Minimizing the Number of Tardy Jobs with Uniform Processing Times on Parallel Machines

Klaus Heeger[1] Hendrik Molter[2]

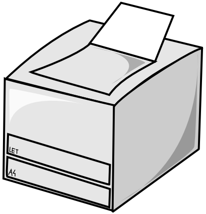
Ben-Gurion University of the Negev, Israel

[1]Department of Industrial Engineering and Management
[2]Department of Computer Science

STACS 2025

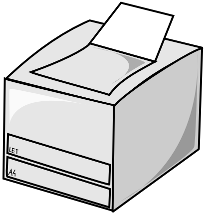
Supported by the European Union's Horizon Europe research and innovation program under grant agreement 949707.

Machines



Introduction to Scheduling

Machines

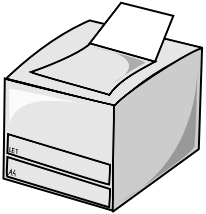


Jobs

document.pdf

Introduction to Scheduling

Machines



Jobs

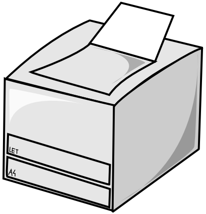
document.pdf

Objective

Provide a schedule that
prints all documents
as quickly as possible.

Introduction to Scheduling

Machines



Jobs

document.pdf

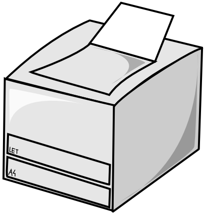
Objective

Provide a schedule that
prints all documents
as quickly as possible.

- Machines can have different speeds or availabilities.

Introduction to Scheduling

Machines



Jobs

document.pdf

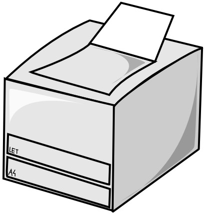
Objective

Provide a schedule that prints all documents as quickly as possible.

- Machines can have different speeds or availabilities.
- Jobs typically have processing times and sometimes weights, release dates, and due dates.

Introduction to Scheduling

Machines



Jobs

document.pdf

Objective

Provide a schedule that prints all documents as quickly as possible.

- Machines can have different speeds or availabilities.
- Jobs typically have processing times and sometimes weights, release dates, and due dates.
- Objective e.g.: print as many documents as possible before their respective deadlines.

Scheduling with Due Dates on a Single Machine

Machines

Single machine

Scheduling with Due Dates on a Single Machine

Machines	**Jobs**
Single machine	Processing time p, due date d

Scheduling with Due Dates on a Single Machine

Machines

Single machine

Jobs

Processing time p,
due date d

Objective

Minimize number of jobs that finish after due date

Scheduling with Due Dates on a Single Machine

Machines

Single machine

Jobs

Processing time p,
due date d

Objective

Minimize number of jobs that
finish after due date

Jobs: $\boxed{p = 2, d = 4}$ $\boxed{p = 4, d = 6}$

$\boxed{p = 2, d = 7}$ $\boxed{p = 3, d = 5}$

Scheduling with Due Dates on a Single Machine

Machines

Single machine

Jobs

Processing time p, due date d

Objective

Minimize number of jobs that finish after due date

Jobs: $p = 2, d = 4$ | $p = 4, d = 6$

$p = 2, d = 7$ | $p = 3, d = 5$

σ_1: $p = 2, d = 4$

Scheduling with Due Dates on a Single Machine

Machines

Single machine

Jobs

Processing time p, due date d

Objective

Minimize number of jobs that finish after due date

Jobs: $p = 2, d = 4$ $p = 4, d = 6$

$p = 2, d = 7$ $p = 3, d = 5$

σ_1: $p = 2, d = 4$ $p = 4, d = 6$

Scheduling with Due Dates on a Single Machine

Machines	**Jobs**	**Objective**
Single machine	Processing time p, due date d	Minimize number of jobs that finish after due date

Jobs: $p = 2, d = 4$ $p = 4, d = 6$

$p = 2, d = 7$ $p = 3, d = 5$

σ_1: $p = 2, d = 4$ $p = 4, d = 6$ $p = 2, d = 7$

Scheduling with Due Dates on a Single Machine

Machines	Jobs	Objective
Single machine	Processing time p, due date d	Minimize number of jobs that finish after due date

Jobs: $p = 2,\ d = 4$ | $p = 4,\ d = 6$

$p = 2,\ d = 7$ | $p = 3,\ d = 5$

σ_1: $p = 2,\ d = 4$ | $p = 4,\ d = 6$ | $p = 2,\ d = 7$ | $p = 3,\ d = 5$

Scheduling with Due Dates on a Single Machine

Machines

Single machine

Jobs

Processing time p,
due date d

Objective

Minimize number of jobs that
finish after due date

Jobs: $p = 2,\ d = 4$ | $p = 4,\ d = 6$

$p = 2,\ d = 7$ | $p = 3,\ d = 5$

σ_1: $p = 2,\ d = 4$ | $p = 4,\ d = 6$ | $p = 2,\ d = 7$ | $p = 3,\ d = 5$

σ_2: $p = 2,\ d = 4$ | $p = 3,\ d = 5$ | $p = 2,\ d = 7$ | $p = 4,\ d = 6$

Background: Most Important Known Results

Single Machine Scheduling with Due Dates

Solvable in polynomial time. [Moore '68]

