Optimization Algorithms

2. Structure

Thomas Weise · 汤卫思
tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn/5

Institute of Applied Optimization (IAO)
School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

应用优化研究所
人工智能与大数据学院
合肥学院
中国安徽省合肥市
Outline

1. Introduction
2. Example Problem: Job Shop Scheduling
3. Problem Instance
4. Solution Space
5. Objective Function
6. From Solution Space to Search Space
7. Number of Possible Solutions
8. Search Operators
9. Termination
10. Summary
Introduction
The Structure of Optimization

- So we know roughly what an optimization problem is and that metaheuristics\textsuperscript{1–4} are algorithms to solve them.
The Structure of Optimization

- So we know roughly what an optimization problem is and that metaheuristics\textsuperscript{1–4} are algorithms to solve them.
- But we do not really know yet how that works.
The Structure of Optimization

- So we know roughly what an optimization problem is and that metaheuristics\textsuperscript{1-4} are algorithms to solve them.
- But we do not really know yet how that works.
- We will approach this topic based on an example from the field of Smart Manufacturing.
The Structure of Optimization

• So we know roughly what an optimization problem is and that metaheuristics\textsuperscript{1–4} are algorithms to solve them.
• But we do not really know yet how that works.
• We will approach this topic based on an example from the field of Smart Manufacturing.
• We will first learn about the basic ingredients that make up an optimization task.
So we know roughly what an optimization problem is and that metaheuristics\textsuperscript{1–4} are algorithms to solve them.

But we do not really know yet how that works.

We will approach this topic based on an example from the field of Smart Manufacturing.

We will first learn about the basic ingredients that make up an optimization task.

Then we will step-by-step work our way from stupid to good metaheuristics for solving it.
Warnings

• This will be one of the tougher and probably the longest lesson in this lecture.
Warnings

• This will be one of the tougher and probably the longest lesson in this lecture.
• We will learn key ideas and concepts that apply to many different scenarios.
Warnings

• This will be one of the tougher and probably the longest lesson in this lecture.
• We will learn key ideas and concepts that apply to many different scenarios.
• We could look at them from an abstract point of view, similar to an abstract Maths class.
Warnings

• This will be one of the tougher and probably the longest lesson in this lecture.
• We will learn key ideas and concepts that apply to many different scenarios.
• We could look at them from an abstract point of view, similar to an abstract Maths class.
• Then this lesson would be short...
Warnings

• This will be one of the tougher and probably the longest lesson in this lecture.
• We will learn key ideas and concepts that apply to many different scenarios.
• We could look at them from an abstract point of view, similar to an abstract Maths class.
• Then this lesson would be short... ... but maybe you won’t get a very good feeling for the topic.
Warnings

- This will be one of the tougher and probably the longest lesson in this lecture.
- We will learn key ideas and concepts that apply to many different scenarios.
- We could look at them from an abstract point of view, similar to an abstract Maths class.
- Then this lesson would be short... but maybe you won’t get a very good feeling for the topic.
- Instead, we will directly take the abstract concepts and look how they are implemented on one concrete problem.
Warnings

• This will be one of the tougher and probably the longest lesson in this lecture.
• We will learn key ideas and concepts that apply to many different scenarios.
• We could look at them from an abstract point of view, similar to an abstract Maths class.
• Then this lesson would be short... but maybe you won’t get a very good feeling for the topic.
• Instead, we will directly take the abstract concepts and look how they are implemented on one concrete problem.
• This makes the lesson longer, but I hope it will provide for a better understanding.
Warnings

• This will be one of the tougher and probably the longest lesson in this lecture.
• We will learn key ideas and concepts that apply to many different scenarios.
• We could look at them from an abstract point of view, similar to an abstract Maths class.
• Then this lesson would be short... but maybe you won’t get a very good feeling for the topic.
• Instead, we will directly take the abstract concepts and look how they are implemented on one concrete problem.
• This makes the lesson longer, but I hope it will provide for a better understanding.
• The example we will use is just an example – the concepts can be implemented differently for almost all optimization problems.
Components of an Optimization Problem

- From the perspective of a programmer, we can say that an optimization problem has the following components.
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $I$ to be solved
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $I$ to be solved – we develop software for solving a class of problems, but this software is applied to specific problem instances, the actual scenarios
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $\mathcal{I}$ to be solved
  2. a data type $\mathcal{Y}$ for the candidate solutions $y \in \mathcal{Y}$, and
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance \( \mathcal{I} \) to be solved
  2. a data type \( \mathbb{Y} \) for the candidate solutions \( y \in \mathbb{Y} \), and
  3. an objective function \( f : \mathbb{Y} \mapsto \mathbb{R} \), which rates “how good” a candidate solution \( y \in \mathbb{Y} \) is.
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $\mathcal{I}$ to be solved
  2. a data type $\mathcal{Y}$ for the candidate solutions $y \in \mathcal{Y}$, and
  3. an objective function $f : \mathcal{Y} \mapsto \mathbb{R}$.

• Usually, in order to practically implement an optimization approach, there also will be
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $\mathcal{I}$ to be solved
  2. a data type $\mathbb{Y}$ for the candidate solutions $y \in \mathbb{Y}$, and
  3. an objective function $f : \mathbb{Y} \mapsto \mathbb{R}$.
• Usually, in order to **practically implement** an optimization approach, there also will be
  4. a search space $\mathbb{X}$, i.e., a simpler data structure for internal use, which can more efficiently be processed by an optimization algorithm than $\mathbb{Y}$
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $\mathcal{I}$ to be solved
  2. a data type $\mathbb{Y}$ for the candidate solutions $y \in \mathbb{Y}$, and
  3. an objective function $f : \mathbb{Y} \mapsto \mathbb{R}$.

• Usually, in order to practically implement an optimization approach, there also will be
  4. a search space $\mathbb{X}$,
  5. a representation mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$, which translates “points” $x \in \mathbb{X}$ to candidate solutions $y \in \mathbb{Y}$.
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $\mathcal{I}$ to be solved
  2. a data type $\mathbb{Y}$ for the candidate solutions $y \in \mathbb{Y}$, and
  3. an objective function $f : \mathbb{Y} \mapsto \mathbb{R}$.

• Usually, in order to practically implement an optimization approach, there also will be
  4. a search space $\mathbb{X}$,
  5. a representation mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$,
  6. search operators $\text{searchOp} : \mathbb{X}^n \mapsto \mathbb{X}$, which allow for the iterative exploration of the search space $\mathbb{X}$.
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $\mathcal{I}$ to be solved
  2. a data type $\mathbb{Y}$ for the candidate solutions $y \in \mathbb{Y}$, and
  3. an objective function $f : \mathbb{Y} \mapsto \mathbb{R}$.

• Usually, in order to **practically implement** an optimization approach, there also will be
  4. a search space $\mathbb{X}$,
  5. a representation mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$,
  6. search operators $\text{searchOp} : \mathbb{X}^n \mapsto \mathbb{X}$, and
  7. a termination criterion, which tells the optimization process when to stop.
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $I$ to be solved
  2. a data type $\mathbb{Y}$ for the candidate solutions $y \in \mathbb{Y}$, and
  3. an objective function $f : \mathbb{Y} \mapsto \mathbb{R}$.

• Usually, in order to practically implement an optimization approach, there also will be
  4. a search space $\mathbb{X}$,
  5. a representation mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$,
  6. search operators $\text{searchOp} : \mathbb{X}^n \mapsto \mathbb{X}$, and
  7. a termination criterion.

• Looks complicated..
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $\mathcal{I}$ to be solved
  2. a data type $\mathbb{Y}$ for the candidate solutions $y \in \mathbb{Y}$, and
  3. an objective function $f : \mathbb{Y} \mapsto \mathbb{R}$.

• Usually, in order to practically implement an optimization approach, there also will be
  4. a search space $\mathbb{X}$,
  5. a representation mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$,
  6. search operators $\text{searchOp} : \mathbb{X}^n \mapsto \mathbb{X}$, and
  7. a termination criterion.

• Looks complicated, but don’t worry.
Components of an Optimization Problem

• From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $I$ to be solved
  2. a data type $Y$ for the candidate solutions $y \in Y$, and
  3. an objective function $f : Y \mapsto \mathbb{R}$.

• Usually, in order to **practically implement** an optimization approach, there also will be
  4. a search space $X$,
  5. a representation mapping $\gamma : X \mapsto Y$,
  6. search operators $\text{searchOp} : X^n \mapsto X$, and
  7. a termination criterion.

• Looks complicated, but don’t worry. We will do this one-by-one.
Components of an Optimization Problem

- From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $I$ to be solved
  2. a data type $Y$ for the candidate solutions $y \in Y$, and
  3. an objective function $f : Y \mapsto \mathbb{R}$.
- Usually, in order to practically implement an optimization approach, there also will be
  4. a search space $X$,
  5. a representation mapping $\gamma : X \mapsto Y$,
  6. search operators $\text{searchOp} : X^n \mapsto X$, and
  7. a termination criterion.
- Looks complicated, but don’t worry. We will do this one-by-one.
- We want to get an understanding of the structure of optimization problems from the metaheuristic perspective by looking at one concrete problem from production planning.
Example Problem: Job Shop Scheduling
Job Shop Problem
Job Shop Problem
Job Shop Problem
Job Shop Problem
Job Shop Problem
Job Shop Problem
Job Shop Problem
Job Shop Problem
Job Shop Problem
Job Shop Problem
Job Shop Problem
Job Shop Problem
Job Shop Problem
The Job Shop Scheduling Problem (JSSP)\textsuperscript{5–9} is a classical optimization problem.
Job Shop Scheduling Problem

- The Job Shop Scheduling Problem (JSSP)$^{5-9}$ is a classical optimization problem.
- We have a factory with $m$ machines.
The Job Shop Scheduling Problem (JSSP)\(^{5-9}\) is a classical optimization problem.

- We have a factory with \(m\) machines.
- We need to fulfill \(n\) production requests, the jobs.
Job Shop Scheduling Problem

- The Job Shop Scheduling Problem (JSSP)\(^{5-9}\) is a classical optimization problem.
- We have a factory with \(m\) machines.
- We need to fulfill \(n\) production requests, the jobs.
- Each job will need to be processed by some or all of the machines in a job-specific order.
The Job Shop Scheduling Problem (JSSP) is a classical optimization problem. We have a factory with \( m \) machines. We need to fulfill \( n \) production requests, the jobs. Each job will need to be processed by some or all of the machines in a job-specific order. Also, each job will require a job-specific time at a given machine.
Job Shop Scheduling Problem

- The Job Shop Scheduling Problem (JSSP)\textsuperscript{5–9} is a classical optimization problem.
- We have a factory with $m$ machines.
- We need to fulfill $n$ production requests, the jobs.
- Each job will need to be processed by some or all of the machines in a job-specific order.
- Also, each job will require a job-specific time at a given machine.
- The goal is to fulfill all tasks as quickly as possible.
The Job Shop Scheduling Problem (JSSP)\textsuperscript{5–9} is a classical optimization problem.

- We have a factory with $m$ machines.
- We need to fulfill $n$ production requests, the jobs.
- Each job will need to be processed by some or all of the machines in a job-specific order.
- Also, each job will require a job-specific time at a given machine.
- The goal is to fulfill all tasks as quickly as possible.
- This scenario also encompasses simpler problems, e.g., where all jobs “are the same.”
The Job Shop Scheduling Problem (JSSP)\(^{5-9}\) is a classical optimization problem.

