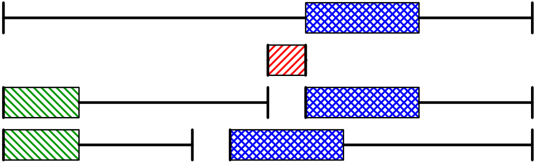
Open problems in throughput scheduling

Jiří Sgall

Computer Science Inst. of the Charles Univ. Prague

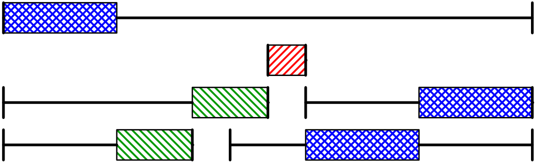
ESA, Sept 2012

A puzzle



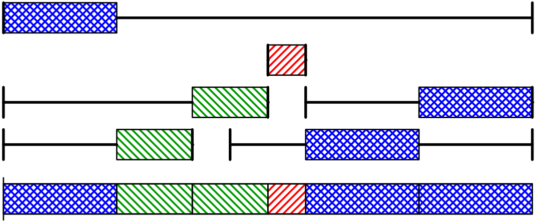
- Move boxes within their ranges.

A puzzle



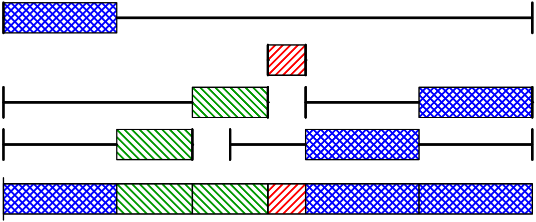
- Move boxes within their ranges.

A puzzle



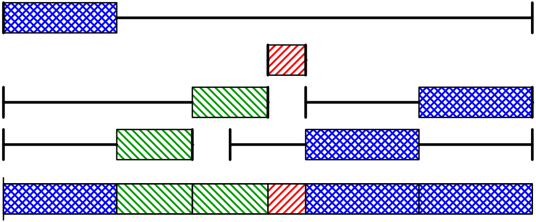
- Move boxes within their ranges.
- Align them so that they do not overlap vertically.

A puzzle



- Move boxes within their ranges.
- Align them so that they do not overlap vertically.
- Is this easy (in P) or difficult (NP-hard)?

A puzzle



- Move boxes within their ranges.
- Align them so that they do not overlap vertically.
- Is this easy (in P) or difficult (NP-hard)?
- What if there are only two (or 1000) diferent sizes of boxes?

Throughput scheduling

- Environment: One or more machines.
- Input: Jobs with length p_j, release time r_j, deadline d_j, and weight w_j. (Parameters are integers.)
- Output: Each job is assigned to a machine for a subinterval of $[r_j, d_j)$ of length p_j or rejected. No overlaps.
- Objective: Maximize the number (weight) of the completed jobs.

Throughput scheduling

- Environment: One or more machines.
- Input: Jobs with length p_j, release time r_j, deadline d_j, and weight w_j. (Parameters are integers.)
- Output: Each job is assigned to a machine for a subinterval of $[r_j, d_j)$ of length p_j or rejected. No overlaps.
- Objective: Maximize the number (weight) of the completed jobs.

This talk

- Online algorithms.
- Usually a single machine.

Throughput scheduling

- Environment: One or more machines.
- Input: Jobs with length p_j, release time r_j, deadline d_j, and weight w_j. (Parameters are integers.)
- Output: Each job is assigned to a machine for a subinterval of $[r_j, d_j)$ of length p_j or rejected. No overlaps.
- Objective: Maximize the number (weight) of the completed jobs.

This talk

- Online algorithms.
- Usually a single machine.
- Either jobs of equal length ($p_j = p$) and no weights
- or jobs of unit length ($p_j = 1$) with weights.

Online scheduling

- At time r_j, the other parameters of the job become known.
- At each time, if a machine is idle, the algorithm may decide to start a job.

Online scheduling

- At time r_j, the other parameters of the job become known.
- At each time, if a machine is idle, the algorithm may decide to start a job.

Competitive ratio

An algorithm A is R-competitive if for every instance I

- $OPT(I) \leq R \cdot A(I)$ for a deterministic algorithm

Online scheduling

- At time r_j, the other parameters of the job become known.
- At each time, if a machine is idle, the algorithm may decide to start a job.