Background: Most Important Known Results

Single Machine Scheduling with Due Dates

Solvable in polynomial time. [Moore '68]

Generalizations:

Background: Most Important Known Results

Single Machine Scheduling with Due Dates

Solvable in polynomial time. [Moore '68]

Generalizations:

Adding weights:

Background: Most Important Known Results

Single Machine Scheduling with Due Dates

Solvable in polynomial time. [Moore '68]

Generalizations:

Adding weights:

- Weakly NP-hard (Knapsack).

  [Karp '72]

Background: Most Important Known Results

Single Machine Scheduling with Due Dates

Solvable in polynomial time. [Moore ’68]

Generalizations:

Adding weights:

- Weakly NP-hard (Knapsack).
  [Karp ’72]
- Solvable in pseudo-polynomial time.
  [Lawler & Moore ’69]

Background: Most Important Known Results

Single Machine Scheduling with Due Dates

Solvable in polynomial time. [Moore '68]

Generalizations:

Adding weights:

- Weakly NP-hard (Knapsack).
  [Karp '72]
- Solvable in pseudo-polynomial time.
  [Lawler & Moore '69]

Adding more machines:

Background: Most Important Known Results

Single Machine Scheduling with Due Dates

Solvable in polynomial time. [Moore ’68]

Generalizations:

Adding weights:

- Weakly NP-hard (Knapsack).
  [Karp ’72]
- Solvable in pseudo-polynomial time.
  [Lawler & Moore ’69]

Adding more machines:

- Weakly NP-hard for two machines (Partition).
  [Karp ’72]

Background: Most Important Known Results

Single Machine Scheduling with Due Dates

Solvable in polynomial time. [Moore '68]

Generalizations:

Adding weights:

- Weakly NP-hard (Knapsack).
  [Karp '72]
- Solvable in pseudo-polynomial time.
  [Lawler & Moore '69]

Adding more machines:

- Weakly NP-hard for two machines (Partition).
  [Karp '72]
- Strongly NP-hard for many machines (Bin Packing).
  [Garey & Johnson '79]

Background: Most Important Known Results

Single Machine Scheduling with Due Dates

Solvable in polynomial time. [Moore '68]

Generalizations:

Adding weights:

- Weakly NP-hard (Knapsack).
  [Karp '72]
- Solvable in pseudo-polynomial time.
  [Lawler & Moore '69]

Adding more machines:

- Weakly NP-hard for two machines (Partition).
  [Karp '72]
- Strongly NP-hard for many machines (Bin Packing).
  [Garey & Johnson '79]

Adding release dates:

Background: Most Important Known Results

Single Machine Scheduling with Due Dates

Solvable in polynomial time. [Moore ’68]

Generalizations:

Adding weights:

- Weakly NP-hard (Knapsack).
  [Karp ’72]
- Solvable in pseudo-polynomial time.
  [Lawler & Moore ’69]

Adding more machines:

- Weakly NP-hard for two machines (Partition).
  [Karp ’72]
- Strongly NP-hard for many machines (Bin Packing).
  [Garey & Johnson ’79]

Adding release dates:

- Strongly NP-hard (Bin Packing).
  [Lenstra et al. ’77]

Our Setting

Machines

Parallel identical machines

Our Setting

Machines

Parallel identical machines

Jobs

Processing time p,
due date d, release date r,
weight w.

Our Setting

Machines

Parallel identical machines

Jobs

Processing time p, due date d, release date r, weight w.

Objective

Minimize weighted number of jobs that finish after due date

Our Setting

Machines	Jobs	Objective
Parallel identical machines	Processing time p, due date d, release date r, weight w.	Minimize weighted number of jobs that finish after due date

All jobs have the same processing time!

Our Setting

Machines

Parallel identical machines

Jobs

Processing time p,
due date d, release date r,
weight w.

Objective

Minimize weighted number of jobs that finish after due date

All jobs have the same processing time!

Jobs ($p = 3$): $r = 0, d = 7$ | $r = 1, d = 6$ | $r = 4, d = 7$ | $r = 3, d = 6$

Our Setting

Machines	Jobs	Objective
Parallel identical machines	Processing time p, due date d, release date r, weight w.	Minimize weighted number of jobs that finish after due date

All jobs have the same processing time!

Jobs ($p = 3$): | $r = 0$, $d = 7$ | $r = 1$, $d = 6$ | $r = 4$, $d = 7$ | $r = 3$, $d = 6$ |

Machine 1:

Machine 2:

Our Setting

Machines

Parallel identical machines

Jobs

Processing time p, due date d, release date r, weight w.

Objective

Minimize weighted number of jobs that finish after due date

All jobs have the same processing time!

