A $\phi$-Competitive Algorithm for Scheduling Packets with Deadlines

Pavel Veselý
University of Warwick

Joint work with Marek Chrobak (UC Riverside), Łukasz Jeż (Wrocław), and Jiří Sgall (Charles University, Prague)

DIMAP Seminar, October 2
To appear in SODA ’19
Outline

- Introduction to competitive analysis
- Model & result
- Algorithm
- Analysis techniques
- Further research directions
Introduction to competitive analysis
An Example: Cabinetmaker

- Each week you make one cabinet
An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward
An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward

You have two orders on the table:
- \( u \): deadline this week, reward 10,000 CZK
- \( v \): deadline next week, reward 16,180 CZK
An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward
- You have two orders on the table:
  - $u$: deadline this week, reward 10 000 CZK
  - $v$: deadline next week, reward 16 180 CZK

1) If you select $u$, then:
An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward

You have two orders on the table:
- $u$: deadline this week, reward 10 000 CZK
- $v$: deadline next week, reward 16 180 CZK

1) If you select $u$, then:
- new order $v'$ arrives
  - deadline next week, reward 16 180 CZK
An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward

You have two orders on the table:
- $u$: deadline this week, reward 10 000 CZK
- $v$: deadline next week, reward 16 180 CZK

1) If you select $u$, then:
- new order $v'$ arrives
  - deadline next week, reward 16 180 CZK
- only one of $v$ and $v'$ served
An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward

- You have two orders on the table:
  - \( u \): deadline this week, reward 10,000 CZK
  - \( v \): deadline next week, reward 16,180 CZK

1) If you select \( u \), then:
   - new order \( v' \) arrives
     - deadline next week, reward 16,180 CZK
   - only one of \( v \) and \( v' \) served

2) If you select \( v \), then:
   - no order arrives for next week
   - \( u \) expires unserved
   - These are worst-case scenarios
An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward

You have two orders on the table:
- \( u \): deadline this week, reward 10 000 CZK
- \( v \): deadline next week, reward 16 180 CZK

1) If you select \( u \), then:
- new order \( v' \) arrives
  - deadline next week, reward 16 180 CZK
- only one of \( v \) and \( v' \) served

2) If you select \( v \), then:
- no order arrives for next week
- \( u \) expires unserved
An Example: Cabinetmaker

- Each week you make one cabinet
- Customers order cabinets, each order has
  - a deadline
  - a reward

You have two orders on the table:
- \( u \): deadline this week, reward 10,000 CZK
- \( v \): deadline next week, reward 16,180 CZK

1) If you select \( u \), then:
   - new order \( v' \) arrives
     - deadline next week, reward 16,180 CZK
   - only one of \( v \) and \( v' \) served

2) If you select \( v \), then:
   - no order arrives for next week
   - \( u \) expires unserved

These are worst-case scenarios
### Online optimization & algorithms

<table>
<thead>
<tr>
<th><strong>Online computation</strong></th>
<th><strong>Offline computation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## Online optimization & algorithms

<table>
<thead>
<tr>
<th>Online computation</th>
<th>Offline computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input arriving piece by piece</td>
<td>Whole input available at the beginning</td>
</tr>
</tbody>
</table>
Online computation

- Input arriving piece by piece
- Making decisions without knowing future

Offline computation

- Whole input available at the beginning
- All decisions made at once
<table>
<thead>
<tr>
<th><strong>Online computation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Input arriving piece by piece</td>
</tr>
<tr>
<td>- Making decisions without knowing future</td>
</tr>
<tr>
<td>- Decisions irrevocable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Offline computation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Whole input available at the beginning</td>
</tr>
<tr>
<td>- All decisions made at once</td>
</tr>
</tbody>
</table>
Online optimization & algorithms

Online computation
- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

Offline computation
- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
## Online computation
- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

## Offline computation
- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms
Online computation

- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

Offline computation

- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms
Online optimization & algorithms

Online computation
- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

Offline computation
- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms

Online model
- Sequence of events (orders), arrive over time
Online optimization & algorithms

Online computation
- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

Offline computation
- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms

Online model
- Sequence of events (orders), arrive over time
- Algorithm knows only events that arrived so far
### Online optimization & algorithms

**Online computation**
- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

**Offline computation**
- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms

**Online model**
- Sequence of events *(orders)*, arrive over time
- Algorithm knows only events that arrived so far
- Some events ask to make decisions *(Monday mornings)*
Online optimization & algorithms

### Online computation
- Input arriving piece by piece
- Making decisions without knowing future
- Decisions irrevocable
- Cannot be optimal (usually)

### Offline computation
- Whole input available at the beginning
- All decisions made at once
- Find an optimal solution
- Time/memory efficient algorithms

### Online model
- Sequence of events (orders), arrive over time
- Algorithm knows only events that arrived so far
- Some events ask to make decisions (Monday mornings)
- Decisions influence the objective function (rewards served orders)
Competitive ratio of online algorithms

- Worst-case ratio between
  - value of the optimum solution $OPT$ and
  - value of the algorithm’s solution $ALG$
Competitive ratio of online algorithms

- Worst-case ratio between
  - value of the optimum solution \( \text{OPT} \) and
  - value of the algorithm’s solution \( \text{ALG} \)

- Algorithm is \( R \)-competitive if for any instance \( I \)

\[
\text{OPT}(I) \leq R \cdot \text{ALG}(I)
\]

(assuming maximization)
Competitive ratio of online algorithms

- Worst-case ratio between
  - value of the optimum solution OPT and
  - value of the algorithm’s solution ALG

- Algorithm is $R$-competitive if for any instance $I$

\[
\text{OPT}(I) \leq R \cdot \text{ALG}(I)
\]

(assuming maximization)

- Game: the algorithm vs. an adversary
  - The adversary decides on further input to maximize OPT/ALG
Model & Result
Online Packet Scheduling with Deadlines

Network switch:

Input port → Buffer → Output port
Network switch:

- Packets arrive over time
- Each has a deadline and a weight

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_p$</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$r_p$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$d_p$</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
Online Packet Scheduling with Deadlines

Network switch:

- Packets arrive over time
- Each has a deadline and a weight
- Time discrete, consisting of *slots* or *steps*
- One packet transmitted in each step

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_p$</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$r_p$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$d_p$</td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

Goal: maximize total weight of scheduled packets
Packets arrive over time
Each has a deadline and a weight
Time discrete, consisting of slots or steps
One packet transmitted in each step
Goal: maximize total weight of scheduled packets
Online Packet Scheduling with Deadlines

Network switch:

- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of slots or steps
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets

\[
\begin{array}{c|c|c|c}
1 & 2 & 3 \\
\hline
w_p & 2 \\
\hline
r_p & 1 & d_p = 3 \\
\end{array}
\]
Online Packet Scheduling with Deadlines

Network switch:

- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of slots or steps
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ w_p = 2 \quad r_p = 1 \quad d_p = 3 \]

ALG
OPT
Packets arrive over time (orders)
Each has a deadline and a weight (reward)
Time discrete, consisting of slots or steps
One packet transmitted in each step (weeks)
Goal: maximize total weight of scheduled packets
Online Packet Scheduling with Deadlines

Network switch:

- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of slots or steps
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets

\[ \frac{OPT}{ALG} = \frac{1.7 + 1}{1.7} \approx 1.59 \]
Network switch:

- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of slots or steps
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Packets are scheduled as follows:

- Packet 1 first, scheduled in step 1.
- Packet 2 is scheduled in step 3.
- Packet 3 is scheduled in step 1.
- Packet 4 is scheduled in step 3.

