On Packet Scheduling with Adversarial Jamming and Speedup

Martin Böhm[1] Łukasz Jeż[2] Jiří Sgall[1] **Pavel Veselý**[1]

[1]Charles University, Prague, Czech Republic

[2]University of Wrocław, Poland

Aussios, April 7, 2018

Goal of this talk

Simple online scheduling model with an open problem.

Goal of this talk

Simple online scheduling model with an open problem.

Outline

- Model
- Algorithm
- Local analysis
- Non-local analysis
- Lower bounds

Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets

Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets

Easy

Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets
- Adversarial errors, immediately known, retransmission possible

Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets
- Adversarial errors, immediately known, retransmission possible

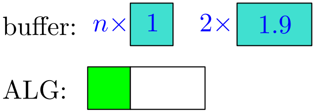
buffer: $n\times$ 1 $2\times$ 1.9

ALG:

Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

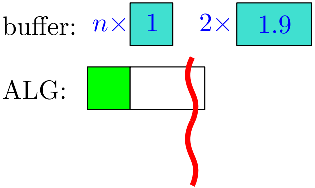
- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets
- Adversarial errors, immediately known, retransmission possible



Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

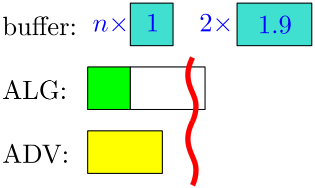
- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets
- Adversarial errors, immediately known, retransmission possible



Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

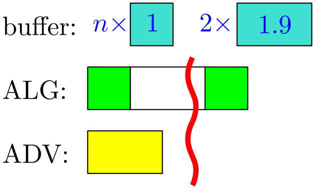
- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets
- Adversarial errors, immediately known, retransmission possible



Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

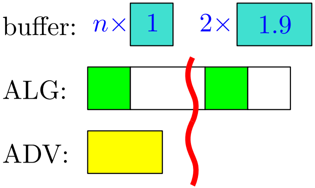
- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets
- Adversarial errors, immediately known, retransmission possible



Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

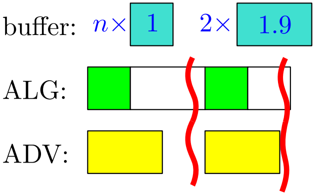
- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets
- Adversarial errors, immediately known, retransmission possible



Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

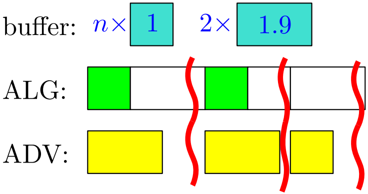
- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets
- Adversarial errors, immediately known, retransmission possible



Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

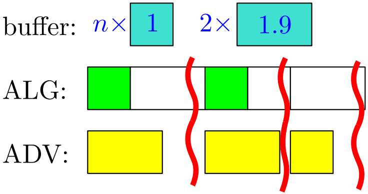
- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets
- Adversarial errors, immediately known, retransmission possible



Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets
- Adversarial errors, immediately known, retransmission possible



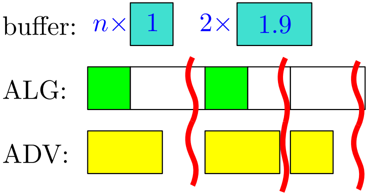
Goal

- R-competitive algorithms, i.e., $\mathrm{OPT} \leq R \cdot \mathrm{ALG} + C$
- k and ℓ_k are constants, allowed in C

Model

[Anta, Georgiou, Kowalski, Widmer, Zavou '13], [Jurdzinski, Kowalski, Loryś '14]

- Packets of sizes $\ell_1 < \cdots < \ell_k$, released over time, no deadlines
- Single channel (machine), no preemption
- Objective: Total size of completed packets
- Adversarial errors, immediately known, retransmission possible



Goal

- R-competitive algorithms, i.e., $\text{OPT} \leq R \cdot \text{ALG} + C$
- k and ℓ_k are constants, allowed in C

We focus on deterministic algorithms only.

Previous work [Anta et al.], [Jurdzinski et al.]

General instances

- 3-competitive algorithm

Previous work [Anta et al.], [Jurdzinski et al.]

General instances

- 3-competitive algorithm
- a matching lower bound on the comp. ratio

Previous work [Anta et al.], [Jurdzinski et al.]

General instances

- 3-competitive algorithm
- a matching lower bound on the comp. ratio

Also more work on restricted instances, with multiple channels, stochastic models etc.

Previous work [Anta et al.], [Jurdzinski et al.]

General instances

- 3-competitive algorithm
- a matching lower bound on the comp. ratio

Also more work on restricted instances, with multiple channels, stochastic models etc.

Problem solved . . .

Previous work [Anta et al.], [Jurdzinski et al.]

General instances

- 3-competitive algorithm – but needs to know sizes
- a matching lower bound on the comp. ratio – uses only 2 sizes

Also more work on restricted instances, with multiple channels, stochastic models etc.

Problem solved . . .

Previous work [Anta et al.], [Jurdzinski et al.]

General instances

- 3-competitive algorithm – but needs to know sizes
- a matching lower bound on the comp. ratio – uses only 2 sizes

Also more work on restricted instances, with multiple channels, stochastic models etc.

Problem solved . . .

Speedup for 1-competitiveness on general instances

- a lower bound of 2 [Anta et al. '15]
- but no good algorithm

Speedup s = ALG needs time only ℓ/s to send a packet of size ℓ

Our main results – A universal & (quite) simple algorithm

Our main results – A universal & (quite) simple algorithm

Simple generic local analysis

- 3-competitive

Our main results – A universal & (quite) simple algorithm

Simple generic local analysis

- 3-competitive
- 1-competitive with speedup 6

Our main results – A universal & (quite) simple algorithm

Simple generic local analysis

- 3-competitive
- 1-competitive with speedup 6
- tradeoffs, better results for restricted instances

Our main results – A universal & (quite) simple algorithm

Simple generic local analysis

- 3-competitive – matching a lower bound
- 1-competitive with speedup 6 – not tight
- tradeoffs, better results for restricted instances

Our main results – A universal & (quite) simple algorithm

Simple generic local analysis

- 3-competitive – matching a lower bound
- 1-competitive with speedup 6 – not tight
- tradeoffs, better results for restricted instances

Worked hard ...