Jobs ($p = 3$): $r = 0, d = 7$ | $r = 1, d = 6$ | $r = 4, d = 7$ | $r = 3, d = 6$

Machine 1: $r = 0, d = 7$

Machine 2:

Our Setting

Machines	Jobs	Objective
Parallel identical machines	Processing time p, due date d, release date r, weight w.	Minimize weighted number of jobs that finish after due date

All jobs have the same processing time!

Jobs ($p = 3$): $r = 0$, $d = 7$ | $r = 1$, $d = 6$ | $r = 4$, $d = 7$ | $r = 3$, $d = 6$

Machine 1: $r = 0$, $d = 7$

Machine 2: idle | $r = 1$, $d = 6$

Our Setting

Machines	Jobs	Objective
Parallel identical machines	Processing time p, due date d, release date r, weight w.	Minimize weighted number of jobs that finish after due date

All jobs have the same processing time!

Jobs ($p = 3$): $r = 0, d = 7$ | $r = 1, d = 6$ | $r = 4, d = 7$ | $r = 3, d = 6$

Machine 1: $r = 0, d = 7$ | idle | $r = 4, d = 7$

Machine 2: idle | $r = 1, d = 6$

Our Setting

Machines	Jobs	Objective
Parallel identical machines	Processing time p, due date d, release date r, weight w.	Minimize weighted number of jobs that finish after due date

All jobs have the same processing time!

Jobs ($p = 3$): $r = 0, d = 7$ | $r = 1, d = 6$ | $r = 4, d = 7$ | $r = 3, d = 6$

Machine 1: $r = 0, d = 7$ | idle | $r = 4, d = 7$

Machine 2: idle | $r = 1, d = 6$ | $r = 3, d = 6$

Our Setting

Machines

Parallel identical machines

Jobs

Processing time p, due date d, release date r, weight w.

Objective

Minimize weighted number of jobs that finish after due date

All jobs have the same processing time!

Jobs ($p = 3$): $r = 0, d = 7$ | $r = 1, d = 6$ | $r = 4, d = 7$ | $r = 3, d = 6$

Machine 1: $r = 0, d = 7$ | $r = 3, d = 6$

Machine 2: idle | $r = 1, d = 6$ | $r = 4, d = 7$

Our Setting

Machines

Parallel identical machines

Jobs

Processing time p, due date d, release date r, weight w.

Objective

Minimize weighted number of jobs that finish after due date

All jobs have the same processing time!

Jobs ($p = 3$): $r = 0, d = 7$ | $r = 1, d = 6$ | $r = 4, d = 7$ | $r = 3, d = 6$

Machine 1: $r = 0, d = 7$ | $r = 3, d = 6$

Machine 2: idle | $r = 1, d = 6$ | $r = 4, d = 7$

Minimizing the Weighted Number of Tardy Jobs with Uniform Processing Times on Parallel Machines

Our Setting

Machines

Parallel identical machines

Jobs

Processing time p, due date d, release date r, weight w.

Objective

Minimize weighted number of jobs that finish after due date

All jobs have the same processing time!

Jobs ($p = 3$): $r = 0, d = 7$ | $r = 1, d = 6$ | $r = 4, d = 7$ | $r = 3, d = 6$

Machine 1: $r = 0, d = 7$ | $r = 3, d = 6$

Machine 2: idle | $r = 1, d = 6$ | $r = 4, d = 7$

Minimizing the Weighted Number of Tardy Jobs with Uniform Processing Times on Parallel Machines

“Our Scheduling Problem”

Our Scheduling Problem: Motivation

Manufacturing processes, where:

- Exact specifications have negligible effect on production time.

Our Scheduling Problem: Motivation

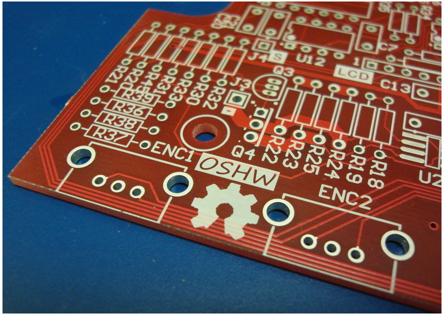
Manufacturing processes, where:

- Exact specifications have negligible effect on production time.
- Specifications only become available at certain times.

Manufacturing processes, where:

- Exact specifications have negligible effect on production time.
- Specifications only become available at certain times.

Examples:



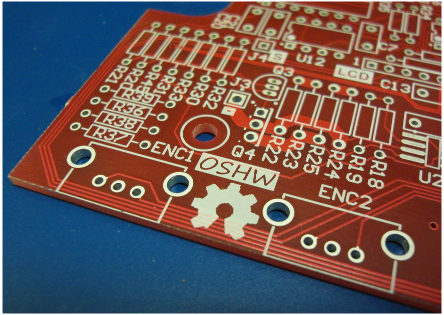
Etching of PCBs.

Our Scheduling Problem: Motivation

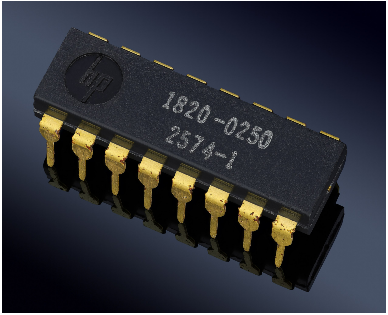
Manufacturing processes, where:

- Exact specifications have negligible effect on production time.
- Specifications only become available at certain times.