Competitive ratio

An algorithm A is R-competitive if for every instance I

- $OPT(I) \leq R \cdot A(I)$ for a deterministic algorithm, or
- $OPT(I) \leq R \cdot E[A(I)]$ for a randomized algorithm.

Other scheduling problems

Variants

- Machine environments: speeds, shop scheduling (more operations) etc.
- Job parameters and restrictions: preemption, dependencies, resources etc.

Other scheduling problems

Variants

- Machine environments: speeds, shop scheduling (more operations) etc.
- Job parameters and restrictions: preemption, dependencies, resources etc.

Typical objectives

- MinMax: Minimize the length of schedule (or another global measure of balance).
- MinSum: Minimize the average completion time of a job (or waiting time, flow time, stretch, possibly weighted).

Jobs of equal length

Setting

- Equal lengths of jobs ($p_j = p$).
- No weights.
- Single machine.

Jobs of equal length

Setting

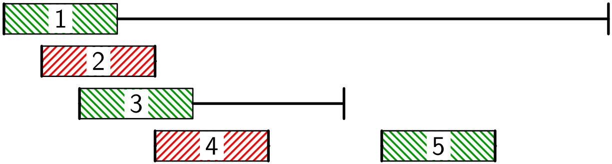
- Equal lengths of jobs ($p_j = p$).
- No weights.
- Single machine.

Outline

1. Offline problem is polynomial.
2. Greedy algorithms are 2-competitive.
3. Lower bounds.
4. A better randomized algorithm.
5. Generalizations, variants.

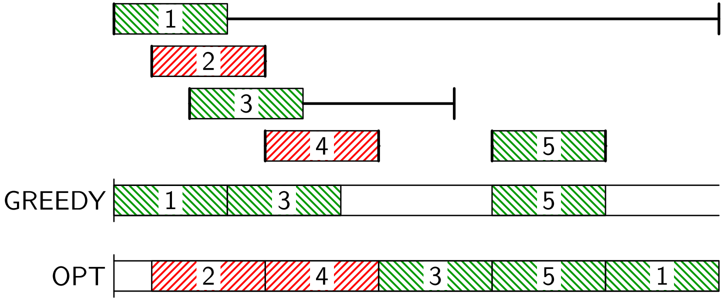
Greedy algorithms

GREEDY: If idle, start an arbitrary job.



Greedy algorithms

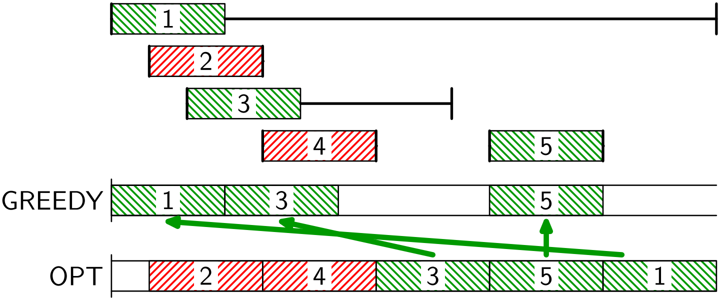
GREEDY: If idle, start an arbitrary job.



- Any such algorithm is 2-competitive.

Greedy algorithms

GREEDY: If idle, start an arbitrary job.

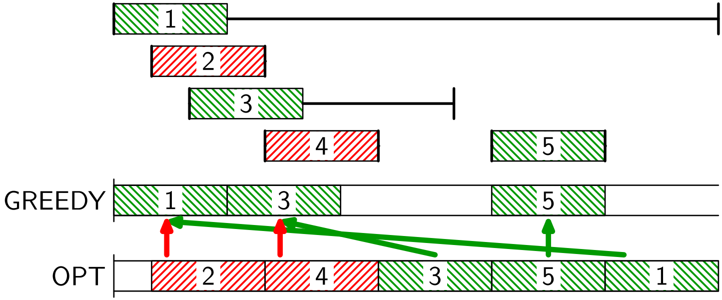


Charging scheme – GREEDY is 2-competitive

- Charge (map) a job in OPT to itself in GREEDY, if scheduled.