Our main results – A universal & (quite) simple algorithm

Simple generic local analysis

- 3-competitive – matching a lower bound
- 1-competitive with speedup 6 – not tight
- tradeoffs, better results for restricted instances

Worked hard . . . took a break . . .

Our main results – A universal & (quite) simple algorithm

Simple generic local analysis

- 3-competitive – matching a lower bound
- 1-competitive with speedup 6 – not tight
- tradeoffs, better results for restricted instances

Worked hard . . . took a break . . . worked hard . . .

Our main results – A universal & (quite) simple algorithm

Simple generic local analysis

- 3-competitive – matching a lower bound
- 1-competitive with speedup 6 – not tight
- tradeoffs, better results for restricted instances

Worked hard . . . took a break . . . worked hard . . . more sophisticated analysis!

Our main results – A universal & (quite) simple algorithm

Simple generic local analysis

- 3-competitive – matching a lower bound
- 1-competitive with speedup 6 – not tight
- tradeoffs, better results for restricted instances

Worked hard . . . took a break . . . worked hard . . . more sophisticated analysis!

Upper bound

- 1-competitive with speedup 4 (tight)

Our main results – A universal & (quite) simple algorithm

Simple generic local analysis

- 3-competitive – matching a lower bound
- 1-competitive with speedup 6 – not tight
- tradeoffs, better results for restricted instances

Worked hard ... took a break ... worked hard ... more sophisticated analysis!

Upper bound

- 1-competitive with speedup 4 (tight)

Lower bounds

No 1-competitive deterministic algorithm with speedup $s < \phi + 1 \approx 2.618$

Algorithm – Description

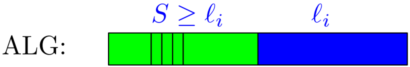
Algorithm – Description

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Algorithm – Description

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.



Algorithm – Description

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

ALG:

Phase ends when:

- fault occurs,
- no packet is pending, or
- only packets of size $> S$ are pending.

buffer: $1\times\frac{1}{2}$ 4×1 1×4 8×5

ALG:

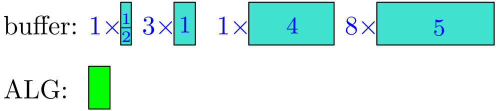
Algorithm – Description

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Phase ends when:

- fault occurs,
- no packet is pending, or
- only packets of size $> S$ are pending.



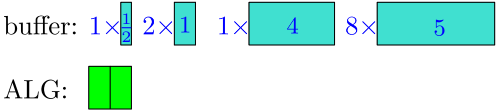
Algorithm – Description

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Phase ends when:

- fault occurs,
- no packet is pending, or
- only packets of size $> S$ are pending.



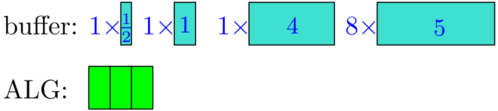
Algorithm – Description

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Phase ends when:

- fault occurs,
- no packet is pending, or
- only packets of size $> S$ are pending.



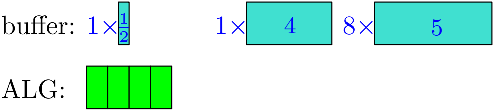
Algorithm – Description

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Phase ends when:

- fault occurs,
- no packet is pending, or
- only packets of size $> S$ are pending.



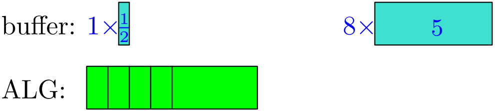
Algorithm – Description

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Phase ends when:

- fault occurs,
- no packet is pending, or
- only packets of size $> S$ are pending.



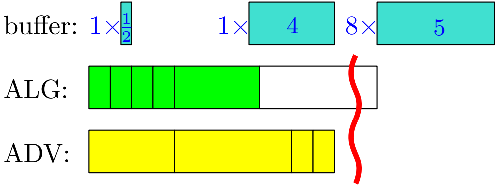
Algorithm – Description

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Phase ends when:

- fault occurs,
- no packet is pending, or
- only packets of size $> S$ are pending.



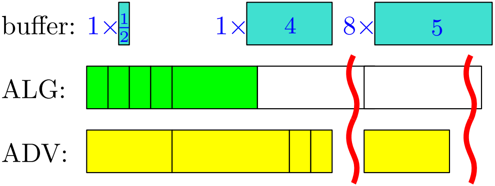
Algorithm – Description

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Phase ends when:

- fault occurs,
- no packet is pending, or
- only packets of size $> S$ are pending.



Algorithm – Properties

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Nice properties of the new algorithm

- no unnecessary idle time
- the same algorithm for all speeds
- no need to know the speed and packet sizes in advance

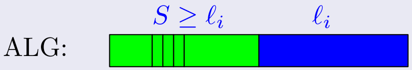
Algorithm – Observations

Start phase: Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step: Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Observation: In each phase

- If the 1^{st} packet completes: size of completed packets $\geq \frac{1}{2}$ of length of the phase.



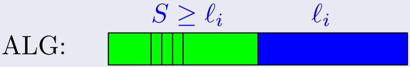
Algorithm – Observations

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Observation: In each phase

- If the 1^{st} packet completes:
  size of completed packets $\geq \frac{1}{2}$ of length of the phase.



- All jobs are completed if no fault or arrival occurs.

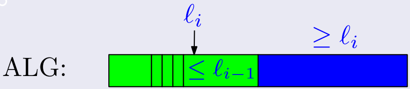
Algorithm – Observations

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Observation: In each phase

- If the 1^{st} packet completes:
  size of completed packets $\geq \frac{1}{2}$ of length of the phase.
- All jobs are completed if no fault or arrival occurs.
- If a packet of size ℓ_i is pending during the whole phase, then the size of completed smaller packets is less than $2 \cdot \ell_i$.



