Nicole Megow

Scheduling to meet deadlines: Online algorithms & feasibility tests

with S. Anand and Naveen Garg (IIT Delhi)

Interdisciplinary Workshop on Algorithmic Challenges in Real-Time Scheduling
Berlin, February 27-29, 2012
The problem

- set of jobs, with job $j$ defined by:
  - processing time $p_j \in \mathbb{N}$
  - release date $r_j \in \mathbb{N}$
  - deadline $d_j \in \mathbb{N}$
- $m$ identical parallel machines
- preemption & migration allowed
The problem

- set of jobs, with job $j$ defined by:
  - processing time $p_j \in \mathbb{N}$
  - release date $r_j \in \mathbb{N}$
  - deadline $d_j \in \mathbb{N}$

- $m$ identical parallel machines
- preemption & migration allowed

An algorithm is **optimal** if it finds a feasible solution if there is one.
The problem

- set of jobs, with job $j$ defined by:
  - processing time $p_j \in \mathbb{N}$
  - release date $r_j \in \mathbb{N}$
  - deadline $d_j \in \mathbb{N}$
- $m$ identical parallel machines
- preemption & migration allowed

An algorithm is \textbf{optimal} if it finds a feasible solution if there is one.

\textbf{Offline}: solvable in polynomial time as a maximum flow problem
The online problem

Jobs arrive **online over time** at their release date.
The online problem

Jobs arrive **online over time** at their release date.

- \( m = 1 \): several optimal algorithms (EDF, LLF, etc.)
The online problem

Jobs arrive online over time at their release date.

- $m = 1$: several optimal algorithms (EDF, LLF, etc.)
- $m \geq 2$: no optimal algorithm [Dertouzos & Mok ’89]
The online problem

Jobs arrive online over time at their release date.

- $m = 1$: several optimal algorithms (EDF, LLF, etc.)
- $m \geq 2$: no optimal algorithm [Dertouzos & Mok ’89]

**Resource augmentation**: extra speed or additional machines
The online problem

Jobs arrive online over time at their release date.

- $m = 1$: several optimal algorithms (EDF, LLF, etc.)
- $m \geq 2$: no optimal algorithm [Dertouzos & Mok ’89]

Resource augmentation: extra speed or additional machines

**Performance of an Algorithm A:**
Required speed $s \geq 1$ such that A is optimal on $m$ machines of speed $s$ for any instance that is feasible on $m$ speed-1 machines.
The online problem

Jobs arrive online over time at their release date.

- $m = 1$: several optimal algorithms (EDF, LLF, etc.)
- $m \geq 2$: no optimal algorithm [Dertouzos & Mok ’89]

Resource augmentation: extra speed or additional machines

Performance of an Algorithm A:
Required speed $s \geq 1$ such that A is optimal on $m$ machines of speed $s$ for any instance that is feasible on $m$ speed-1 machines.

Goal: Find algorithm with minimum speed requirement.
Known results

- **General lower bound**: $s \geq 6/5$  
  [Phillips, Stein, Torng, Wein '96]
Known results

- General lower bound: $s \geq 6/5$  
  [Phillips, Stein, Torng, Wein '96]

- Deadline ordered algorithms  
  (depend only on relative order of deadlines and not on values)
Known results

- **General lower bound**: $s \geq 6/5$  
  [Phillips, Stein, Torng, Wein '96]

- **Deadline ordered algorithms**  
  (depend only on relative order of deadlines and not on values)
  - **EDF**: $s = 2 - \frac{1}{m}$  
    [Phillips, Stein, Torng, Wein '96]
Known results

- **General lower bound**: \( s \geq 6/5 \)  
  [Phillips, Stein, Torng, Wein '96]

- **Deadline ordered algorithms**  
  (depend only on relative order of deadlines and not on values)
  - EDF: \( s = 2 - \frac{1}{m} \)  
    [Phillips, Stein, Torng, Wein '96]
  - Lam & To: \( s \leq 2 - \frac{2}{m+1} \)  
    [Lam, To '99]

- **Lower bound**: \( \alpha_m := \frac{1}{1 - (1 - \frac{1}{m})^m} \leq e - 1 \)  
  [Lam, To '99]
Known results

- **General lower bound:** $s \geq \frac{6}{5}$ [Phillips, Stein, Torng, Wein '96]

