Online Algorithms
Lectures 1 and 2

Jiří Sgall

Computer Science Institute of the Charles Univ., Praha

EWSCS, Palmse, March 2020
Outline of the course

Four mostly independent lectures:

1. Makespan scheduling
2. Paging and $k$-server
3. Bin packing
4. Throughput scheduling
Makespan Scheduling — Definitions

- Environment: $m$ machines.
- Input: Sequence of **jobs (tasks)** with processing times $p_1, \ldots, p_n$
- Output: Schedule of jobs on $m$ machines
  Formally: Partition $\{1, \ldots, n\}$ into sets $I_1, \ldots, I_m$
- Objective: Minimize the **makespan (length of schedule)**
  Formally: minimize $\max_{i \leq m} \sum_{j \in I_i} p_j$
Makespan Scheduling — Definitions

Makespan Scheduling

- Environment: \( m \) machines.
- Input: Sequence of jobs (tasks) with processing times \( p_1, \ldots, p_n \)
- Output: Schedule of jobs on \( m \) machines
  Formally: Partition \( \{1, \ldots, n\} \) into sets \( I_1, \ldots, I_m \)
- Objective: Minimize the makespan (length of schedule)
  Formally: minimize \( \max_{i \leq m} \sum_{j \in I_i} p_j \)

Online setting

Jobs come one by one, with known \( p_j \);
need to be assigned immediately, no changes later
Competitive ratio

Algorithm $ALG$ is $R$-competitive if there exists a constant $C$ such that for each instance $I$, the algorithm gives

$$ALG(I) \leq R \cdot OPT(I) + C$$
### Competitive ratio

Algorithm $\text{ALG}$ is $R$-competitive if there exists a constant $C$ such that for each instance $I$, the algorithm gives

$$E[\text{ALG}(I)] \leq R \cdot \text{OPT}(I) + C$$
**Competitive ratio**

Algorithm $ALG$ is $R$-competitive if there exists a constant $C$ such that for each instance $I$, the algorithm gives

$$E[ALG(I)] \leq R \cdot OPT(I) + C$$

**Online setting**

Jobs come one by one, need to be assigned immediately
### Competitive ratio

Algorithm \( ALG \) is \( R \)-competitive if there exists a constant \( C \) such that for each instance \( I \), the algorithm gives

\[
E[ALG(I)] \leq R \cdot OPT(I) + C
\]

### Online setting

Jobs come one by one, need to be assigned immediately

### Alternative online settings (not today)

- Jobs arrive over time (release times); possibly unknown running times
- Jobs have dependencies, arrive when predecessors completed
Greedy algorithm

- Schedule each job on the least loaded machine.
- Greedy is \((2 - 1/m)\)-competitive.
- Greedy is optimal for \(m = 2, 3\).
Greedy algorithm
- Schedule each job on the least loaded machine.
- Greedy is \((2 - 1/m)\)-competitive.
- Greedy is optimal for \(m = 2, 3\).

Randomized algorithm for two machines
- Keep the ratio of the expected loads \(2 : 1\).
- This is \(4/3\)-competitive and this is optimal.
**Greedy algorithm**
- Schedule each job on the least loaded machine.
- Greedy is \((2 - 1/m)\)-competitive.
- Greedy is optimal for \(m = 2, 3\).

**Randomized algorithm for two machines**
- Keep the ratio of the expected loads \(2 : 1\).
- This is \(4/3\)-competitive and this is optimal.

**Current best bounds**
- Deterministic: between 1.88 and 1.923 for large \(m\)
- Randomized: at least \(e/(e - 1)\) for \(m \to \infty\), at most 1.916
Preemptive Scheduling

**Definition**

- execution of jobs can be interrupted, moved to a different machine
- schedule: assign at most one job to each machine/time pair; a job cannot run on two machines simultaneously
- jobs come one by one, need to be scheduled completely
Preemptive Scheduling

**Definition**
- Execution of jobs can be interrupted, moved to a different machine.
- Schedule: assign at most one job to each machine/time pair; a job cannot run on two machines simultaneously.
- Jobs come one by one, need to be scheduled completely.