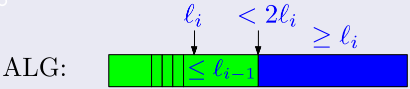
Algorithm – Observations

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run packet of the largest size ℓ_j such that $\ell_j \leq S$,
S: total size of packets completed in this phase.

Observation: In each phase

- If the 1^{st} packet completes:
  size of completed packets $\geq \frac{1}{2}$ of length of the phase.
- All jobs are completed if no fault or arrival occurs.
- If a packet of size ℓ_i is pending during the whole phase, then the size of completed smaller packets is less than $2 \cdot \ell_i$.



Local analysis – The method

For each size ℓ_i: critical time C_i.

Local analysis – The method

For each size ℓ_i: critical time C_i.
Critical times $C_k \leq C_{k-1} \leq \cdots \leq C_1$ satisfy:

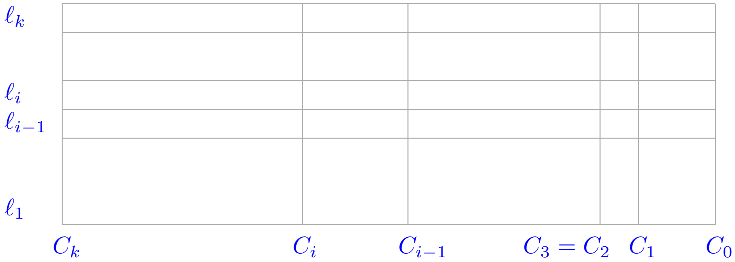
- almost no packets of size ℓ_i pending just before C_i,
- a packet of size ℓ_i is always pending in $(C_i, C_{i-1}]$.

Local analysis – The method

For each size ℓ_i: critical time C_i.

Critical times $C_k \leq C_{k-1} \leq \cdots \leq C_1$ satisfy:

- almost no packets of size ℓ_i pending just before C_i,
- a packet of size ℓ_i is always pending in $(C_i, C_{i-1}]$.

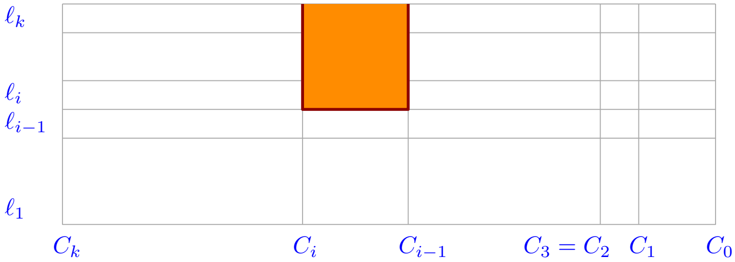


Local analysis – The method

For each size ℓ_i: critical time C_i.

Critical times $C_k \leq C_{k-1} \leq \cdots \leq C_1$ satisfy:

- almost no packets of size ℓ_i pending just before C_i,
- a packet of size ℓ_i is always pending in $(C_i, C_{i-1}]$.



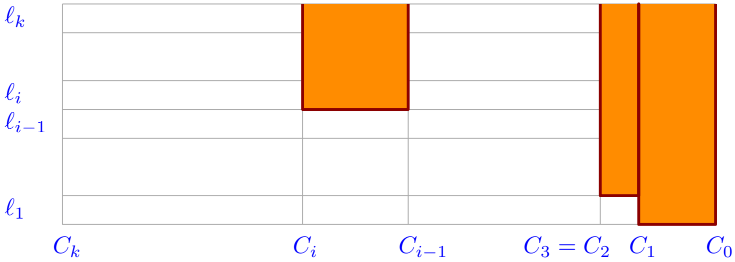
Proof method (for 1-competitiveness)

For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\geq \ell_i$) completed by ALG is at least that of ADV

Local analysis – The method

For each size ℓ_i: critical time C_i.
Critical times $C_k \leq C_{k-1} \leq \cdots \leq C_1$ satisfy:

- almost no packets of size ℓ_i pending just before C_i,
- a packet of size ℓ_i is always pending in $(C_i, C_{i-1}]$.



Proof method (for 1-competitiveness)

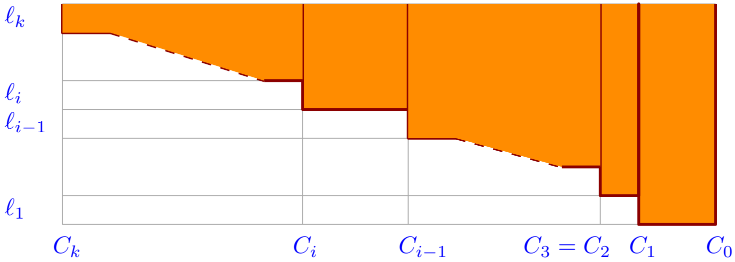
For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\geq \ell_i$) completed by ALG is at least that of ADV

Local analysis – The method

For each size ℓ_i: critical time C_i.

Critical times $C_k \le C_{k-1} \le \cdots \le C_1$ satisfy:

- almost no packets of size ℓ_i pending just before C_i,
- a packet of size ℓ_i is always pending in $(C_i, C_{i-1}]$.



Proof method (for 1-competitiveness)

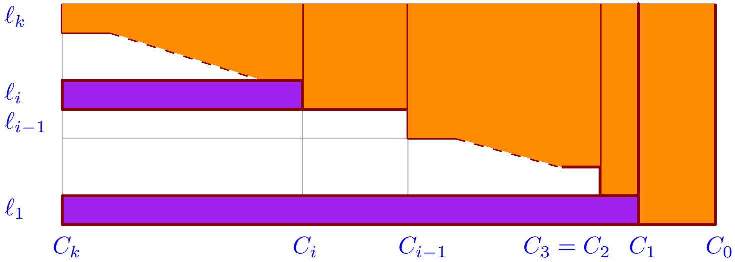
For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\ge \ell_i$) completed by ALG is at least that of ADV

Local analysis – The method

For each size ℓ_i: critical time C_i.

Critical times $C_k \leq C_{k-1} \leq \cdots \leq C_1$ satisfy:

- almost no packets of size ℓ_i pending just before C_i,
- a packet of size ℓ_i is always pending in $(C_i, C_{i-1}]$.