Examples:



Etching of PCBs.



Burn-in of ICs.

Known Results

Theorem [Babtiste et al. ’00, ’04]

Our Scheduling Problem can be solved in $n^{O(m)}$ time, where n is the number of jobs and m is the number of machines.

Known Results

Theorem [Babtiste et al. '00, '04]

Our Scheduling Problem can be solved in $n^{O(m)}$ time, where n is the number of jobs and m is the number of machines.

NP-hardness vs. polynomial-time solvability: **Open!**

Known Results

Theorem [Babtiste et al. '00, '04]

Our Scheduling Problem can be solved in $n^{O(m)}$ time, where n is the number of jobs and m is the number of machines.

NP-hardness vs. polynomial-time solvability: **Open!**

J Sched (2011) 14:435–444
DOI 10.1007/s10951-011-0231-3

Parallel machine problems with equal processing times: a survey

Svetlana A. Kravchenko · Frank Werner

by Kravchenko and Werner [2011]

2.1.19 Note that currently the most interesting open problems are $P \mid r_j, p_j = p \mid \sum U_j$ and $P \mid r_j, p_j = p, D_j \mid \sum w_j C_j$.

Known Results

Theorem [Babtiste et al. ’00, ’04]

Our Scheduling Problem can be solved in $n^{O(m)}$ time, where n is the number of jobs and m is the number of machines.

NP-hardness vs. polynomial-time solvability: **Open!**

Open Problems in Throughput Scheduling

Jiří Sgall*

Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Malostranské nám. 25, CZ-11800 Praha 1, Czech Republic
sgall@iuuk.mff.cuni.cz

Abstract. In this talk we survey the area of scheduling with the objective to maximize the throughput, i.e., the (weighted) number of completed jobs. We focus on several open problems.

by Sgall [2012]

In case of equal length jobs, the problem of maximizing the number of scheduled jobs is still far from trivial. On a single machine it is polynomial [3,7], but on parallel machines, it is known to be polynomial only if the number of machines m is a constant [1]. The following is still open:

Open problem 2. *Consider scheduling of equal-length jobs on m parallel machines with m a part of the input. Is there a polynomial-time algorithm to compute a schedule maximizing the number of completed jobs? Is it NP-hard to compute the maximal weight of scheduled jobs for some?*

Known Results

Theorem [Babtiste et al. ’00, ’04]

Our Scheduling Problem can be solved in $n^{O(m)}$ time, where n is the number of jobs and m is the number of machines.

NP-hardness vs. polynomial-time solvability: **Open!**

ELSEVIER

Computers & Operations Research
Volume 100, December 2018, Pages 254-261

Survey in Operations Research and Management Science

Parameterized complexity of machine scheduling: 15 open problems

Matthias Mnich [a, b], René van Bevern [c, d]

by Mnich and van Bevern [2018]

Baptiste et al. (2004) showed that $Pm|r_j, p_j{=}p|\Sigma w_j U_j$ is polynomial-time solvable. According to Sgall (2012), this is open when the number of machines is not a constant. A first step to resolving this question is solving the following problem.

Open Problem 7. *Are $P|r_j, p_j{=}p|\Sigma w_j U_j$ and $P|r_j, p_j{=}p|\Sigma U_j$ fixed-parameter tractable parameterized by the number of machines?*

A negative answer to this question would also be interesting since it is open whether these problems are even NP-hard if the number m of machines is part of the input. Notably, the special

Reminder: Parameterized Algorithms and Complexity

Parameterized Problem

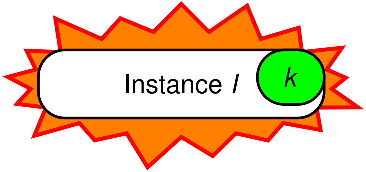
Each instance I is associated with a (small) parameter k.

Reminder: Parameterized Algorithms and Complexity

Parameterized Problem

Each instance I is associated with a (small) parameter k.

Classical worst-case running time:

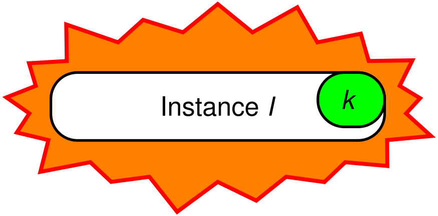


Reminder: Parameterized Algorithms and Complexity

Parameterized Problem

Each instance I is associated with a (small) parameter k.

Classical worst-case running time:

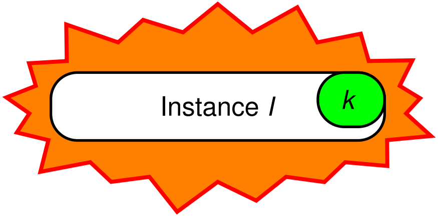


Reminder: Parameterized Algorithms and Complexity

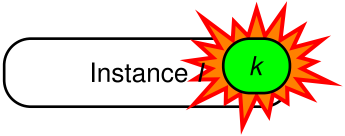
Parameterized Problem

Each instance I is associated with a (small) parameter k.

