Batch Scheduling with Interval Compatibilities

Tim Nonner
Albert Ludwigs University of Freiburg

IBM Research Zürich
April 27th 2010
What is batch scheduling?

Batch scheduling is about optimally grouping jobs into batches.

Example: jobs $\approx$ containers, batches $\approx$ ships

Many other examples: jobs $\approx$ messages in a network, parts in a manufacturing process, you, ...
What is batch scheduling?

**Batch scheduling** is about optimally grouping jobs into batches.

**Example:** jobs $\approx$ containers, batches $\approx$ ships

Many other examples: jobs $\approx$ messages in a network, parts in a manufacturing process, you, ...
What is batch scheduling?

**Batch scheduling** is about optimally grouping jobs into batches.

**Example**: jobs $\approx$ containers, batches $\approx$ ships

Many other examples: jobs $\approx$ messages in a network, parts in a manufacturing process, you, ...
What is batch scheduling?

**Batch scheduling** is about optimally grouping jobs into batches.

**Example:** jobs $\approx$ containers, batches $\approx$ ships

Many other examples: jobs $\approx$ messages in a network, parts in a manufacturing process, you, ...
Defining a batching problem

Given is a set of jobs $J = \{1, 2, \ldots, n\}$, what else?

- Which jobs may be added to the same batch?
- What is the cost of each batch?
- How many jobs may be added to each batch?
Defining a batching problem

Given is a set of jobs $J = \{1, 2, \ldots, n\}$, what else?

- Which jobs may be added to the same batch?
- What is the cost of each batch?
- How many jobs may be added to each batch?
Defining a batching problem

Given is a set of jobs $J = \{1, 2, \ldots, n\}$, what else?

- Which jobs may be added to the same batch?
- What is the cost of each batch?
- How many jobs may be added to each batch?
Defining a batching problem

Given is a set of jobs $J = \{1, 2, \ldots, n\}$, what else?

- Which jobs may be added to the same batch?
- What is the cost of each batch?
- How many jobs may be added to each batch?
Which jobs may be added to the same batch?

**Compatibility constraint:** each job has an associated interval with endpoints in \( \{1, 2, \ldots, T\} \), the periods, such that two jobs may be added to the same batch iff their intervals intersect. Assume that \( T := 2n \).

We use interval instead of job, and if \( I \in C \) then \( I \) is stabbed by \( C \) and assigned to period \( t_C \) \( \implies \) each batch forms a clique in the corresponding interval graph.
Which jobs may be added to the same batch?

**Compatibility constraint:** each job has an associated interval with endpoints in \( \{1, 2, \ldots, T\} \), the periods, such that two jobs may be added to the same batch iff their intervals intersect. Assume that \( T := 2n \).

We use interval instead of job, and if \( I \in C \) then \( I \) is stabbed by \( C \) and assigned to period \( t_C \) \( \implies \) each batch forms a clique in the corresponding interval graph.
Which jobs may be added to the same batch?

Compatibility constraint: each job has an associated interval with endpoints in \( \{1, 2, \ldots, T\} \), the periods, such that two jobs may be added to the same batch iff their intervals intersect. Assume that \( T := 2n \).

We use interval instead of job, and if \( I \in C \) then \( I \) is stabbed by \( C \) and assigned to period \( t_C \) \( \implies \) each batch forms a clique in the corresponding interval graph.
What is the cost of each batch?

Each interval $I$ has a weight $w_I \in \mathbb{R}^+$ such that the cost of a batch $C$ is $w_C := \max_{I \in C} w_I$, called max-batching.

Schedule $\sigma$ is set of batches with $\text{cost}(\sigma) = w_C + w_{C'}$.

Motivation: interval $\approx$ temperature range of coil, weight $\approx$ burning time in oven (Hochbaum et al.'97)
What is the cost of each batch?

Each interval $I$ has a weight $w_I \in \mathbb{R}^+$ such that the cost of a batch $C$ is $w_C := \max_{I \in C} w_I$, called max-batching.

Schedule $\sigma$ is set of batches with $\text{cost}(\sigma) = w_C + w_{C'}$.