Proof method (for 1-competitiveness)

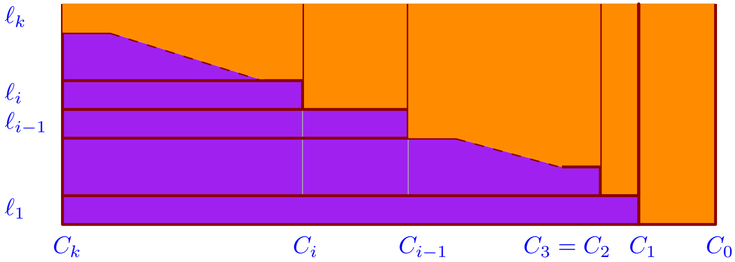
For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\geq \ell_i$) completed by ALG is at least that of ADV

Local analysis – The method

For each size ℓ_i: critical time C_i.

Critical times $C_k \leq C_{k-1} \leq \cdots \leq C_1$ satisfy:

- almost no packets of size ℓ_i pending just before C_i,
- a packet of size ℓ_i is always pending in $(C_i, C_{i-1}]$.



Proof method (for 1-competitiveness)

For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\geq \ell_i$) completed by ALG is at least that of ADV

Local analysis – General instances, speedup 6

Proof method (for 1-competitiveness)

For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\geq \ell_i$) completed by ALG is at least that of ADV

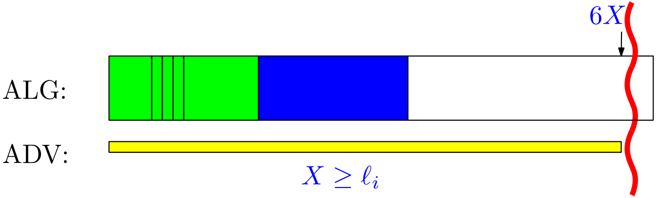
- a packet of size ℓ_i is always pending during the phase.

Local analysis – General instances, speedup 6

Proof method (for 1-competitiveness)

For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\geq \ell_i$) completed by ALG is at least that of ADV

- a packet of size ℓ_i is always pending during the phase.

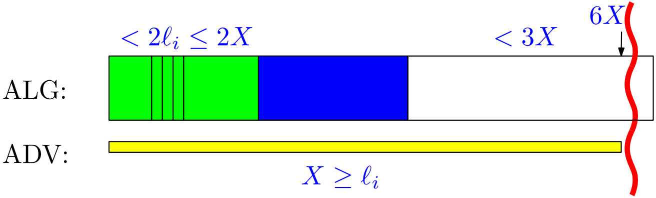


Local analysis – General instances, speedup 6

Proof method (for 1-competitiveness)

For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\geq \ell_i$) completed by ALG is at least that of ADV

- a packet of size ℓ_i is always pending during the phase.

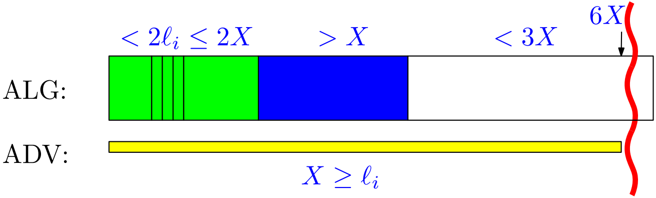


Local analysis – General instances, speedup 6

Proof method (for 1-competitiveness)

For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\geq \ell_i$) completed by ALG is at least that of ADV

- a packet of size ℓ_i is always pending during the phase.

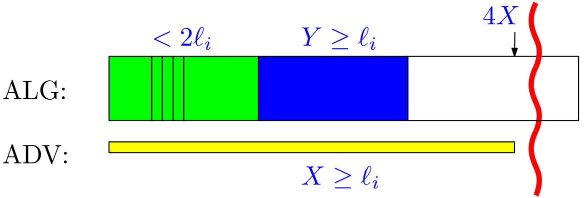


General instances, speedup 4

Proof method (for 1-competitiveness)

For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\geq \ell_i$) completed by ALG is at least that of ADV

- a packet of size ℓ_i is always pending during the phase.



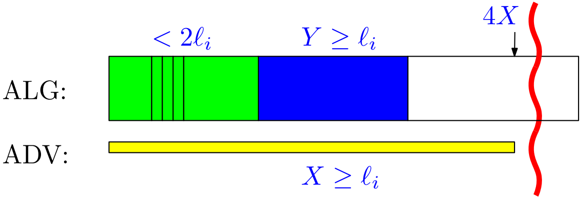
- Y completed as $Y < 2\ell_i$
  - Does not hold for speedup below 4

General instances, speedup 4

Proof method (for 1-competitiveness)

For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\geq \ell_i$) completed by ALG is at least that of ADV

- a packet of size ℓ_i is always pending during the phase.



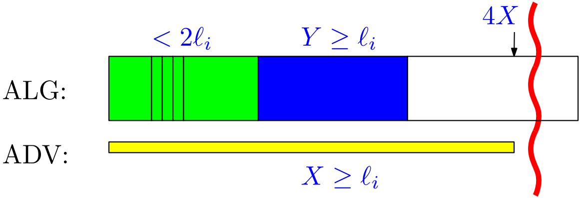
- Y completed as $Y < 2\ell_i$
  - Does not hold for speedup below 4
- It may happen that $X = \ell_{i+1} > \ell_i = Y$ ...

General instances, speedup 4

Proof method (for 1-competitiveness)

For each phase within $(C_i, C_{i-1}]$, show that the total size of long packets ($\geq \ell_i$) completed by ALG is at least that of ADV

- a packet of size ℓ_i is always pending during the phase.



- Y completed as $Y < 2\ell_i$
  - Does not hold for speedup below 4
- It may happen that $X = \ell_{i+1} > \ell_i = Y$ ...
  ... but only if no packet of size ℓ_{i+1} is pending.

General instances, speedup 4

Redefine critical times: C'_i satisfy:

- almost no packets of size ℓ_i pending just before C'_i,
- a packet of size ℓ_i is always pending all the time after C'_i.