Greedy algorithms

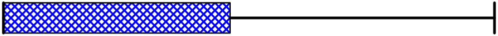
GREEDY: If idle, start an arbitrary job.



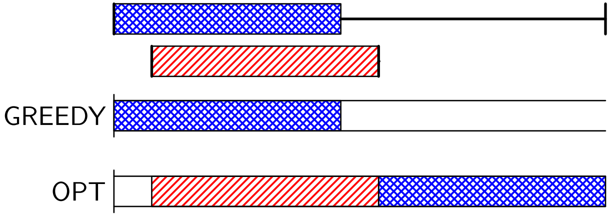
Charging scheme – GREEDY is 2-competitive

- Charge (map) a job in OPT to itself in GREEDY, if scheduled.
- Otherwise charge a job that OPT starts at t to the job GREEDY runs at t.

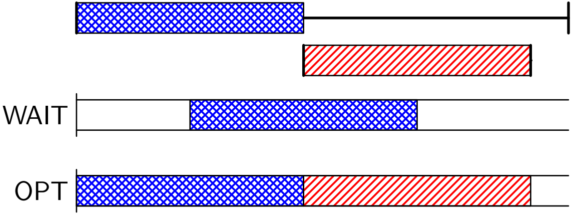
Lower bounds



Lower bounds

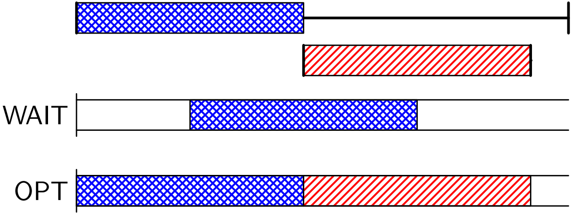


Lower bounds



- No deterministic algorithm is better than 2-competitive.

Lower bounds



- No deterministic algorithm is better than 2-competitive.
- No randomized algorithm is better than $4/3$-competitive. (For one of the two instances, on average, runs at most 1.5 jobs out of 2.)

A barely random algorithm I

- Generate two schedules, A and B. Flip a coin to choose one of them.

A barely random algorithm I

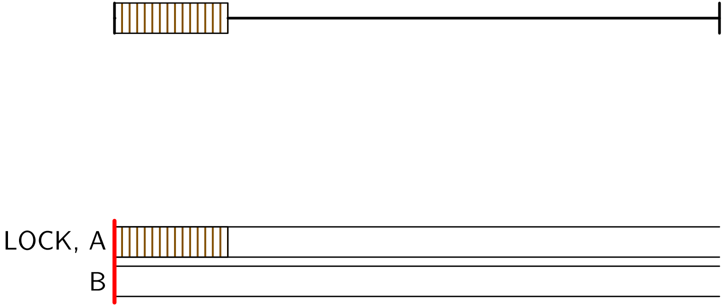
- Generate two schedules, A and B. Flip a coin to choose one of them.
- A and B are produced by two identical processes using a common lock.

A barely random algorithm I

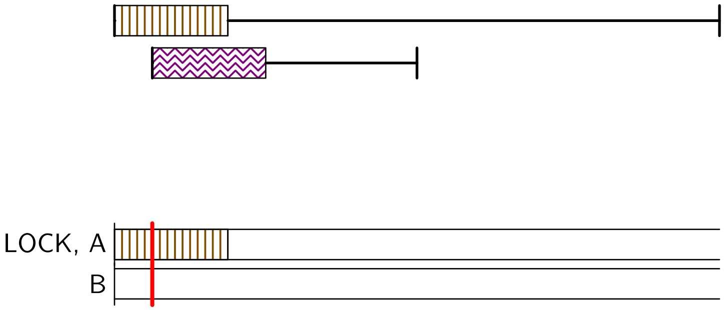
- Generate two schedules, A and B. Flip a coin to choose one of them.
- A and B are produced by two identical processes using a common lock.

- If the machine is idle (in A or B) and the set of pending jobs is not flexible (idling for time p would lose some job), start the most urgent job.
- If the machine is idle (in A or B) and the set of pending jobs is flexible (idling for time p does no harm):
  - If the lock is available, acquire it, start the most urgent job and release the lock after the job is completed.
  - Otherwise stay idle.