Weights:

- $w_p = 2$
- $r_p = 1$
- $d_p = 3$

Optimal scheduling:

- ALG: 1, 4, 3
- OPT: 1, 2, 3

Scheduling problem:

$$\sum w_j (1 - U_j)$$

A.k.a. Buffer Management in Quality of Service Switches
Online Packet Scheduling with Deadlines

Network switch:

- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of slots or steps
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$w_p = 2$  
$r_p = 1$  
$d_p = 3$
Network switch:

- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of slots or steps
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
1 & & & \\
1.7 & & & \\
1.7 & & & \\
2.7 & & & \\
\end{array}
\]
Online Packet Scheduling with Deadlines

Network switch:

- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of slots or steps
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets

\[
\text{Opt} = 2 \cdot 1.7 + 2.7 \\\n\text{ALG} = 1.7 + 2.7 \\
\frac{\text{Opt}}{\text{ALG}} = \frac{2 \cdot 1.7 + 2.7}{1.7 + 2.7} \approx 1.39
\]
Packets arrive over time (orders)
Each has a deadline and a weight (reward)
Time discrete, consisting of slots or steps
One packet transmitted in each step (weeks)
Goal: maximize total weight of scheduled packets
Packets arrive over time (orders)
Each has a deadline and a weight (reward)
Time discrete, consisting of slots or steps
One packet transmitted in each step (weeks)
Goal: maximize total weight of scheduled packets

Scheduling problem 1|online, r_j, p_j = 1| \sum w_j(1 - U_j)
Network switch:

- Packets arrive over time (orders)
- Each has a deadline and a weight (reward)
- Time discrete, consisting of slots or steps
- One packet transmitted in each step (weeks)
- Goal: maximize total weight of scheduled packets

Scheduling problem $1|\text{online}, r_j, p_j = 1| \sum w_j (1 - U_j)$
A.k.a. Buffer Management in Quality of Service Switches
Previous work

- We focus on deterministic algorithms

\[ \phi = \frac{1 + \sqrt{5}}{2} \approx 1.618 \]  
\[ 1 + \frac{1}{\phi} = \frac{\phi}{1} = \phi \approx 1.828 \]  
\[ \phi \text{-competitive algorithms for some special instances} \]  
\[ \text{Is there a } \phi \text{-competitive algorithm? Yes!} \]
Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet

Lower bound of the golden ratio $\phi = \frac{1}{2} (\sqrt{5} + 1) \approx 1.618$ [Hajek '01, Andelman et al. '03, Chin & Fung '03]

$1 + 1/\phi = \phi^2 \sqrt{2} - 1 \approx 1.828$-competitive algorithm [Englert & Westermann '07]

$\phi$-competitive algorithms for some special instances [Kesselman et al. '01, Chin et al. '04, Li et al. '05, Bienkowski et al. '13, Böhm et al. '16]

Is there a $\phi$-competitive algorithm?

Yes!
Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio
  \[ \phi = \frac{1}{2}(\sqrt{5} + 1) \approx 1.618 \] [Hájek '01, Andelman et al. '03, Chin & Fung '03]

\[ 1 + \frac{1}{\phi} = \phi \]
Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio
  \[
  \phi = \frac{1}{2}(\sqrt{5} + 1) \approx 1.618 \quad [\text{Hájek '01, Andelman et al. '03, Chin & Fung '03}]
  \]
  \[1 + \frac{1}{\phi} = \phi\]

\[
\begin{array}{c}
1 \\
\phi \\
\phi^2 \\
\phi^3 \\
\vdots
\end{array}
\]
Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio
  \[
  \phi = \frac{1}{2}(\sqrt{5} + 1) \approx 1.618 \quad \text{[Hájek '01, Andelman et al. '03, Chin & Fung '03]}
  \]
  \[
  1 + \frac{1}{\phi} = \phi
  \]
- \(2\sqrt{2} - 1 \approx 1.828\)-competitive algorithm \([\text{Englert & Westermann '07}]\)
Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio
  \[
  \phi = \frac{1}{2} (\sqrt{5} + 1) \approx 1.618 \quad \text{[Hájek '01, Andelman et al. '03, Chin & Fung '03]}
  \]
  \[
  1 + \frac{1}{\phi} = \phi
  \]
- \(2\sqrt{2} - 1 \approx 1.828\)-competitive algorithm \([\text{Englert & Westermann '07}]\)
- \(\phi\)-competitive algorithms for some special instances \([\text{Kesselman et al. '01, Chin et al. '04, Li et al. '05, Bienkowski et al. '13, Böhm et al. '16}]\)
Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio
  \[ \phi = \frac{1}{2}(\sqrt{5} + 1) \approx 1.618 \] [Hájek '01, Andelman et al. '03, Chin & Fung '03]
  \[ 1 + \frac{1}{\phi} = \phi \]
- \[ 2\sqrt{2} - 1 \approx 1.828 \]-competitive algorithm [Englert & Westermann '07]
- \( \phi \)-competitive algorithms for some special instances [Kesselman et al. '01, Chin et al. '04, Li et al. '05, Bienkowski et al. '13, Böhm et al. '16]

Is there a \( \phi \)-competitive algorithm?
Previous work

- We focus on deterministic algorithms
- Greedy algorithm 2-competitive
  - Schedules always the heaviest pending packet
- Lower bound of the golden ratio
  \[ \phi = \frac{1}{2}(\sqrt{5} + 1) \approx 1.618 \]  
  \[ 1 + \frac{1}{\phi} = \phi \]

- \(2\sqrt{2} - 1 \approx 1.828\)-competitive algorithm [Englert & Westermann '07]
- \(\phi\)-competitive algorithms for some special instances [Kesselman et al. '01, Chin et al. '04, Li et al. '05, Bienkowski et al. '13, Böhm et al. '16]

Is there a \(\phi\)-competitive algorithm?

Yes!
Theorem

There is a $\phi$-competitive deterministic algorithm.
New result

Theorem

There is a $\phi$-competitive deterministic algorithm.

Key technique: Plan

- Max-weight feasible subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t+1, \ldots$
New result

Theorem

There is a $\phi$-competitive deterministic algorithm.

Key technique: Plan

- Max-weight feasible subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t+1, \ldots$
- Optimal future profit unless new packets arrive
New result

Theorem

There is a $\phi$-competitive deterministic algorithm.