- **Deadline ordered algorithms**
  (depend only on relative order of deadlines and not on values)
  - EDF: $s = 2 - \frac{1}{m}$ [Phillips, Stein, Torng, Wein '96]
  - Lam & To: $s \leq 2 - \frac{2}{m+1}$ [Lam, To '99]
  - Lower bound: $\alpha_m := \frac{1}{1-(1-\frac{1}{m})^m} \leq \frac{e}{e-1}$ [Lam, To '99]
Known results

- **General lower bound**: $s \geq 6/5$  
  [Phillips, Stein, Torng, Wein '96]

- **Deadline ordered algorithms**  
  (depend only on relative order of deadlines and not on values)
  - EDF: $s = 2 - \frac{1}{m}$  
    [Phillips, Stein, Torng, Wein '96]
  - Lam & To: $s \leq 2 - \frac{2}{m+1}$  
    [Lam, To '99]
  - Lower bound: $\alpha_m := \frac{1}{1-(1-\frac{1}{m})^m} \leq \frac{e}{e-1}$  
    [Lam, To '99]

- **Non-deadline ordered algorithms**
  - Earliest Deadline until Zero Laxity (EDZL): $s \leq 2 - \frac{1}{m}$  
    [Cho, Li, Ahn, Lin ’02]
  - Least Laxity First (LLF): $s \leq 2 - \frac{1}{m}$  
    [Phillips, Stein, Torng, Wein '96]
Our results

- **New deadline ordered algorithm** with exact speed requirement

\[ \alpha_m \leq \frac{e}{e - 1} \approx 1.58 \]

which is optimal in this class of algorithms.
Our results

- **New deadline ordered algorithm** with exact speed requirement
  \[ \alpha_m \leq \frac{e}{e - 1} \approx 1.58 \]
  which is optimal in this class of algorithms.

- **Lower bounds** for non-deadline ordered algorithms
  - EDZL not better than EDF
  - LLF not better than the new algorithm for \( m \geq 7 \)
Our results

- **New deadline ordered algorithm** with exact speed requirement

  \[ \alpha_m \leq \frac{e}{e - 1} \approx 1.58 \]

  which is optimal in this class of algorithms.

- **Lower bounds** for non-deadline ordered algorithms
  - EDZL not better than EDF
  - LLF not better than the new algorithm for \( m \geq 7 \)

- **Improved feasibility test** for recurrent task systems.
Yardstick Schedule by Lam & To (1999)

Key challenge: “lower bound” resp. estimate of an optimal schedule
Yardstick Schedule by Lam & To (1999)

Key challenge: “lower bound” resp. estimate of an optimal schedule

Relaxation: allow parallel processing of job $j$

but only when job is underworked: $p_j - p_j(t) < t - r_j$
Yardstick Schedule by Lam & To (1999)

Key challenge: “lower bound” resp. estimate of an optimal schedule

Relaxation: allow parallel processing of job $j$

$\rightarrow$ but only when job is underworked: $p_j - p_j(t) < t - r_j$

Yardstick algorithm

1. Consider jobs in EDF-order.
2. Schedule each job as early as possible on as many machines as possible until it is not underworked anymore. Then use one machine until completion.
Example yardstick

For technical reasons, we maintain a staircase profile.
Example yardstick

For technical reasons, we maintain a staircase profile.
Example yardstick

For technical reasons, we maintain a staircase profile.

Properties of the yardstick schedule

1. Values of $f_j$ and $x_j$ do not decrease over time.
2. If $d_j < d_k$ then $x_j < x_k$ (whereas $f_j, f_k$ arbitrary).
3. For all $j$, $f_j \geq r_j + p_j$.
4. If an instance is feasible then $f_j \leq d_j$, for all $j$. 
Example yardstick

For technical reasons, we maintain a staircase profile.
Example yardstick

For technical reasons, we maintain a staircase profile.
Example yardstick

For technical reasons, we maintain a staircase profile.
Example yardstick

For technical reasons, we maintain a staircase profile.

In yardstick at time $t$:
- $f_j(t)$: finishing time of $j$
- $x_j(t)$: last time that $j$ is underworked
Example yardstick

For technical reasons, we maintain a staircase profile.

In yardstick at time $t$:
- $f_j(t)$: finishing time of $j$
- $x_j(t)$: last time that $j$ is underworked
Example yardstick

For technical reasons, we maintain a staircase profile.