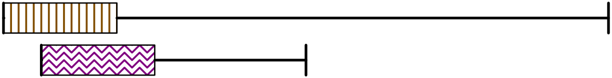
A barely random algorithm II

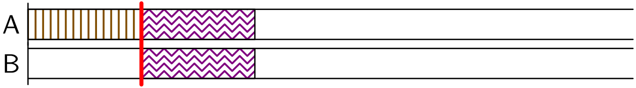


A barely random algorithm II

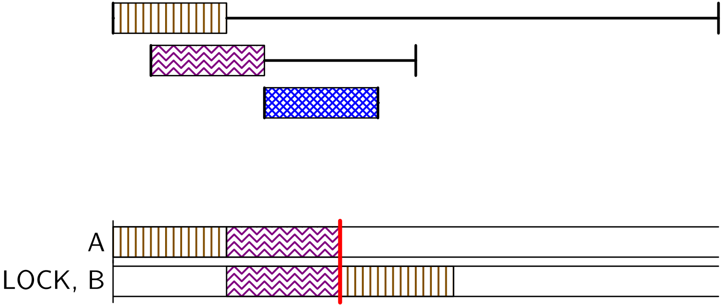


A barely random algorithm II

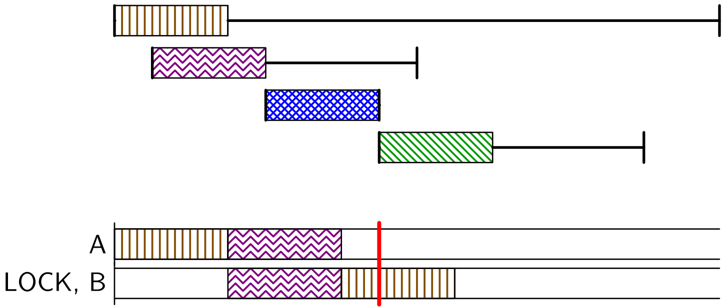




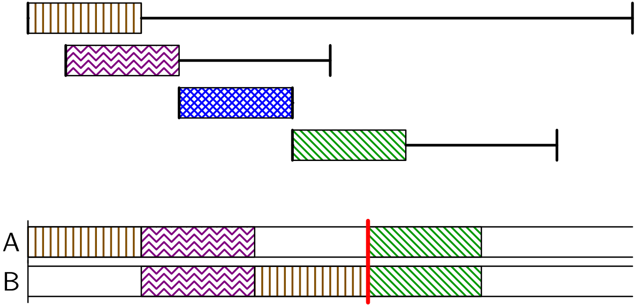
A barely random algorithm II



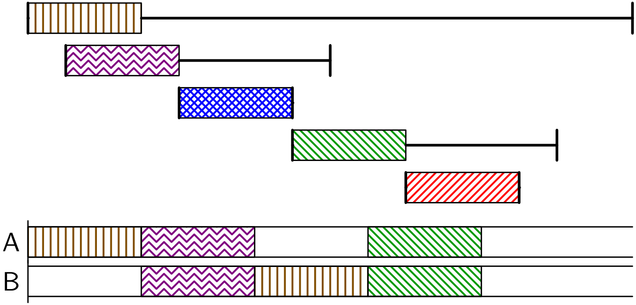
A barely random algorithm II



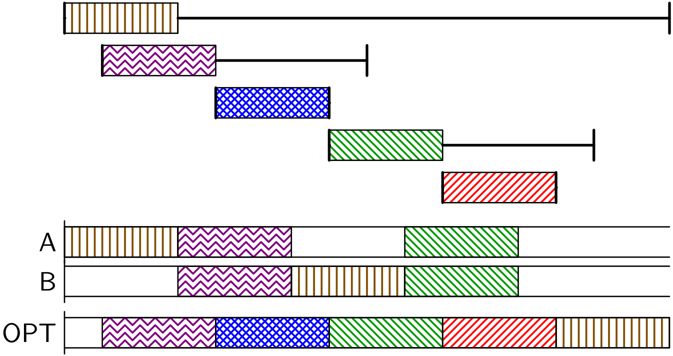
A barely random algorithm II



A barely random algorithm II



A barely random algorithm II



A barely random algorithm III

- Analyzed by a more complex charging scheme.
- Each job in OPT charges $1/2$, $1/3$, or $1/6$ to itself or to the job running at the same time in A and B.
- Each job in A or B is charged at most $5/6$.