We may have e.g. $C'_4 > C'_1 > C'_2 = C'_5 > C'_3$.

General instances, speedup 4

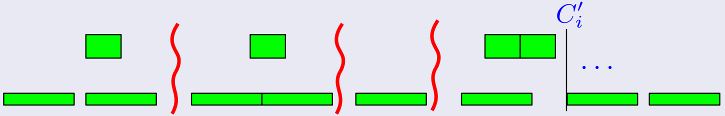
Redefine critical times: C'_i satisfy:

- almost no packets of size ℓ_i pending just before C'_i,
- a packet of size ℓ_i is always pending all the time after C'_i.

We may have e.g. $C'_4 > C'_1 > C'_2 = C'_5 > C'_3$.

Focus on packets of size ℓ_i

1. Till C'_i: 1-to-1 charges



General instances, speedup 4

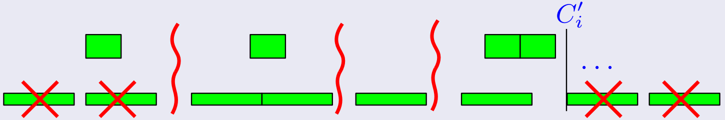
Redefine critical times: C_i' satisfy:

- almost no packets of size ℓ_i pending just before C_i',
- a packet of size ℓ_i is always pending all the time after C_i'.

We may have e.g. $C_4' > C_1' > C_2' = C_5' > C_3'$.

Focus on packets of size ℓ_i

1. Till C_i': 1-to-1 charges



General instances, speedup 4

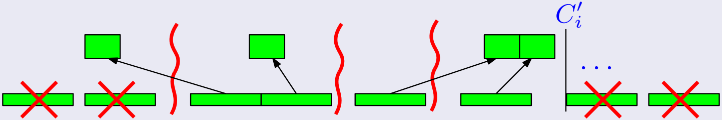
Redefine critical times: C'_i satisfy:

- almost no packets of size ℓ_i pending just before C'_i,
- a packet of size ℓ_i is always pending all the time after C'_i.

We may have e.g. $C'_4 > C'_1 > C'_2 = C'_5 > C'_3$.

Focus on packets of size ℓ_i

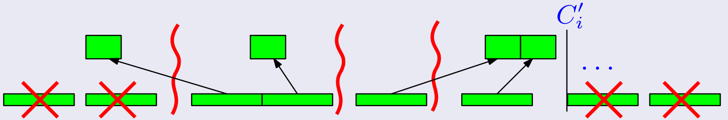
1. Till C'_i: 1-to-1 charges



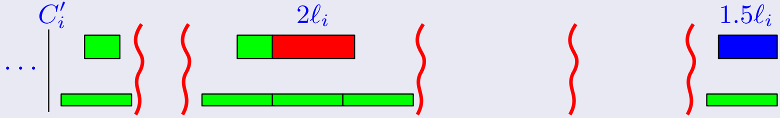
General instances, speedup 4

Focus on packets of size ℓ_i

1. Till C_i': 1-to-1 charges



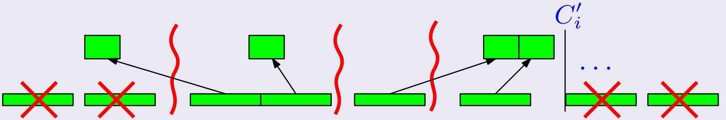
2. After C_i' : ℓ_i always pending



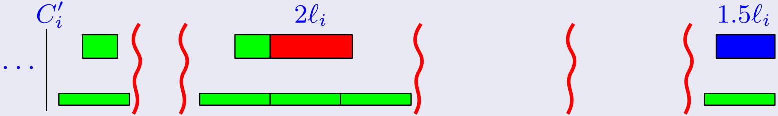
General instances, speedup 4

Focus on packets of size ℓ_i

1. Till C_i': 1-to-1 charges



2. After C_i' : ℓ_i always pending

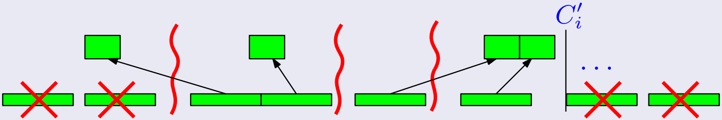


Main idea: assign ℓ_i's in ADV to $\geq \ell_i$'s in ALG

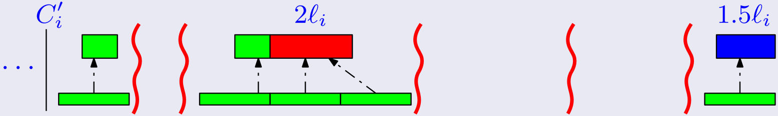
General instances, speedup 4

Focus on packets of size ℓ_i

1. Till C_i': 1-to-1 charges



2. After C_i' : ℓ_i always pending



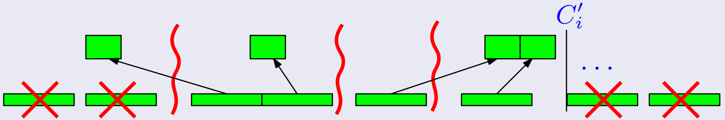
Main idea: assign ℓ_i's in ADV to $\geq \ell_i$'s in ALG

- if ADV has $a \times \ell_i$, ALG has $> (2a-2) \cdot \ell_i$ of packets $\geq \ell_i$
  - if $a = 1$ ALG has $\geq \ell_i$

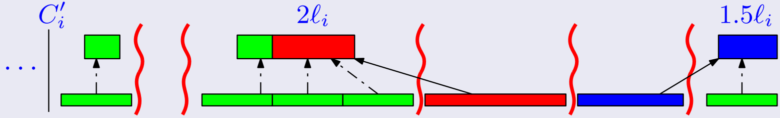
General instances, speedup 4

Focus on packets of size ℓ_i

1. Till C_i': 1-to-1 charges



2. After C_i' : ℓ_i always pending



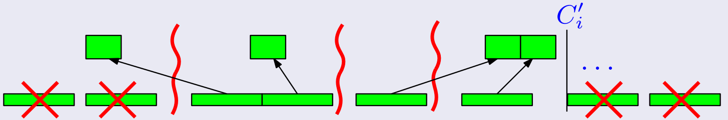
Main idea: assign ℓ_i's in ADV to $\geq \ell_i$'s in ALG