- We have a factory with \(m\) machines.
- We need to fulfill \(n\) production requests, the jobs.
- Each job will need to be processed by some or all of the machines in a job-specific order.
- Also, each job will require a job-specific time at a given machine.
- The goal is to fulfill all tasks as quickly as possible.
- This scenario also encompasses simpler problems, e.g., where all jobs “are the same.”
- This problem is \(\mathcal{NP}\)-hard.\(^{10}^{11}\)
What we will do

- In this course, we will use the JSSP as example domain.
What we will do

• In this course, we will use the JSSP as example domain.
• We will discuss all components of an optimization problem based on this example.
What we will do

- In this course, we will use the JSSP as example domain.
- We will discuss all components of an optimization problem based on this example.
- We will discuss several different optimization algorithms – and apply them to this problem.
What we will do

• In this course, we will use the JSSP as example domain.
• We will discuss all components of an optimization problem based on this example.
• We will discuss several different optimization algorithms – and apply them to this problem.
• But
What we will do

• In this course, we will use the JSSP as example domain.
• We will discuss all components of an optimization problem based on this example.
• We will discuss several different optimization algorithms – and apply them to this problem.
• But we will do this from an educational perspective
• We will not focus on the best possible data structures or highest possible efficiency.
What we will do

• In this course, we will use the JSSP as example domain.
• We will discuss all components of an optimization problem based on this example.
• We will discuss several different optimization algorithms – and apply them to this problem.
• But we will do this from an educational perspective
• We will not focus on the best possible data structures or highest possible efficiency.
• It needs years of research to get there...
What we will do

- In this course, we will use the JSSP as example domain.
- We will discuss all components of an optimization problem based on this example.
- We will discuss several different optimization algorithms – and apply them to this problem.
- **But** we will do this from an educational perspective
- We will **not** focus on the best possible data structures or highest possible efficiency.
- It needs years of research to get there...
- We will, instead, approach the JSSP in the same way you would approach a completely new problem domain
What we will do

• In this course, we will use the JSSP as example domain.
• We will discuss all components of an optimization problem based on this example.
• We will discuss several different optimization algorithms – and apply them to this problem.
• But we will do this from an educational perspective
• We will not focus on the best possible data structures or highest possible efficiency.
• It needs years of research to get there…
• We will, instead, approach the JSSP in the same way you would approach a completely new problem domain: develop a working approach
What we will do

• In this course, we will use the JSSP as example domain.
• We will discuss all components of an optimization problem based on this example.
• We will discuss several different optimization algorithms – and apply them to this problem.
• But we will do this from an educational perspective
• We will not focus on the best possible data structures or highest possible efficiency.
• It needs years of research to get there... 
• We will, instead, approach the JSSP in the same way you would approach a completely new problem domain: develop a working approach, test and compare different working approaches
What we will do

• In this course, we will use the JSSP as example domain.
• We will discuss all components of an optimization problem based on this example.
• We will discuss several different optimization algorithms – and apply them to this problem.
• But we will do this from an educational perspective
• We will not focus on the best possible data structures or highest possible efficiency.
• It needs years of research to get there . . .
• We will, instead, approach the JSSP in the same way you would approach a completely new problem domain: develop a working approach, test and compare different working approaches, (normally you would then improve them further, but we will skip this)
Problem Instance
The Input: Problem Instances

- The JSSP is a type of problem.
The Input: Problem Instances

- The JSSP is a type of problem.
- A concrete scenario, with a specific number of machines and with specific jobs, is called an instance $\mathcal{I}$. 
The Input: Problem Instances

- The JSSP is a type of problem.
- A concrete scenario, with a specific number of machines and with specific jobs, is called an instance $\mathcal{I}$.
- It is common in research that there collections of instances for a given problem, so that we can test algorithms and compare their performance (of course, you can only compare results if they are for the same scenario).
The Input: Problem Instances

• The JSSP is a type of problem.
• A concrete scenario, with a specific number of machines and with specific jobs, is called an instance $\mathcal{I}$.
• It is common in research that there collections of instances for a given problem, so that we can test algorithms and compare their performance (of course, you can only compare results if they are for the same scenario).
• Beasley$^{12}$ manages the OR Library of benchmark datasets from different fields of operations research (OR)
The Input: Problem Instances

• The JSSP is a type of problem.
• A concrete scenario, with a specific number of machines and with specific jobs, is called an instance $\mathcal{I}$.
• It is common in research that there collections of instances for a given problem, so that we can test algorithms and compare their performance (of course, you can only compare results if they are for the same scenario).
• Beasley\(^{12}\) manages the OR Library of benchmark datasets from different fields of operations research (OR)
• He also provides several example instances of the JSSP at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html.
The Input: Problem Instances

• The JSSP is a type of problem.
• A concrete scenario, with a specific number of machines and with specific jobs, is called an instance \( I \).
• It is common in research that there collections of instances for a given problem, so that we can test algorithms and compare their performance (of course, you can only compare results if they are for the same scenario).
• Beasley\(^1\) manages the OR Library of benchmark datasets from different fields of operations research (OR)
• He also provides several example instances of the JSSP at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html.
• More information about these instances has been collected by van Hoorn\(^2\) at http://jobshop.jjvh.nl.
The Input: Problem Instances

- The JSSP is a type of problem.
- A concrete scenario, with a specific number of machines and with specific jobs, is called an instance $\mathcal{I}$.
- It is common in research that there collections of instances for a given problem, so that we can test algorithms and compare their performance (of course, you can only compare results if they are for the same scenario).
- Beasley\textsuperscript{12} manages the OR Library of benchmark datasets from different fields of operations research (OR).
- He also provides several example instances of the JSSP at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html.
- More information about these instances has been collected by van Hoorn\textsuperscript{13,14} at http://jobshop.jjvh.nl.
- What do such JSSP instances look like?
## Demo Instance

A simple demo

```
4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15
```

```
# Demo Instance

<table>
<thead>
<tr>
<th>number $n$ of jobs</th>
<th>A simple demo</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5</td>
<td>0 10 1 20 2 20 3 40 4 10</td>
</tr>
<tr>
<td></td>
<td>1 20 0 10 3 30 2 50 4 30</td>
</tr>
<tr>
<td></td>
<td>2 30 1 20 4 12 3 40 0 10</td>
</tr>
<tr>
<td></td>
<td>4 50 3 30 2 15 0 20 1 15</td>
</tr>
<tr>
<td></td>
<td>0 20 1 15</td>
</tr>
</tbody>
</table>
**Demo Instance**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>0 10</td>
<td>1 20</td>
<td>2 20</td>
<td>3 40</td>
<td>4 10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0 10</td>
<td>3 30</td>
<td>2 50</td>
<td>4 30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1 20</td>
<td>4 12</td>
<td>3 40</td>
<td>0 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>3 30</td>
<td>2 15</td>
<td>0 20</td>
<td>1 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A simple demo
## Demo Instance

A simple demo

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>40</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>10</td>
<td>30</td>
<td>50</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>20</td>
<td>12</td>
<td>40</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>30</td>
<td>15</td>
<td>20</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

number $n$ of jobs

number $m$ of machines
Demo Instance

A simple demo

4 5

0 10 1 20 2 20 3 40 4 10

1 20 0 10 3 30 2 50 4 30

2 30 1 20 4 12 3 40 0 10

4 50 3 30 2 15 0 20 1 15

number $n$ of jobs

number $m$ of machines

job 1
## Demo Instance

<table>
<thead>
<tr>
<th>Job 0</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10</td>
<td>1 20</td>
<td>2 20</td>
<td>3 40</td>
</tr>
<tr>
<td>1 20</td>
<td>0 10</td>
<td>3 30</td>
<td>2 50</td>
</tr>
<tr>
<td>2 30</td>
<td>1 20</td>
<td>4 12</td>
<td>3 40</td>
</tr>
<tr>
<td>4 50</td>
<td>3 30</td>
<td>2 15</td>
<td>0 20</td>
</tr>
</tbody>
</table>

### Number $n$ of Jobs

### Number $m$ of Machines

---

A simple demo

---

---
## Demo Instance

A simple demo

<table>
<thead>
<tr>
<th>Job 0</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

#### Number of Jobs: 4
#### Number of Machines: 5

---

The table above represents a simple demo instance. Each row corresponds to a job, and each column corresponds to a machine. The values in the table indicate the processing times of jobs on different machines. For example, job 0 has processing times of 0, 10, 20, and 2 on machines 0, 1, 2, and 3, respectively, and a processing time of 4 on machine 4.
Demo Instance

<table>
<thead>
<tr>
<th>number n of jobs</th>
<th>number m of machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5</td>
<td>4 5</td>
</tr>
</tbody>
</table>

A simple demo

<table>
<thead>
<tr>
<th>job 0</th>
<th>job 1</th>
<th>job 2</th>
<th>job 3</th>
<th>job 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10</td>
<td>1 20</td>
<td>2 20</td>
<td>3 40</td>
<td>4 10</td>
</tr>
<tr>
<td>1 20</td>
<td>0 10</td>
<td>3 30</td>
<td>2 50</td>
<td>4 30</td>
</tr>
<tr>
<td>2 30</td>
<td>1 20</td>
<td>4 12</td>
<td>3 40</td>
<td>0 10</td>
</tr>
<tr>
<td>4 50</td>
<td>3 30</td>
<td>2 15</td>
<td>0 20</td>
<td>1 15</td>
</tr>
</tbody>
</table>

+++
Job 0 first needs to be processed by machine 0 for 10 time units.
Job 0 first needs to be processed by machine 0 for 10 time units, it then goes to machine 1 for 20 time units.
Job 0 first needs to be processed by machine 0 for 10 time units, it then goes to machine 1 for 20 time units, it then goes to machine 2 for 20 time units.
Job 0 first needs to be processed by machine 0 for 10 time units, it then goes to machine 1 for 20 time units, it then goes to machine 2 for 20 time units, it then goes to machine 3 for 40 time units.
Job 0 first needs to be processed by machine 0 for 10 time units, it then goes to machine 1 for 20 time units, it then goes to machine 2 for 20 time units, it then goes to machine 3 for 40 time units, and finally it goes to machine 4 for 10 time units.
Similarly, Job 1 first needs to be processed by machine 1 for 20 time units.
Similarly, Job 1 first needs to be processed by machine 1 for 20 time units, it then goes to machine 0 for 10 time units.
Similarly, Job 1 first needs to be processed by machine 1 for 20 time units, it then goes to machine 0 for 10 time units, it then goes to machine 3 for 30 time units.
Similarly, Job 1 first needs to be processed by machine 1 for 20 time units, it then goes to machine 0 for 10 time units, it then goes to machine 3 for 30 time units, it then goes to machine 2 for 50 time units.
Similarly, Job 1 first needs to be processed by machine 1 for 20 time units, it then goes to machine 0 for 10 time units, it then goes to machine 3 for 30 time units, it then goes to machine 2 for 50 time units, and finally it goes to machine 4 for 30 time units.
Job 2 first needs to be processed by machine 2 for 30 time units, it then goes to machine 1 for 20 time units, it then goes to machine 4 for 12 time units, it then goes to machine 3 for 40 time units, and finally it goes to machine 0 for 10 time units.
And Job 3 first needs to be processed by machine 4 for 50 time units, it then goes to machine 3 for 30 time units, it then goes to machine 2 for 15 time units, it then goes to machine 0 for 20 time units, and finally it goes to machine 1 for 15 time units.
Each of the \( n \) jobs has \( m \) operations, each consisting of a machine index and a time requirement.

A simple demo

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>120</td>
<td>220</td>
<td>340</td>
<td>410</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>010</td>
<td>330</td>
<td>250</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>120</td>
<td>412</td>
<td>340</td>
<td>010</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>330</td>
<td>215</td>
<td>020</td>
<td>115</td>
<td></td>
</tr>
</tbody>
</table>
### Instance abz7

Instance abz7 by Adams et al.¹⁵

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EOF
Instance 1a24

Instance 1a24 by Lawrence\textsuperscript{16}.

<table>
<thead>
<tr>
<th>7</th>
<th>8</th>
<th>9</th>
<th>75</th>
<th>0</th>
<th>72</th>
<th>6</th>
<th>74</th>
<th>4</th>
<th>30</th>
<th>8</th>
<th>43</th>
<th>2</th>
<th>38</th>
<th>5</th>
<th>98</th>
<th>1</th>
<th>26</th>
<th>3</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>19</td>
<td>8</td>
<td>73</td>
<td>3</td>
<td>43</td>
<td>0</td>
<td>23</td>
<td>1</td>
<td>85</td>
<td>4</td>
<td>39</td>
<td>5</td>
<td>13</td>
<td>9</td>
<td>26</td>
<td>2</td>
<td>67</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>3</td>
<td>93</td>
<td>5</td>
<td>80</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>55</td>
<td>2</td>
<td>61</td>
<td>6</td>
<td>57</td>
<td>8</td>
<td>72</td>
<td>9</td>
<td>42</td>
<td>7</td>
<td>46</td>
</tr>
<tr>
<td>1</td>
<td>68</td>
<td>7</td>
<td>43</td>
<td>4</td>
<td>99</td>
<td>6</td>
<td>60</td>
<td>5</td>
<td>68</td>
<td>0</td>
<td>91</td>
<td>8</td>
<td>11</td>
<td>3</td>
<td>96</td>
<td>9</td>
<td>11</td>
<td>2</td>
<td>72</td>
</tr>
<tr>
<td>7</td>
<td>84</td>
<td>2</td>
<td>34</td>
<td>8</td>
<td>40</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>70</td>
<td>6</td>
<td>74</td>
<td>3</td>
<td>12</td>
<td>0</td>
<td>43</td>
<td>9</td>
<td>69</td>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
<td>0</td>
<td>49</td>
<td>4</td>
<td>59</td>
<td>5</td>
<td>72</td>
<td>9</td>
<td>63</td>
<td>1</td>
<td>69</td>
<td>7</td>
<td>99</td>
<td>6</td>
<td>45</td>
<td>3</td>
<td>27</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>71</td>
<td>2</td>
<td>91</td>
<td>8</td>
<td>65</td>
<td>1</td>
<td>90</td>
<td>9</td>
<td>98</td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>50</td>
<td>0</td>
<td>75</td>
<td>5</td>
<td>37</td>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>62</td>
<td>7</td>
<td>90</td>
<td>5</td>
<td>98</td>
<td>3</td>
<td>31</td>
<td>2</td>
<td>91</td>
<td>4</td>
<td>38</td>
<td>9</td>
<td>72</td>
<td>1</td>
<td>9</td>
<td>0</td>
<td>72</td>
<td>6</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>0</td>
<td>39</td>
<td>9</td>
<td>74</td>
<td>5</td>
<td>25</td>
<td>7</td>
<td>47</td>
<td>3</td>
<td>52</td>
<td>2</td>
<td>63</td>
<td>8</td>
<td>21</td>
<td>6</td>
<td>35</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>58</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>50</td>
<td>8</td>
<td>52</td>
<td>1</td>
<td>88</td>
<td>6</td>
<td>20</td>
<td>2</td>
<td>68</td>
<td>5</td>
<td>24</td>
<td>4</td>
<td>53</td>
<td>7</td>
<td>57</td>
</tr>
<tr>
<td>7</td>
<td>99</td>
<td>3</td>
<td>91</td>
<td>4</td>
<td>33</td>
<td>5</td>
<td>19</td>
<td>2</td>
<td>18</td>
<td>6</td>
<td>38</td>
<td>0</td>
<td>24</td>
<td>9</td>
<td>35</td>
<td>1</td>
<td>49</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>68</td>
<td>3</td>
<td>60</td>
<td>2</td>
<td>77</td>
<td>7</td>
<td>10</td>
<td>8</td>
<td>60</td>
<td>5</td>
<td>15</td>
<td>9</td>
<td>72</td>
<td>1</td>
<td>18</td>
<td>6</td>
<td>90</td>
<td>4</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>79</td>
<td>1</td>
<td>60</td>
<td>3</td>
<td>56</td>
<td>6</td>
<td>91</td>
<td>2</td>
<td>40</td>
<td>8</td>
<td>86</td>
<td>7</td>
<td>72</td>
<td>0</td>
<td>80</td>
<td>5</td>
<td>89</td>
<td>4</td>
<td>51</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>2</td>
<td>92</td>
<td>5</td>
<td>23</td>
<td>6</td>
<td>46</td>
<td>8</td>
<td>40</td>
<td>7</td>
<td>72</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>23</td>
<td>0</td>
<td>95</td>
<td>9</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>5</td>
<td>29</td>
<td>9</td>
<td>49</td>
<td>8</td>
<td>55</td>
<td>0</td>
<td>47</td>
<td>6</td>
<td>77</td>
<td>3</td>
<td>77</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>28</td>
<td>4</td>
<td>48</td>
</tr>
</tbody>
</table>

Lawrence 15x10 instance (Table 7, instance 4)
### Instance swv15

Instance swv15 by Storer et al.\textsuperscript{17}

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>68</td>
<td>19</td>
<td>15</td>
<td>23</td>
<td>11</td>
<td>7</td>
<td>22</td>
<td>23</td>
<td>32</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>19</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>32</td>
<td>22</td>
<td>23</td>
<td>32</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>44</td>
<td>23</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>11</td>
<td>22</td>
<td>20</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>23</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>11</td>
<td>22</td>
<td>20</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>44</td>
<td>23</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>11</td>
<td>22</td>
<td>20</td>
<td>34</td>
<td>35</td>
</tr>
</tbody>
</table>

**Storer, Wu, and Vaccari hard 50x10 instance (Table 2, instance 15)**

50 jobs

10 machines

---

Gráfico 50x10
Instance **yn4**

Instance **yn4** by Yamada and Nakano\(^\text{18}\).
Problem Instance Data in Java

- How can we represent such data in Java program code?
Problem Instance Data in Java

- How can we represent such data in Java program code?