A barely random algorithm III

- Analyzed by a more complex charging scheme.
- Each job in OPT charges $1/2$, $1/3$, or $1/6$ to itself or to the job running at the same time in A and B.
- Each job in A or B is charged at most $5/6$.

Theorem

This algorithm is $5/3$-competitive.

A barely random algorithm III

- Analyzed by a more complex charging scheme.
- Each job in OPT charges $1/2$, $1/3$, or $1/6$ to itself or to the job running at the same time in A and B.
- Each job in A or B is charged at most $5/6$.

Theorem

This algorithm is $5/3$-competitive.

Open problem

Find a randomized algorithm with the optimal competitive ratio.

More machines

Parallel machines make the problem easier!

More machines

Parallel machines make the problem easier!

Results

- For 2 machines, there is a $3/2$-competitive deterministic algorithm and this is optimal.
- For m machines, there is an R-competitive deterministic algorithm with $R \to e/(e-1) \approx 1.58$ for $m \to \infty$.
- The lower bound approaches $6/5$ for $m \to \infty$.

More machines

Parallel machines make the problem easier!

Results

- For 2 machines, there is a $3/2$-competitive deterministic algorithm and this is optimal.
- For m machines, there is an R-competitive deterministic algorithm with $R \to e/(e-1) \approx 1.58$ for $m \to \infty$.
- The lower bound approaches $6/5$ for $m \to \infty$.

Open problem

Decrease the gap for $m \to \infty$.

Jobs with fixed start times

- Each job has to be started at its release r_j or rejected.
- Jobs have a length p_j and a weight w_j.
- Jobs can be stopped (preempted).

Jobs with fixed start times

- Each job has to be started at its release r_j or rejected.
- Jobs have a length p_j and a weight w_j.
- Jobs can be stopped (preempted).

Results

- There is a 4-competitive algorithm for various cases, including equal times ($p_j = p$), unit weights ($w_j = 1$), and uniform weights ($w_j = p_j$); it works for parallel machines.
- There is a matching lower bound.

Machines with speeds

- Each job has to be started at its release r_j or rejected.
- A machine with speed s_i processes job j in time p_j/s_i.

Machines with speeds

- Each job has to be started at its release r_j or rejected.
- A machine with speed s_i processes job j in time p_j/s_i.
- Jobs are identical ($p_j = 1$ and $w_j = 1$).

GREEDY: Start the released job on the fastest available machine.

Machines with speeds

- Each job has to be started at its release r_j or rejected.
- A machine with speed s_i processes job j in time p_j/s_i.
- Jobs are identical ($p_j = 1$ and $w_j = 1$).

GREEDY: Start the released job on the fastest available machine.

Results for the greedy algorithm

- For two machines, GREEDY is $4/3$-competitive and this is optimal.
- For $m \to \infty$ the competitive ratio is between 1.56 and 2.

Open problem(s)

Analyze GREEDY, or find another algorithm with a competitive ratio below 2.

Unit time jobs with weights

Setting

- Unit length of jobs ($p_j = 1$).
- General weights.
- Single machine.

Unit time jobs with weights

Setting

- Unit length of jobs ($p_j = 1$).
- General weights.
- Single machine.

Outline

1. Offline problem is easy (matching).
2. Greedy algorithm is 2-competitive.
3. A better randomized algorithm.
4. A better deterministic algorithm.
5. Generalizations, variants.

Motivation and variants

Forwarding packets in network switches

Motivation and variants

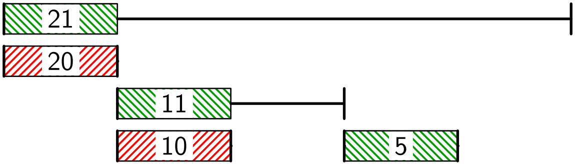
Forwarding packets in network switches

Restricted scenarios

- 2-bounded: Some packets may wait a single step, some packets not at all. ($d_j \leq r_j + 2$)
- Agreeable deadlines: $r_j < r_k$ implies $d_j \leq d_k$.
- Weighted queues: The deadlines are not known, only their order.
- Limited number of weights.