Classical worst-case running time:



Fixed-parameter tractability (FPT):

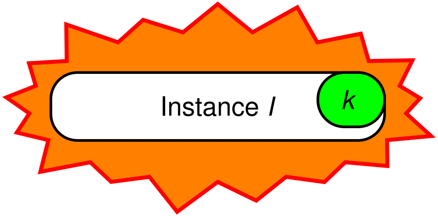


Reminder: Parameterized Algorithms and Complexity

Parameterized Problem

Each instance I is associated with a (small) parameter k.

Classical worst-case running time:



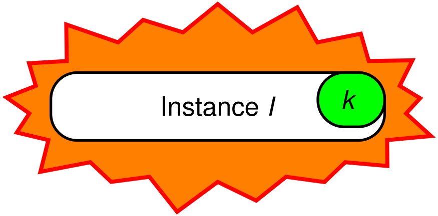
Fixed-parameter tractability (FPT):

Reminder: Parameterized Algorithms and Complexity

Parameterized Problem

Each instance I is associated with a (small) parameter k.

Classical worst-case running time:



Fixed-parameter tractability (FPT):

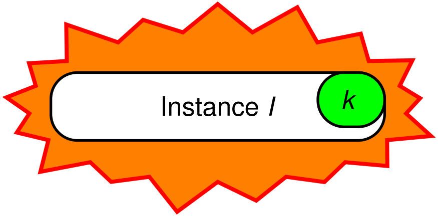
Running time: $f(k) \cdot |I|^{O(1)}$.

Reminder: Parameterized Algorithms and Complexity

Parameterized Problem

Each instance I is associated with a (small) parameter k.

Classical worst-case running time:



Fixed-parameter tractability (FPT):

Running time: $f(k) \cdot |I|^{O(1)}$.

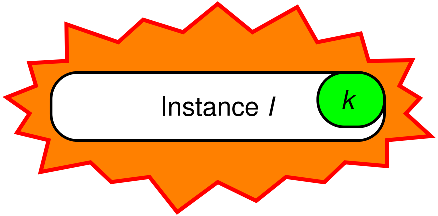
XP: Running time: $|I|^{f(k)}$.

Reminder: Parameterized Algorithms and Complexity

Parameterized Problem

Each instance I is associated with a (small) parameter k.

Classical worst-case running time:



Fixed-parameter tractability (FPT):

Running time: $f(k) \cdot |I|^{O(1)}$.

XP: Running time: $|I|^{f(k)}$.

Parameterized Hardness: W[1]-hard or W[2]-hard $\Rightarrow$ presumably not in FPT.

Our Results

Theorem 1 (Main Result)

Our Scheduling Problem is (strongly) **NP-hard** and **W[2]-hard** when parameterized by the number m of machines, even if all jobs have the same weight.

Our Results

Theorem 1 (Main Result)

Our Scheduling Problem is (strongly) **NP-hard** and **W[2]-hard** when parameterized by the number m of machines, even if all jobs have the same weight.

Theorem 2

Our Scheduling Problem in **XP** when parameterized by the processing time p and in **FPT** when parameterized by the combination of p and m.

Our Results

Theorem 1 (Main Result)

Our Scheduling Problem is (strongly) **NP-hard** and **W[2]-hard** when parameterized by the number m of machines, even if all jobs have the same weight.

Theorem 2

Our Scheduling Problem in **XP** when parameterized by the processing time p and in **FPT** when parameterized by the combination of p and m.

We obtain this by giving alternative running time analyses of the algorithm by Baptiste et al.

Our Results

Theorem 1 (Main Result)

Our Scheduling Problem is (strongly) **NP-hard** and **W[2]-hard** when parameterized by the number m of machines, even if all jobs have the same weight.

Theorem 2

Our Scheduling Problem in **XP** when parameterized by the processing time p and in **FPT** when parameterized by the combination of p and m.

We obtain this by giving alternative running time analyses of the algorithm by Baptiste et al.

Theorem 3

Our Scheduling Problem in **FPT** when parameterized by number of *different* release dates or the number of *different* due dates.

Our Results

Theorem 1 (Main Result)

Our Scheduling Problem is (strongly) **NP-hard** and **W[2]-hard** when parameterized by the number m of machines, even if all jobs have the same weight.

Theorem 2

Our Scheduling Problem in **XP** when parameterized by the processing time p and in **FPT** when parameterized by the combination of p and m.

We obtain this by giving alternative running time analyses of the algorithm by Baptiste et al.

Theorem 3

Our Scheduling Problem in **FPT** when parameterized by number of *different* release dates or the number of *different* due dates.

We obtain this by giving an appropriate MILP formulation of the problem.

Hardness of Our Scheduling Problem I

Reduction from **Hitting Set**.

Hardness of Our Scheduling Problem I

Reduction from **Hitting Set**.