```java
package aitoa.examples.jssp;

public class JSSPInstance {

    public final int m; // number of machines
    public final int n; // number of jobs
    public final int[][] jobs; // one row per job

    /** Some stuff that is not relevant here has been omitted. 
     * You can find it in the full code online. */

}
```
Solution Space
Output: Candidate Solutions and Solution Space

- We now know how a problem instance of the JSSP looks like, i.e., the **input** we get.
Output: Candidate Solutions and Solution Space

- We now know how a problem instance of the JSSP looks like, i.e., the input we get.
- But what output should we produce?
Output: Candidate Solutions and Solution Space

• We now know how a problem instance of the JSSP looks like, i.e., the input we get.
• But what output should we produce?
• In other words, what is a solution for an instance of the JSSP?
Output: Candidate Solutions and Solution Space

• We now know how a problem instance of the JSSP looks like, i.e., the input we get.
• But what output should we produce?
• In other words, what is a solution for an instance of the JSSP?
• Basically, a Gantt Chart\textsuperscript{19} \textsuperscript{20}.
one possible solution for the demo instance, illustrated as Gantt chart
one possible solution for the la24 instance, illustrated as Gantt chart
one possible solution for the yn4 instance, illustrated as Gantt chart
one possible solution for the swv15 instance, illustrated as Gantt chart
We now know how a problem instance of the JSSP looks like, i.e., the input we get.

But what output should we produce?

In other words, what is a solution for an instance of the JSSP?

Basically, a Gantt Chart\textsuperscript{19,20}.

A Gantt chart is a diagram which assigns each sub-job on each machine a start and end time.
We now know how a problem instance of the JSSP looks like, i.e., the input we get.

But what output should we produce?

In other words, what is a solution for an instance of the JSSP?

Basically, a Gantt Chart\textsuperscript{19, 20}.

A Gantt chart is a diagram which assigns each sub-job on each machine a start and end time.

The solution space $\mathbb{Y}$ is the set of all possible feasible solutions for one JSSP instance.
Output: Candidate Solutions and Solution Space

- We now know how a problem instance of the JSSP looks like, i.e., the input we get.
- But what output should we produce?
- In other words, what is a solution for an instance of the JSSP?
- Basically, a Gantt Chart\(^{19}\)\(^{20}\).
- A Gantt chart is a diagram which assigns each sub-job on each machine a start and end time.
- The solution space \(\mathbb{Y}\) is the set of all possible feasible solutions for one JSSP instance.
- One possible solution is called candidate solution and it can be illustrated as Gantt chart.
As Java Class

- We now need to represent this information as a Java class.
As Java Class

- We now need to represent this information as a Java class.

```java
package aitoa.examples.jssp;

public class JSSPCandidateSolution {

    public int[][] schedule; // one row per machine

    /** Some stuff that is not relevant here has been omitted. 
     * You can find it in the full code online. */
}
```
As Java Class

• We now need to represent this information as a Java class.
• Each of the $m$ int[] lists in schedule holds $n$ operations for each machine as three values jobID, start time, end time, i.e., has length $3n$.

```java
package aitoa.examples.jssp;

public class JSSPCandidateSolution {

    public int[][] schedule; // one row per machine

    /** Some stuff that is not relevant here has been omitted. You can find it in the full code online. */
}
```
As Java Class

```java
new int[][] {
    {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
    {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
    {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
    {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
    {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
```
As Java Class
new int[][] {
    M0 {0, 10, 20, 30, 155, 175, 2, 175, 185},
    M1 {1, 20, 30, 50, 70, 175, 190},
    M2 {2, 0, 30, 0, 90, 140, 155},
    M3 {1, 60, 90, 90, 130, 140, 170},
    M4 {3, 50, 60, 50, 130, 140, 170}
}
As Java Class

```java
new int[][][] {
  {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
  {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
  {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
```
As Java Class

```java
new int[][][] {
    {{0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},,
     {1, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},,
     {2, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},,
     {1, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},,
     {3, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
};
```
As Java Class

new int[][][] {
  {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
  {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
  {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}

time
machine
As Java Class

```java
new int[][][] {
  {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
  {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
  {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
};
```
new int[][][] {
    {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
    {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
    {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
    {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
    {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
As Java Class

```java
new int[][][] {
  {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
  {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
  {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
```
new int[][][] {
    {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
    {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
    {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
    {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
    {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
As Java Class

```java
new int[][][] {
  {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
  {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
  {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
};
```
As Java Class

```
new int[][][] {
  {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
  {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
  {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
```
new int[][][] { 
  {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
  {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
  {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
As Java Class

```java
new int[][][] {
  {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
  {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
  {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
```

As Java Class

```java
new int[][][] {
    {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
    {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
    {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
    {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
    {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
};
```
As Java Class

```java
new int[][][] {
    {{0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
     {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
     {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
     {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
     {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
```

```
new int[][][] { 
    {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
    {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
    {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
    {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
    {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
As Java Class

```java
new int[][][] {
  {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
  {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
  {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
};
```

![Diagram of the int[][][] array with time machine and machine axes]
As Java Class

new int[][][] {
    {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
    {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
    {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
    {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
    {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
As Java Class

```java
new int[][][] {
  {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
  {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
  {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
```
As Java Class

```java
new int[][][] {
    {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
    {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
    {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
    {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
    {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
};
```
As Java Class

```java
new int[][][] {
    {0,  0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
    {1,  0, 20, 2, 30, 50, 0,  50,  70, 3, 175, 190},
    {2,  0, 30, 0, 70, 90, 1,  90, 140, 3, 140, 155},
    {1, 30, 60, 3, 60, 90, 0,  90, 130, 2, 130, 170},
    {3,  0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
```
new int[][][] {
    {0,  0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
    {1,  0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
    {2,  0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
    {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
    {3,  0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
As Java Class

```java
new int[][][] {
    {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
    {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
    {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
    {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
    {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
```

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>20</td>
<td>2</td>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>70</td>
<td>90</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>60</td>
<td>3</td>
<td>60</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>50</td>
<td>2</td>
<td>50</td>
<td>62</td>
</tr>
</tbody>
</table>
```
As Java Class

```java
new int[][][] {
  {0,  0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1,  0, 20, 2, 30, 50, 0,  50,  70, 3, 175, 190},
  {2,  0, 30, 0, 70, 90, 1,  90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0,  90, 130, 2, 130, 170},
  {3,  0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
```

```
As Java Class

```java
new int[][][] {
  {0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
  {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
  {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
  {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
  {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
```
As Java Class

```java
new int[][][] {
    {{0, 0, 10, 1, 20, 30, 3, 155, 175, 2, 175, 185},
     {1, 0, 20, 2, 30, 50, 0, 50, 70, 3, 175, 190},
     {2, 0, 30, 0, 70, 90, 1, 90, 140, 3, 140, 155},
     {1, 30, 60, 3, 60, 90, 0, 90, 130, 2, 130, 170},
     {3, 0, 50, 2, 50, 62, 0, 130, 140, 1, 140, 170}
}
}
```
Objective Function
Solution Quality

• So we have identified what the possible solutions to our problems are and know how to store them in a data structure.
Solution Quality

- So we have identified what the possible solutions to our problems are and know how to store them in a data structure.
- How do we rate the quality of a solution?
Solution Quality

• So we have identified what the possible solutions to our problems are and know how to store them in a data structure.

