Approximation algorithms for the job interval scheduling problem

presented by Daniel Bauer
supervised by Kelin Luo

Rheinische Friedrich-Wilhelms-Universität Bonn

January 31, 2023
Structure of presentation

1. The job interval scheduling problem
2. The paper: methods and results
3. 1.582 Approximation for arbitrary instances
   - First phase
     - First iteration
     - $i$-th iteration
     - Last iteration
   - Second phase
     - ILP Formulation
     - ILP Relaxation Rounding
   - Main Result
4. Further results
   - Dynamic programming algorithm
   - Poly. approx. scheme
Definition (Job interval scheduling problem)

**Given**: set of jobs $S = \{1, \ldots, n\}$

- job $j \in S$ is a set of intervals $j \subseteq \mathbb{I} (\mathbb{R})$ and defined by
  - $r_j$: release date
  - $d_j$: deadline
  - $p_j$: processing time

**Objective**: maximise amount of scheduled jobs

- **integer** values only
- example $r_j = 3, d_j = 7, p_j = 2 \rightarrow j = \{[3; 5], [4; 6], [5; 7]\}$
Alice’s schedule

- Alice is a consult
- works on **timeline** [08:00; 16:00]
- schedule as many **clients/jobs** $S = \{Bob, Chad, Tom\}$ as possible
- cannot meet two clients at once (it’s confidential)
- every client has **time window** (release and deadline)
  - Bob takes 2 hours to consult $\rightarrow p_{Bob} = 2$
  - time from 10:00 to 14:00 $\rightarrow r_{Bob} = 10:00, d_{Bob} = 14:00$
Alice’s schedule

How to schedule the clients?

Bob = \{[10:00; 12:00], [11:00; 13:00], [12:00; 14:00]\},
Chad = \{[10:00; 12:00]\},
Tom = \{[11:00; 12:00], [12:00; 13:00]\}

Schedule 1:

Bob

Schedule 2:

Chad  Tom
JISP variant examples

Alternative **models**
- multiple time lines / machines
- one job has multiple processing times

Alternative **objectives**
**obj.:** min. max. lateness
- allow jobs scheduled after deadline
- lateness = completion time − deadline

**obj.:** \( \sum_{j \in S|j_{\text{scheduled}}} w_j \times j \) with \( w_j \geq 0 \)

**Example weighted instance:**
Bob pays 6€, Chad pays 3€, Tom pays 1€;
Alice wants to max. profit
Complexity overview

- NP-complete
- Reduction to satisfiability
- Sub-problems can be easier to solve
  - Singleton job i.e.: $j = \{[x; y]\}$ reduction to stable set
  - Polynomial solvable
Consider a JISP with release, deadline and processing time.

**Definition (GREEDY)**

If the machine (i.e. Alice) becomes idle, schedule that job with the earliest finish time.

**Example**

\[ j_0 = \{[0; 2], \ldots, [4; 6]\}, j_1 = \{[1, 3]\}, j_2 = \{[7, 10]\} \]
Consider a JISP with release, deadline and processing time.

**Definition (GREEDY)**

If the machine (i.e. Alice) becomes idle, schedule that job with the earliest finish time.

**Example**

\[ j_0 = \{[0; 2], \ldots, [4; 6]\}, j_1 = \{[1, 3]\}, j_2 = \{[7, 10]\} \]
Consider a JISP with release, deadline and processing time.

**Definition (GREEDY)**

If the machine (i.e. Alice) becomes idle, schedule that job with the earliest finish time.

**Example**

\[ j_0 = \{[0; 2], \ldots, [4; 6]\}, j_1 = \{[1, 3]\}, j_2 = \{[7, 10]\} \]
Consider a JISP with release, deadline and processing time.

**Definition (GREEDY)**

If the machine (i.e. Alice) becomes idle, schedule that job with the earliest finish time.