Hitting Set

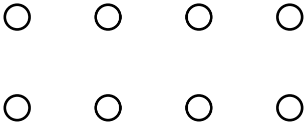
Input: A universe $U = \{u_1, u_2, \ldots, u_n\}$, a family $\mathscr{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of U, and an integer k.

Hardness of Our Scheduling Problem I

Reduction from **Hitting Set**.

Hitting Set

Input: A universe $U = \{u_1, u_2, \ldots, u_n\}$, a family $\mathscr{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of U, and an integer k.

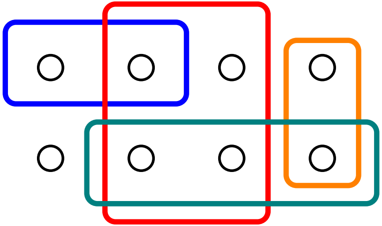


Hardness of Our Scheduling Problem I

Reduction from **Hitting Set**.

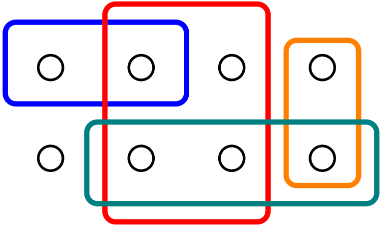
Hitting Set

Input: A universe $U = \{u_1, u_2, \ldots, u_n\}$, a family $\mathscr{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of U, and an integer k.



Hardness of Our Scheduling Problem I

Reduction from **Hitting Set**.



Hitting Set

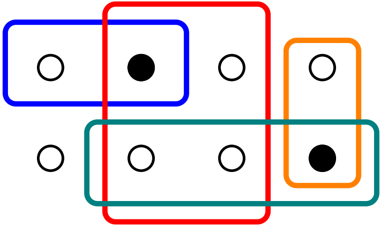
Input: A universe $U = \{u_1, u_2, \ldots, u_n\}$, a family $\mathscr{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of U, and an integer k.
Question: Is there a set $X \subseteq U$ with $|X| \leq k$ such that for all $i \in [m]$: $S_i \cap X \neq \emptyset$.

Hardness of Our Scheduling Problem I

Reduction from **Hitting Set**.

Hitting Set

Input: A universe $U = \{u_1, u_2, \ldots, u_n\}$, a family $\mathscr{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of U, and an integer k.
Question: Is there a set $X \subseteq U$ with $|X| \leq k$ such that for all $i \in [m]$: $S_i \cap X \neq \emptyset$.

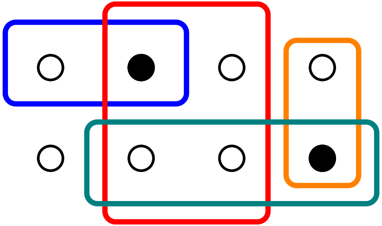


Hardness of Our Scheduling Problem I

Reduction from **Hitting Set**.

Hitting Set

Input: A universe $U = \{u_1, u_2, \ldots, u_n\}$, a family $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of U, and an integer k.
Question: Is there a set $X \subseteq U$ with $|X| \leq k$ such that for all $i \in [m]$: $S_i \cap X \neq \emptyset$.



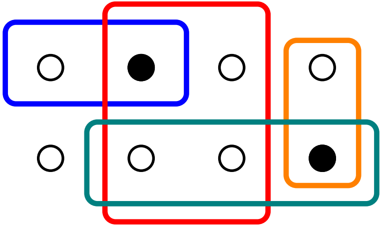
Main Idea:

Hardness of Our Scheduling Problem I

Reduction from **Hitting Set**.

Hitting Set

Input: A universe $U = \{u_1, u_2, \dots, u_n\}$, a family $\mathscr{S} = \{S_1, S_2, \dots, S_m\}$ of subsets of U, and an integer k.
Question: Is there a set $X \subseteq U$ with $|X| \leq k$ such that for all $i \in [m]$: $S_i \cap X \neq \emptyset$.



Main Idea:

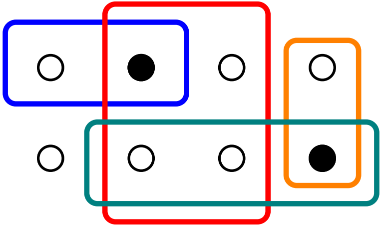
- Use k machines, set $p = 2n$.

Hardness of Our Scheduling Problem I

Reduction from **Hitting Set**.

Hitting Set

Input: A universe $U = \{u_1, u_2, \ldots, u_n\}$, a family $\mathscr{S} = \{S_1, S_2, \ldots, S_m\}$ of subsets of U, and an integer k.
Question: Is there a set $X \subseteq U$ with $|X| \leq k$ such that for all $i \in [m]$: $S_i \cap X \neq \emptyset$.



Main Idea:

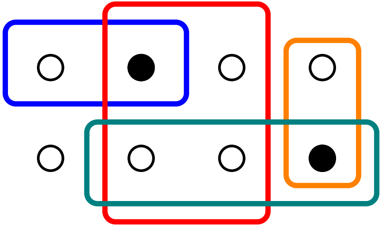
- Use k machines, set $p = 2n$.
- We want to enforce that each machine only has idle time at the beginning.
  The idle time encodes an element of U that is selected for the set cover.