In yardstick at time $t$:
- $f_j(t)$: finishing time of $j$
- $x_j(t)$: last time that $j$ is underworked

Properties of the yardstick schedule

1. Values of $f_j$ and $x_j$ do not decrease over time.
Example yardstick

For technical reasons, we maintain a staircase profile.

\[ x_1 = x_2 = x_3 = x_4 \]

\[ x_j = x_5 = x_6 \]

In yardstick at time \( t \):
- \( f_j(t) \): finishing time of \( j \)
- \( x_j(t) \): last time that \( j \) is underworked

Properties of the yardstick schedule

1. Values of \( f_j \) and \( x_j \) do not decrease over time.
2. If \( d_j < d_k \) then \( x_j < x_k \) (whereas \( f_j, f_k \) arbitrary).
Example yardstick

For technical reasons, we maintain a staircase profile.

In yardstick at time $t$:
- $f_j(t)$: finishing time of $j$
- $x_j(t)$: last time that $j$ is underworked

Properties of the yardstick schedule

1. Values of $f_j$ and $x_j$ do not decrease over time.
2. If $d_j < d_k$ then $x_j < x_k$ (whereas $f_j, f_k$ arbitrary).
3. For all $j$, $f_j \geq r_j + p_j$. 
Example yardstick

For technical reasons, we maintain a staircase profile.

In yardstick at time $t$:
- $f_j(t)$: finishing time of $j$
- $x_j(t)$: last time that $j$ is underworked

Properties of the yardstick schedule

1. Values of $f_j$ and $x_j$ do not decrease over time.
2. If $d_j < d_k$ then $x_j < x_k$ (whereas $f_j, f_k$ arbitrary).
3. For all $j$, $f_j \geq r_j + p_j$.
4. If an instance is feasible then $f_j \leq d_j$, for all $j$.
New algorithm

**Key idea:** Use extra speed **only if a job is underworked** in yardstick.
New algorithm

**Key idea:** Use extra speed only if a job is underworked in yardstick.

**Our algorithm:**

1. Keep yardstick up to date.
2. Consider remaining jobs in EDF-order.
3. For any job $j$, we schedule (from right to the left):
   - **one unit** in each time slot between $x_j$ and $f_j$.
   - **$\alpha$ units** per time slot between $x_j - (p_j(t) - f_j + x_j)/\alpha$ and $x_j$. 
New algorithm

Key idea: Use extra speed only if a job is underworked in yardstick.

Our algorithm:

1. Keep yardstick up to date.
2. Consider remaining jobs in EDF-order.
3. For any job $j$, we schedule (from right to the left):
   - one unit in each time slot between $x_j$ and $f_j$.
   - $\alpha$ units per time slot between $x_j - (p_j(t) - f_j + x_j)/\alpha$ and $x_j$. 

Nicole Megow
New algorithm

**Key idea:** Use extra speed only if a job is underworked in yardstick.

**Our algorithm:**

1. Keep yardstick up to date.
2. Consider remaining jobs in EDF-order.
3. For any job \( j \), we schedule (from right to the left):
   - one unit in each time slot between \( x_j \) and \( f_j \).
   - \( \alpha \) units per time slot between \( x_j - (p_j(t) - f_j + x_j)/\alpha \) and \( x_j \).
New algorithm

**Key idea:** Use extra speed only if a job is underworked in yardstick.

**Our algorithm:**

1. Keep yardstick up to date.
2. Consider remaining jobs in EDF-order.
3. For any job $j$, we schedule (from right to the left):
   - **one unit** in each time slot between $x_j$ and $f_j$.
   - **$\alpha$ units** per time slot between $x_j - (p_j(t) - f_j + x_j)/\alpha$ and $x_j$
New algorithm

Key idea: Use extra speed only if a job is underworked in yardstick.

Our algorithm:

1. Keep yardstick up to date.
2. Consider remaining jobs in EDF-order.
3. For any job $j$, we schedule (from right to the left):
   - one unit in each time slot between $x_j$ and $f_j$.
   - $\alpha$ units per time slot between $x_j - (p_j(t) - f_j + x_j)/\alpha$ and $x_j$. 

Nicole Megow
New algorithm

Key idea: Use extra speed only if a job is underworked in yardstick.