- if ADV has $a \times \ell_i$, ALG has $> (2a - 2) \cdot \ell_i$ of packets $\geq \ell_i$
  - if $a = 1$ ALG has $\geq \ell_i$

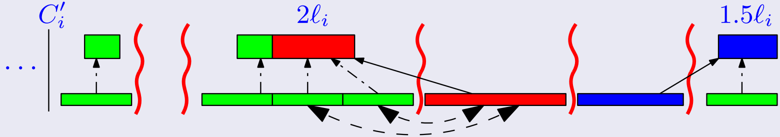
General instances, speedup 4

Focus on packets of size ℓ_i

1. Till C'_i: 1-to-1 charges



2. After C'_i : ℓ_i always pending



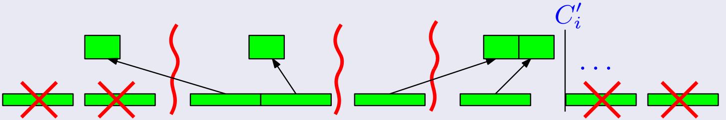
Main idea: assign ℓ_i's in ADV to $\geq \ell_i$'s in ALG

- if ADV has $a \times \ell_i$, ALG has $> (2a-2) \cdot \ell_i$ of packets $\geq \ell_i$
  - if $a = 1$ ALG has $\geq \ell_i$

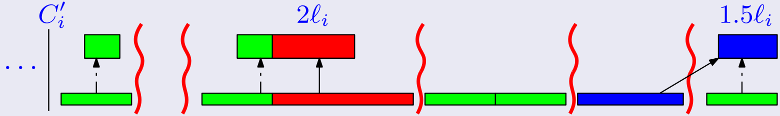
General instances, speedup 4

Focus on packets of size ℓ_i

1. Till C_i': 1-to-1 charges



2. After C_i' : ℓ_i always pending



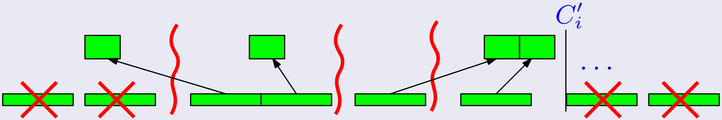
Main idea: assign ℓ_i's in ADV to $\geq \ell_i$'s in ALG

- if ADV has $a \times \ell_i$, ALG has $> (2a - 2) \cdot \ell_i$ of packets $\geq \ell_i$
    - if $a = 1$ ALG has $\geq \ell_i$

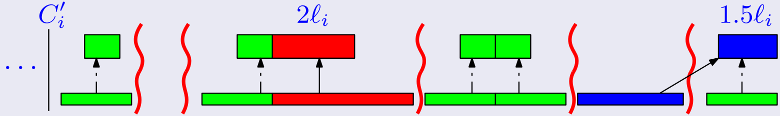
General instances, speedup 4

Focus on packets of size ℓ_i

1. Till C_i': 1-to-1 charges



2. After C_i' : ℓ_i always pending



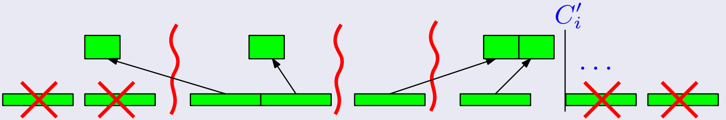
Main idea: assign ℓ_i's in ADV to $\geq \ell_i$'s in ALG

- if ADV has $a \times \ell_i$, ALG has $> (2a - 2) \cdot \ell_i$ of packets $\geq \ell_i$
  - if $a = 1$ ALG has $\geq \ell_i$

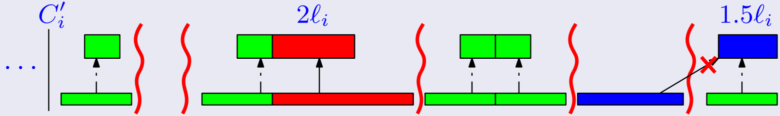
General instances, speedup 4

Focus on packets of size ℓ_i

1. Till C_i': 1-to-1 charges



2. After C_i' : ℓ_i always pending



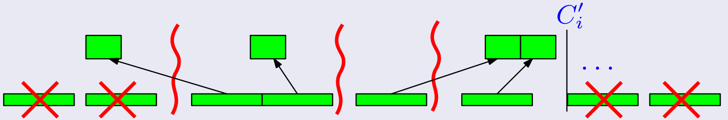
Main idea: assign ℓ_i's in ADV to $\geq \ell_i$'s in ALG

- if ADV has $a \times \ell_i$, ALG has $> (2a-2) \cdot \ell_i$ of packets $\geq \ell_i$
  - if $a = 1$ ALG has $\geq \ell_i$

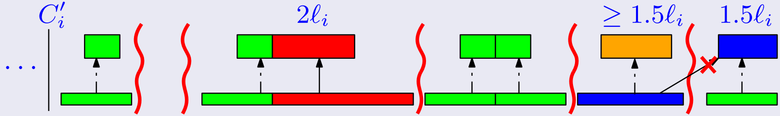
General instances, speedup 4

Focus on packets of size ℓ_i

1. Till C_i': 1-to-1 charges



2. After C_i' : ℓ_i always pending



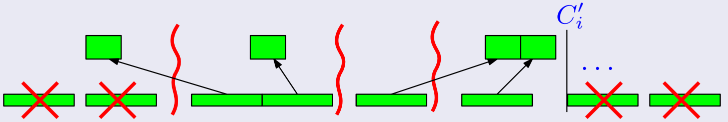
Main idea: assign ℓ_i's in ADV to $\geq \ell_i$'s in ALG