**Optimal algorithm**
- Maintain the ratio of loads \( m : (m - 1) \) if possible.
- Competitive ratio \( \frac{1}{1 - \left(1 - \frac{1}{m}\right)^m} \rightarrow \frac{e}{e - 1} \).
Preemptive Scheduling

**Definition**
- execution of jobs can be interrupted, moved to a different machine
- schedule: assign at most one job to each machine/time pair; a job cannot run on two machines simultaneously
- jobs come one by one, need to be scheduled completely

**Optimal algorithm**
- maintain the ratio of loads $m : (m - 1)$ if possible
- competitive ratio $1/(1 - (1 - 1/m)^m) \to e/(e - 1)$

**Generalizations**
- machines with speeds
- semi-online scenarios
Paging — Definitions

Paging (Caching) — basic model

- Environment:
  - $k$ — number of pages in the fast memory
  - $1, \ldots, N$ — pages in the slow memory
- Input: request sequence $r_1, r_2, \ldots$, of pages
- Output: service — upon a page fault, bring the requested page in the fast memory
- Objective: minimize the number of page faults
Paging — Definitions

Paging (Caching) — basic model

- Environment:
  - $k$ — number of pages in the fast memory
  - $1, \ldots, N$ — pages in the slow memory
- Input: request sequence $r_1, r_2, \ldots$, of pages
- Output: service — upon a page fault, bring the requested page in the fast memory
- Objective: minimize the number of page faults

Generalizations and variants

- Weighted caching — different pages may have different costs
- File caching — in addition, the requested files may have different size
- restrictions on request sequences
Paging — Results

Deterministic algorithms

- many $k$-competitive algorithms — FIFO, LRU, FWF
- lower bound of $k$
Paging — Results

Deterministic algorithms

- many $k$-competitive algorithms — FIFO, LRU, FWF
- lower bound of $k$

Randomized algorithms

- MARK
  - $H_k$-competitive for $N = k + 1$
  - $(2H_k - 1)$-competitive in general
- $H_k$-competitive algorithms for any $N$
- lower bound of $H_k$

$$H_k = 1 + 1/2 + 1/3 + \cdots + 1/k = \Theta(\log k)$$
Algorithm MARK

- Initially, all slots in the fast memory are unmarked
- Upon request $r$
  - If $r$ is in the fast memory, mark its slot
  - If all slots are marked, unmark all
  - Bring $r$ to a random unmarked slot, mark it
**k-server Problem — Definitions**

**k-server**

- **Environment:**
  - \( k \) — number of servers
  - \((M, d)\) — metric on \( N \) points
- **Input:** request sequence \( r_1, r_2, \ldots \), of points in \( M \)
- **Output:** service — upon a request, a server needs to be moved to the requested point
- **Objective:** minimize the total distance of moves of all servers

Generalizes:
- Paging — uniform metric, \( d(x, y) = 1 \) for \( x \neq y \)
- Weighted caching — metric is a star
- Ski rental — 3-point metric

Jiří Sgall

Online Algorithms Lectures 1 and 2
The $k$-server problem is defined as follows:

**Environment:**
- $k$ — number of servers
- $(M, d)$ — metric on $N$ points

**Input:** request sequence $r_1, r_2, \ldots$, of points in $M$

**Output:** service — upon a request, a server needs to be moved to the requested point

**Objective:** minimize the total distance of moves of all servers

**Generalizes:**
- Paging — uniform metric, $d(x, y) = 1$ for $x \neq y$
- Weighted caching — metric is a star
- Ski rental — 3-point metric
**k-server — Results**

**Deterministic algorithms**

- $k$-competitive algorithm on special spaces: line, tree, $N = k + 1$, also $k = 2$
- work function algorithm $(2k - 1)$-competitive
- lower bound $k$ for any metric space
Deterministic algorithms

- $k$-competitive algorithm on special spaces: line, tree, $N = k + 1$, also $k = 2$
- work function algorithm ($2k - 1$)-competitive
- lower bound $k$ for any metric space

Randomized algorithms

- HARMONIC — $O(k2^k)$-competitive, conjectured $O(k^2)$
- $O(\log k)$-competitive alg. for weighted caching
- $O((\log k)^6)$-competitive alg. for any metric
- $\Omega(\log k / \log \log k)$ lower bound for any metric