Scheduling
Online Algorithms

Tim Nieberg
General Introduction

- on-line scheduling can be seen as scheduling with incomplete information
- at certain points, decisions have to be made without knowing the complete instance
- depending on the way how new information becomes known, different on-line paradigms are possible
On-Line Scheduling

On-Line paradigms

- scheduling jobs one by one
  - in this paradigm jobs are ordered in some list (sequence)
  - jobs are presented one by one to the decision maker
  - the moment the job is presented, its characteristics get available
  - the scheduling decision for the job has to be taken before the next job is presented
  - the scheduling decision is irreversible

Remarks:

- scheduling jobs one by one is list scheduling!
- in Lecture 5, we have shown that list scheduling is a $2 - 1/m$-approximation for $P||C_{max}$
On-Line paradigms (cont.)

- jobs arrive over time
  - jobs become known at their release date
  - the scheduling decision for a job may be delayed
  - at any time all currently available jobs are at the disposal of the decision maker
  - decisions of the past are irreversible

Remark:
- we consider this paradigm
Performance measure

- Quality of an on-line algorithm is mostly measured by evaluating its worst case performance.
- As reference value the best off-line value is used.
- Has a 'game theoretic' character:
  - The on-line algorithm plays against an 'adversary'.
  - The adversary makes a sequence of requests (jobs) to be served by the on-line algorithm.
  - The adversary also serves the request, but only after it knows all requests.
  - The adversary tries to get the costs of the on-line algorithm as high as possible compared to its own cost.
Performance measure - competitive analysis

- an on-line algorithm is $\rho$-competitive if its objective value is no more than $\rho$ times the optimal off-line value for all instances
- the competitive ratio is related to the approximation factor in off-line settings
Performance measure - competitive analysis

- an on-line algorithm is $\rho$-competitive if its objective value is no more than $\rho$ times the optimal off-line value for all instances
- the competitive ratio is related to the approximation factor in off-line settings
- if *randomization* is allowed within the on-line algorithm (i.e. random choices are allowed) the expected objective value is used for the competitive analysis
Performance measure - lower bounds

- how much does one lose by not having complete information or how much is it worth to know the future?
On-Line Scheduling

Performance measure - lower bounds

- how much does one lose by not having complete information or how much is it worth to know the future?
- the competitive ratio of a specific on-line algorithm is not the answer to this problem
- a lower bound on the competitive ratio of every possible on-line algorithm answers the question!
- such lower bounds can be achieved by providing a specific set of instances on which no on-line algorithm can perform well
Problem $1 | r_j | \sum C_j$

- problem is NP-hard
- if all release dates are equal, the SPT-rule solves the problem
- in the general case, SPT (each time the machine gets idle, process an available job with smallest processing time) is an on-line algorithm
- Theorem: For problem $1 | r_j | \sum C_j$ the SPT-algorithm has not a constant competitive ratio.
  (proof on board)
- Can we do better?
- How good can we do?
Problem 1 |r_j| \sum C_j - lower bound

- Theorem: Any deterministic on-line algorithm for problem
1 |r_j| \sum C_j has a competitive ratio of at least 2
(proof on the board)
- Remark: Proof of the theorem shows that any on-line
algorithm which has a constant competitive ratio needs a
'waiting' strategy
Problem 1\( |r_j| \sum C_j \) - algorithm

Algorithm delayed SPT (DSPT):

1. IF machine gets idle THEN
2. calculate next time \( t \) at which a job is available;
3. let \( j \) be unscheduled available job with smallest processing time
   (if choice, select job with smallest release date);
4. IF \( p_j \leq t \) THEN
   5. schedule job \( j \) at \( t \)
6. ELSE
7. wait until \( t = p_j \) or until a next job becomes available;
Problem 1\( |r_j| \sum C_j \) - algorithm (cont.)