- if ADV has $a \times \ell_i$, ALG has $> (2a-2) \cdot \ell_i$ of packets $\geq \ell_i$
    - if $a = 1$ ALG has $\geq \ell_i$

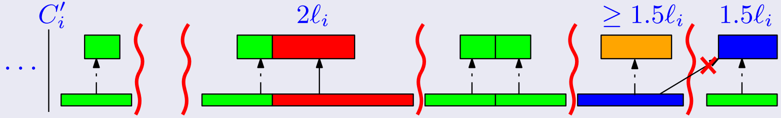
General instances, speedup 4

Focus on packets of size ℓ_i

1. Till C'_i: 1-to-1 charges



2. After C'_i : ℓ_i always pending



Main idea: assign ℓ_i's in ADV to $\geq \ell_i$'s in ALG

- if ADV has $a \times \ell_i$, ALG has $> (2a - 2) \cdot \ell_i$ of packets $\geq \ell_i$
  - if $a = 1$ ALG has $\geq \ell_i$

+ quite a lot of technical work (e.g., phases in which ALG completes no packet)

Lower bound – Speed below 2

No deterministic 1-competitive algorithm with speedup < 2

Input

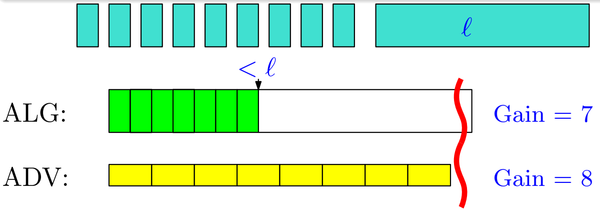
- packet sizes 1 and ℓ, all packets arrive at time 0
- $\#$ of small packets $\gg$ $\#$ of ℓ-packets

Lower bound – Speed below 2

No deterministic 1-competitive algorithm with speedup < 2

Input

- packet sizes 1 and ℓ, all packets arrive at time 0
- # of small packets $\gg$ # of ℓ-packets



Adversary strategy in each phase

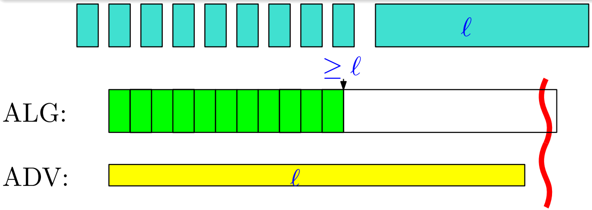
- If ALG starts ℓ soon, interrupt it and finish more small packets than ALG.

Lower bound – Speed below 2

No deterministic 1-competitive algorithm with speedup < 2

Input

- packet sizes 1 and ℓ, all packets arrive at time 0
- # of small packets $\gg$ # of ℓ-packets



Adversary strategy in each phase

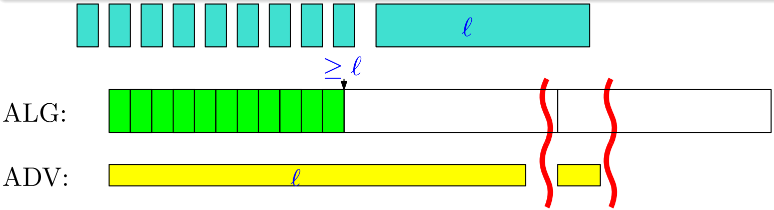
- If ALG starts ℓ soon, interrupt it and finish more small packets than ALG.
- Otherwise finish ℓ and make a fault.

Lower bound – Speed below 2

No deterministic 1-competitive algorithm with speedup < 2

Input

- packet sizes 1 and ℓ, all packets arrive at time 0
- # of small packets $\gg$ # of ℓ-packets



Adversary strategy in each phase

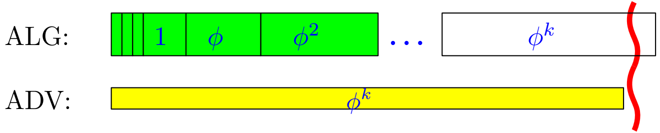
- If ALG starts ℓ soon, interrupt it and finish more small packets than ALG.
- Otherwise finish ℓ and make a fault.

Lower bound – Speed below $\phi + 1 \approx 2.618$

- packet sizes ε, 1, ϕ, ϕ^2, $\dots$, ϕ^k; all arrive at time 0,
- # of smaller packets $\gg$ # of bigger packets,

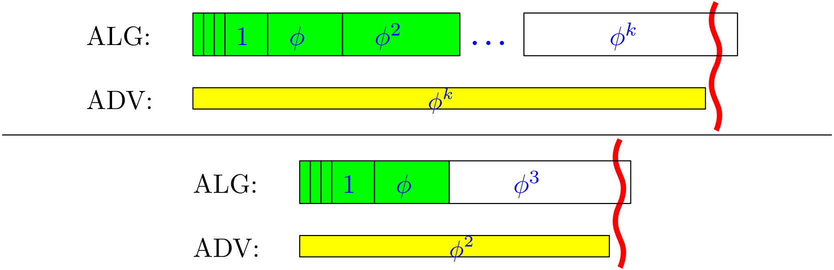
Lower bound – Speed below $\phi + 1 \approx 2.618$

- packet sizes ε, 1, ϕ, ϕ^2, . . . , ϕ^k; all arrive at time 0,
- # of smaller packets $\gg$ # of bigger packets,
- force ALG to schedule size $\phi - 1$ of tiny packets, then one packet of each size 1, ϕ, . . .



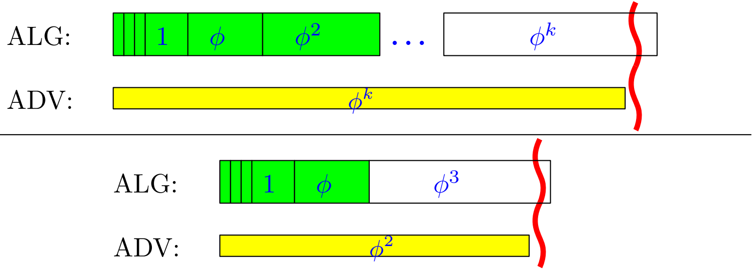
Lower bound – Speed below $\phi + 1 \approx 2.618$

- packet sizes ε, 1, ϕ, ϕ^2, ..., ϕ^k; all arrive at time 0,
- # of smaller packets $\gg$ # of bigger packets,
- force ALG to schedule size $\phi - 1$ of tiny packets, then one packet of each size 1, ϕ, ...