Key technique: Plan

- Max-weight feasible subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t + 1, \ldots$
- Optimal future profit unless new packets arrive
- Scheduled plans (a.k.a. provisional schedules) used already by
  [Li et al. '05, Li et al. '07, Englert & Westermann '07]
Algorithm
Algorithm $\text{PLAN}(\phi)$

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t+1, \ldots$
Algorithm $\text{PLAN}(\phi)$

Plan $\mathcal{P}$
- Max-weight $\textit{feasible}$ subset of pending packets in step $t$
  - $\text{feasible} = \text{can be scheduled in slots } t, t+1, \ldots$

Algorithm $\text{PLAN}(\phi)$
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(\mathcal{Q}_p)$

Very elegant algorithm . . . but $\phi$-competitive
Algorithm $\text{PLAN}(\phi)$

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t+1, \ldots$

Algorithm $\text{PLAN}(\phi)$

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
Algorithm $\text{PLAN}(\phi)$

**Plan $\mathcal{P}$**
- Max-weight *feasible* subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t + 1, \ldots$

**Algorithm $\text{PLAN}(\phi)$**
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
  - $w_p$ is the gain in this step
  - $w(Q_p)$ is the optimal *future* profit unless new packets arrive
**Algorithm** $\text{PLAN}(\phi)$

**Plan $\mathcal{P}$**
- Max-weight *feasible* subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t+1, \ldots$

**Algorithm  $\text{PLAN}(\phi)$**
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
  - $w_p$ is the gain in this step
  - $w(Q_p)$ is the optimal *future* profit unless new packets arrive

- Very elegant algorithm . . .
**Algorithm PLAN(φ)**

**Plan P**
- Max-weight *feasible* subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t+1, \ldots$

**Algorithm PLAN(φ)**
- Schedule packet $p \in P$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
  - $w_p$ is the gain in this step
  - $w(Q_p)$ is the optimal *future* profit unless new packets arrive

- Very elegant algorithm . . .
- . . . but not $\phi$-competitive
Plan and its Structure

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step $t$
  - feasible $=$ can be scheduled in slots $t, t + 1, \ldots$
Plan and its Structure

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t + 1, \ldots$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t+1, \ldots$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Plan:  0.5  1  1.6  0.6  0.1
Plan and its Structure

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t + 1, \ldots$

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Plan: 1 0.5 1.6 0.6 0.1

Definition

Slot $\tau$ is tight w.r.t. plan $\mathcal{P}$ iff

\[ \text{# of slots till } \tau = \text{# of packets } j \in \mathcal{P}: d_j \leq \tau \]
Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t + 1, \ldots$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Plan: 1.6 0.5 1 0.6 0.1
Plan and its Structure

Plan $\mathcal{P}$

- Max-weight feasible subset of pending packets in step $t$
  - feasible $=$ can be scheduled in slots $t, t+1, \ldots$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Plan: 1.6 0.5 1 0.6 0.1

\[\uparrow \uparrow \uparrow\]

tight

Definition

Slot $\tau$ is **tight** w.r.t. plan $\mathcal{P}$ iff

$\#$ of slots till $\tau = \#$ of packets $j \in \mathcal{P} : d_j \leq \tau$
Plan and its Structure

Plan $\mathcal{P}$

- Max-weight feasible subset of pending packets in step $t$
  - feasible = can be scheduled in slots $t, t + 1, \ldots$

Plan:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Plan: 1.6 0.5 1 0.6 0.1
Segments: 

Definition

Slot $\tau$ is tight w.r.t. plan $\mathcal{P}$ iff

$\#$ of slots till $\tau = \#$ of packets $j \in \mathcal{P}: d_j \leq \tau$

Definition

Segment = interval between tight slots
Plan and its Structure

Plan $\mathcal{P}$

- Max-weight *feasible* subset of pending packets in step $t$
  - feasible $=$ can be scheduled in slots $t, t+1, \ldots$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Plan: 1.6 0.5 1 0.6 0.1

Segments:

Definition

Slot $\tau$ is *tight* w.r.t. plan $\mathcal{P}$ iff

$\#$ of slots till $\tau = \#$ of packets $j \in \mathcal{P} : d_j \leq \tau$

Definition

*Segment* = interval between tight slots
Plan Updates After Packet \( p \) isScheduled

\( p \) in the 1st segment ("greedy step")

\[
\begin{align*}
\mathcal{P} & \quad d_p \\
\mathcal{Q}_p & \quad d_p 
\end{align*}
\]
Plan Updates After Packet $p$ is Scheduled

$p$ in the 1st segment ("greedy step")

$p$ in a later segment ("leap step")

$Pavel$ $Veselý$
Plan Updates After Packet $p$ is Scheduled

$p$ in the 1st segment ("greedy step")

$p$ in a later segment ("leap step")

$\ell = \text{lightest in the 1st segment}$
Plan Updates After Packet $p$ is Scheduled

$p$ in the 1st segment (“greedy step”)

$p$ in a later segment (“leap step”)

- $\ell = \text{lightest in the 1st segment}$
- $\varrho = \text{heaviest not in } \mathcal{P} \text{ which can replace } p$

$\triangleright \text{replacement packet for } p$
Problem of $\text{PLAN}(\phi)$: Weight Decreases in the Plan
Problem of $\text{PLAN}(\phi)$: Weight Decreases in the Plan

- $\text{minwt}(\tau) =$ min-weight in $\mathcal{P}$ till the next tight slot after $\tau$
Problem of $\text{PLAN}(\phi)$: Weight Decreases in the Plan

- $\text{minwt}(\tau) = \text{min-weight in } \mathcal{P} \text{ till the next tight slot after } \tau$
  - In a schedule of $\mathcal{P}$, any packet can be in the 1st slot of a segment

![Diagram showing a schedule with a segment and a slot labeled $\tau$.]
Problem of $\text{PLAN}(\phi)$: Weight Decreases in the Plan

- $\text{minwt}(\tau) = \text{min-weight in } \mathcal{P} \text{ till the next tight slot after } \tau$
  - In a schedule of $\mathcal{P}$, any packet can be in the 1st slot of a segment

![Diagram showing a schedule with slots labeled $d_\ell$, $\ell$, $\tau$]
Problem of \textbf{PLAN}(\phi): Weight Decreases in the Plan

- \(\text{minwt}(\tau) = \text{min-weight in } \mathcal{P} \text{ till the next tight slot after } \tau\)
  - In a schedule of \(\mathcal{P}\), any packet can be in the 1st slot of a segment

![Diagram showing a schedule with segments and slots with \(\tau\) and \(d_\ell\)]
Problem of $\text{PLAN}(\phi)$: Weight Decreases in the Plan

- $\text{minwt}(\tau) = \text{min-weight in } \mathcal{P} \text{ till the next tight slot after } \tau$
  - In a schedule of $\mathcal{P}$, any packet can be in the 1st slot of a segment
Problem of $\text{PLAN}(\phi)$: Weight Decreases in the Plan

- $\minwt(\tau) = \text{min-weight in } P \text{ till the next tight slot after } \tau$
  - In a schedule of $P$, any packet can be in the 1st slot of a segment