**Example**

\[ j_0 = \{ [0; 2], \ldots, [4; 6] \}, j_1 = \{ [1; 3] \}, j_2 = \{ [7; 10] \} \]
Approximation Factor

- **GREEDY** obtains tight approx of 2 (small instance)
- let $k = \max_{j \in S} |j|$ max. job size
- **GREEDY** 2-approx asymptotically $\lim_{k \to \infty}$
- best approx for general case so far...

**Example 2-approx:**

$j_0 = \{[0, 2], [4, 6]\}, j_1 = \{[1, 3]\}$

![Diagram showing job intervals and optimal solution]
Methods and results

Paper:

Approximation Algorithms for the Job Interval Selection Problem and Related Scheduling Problems

- presents three algorithms for general and special JISP instances
  - exp. 1.582-approximation for JISP
  - optimal pseudo-poly algorithm for constant relative window size
  - poly.-time \((1 - \epsilon)\)-approx for const. amount of processing times and const. proc. time ratio

- improves approximation factor of 2 for arbitrary instances

- uses among other techniques dynamic programming, partitioning and LP relaxation
The job interval scheduling problem

The paper: methods and results

1.582 Approximation for arbitrary instances
- First phase
  - First iteration
  - i-th iteration
  - last iteration
- Second phase
  - ILP Formulation
  - ILP Relaxation Rounding
- Main Result

Further results
- Dynamic programming algorithm
- Poly. approx. scheme
Algorithm overview

input
- set $S$ of unweighted jobs
- $\epsilon > 0$

Procedure:
- First phase: schedule jobs by using GREEDY as subroutine
- Second phase: schedule jobs by using ILP relaxation

Output:

\[ 1.582 > \left( \frac{e}{e-1} + \epsilon \right) \text{-approx to opt. sol. } OPT(S) \text{ (in expectation)} \]
Main Theorem

**Theorem**

For all $\epsilon > 0$ we get a polynomial $\frac{e}{(e-1)} + \epsilon$-approximation in expectation for JISP!

$$E[|S^I| + |S^II|] \geq \left(\frac{e}{(e-1)} + \epsilon\right)^{-1}|OPT(S)|$$

- $S^I$: jobs scheduled first phase
- $S^II$: jobs scheduled second phase
- $\frac{e}{(e-1)} + \epsilon < 1.582$
Main Idea: First phase

Let $OPT$ refer to optimal schedule

1. identify **dense areas** with many jobs scheduled by $OPT$
2. use **GREEDY** to find **dense** areas
3. **partition** time line
4. fine partition $\leftrightarrow$ dense area
5. reschedule with **GREEDY**
6. **refine** partition per **iteration**
Notation

- $OPT(X)$: refers to optimal **schedule** given set of jobs $X$
- $OPT(X)$: also refers to **set** of scheduled jobs by $OPT(X)$
- $OPT_B(X)$: opt. schedule respecting **partition** $B$ of time line (no job crosses a boundary)
Notation

During iterations we make use of the following variables:

- $S_i \subseteq S_{i-1}$: set of jobs scheduled in iteration $i$ ($S_0 = S$)
- $B_i$: refined partition after iteration $i$
1 The job interval scheduling problem

2 The paper: methods and results

3 1.582 Approximation for arbitrary instances
   - First phase
     - First iteration
     - i-th iteration
     - last iteration
   - Second phase
     - ILP Formulation
     - ILP Relaxation Rounding
   - Main Result

4 Further results
   - Dynamic programming algorithm
   - Poly. approx. scheme
Phase 1: first iteration

**Given**: job set $S$, $\epsilon > 0$
Let $k = \lceil \frac{6}{\epsilon} \rceil$

1. run **GREEDY**: If machine is idle, schedule job which finishes first
2. create **partition** $B_1$: cuts after every $k^3$ scheduled jobs
Phase 1: first iteration: example