Motivation: interval $\approx$ temperature range of coil, weight $\approx$ burning time in oven (Hochbaum et al.'97)
What is the cost of each batch?

Each interval $I$ has a weight $w_I \in \mathbb{R}^+$ such that the cost of a batch $C$ is $w_C := \max_{I \in C} w_I$, called max-batching.

Schedule $\sigma$ is set of batches with $\text{cost}(\sigma) = w_C + w_{C'}$.

Motivation: interval $\approx$ temperature range of coil, weight $\approx$ burning time in oven (Hochbaum et al.'97)
What is the cost of each batch?

Each interval \( I \) has a weight \( w_I \in \mathbb{R^+} \) such that the cost of a batch \( C \) is \( w_C := \max_{I \in C} w_I \), called max-batching.

\[
\begin{align*}
  w_C &= w_I \\
  t_C & \quad I \quad t_C'
\end{align*}
\]

Schedule \( \sigma \) is set of batches with \( \text{cost}(\sigma) = w_C + w_{C'} \).

Motivation: interval \( \approx \) temperature range of coil, weight \( \approx \) burning time in oven (Hochbaum et al.'97)
How many intervals may be added to each batch?

**Capacity constraints:**
- **uniform:** $|C| \leq k$ for each batch $C$
- **non-uniform:** $|C| \leq k_t$ for each batch $C$ with $t_C = t$
- **no:** $k = \infty$

Analogously for weights:
- **no $\approx$ uniform:** all interval weights $w_I$ are identical
- **non-uniform:** arbitrary interval weights $w_I$

... Capacitated max-Batching with interval graph compatibilities

We write capacity/weights to describe special case.
How many intervals may be added to each batch?

**Capacity constraints:**

- **uniform:** \(|C| \leq k\) for each batch \(C\)
- **non-uniform:** \(|C| \leq k_t\) for each batch \(C\) with \(t_C = t\)
- **no:** \(k = \infty\)

Analogously for weights:

- **no \approx uniform:** all interval weights \(w_I\) are identical
- **non-uniform:** arbitrary interval weights \(w_I\)

... Capacitated max-Batching with interval graph compatibilities

We write capacity/weights to describe special case.
How many intervals may be added to each batch?

**Capacity constraints:**

- **uniform:** $|C| \leq k$ for each batch $C$
- **non-uniform:** $|C| \leq k_t$ for each batch $C$ with $t_C = t$
- **no:** $k = \infty$

Analogously for weights:

- **no $\approx$ uniform:** all interval weights $w_I$ are identical
- **non-uniform:** arbitrary interval weights $w_I$

... Capacitated max-Batching with interval graph compatibilities

We write capacity/weights to describe special case.
How many intervals may be added to each batch?

Capacity constraints:
- uniform: \(|C| \leq k\) for each batch \(C\)
- non-uniform: \(|C| \leq k_t\) for each batch \(C\) with \(t_C = t\)
- no: \(k = \infty\)

Analogously for weights:
- no \(\approx\) uniform: all interval weights \(w_I\) are identical
- non-uniform: arbitrary interval weights \(w_I\)

... Capacitated max-Batching with interval graph compatibilities

We write capacity/weights to describe special case.
How many intervals may be added to each batch?

Capacity constraints:
- uniform: $|C| \leq k$ for each batch $C$
- non-uniform: $|C| \leq k_t$ for each batch $C$ with $t_C = t$
- no: $k = \infty$

Analogously for weights:
- no $\approx$ uniform: all interval weights $w_I$ are identical
- non-uniform: arbitrary interval weights $w_I$

... Capacitated max-Batching with interval graph compatibilities

We write capacity/weights to describe special case.
How many intervals may be added to each batch?

Capacity constraints:

- **uniform**: $|C| \leq k$ for each batch $C$
- **non-uniform**: $|C| \leq k_t$ for each batch $C$ with $t_C = t$
- **no**: $k = \infty$

Analogously for weights:

- **no \approx uniform**: all interval weights $w_I$ are identical
- **non-uniform**: arbitrary interval weights $w_I$

... Capacitated max-Batching with interval graph compatibilities

We write capacity/weights to describe special case.
Approximation algorithms

P vs. NP

An $\alpha$-approximation algorithm finds a schedule $\sigma$ with

- $\text{cost}(\sigma) \leq \alpha \cdot \text{OPT}$ in
- time polynomial in $n$.