- **Remarks on DSPT:**
  - The algorithm would like to order jobs by increasing processing times, but does not know if in the future smaller jobs arrive and how long to wait.
  - To cope with this, the algorithm waits so long that if it makes a 'mistake' and schedules a large job \( j \), all smaller jobs coming after \( j \) have a release date \( \geq p_j \).
  - This makes that the 'mistake' can not contribute too much to the criterion.
Theorem: Algorithm DSPT for problem 1\(|r_j| \sum C_j\) has competitive ratio 2

Proof (sketch):
- Notation:
  - $I$: instance with a minimal number of jobs for which DSPT has largest performance ratio
  - $\sigma$: schedule created by algorithm DSPT for instance $I$
- Observation: Schedule $\sigma$ consist of a single block (i.e. all jobs are processed without idle time in between)
- Assumption: jobs are numbered according to their position in $\sigma$
Problem 1 | r_j | \Sigma C_j - algorithm (cont.)

- Proof (cont.):
  - partition of \sigma into subblocks B_1, \ldots, B_k:
    - within B_i jobs are ordered according to increasing processing times
    - last job of B_i is larger than first job of B_{i+1}
    - B_i consist of jobs b(i - 1) + 1, \ldots, b(i)
      (i.e. \( b(i) = \min\{j > b(i - 1)| p_j > p_{j+1}\} \))
  - define \( m(i) \) such that \( p_{m(i)} = \max_{0 \leq j \leq b(i)} p_j \)
  - define pseudo schedule \( \psi \) by scheduling jobs in same order as in \sigma where job j from subblock B_{i+1} starts at \( S_j(\sigma) - p_{m(i)} \)
Problem 1\(|r_j|\sum C_j - \text{algorithm (cont.)}

Proof (cont.):

- in \(\psi\) job may overlap or start before their release date
- Notation:
  - \(\phi\): optimal preemptive schedule for \(I\)

- Lemma 1: For all \(j \in I\) we have: \(C_j(\sigma) - C_j(\psi) \leq C_j(\phi)\).
  (Proof on the board)

- Lemma 2: \(\sum C_j(\psi) \leq \sum C_j(\phi)\)
  (Proof in the handouts)
Problem 1\( |r_j| \sum C_j \) - randomized algorithm

- algorithm is based on optimal preemptive solution of problem 1\( |r_j, pmtn| \sum C_j \)
- SRPT (at each point in time schedule an available job with shortest remaining processing time) solves problem 1\( |r_j, pmtn| \sum C_j \)
- SRPT is an on-line algorithm and, thus, an on-line algorithm for problem 1\( |r_j| \sum C_j \) may use the result of SRPT
Problem 1 $|r_j| \sum C_j$ - randomized algorithm

- **algorithm $\alpha$-scheduler:**

1. $L$: list of jobs for which in the optimal preemptive schedule an $\alpha$ fraction has already been scheduled at the current time; initially: $L = \emptyset$;
2. proceed in time whereby the preemptive schedule is updated
3. IF $\alpha$ fraction of job $j$ is finished in preemptive schedule THEN
4. add $j$ at the end of $L$;
5. IF machine gets idle THEN
6. schedule first job of $L$ or if $L$ is empty, proceed in time;
Problem 1\(| r_j \mid \sum C_j\) - randomized algorithm

- for fixed \( \alpha \) the \( \alpha \)-scheduler is a deterministic algorithm
- for \( \alpha = 1 \), the \( \alpha \)-scheduler has a competitive ratio of 2 (proof by Phillips, Stein and Wein [1995])
- other values of \( \alpha \) lead to larger competitive ratios
- Theorem: The randomized on-line algorithm \( \alpha \)-scheduler, where \( \alpha \) is chosen according to probability density function \( f(\alpha) = e^\alpha/(e - 1) \), has competitive ratio \( e/(e - 1) \approx 1.582 \) (proof by Chekuri, Motwani, Natarajan and Stein [1997])
- Theorem: Any randomized on-line algorithm for problem 1\(| r_j \mid \sum C_j\) has a competitive ratio of at least \( e/(e - 1) \)