[minwt after plan updates]

- $\minwt(\tau)$ does not decrease for any $\tau$:
  - after arrival of a new packet
  - after scheduling a packet from the 1st segment (greedy step)
Problem of $\text{PLAN}(\phi)$: Weight Decreases in the Plan

- $\text{minwt}(\tau) = \text{min-weight in } P \text{ till the next tight slot after } \tau$
  - In a schedule of $P$, any packet can be in the 1st slot of a segment

\begin{itemize}
  \item $\text{minwt}(\tau)$ does not decrease for any $\tau$:
    \begin{itemize}
      \item after arrival of a new packet
      \item after scheduling a packet from the 1st segment (greedy step)
    \end{itemize}
  \item $\text{minwt}(\tau)$ decreases for some $\tau$ after sch. a packet from later segment
\end{itemize}
Problem of $\text{PLAN}(\phi)$: Weight Decreases in the Plan

- $\text{minwt}(\tau) = \text{min-weight in } P \text{ till the next tight slot after } \tau$
  - In a schedule of $P$, any packet can be in the 1st slot of a segment

\[ \text{minwt}(\tau) \]

\[ t \]

\[ \tau \]

\text{minwt after plan updates}

- $\text{minwt}(\tau)$ does not decrease for any $\tau$:
  - after arrival of a new packet
  - after scheduling a packet from the 1st segment (greedy step)

- $\text{minwt}(\tau)$ decreases for some $\tau$ after scheduling a packet from later segment

$P$:

\[
\begin{array}{c}
\text{l} \\
\text{p} \\
\text{d_p} \\
\text{d_q} \\
\hline
\end{array}
\]

$Q_p$:

\[
\begin{array}{c}
\text{d_p} \\
\text{d_q} \\
\hline
\end{array}
\]

The problem:

$\varrho \notin P \Rightarrow w_{\varrho} < \text{minwt}(d_q)$
Solution: Maintaining Slot-Monotonicity of \text{minwt}

- Idea: modify \text{PLAN}(\phi) so that \text{minwt}(\tau) never decreases for any \tau
Solution: Maintaining Slot-Monotonicity of minwt

- Idea: modify \( \text{PLAN}(\phi) \) so that \( \text{minwt}(\tau) \) never decreases for any \( \tau \)

The problem:
\[ \varrho \not\in \mathcal{P} \Rightarrow w_{\varrho} < \text{minwt}(d_{\varrho}) \]
Solution: Maintaining Slot-Monotonicity of \( \text{minwt} \)

- **Idea:** modify \( \text{PLAN}(\phi) \) so that \( \text{minwt}(\tau) \) never decreases for any \( \tau \)

The problem:
\[
\varrho \not\in \mathcal{P} \Rightarrow w_\varrho < \text{minwt}(d_\varrho)
\]

\[
\begin{array}{c}
\mathcal{P} \\
\hline
\ell & p \\
\hline
\vdots & \vdots \\
\hline
\end{array}
\]

\[
\begin{array}{c}
Q_\varrho \\
\hline
\varrho \\
\hline
\vdots & \vdots \\
\hline
\end{array}
\]

- Increase the weight of \( \varrho \) to \( \text{minwt}(d_\varrho) \)
Solution: Maintaining Slot-Monotonicity of $\text{minwt}$

- **Idea:** modify $\text{PLAN}(\phi)$ so that $\text{minwt}(\tau)$ never decreases for any $\tau$.

  - The problem:
    
    $\varrho \not\in \mathcal{P} \Rightarrow w_\varrho < \text{minwt}(d_\varrho)$

  - $\Rightarrow$ increase the weight of $\varrho$ to $\text{minwt}(d_\varrho)$

  - Not enough if segments merge:
Solution: Maintaining Slot-Monotonicity of minwt

- Idea: modify $\text{PLAN}(\phi)$ so that $\text{minwt}(\tau)$ never decreases for any $\tau$

The problem:

$$\varrho \not\in P \Rightarrow w_\varrho < \text{minwt}(d_\varrho)$$

⇒ increase the weight of $\varrho$ to $\text{minwt}(d_\varrho)$

Not enough if segments merge:

$$\varrho \not\in P \Rightarrow w_\varrho < \text{minwt}(d_\varrho)$$

- $t$
- $t + 1$
Solution: Maintaining Slot-Monotonicity of minwt

- Idea: modify PLAN($\phi$) so that $\text{minwt}(\tau)$ never decreases for any $\tau$

  The problem:
  $q \notin \mathcal{P} \Rightarrow w_q < \text{minwt}(d_q)$

- $\Rightarrow$ increase the weight of $q$ to $\text{minwt}(d_q)$

- Not enough if segments merge:
  $\Rightarrow$ avoid merging segments
Algorithm $\text{PLANM}(\phi)$ Maintaining Slot-Monotonicity

- Schedule packet $p \in P$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
Algorithm $\text{PLANM}(\phi)$ Maintaining Slot-Monotonicity

- Schedule packet $p \in P$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
- If $p$ is not in the 1st segment of $P$ (leap step):
  - Increase the weight of $\varrho$ to $\minwt(d_\varrho)$
Algorithm $\text{PLANM}(\phi)$ Maintaining Slot-Monotonicity

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
- If $p$ is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of $\varrho$ to $\minwt(d_\varrho)$
  - Avoid merging segments:

\[
\begin{array}{cccccccccc}
\mathcal{P} & & & & & & & & & \\
\vdots & & d_p & & & & & & d_\varrho & \\
\varrho & & & & & & & & \varrho & \\
\mathcal{Q}_p & & \vdots & & & \varrho & & \vdots & \\
\end{array}
\]
Algorithm $\text{PLANM}(\phi)$ Maintaining Slot-Monotonicity

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
- If $p$ is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of $\varrho$ to $\minwt(d_{\varrho})$
  - Avoid merging segments:

$$
\begin{array}{ccccccc}
\mathcal{P} & \cdots & d_p & \tau_0 & h_1 & \cdots & d_{\varrho} & \gamma \\
\mathcal{Q}_p & \cdots & & & \varrho & \cdots \\
\end{array}
$$

- $h_1 =$ heaviest packet in $(\tau_0, \gamma]$,
Algorithm $\text{PLANM}(\phi)$ Maintaining Slot-Monotonicity

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
- If $p$ is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of $\varrho$ to $\text{minwt}(d_\varrho)$
  - Avoid merging segments:

\[
\begin{array}{cccccc}
\mathcal{P} & d_p & \tau_0 & \tau_1 & \cdots & d_\varrho & \gamma \\
\hline
\cdots & p & h_1 & \cdots & \cdots & \cdots & \cdots \\
\end{array}
\]

\[
\begin{array}{cccccc}
\mathcal{Q}_p & \cdots & h_1 & \cdots & \cdots & \varrho & \cdots \\
\end{array}
\]

- $h_1 = \text{heaviest packet in } (\tau_0, \gamma]$,
- decrease deadline of $h_1$ to $\tau_0$
Algorithm $\text{PLANM}(\phi)$ Maintaining Slot-Monotonicity