PTAS: for any $\alpha > 1$, there is an $\alpha$-approximation algorithm.
Approximation algorithms

P vs. NP

An \( \alpha \)-approximation algorithm finds a schedule \( \sigma \) with
- \( \text{cost}(\sigma) \leq \alpha \cdot \text{OPT} \) in
- time polynomial in \( n \).

PTAS: for any \( \alpha > 1 \), there is an \( \alpha \)-approximation algorithm.
Approximation algorithms

P vs. NP

An $\alpha$-approximation algorithm finds a schedule $\sigma$ with
- $\text{cost}(\sigma) \leq \alpha \cdot \text{OPT}$ in
- time polynomial in $n$.

PTAS: for any $\alpha > 1$, there is an $\alpha$-approximation algorithm.
Outline

- **Previous work:** greedy and dynamic programming for special cases.

- **New result:** dynamic program for the general case of non-uniform capacities and weights.

Greedy: iteratively take the best extension of a partial solution.

Dynamic Programming: combine optimal solutions of subproblems to optimal solutions of larger subproblems via recurrence relation:

- subproblems $\Rightarrow$ DP array
- recurrence relation
Outline

- **Previous work:** greedy and dynamic programming for special cases.
- **New result:** dynamic program for the general case of non-uniform capacities and weights.

**Greedy:** iteratively take the best extension of a partial solution.

**Dynamic Programming:** combine optimal solutions of subproblems to optimal solutions of larger subproblems via recurrence relation:

- subproblems $\Rightarrow$ DP array
- recurrence relation
Outline

- **Previous work:** greedy and dynamic programming for special cases.
- **New result:** dynamic program for the general case of non-uniform capacities and weights.

**Greedy:** iteratively take the best extension of a partial solution.

**Dynamic Programming:** combine optimal solutions of subproblems to optimal solutions of larger subproblems via recurrence relation:

- subproblems $\Rightarrow$ DP array
- recurrence relation
**Greedy:** stab the leftmost interval at its right endpoint and as many other intervals as possible. Repeat with remaining unstabbed intervals.

⇒ optimal (also on-line).
**Greedy:** stab the leftmost interval at its right endpoint and as many other intervals as possible. Repeat with remaining unstabbed intervals.

⇒ optimal (also on-line).
Greedy: stab the leftmost interval at its right endpoint and as many other intervals as possible. Repeat with remaining unstabbed intervals.

⇒ optimal (also on-line).
**Greedy**: stab the leftmost interval at its right endpoint and as many other intervals as possible. Repeat with remaining unstabbed intervals.

⇒ optimal (also on-line).
**Greedy:** stab the leftmost interval at its right endpoint and the up to $k - 1$ leftmost other intervals. Repeat with remaining unstabbed intervals.

$\Rightarrow$ optimal (also on-line).
**Greedy:** stab the leftmost interval at its right endpoint with a batch of twice the necessary weight, and add all possible intervals. Repeat with remaining unstabbed intervals.

⇒ 4-approximation algorithm (also on-line)
no/non-uniform

**Greedy:** stab the leftmost interval at its right endpoint with a batch of twice the necessary weight, and add all possible intervals. Repeat with remaining unstabbed intervals.

\[ w_C = 2w_I \]

\[ \Rightarrow 4\text{-approximation algorithm (also on-line)} \]
**Greedy:** stab the leftmost interval at its right endpoint with a batch of twice the necessary weight, and add all possible intervals. Repeat with remaining unstabbed intervals.

\[ w_C = 2 w_I \]

\[ \Rightarrow 4\text{-approximation algorithm (also on-line)} \]
Dynamic Programming: let $J(s, t)$ be the subproblem consisting of all intervals between periods $s$ and $t$.

To define recurrence relation, choose interval $I$ with maximal $w_I$.

$\Rightarrow$ DP in time $O(n^3)$ (Finke et al.'08, Bechetti et al.'06)
Dynamic Programming: let $J(s, t)$ be the subproblem consisting of all intervals between periods $s$ and $t$.

