Introduction

In this lecture we’ll continue to discuss Admission Control on the line, but now each request’s value equals its length. next, we’ll get back to job scheduling - with a time factor added.

1 Admission Control: Value=Length

In this model, our goal is to maximize the overall value, where each request’s value equal its size (in the previous model, each request’s value was 1)

1.1 Deterministic & Non-Preemptive Algorithm

Theorem 1.1: Greedy is \( n \)-competitive

Proof.

• Greedy will accept at least a single request with size 1: \( \text{ON}(\sigma) \geq 1 \)
• Opt will accept at most \( n \): \( \text{OPT}(\sigma) \leq n \)

Theorem: Every deterministic non-preemptive algorithm is at least \( n \)-competitive

Proof.

• every algorithm will have to accept the first request with size 1 (otherwise he would be non-competitive)
• the following request will be of size \( n \): Opt will accept that one.

1.2 Random & Non-Preemptive Algorithm

Theorem: CRS (Classify & Randomly Select)+Greedy is \( O(\log n) \)-competitive

Proof. in each class, if every accepted request is of size \( (2^{i-1}, 2^i) \):

• \( \text{ON}(\sigma) \) will accept a request with a size \( a \): \( a \geq 2^{i-1} \)
• Opt can accept 2 additional requests with a size \( \leq 2^i \): \( \text{OPT}(\sigma) \leq 2 \cdot 2^i + a \)
• \( \frac{\text{OPT}(\sigma)}{\text{ON}(\sigma)} \leq \frac{2 \cdot 2^i + a}{a} \leq 1 + \frac{2 \cdot 2^i}{a} \leq 5 \)
• there are \( \log n \) classes, so overall we get \( 5 \cdot \log n = O(\log n) \) competitive algorithm.
Theorem: Lower bound for random and non-preemptive algorithm is $\Omega(\log n)$ - competitive

Proof. each time $ON(\sigma)$ accepts a request, it misses a request sized $n$. So, as we’ve seen in One Way Trading:

- in this case, stock price range (here, request size range ) in $[1, n]$. i.e., $\mu = n$
- we already know that the lower bound for One Way Trading is $\log \mu \rightarrow \Omega(\log n)$

1.3 Deterministic Algorithm with Preemption Allowed

"Anti-Reasonable" Algorithm: for any request $R$, accept if both intersected requests (from both sides) is at least 2(or perhaps a better constant: $g$)-times shorter.

Figure 1: request no. 4 accepted because 1,2,3 are more than 2-times shorter. thus, they are preempted

Theorem: "Anti-Reasonable" is constant-competitive

Proof. We can construct the same request-tree, as we did in the case where value=1. When $ON(\sigma)$ accepts a request with a size $|x|:

Figure 2: in both cases: $ON(\sigma)$ lost $2|gx|$

- Thus, $ON(\sigma) \geq x$ and $OPT(\sigma) \leq x(1 + 2g)$
- If we set $g = \frac{1}{g} + 1$ then according to the golden ratio: $g = \frac{1 + \sqrt{5}}{2}$
- Following that the competitive ratio equals $1 + 2g = 2 + \sqrt{5} = 4.236...$

2 Task Scheduling: with release & completion time

in this case every task has, in addition to its size property, a release time. A machine can process only one task at a given time, and our goal here is to process all tasks as fast as possible.
2.1 Reduction to Batch

First, we'll discuss the case of identical machines. We will use the greedy algorithm: apply a task on the first idle machine. Remark: greedy must be at least \( 2 - \frac{1}{m} \) competitive, because Load Balancing for identical machine is a special case here (release time of every task is in \([0, + \epsilon]\)).

**Theorem:** Greedy is 2-competitive (for the identical machine model)

**Proof.**

\[
\begin{align*}
\text{Time} & \quad |W| \\
X \text{- additional tasks performed by greedy} & \quad R \text{- Release time of task } W
\end{align*}
\]

- \( \text{OPT}(\sigma) \geq |x|, \text{OPT}(\sigma) \geq r + |w| \) - due to volume conservation
- \( \text{ON}(\sigma) = r + |x| + |w| \leq 2\text{OPT}(\sigma)x \)

**Reduction to batch:** we can wait certain time period, and then process all tasks that have arrived as offline \( \text{OPT}(\sigma) \) would.

- \( t_0 = 0, k = 0 \)
- collect all tasks with release time \( \leq t_k \), and yet to be processed
- solve the offline problem (or use a polynomial time approximation and by so lose a constant \( \alpha \))

**Theorem:** Reduction to Batch is \( 2\alpha \)-competitive for any machine model.

**Proof.**

- assume that the last task arrived between \( t_{k-1} \) to \( t_k \)
- define \( \Delta t_k \): processing time of tasks arrived in \( t_{k-1} \) to \( t_k \)
- \( \text{OPT}(\sigma) \geq \frac{\Delta t_k}{\alpha} \) (processing of tasks arrived in \([t_{k-1}, t_k]\)), \( \text{OPT}(\sigma) \geq t_{k-1} + \frac{\Delta t_{k+1}}{\alpha} \)
- \( \text{ON}(\sigma) = t_{k-1} + \Delta t_k + \Delta t_{k+1} \leq \alpha \text{OPT}(\sigma) + \alpha \text{OPT}(\sigma) = 2\alpha \text{OPT}(\sigma) \)

2.2 Minimum Flow Time

- define Flow Time of task \( i \): \( f_i = c_i - r_i \), where \( c_i \) is the completion time of task \( i \).
- we can use preemption: stop processing a task and resume later on, or migration: transfer a task to another machine.
• Now, our goal is to minimize the overall flow time: \( \text{Min } \sum_i f_i \)

At first, we assume \( m=1 \) and that all tasks are given at time \( t_0 = 0 \).

• **SPT** (Shortest Processing Time First) **Algorithm**: assign tasks by shortest processing time.

**Theorem**: SPT is optimal (if all tasks are given at time 0)

**Proof**.

if Preemption is not allowed:

• assuming OPT didn’t order the tasks by size, i.e there are 2 sequential tasks \( x,y \) where \( x \) is processed before \( y \) but \( |x| > |y| \).

• if we’ll process \( y \) before \( x \):
  – completion time of any other request will not change (\( x \) and \( y \) are sequential)
  – \( c_x \) will not change but \( c_y \) will diminish: **contradiction to the fact that this Algorithm is optimal**

if Preemption is allowed:

• assuming OPT performed a preemption at least once, lets say, it was processing \( x \) and switched to \( y \).

• if we were to process \( y \) before \( x \): \( c_y \) will diminish: **contradiction to the fact that this Algorithm is optimal**

Now lets relate to a more general case- where \( r_i \) is not fixed:

• **SRPT** (Shortest Remaining Processing Time First) **Algorithm**: assign tasks by their remaining shortest processing time.

**Theorem**: SRPT is 1-competitive

**Proof**. Assuming there is an optimal scheduling that is not SRPT:

• let A and B be tasks that are not assigned SRPT.

• now, we switch every process-slot of A with process-slot of B and vice versa.
  – Other tasks remain in tact.
  – the task that finishes last, will have the same completion time
  – but for the other one, it’s completion time diminish: **contradiction to the fact that this Algorithm is optimal**