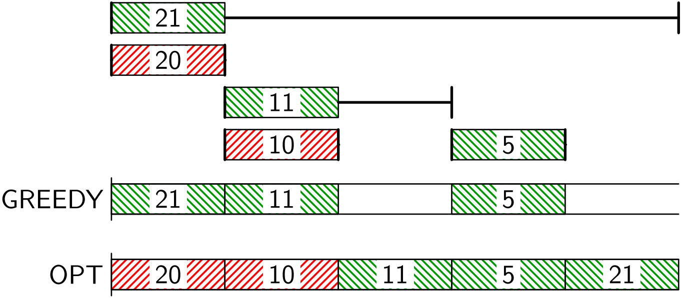
Greedy algorithm

GREEDY: If idle, start a pending job with the maximal weight.



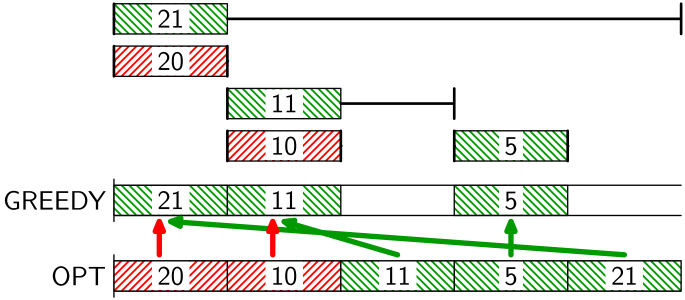
Greedy algorithm

GREEDY: If idle, start a pending job with the maximal weight.



Greedy algorithm

GREEDY: If idle, start a pending job with the maximal weight.



Charging scheme – GREEDY is 2-competitive

- Charge (map) a job in OPT to itself in GREEDY, if scheduled.
- Otherwise charge a job in OPT to the job GREEDY runs at the same time.

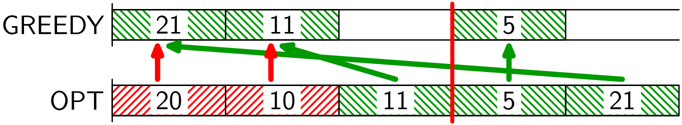
A randomized algorithm

- At each time, pick uniformly random real $x \in (-1, 0)$.
- Let h be the largest weight of a pending job.
- Among all the pending jobs with $w_j \geq e^x \cdot h$, schedule a job with the earlieast deadline.

Theorem

This algorithm is $e/(e-1) \approx 1.58$-competitive.

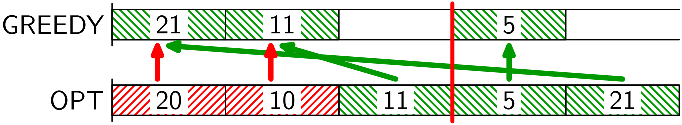
A potential function



How much "money" we need at a given time and configuration?

We earn $R \cdot w_j$ for running a job and pay w_j if OPT runs a job.

A potential function

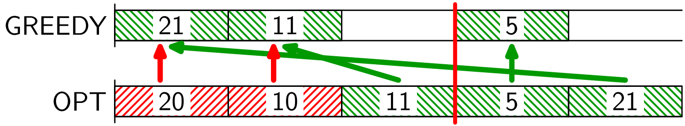


How much "money" we need at a given time and configuration?

We earn $R \cdot w_j$ for running a job and pay w_j if OPT runs a job.

Let $\Phi = \sum_{j \in X} w_j$, where X are the that the algorithm completed but the adversary will schedule in the future.

A potential function



How much "money" we need at a given time and configuration?

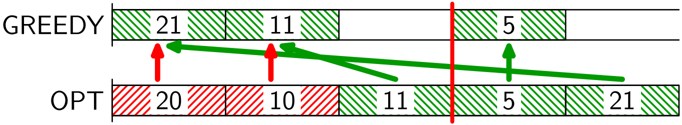
We earn $R \cdot w_j$ for running a job and pay w_j if OPT runs a job.

Let $\Phi = \sum_{j \in X} w_j$, where X are the that the algorithm completed but the adversary will schedule in the future.

To prove that ON is R-competitive, we show that in each step

$$\Phi_{old} + R \cdot \quad w_{ON} \; - w_{OPT} \geq \quad \Phi_{new}$$

A potential function



How much "money" we need at a given time and configuration?