Our algorithm:

1. Keep yardstick up to date.
2. Consider remaining jobs in EDF-order.
3. For any job $j$, we schedule (from right to the left):
   - one unit in each time slot between $x_j$ and $f_j$.
   - $\alpha$ units per time slot between $x_j - (p_j(t) - f_j + x_j)/\alpha$ and $x_j$.
New algorithm

Key idea: Use extra speed only if a job is underworked in yardstick.

Our algorithm:

1. Keep yardstick up to date.
2. Consider remaining jobs in EDF-order.
3. For any job $j$, we schedule (from right to the left):
   - one unit in each time slot between $x_j$ and $f_j$.
   - $\alpha$ units per time slot between $x_j - (p_j(t) - f_j + x_j)/\alpha$ and $x_j$. 
New algorithm

Key idea: Use extra speed only if a job is underworked in yardstick.

Our algorithm:

1. Keep yardstick up to date.
2. Consider remaining jobs in EDF-order.
3. For any job $j$, we schedule (from right to the left):
   - one unit in each time slot between $x_j$ and $f_j$.
   - $\alpha$ units per time slot between $x_j - (p_j(t) - f_j + x_j)/\alpha$ and $x_j$. 
Feasibility

Lemma

The algorithm assigns
- at most $\alpha$ units of a job to the same slot
Feasibility

Lemma

The algorithm assigns

- at most $\alpha$ units of a job to the same slot
- $p_j$ units only to time slots between $r_j$ and $d_j$
- no workload of $j$ to slots before $t$
Feasibility

Lemma

The algorithm assigns

- at most $\alpha$ units of a job to the same slot
- $p_j$ units only to time slots between $r_j$ and $d_j$
- no workload of $j$ to slots before $t$

- When scheduling at time $r_j$ this is per definition true.
Feasibility

Lemma

The algorithm assigns

- at most $\alpha$ units of a job to the same slot
- $p_j$ units only to time slots between $r_j$ and $d_j$
- no workload of $j$ to slots before $t$

- When scheduling at time $r_j$ this is per definition true.
- Rescheduling $j$ at some later release date $t$, only postpones processing of jobs.
Feasibility

Lemma

The algorithm assigns
- at most $\alpha$ units of a job to the same slot
- $p_j$ units only to time slots between $r_j$ and $d_j$
- no workload of $j$ to slots before $t$

- When scheduling at time $r_j$ this is per definition true.
- Rescheduling $j$ at some later release date $t$, only postpones processing of jobs.

Remains to show: capacity constraint is met for $\alpha = \alpha_m$. 
Resource feasibility

Theorem

The algorithm respects the processing capacity when given speed $\alpha_m$. 
The algorithm respects the processing capacity when given speed $\alpha_m$.

Suppose not. Then there is a step in A that exceeds height $\alpha m$. 
Resource feasibility

**Theorem**
The algorithm respects the processing capacity when given speed $\alpha_m$.

Suppose not. Then there is a step in $A$ that exceeds height $\alpha m$.

---

**Lemma**: At any time, the remaining work of job $j$ in our schedule is not more than that remaining in the yardstick schedule.
Resource feasibility

Theorem
The algorithm respects the processing capacity when given speed $\alpha_m$.

Volume argument

$Y S$

$A$

$t$ $z$
Resource feasibility

Theorem
The algorithm respects the processing capacity when given speed $\alpha_m$.

Volume argument
Partition set of jobs contributing to first step in $A$:
Theorem

The algorithm respects the processing capacity when given speed $\alpha_m$. 

Volume argument

Partition set of jobs contributing to first step in A:

$B := \{j \mid f_j \leq z\}$
Theorem

The algorithm respects the processing capacity when given speed $\alpha_m$.

Volume argument

Partition set of jobs contributing to first step in $A$:

- $B := \{j \mid f_j \leq z\}$
- $C := \{j \mid x_j \leq z \leq f_j\}$
Theorem

The algorithm respects the processing capacity when given speed $\alpha_m$.

Volume argument

Partition set of jobs contributing to first step in $A$:

- $B := \{ j \mid f_j \leq z \}$
- $C := \{ j \mid x_j \leq z \leq f_j \}$
- $D := \{ j \mid z < x_j \}$
The algorithm respects the processing capacity when given speed $\alpha_m$.