Hardness of Our Scheduling Problem I

Reduction from **Hitting Set**.

Hitting Set

Input: A universe $U = \{u_1, u_2, \dots, u_n\}$, a family $\mathscr{S} = \{S_1, S_2, \dots, S_m\}$ of subsets of U, and an integer k.
Question: Is there a set $X \subseteq U$ with $|X| \leq k$ such that for all $i \in [m]$: $S_i \cap X \neq \emptyset$.



Main Idea:

- Use k machines, set $p = 2n$.
- We want to enforce that each machine only has idle time at the beginning. The idle time encodes an element of U that is selected for the set cover.
- For each set S_i we create $|S_i|$ "element jobs" and $k-1$ "dummy jobs" jobs. We want to schedule k jobs for each set S_i.

Hardness of Our Scheduling Problem II

$$U = \{u_1, u_2, u_3, u_4, u_5, u_6\},\ S_1 = \{u_1, u_2, u_3\},\ S_2 = \{u_3, u_4, u_5\},\ S_3 = \{u_1, u_5, u_6\},\ k = 2$$

Hardness of Our Scheduling Problem II

$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

- For each set S_i, there is a “region” in the processing time.

Hardness of Our Scheduling Problem II

$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

- For each set S_i, there is a “region” in the processing time.

Hardness of Our Scheduling Problem II

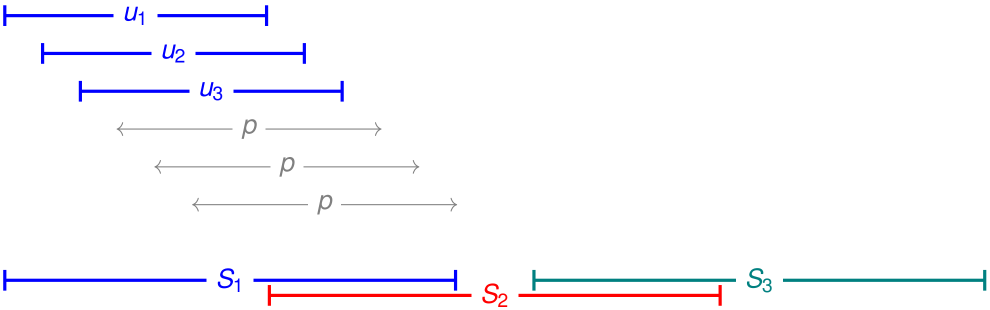
$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

- For each set S_i, there is a “region” in the processing time.
- The element job can only be placed in a specific interval. It creates no idle time if beginning idle time encodes element.

Hardness of Our Scheduling Problem II

$$U = \{u_1, u_2, u_3, u_4, u_5, u_6\},\ S_1 = \{u_1, u_2, u_3\},\ S_2 = \{u_3, u_4, u_5\},\ S_3 = \{u_1, u_5, u_6\},\ k = 2$$

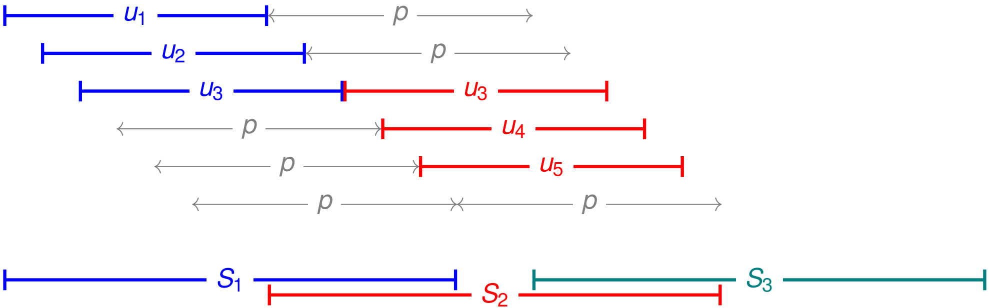
- For each set S_i, there is a "region" in the processing time.
- The element job can only be placed in a specific interval. It creates no idle time if beginning idle time encodes element.



Hardness of Our Scheduling Problem II

$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

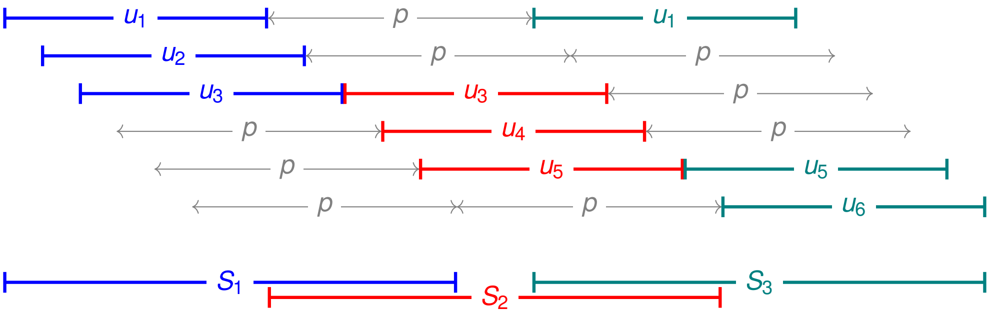
- For each set S_i, there is a “region” in the processing time.
- The element job can only be placed in a specific interval. It creates no idle time if beginning idle time encodes element.