• How do we rate the quality of a solution?

• A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if it allows us to complete our work faster.
Solution Quality

• So we have identified what the possible solutions to our problems are and know how to store them in a data structure.
• How do we rate the quality of a solution?
• A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if it allows us to complete our work faster.
• The objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ is the makespan
Solution Quality

- So we have identified what the possible solutions to our problems are and know how to store them in a data structure.
- How do we rate the quality of a solution?
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if it allows us to complete our work faster.
- The objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ is the makespan, the time when the last sub-job is completed.
Solution Quality

- So we have identified what the possible solutions to our problems are and know how to store them in a data structure.
- How do we rate the quality of a solution?
- A Gantt chart $y_1 \in Y$ is a better solution to our problem than another chart $y_2 \in Y$ if it allows us to complete our work faster.
- The objective function $f : Y \mapsto \mathbb{R}$ is the makespan, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.
Solution Quality

- The objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ is the makespan, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.

![Gantt chart with makespan indicating the end time of the last sub-job.](chart.png)
Solution Quality

• The objective function $f : Y \mapsto \mathbb{R}$ is the makespan, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.
Solution Quality

- The objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ is the makespan, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.
• The objective function $f : \mathbb{Y} \rightarrow \mathbb{R}$ is the makespan, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.
Solution Quality

• The objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ is the makespan, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.
The objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ is the makespan, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.
Solution Quality

- The objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ is the **makespan**, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.
Solution Quality

- The objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ is the **makespan**, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.

![Gantt chart with makespan annotation](image-url)
Solution Quality

- The objective function $f: \mathbb{Y} \mapsto \mathbb{R}$ is the makespan, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.
Solution Quality

- The objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ is the makespan, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.
Solution Quality

- So we have identified what the possible solutions to our problems are and know how to store them in a data structure.
- How do we rate the quality of a solution?
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if it allows us to complete our work faster.
- The objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ is the makespan, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.
- This objective function is subject to minimization: smaller values are better.
Solution Quality

- So we have identified what the possible solutions to our problems are and know how to store them in a data structure.
- How do we rate the quality of a solution?
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if it allows us to complete our work faster.
- The objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ is the makespan, the time when the last sub-job is completed, the right-most edge of any bar in the Gantt chart.
- This objective function is subject to minimization: smaller values are better.
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$. 


Solution Quality

- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$. 
A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$.
Solution Quality

- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$. 

![Gantt Chart Example]

- Another Gantt chart example with different tasks and time slots.
• A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$. 
Solution Quality

- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$. 
Solution Quality

- A Gantt chart \( y_1 \in \mathbb{Y} \) is a better solution to our problem than another chart \( y_2 \in \mathbb{Y} \) if \( f(y_1) < f(y_2) \).
Solution Quality

- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$.
Solution Quality

- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$. 
package aitoa.structure;

public interface IObjectiveFunction<Y> {
    double evaluate(Y y);
}

An Interface for Objective Functions in Java
package aitoa.examples.jssp;

public class JSSPMakespanObjectiveFunction {
   //

   /** Some stuff that is not relevant here has been omitted. 
       You can find it in the full code online. */

   //
   //
   //
   //
   //
   //
   //
   //
   //
   //
   //
   //
   //
   //
   //
   //
   //
   //
   //
   }

package aitoa.examples.jssp;

public class JSSPMakespanObjectiveFunction implements IObjectiveFunction<JSSPCandidateSolution> {

    /** Some stuff that is not relevant here has been omitted. You can find it in the full code online. */

    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
}

The JSSP Objective Function in Java

```java
package aitoa.examples.jssp;

public class JSSPMakespanObjectiveFunction implements IObjectiveFunction<JSSPCandidateSolution> {

    /** Some stuff that is not relevant here has been omitted. 
     * You can find it in the full code online. */

    public double evaluate(final JSSPCandidateSolution y) {
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
    }
}
```
package aitoa.examples.jssp;

public class JSSPMakespanObjectiveFunction implements IObjectiveFunction<JSSPCandidateSolution> {

  /** Some stuff that is not relevant here has been omitted. You can find it in the full code online. */

  public double evaluate(final JSSPCandidateSolution y) {
    int makespan = 0; // biggest end time
package aitoa.examples.jssp;

public class JSSPMakespanObjectiveFunction implements IObjectiveFunction<JSSPCandidateSolution> {

    /** Some stuff that is not relevant here has been omitted.
     * You can find it in the full code online. */

    public double evaluate(final JSSPCandidateSolution y) {
        int makespan = 0; // biggest end time
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        return makespan;
    }
}

package aitoa.examples.jssp;

public class JSSPMakespanObjectiveFunction implements IObjectiveFunction<JSSPCandidateSolution> {

    /** Some stuff that is not relevant here has been omitted. You can find it in the full code online. */

    public double evaluate(final JSSPCandidateSolution y) {
        int makespan = 0; // biggest end time
        for (int[] machine : y.schedule) {
            //
            //
            //
            //
        }
        return makespan;
    }
}
The JSSP Objective Function in Java

```java
package aitoa.examples.jssp;

public class JSSPMakespanObjectiveFunction
        implements IObjectiveFunction<JSSPCandidateSolution> {

    /** Some stuff that is not relevant here has been omitted. 
        You can find it in the full code online. */

    public double evaluate(final JSSPCandidateSolution y) {
        int makespan = 0; // biggest end time
        for (int[] machine : y.schedule) {
            int end = machine[machine.length - 1];
            //
            //
            //
        }
        return makespan;
    }
}
```
package aitoa.examples.jssp;

public class JSSPMakespanObjectiveFunction implements IObjectiveFunction<JSSPCandidateSolution> {

    /** Some stuff that is not relevant here has been omitted. You can find it in the full code online. */

    public double evaluate(final JSSPCandidateSolution y) {
        int makespan = 0; // biggest end time
        for (int[] machine : y.schedule) {
            int end = machine[machine.length - 1];
            if (end > makespan) { // this machine ends later
                makespan = end; // remember biggest end time
            }
        }
        return makespan;
    }
}
The Global Optimum $y^*$ in $\mathbb{Y}$

- There must be at least one *globally optimal* solution $y^*$. 
The Global Optimum $y^*$ in $\mathbb{Y}$

- There must be at least one **globally optimal** solution $y^*$ for which $f(y^*) \leq f(y) \ \forall y \in \mathbb{Y}$ holds.
The Global Optimum $y^*$ in $\mathbb{Y}$

- There must be at least one **globally optimal** solution $y^*$ for which $f(y^*) \leq f(y) \forall y \in \mathbb{Y}$ holds.
- How do we find such a solution?
The Global Optimum $y^*$ in $\mathbb{Y}$

- There must be at least one **globally optimal** solution $y^*$ for which $f(y^*) \leq f(y)$ $\forall y \in \mathbb{Y}$ holds.
- How do we find such a solution?
- We know the problem is $\mathcal{NP}$-hard\textsuperscript{10,11}, so any algorithm that guarantees that it will always find this solution may sometimes need a runtime exponential in $m$ or $n$ in the worst case.
The Global Optimum $y^*$ in $\mathcal{Y}$

- $f(s) = s^8$
- $f(s) = s^{10}$
- $f(s) = s^2$
- $f(s) = s$
- $f(s) = e^s$
- $f(s) = 2^s$
- $f(s) = 1.1^s$

Since the big bang, picoseconds

$1 \text{ trillion}$
$1 \text{ billion}$
$1 \text{ million}$

$10^{-1}$ $10^0$ $10^1$ $10^2$ $10^3$ $10^4$ $10^5$ $10^6$ $10^7$ $10^8$ $10^9$ $10^{10}$ $10^{11}$ $10^{12}$ $10^{13}$ $10^{14}$ $10^{15}$ $10^{16}$ $10^{17}$ $10^{18}$ $10^{19}$ $10^{20}$ $10^{21}$ $10^{22}$ $10^{23}$ $10^{24}$ $10^{25}$ $10^{26}$ $10^{27}$ $10^{28}$ $10^{29}$ $10^{30}$ $10^{31}$ $10^{32}$ $10^{33}$ $10^{34}$ $10^{35}$ $10^{36}$ $10^{37}$ $10^{38}$ $10^{39}$ $10^{40}$

ms per day
The Global Optimum $y^*$ in $\mathbb{Y}$

• There must be at least one globally optimal solution $y^*$ for which $f(y^*) \leq f(y) \ \forall y \in \mathbb{Y}$ holds.

• How do we find such a solution?

• We know the problem is $\mathcal{NP}$-hard\textsuperscript{10,11}, so any algorithm that guarantees that it will always find this solution may sometimes need a runtime exponential in $m$ or $n$ in the worst case.

• So we cannot guarantee to always find the best possible solution for a normal-sized JSSP in reasonable time.
The Global Optimum $y^*$ in $\mathbb{Y}$

- There must be at least one globally optimal solution $y^*$ for which $f(y^*) \leq f(y) \forall y \in \mathbb{Y}$ holds.
- How do we find such a solution?
- We know the problem is $\mathcal{NP}$-hard, so any algorithm that guarantees that it will always find this solution may sometimes need a runtime exponential in $m$ or $n$ in the worst case.
- So we cannot guarantee to always find the best possible solution for a normal-sized JSSP in reasonable time.
- What we can always do is search in $\mathbb{Y}$ and hope to get as close to $y^*$ within reasonable time as possible.
The Global Optimum $y^*$ in $\mathbb{Y}$

- There must be at least one **globally optimal** solution $y^*$ for which $f(y^*) \leq f(y) \ \forall y \in \mathbb{Y}$ holds.
- How do we find such a solution?
- We know the problem is $NP$-hard, so any algorithm that guarantees that it will always find this solution may sometimes need a runtime exponential in $m$ or $n$ in the worst case.
- So we cannot guarantee to always find the best possible solution for a normal-sized JSSP in reasonable time.
- What we can always do is search in $\mathbb{Y}$ and hope to get as close to $y^*$ within reasonable time as possible.
- If we can find a solution with a slightly larger makespan than the best possible solution, but we can get it within a few minutes, that would also be nice...
From Solution Space to Search Space
Feasibility of Solutions

- So what do we need to consider when searching in $Y$?
Feasibility of Solutions

• So what do we need to consider when searching in $\mathcal{Y}$?

• A candidate solution $y \in \mathcal{Y}$ is feasible, i.e., can actually be “used,” if and only if it fulfills all constraints.
Feasibility of Solutions

• So what do we need to consider when searching in $\mathbb{Y}$?

• A candidate solution $y \in \mathbb{Y}$ is feasible, i.e., can actually be “used,” if and only if it fulfills all *constraints*.

• Indeed, there are several constraints we need to impose on our Gantt charts
Feasibility of Solutions

- So what do we need to consider when searching in $Y$?
- A candidate solution $y \in Y$ is feasible, i.e., can actually be “used,” if and only if it fulfills all constraints.
- Indeed, there are several constraints we need to impose on our Gantt charts:
  1. all operations of all jobs must be assigned to their respective machines and properly be completed
Feasibility of Solutions

cannot omit operation
Feasibility of Solutions

[Diagram with a skull and colored bars indicating some data]
Feasibility of Solutions

![Bar Chart](Image)
Feasibility of Solutions

cannot move operation
Feasibility of Solutions
Feasibility of Solutions

• So what do we need to consider when searching in $\mathbb{Y}$?