Say $k^3 = 2$ for simplicity

Say $k^3 = 2$ for simplicity
Phase 1: first iteration: Lemma 1.1

**Lemma (1.1)**

$$OPT_{B_1} \geq (1 - \frac{1}{k^3})OPT$$

→ opt. schedule resp. $B_1$ cannot get arbitrarily worse compared $OPT$

**Proof:**

- take the opt. schedule $OPT$
- consider $B_1$
- remove $j \in OPT$ which crosses $B_1$
- we get schedule $T \subset OPT$ which respects $B_1$
- $|OPT_{B_1}| \geq |T| \geq (1 - \frac{1}{k^3})|OPT|$
Phase 1: \(i\)-th iteration

**Input:** \(S_{i-1}, B_{i-1}\);

**Output:** \(S_i \subset S_{i-1}, B_i\) refining \(B_{i-1}\);

**foreach** block \(b \in B_{i-1}\) (along time line) **do**

- schedule \(b\) using \textbf{GREEDY} on \(S_{i-1}\);
- **if** \(b\) contains \(> k^{i+2}\) jobs **then**
  - refine \(b\) s.t.: at most \(k^{i+2}\) jobs per block;
- **else**
  - empty \(b\);
  - jobs in \(b\) become scheduable again;

**end**

**end**
Phase 1: $i$-th iteration: example

Current block $b_i$

**case:** $> k^{i+2}$ jobs scheduled

\[
= k^{i+2} + 1
\]
Phase 1: $i$-th iteration: example

Current block $b_i$

**case:** $> k^{i+2}$ jobs scheduled

\[ = k^{i+2} \]
$i$-th iteration: example

Current block $b_i$

**case:** $\leq k^{i+2}$ scheduled jobs

---

Daniel Bauer  
Approximation algorithms for the job interval scheduling problem  
January 31, 2023  23 / 55
$i$-th iteration: example

Current block $b_i$

**case:** $\leq k^{i+2}$ scheduled jobs

- empty block $b_i$
$i$-th iteration: example

Current block $b_i$

\textbf{case: } $\leq k^{i+2}$ scheduled jobs

- $j$ schedule label for next block $b_{i+1}$
$i$-th iteration: example

Current block $b_i$

**case:** $\leq k^{i+2}$ scheduled jobs

- $j$ schedulabel for next block $b_{i+1}$

![Diagram illustrating job scheduling](image-url)
Phase 1: $i$-th iteration: Lemma 2.1

Lemma (2.1)

$$OPT_{B_i} \geq (1 - \frac{i}{k^3})OPT$$

$OPT_{B_i}$ cannot get arbitrarily worse compared to $OPT$

- proof based on Lemma 1.1
Phase 1: last iteration

First breaking condition;
\[
\textbf{if } |S_i| \geq (1 - \frac{1}{k})|S_{i-1}| \textbf{ then}
\]
\[
\quad \text{set } B_i = B_{i-1};
\]
\[
\quad \text{set } S' = S_i;
\]
\[
\quad \text{set } S_{pass} = S \setminus S_{i-1};
\]
\[
\textbf{end}
\]

set \( r = i \) (last iteration);

- let \( r \) denote last iteration
- \( S_{pass} \): set jobs we pass second phase
Phase 1: last iteration

First breaking condition;

\[
\text{if } |S_i| \geq (1 - \frac{1}{k})|S_{i-1}| \text{ then }
\]

\[
\text{set } B_i = B_{i-1};
\]

\[
\text{set } S' = S_i;
\]

\[
\text{set } S_{pass} = S \setminus S_{i-1};
\]

end

Second breaking condition;

\[
\text{if } i = k \ln k + 1 \text{ then }
\]

\[
\text{set } S' = \emptyset;
\]

\[
\text{set } S_{pass} = S \setminus S_{k \ln k + 1};
\]

end

set \( r = i \) (last iteration);

- let \( r \) denote last iteration

- \( S_{pass} \): set jobs we pass second phase
Phase 1: last iteration: example

case: scheduled too many jobs

\[ B_{i-1} \quad B_i \]

