell$ — requests
- $q^1, q^2, \ldots, q^\ell$ — states of the algorithm
- matrix $M \in \mathbb{R}^{\ell\times\ell}$: $M[t,t'] = p(q^t, r^{t'})$

$$M = \begin{array}{c} \\ r^1 \\ r^2 \\ r^3 \\ \vdots \\ r^\ell \end{array} \begin{array}{c} \begin{array}{ccccc} q^1 & q^2 & q^3 & \ldots & q^\ell \end{array} \\ \begin{pmatrix} \times & 0 & 0 & \cdots & 0 \\ . & \times & 0 & \cdots & 0 \\ . & . & \times & \ddots & 0 \\ . & . & . & \ddots & 0 \\ . & . & . & . & \times \end{pmatrix} \\ \text{configurations} \end{array} \text{requests}$$

to server request r^t:

- remove from F all states unfeasible w.r.t. r^t
- move to some $q^{t+1} \in F$

- M has zeros above the diagonal
  - ALG always moves to some $q^{t+1} \in F$

Bounding the length of the phase

During a fixed phase:

- $r^1, r^2, \dots, r^\ell$ — requests
- $q^1, q^2, \dots, q^\ell$ — states of the algorithm
- matrix $M \in \mathbb{R}^{\ell \times \ell}$: $M[t, t'] = p(q^t, r^{t'})$

$$M = \begin{array}{c} \\ r^1 \\ r^2 \\ r^3 \\ \vdots \\ r^\ell \end{array} \begin{array}{c} \begin{array}{ccccc} q^1 & q^2 & q^3 & \dots & q^\ell \end{array} \\ \begin{pmatrix} \times & 0 & 0 & \cdots & 0 \\ . & \times & 0 & \cdots & 0 \\ . & . & \times & \ddots & 0 \\ . & . & . & \ddots & 0 \\ . & . & . & . & \times \end{pmatrix} \\ \text{configurations} \end{array} \text{requests}$$

to server request r^t:

- remove from F all states unfeasible w.r.t. r^t
- move to some $q^{t+1} \in F$

- M has zeros above the diagonal
- M non-zero entries in the diagonal
  - w.l.o.g. each request forces ALG to move

Bounding the length of the phase

During a fixed phase:

- $r^1, r^2, \ldots, r^\ell$ — requests
- $q^1, q^2, \ldots, q^\ell$ — states of the algorithm
- matrix $M \in \mathbb{R}^{\ell \times \ell}$: $M[t, t'] = p(q^t, r^{t'})$

$$M = \begin{array}{c} \\ r^1 \\ r^2 \\ r^3 \\ \vdots \\ r^\ell \end{array} \begin{array}{c} \begin{array}{ccccc} q^1 & q^2 & q^3 & \ldots & q^\ell \end{array} \\ \left(\begin{array}{ccccc} \times & 0 & 0 & \cdots & 0 \\ . & \times & 0 & \cdots & 0 \\ . & . & \times & \ddots & 0 \\ . & . & . & \ddots & 0 \\ . & . & . & . & \times \end{array}\right) \\ \text{configurations} \end{array} \text{requests}$$

to server request r^t:

- remove from F all states unfeasible w.r.t. r^t
- move to some $q^{t+1} \in F$

- M has zeros above the diagonal
- M non-zero entries in the diagonal

$\Rightarrow$ M has full rank

Bounding the length of the phase

Crucial claim:

- Rank of M is at most 2^k.

Bounding the length of the phase

Crucial claim:

- Rank of M is at most 2^k.

Proof:

- we factorize M as $M = AB$
  - where rank of both A and B is at most 2^k

$$p(q, r) = \prod_{i=1}^{k} (q_i - r_i) \quad \text{contains } 2^k \text{ monomials}$$

- M has full rank $\Rightarrow$ length of the phase is at most 2^k

End of the proof

Cost of ALG per phase

- ALG moves at most 2^k times, each move costs at most k
- $\text{cost}(\text{ALG}) \leqslant k2^k$

Cost of OPT per phase

- $F = \emptyset$ at the end of each phase
  - no state can serve all requests of the phase
- $\text{cost}(\text{OPT}) \geqslant 1$

Competitive ratio

$$\frac{\text{cost}(\text{ALG})}{\text{cost}(\text{OPT})} \leqslant k2^k$$

Randomized algorithm

Naïve algorithm

- F: the set of feasible states in the current phase
- move to $q \in F$ chosen uniformly at random
- $\log n$ factor in the competitive ratio

Randomized algorithm

Naïve algorithm

- F: the set of feasible states in the current phase
- move to $q \in F$ chosen uniformly at random
- $\log n$ factor in the competitive ratio

We need more structure

- we reprezent F as a collection of subspaces of feasible states
- this helps to guide the alogrithm's decisions
- random choice is done over subspaces instead of states

Concluding remarks

	upper bound	lower bound
uniform (deterministic):	$k2^k$	$2^k - 1$
uniform (randomized):	$k^3 \log k$	$k/\log^2 k$
weighted uniform:	$2^{2^{O(k)}}$	$2^{2^{\Omega(k)}}$

- no upper bounds known for other metrics whenever $k > 2$

Concluding remarks

	upper bound	lower bound
uniform (deterministic):	$k2^k$	$2^k - 1$
uniform (randomized):	$k^3 \log k$	$k/\log^2 k$
weighted uniform:	$2^{2^{O(k)}}$	$2^{2^{\Omega(k)}}$

- no upper bounds known for other metrics whenever $k > 2$
  - e.g. line metric

Concluding remarks

	upper bound	lower bound
uniform (deterministic):	$k2^k$	$2^k - 1$
uniform (randomized):	$k^3 \log k$	$k/\log^2 k$
weighted uniform:	$2^{2^{O(k)}}$	$2^{2^{\Omega(k)}}$

- no upper bounds known for other metrics whenever $k > 2$
  - e.g. line metric

Small announcement

- I am graduating this year and I am looking for a postdoc