• A candidate solution $y \in \mathbb{Y}$ is feasible, i.e., can actually be “used,” if and only if it fulfills all constraints.

• Indeed, there are several constraints we need to impose on our Gantt charts:
  1. all operations of all jobs must be assigned to their respective machines and properly be completed,
  2. only the jobs and machines specified by the problem instance must occur in the chart
Feasibility of Solutions

can’t add machines
Feasibility of Solutions

can’t add machines

0 1

3 2

1 1

2

1 2 0

3

0 1

3 2
Feasibility of Solutions

• So what do we need to consider when searching in $\mathbb{Y}$?

• A candidate solution $y \in \mathbb{Y}$ is feasible, i.e., can actually be “used,” if and only if it fulfills all constraints.

• Indeed, there are several constraints we need to impose on our Gantt charts:
  1. all operations of all jobs must be assigned to their respective machines and properly be completed,
  2. only the jobs and machines specified by the problem instance must occur in the chart,
  3. an operation must be assigned a time window on its corresponding machine which is exactly as long as the operation needs on that machine.
Feasibility of Solutions
Feasibility of Solutions

cannot shorten jobs
Feasibility of Solutions
Feasibility of Solutions

- So what do we need to consider when searching in $\mathbb{Y}$?
- A candidate solution $y \in \mathbb{Y}$ is **feasible**, i.e., can actually be “used,” if and only if it fulfills all **constraints**.
- Indeed, there are several constraints we need to impose on our Gantt charts:
  1. all operations of all jobs must be assigned to their respective machines and properly be completed,
  2. only the jobs and machines specified by the problem instance must occur in the chart,
  3. an operation must be assigned a time window on its corresponding machine which is exactly as long as the operation needs on that machine,
  4. the operations cannot intersect or overlap, each machine can only carry out one job at a time
Feasibility of Solutions
Feasibility of Solutions

Operations must not overlap!
Feasibility of Solutions
Feasibility of Solutions

- So what do we need to consider when searching in $\mathbb{Y}$?
- A candidate solution $y \in \mathbb{Y}$ is feasible, i.e., can actually be “used,” if and only if it fulfills all constraints.
- Indeed, there are several constraints we need to impose on our Gantt charts:
  1. all operations of all jobs must be assigned to their respective machines and properly be completed,
  2. only the jobs and machines specified by the problem instance must occur in the chart,
  3. an operations must be assigned a time window on its corresponding machine which is exactly as long as the operation needs on that machine,
  4. the operations cannot intersect or overlap, each machine can only carry out one job at a time, and
  5. the precedence constraints of the operations must be honored.
Feasibility of Solutions
Feasibility of Solutions

order of operations must be preserved
Feasibility of Solutions
Feasibility of Solutions

• So what do we need to consider when searching in $Y$?
• A candidate solution $y \in Y$ is feasible, i.e., can actually be “used,” if and only if it fulfills all constraints.
• Indeed, there are several constraints we need to impose on our Gantt charts:
  1. all operations of all jobs must be assigned to their respective machines and properly be completed,
  2. only the jobs and machines specified by the problem instance must occur in the chart,
  3. an operation must be assigned a time window on its corresponding machine which is exactly as long as the operation needs on that machine,
  4. the operations cannot intersect or overlap, each machine can only carry out one job at a time, and
  5. the precedence constraints of the operations must be honored.
• Only a Gantt chart obeying all of these constraints is feasible, i.e., can be implemented in practice.
Hardships when Searching in \( Y \)

- So how do we search in the space of Gantt charts?
Hardships when Searching in Gantt Charts

- So how do we search in the space of Gantt charts?
- We need to create Gantt charts that fulfill all the constraints.
Hardships when Searching in $\mathcal{Y}$

- So how do we search in the space of Gantt charts?
- We need to create Gantt charts that fulfill all the constraints.
- For different instances, different solutions are feasible!
Hardships when Searching in \( \mathcal{Y} \)

Instance A with 2 jobs and 2 machines:

\[
\begin{array}{lll}
2 & 2 \\
0 & 10 & 1 & 20 \\
0 & 10 & 1 & 20 \\
\end{array}
\]
Hardships when Searching in \( Y \)

instance A with 2 jobs and 2 machines

\[
\begin{array}{cccc}
2 & 2 \\
0 & 10 & 1 & 20 \\
0 & 10 & 1 & 20 \\
\end{array}
\]
Hardships when Searching in

Instance A with 2 jobs and 2 machines

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 2 | 2 | 0 | 10
| 1 | 20| 0 | 10
| 1 | 20|   |   |

+-------------------------------+
Hardships when Searching in instance A with 2 jobs and 2 machines

+ + + + + + + + + + + + + + + + + +
instance A with 2 jobs and 2 machines
2 2
0 10 1 20
0 10 1 20
+ + + + + + + + + + + + + + + + + +
Hardships when Searching in Y

instance A with 2 jobs and 2 machines

<table>
<thead>
<tr>
<th>Job</th>
<th>Start Time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

job 0
Hardships when Searching in Y

instance A with 2 jobs and 2 machines

job 0

job 1
Hardships when Searching in Y

M0: Job 0, Job 1
Hardships when Searching in $Y$

instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in \( \mathbb{Y} \)

instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in Y

Instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in $\mathcal{Y}$

Instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in

Instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in

Instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in $\exists$

-instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 0, Job 1
instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0
Hardships when Searching in $\mathbb{Y}$

Instance A with 2 jobs and 2 machines:

- **M0**: Job 0, Job 1;
- **M1**: Job 0, Job 1

![Gantt chart]

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0
Hardships when Searching in Y

Instance A with 2 jobs and 2 machines:

+----------------------------------+
<table>
<thead>
<tr>
<th>job 0</th>
<th>job 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10</td>
<td>1 20</td>
</tr>
<tr>
<td>0 10</td>
<td>1 20</td>
</tr>
</tbody>
</table>
+----------------------------------+

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0
Hardships when Searching in $\mathbb{Y}$

Instance A with 2 jobs and 2 machines

<table>
<thead>
<tr>
<th>M0: Job 0, Job 1; M1: Job 0, Job 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>M0: Job 0, Job 1; M1: Job 1, Job 0</td>
</tr>
</tbody>
</table>

Job 0
Job 1
Hardships when Searching in Y

Instance A with 2 jobs and 2 machines:

M0: Job 0, Job 1; M1: Job 0, Job 1

M0:
10 30 20 40

M1:
0 60 50

M0: Job 0, Job 1; M1: Job 1, Job 0

M0:
10 20

M1:
0 60 50
Hardships when Searching in Y

instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0
Hardships when Searching in $Y$

Instance A with 2 jobs and 2 machines:

- M0: Job 0, Job 1
- M1: Job 0, Job 1

M0: Job 1, Job 0

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in $Y$

instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 1, Job 0

M0: Job 1, Job 0; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in \text{Y}

Instance A with 2 jobs and 2 machines

\begin{align*}
&\text{M0: Job 0, Job 1; M1: Job 0, Job 1} \\
&\text{M0: Job 1, Job 0; M1: Job 0, Job 1}
\end{align*}
Hardships when Searching in

instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in $Y$

Instance A with 2 jobs and 2 machines:

- Job 0:
  - M0: 0-10 on M0, 10-20 on M1
  - M1: 0-10 on M1, 10-20 on M0

- Job 1:
  - M0: 20-30 on M0, 30-40 on M1
  - M1: 20-30 on M1, 30-40 on M0

Graphs showing different job assignments and their corresponding times.
Hardships when Searching in

instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in $\mathcal{Y}$

instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in $\mathbb{Y}$

Instance A with 2 jobs and 2 machines

$\begin{align*}
M0: & \text{Job 0, Job 1} \\
M1: & \text{Job 0, Job 1}
\end{align*}$

$\begin{align*}
M0: & \text{Job 0, Job 1} \\
M1: & \text{Job 1, Job 0}
\end{align*}$

$\begin{align*}
M0: & \text{Job 1, Job 0} \\
M1: & \text{Job 0, Job 1}
\end{align*}$

$\begin{align*}
M0: & \text{Job 1, Job 0}
\end{align*}$
Hardships when Searching in $\text{Y}$

Instance A with 2 jobs and 2 machines

M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in \( Y \)

instance A with 2 jobs and 2 machines

---

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0

M0: Job 1, Job 0; M1: Job 1, Job 0
Hardships when Searching in Y

Instance A with 2 jobs and 2 machines:

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0

M0: Job 1, Job 0; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 1, Job 0
Hardships when Searching in \( Y \)

Instance A with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0

M0: Job 1, Job 0; M1: Job 1, Job 0
Hardships when Searching in $\mathcal{Y}$

Instance A with 2 jobs and 2 machines

```
job 0
job 1
```

```
0 10 1 20
0 10 1 20
```

M0: Job 1, Job 0; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0

M0: Job 1, Job 0; M1: Job 1, Job 0
Hardships when Searching in $\mathcal{Y}$

Instance A with 2 jobs and 2 machines

+ + + + + + + + + + + + + + + + + + +

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in $\mathcal{Y}$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

 Instance B with 2 jobs and 2 machines.

++++++++++++++
Hardships when Searching in $Y$

instance B with 2 jobs and 2 machines
2 2
0 10 1 20
1 20 0 10

+++++++++++++++++++++++++++
Hardships when Searching in \( Y \)

Instance B with 2 jobs and 2 machines

\[
\begin{array}{llll}
2 & 2 \\
0 & 10 & 1 & 20 \\
1 & 20 & 0 & 10 \\
\end{array}
\]
Hardships when Searching in Y

Instance B with 2 jobs and 2 machines:
2 2
0 10 1 20
1 20 0 10

+++++++++++++++
Hardships when Searching in Y

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>instance B</td>
<td>2 jobs and 2 machines</td>
</tr>
<tr>
<td>2 2</td>
<td>0 10 1 20</td>
</tr>
<tr>
<td>1 20</td>
<td>0 10</td>
</tr>
</tbody>
</table>

job 0
Hardships when Searching in $\mathbb{Y}$

instance B with 2 jobs and 2 machines

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>
Hardships when Searching in Y

instance B with 2 jobs and 2 machines

job 0
job 1

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in

Instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in

Instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in

instance B with 2 jobs and 2 machines

job 0
job 1

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in

Instance B with 2 jobs and 2 machines:

M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in Y

instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

Diagram showing the schedule for jobs on machines M0 and M1.
Hardships when Searching in

Instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0
Hardships when Searching in Y

instance B with 2 jobs and 2 machines

<table>
<thead>
<tr>
<th>Job</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>
| M0: | Job 0, Job 1; M1: Job 0, Job 1

<table>
<thead>
<tr>
<th>M1:</th>
<th>0</th>
<th>60</th>
</tr>
</thead>
</table>
| M0: | Job 0, Job 1; M1: Job 1, Job 0

M1:

M0:

T: 0-10-20-30-40-50-60

T: 0-10-20-30-40-50-60
Hardships when Searching in $\mathcal{Y}$

Instance B with 2 jobs and 2 machines:

- Machine M0: Job 0, Job 1
- Machine M1: Job 1, Job 0
Hardships when Searching in Y

instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0
Hardships when Searching in

Instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 1, Job 0
Hardships when Searching in $Y$

Instance B with 2 jobs and 2 machines:

- **Job 0** and **Job 1**

---

**M0:** Job 0, Job 1; **M1:** Job 0, Job 1

---

**M0:** Job 0, Job 1; **M1:** Job 1, Job 0
Hardships when Searching in $\mathcal{Y}$

instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in $\mathbb{Y}$

instance B with 2 jobs and 2 machines

<table>
<thead>
<tr>
<th>Job 0</th>
<th>Job 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

M0: Job 0, Job 1; M1: Job 1, Job 0

M0: Job 1, Job 0; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0
Hardships when Searching in $\mathcal{Y}$

instance B with 2 jobs and 2 machines

0 10 1 20
1 20 0 10

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in \( \bigwedge \)

Instance B with 2 jobs and 2 machines:

- M0: Job 0, Job 1
- M1: Job 1, Job 0

0 10 1 20
1 20 0 10

0 10 20 40
M0: Job 0, Job 1
M1: Job 1, Job 0

0 60 50
M0: Job 1, Job 0
M1: Job 0, Job 1
Hardships when Searching in $Y$

Instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in $\mathcal{Y}$

Instance B with 2 jobs and 2 machines.