- throw away \( B_i \)
Phase 1: last iteration: example

**case:** scheduled too many jobs

\[ B_{i-1} \quad B_i \]

- throw away \( B_i \)
- fix current schedule \( S_i \)
last iteration: example

case: too many iterations

\[ B_{i-1}, B_i \]

- keep \( B_i \)
last iteration: example

**case:** too many iterations

$B_{i-1}$  $B_i$

- keep $B_i$
- throwaway $S_i$
Phase 1: last iteration: Lemma 3.1

Lemma (3.1)
\[ |OPT_{Br}| \geq (1 - \frac{1}{k})|OPT| \]

Proof.

since \( r \leq k \ln k + 1 \)

\[ |OPT_{Br}| \geq (1 - \frac{k \ln k + 1}{k^3})|OPT| \]

\[ \geq (1 - \frac{1}{k})|OPT| \]
Summary of first phase

What we get:

- $B_r$: most refined partition of time line
- $S^I$ scheduled in $B_r$

What we pass:

- $B_{empty}$: set of empty blocks
- $S_{pass} = S \setminus S_r$: set of jobs we know are not densely scheduled
Summary of first phase

- **polynomial** running time
- densely scheduled areas $\leftrightarrow$ fine partition
- $S^I$ finished schedule first phase
- $S_{pass}$ yet to be scheduled

**Lemma**

$$|S^I| + |OPT_{B_{empty}}(S_{pass})| \geq (1 - \epsilon)|OPT|$$

**Proof.**

left as an exercise...
The job interval scheduling problem

The paper: methods and results

1.582 Approximation for arbitrary instances
- First phase
  - First iteration
  - i-th iteration
  - last iteration
- Second phase
  - ILP Formulation
  - ILP Relaxation Rounding
- Main Result

Further results
- Dynamic programming algorithm
- Poly. approx. scheme
Main Idea: Second phase

- **ILP** for $OPT_{B_{empty}}(S_{pass})$
- solve ILP **relaxation** with $y$
- **round** $y \rightarrow y_{int}$ feasible schedule
- analysis yields a 1.582-approximation in expectation
1. The job interval scheduling problem

2. The paper: methods and results

3. 1.582 Approximation for arbitrary instances
   - First phase
     - First iteration
     - i-th iteration
     - last iteration
   - Second phase
     - ILP Formulation
     - ILP Relaxation Rounding
   - Main Result

4. Further results
   - Dynamic programming algorithm
   - Poly. approx. scheme
Guarantees from first phase

Polynomial sized ILP description!

Because:

**Lemma**

\( \text{OPT schedules at most } 4k^k \ln k + 1 + 3 \text{ jobs from } S_{\text{pass}} \text{ into } b \in B_{\text{empty}} \)
Guarantees from first phase

Polynomial sized ILP description!

Because:

**Lemma**

\( \text{OPT schedules at most } 4k^k \ln k^{k+1+3} \text{ jobs from } S_{\text{pass}} \text{ into } b \in B_{\text{empty}} \)

- schedule described by **ordered set**
- i.e. schedule \( T \):
  - \( j_0 \) at time 2
  - \( j_1 \) at time 1
- \( T \leftrightarrow (j_1, j_0) \)
Example ordered set

\[ (j_1, j_0) \]

schedules in \( b \in B_{empty} \) \( \leq \) tuples (size \( \leq 4k^k \ln k + 1 + 3 \)) over \( S_{pass} \)
ILP formulation