We earn $R \cdot w_j$ for running a job and pay w_j if OPT runs a job.

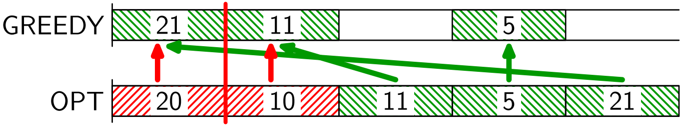
Let $\Phi = \sum_{j \in X} w_j$, where X are the that the algorithm completed but the adversary will schedule in the future.

To prove that ON is R-competitive, we show that in each step

$$\Phi_{old} + R \cdot E[w_{ON}] - w_{OPT} \geq E[\Phi_{new}]$$

Analysis

- At each time, pick uniformly random real $x \in (-1, 0)$.
- Let h be the largest weight of a pending job.
- Among all the pending jobs with $w_j \geq e^x \cdot h$, schedule a job with the earlieast deadline.



$$\Phi_{old} + R \cdot E[w_{ON}] - w_{OPT} \geq E[\Phi_{new}]$$

Theorem

This algorithm is $e/(e-1) \approx 1.58$-competitive. This is optimal against the adaptive online adversary. I.e., it is optimal among the algorithms analyzed using a potential.

Deterministic algorithms I

Charging scheme

Alternating heavy and urgent packets eventually leads to 1.939-competitive algorithm.

Deterministic algorithms I

Charging scheme

Alternating heavy and urgent packets eventually leads to 1.939-competitive algorithm.

Potential function

Can be used to give a 1.828-competitive algorithm.

Deterministic algorithms II

Modifying the optimal schedule

At each step, the configuration of the optimal schedule is made identical with that of the online algorithm, with some advantage to the optimum:

- Schedule a job and keep it pending,
- Schedule two jobs,
- Increase the weight or deadline of some pending job.

Can be used to give a $\phi \approx 1.618$-competitive algorithm for instances with agreeable deadlines.

Deterministic algorithms II

Modifying the optimal schedule

At each step, the configuration of the optimal schedule is made identical with that of the online algorithm, with some advantage to the optimum:

- Schedule a job and keep it pending,
- Schedule two jobs,
- Increase the weight or deadline of some pending job.

Can be used to give a $\phi \approx 1.618$-competitive algorithm for instances with agreeable deadlines.

Weighted queues

There exists a 1.897-competitive algorithm.

Lower bounds

2-bounded instances

- The $\phi \approx 1.618$-competitive deterministic algorithm is optimal.
- There exists a 1.25-competitive randomized algorithm and this is optimal.

No other lower bounds for the general problem are known.

Lower bounds

2-bounded instances

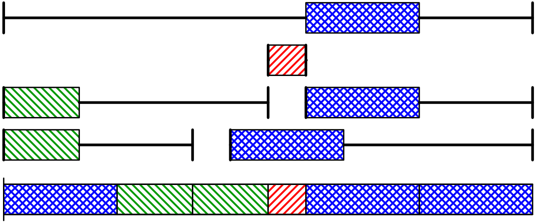
- The $\phi \approx 1.618$-competitive deterministic algorithm is optimal.
- There exists a 1.25-competitive randomized algorithm and this is optimal.

No other lower bounds for the general problem are known.

Open problem

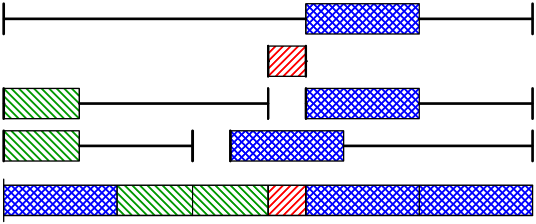
Is the general problem harder than the 2-bounded case?

Offline scheduling



- For unrestricted job lengths, the problem is strongly NP-hard
- For unit jobs ($p_j = 1$) and arbitrary weights we can maximize the weight of scheduled jobs in polynomial time.

Offline scheduling



- Even maximizing the weight for equal-length jobs ($p_j = p$) on a single machine is in P.
- For unit jobs and one more job length ($p_j = \{1, p\}$), we can test if all the jobs can be scheduled.