<table>
<thead>
<tr>
<th>Job</th>
<th>Start Time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>0</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
<td>20</td>
</tr>
</tbody>
</table>

M0: Job 0, Job 1; M1: Job 0, Job 1
M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in $\mathcal{Y}$

Instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0

M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in \( \mathcal{Y} \)

Instance B with 2 jobs and 2 machines

<table>
<thead>
<tr>
<th>Job 0</th>
<th>Job 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 2</td>
<td></td>
</tr>
<tr>
<td>0 10</td>
<td>1 20</td>
</tr>
<tr>
<td>1 20</td>
<td>0 10</td>
</tr>
</tbody>
</table>

M0: Job 0, Job 1; M1: Job 0, Job 1
M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in $\bigtriangledown$

instance B with 2 jobs and 2 machines

2 2
0 10 1 20
1 20 0 10

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 0, Job 1; M1: Job 1, Job 0

M0: Job 1, Job 0; M1: Job 0, Job 1
Hardships when Searching in $\mathbb{Y}$

Instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

0 10 1 20 1 20 0 10

M0: Job 1, Job 0; M1: Job 0, Job 1

Deadlock
Hardships when Searching in $\mathbb{Y}$

Instance B with 2 jobs and 2 machines:

<table>
<thead>
<tr>
<th>Job</th>
<th>Machine</th>
<th>Start Time</th>
<th>Finish Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>M0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>M0</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>0</td>
<td>M1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>M1</td>
<td>30</td>
<td>50</td>
</tr>
</tbody>
</table>

- M0: Job 0, Job 1; M1: Job 0, Job 1
- M0: Job 1, Job 0; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1 (deadlock)
Hardships when Searching in \( \mathcal{Y} \)

Machine 0 should begin by doing job 1.
Hardships when Searching in $\mathbb{Y}$

Machine 0 should begin by doing job 1. Job 1 can only start on machine 0 after it has been finished on machine 1.
Hardships when Searching in $\mathbb{Y}$

Machine 0 should begin by doing job 1. Job 1 can only start on machine 0 after it has been finished on machine 1. At machine 1, we should begin with job 0.
Hardships when Searching in Y

Instance B with 2 jobs and 2 machines

Job 1 can only start on machine 0 after it has been finished on machine 1. At machine 1, we should begin with job 0. Before job 0 can be put on machine 1, it must go through machine 0.
Hardships when Searching in $\mathbb{Y}$

instance B with 2 jobs and 2 machines

job 0
job 1

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1

deadlock

So job 1 cannot go to machine 0 until it has passed through machine 1, but in order to be executed on machine 1, job 0 needs to be finished there first.
Hardships when Searching in Y

instance B with 2 jobs and 2 machines
2 2
0 10 1 20
1 20 0 10

Job 0 cannot begin on machine 1 until it has been passed through machine 0, but it cannot be executed there, because job 1 needs to be finished there first.
Hardships when Searching in $\mathcal{Y}$

Instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1

A cyclic blockage has appeared: no job can be executed on any machine if we follow this schedule.
A cyclic blockage has appeared: no job can be executed on any machine if we follow this schedule. This is called a deadlock.
Hardships when searching in $\mathcal{Y}$

Instance B with 2 jobs and 2 machines

<table>
<thead>
<tr>
<th>Job 0</th>
<th>Job 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10</td>
<td>1 20</td>
</tr>
<tr>
<td>1 20</td>
<td>0 10</td>
</tr>
</tbody>
</table>

This is called a deadlock. The schedule is infeasible, because it cannot be executed or written down without breaking the precedence constraint.
Hardships when Searching in $\exists$

Instance B with 2 jobs and 2 machines:

- **M0**: Job 0, Job 1
- **M1**: Job 0, Job 1

M0: Job 1, Job 0
M1: Job 0, Job 1

M0: Job 1, Job 0;
M1: Job 0, Job 1

deadlock
Hardships when Searching in $\bigvee$

Instance B with 2 jobs and 2 machines

M0: Job 1, Job 0; M1: Job 0, Job 1
M0: Job 0, Job 1; M1: Job 1, Job 0

M0: Job 1, Job 0; M1: Job 1, Job 0
M0: Job 0, Job 1; M1: Job 0, Job 1
Hardships when Searching in \( Y \)

Instance B with 2 jobs and 2 machines

\[
\begin{array}{ll}
M0: & \text{Job 0, Job 1; M1: Job 0, Job 1} \\
M0: & \text{Job 1, Job 0; M1: Job 0, Job 1} \\
\end{array}
\]

\[
\begin{array}{ll}
M0: & \text{Job 1, Job 0; M1: Job 1, Job 0} \\
\end{array}
\]

\[
\begin{array}{ll}
M0: & \text{Job 0, Job 1; M1: Job 0, Job 1} \\
M0: & \text{Job 0, Job 1; M1: Job 1, Job 0} \\
\end{array}
\]

\[
\begin{array}{ll}
M0: & \text{Job 1, Job 0; M1: Job 0, Job 1} \\
M0: & \text{Job 1, Job 0; M1: Job 1, Job 0} \\
\end{array}
\]
Hardships when Searching in $Y$

Instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1

deadlock

M0: Job 1, Job 0; M1: Job 1, Job 0
Hardships when Searching in $\gamma$

Instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1

Deadlock
Hardships when Searching in $\mathbb{Y}$

instance B with 2 jobs and 2 machines

M0: Job 0, Job 1; M1: Job 0, Job 1

M0: Job 1, Job 0; M1: Job 0, Job 1

deadlock
Hardships when Searching in \( \mathbb{Y} \)

- So how do we search in the space of Gantt charts?
- We need to create Gantt charts that fulfill all the constraints.
- For different instances, different solutions are feasible!
- Writing Java code that works directly on the Gantt charts is cumbersome and error-prone.
Hardships when Searching in $\forall$

- So how do we search in the space of Gantt charts?
- We need to create Gantt charts that fulfill all the constraints.
- For different instances, different solutions are feasible!
- Writing Java code that works directly on the Gantt charts is cumbersome and error-prone.
- Actually, the vast majority of possible Gantt charts will often be infeasible and have deadlocks...
Hardships when Searching in \( \mathbb{Y} \)

- So how do we search in the space of Gantt charts?
- We need to create Gantt charts that fulfill all the constraints.
- For different instances, different solutions are feasible!
- Writing Java code that works directly on the Gantt charts is cumbersome and error-prone.
- Actually, the vast majority of possible Gantt charts will often be infeasible and have deadlocks…
- We would like to have a handy representation for Gantt charts.
Hardships when Searching in  

- So how do we search in the space of Gantt charts?
- We need to create Gantt charts that fulfill all the constraints.
- For different instances, different solutions are feasible!
- Writing Java code that works directly on the Gantt charts is cumbersome and error-prone.
- Actually, the vast majority of possible Gantt charts will often be infeasible and have deadlocks...
- We would like to have a handy representation for Gantt charts.
- The representation should allow us to easy create and modify the candidate solutions.
Hardships when Searching in $\mathbb{Y}$

- So how do we search in the space of Gantt charts?
- We need to create Gantt charts that fulfill all the constraints.
- For different instances, different solutions are feasible!
- Writing Java code that works directly on the Gantt charts is cumbersome and error-prone.
- Actually, the vast majority of possible Gantt charts will often be infeasible and have deadlocks...
- We would like to have a handy representation for Gantt charts.
- The representation should allow us to easily create and modify the candidate solutions.
- **Solution:** We develop a data structure $\mathbb{X}$ which we can handle easily and which can always be translated to feasible Gantt charts by a mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$. 
The Search Space $X$

- The solution space $Y$ is complicated and constrained.
The Search Space

- The solution space $Y$ is complicated and constrained.
- In a real-world JSSP, there would even be more issues, such as job- and machine-specific setup times and transfer times...
The Search Space $X$

- The solution space $Y$ is complicated and constrained.
- In a real-world JSSP, there would even be more issues, such as job- and machine-specific setup times and transfer times...
- If we would have a valid Gantt chart $y \in Y$, then trying to improve it would be quite complicated.
The Search Space

- The solution space $\mathcal{Y}$ is complicated and constrained.
- In a real-world JSSP, there would even be more issues, such as job- and machine-specific setup times and transfer times...
- If we would have a valid Gantt chart $\gamma \in \mathcal{Y}$, then trying to improve it would be quite complicated.
- If we imagine the space $\mathcal{Y}$ of possible Gantt charts for a JSSP, then searching through this space in some kind of targeted way would be complicated.
The Search Space

- The solution space $\mathbb{Y}$ is complicated and constrained.
- In a real-world JSSP, there would even be more issues, such as job- and machine-specific setup times and transfer times...
- If we would have a valid Gantt chart $y \in \mathbb{Y}$, then trying to improve it would be quite complicated.
- If we imagine the space $\mathbb{Y}$ of possible Gantt charts for a JSSP, then searching through this space in some kind of targeted way would be complicated.
- We want to search in a simpler space that we can easily understand.
The Search Space $X$

- The solution space $Y$ is complicated and constrained.
- In a real-world JSSP, there would even be more issues, such as job- and machine-specific setup times and transfer times...
- If we would have a valid Gantt chart $y \in Y$, then trying to improve it would be quite complicated.
- If we imagine the space $Y$ of possible Gantt charts for a JSSP, then searching through this space in some kind of targeted way would be complicated.
- We want to search in a simpler space that we can easily understand, where we do not need to worry about the constraints and feasibility.
- This space is therefore called the search space $X$. 
The Search Space $X$

- The solution space $Y$ is complicated and constrained.
- In a real-world JSSP, there would even be more issues, such as job- and machine-specific setup times and transfer times.
- If we would have a valid Gantt chart $y \in Y$, then trying to improve it would be quite complicated.
- If we imagine the space $Y$ of possible Gantt charts for a JSSP, then searching through this space in some kind of targeted way would be complicated.
- We want to search in a simpler space that we can easily understand, where we do not need to worry about the constraints and feasibility.
- This space is therefore called the search space $X$.
- Of course, $X$ must somehow be related to $Y$. 
The Search Space $\mathbb{X}$

- The solution space $\mathbb{Y}$ is complicated and constrained.
- In a real-world JSSP, there would even be more issues, such as job- and machine-specific setup times and transfer times. . .
- If we would have a valid Gantt chart $y \in \mathbb{Y}$, then trying to improve it would be quite complicated.
- If we imagine the space $\mathbb{Y}$ of possible Gantt charts for a JSSP, then searching through this space in some kind of targeted way would be complicated.
- We want to search in a simpler space that we can easily understand, where we do not need to worry about the constraints and feasibility.
- This space is therefore called the search space $\mathbb{X}$.
- Of course, $\mathbb{X}$ must somehow be related to $\mathbb{Y}$: We need a representation mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$ which translates from $\mathbb{X}$ to $\mathbb{Y}$. 
One Search Space $\mathcal{X}$ for the JSSP

- So how could a simple search space $\mathcal{X}$ for the JSSP look like?
One Search Space \( X \) for the JSSP

- So how could a simple search space \( X \) for the JSSP look like?
- Let us revisit the demo problem instance.
The JSSP involves scheduling jobs on machines. Each job has a sequence of operations on different machines, each with a time requirement.