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
- If $p$ is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of $\varrho$ to $\minwt(d_\varrho)$
  - Avoid merging segments:

$$
\begin{array}{c|c|c|c|c|c}
\mathcal{P} & \cdots & d_p & \tau_0 & \tau_1 & \tau_2 & \cdots \\
\hline 
\mathcal{Q}_p & \cdots & h_1 & h_2 & \varrho & \cdots \\
\end{array}
$$

- $h_2 = \text{heaviest packet in } (\tau_1, \gamma]$,
- decrease deadline of $h_2$ to $\tau_1$
Algorithm $\text{PLANM}(\phi)$ Maintaining Slot-Monotonicity

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
- If $p$ is not in the 1st segment of $\mathcal{P}$ (leap step):
  - Increase the weight of $\varrho$ to $\min wt(d_\varrho)$
  - Avoid merging segments:

\[
\begin{array}{cccccccc}
\mathcal{P} & \cdots & d_p & \tau_0 & \tau_1 & \tau_2 & \tau_3 & d_\varrho & \gamma \\
\mathcal{Q}_p & \cdots & h_1 & h_2 & h_3 & \varrho & \cdots \\
\end{array}
\]

- $h_3 =$ heaviest packet in $(\tau_2, \gamma]$,
- decrease deadline of $h_3$ to $\tau_2$
Algorithm \textbf{PLANM}(\(\phi\)) Maintaining Slot-Monotonicity

- Schedule packet \(p \in \mathcal{P}\) maximizing \(\phi \cdot w_p + w(Q_p)\)
  - \(Q_p\) is the plan after \(p\) is scheduled and time is incremented (\(p \notin Q_p\))
- If \(p\) is not in the 1st segment of \(\mathcal{P}\) (leap step):
  - Increase the weight of \(\varrho\) to \(\text{minwt}(d_\varrho)\)
  - Avoid merging segments:

\[
\begin{array}{cccccccc}
\mathcal{P} & \cdots & p & h_1 & h_2 & h_3 & \cdots & h_{k-1} & h_k \\
Q_p & \cdots & h_1 & h_2 & h_3 & h_4 & \cdots & h_k & \varrho
\end{array}
\]

- for \(i = 1, 2, \ldots\) : \(h_i = \text{heaviest packet in } (\tau_{i-1}, \gamma]\),
- decrease deadline of \(h_i\) to \(\tau_{i-1}\)
- stop when \(\tau_i = \gamma\)
Algorithm $\text{PLANM}(\phi)$ Maintaining Slot-Monotonicity

- Schedule packet $p \in P$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
- If $p$ is not in the 1st segment of $P$ (leap step):
  - Increase the weight of $\varrho$ to $\min wt(d_{\varrho})$
  - Avoid merging segments:

<table>
<thead>
<tr>
<th>$P$</th>
<th>$d_p$</th>
<th>$\tau_0$</th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$\tau_{k-1}$</th>
<th>$d_{\varrho}$</th>
<th>$\gamma = \tau_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\cdots$</td>
<td>$p$</td>
<td>$h_1$</td>
<td>$h_2$</td>
<td>$h_3$</td>
<td>$\cdots$</td>
<td>$h_{k-1}$</td>
<td>$h_k$</td>
<td>$\varrho$</td>
</tr>
</tbody>
</table>

  | $Q_p$ | $\cdots$ | $h_1$ | $h_2$ | $h_3$ | $h_4$ | $\cdots$ | $h_k$ | $\varrho$ | $\cdots$ |

  - for $i = 1, 2, \ldots$: $h_i = \text{heaviest packet in } (\tau_{i-1}, \gamma]$,
  - decrease deadline of $h_i$ to $\tau_{i-1}$
  - stop when $\tau_i = \gamma$
  - ensure: $w_{h_i} \geq \min wt(\tau_{i-1})$
    - if $w_{h_i} < \min wt(\tau_{i-1})$, then set new weight of $h_i$ to $\min wt(\tau_{i-1})$
Algorithm \textsc{PlanM}(\phi) Maintaining Slot-Monotonicity

- Schedule packet \( p \in \mathcal{P} \) maximizing \( \phi \cdot w_p + w(Q_p) \)
  - \( Q_p \) is the plan after \( p \) is scheduled and time is incremented (\( p \notin Q_p \))
- If \( p \) is not in the 1st segment of \( \mathcal{P} \) (leap step):
  - Increase the weight of \( \varrho \) to \( \minwt(d_\varrho) \)

\textbf{In a nutshell}

Avoid merging segments and \( \minwt \) decreases in a right way
- Done by decreasing deadlines and increasing weights of certain packets
Analysis
Analysis Overview

- Competitive analysis
  - Goal: $w(\text{OPT}) \leq \phi \cdot w(\text{ALG})$ for any instance
Analysis Overview

- Competitive analysis
  - Goal: \( w(\text{OPT}) \leq \phi \cdot w(\text{ALG}) \) for any instance
  - Game between algorithm and adversary
    - Adversary schedules packets from OPT
Analysis Overview

- Competitive analysis
  - Goal: \( w(\text{OPT}) \leq \phi \cdot w(\text{ALG}) \) for any instance
  - Game between algorithm and adversary
    - Adversary schedules packets from OPT

Amortization Techniques

1. Increasing weights
   - Algorithm’s future profit may get higher
Analysis Overview

- Competitive analysis
  - Goal: \( w(\text{OPT}) \leq \phi \cdot w(\text{ALG}) \) for any instance
  - Game between algorithm and adversary
    - Adversary schedules packets from OPT

Amortization Techniques

1. Increasing weights
   - Algorithm’s future profit *may* get higher
     - Decrease algorithm’s current profit by weight increase
Analysis Overview

- Competitive analysis
  - Goal: $w(\text{OPT}) \leq \phi \cdot w(\text{ALG})$ for any instance
  - Game between algorithm and adversary
    - Adversary schedules packets from OPT

Amortization Techniques

1. Increasing weights
   - Algorithm’s future profit may get higher
     - Decrease algorithm’s current profit by weight increase

2. Potential function

3. Modifications of the adversary (optimal) schedule ADV
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots
**Adversary Schedule ADV**

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>ADV</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
</tr>
</tbody>
</table>

Adversary’s gain: Packet in ADV can be replaced by another packet, fictitious "treasure packet". Adversary’s gain increased by total weight decrease in ADV. Fictitious "treasure packet" not pending for the algorithm. Tied to a slot $\tau$ in ADV, no release time or deadline, never changes in future. Deposit of profit to be collected by the adversary. Weight bounded by $\minwt(\tau)$. Slot-monotonicity: $\minwt(\tau)$ never decreases.