**Volume argument**

Partition set of jobs contributing to first step in $A$:

- $B := \{ j \mid f_j \leq z \}$
- $C := \{ j \mid x_j \leq z \leq f_j \}$
- $D := \{ j \mid z < x_j \}$

$$m\alpha(z - t) < \text{volume}(B, C, D)$$
Theorem

The algorithm respects the processing capacity when given speed $\alpha_m$.

- Jobs in B and C contribute in our schedule not more than in YS.
Theorem

The algorithm respects the processing capacity when given speed $\alpha_m$.

- Jobs in B and C contribute in our schedule not more than in YS.
- For $j \in D$ define:
  
  $a_j = \frac{x_j}{z}$
  
  $b_j = \frac{(p_j - (f_j - x_j))}{z}$
Resource feasibility

**Theorem**

The algorithm respects the processing capacity when given speed \( \alpha_m \).

- Jobs in B and C contribute in our schedule not more than in YS.
- For \( j \in D \) define:
  \[
  a_j = \frac{x_j}{z} \\
  b_j = \frac{(p_j - (f_j - x_j))}{z}
  \]
- It is sufficient to set \( \alpha \) to optimum of
  \[
  \max \quad \frac{m + \sum_{i=1}^{D} b_i}{m - k + \sum_{i=1}^{D} a_i} \\
  \text{s.t.} \quad 0 \leq b_i, b_i \leq a_i, a_i \geq 1, \\
  a_i \geq a_{i-1} + b_i/m
  \]
Our results

- **New deadline ordered algorithm** with exact speed requirement

  \[ \alpha_m \leq \frac{e}{e - 1} \approx 1.58 \]

  which is optimal in this class of algorithms.
Our results

- **New deadline ordered algorithm** with exact speed requirement

\[ \alpha_m \leq \frac{e}{e - 1} \approx 1.58 \]

which is optimal in this class of algorithms.

- **Lower bounds** for non-deadline ordered algorithms
  - EDZL not better than EDF
  - LLF not better than the new algorithm for \( m \geq 7 \)
Our results

- **New deadline ordered algorithm** with exact speed requirement
  \[ \alpha_m \leq \frac{e}{e-1} \approx 1.58 \]
  which is optimal in this class of algorithms.

- **Lower bounds** for non-deadline ordered algorithms
  - EDZL not better than EDF
  - LLF not better than the new algorithm for \( m \geq 7 \)

- **Improved feasibility test** for recurrent task systems.
A task system is **feasible** if every (legal) job sequence admits a feasible schedule.
Real-time Scheduling

A task system is **feasible** if every (legal) job sequence admits a feasible schedule.

- Periodic systems: exactly one legal job sequence
- Sporadic systems: infinitely many legal job sequences
Real-time Scheduling

A task system is **feasible** if every (legal) job sequence admits a feasible schedule.

- **Periodic systems**: exactly one legal job sequence
- **Sporadic systems**: infinitely many legal job sequences

**Feasibility Problem**

**Input**: task system $\mathcal{T}$, number of processors $m$

**Output**: YES / NO such that

- YES $\Rightarrow \mathcal{T}$ is feasible on $m$ processors
- NO $\Rightarrow \mathcal{T}$ is not feasible on $m$ processors

Bad news: the feasibility problem is often intractable.
Real-time Scheduling

A task system is feasible if every (legal) job sequence admits a feasible schedule.

- Periodic systems: exactly one legal job sequence
- Sporadic systems: infinitely many legal job sequences

Feasibility Problem

Input: task system $T$, number of processors $m$
Output: YES / NO such that
- YES $\Rightarrow T$ is feasible on $m$ processors
- NO $\Rightarrow T$ is not feasible on $m$ processors

Bad news: the feasibility problem is often intractable.
**σ-Approximate Feasibility Problem**

**Input:** task system $\mathcal{T}$, number of processors $m$

**Output:** YES / NO such that

YES $\Rightarrow \mathcal{T}$ is feasible on $m$ speed-$\sigma$ processors

NO $\Rightarrow \mathcal{T}$ is not feasible on $m$ speed-1 processors
Results for Approximate Testing

Sporadic task systems

- \((2 - \frac{1}{m} + \epsilon)\)-approximate feasibility test
  [Bonifaci, Marchetti-S., Stiller '08]