- $M(b)$: set of **tuples** (size $\leq 4k^k \ln k + 1 + 3$) for block $b \in B_{empty}$
- $y \in \{0, 1\}^{\vert M(b)\vert \times \vert B_{empty}\vert}$
- $y^b_m = 1 \iff b$ scheduled as described by $m \in M(b)$
- relaxation: $y^b_m \in \{0, 1\} \to y^b_m \geq 0$
ILP formulation

\[
\begin{align*}
\text{max} & \quad \sum_{b \in B_{\text{empty}}} \sum_{m \in M(b)} \sum_{j \in m} y^b_m \\
\text{s.t.:} & \quad \sum_{b \in B_{\text{empty}}} \sum_{m \in M(b) \setminus j \in m} y^b_m \leq 1 \quad \forall j \in S_{\text{pass}} \\
& \quad \sum_{m \in M(b)} y^b_m = 1 \quad \forall b \in B_{\text{empty}} \\
& \quad y^b_m \in \{0, 1\} \quad \forall b \in B_{\text{empty}}, \forall m \in M(b)
\end{align*}
\]

- \( M(b) \): set of tuples (size \( \leq 4k^k\ln k + 1 + 3 \)) for block \( b \in B_{\text{empty}} \)
- \( y \in \{0, 1\}^{\left|M(b)\right| \times \left|B_{\text{empty}}\right|} \)
- \( y^b_m = 1 \iff b \text{ scheduled as described by } m \in M(b) \)
- relaxation: \( y^b_m \in \{0, 1\} \rightarrow y^b_m \geq 0 \)
1. The job interval scheduling problem

2. The paper: methods and results

3. 1.582 Approximation for arbitrary instances
   - First phase
     - First iteration
     - $i$-th iteration
     - Last iteration
   - Second phase
     - ILP Formulation
     - ILP Relaxation Rounding
   - Main Result

4. Further results
   - Dynamic programming algorithm
   - Poly. approx. scheme
Rounding

Let $y$ opt. for ILP relaxation
Rounding

Let $y$ opt. for ILP relaxation
round $y \rightarrow y_{int}$ feasible schedule
Rounding

Let $y$ opt. for ILP relaxation round $y \rightarrow y_{int}$ feasible schedule

```latex
\textbf{foreach} block $b \in B_{\text{empty}} \textbf{ do}
\quad \text{with probability } y^b_m \text{ pick } m \in M(b)
\textbf{end}
```
Rounding

Let \( y \) opt. for ILP relaxation
round \( y \rightarrow y_{\text{int}} \) feasible schedule

\[
\text{foreach block } b \in B_{\text{empty}} \text{ do}
\quad \text{with probability } y^b_m \text{ pick } m \in M(b)
\text{end}
\]

\[
\text{foreach } j \in S_{\text{pass}} \text{ do}
\quad \text{if more than one block contain } j \text{ then}
\quad \quad \text{arbitrarily remove } j \text{ from every block except one}
\quad \text{end}
\text{end}
\]
y_{m_1}^b = 0.1

m_1 = (j_0, j_1) \in M(b)

y_{m_2}^b = 0.9

m_2 = (j_0) \in M(b)

with prob. 0.9 pick \( m_2 \).
Rounding example

After rounding...

Remove abr. $j_1$ except one!
Rounding example

After rounding...

Remove abr. \( j_1 \) except one!
Rounding example

After rounding...

\[ j_1 \text{ appears not twice} \]

Remove abr. \( j_1 \) except one!

Result:

- jobs appear in only one block
- feasible schedule
Analysis

We have:

\[ y \leftarrow \text{opt. sol. of ILP Relaxation} \]

\[ z \text{ value of } y \]

\[ y \text{ int} \leftarrow \text{rounded } y \]

\[ y \text{ int} \text{ is our schedule } S \]

\[ z \leftarrow \text{val} = \text{\# } S \]

We observe:

\[ z \geq \text{\# OPT} - \text{\# empty } (S) \]

\[ E[z \text{ int}] \geq (1 - 1/e) z \geq (1 - 1/e) \text{\# OPT} - \text{\# empty } (S) \]

Proof: some calculus

\[ E[\text{\# } S] = E[z \text{ int}] \geq (1 - 1/e) \text{\# OPT} - \text{\# empty } (S) \]
Analysis

We have:

- \( y \leftarrow \text{opt. sol. of ILP Relaxation} \)
Analysis

We have:

- $y \leftarrow \text{opt. sol. of ILP Relaxation}$
- $z$ value of $y$

We observe:

\[ z \geq |\text{OPT}B_{\text{empty}}(S_{\text{pass}})| \]

\[ E[z_{\text{int}}] \geq (1 - \frac{1}{e})z \geq (1 - \frac{1}{e})|\text{OPT}B_{\text{empty}}(S_{\text{pass}})| \]

Proof: some calculus
Analysis

We have:

- $y \leftarrow \text{opt. sol. of ILP Relaxation}$
- $z$ value of $y$
- $y_{\text{int}} \leftarrow \text{rounded } y$
Analysis

We have:

- $y \leftarrow$ opt. sol. of ILP Relaxation
- $z$ value of $y$
- $y_{\text{int}} \leftarrow$ rounded $y$
- $y_{\text{int}}$ is our schedule $S''$ for $S_{\text{pass}}$

We observe:

- $z \geq |OPT_{empty}(S_{\text{pass}})|$

Proof: some calculus

$$E[|S''|] \geq (1 - 1/e) |OPT_{empty}(S_{\text{pass}})|$$
Analysis

We have:

- $y \leftarrow$ opt. sol. of ILP Relaxation
- $z$ value of $y$
- $y_{\text{int}} \leftarrow$ rounded $y$
- $y_{\text{int}}$ is our schedule $S^{ll}$ for $S_{\text{pass}}$
- $z_{\text{int}} (= |S^{ll}|)$ value of $y_{\text{int}}$
Analysis

We have:

- $y \leftarrow \text{opt. sol. of ILP Relaxation}$
- $z$ value of $y$
- $y_{int} \leftarrow \text{rounded } y$
- $y_{int}$ is our schedule $S''$ for $S_{pass}$
- $z_{int}(=|S''|)$ value of $y_{int}$

We observe:

- $z \geq \left| \text{OPT} \right|$ when $B$ is empty ($S_{pass}$)
- $E[z_{int}] \geq (1-\frac{1}{e})z \geq (1-\frac{1}{e})\left| \text{OPT} \right|$
**Analysis**

We have:

- $y \leftarrow \text{opt. sol. of ILP Relaxation}$
- $z$ value of $y$
- $y_{\text{int}} \leftarrow \text{rounded } y$
- $y_{\text{int}}$ is our schedule $S''$ for $S_{\text{pass}}$
- $z_{\text{int}} (= |S''|)$ value of $y_{\text{int}}$

We observe:

- $z \geq |OPT_{B_{\text{empty}}}(S_{\text{pass}})|$
Analysis

We have:

- $y \leftarrow$ opt. sol. of ILP Relaxation
- $z$ value of $y$
- $y_{int} \leftarrow$ rounded $y$
- $y_{int}$ is our schedule $S''$ for $S_{pass}$
- $z_{int} (= |S''|)$ value of $y_{int}$

We observe:

- $z \geq |OPT_{Bempty}(S_{pass})|$
- $E[z_{int}] \geq (1 - \frac{1}{e})z$
  - Proof: some calculus
Analysis

We have:

- $y \leftarrow \text{opt. sol. of ILP Relaxation}$
- $z$ value of $y$
- $y_{int} \leftarrow$ rounded $y$
- $y_{int}$ is our schedule $S''$ for $S_{pass}$
- $z_{int}(=|S''|)$ value of $y_{int}$

We observe:

- $z \geq |OPT_{B_{empty}}(S_{pass})|$
- $E[z_{int}] \geq (1 - \frac{1}{e})z$
  - Proof: some calculus

\[ \Rightarrow E[|S''|] = E[z_{int}] \geq (1 - \frac{1}{e})z \geq (1 - \frac{1}{e})|OPT_{B_{empty}}(S_{pass})| \]
Summary

First phase result:

Lemma

\[ |S^I| + |OPT_{B_{empty}}(S_{pass})| \geq (1 - \epsilon)|OPT| \]