Lower bound – Speed below $\phi + 1 \approx 2.618$

- packet sizes ε, 1, ϕ, ϕ^2, $\dots$, ϕ^k; all arrive at time 0,
- # of smaller packets $\gg$ # of bigger packets,
- force ALG to schedule size $\phi - 1$ of tiny packets, then one packet of each size 1, ϕ, $\dots$



\+ two other similar cases
\+ when ADV completes all packets ϕ^i, then it completes packets $< \phi^i$ preventing ALG to finish a packet $\geq \phi^i$

Open Problems (for the way back home)

- 1-competitive algorithm with speedup $s < 4$

Open Problems (for the way back home)

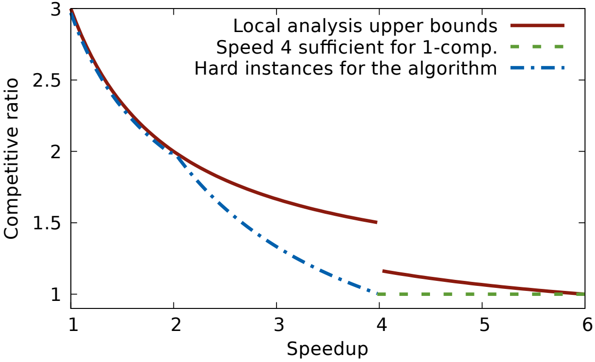
- 1-competitive algorithm with speedup $s < 4$
  - Or a lower bound better than $\phi + 1$
  - Do release times change the required speedup?

Open Problems (for the way back home)

- 1-competitive algorithm with speedup $s < 4$
  - Or a lower bound better than $\phi + 1$
  - Do release times change the required speedup?
- Randomization
- Weights (we have $w_p = \ell_p$)

Open Problems (for the way back home)

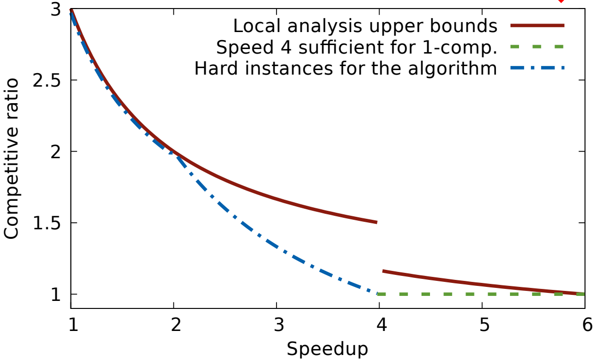
- 1-competitive algorithm with speedup $s < 4$
  - Or a lower bound better than $\phi + 1$
  - Do release times change the required speedup?
- Randomization
- Weights (we have $w_p = \ell_p$)
- Tradeoffs (e.g., speed vs. competitive ratio)



Open Problems (for the way back home)

- 1-competitive algorithm with speedup $s < 4$
  - Or a lower bound better than $\phi + 1$
  - Do release times change the required speedup?
- Randomization
- Weights (we have $w_p = \ell_p$)
- Tradeoffs (e.g., speed vs. competitive ratio)

Thank You!



Local analysis results for special cases

Divisible instances (ℓ_i divides ℓ_{i+1} for each i):

- Our algorithm is 1-competitive with speedup 2.5

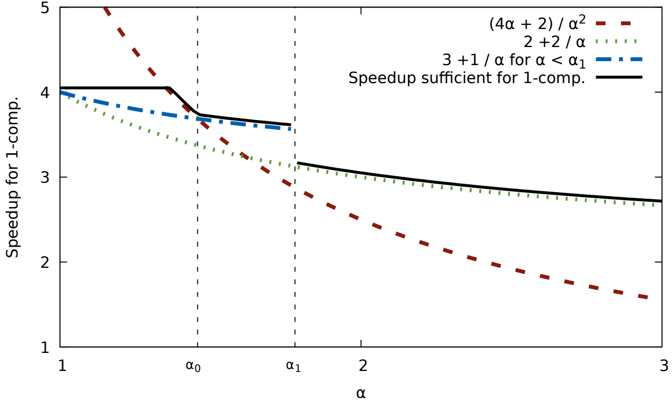
Local analysis results for special cases

Divisible instances (ℓ_i divides ℓ_{i+1} for each i):

- Our algorithm is 1-competitive with speedup 2.5

Well-separated instances:

- $\ell_{i+1} \geq \alpha\ell_i$ for some parameter $\alpha > 1$
- Our algorithm is 1-competitive with speedup S_α:



Algorithm for divisible instances

Start phase: Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step: Run ... largest ℓ_j such that $\ell_j \leq S$ and ℓ_j divides S,
S: total size of packets completed in this phase.

Algorithm for divisible instances

Start phase Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step Run ... largest ℓ_j such that $\ell_j \leq S$ and ℓ_j divides S,
S: total size of packets completed in this phase.

- Same properties as the algorithm for general instances
- Key observation: if a packet of size ℓ_i is pending during the whole phase, then the size of completed smaller packets is at most $1 \cdot \ell_i$.

Algorithm for divisible instances

Start phase: Run packet of the largest size ℓ_j such that $P^{<j} < \ell_j$,
$P^{<j}$: total size of pending packets smaller than ℓ_j.

Regular step: Run ... largest ℓ_j such that $\ell_j \leq S$ and ℓ_j divides S,
S: total size of packets completed in this phase.

- Same properties as the algorithm for general instances
- Key observation: if a packet of size ℓ_i is pending during the whole phase, then the size of completed smaller packets is at most $1 \cdot \ell_i$.

Results using local analysis

- 2-competitive (optimal),
- 1-competitive with speedup 2 (optimal),
- both algorithmic results also done by [Jurdzinski et al.]