Invariant (A): ADV consists of two types of packets: *(real) packets in plan $P$* all other packets are treasures.
**Adversary Schedule ADV**

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>ADV</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPT</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Adversary’s gain: $[\text{another lighter packet}, \text{fictitious “treasure packets”}]$

- Adversary’s gain increased by total weight decrease in ADV
- Fictitious “treasure packet” is not pending for the algorithm
- Tied to a slot $\tau$ in ADV, no release time or deadline, never changes in future
- Deposit of profit to be collected by the adversary
- Weight bounded by $\text{minwt}(\tau)$
- Slot-monotonicity: $\text{minwt}(\tau)$ never decrease

Invariant (A)

ADV consists of two types of packets:
- (real) packets in plan $P$
- all other packets are treasures
### Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADV</td>
<td>b</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPT</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Adversary’s gain:

- The adversary can replace packets in ADV with other packets, fictitious "treasure packets".
- Adversary’s gain increased by the total weight decrease in ADV.
- Fictitious "treasure packet" is not pending for the algorithm.
- Tied to a slot $\tau$ in ADV, no release time or deadline, never changes in future.
- Deposit of profit to be collected by the adversary.
- Weight bounded by $\minwt(\tau)$.
- Slot-monotonicity: $\minwt(\tau)$ never decrease.

**Invariant (A)**

ADV consists of two types of packets:

- (real) packets in plan $P$
- all other packets are treasures
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>ADV</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 \ a</td>
<td>1 \ a</td>
</tr>
<tr>
<td>2 \ b</td>
<td>2 \ b</td>
</tr>
<tr>
<td>3 \ d</td>
<td>3 \ c</td>
</tr>
<tr>
<td>4 \</td>
<td>4 \ d</td>
</tr>
<tr>
<td>5 \</td>
<td>5 \ e</td>
</tr>
</tbody>
</table>

Adversary’s gain:

- Packet in ADV can be replaced by another lighter packet, fictitious "treasure packets"
- Adversary’s gain increased by total weight decrease in ADV

- Fictitious "treasure packets": Not pending for the algorithm
- Tied to a slot $\tau$ in ADV, no release time or deadline, never changes in future
- Deposit of profit to be collected by the adversary
- Weight bounded by $\minwt(\tau)$
- Slot-monotonicity: $\minwt(\tau)$ never decreases

**Invariant (A)**

ADV consists of two types of packets:

(1) (real) packets in plan $P$
(2) all other packets are treasures
**Adversary Schedule ADV**

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADV</td>
<td>b</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPT</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Adversary’s gain: $w_a$
## Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th></th>
<th>ADV</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>e</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>e</td>
</tr>
</tbody>
</table>

Adversary’s gain: $w_a$
### Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>ADV</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADV</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Adversary’s gain: $w_a + w_b$

**Fictitious “treasure packet”**

- Not pending for the algorithm
- Tied to a slot $\tau$ in ADV, no release time or deadline, never changes in future
- Deposit of profit to be collected by the adversary

**Weight bounded by** $\text{minwt}(\tau)$

**Slot-monotonicity:** $\text{minwt}(\tau)$ never decrease
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADV</td>
<td></td>
<td></td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>OPT</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Adversary’s gain: $w_a + w_b$

- Packet in ADV can be replaced by \(\left\{\text{another lighter packet}, + w_a \right\}$
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>ADV</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c</td>
<td>d</td>
<td>e</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OPT</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
</table>

Adversary’s gain: \( w_a + w_b \)

- Packet in ADV can be replaced by
  \( \left\{ \text{another lighter packet, fictitious "treasure packets"} \right\} \)
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>ADV</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adversary’s gain: \( w_a + w_b \)

- Packet in ADV can be replaced by \( \{ \text{another lighter packet, fictitious “treasure packets”} \} \)

- Adversary’s gain increased by total weight decrease in ADV
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADV</td>
<td></td>
<td></td>
<td>c</td>
<td>f</td>
<td>e</td>
</tr>
<tr>
<td>OPT</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Adversary’s gain: \( w_a + w_b + (w_d - w_f) \)

- Packet in ADV can be replaced by \{ another lighter packet, fictitious “treasure packets” \}

- Adversary’s gain increased by total weight decrease in ADV
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ADV

OPT

Adversary’s gain: \( w_a + w_b + (w_d - w_f) + (w_c - w_{t_c}) \)

- Packet in ADV can be replaced by \( \{ \) another lighter packet, fictitious “treasure packets” \( \} \)
- Adversary’s gain increased by total weight decrease in ADV
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>ADV</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
</tr>
</tbody>
</table>

Adversary’s gain: \( w_a + w_b + (w_d - w_f) + (w_c - w_{t_c}) + w_{t_c} \)

- Packet in ADV can be replaced by \{ another lighter packet, fictitious “treasure packets” \}

- Adversary’s gain increased by total weight decrease in ADV

Fictitious “treasure packet”

Not pending for the algorithm

Tied to a slot \( \tau \) in ADV, no release time or deadline, never changes in future

Deposit of profit to be collected by the adversary

Weight bounded by \( \min_{\tau} w(\tau) \)

Slot-monotonicity: \( \min_{\tau} w(\tau) \) never decrease

Invariant (A)

ADV consists of two types of packets:

\{ (real) packets in plan \( P \) all other packets are treasures \}
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>ADV</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>OPT</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

Packet in ADV can be replaced by:
- another lighter packet,
- fictitious “treasure packets”

Adversary’s gain increased by total weight decrease in ADV

Fictitious “treasure packet”

- Not pending for the algorithm

Adversary’s gain: \[ w_a + w_b + (w_d - w_f) + (w_c - w_{t_c}) + w_{t_c} \]
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>ADV</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
</tr>
</tbody>
</table>

- Adversary’s gain: $w_a + w_b + (w_d - w_f) + (w_c - w_{t_c}) + w_{t_c}$

- Packet in ADV can be replaced by another *lighter* packet, fictitious “treasure packets”

- Adversary’s gain increased by total weight decrease in ADV

Fictitious “treasure packet”

- *Not* pending for the algorithm

- Tied to a slot $\tau$ in ADV, no release time or deadline, never changes in future
**Adversary Schedule ADV**

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>ADV</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
</tr>
</tbody>
</table>

- Packet in ADV can be replaced by \{ another lighter packet, fictitious “treasure packets” \}

- Adversary’s gain increased by total weight decrease in ADV

**Fictitious “treasure packet”**

- *Not* pending for the algorithm
- Tied to a slot \( \tau \) in ADV, no release time or deadline, never changes in future
- Deposit of profit to be collected by the adversary

Adversary’s gain: \( w_a + w_b + (w_d - w_f) + (w_c - w_{tc}) + w_{tc} \)
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADV</td>
<td></td>
<td>f</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>OPT</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Adversary’s gain: \( w_a + w_b + (w_d - w_f) + (w_c - w_{t_c}) + w_{t_c} \)

- Packet in ADV can be replaced by another lighter packet, fictitious “treasure packets”
- Adversary’s gain increased by total weight decrease in ADV

Fictitious “treasure packet”

- Not pending for the algorithm
- Tied to a slot \( \tau \) in ADV, no release time or deadline, never changes in future
- Deposit of profit to be collected by the adversary
- Weight bounded by \( \text{minwt}(\tau) \)
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots

<table>
<thead>
<tr>
<th>ADV</th>
<th>OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
</tr>
</tbody>
</table>

ADV consists of already-released packets from OPT in future slots. The adversary’s gain is calculated as follows:

\[
w_a + w_b + (w_d - w_f) + (w_c - w_{tc}) + w_{tc}
\]

- Packet in ADV can be replaced by another lighter packet, fictitious “treasure packets”
- Adversary’s gain increased by total weight decrease in ADV

Fictitious “treasure packet”

- Not pending for the algorithm
- Tied to a slot \( \tau \) in ADV, no release time or deadline, never changes in future
- Deposit of profit to be collected by the adversary
- Weight bounded by \( \text{minwt}(\tau) \)
- Slot-monotonicity: \( \text{minwt}(\tau) \) never decrease

Pavel Veselý
Online Packet Scheduling
Adversary Schedule ADV

- Consists of already-released packets from OPT in future slots
- Packet in ADV can be replaced by \(\{\) another lighter packet, fictitious “treasure packets”\(\})
- Adversary’s gain increased by total weight decrease in ADV

Fictitious “treasure packet”

- Not pending for the algorithm
- Tied to a slot \(\tau\) in ADV, no release time or deadline, never changes in future
- Deposit of profit to be collected by the adversary
- **Weight bounded by** \(\text{minwt}(\tau)\)
- Slot-monotonicity: \(\text{minwt}(\tau)\) never decrease

Invariant (A)

ADV consists of two types of packets: \(\{\) (real) packets in plan \(P\), all other packets are treasures \(\})\)
Potential Function

Relative advantage of the algorithm over the adversary:

\[ P_{ADV} = \text{packets in the plan that the adversary will not schedule} \]

\[ \text{Set} \quad F \quad \text{Pending packets forced out of the plan} \]

\[ \text{Can be used as replacement packets in a leap step} \]

"Backup plan"

\[ R = P_{ADV} \cup R \]

**Invariant**

Backup plan \( R \) is feasible

\[ R \text{ feasible} = \text{packets in } R \text{ can be scheduled in future slots } t, t+1, ... \]

Potential \( \Psi := 1 \phi_w(R) \)
Potential Function

Relative advantage of the algorithm over the adversary:

- $\mathcal{P} \setminus \text{ADV} = \text{packets in the plan that the adversary will not schedule}$
Potential Function

Relative advantage of the algorithm over the adversary:

- $\mathcal{P} \setminus \text{ADV} = \text{packets in the plan that the adversary will not schedule}$
- Set $\mathcal{F}$
  - Pending packets forced out of the plan
Potential Function

Relative advantage of the algorithm over the adversary:

- $\mathcal{P} \setminus \text{ADV} =$ packets in the plan that the adversary will not schedule
- Set $\mathcal{F}$
  - Pending packets forced out of the plan
  - Can be used as replacement packets in a leap step

$R = \mathcal{P} \setminus \text{ADV} \cup \mathcal{F}$

Invariant
$R$ is feasible
$R$ feasible = packets in $R$ can be scheduled in future slots $t, t+1, \ldots$
Potential Function

Relative advantage of the algorithm over the adversary:

- \( \mathcal{P} \setminus \text{ADV} = \text{packets in the plan that the adversary will not schedule} \)
- Set \( \mathcal{F} \)
  - Pending packets forced out of the plan
  - Can be used as replacement packets in a leap step
- “Backup plan” \( R = \mathcal{P} \setminus \text{ADV} \cup R \)
Relative advantage of the algorithm over the adversary:

- $\mathcal{P} \setminus \text{ADV} = \text{packets in the plan that the adversary will not schedule}$
- Set $\mathcal{F}$
  - Pending packets forced out of the plan
  - Can be used as replacement packets in a leap step
- “Backup plan” $R = \mathcal{P} \setminus \text{ADV} \cup R$

**Invariant**

Backup plan $R$ is feasible

$R$ feasible = packets in $R$ can be scheduled in future slots $t, t + 1, \ldots$
Potential Function

Relative advantage of the algorithm over the adversary:

- $\mathcal{P} \setminus \text{ADV} =$ packets in the plan that the adversary will not schedule
- Set $\mathcal{F}$
  - Pending packets forced out of the plan
  - Can be used as replacement packets in a leap step
- “Backup plan” $R = \mathcal{P} \setminus \text{ADV} \cup \mathcal{F}$

Invariant

Backup plan $R$ is feasible

$R$ feasible = packets in $R$ can be scheduled in future slots $t, t + 1, \ldots$

Potential

$$\Psi := \frac{1}{\phi} \cdot w(R)$$
Packet Types in the Analysis

- \( F \): not in plan \( P \)
- \( P \setminus \text{ADV} \)
- \( P \cap \text{ADV} \)
- fictitious
- pending for PlanM
- “backup plan” \( R \) (potential)
Overview of the Analysis

To prove
- Packet arrival: $\Delta \Psi \geq 0$
Overview of the Analysis

To prove

- Packet arrival: $\Delta \Psi \geq 0$
- Scheduling step $t$
  - $j = \text{ADV}[t]$ scheduled by the adversary (possibly $j \neq \text{OPT}[t]$)
  - $p = \text{ALG}[t]$ scheduled by the algorithm

Proof of $\phi$-competitiveness

Potential equal to 0 at the beginning and at the end

$\sum_{t} \text{advgain}[t] \leq \sum_{t} \left[ \phi \cdot \left( w_{t}(\text{ALG}[t]) - \Delta \text{Weights} \right) \right] \leq \phi \cdot w_{0}(\text{ALG})$
Overview of the Analysis

To prove

- Packet arrival: $\Delta \Psi \geq 0$
- Scheduling step $t$
  - $j = \text{ADV}[t]$ scheduled by the adversary (possibly $j \neq \text{OPT}[t]$)
  - $p = \text{ALG}[t]$ scheduled by the algorithm
  - Adversary gain $\text{advgain}^t = w_j^t + \text{credit for replacing packets}$
Overview of the Analysis

To prove

- Packet arrival: $\Delta \Psi \geq 0$
- Scheduling step $t$
  - $j = \text{ADV}[t]$ scheduled by the adversary (possibly $j \neq \text{OPT}[t]$)
  - $p = \text{ALG}[t]$ scheduled by the algorithm
  - Adversary gain $\text{advgain}^t = w_j^t + \text{credit for replacing packets}$
  - $\Delta^t\text{Weights} = \text{amount by which the weights are increased in step } t$

\[
\text{advgain}^t \leq \phi \cdot (w_p^t - \Delta^t\text{Weights}) + \Delta \Psi
\]
Overview of the Analysis

To prove

- Packet arrival: $\Delta \Psi \geq 0$
- Scheduling step $t$
  - $j = \text{ADV}[t]$ scheduled by the adversary (possibly $j \neq \text{OPT}[t]$)
  - $p = \text{ALG}[t]$ scheduled by the algorithm
  - Adversary gain $\text{advgain}^t = w_j^t + \text{credit for replacing packets}$
  - $\Delta^t \text{Weights} = \text{amount by which the weights are increased in step } t$

\[
\text{advgain}^t \leq \phi \cdot (w_p^t - \Delta^t \text{Weights}) + \Delta \Psi
\]