Periodic task systems

- coNP-hard even if \(\sigma = n^{1-\epsilon}\)
  [Bonifaci, Chan, Marchetti-S., M. '10]
- Synchronous tasks: \((2 - \frac{1}{m} + \epsilon)\)-approximate feasibility test
  [Albers, Slomka '04], [Bonifaci, Chan, Marchetti-S., M. '10]
- Constant \# of task types: \((2 - \frac{1}{m})\)-approximate feasibility test
  [Baruah, Rosier, Howell '90], [Bonifaci, Chan, Marchetti-S., M. '10]
Notion of Workload

Bonifaci, Marchetti-S., Stiller 2008

feasible on $m$ processors $\Rightarrow ffd([t_1, t_2]) \leq m$ for all $[t_1, t_2]$

$\Rightarrow T$ EDF-schedulable on $m$ speed-$(2^\log m)$ processors

FPTAS for approximating max $t_1, t_2$ ffd$([t_1, t_2])$ within $(1 + \epsilon)$
Notion of Workload

\[ \text{ffd}([t_1, t_2]): \]

- For all \([t_1, t_2]\) \[ \text{ffd}([t_1, t_2]) = t_2 - t_1 \leq m \] for all \([t_1, t_2]\)

- \(T\) feasible on \(m\) processors \[ \Rightarrow \text{ffd}([t_1, t_2]) \]

- \(T\) EDF-schedulable on \(m\) speed-\((2 - 1/m)\) processors

- FPTAS for approximating max

\[ t_2 - t_1 \leq m \] for all \([t_1, t_2]\)

\[ \text{ffd}([t_1, t_2]) \]

\[ \text{ffd}([t_1, t_2]) \]

Nicole Megow
Notion of Workload

$\text{ffd}([t_1, t_2])$:

$\text{T feasible on } m \text{ processors } \Rightarrow \frac{\text{ffd}([t_1, t_2])}{t_2 - t_1} \leq m \text{ for all } [t_1, t_2]$

Bonifaci, Marchetti-S., Stiller 2008
Notion of Workload

\[ \text{ffd}([t_1, t_2]): \]

\[ \frac{\text{ffd}([t_1, t_2])}{t_2 - t_1} \leq m \text{ for all } [t_1, t_2] \]

\[ \Rightarrow \mathcal{T} \text{ EDF-schedulable on } m \text{ speed-}(2 - \frac{1}{m}) \text{ processors} \]
Notion of Workload

$$ffd([t_1, t_2]):$$

- \( T \) feasible on \( m \) processors \( \Rightarrow \frac{ffd([t_1, t_2])}{t_2 - t_1} \leq m \) for all \([t_1, t_2] \)
- \( \frac{ffd([t_1, t_2])}{t_2 - t_1} \leq m \) for all \([t_1, t_2] \)
  \( \Rightarrow T \) EDF-schedulable on \( m \) speed-\((2 - \frac{1}{m}) \) processors
- FPTAS for approximating \( \max_{t_1, t_2} \frac{ffd([t_1, t_2])}{t_2 - t_1} \) within \((1 + \epsilon)\)
**ffd and yardstick**

$$ffd([t_1, t_2]):$$

$$ffd$$

$$p_j$$

$$t_1$$

$$t_2$$

$$p_j$$

$$ffd$$

$$t_1$$

$$t_2$$

**yardstick:**

Yields improved approximate feasibility tests.
**Theorem**

\[
\frac{\text{ffd}([t_1, t_2])}{t_2 - t_1} \leq m \text{ for all } [t_1, t_2]
\]

⇒ online yardstick schedule finishes all jobs by their deadlines
ffd and yardstick

ffd([t₁, t₂]):

yardstick:

Theorem

\[
\frac{\text{ffd}([t₁, t₂])}{t₂ - t₁} \leq m \text{ for all } [t₁, t₂]
\]

⇒ online yardstick schedule finishes all jobs by their deadlines

⇒ our algorithm schedules ℋ on m machines of speed α_m < 1.582
**ffd and yardstick**

**ffd([t₁, t₂]):**

**yardstick:**

\[
\text{ffa}([t₁, t₂]) \leq m \quad \text{for all } [t₁, t₂]
\]

⇒ online yardstick schedule finishes all jobs by their deadlines

⇒ our algorithm schedules \( T \) on \( m \) machines of speed \( \alpha_m < 1.582 \)

**Yields improved approximate feasibility tests.**
Summary & Open problems

- New deadline ordered algorithm with optimal speed $\alpha_m \leq \frac{e}{e-1}$.
- Non-deadline ordered algorithms LLF and EDZL do not beat it.
- Improved feasibility tests for recurrent tasks.