Hardness of Our Scheduling Problem II

$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

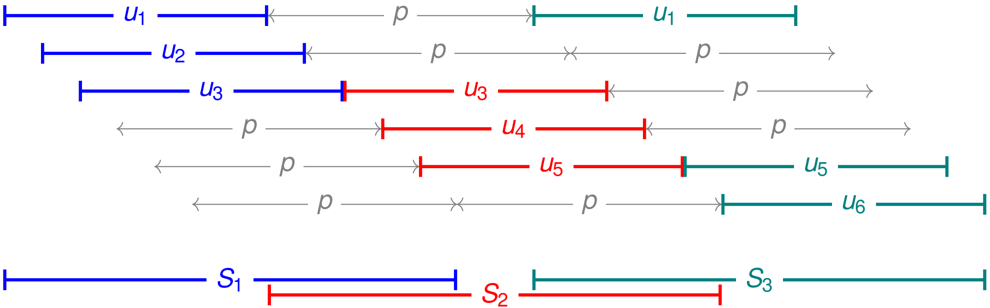
- For each set S_i, there is a “region” in the processing time.
- The element job can only be placed in a specific interval. It creates no idle time if beginning idle time encodes element.



Hardness of Our Scheduling Problem II

$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

- For each set S_i, there is a “region” in the processing time.
- The element job can only be placed in a specific interval. It creates no idle time if beginning idle time encodes element.
- Dummy jobs can be placed anywhere in the region.



Hardness of Our Scheduling Problem III

$$U = \{u_1, u_2, u_3, u_4, u_5, u_6\},\ S_1 = \{u_1, u_2, u_3\},\ S_2 = \{u_3, u_4, u_5\},\ S_3 = \{u_1, u_5, u_6\},\ k = 2$$

Hardness of Our Scheduling Problem III

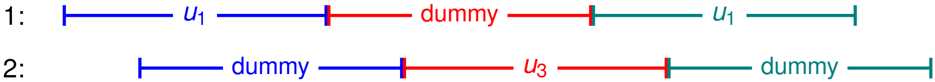
$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

Solution: $\{u_1, u_3\}$.

Hardness of Our Scheduling Problem III

$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

Solution: $\{u_1, u_3\}$.



Hardness of Our Scheduling Problem III

$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

Solution: $\{u_1, u_3\}$.

Problem: Index of initially selected element on a machine can be increased.

Hardness of Our Scheduling Problem III

$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

Solution: $\{u_1, u_3\}$.

Problem: Index of initially selected element on a machine can be increased.

Hardness of Our Scheduling Problem III

$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

Solution: $\{u_1, u_3\}$.

Problem: Index of initially selected element on a machine can be increased.

Observation:

Whenever “cheating” happens, the index of at least one encoded element is increased. This can only happen $k \cdot (n - 1)$ times.

Hardness of Our Scheduling Problem III

$U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$, $S_1 = \{u_1, u_2, u_3\}$, $S_2 = \{u_3, u_4, u_5\}$, $S_3 = \{u_1, u_5, u_6\}$, $k = 2$

Solution: $\{u_1, u_3\}$.

Problem: Index of initially selected element on a machine can be increased.

Observation:

Whenever “cheating” happens, the index of at least one encoded element is increased. This can only happen $k \cdot (n-1)$ times. $\Rightarrow$ Repeat sketched construction $k \cdot (n-1)+1$ times.

Conclusion

Summary:

We give an almost complete picture of the parameterized complexity of **Our Scheduling Problem** wrt.

- the processing time p,
- the nr. $w_\#$ of weights,
- the nr. $d_\#$ of due dates, and
- the nr. $r_\#$ of release dates.

Conclusion

Summary:

We give an almost complete picture of the parameterized complexity of **Our Scheduling Problem** wrt.

- the processing time p,
- the nr. $w_{\#}$ of weights,
- the nr. $d_{\#}$ of due dates, and
- the nr. $r_{\#}$ of release dates.

Open Question:

- Is **Our Scheduling Problem** in FPT when parameterized by the processing time p?

Conclusion

Summary:

We give an almost complete picture of the parameterized complexity of **Our Scheduling Problem** wrt.

- the processing time p,
- the nr. $w_\#$ of weights,
- the nr. $d_\#$ of due dates, and
- the nr. $r_\#$ of release dates.

Link to arXiv.

Open Question:

- Is **Our Scheduling Problem** in FPT when parameterized by the processing time p?

Conclusion

Summary:

We give an almost complete picture of the parameterized complexity of **Our Scheduling Problem** wrt.

- the processing time p,
- the nr. $w_\#$ of weights,
- the nr. $d_\#$ of due dates, and
- the nr. $r_\#$ of release dates.

Link to arXiv.

Open Question:

- Is **Our Scheduling Problem** in FPT when parameterized by the processing time p?

Thank you!