To define recurrence relation, choose interval $I$ with maximal $w_I$.

⇒ DP in time $O(n^3)$ (Finke et al.'08, Bechetti et al.'06)
Dynamic Programming: let $J(s, t)$ be the subproblem consisting of all intervals between periods $s$ and $t$.

To define recurrence relation, choose interval $I$ with maximal $w_I$.

$\Rightarrow$ DP in time $\mathcal{O}(n^3)$ (Finke et al.'08, Bechetti et al.'06)
Dynamic Programming: let $J(s, t)$ be the subproblem consisting of all intervals between periods $s$ and $t$.

To define recurrence relation, choose interval $I$ with maximal $w_I$.

$\Rightarrow$ DP in time $\mathcal{O}(n^3)$ (Finke et al.'08, Bechetti et al.'06)
Dynamic Programming: let \( J(s, t) \) be the subproblem consisting of all intervals between periods \( s \) and \( t \).

To define recurrence relation, choose interval \( I \) with maximal \( w_I \).

\[ \Rightarrow \text{DP in time } O(n^3) \text{ (Finke et al.'08, Bechetti et al.'06)} \]
First DP for no/non-uniform, and then decompose each batch top-down.

$\Rightarrow$ 2-approximation algorithm (Correa et al.'09)
First DP for no/non-uniform, and then decompose each batch top-down.

⇒ 2-approximation algorithm (Correa et al.'09)
Dynamic Programming: let $J(s, t, r)$ be the subproblem consisting of all intervals in $\{I_1, I_2, \ldots, I_r\}$ with left endpoint between periods $s$ and $t$. All these intervals need to be assigned to periods between $s$ and $t$.

$J(s, t, r)$ for $r = 5$

$\Rightarrow$ DP in time $O(n^4)$ (Even et al.’08, Baptiste’06)
## Previous work and results

<table>
<thead>
<tr>
<th>capacities</th>
<th>weights</th>
<th>method</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>no</td>
<td>greedy</td>
</tr>
<tr>
<td>uniform</td>
<td>no</td>
<td>greedy</td>
</tr>
<tr>
<td>no</td>
<td>non-uniform</td>
<td>greedy</td>
</tr>
<tr>
<td>no</td>
<td>non-uniform</td>
<td>DP</td>
</tr>
<tr>
<td>uniform</td>
<td>non-uniform</td>
<td>DP+top-down</td>
</tr>
<tr>
<td>non-uniform</td>
<td>no</td>
<td>DP</td>
</tr>
<tr>
<td>uniform</td>
<td>non-uniform</td>
<td>?</td>
</tr>
<tr>
<td>non-uniform</td>
<td>non-uniform</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>capacities</th>
<th>weights</th>
<th>method</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-uniform</td>
<td>no</td>
<td>opt</td>
</tr>
<tr>
<td>uniform</td>
<td>non-uniform</td>
<td>4-approx.</td>
</tr>
<tr>
<td>non-uniform</td>
<td>non-uniform</td>
<td>2-approx.</td>
</tr>
<tr>
<td>uniform</td>
<td>non-uniform</td>
<td>?</td>
</tr>
<tr>
<td>non-uniform</td>
<td>non-uniform</td>
<td>?</td>
</tr>
</tbody>
</table>

### Results (accepted to SWAT’10):

- strongly NP-hard, even for $k = 3$ (tight, since $k = 2 \in P$)
- PTAS for constant $k_t \geq 1$ (based on dynamic program)
### Previous work and results

<table>
<thead>
<tr>
<th>Capacitites</th>
<th>Weights</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>no</td>
<td>greedy</td>
</tr>
<tr>
<td>uniform</td>
<td>no</td>
<td>greedy</td>
</tr>
<tr>
<td>no</td>
<td>non-uniform</td>
<td>greedy</td>
</tr>
<tr>
<td>no</td>
<td>non-uniform</td>
<td>DP</td>
</tr>
<tr>
<td>uniform</td>
<td>non-uniform</td>
<td>DP + top-down</td>
</tr>
<tr>
<td>non-uniform</td>
<td>no</td>
<td>DP</td>
</tr>
<tr>
<td>uniform</td>
<td>non-uniform</td>
<td>?</td>
</tr>
<tr>
<td>non-uniform</td>
<td>non-uniform</td>
<td>?</td>
</tr>
</tbody>
</table>