Open problem I: improvement using absolute deadlines
- Relative (instead of absolute) laxity?
- Bad news: Any algorithm that must meet yardsticks finishing time requires speed $\alpha_m$.

Open problem II: minimize the number of extra machines
- No constant known, not even $O(m)$-approximation
- Bad news: Any deadline ordered algorithm requires $n$ machines.
Summary & Open problems

- New deadline ordered algorithm with optimal speed $\alpha_m \leq \frac{e}{e-1}$.
- Non-deadline ordered algorithms LLF and EDZL do not beat it.
- Improved feasibility tests for recurrent tasks.

Open problem I: improvement using absolute deadlines
- Relative (instead of absolute) laxity?

Open problem II: minimize the number of extra machines
- No constant known, not even $O(m)$-approximation
- Bad news: Any deadline ordered algorithm requires $n$ machines.
Summary & Open problems

- New deadline ordered algorithm with optimal speed $\alpha_m \leq \frac{e}{e-1}$.
- Non-deadline ordered algorithms LLF and EDZL do not beat it.
- Improved feasibility tests for recurrent tasks.

Open problem I: improvement using absolute deadlines

Open problem II: minimize the number of extra machines
- No constant known, not even $O(m)$-approximation.

- Bad news: Any deadline ordered algorithm requires $n$ machines.
Summary & Open problems

- New deadline ordered algorithm with optimal speed $\alpha_m \leq \frac{e}{e-1}$.
- Non-deadline ordered algorithms LLF and EDZL do not beat it.
- Improved feasibility tests for recurrent tasks.

---

- **Open problem I**: improvement using absolute deadlines
  - Relative (instead of absolute) laxity?

Bad news: Any algorithm that must meet yardsticks finishing time requires speed $\alpha_m \leq \frac{e}{e-1}$.

Bad news: Any deadline ordered algorithm requires $n$ machines.

Nicole Megow
Summary & Open problems

- New deadline ordered algorithm with optimal speed $\alpha_m \leq \frac{e}{e-1}$.
- Non-deadline ordered algorithms LLF and EDZL do not beat it.
- Improved feasibility tests for recurrent tasks.

---

- Open problem I: improvement using absolute deadlines
  - Relative (instead of absolute) laxity?
  - Bad news: Any algorithm that must meet yardsticks finishing time requires speed $\alpha_m$. 
Summary & Open problems

- New deadline ordered algorithm with optimal speed $\alpha_m \leq \frac{e}{e-1}$.
- Non-deadline ordered algorithms LLF and EDZL do not beat it.
- Improved feasibility tests for recurrent tasks.

---

- **Open problem I**: improvement using absolute deadlines
  - Relative (instead of absolute) laxity?
  - Bad news: Any algorithm that must meet yardsticks finishing time requires speed $\alpha_m$.

- **Open problem II**: minimize the number of extra machines
Summary & Open problems

- New deadline ordered algorithm with optimal speed $\alpha_m \leq \frac{e}{e-1}$.
- Non-deadline ordered algorithms LLF and EDZL do not beat it.
- Improved feasibility tests for recurrent tasks.

Open problem I: improvement using absolute deadlines
- Relative (instead of absolute) laxity?
- Bad news: Any algorithm that must meet yardsticks finishing time requires speed $\alpha_m$.

Open problem II: minimize the number of extra machines
- no constant known, not even $O(m)$-approximation
Summary & Open problems

- New deadline ordered algorithm with optimal speed \( \alpha_m \leq \frac{e}{e-1} \).
- Non-deadline ordered algorithms LLF and EDZL do not beat it.
- Improved feasibility tests for recurrent tasks.

---

**Open problem I**: improvement using absolute deadlines
- Relative (instead of absolute) laxity?
- Bad news: Any algorithm that must meet yardsticks finishing time requires speed \( \alpha_m \).

**Open problem II**: minimize the number of extra machines
- no constant known, not even \( O(m) \)-approximation
- Bad news: Any deadline ordered algorithm requires \( n \) machines.