A simple demo:

<table>
<thead>
<tr>
<th>Job</th>
<th>Operations</th>
<th>Time Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 10 1 20 2 20 3 40</td>
<td>4 10</td>
</tr>
<tr>
<td>1</td>
<td>1 20 0 10 3 30 2 50</td>
<td>4 30</td>
</tr>
<tr>
<td>2</td>
<td>2 30 1 20 4 12 3 40</td>
<td>0 10</td>
</tr>
<tr>
<td>3</td>
<td>4 50 3 30 2 15 0 20</td>
<td>1 15</td>
</tr>
</tbody>
</table>

This is information that we have, which does not need to be stored in the elements $x \in X$. 
The instance data $\mathcal{I}$ and the data from one point $x \in \mathbb{X}$ should, together, encode such a Gantt chart $y \in \mathbb{Y}$. 
One Search Space $\mathcal{X}$ for the JSSP

- So how could a simple search space $\mathcal{X}$ for the JSSP look like?
- Let us revisit the demo problem instance.
- Ideally, we want to encode this two-dimensional structure in something very simple.
One Search Space X for the JSSP

- So how could a simple search space X for the JSSP look like?
- Let us revisit the demo problem instance.
- Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
One Search Space \( X \) for the JSSP

- So how could a simple search space \( X \) for the JSSP look like?
- Let us revisit the demo problem instance.
- Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
- In the demo, we have \( m = 5 \) machines and \( n = 4 \) jobs.
One Search Space $\mathcal{X}$ for the JSSP

- So how could a simple search space $\mathcal{X}$ for the JSSP look like?
- Let us revisit the demo problem instance.
- Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
- In the demo, we have $m = 5$ machines and $n = 4$ jobs.
- We could give each of the $m \times n = 20$ operation one ID, a number in $0 \ldots 19$. 
One Search Space for the JSSP

• So how could a simple search space for the JSSP look like?
• Let us revisit the demo problem instance.
• Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
• In the demo, we have \( m = 5 \) machines and \( n = 4 \) jobs.
• We could give each of the \( m \times n = 20 \) operation one ID, a number in \( 0 \ldots 19 \).
• Then, a linear string containing a permutation of these IDs could denote the exact processing order of the operations.
One Search Space for the JSSP

- So how could a simple search space for the JSSP look like?
- Let us revisit the demo problem instance.
- Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
- In the demo, we have \( m = 5 \) machines and \( n = 4 \) jobs.
- We could give each of the \( m \times n = 20 \) operation one ID, a number in \( 0 \ldots 19 \).
- Then, a linear string containing a permutation of these IDs could denote the exact processing order of the operations.
- We could easily translate such strings to Gantt charts.
One Search Space $X$ for the JSSP

• So how could a simple search space $X$ for the JSSP look like?
• Let us revisit the demo problem instance.
• Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
• In the demo, we have $m = 5$ machines and $n = 4$ jobs.
• We could give each of the $m \times n = 20$ operation one ID, a number in $0 \ldots 19$.
• Then, a linear string containing a permutation of these IDs could denote the exact processing order of the operations.
• We could easily translate such strings to Gantt charts, but we could end up with infeasible solutions and deadlocks or a string telling us to do the second operation of a job before the first one...
One Search Space \( X \) for the JSSP

- So how could a simple search space \( X \) for the JSSP look like?
- Let us revisit the demo problem instance.
- Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
- In the demo, we have \( m = 5 \) machines and \( n = 4 \) jobs.
- We could give each of the \( m \times n = 20 \) operation one ID, a number in \( 0 \ldots 19 \).
- How can we use a linear encoding without deadlocks?
One Search Space \( \mathcal{X} \) for the JSSP

- So how could a simple search space \( \mathcal{X} \) for the JSSP look like?
- Let us revisit the demo problem instance.
- Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
- In the demo, we have \( m = 5 \) machines and \( n = 4 \) jobs.
- We could give each of the \( m \times n = 20 \) operation one ID, a number in \( 0 \ldots 19 \).
- How can we use a linear encoding without deadlocks?
- Each job has \( m = 5 \) operations that must be distributed to the machines in the sequence prescribed in the problem instance data.
One Search Space $X$ for the JSSP

- So how could a simple search space $X$ for the JSSP look like?
- Let us revisit the demo problem instance.
- Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
- In the demo, we have $m = 5$ machines and $n = 4$ jobs.
- We could give each of the $m \times n = 20$ operation one ID, a number in $0 \ldots 19$.
- How can we use a linear encoding without deadlocks?
- Each job has $m = 5$ operations that must be distributed to the machines in the sequence prescribed in the problem instance data.
- We know the order of the operations per job.
One Search Space for the JSSP

• So how could a simple search space for the JSSP look like?
• Let us revisit the demo problem instance.
• Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
• In the demo, we have \( m = 5 \) machines and \( n = 4 \) jobs.
• We could give each of the \( m \times n = 20 \) operation one ID, a number in \( 0 \ldots 19 \).
• How can we use a linear encoding without deadlocks?
• Each job has \( m = 5 \) operations that must be distributed to the machines in the sequence prescribed in the problem instance data.
• We know the order of the operations per job \( \implies \) we do not need to encode it.
One Search Space \( X \) for the JSSP

- So how could a simple search space \( X \) for the JSSP look like?
- Let us revisit the demo problem instance.
- Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
- In the demo, we have \( m = 5 \) machines and \( n = 4 \) jobs.
- We could give each of the \( m \times n = 20 \) operation one ID, a number in \( 0 \ldots 19 \).
- How can we use a linear encoding without deadlocks?
- Each job has \( m = 5 \) operations that must be distributed to the machines in the sequence prescribed in the problem instance data.
- We know the order of the operations per job \( \implies \) we do not need to encode it.
- We just include each job id \( m \) times in the string.\(^{21\text{--}24}\)
One Search Space $X$ for the JSSP

• So how could a simple search space $X$ for the JSSP look like?
• Let us revisit the demo problem instance.
• Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
• In the demo, we have $m = 5$ machines and $n = 4$ jobs.
• We could give each of the $m \times n = 20$ operation one ID, a number in $0 \ldots 19$.
• How can we use a linear encoding without deadlocks?
• Each job has $m = 5$ operations that must be distributed to the machines in the sequence prescribed in the problem instance data.
• We know the order of the operations per job $\implies$ we do not need to encode it.
• We just include each job id $m$ times in the string.$^{21–24}$
• The first occurrence of a job’s ID stands for its first operation, the second occurrence for the second operation, and so on.
One Search Space $X$ for the JSSP

- So how could a simple search space $X$ for the JSSP look like?
- Let us revisit the demo problem instance.
- Ideally, we want to encode this two-dimensional structure in a simple one-dimensional string of integer numbers.
- In the demo, we have $m = 5$ machines and $n = 4$ jobs.
- We could give each of the $m \times n = 20$ operation one ID, a number in $0 \ldots 19$.
- How can we use a linear encoding without deadlocks?
- Each job has $m = 5$ operations that must be distributed to the machines in the sequence prescribed in the problem instance data.
- We know the order of the operations per job $\Longrightarrow$ we do not need to encode it.
- We just include each job id $m$ times in the string.$^{21-24}$
- The first occurrence of a job’s ID stands for its first operation, the second occurrence for the second operation, and so on.
- This way, we will always have the operations in the right order.
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]
Demo Example for the Search Space

\[ x \in X \]
\[
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}
\]
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \mapsto Y \]

\[ y \in Y \]
Demo Example for the Search Space

\( x \in X \)
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\( \gamma : X \mapsto Y \)

A simple demo

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 4 | 5 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0 | 10| 1 | 20 | 2 | 20 | 3 | 40 | 4 | 10 |   |   |   |   |   |   |   |   |   |   |
| 1 | 20 | 0 | 10 | 3 | 30 | 2 | 50 | 4 | 30 |   |   |   |   |   |   |   |   |   |   |
| 2 | 30 | 1 | 20 | 4 | 12 | 3 | 40 | 0 | 10 |   |   |   |   |   |   |   |   |   |   |
| 4 | 50 | 3 | 30 | 2 | 15 | 0 | 20 | 1 | 15 |   |   |   |   |   |   |   |   |   |   |

machine

time

\( y \in Y \)
Demo Example for the Search Space

\[ x \in X \\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma: X \rightarrow Y \]

A simple demo

\[
\begin{array}{cccccccc}
4 & 5 & \\
0 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
\textbf{1} & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 4 & 30 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array}
\]
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

A simple demo
4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15
Demo Example for the Search Space

\[ x \in X \]
{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}\n
\[ \gamma : X \rightarrow Y \]

A simple demo

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10</td>
<td>1 20</td>
<td>2 20</td>
</tr>
<tr>
<td>1 20</td>
<td>0 10</td>
<td>3 30</td>
</tr>
<tr>
<td>2 30</td>
<td>1 20</td>
<td>4 12</td>
</tr>
<tr>
<td>4 50</td>
<td>3 30</td>
<td>2 15</td>
</tr>
</tbody>
</table>

machine
time
Demo Example for the Search Space

$x \in X$
{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}

$\gamma : X \mapsto Y$

A simple demo

<table>
<thead>
<tr>
<th>?</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>1 20 2 20 3 40 4 10</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0 10 3 30 2 50 4 30</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1 20 4 12 3 40 0 10</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>3 30 2 15 0 20 1 15</td>
</tr>
</tbody>
</table>

$y \in Y$
Demo Example for the Search Space

\[ x \in X \]
\[
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}
\]

\[ \gamma : X \mapsto Y \]

\[ y \in Y \]

A simple demo:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>20</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>12</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>10</td>
<td>4</td>
<td>10</td>
<td>4</td>
<td>30</td>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>4</td>
<td>10</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>2</td>
<td>50</td>
<td>4</td>
<td>30</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>20</td>
<td>1</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Demo Example for the Search Space

$x \in X$
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

$\gamma : X \rightarrow Y$

+++

A simple demo
4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15
+++

+++

An example graph with labeled axes and data points.

machine

2
0

1
0

y \in Y

time

0
50
100
150
Demo Example for the Search Space

\[ x \in X \trianglerighteq \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

\[ y \in Y \]

A simple demo

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
</tr>
</tbody>
</table>
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 3, 1, 2, 0, 3, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

A simple demo

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>1</td>
<td>20</td>
<td>2</td>
<td>20</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>12</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

\[ y \in Y \]
Demo Example for the Search Space

$x \in X$
{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}

$\gamma : X \mapsto Y$

A simple demo
4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15

$y \in Y$
Demo Example for the Search Space

\[ x \in X \]
{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}
Demo Example for the Search Space

\[ x \in X \]\n\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \mapsto Y \]

\[ y \in Y \]

A simple demo

<table>
<thead>
<tr>
<th>time</th>
<th>machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10</td>
<td>1 20</td>
</tr>
<tr>
<td>1 20</td>
<td>0 10</td>
</tr>
<tr>
<td>2 30</td>
<td>1 20</td>
</tr>
<tr>
<td>4 50</td>
<td>3 30</td>
</tr>
</tbody>
</table>

...
$x \in X$
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

$\gamma: X \rightarrow Y$

+++
A simple demo
4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15
+++

machine

\begin{tikzpicture}
\draw[->](0,0)--(5,0);
\draw(5,0)--(5,-5);
\draw(0,0)--(0,-5);
\draw(0,0)--(5,0);
\draw(0,0)--(0,5);
\draw[thick](0,0)--(0,1);
\draw[thick](0,0)--(0,2);
\draw[thick](0,0)--(0,3);
\draw[thick](0,0)--(0,4);
\draw[thick](0,1)--(1,2);
\draw[thick](0,2)--(2,3);
\draw[thick](0,3)--(3,4);
\draw[thick](0,4)--(4,5);
\node[red] at (0.5,-0.5) {0};
\node[red] at (1.5,-0.5) {1};
\node[blue] at (2.5,-0.5) {1};
\node[blue] at (3.5,-0.5) {2};
\node[green] at (4.5,-0.5) {2};
\node[blue] at (5.5,-0.5) {1};
\node[red] at (1,-1) {1};
\node[blue] at (2,-1) {1};
\node[green] at (3,-1) {2};
\node[red] at (4,-1) {1};
\node[blue] at (5,-1) {1};
\node[green] at (6,-1) {2};
\node at (2,-3) {	extbf{time}};
\node at (2,-4) {	extbf{machine}};
\end{tikzpicture}