#### Results (accepted to SWAT’10):

- strongly NP-hard, even for $k = 3$ (tight, since $k = 2 \in P$)
- PTAS for constant $k_t \geq 1$ (based on dynamic program)
## Previous work and results

<table>
<thead>
<tr>
<th>capacities</th>
<th>weights</th>
<th>method</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>no</td>
<td>greedy opt</td>
</tr>
<tr>
<td>uniform</td>
<td>no</td>
<td>greedy opt</td>
</tr>
<tr>
<td>no</td>
<td>non-uniform</td>
<td>greedy 4-approx.</td>
</tr>
<tr>
<td>no</td>
<td>non-uniform</td>
<td>DP opt</td>
</tr>
<tr>
<td>uniform</td>
<td>non-uniform</td>
<td>DP+top-down 2-approx.</td>
</tr>
<tr>
<td>non-uniform</td>
<td>no</td>
<td>DP opt</td>
</tr>
<tr>
<td>uniform</td>
<td>non-uniform</td>
<td>?</td>
</tr>
<tr>
<td>non-uniform</td>
<td>non-uniform</td>
<td>?</td>
</tr>
</tbody>
</table>

### Results (accepted to SWAT’10):

- strongly NP-hard, even for $k = 3$ (tight, since $k = 2 \in P$)
- PTAS for constant $k_t \geq 1$ (based on dynamic program)
Reducing weight classes

A weight class is a set of intervals with the same weight. Let \( m \) be the number of weight classes.

\[
\begin{array}{c}
\text{weight classes} \\
\text{m} \\
\text{...} \\
\text{2} \\
\text{1}
\end{array}
\]

Theorem: For any \( \epsilon > 0 \), by adding an \( (1 + \epsilon) \)-factor to the approximation ratio, we may assume that \( m \) is constant.

Proof sketch: Geometrically round all interval weights, and then apply the shifting technique to resulting weight classes. \( \square \)
Reducing weight classes

A weight class is a set of intervals with the same weight. Let $m$ be the number of weight classes.

Theorem: For any $\epsilon > 0$, by adding an $(1 + \epsilon)$-factor to the approximation ratio, we may assume that $m$ is constant.

Proof sketch: Geometrically round all interval weights, and then apply the shifting technique to resulting weight classes. □
Reducing weight classes

A weight class is a set of intervals with the same weight. Let \( m \) be the number of weight classes.

Theorem: For any \( \epsilon > 0 \), by adding an \( (1 + \epsilon) \)-factor to the approximation ratio, we may assume that \( m \) is constant.

Proof sketch: Geometrically round all interval weights, and then apply the shifting technique to resulting weight classes. □
Why does DP for no/non-uniform not work for $k < \infty$?

Which intervals should be stabbed by $C$?

Should an unstabbed interval be added to $J(s, t_C)$ or $J(t_C, t)$? (left-right assignment dilemma)

Assume that $m = 1$ $\implies$ no weights
Why does DP for no/non-uniform not work for $k < \infty$?

Which intervals should be stabbed by $C$?

Should an unstabbed interval be added to $J(s, t_C)$ or $J(t_C, t)$? (left-right assignment dilemma)

Assume that $m = 1 \iff$ no weights
Left-right assignment dilemma

Let $R$ be a set of intervals with non-empty intersection.

Let $\sigma^*$ be some optimal schedule (for appropriate $k_t$).

Problem: first assign all intervals either to the left and right without knowing $\sigma^*$, and then, satisfying this left-right assignment, construct a schedule $\sigma$ which

- assigns less intervals to each period than $\sigma^*$ and
  \[ \Rightarrow \text{cost}(\sigma) \leq \text{cost}(\sigma^*) \]
- assigns as many intervals as possible.
Left-right assignment dilemma

Let $R$ be a set of intervals with non-empty intersection.