Second phase result:

Lemma

\[ E[|S^{II}|] \geq \left(1 - \frac{1}{e}\right)|OPT_{B_{empty}}(S_{pass})| \]
The job interval scheduling problem

The paper: methods and results

1.582 Approximation for arbitrary instances

- First phase
  - First iteration
  - $i$-th iteration
  - last iteration

- Second phase
  - ILP Formulation
  - ILP Relaxation Rounding

- Main Result

Further results

- Dynamic programming algorithm
- Poly. approx. scheme
Main Result

**Theorem**

For all \( \epsilon > 0 \) we get a polynomial \( \frac{e}{e-1} + \epsilon \)-approximation in expectation for JISP!

\[
\frac{e}{e-1} + \epsilon \geq \frac{|OPT|}{E[|S'|+|S''|]}
\]

Proof:

- previous Lemmas for first and second phase
The job interval scheduling problem

The paper: methods and results

1.582 Approximation for arbitrary instances

- First phase
  - First iteration
  - $i$-th iteration
  - last iteration

- Second phase
  - ILP Formulation
  - ILP Relaxation Rounding

- Main Result

Further results

- Dynamic programming algorithm
- Poly. approx. scheme
Second algorithm: Dynamic programming

Problem:

- (weighted) JISP instances with constant relative window size
Second algorithm: Dynamic programming

Problem:

- (weighted) JISP instances with constant relative window size

Definition (Constant Relative window size)

\[ k = \max_{j \in S} \frac{d_j - r_j}{p_j} \in \mathbb{R} \]
Second algorithm: Dynamic programming

Problem:
- (weighted) JISP instances with constant relative window size

Definition (Constant Relative window size)

$$k = \max_{j \in S} \frac{d_j - r_j}{p_j} \in \mathbb{R}$$

Result:
- optimal solution (also for weighted instances)
- pseudo polynomial in $n = |S|$ and $T = \max_{j \in S} d_j$
- exponential in $\text{poly}(k)$
Dynamic program sketch

Dynamic Program table $D$ consists of entries $D(s, x, e, j, IN, OUT)$:

- $s \leq x < e$ integer points on time line
- job $j \in S$
- subsets $IN, OUT \subseteq S$

$D(s, x, e, j, IN, OUT)$ stores optimal sub-schedule on $[s, e]$
Dynamic program sketch

Let $a \in \text{IN}$ and $b \in \text{OUT}$:

- **IN**: release dates in $[0, s]$
- **OUT**: release date in $[s, e]$

Let $r_a$ and $r_b$ be release dates.

$D(s, x, e, j, \text{IN}, \text{OUT})$
Dynamic program sketch

Goal:

- reach opt. schedule in $D(0, 0, T, n, \emptyset, \emptyset)$
- iterate over $j = \{1, \ldots, j, \ldots, n\} = S$
- compute entries $D(s, x, e, j, IN, OUT)$ for every possible $s, x, e, IN, OUT$
Dynamic Program Sketch

How to compute $D(s, x, e, j, IN, OUT)$?

Iterate over all feasible placements $t \in [x, e)$ for time line

$D(s, x, e, j, IN, OUT)$

enumerate over all reasonable sets $E, F, G, H$
Dynamic Program Sketch

How to compute $D(s, x, e, j, IN, OUT)$?
Iterate over all feasible placements $t \in [x, e)$ for $j$
How to compute $D(s, x, e, j, IN, OUT)$?
Iterate over all feasible placements $t \in [x, e)$ for $j$

$$D(s, x, t, j - 1, E, F)$$

schedule: $j$ at $t$

$$D(t, t + pj, e, j - 1, G, H)$$
Dynamic Program Sketch

How to compute $D(s, x, e, j, IN, OUT)$?
Iterate over all feasible placements $t \in [x, e)$ for $j$

\[
D(s, x, t, j - 1, E, F) \quad \text{schedule: } j \text{ at } t \quad D(t, t+p_j, e, j - 1, G, H)
\]

- enumerate over all reasonable sets $E, F, G, H$
Run time?

run time: $O(n^{poly(k)} T^4)$

How do we get pseudo polynomial run time?
Run time?

run time: $\mathcal{O}(n^{\text{poly}(k)} T^4)$

How do we get pseudo polynomial run time?