A linear program

Variables: x_t – the number of long jobs started before time t.
Constraints: For all times s, t:

$$
\begin{aligned}
x_t - x_{t-1} &\geq 0 \\
x_t - x_{t-p} &\leq 1 \\
x_{t+1-p} - x_s &\geq b_{s,t} \\
x_{t+1-p} - x_s &\leq \lfloor (t - s - a_{s,t})/p \rfloor
\end{aligned}
$$

where $a_{s,t}$ and $b_{s,t}$ is the number of short and long jobs, resp., that have to start and complete in $[s, t)$.

A linear program

Variables: x_t – the number of long jobs started before time t.
Constraints: For all times s, t:

$$\begin{aligned} x_t - x_{t-1} &\geq 0 \\ x_t - x_{t-p} &\leq 1 \\ x_{t+1-p} - x_s &\geq b_{s,t} \\ x_{t+1-p} - x_s &\leq \lfloor (t - s - a_{s,t})/p \rfloor \end{aligned}$$

where $a_{s,t}$ and $b_{s,t}$ is the number of short and long jobs, resp., that have to start and complete in $[s, t)$.

Observation

The matrix of the LP is totally unimodular. Thus if the LP is feasible, then there exists an integral solution.

A linear program

Variables: x_t – the number of long jobs started before time t.
Constraints: For all times s, t:

$$
\begin{aligned}
x_t - x_{t-1} &\geq 0 \\
x_t - x_{t-p} &\leq 1 \\
x_{t+1-p} - x_s &\geq b_{s,t} \\
x_{t+1-p} - x_s &\leq \lfloor (t - s - a_{s,t})/p \rfloor
\end{aligned}
$$

where $a_{s,t}$ and $b_{s,t}$ is the number of short and long jobs, resp., that have to start and complete in $[s, t)$.

- A schedule implies a feasible (integral) solution: Easy.

A linear program

Variables: x_t – the number of long jobs started before time t.
Constraints: For all times s, t:

$$\begin{aligned} x_t - x_{t-1} &\geq 0 \\ x_t - x_{t-p} &\leq 1 \\ x_{t+1-p} - x_s &\geq b_{s,t} \\ x_{t+1-p} - x_s &\leq \lfloor (t - s - a_{s,t})/p \rfloor \end{aligned}$$

where $a_{s,t}$ and $b_{s,t}$ is the number of short and long jobs, resp., that have to start and complete in $[s, t)$.

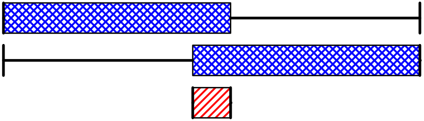
- A schedule implies a feasible (integral) solution: Easy.
- A feasible integral solution implies a schedule: Subtle, holds only for a single machine.

A linear program

Variables: x_t – the number of long jobs started before time t.
Constraints: For all times s, t:

$$\begin{array}{rcl} x_t - x_{t-1} & \geq & 0 \\ x_t - x_{t-p} & \leq & 1 \\ x_{t+1-p} - x_s & \geq & b_{s,t} \\ x_{t+1-p} - x_s & \leq & \lfloor (t - s - a_{s,t})/p \rfloor \end{array}$$

where $a_{s,t}$ and $b_{s,t}$ is the number of short and long jobs, resp., that have to start and complete in $[s, t)$.



Offline scheduling – open problems

Open problems

- If $p_j \in \{2, 3\}$, is it polynomial to decide if all jobs can be scheduled?
- If $p_j \in \{1, 2\}$, is it polynomial to maximize the number of scheduled jobs?

Offline scheduling – open problems

Open problems

- If $p_j \in \{2, 3\}$, is it polynomial to decide if all jobs can be scheduled?
- If $p_j \in \{1, 2\}$, is it polynomial to maximize the number of scheduled jobs?
- For some constant C, is it NP-hard to maximize the weight of the scheduled jobs on instances with $p_j \leq C$ for all jobs?

Offline scheduling – open problems

Open problems

- If $p_j \in \{2, 3\}$, is it polynomial to decide if all jobs can be scheduled?
- If $p_j \in \{1, 2\}$, is it polynomial to maximize the number of scheduled jobs?
- For some constant C, is it NP-hard to maximize the weight of the scheduled jobs on instances with $p_j \leq C$ for all jobs?

THANK YOU!