Let $\sigma^*$ be some optimal schedule (for appropriate $k_t$).

Problem: first assign all intervals either to the left and right without knowing $\sigma^*$, and then, satisfying this left-right assignment, construct a schedule $\sigma$ which

- assigns less intervals to each period than $\sigma^*$ and ($\Rightarrow \text{cost}(\sigma) \leq \text{cost}(\sigma^*)$)
- assigns as many intervals as possible.
Left-right assignment dilemma

Let $R$ be a set of intervals with non-empty intersection.

Let $\sigma^*$ be some optimal schedule (for appropriate $k_t$).

Problem: first assign all intervals either to the left and right without knowing $\sigma^*$, and then, satisfying this left-right assignment, construct a schedule $\sigma$ which

- assigns less intervals to each period than $\sigma^*$ and $\Rightarrow \text{cost}(\sigma) \leq \text{cost}(\sigma^*)$
- assigns as many intervals as possible.
Left-right assignment dilemma

Let $R$ be a set of intervals with non-empty intersection.

Let $\sigma^*$ be some optimal schedule (for appropriate $k_t$).

**Problem:** first assign all intervals either to the left and right without knowing $\sigma^*$, and then, satisfying this left-right assignment, construct a schedule $\sigma$ which

- assigns less intervals to each period than $\sigma^*$ and
  \[ \Rightarrow \text{cost}(\sigma) \leq \text{cost}(\sigma^*) \]
- assigns as many intervals as possible.
Let $R$ be a set of intervals with non-empty intersection.

Let $\sigma^*$ be some optimal schedule (for appropriate $k_t$).

**Problem:** first assign all intervals either to the left and right without knowing $\sigma^*$, and then, satisfying this left-right assignment, construct a schedule $\sigma$ which

- assigns less intervals to each period than $\sigma^*$ and
  \[ \Rightarrow \text{cost}(\sigma) \leq \text{cost}(\sigma^*) \]
- assigns as many intervals as possible.
Left-right assignment dilemma

Let $R$ be a set of intervals with non-empty intersection.

Let $\sigma^*$ be some optimal schedule (for appropriate $k_t$).

**Problem:** first assign all intervals either to the left and right without knowing $\sigma^*$, and then, satisfying this left-right assignment, construct a schedule $\sigma$ which

- assigns less intervals to each period than $\sigma^*$ and
  ($\Rightarrow \text{cost}(\sigma) \leq \text{cost}(\sigma^*)$)
- assigns as many intervals as possible.
Left-right assignment dilemma

Let $R$ be a set of intervals with non-empty intersection.

Let $\sigma^*$ be some optimal schedule (for appropriate $k_t$).

**Problem:** first assign all intervals either to the left and right without knowing $\sigma^*$, and then, satisfying this left-right assignment, construct a schedule $\sigma$ which

- assigns less intervals to each period than $\sigma^*$ and $(\Rightarrow \text{cost}(\sigma) \leq \text{cost}(\sigma^*))$
- assigns as many intervals as possible.
Let $R$ be a set of intervals with non-empty intersection.

Let $\sigma^*$ be some optimal schedule (for appropriate $k_t$).

**Problem:** first assign all intervals either to the left and right without knowing $\sigma^*$, and then, satisfying this left-right assignment, construct a schedule $\sigma$ which

- assigns less intervals to each period than $\sigma^*$ and
  ($\Rightarrow \text{cost}(\sigma) \leq \text{cost}(\sigma^*)$)
- assigns as many intervals as possible.
Left-right assignment dilemma

Let $R$ be a set of intervals with non-empty intersection.

Let $\sigma^*$ be some optimal schedule (for appropriate $k_t$).

**Problem:** first assign all intervals either to the left and right without knowing $\sigma^*$, and then, satisfying this left-right assignment, construct a schedule $\sigma$ which

- assigns less intervals to each period than $\sigma^*$ and
  $(\Rightarrow \text{cost}(\sigma) \leq \text{cost}(\sigma^*))$
- assigns as many intervals as possible.
Left-right assignment dilemma

Let $R$ be a set of intervals with non-empty intersection.

Let $\sigma^*$ be some optimal schedule (for appropriate $k_t$).