- over $n$ jobs
Run time?

run time: $O(n^{\text{poly}(k)} T^4)$

How do we get pseudo polynomial run time?

- over $n$ jobs
- for all choices $s \leq x < e \in [0, T] \Rightarrow T^3$
Run time?

run time: $O(n^{poly(k)} T^4)$

How do we get pseudo polynomial run time?

- over $n$ jobs
- for all choices $s \leq x < e \in [0, T] \Rightarrow T^3$
- enumerating $IN, OUT$?
Run time?

run time: $O(n^{\text{poly}(k)} T^4)$

How do we get pseudo polynomial run time?

- over $n$ jobs
- for all choices $s \leq x < e \in [0, T] \Rightarrow T^3$
- enumerating $IN, OUT$?
- const. relative window size $\Rightarrow |IN|, |OUT| \leq k^2$
Run time?

run time: $O(n^{poly(k)} T^4)$

How do we get pseudo polynomial run time?

- over $n$ jobs
- for all choices $s \leq x < e \in [0, T] \Rightarrow T^3$
- enumerating $IN, OUT$?
- const. relative window size $\Rightarrow |IN|, |OUT| \leq k^2$
- enumeration of $IN, OUT$ exponential in $k$
The job interval scheduling problem

The paper: methods and results

1.582 Approximation for arbitrary instances

- First phase
  - First iteration
  - i-th iteration
  - last iteration

- Second phase
  - ILP Formulation
  - ILP Relaxation Rounding

- Main Result

Further results

- Dynamic programming algorithm
- Poly. approx. scheme
Third algorithm

- poly.time $\frac{1}{1-\epsilon}$-approximation (for weighted instances)
- assume processing time $p_i \in C = \{p, .., P\}$, $|C| = c$ constant
- $\frac{P}{p} \leq r$ ratio of max. min. bounded
Third algorithm

- poly.time $\frac{1}{1-\epsilon}$-approximation (for weighted instances)
- assume processing time $p_i \in C = \{p, .., P\}$, $|C| = c$ constant
- $\frac{P}{p} \leq r$ ratio of max. min. bounded

Main Ideas:
- construct rand. partition
Third algorithm

- poly.time $\frac{1}{1-\epsilon}$-approximation (for weighted instances)
- assume processing time $p_i \in C = \{p, \ldots, P\}, |C| = c$ constant
- $\frac{P}{p} \leq r$ ratio of max. min. bounded

**Main Ideas:**

- construct rand. partition
- generalise the partition $l$-times (randomised)
Third algorithm

- poly.time $\frac{1}{1-\epsilon}$-approximation (for weighted instances)
- assume processing time $p_i \in C = \{p, .., P\}, |C| = c$ constant
- $\frac{P}{p} \leq r$ ratio of max. min. bounded

Main Ideas:
- construct rand. partition
- generalise the partition $l$-times (randomised)
- create sub-schedules for each partition
Third algorithm

- poly.time $\frac{1}{1-\epsilon}$-approximation (for weighted instances)
- assume processing time $p_i \in C = \{p, .., P\}$, $|C| = c$ constant
- $\frac{P}{p} \leq r$ ratio of max. min. bounded

Main Ideas:

- construct rand. partition
- generalise the partition $l$-times (randomised)
- create sub-schedules for each partition
- readjust schedules to gain solution
Summary

What we learned today...

- **JISP** is a difficult problem (NP-complete)
- **GREEDY** has been best general approach
- **GREEDY** approximation factor of 2
- improved with 1.582-approximation factor in expectation
- sub-problems pseudo poly. solvable optimally

Thank you for listening!
Questions?