Proof of $\phi$-competitiveness

- Potential equal to 0 at the beginning and at the end
Overview of the Analysis

To prove

- Packet arrival: \( \Delta \Psi \geq 0 \)
- Scheduling step \( t \)
  - \( j = \text{ADV}[t] \) scheduled by the adversary (possibly \( j \neq \text{OPT}[t] \))
  - \( p = \text{ALG}[t] \) scheduled by the algorithm
  - Adversary gain \( \text{advgain}^t = w_j^t \) + credit for replacing packets
  - \( \Delta^t \text{Weights} = \text{amount by which the weights are increased in step } t \)

\[
\text{advgain}^t \leq \phi \cdot (w_p^t - \Delta^t \text{Weights}) + \Delta \Psi
\]

Proof of \( \phi \)-competitiveness

- Potential equal to 0 at the beginning and at the end

\[
\omega^0(\text{OPT}) = \sum_t \text{advgain}^t
\]
Overview of the Analysis

To prove

- Packet arrival: $\Delta \Psi \geq 0$
- Scheduling step $t$
  - $j = \text{ADV}[t]$ scheduled by the adversary (possibly $j \neq \text{OPT}[t]$)
  - $p = \text{ALG}[t]$ scheduled by the algorithm
  - Adversary gain $\text{advgain}^t = w_j^t + \text{credit for replacing packets}$
  - $\Delta^t\text{Weights} = \text{amount by which the weights are increased in step } t$
    \[
    \text{advgain}^t \leq \phi \cdot (w_p^t - \Delta^t\text{Weights}) + \Delta \Psi
    \]

Proof of $\phi$-competitiveness

- Potential equal to 0 at the beginning and at the end
  \[
  w^0(\text{OPT}) = \sum_t \text{advgain}^t \leq \sum_t \left[ \phi \cdot (w^t(\text{ALG}[t]) - \Delta^t\text{Weights}) \right]
  \]
To prove

- Packet arrival: $\Delta \Psi \geq 0$
- Scheduling step $t$
  - $j = \text{ADV}[t]$ scheduled by the adversary (possibly $j \neq \text{OPT}[t]$)
  - $p = \text{ALG}[t]$ scheduled by the algorithm
  - Adversary gain $\text{advgain}^t = w_j^t + \text{credit for replacing packets}$
  - $\Delta^t \text{Weights} = \text{amount by which the weights are increased in step } t$

\[
\text{advgain}^t \leq \phi \cdot (w_p^t - \Delta^t \text{Weights}) + \Delta \Psi
\]

Proof of $\phi$-competitiveness

- Potential equal to 0 at the beginning and at the end

\[
\text{w}^0(\text{OPT}) = \sum_t \text{advgain}^t \leq \sum_t \left[ \phi \cdot (w^t(\text{ALG}[t]) - \Delta^t \text{Weights}) \right] \leq \phi \cdot \text{w}^0(\text{ALG})
\]
Conclusions
Summary

\( \phi \approx 1.618 \)-competitive deterministic algorithm

- Schedule packet \( p \in \mathcal{P} \) maximizing \( \phi \cdot w_p + w(Q_p) \)
  - \( Q_p \) is the plan after \( p \) is scheduled and time is incremented \( (p \notin Q_p) \)
$\phi \approx 1.618$-competitive deterministic algorithm

- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
- Maintain slot-monotonicity of minwt
  - Done by increasing weights and decreasing deadlines of certain packets
Summary

$\phi \approx 1.618$-competitive deterministic algorithm

- Schedule packet $p \in P$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented $(p \notin Q_p)$
- Maintain slot-monotonicity of $\text{minwt}$
  - Done by increasing weights and decreasing deadlines of certain packets

Analysis

- Potential function
  - Advantage of the algorithm over the adversary in future steps
  - Invariant ensures that this advantage is feasible
Summary

φ ≈ 1.618-competitive deterministic algorithm
- Schedule packet $p \in \mathcal{P}$ maximizing $\phi \cdot w_p + w(Q_p)$
  - $Q_p$ is the plan after $p$ is scheduled and time is incremented ($p \notin Q_p$)
- Maintain slot-monotonicity of minwt
  - Done by increasing weights and decreasing deadlines of certain packets

Analysis

- Potential function
  - Advantage of the algorithm over the adversary in future steps
  - Invariant ensures that this advantage is feasible
- Modifications of adversary schedule to maintain certain invariants
Further Research Directions

$m \geq 1$ packets are sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- The best algorithm has ratio $\frac{1}{1-\left(\frac{m}{m+1}\right)^m} \rightarrow \frac{e}{e-1} \approx 1.58$ [Chin et al. '04]
- Can our algorithm be modified to give a better ratio for $m > 1$?
Further Research Directions

$m \geq 1$ packets are sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- The best algorithm has ratio $\frac{1}{1 - \left(\frac{m}{m+1}\right)^m} \rightarrow \frac{e}{e-1} \approx 1.58$ [Chin et al. '04]
- Can our algorithm be modified to give a better ratio for $m > 1$?

Randomized algorithms

- Improve randomized algorithms using plans
- Gap between 1.25 [Chin & Fung '04] and $\frac{e}{e-1} \approx 1.58$ [Chin et al. '04]
Further Research Directions

$m \geq 1$ packets are sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- The best algorithm has ratio $\frac{1}{1-(\frac{m}{m+1})^m} \rightarrow \frac{e}{e-1} \approx 1.58$ [Chin et al. '04]
- Can our algorithm be modified to give a better ratio for $m > 1$?

Randomized algorithms

- Improve randomized algorithms using plans
- Gap between 1.25 [Chin & Fung '04] and $\frac{e}{e-1} \approx 1.58$ [Chin et al. '04]

Memoryless algorithms

- Is there a lower bound $> \phi$ for memoryless algorithms?
- What is the ratio of $\text{PLAN}(\alpha)$? (Schedule $p \in \mathcal{P}$ max. $\alpha \cdot w_p + w(Q_p)$)
Further Research Directions

$m \geq 1$ packets are sent in each step

- Our algorithm is $\phi \approx 1.618$-competitive for any $m \geq 1$
- The best algorithm has ratio $\frac{1}{1-(\frac{m}{m+1})^m} \rightarrow \frac{e}{e-1} \approx 1.58$ [Chin et al. '04]
- Can our algorithm be modified to give a better ratio for $m > 1$?

Randomized algorithms

- Improve randomized algorithms using plans
- Gap between $1.25$ [Chin & Fung '04] and $\frac{e}{e-1} \approx 1.58$ [Chin et al. '04]

Memoryless algorithms

- Is there a lower bound $> \phi$ for memoryless algorithms?
- What is the ratio of $\text{PLAN}(\alpha)$? (Schedule $p \in \mathcal{P}$ max. $\alpha \cdot w_p + w(Q_p)$)

Thank you!