Problem: first assign all intervals either to the left and right without knowing $\sigma^*$, and then, satisfying this left-right assignment, construct a schedule $\sigma$ which

- assigns less intervals to each period than $\sigma^*$ and
  \[ \Rightarrow \text{cost}(\sigma) \leq \text{cost}(\sigma^*) \]
- assigns as many intervals as possible.
Left-right assignment dilemma (2)

Can we increase the number intervals assigned by $\sigma$ by increasing the number of considered interval sets $A$? (two subsets $\Rightarrow$ half of intervals are assigned, three subsets $\Rightarrow$ ?, ...)

Theorem (L-R): For any $\epsilon > 0$, we can compute a set of subsets $K \subseteq \mathcal{P}(R)$ of constant size $C = 4^{1/\epsilon}$ in polynomial time such that, for any optimal schedule $\sigma^*$, there is an interval set $A \in K$ and a schedule $\sigma$ that

- assigns each interval $I \in A$ to the left, and each interval $I \in R \setminus A$ to the right,
- assigns less intervals to each period than $\sigma^*$,
- assigns all intervals in $R$ except at most $\epsilon \cdot |R|$ many.

Proof sketch: Given some $\sigma^*$, construct an appropriate interval set $A \subseteq R$ making $2/\epsilon$ boolean decisions. Let then $K \subseteq \mathcal{P}(R)$ be the set containing all $2^{2/\epsilon} = 4^{1/\epsilon}$ possible sets $A$. □
Left-right assignment dilemma (2)

Can we increase the number intervals assigned by $\sigma$ by increasing the number of considered interval sets $A$? (two subsets $\Rightarrow$ half of intervals are assigned, three subsets $\Rightarrow$ ?, ...)

Theorem (L-R): For any $\epsilon > 0$, we can compute a set of subsets $K \subseteq \mathcal{P}(R)$ of constant size $C = 4^{1/\epsilon}$ in polynomial time such that, for any optimal schedule $\sigma^*$, there is an interval set $A \in K$ and a schedule $\sigma$ that

- assigns each interval $I \in A$ to the left, and each interval $I \in R \setminus A$ to the right,
- assigns less intervals to each period than $\sigma^*$,
- assigns all intervals in $R$ except at most $\epsilon \cdot |R|$ many.

Proof sketch: Given some $\sigma^*$, construct an appropriate interval set $A \subseteq R$ making $2/\epsilon$ boolean decisions. Let then $K \subseteq \mathcal{P}(R)$ be the set containing all $2^{2/\epsilon} = 4^{1/\epsilon}$ possible sets $A$. □
Left-right assignment dilemma (2)

Can we increase the number intervals assigned by $\sigma$ by increasing the number of considered interval sets $A$? (two subsets $\Rightarrow$ half of intervals are assigned, three subsets $\Rightarrow$ ?, ...)

**Theorem (L-R):** For any $\epsilon > 0$, we can compute a set of subsets $K \subseteq \mathcal{P}(R)$ of constant size $C = 4^{1/\epsilon}$ in polynomial time such that, for any optimal schedule $\sigma^*$, there is an interval set $A \in K$ and a schedule $\sigma$ that

- assigns each interval $I \in A$ to the left, and each interval $I \in R \setminus A$ to the right,
- assigns less intervals to each period than $\sigma^*$,
- assigns all intervals in $R$ except at most $\epsilon \cdot |R|$ many.

**Proof sketch:** Given some $\sigma^*$, construct an appropriate interval set $A \subseteq R$ making $2/\epsilon$ boolean decisions. Let then $K \subseteq \mathcal{P}(R)$ be the set containing all $2^{2/\epsilon} = 4^{1/\epsilon}$ possible sets $A$. □
Application of L-R Theorem via decomposition

Iteratively half 1, 2, \ldots, T yielding a tree of depth $\log T = \mathcal{O}(\log n)$ with vertices $u$ corresponding to positions $p_u \in [1, T]$.

\begin{center}
\begin{tikzpicture}
    \node (pu) at (3,0) {$p_u$};
    \node (pw) at (1,0) {$p_w$};
    \node (pv) at (2,0) {$p_v$};
    \draw[->] (1,0) -- (3,0);
    \node at (0,0) {1}; \node at (1,0) {2}; \node at (2,0) {$\cdots$}; \node at (3,0) {$T$};
\end{tikzpicture}
\end{center}

Let $R_u$ be all intervals where $u$ is the vertex of minimal depth such that $p_u \in I \implies$ Apply the L-R Theorem yielding a set $K_u \subseteq \mathcal{P}(R_u)$ with $|K_u| \leq C$.

Dynamic Programming: each vertex $v$ corresponds to a subproblem between periods $s_v$ and $t_v$ ...
Application of L-R Theorem via decomposition

Iteratively half 1, 2, \ldots, \, T yielding a tree of depth \( \log T = \mathcal{O}(\log n) \) with vertices \( u \) corresponding to positions \( p_u \in [1, \, T] \).

\[ p_u \]
\[ p_w \]
\[ p_v \]
\[ \in R_u \]

Let \( R_u \) be all intervals where \( u \) is the vertex of minimal depth such that \( p_u \in I \Rightarrow \) Apply the L-R Theorem yielding a set \( K_u \subseteq \mathcal{P}(R_u) \) with \( |K_u| \leq C \).

Dynamic Programming: each vertex \( v \) corresponds to a subproblem between periods \( s_v \) and \( t_v \) ...
Application of L-R Theorem via decomposition

Iteratively half 1, 2, \ldots, \ T yielding a tree of depth \ \log T = O(\log n)\ with \ vertices \ u \ corresponding \ to \ positions \ p_u \in [1, \ T].

\[
\begin{align*}
&\text{Let } R_u \text{ be all intervals where } u \text{ is the vertex of minimal depth such that } p_u \in I \implies \\
&\text{Apply the L-R Theorem yielding a set } K_u \subseteq \mathcal{P}(R_u) \text{ with } |K_u| \leq C.
\end{align*}
\]

**Dynamic Programming**: each vertex \( v \) corresponds to a subproblem between periods \( s_v \) and \( t_v \) ...
Application of L-R Theorem via decomposition

Iteratively half 1, 2, \ldots, T yielding a tree of depth $\log T = O(\log n)$ with vertices $u$ corresponding to positions $p_u \in [1, T]$.

<table>
<thead>
<tr>
<th>#endpoints</th>
<th>$p_u$</th>
<th>$C^{O(\log n)} = n^{O(1)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zero</td>
<td></td>
<td></td>
</tr>
<tr>
<td>two</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Let $R_u$ be all intervals where $u$ is the vertex of minimal depth such that $p_u \in I \implies$ Apply the L-R Theorem yielding a set $K_u \subseteq \mathcal{P}(R_u)$ with $|K_u| \leq C$.

**Dynamic Programming:** each vertex $v$ corresponds to a subproblem between periods $s_v$ and $t_v$ ...
Conclusion

- DP can be extended to incorporate gap penalization, etc: powerful technique for other interval stabbing type problems?
- Somehow related to Arora’s PTAS in the plane (hierarchical decompositon).
- To obtain polynomial running time, we require that $k_t$ are constants. This allows us to brute-force assign all intervals which are not yet assigned.

Open problems:

- PTAS for arbitrary $k_t$ or $k$?
- Polynomial time algorithm for constant $k$ and $m$?
- Flow instead of deadlines ($\approx$ holding or delay cost)?
- Practicability of ideas?

THANK YOU FOR ATTENTION? QUESTIONS?
Conclusion

- DP can be extended to incorporate gap penalization, etc: powerful technique for other interval stabbing type problems?
- Somehow related to Arora’s PTAS in the plane (hierarchical decomposition).
- To obtain polynomial running time, we require that $k_t$ are constants. This allows us to brute-force assign all intervals which are not yet assigned.

Open problems:

- PTAS for arbitrary $k_t$ or $k$?
- Polynomial time algorithm for constant $k$ and $m$?
- Flow instead of deadlines ($\approx$ holding or delay cost)?
- Practicability of ideas?

THANK YOU FOR ATTENTION? QUESTIONS?