$y \in Y$
Demo Example for the Search Space

$x \in X$
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

$\gamma : X \mapsto Y$

A simple demo:

\[
\begin{array}{cccccccc}
4 & 5 \\
0 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
1 & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 4 & 30 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array}
\]
Demo Example for the Search Space

\[ x \in X \]
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\[ \gamma : X \mapsto Y \]

\[ y \in Y \]

A simple demo

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>20</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>50</td>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>12</td>
<td>3</td>
<td>40</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>20</td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

machine

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

time
Demo Example for the Search Space

\( x \in X \)
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\( \gamma : X \mapsto Y \)

\[ ++++++++++++++++ \]
A simple demo
\[
4 \quad 5 \\
0 \quad 10 \quad 1 \quad 20 \quad 2 \quad 20 \quad 3 \quad 40 \quad 4 \quad 10 \\
1 \quad 20 \quad 0 \quad 10 \quad 3 \quad 30 \quad 2 \quad 50 \quad 4 \quad 30 \\
2 \quad 30 \quad 1 \quad 20 \quad 4 \quad 12 \quad 3 \quad 40 \quad 0 \quad 10 \\
4 \quad 50 \quad 3 \quad 30 \quad 2 \quad 15 \quad 0 \quad 20 \quad 1 \quad 15 \\
\]

\[ ++++++++++++++++ \]

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \]
{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}
Demo Example for the Search Space

\[ x \in X \]
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\[ \gamma: X \rightarrow Y \]

\[ y \in Y \]

A simple demo

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4 50</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>30</td>
<td>4</td>
<td>12</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>50</td>
<td>4</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>15</td>
<td>0</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Demo Example for the Search Space

\[ x \in X \]
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\[ \gamma : X \mapsto Y \]

\[ y \in Y \]

A simple demo

<table>
<thead>
<tr>
<th>time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>0</td>
<td>30</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>10</td>
<td>30</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>12</td>
<td>40</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>30</td>
<td>15</td>
<td>20</td>
<td>15</td>
</tr>
</tbody>
</table>

machine

- 3
- 2
- 1
- 0

time
Demo Example for the Search Space

\( x \in X \)
{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}

\( \gamma : X \mapsto Y \)

A simple demo

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>120</td>
<td>0</td>
</tr>
<tr>
<td>330</td>
<td>2</td>
</tr>
<tr>
<td>430</td>
<td>10</td>
</tr>
<tr>
<td>450</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

\( y \in Y \)
Demo Example for the Search Space

\[ x \in X \]
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\[ \gamma : X \rightarrow Y \]

A simple demo
4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \mapsto Y \]

\[
\begin{array}{cccccccc}
0 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
1 & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 4 & 30 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array}
\]

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \]
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\[ \gamma: X \rightarrow Y \]

A simple demo
4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15

**y \in Y**
Demo Example for the Search Space

$x \in X$
{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}

$\gamma : X \mapsto Y$

A simple demo

<table>
<thead>
<tr>
<th>Y</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5</td>
<td></td>
</tr>
<tr>
<td>0 10 1 20 2 20 3 40 4 10</td>
<td></td>
</tr>
<tr>
<td>1 20 0 10 3 30 2 50 4 30</td>
<td></td>
</tr>
<tr>
<td>2 30 1 20 4 12 3 40 0 10</td>
<td></td>
</tr>
<tr>
<td>4 50 3 30 2 15 0 20 1 15</td>
<td></td>
</tr>
</tbody>
</table>

$y \in Y$
Demo Example for the Search Space

\[ x \in X \]
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\( \gamma : X \rightarrow Y \)

A simple demo

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>20</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>40</td>
<td>4</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>12</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \subseteq \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

---

### A simple demo

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>120</th>
<th>20</th>
<th>220</th>
<th>3</th>
<th>40</th>
<th>4</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5</td>
<td>0</td>
<td>10</td>
<td>120</td>
<td>20</td>
<td>220</td>
<td>3</td>
<td>40</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>1 20</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>50</td>
<td>4</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2 30</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>12</td>
<td>3</td>
<td>40</td>
<td>0</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4 50</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>20</td>
<td>1</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

### machine

- 4: 3
- 3: 1
- 2: 2
- 1: 1
- 0: 1

### time

- 0: 1
- 50: 2
- 100: 3
- 150: 4
Demo Example for the Search Space

$x \in X$
{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}

$\gamma : X \mapsto Y$

<table>
<thead>
<tr>
<th></th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10</td>
<td>120</td>
</tr>
<tr>
<td>1 20</td>
<td>0 10</td>
</tr>
<tr>
<td>2 30</td>
<td>1 20</td>
</tr>
<tr>
<td>4 50</td>
<td>2 15</td>
</tr>
</tbody>
</table>

A simple demo
4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15

$y \in Y$
Demo Example for the Search Space

\[x \in X\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}\]

\[\gamma : X \rightarrow Y\]

A simple demo

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>1</td>
<td>20</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

\[y \in Y\]
Demo Example for the Search Space

$x \in X$
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

$\gamma : X \mapsto Y$

$y \in Y$

A simple demo
\begin{array}{cccccccc}
4 & 5 \\
0 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
1 & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 4 & 30 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array}
Demo Example for the Search Space

\[ x \in X \]
{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}

\[ \gamma : X \mapsto Y \]

A simple demo

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>15</td>
</tr>
</tbody>
</table>

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X = \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

A simple demo

\[
\begin{array}{cccccccc}
4 & 5 & 0 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
1 & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 4 & 30 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array}
\]
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

\[ y \in Y \]

A simple demo

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 0 2 2 0 3 4 0 4 1 0</td>
</tr>
<tr>
<td>1</td>
<td>2 0 0 1 0 3 3 0 2 5 0 4 3 0</td>
</tr>
<tr>
<td>2</td>
<td>3 0 1 2 0 4 1 2 3 4 0 0 1 0</td>
</tr>
<tr>
<td>4</td>
<td>5 0 3 3 0 2 1 5 0 2 0 1 1 5</td>
</tr>
</tbody>
</table>

machine

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

time
$x \in X$
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

$\gamma : X \rightarrow Y$

\[\begin{array}{cccccccc}
0 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
1 & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 4 & 30 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array}\]
Demo Example for the Search Space

\[ x \in X \]
{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}\n
\[ \gamma : X \rightarrow Y \]

A simple demo

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>20</td>
<td>2</td>
<td>20</td>
<td>3</td>
<td>40</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>50</td>
<td>4</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>12</td>
<td>3</td>
<td>40</td>
<td>0</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>20</td>
<td>1</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

A simple demo

\[
\begin{array}{cccc}
4 & 5 \\
0 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
1 & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 4 & 30 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array}
\]
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \]
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\[ \gamma : X \mapsto Y \]

\[ y \in Y \]

A simple demo

<table>
<thead>
<tr>
<th>4 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10 1 20 2 20 3 40 4 10</td>
</tr>
<tr>
<td>1 20 0 10 3 30 2 50 4 30</td>
</tr>
<tr>
<td>2 30 1 20 4 12 3 40 0 10</td>
</tr>
<tr>
<td>4 50 3 30 2 15 0 20 1 15</td>
</tr>
</tbody>
</table>

+++

machine

<table>
<thead>
<tr>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

time
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

A simple demo

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine 1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Machine 2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \]
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\[ \gamma : X \rightarrow Y \]

\[ y \in Y \]

A simple demo

<table>
<thead>
<tr>
<th>4 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10 1 20 2 20 3 40 4 10</td>
</tr>
<tr>
<td>1 20 0 10 3 30 2 50 4 30</td>
</tr>
<tr>
<td>2 30 1 20 4 12 3 40 0 10</td>
</tr>
<tr>
<td>4 50 3 30 2 15 0 20 1 15</td>
</tr>
</tbody>
</table>
Demo Example for the Search Space

\[ x \in X \]
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

**\[ \gamma : X \mapsto Y \]**

A simple demo

<table>
<thead>
<tr>
<th>I</th>
<th>4 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10</td>
<td>1 20</td>
</tr>
<tr>
<td>1 20</td>
<td>0 10</td>
</tr>
<tr>
<td>2 30</td>
<td>1 20</td>
</tr>
<tr>
<td>4 50</td>
<td>3 30</td>
</tr>
</tbody>
</table>

**\[ y \in Y \]**
Demo Example for the Search Space

$$x \in X$$
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

$$\gamma : X \rightarrow Y$$

A simple demo

<table>
<thead>
<tr>
<th>1</th>
<th>10</th>
<th>2</th>
<th>20</th>
<th>2</th>
<th>20</th>
<th>3</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>12</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$y \in Y$$
Demo Example for the Search Space

\[ x \in X \]
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\[ \gamma: X \rightarrow Y \]

\[ y \in Y \]

A simple demo

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

+++++++
Demo Example for the Search Space

\[ x \in X \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

\[ y \in Y \]

A simple demo

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>0</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>15</td>
</tr>
</tbody>
</table>

Hence, we have:

\[ X = \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ Y \]

\[ \gamma \]

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \mapsto Y \]

A simple demo
4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \mapsto Y \]

\[ y \in Y \]

A simple demo
4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \]
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\[ \gamma: X \mapsto Y \]

<table>
<thead>
<tr>
<th>I</th>
<th>------</th>
<th>------</th>
<th>------</th>
<th>------</th>
<th>------</th>
<th>------</th>
<th>------</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>20</td>
<td>2</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>12</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>3</td>
<td>30</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

\[ y \in Y \]
Demo Example for the Search Space

$x \in X$
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

$\gamma : X \mapsto Y$

I

A simple demo
4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15

$y \in Y$
Demo Example for the Search Space

\( x \in X \)
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\( \gamma : X \rightarrow Y \)

A simple demo

\[
\begin{array}{cccccccc}
4 & 5 \\
0 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
1 & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 4 & 30 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array}
\]
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \mapsto Y \]

\[ y \in Y \]

A simple demo

<table>
<thead>
<tr>
<th>4 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10</td>
</tr>
<tr>
<td>1 20</td>
</tr>
<tr>
<td>2 30</td>
</tr>
<tr>
<td>4 50</td>
</tr>
</tbody>
</table>

+---------------------------------------------+
Demo Example for the Search Space

\[ x \in X \]
\[
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

A simple demo

\[
\begin{array}{cccccccc}
I & & & & & & & \\
4 & 5 & & & & & & \\
0 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
1 & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 4 & 30 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array}
\]

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \mapsto Y \]

A simple demo

I

4 5
0 10 1 20 2 20 3 40 4 10
1 20 0 10 3 30 2 50 4 30
2 30 1 20 4 12 3 40 0 10
4 50 3 30 2 15 0 20 1 15

+++

machine
time

y \in Y
Demo Example for the Search Space

\[ x \in X \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \mapsto Y \]

A simple demo

\[
\begin{array}{cccccccccc}
4 & 5 & 0 & 1 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
1 & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 4 & 30 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array}
\]
Demo Example for the Search Space

\( x \in X \)
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\( \gamma: X \leftrightarrow Y \)

\( y \in Y \)

A simple demo

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

+++

machine
time
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \mapsto Y \]

A simple demo

\[
\begin{array}{cccccccccc}
4 & 5 & 0 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
1 & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 1 & 20 & 3 & 40 & 1 & 20 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 & 1 & 15 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array}
\]
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \leftrightarrow Y \]

A simple demo

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>1 20</td>
<td>2 20</td>
<td>3 40</td>
</tr>
<tr>
<td>1 20</td>
<td>0 10</td>
<td>3 30</td>
<td>2 50</td>
<td></td>
</tr>
<tr>
<td>2 30</td>
<td>1 20</td>
<td>4 12</td>
<td>3 40</td>
<td>0 10</td>
</tr>
<tr>
<td>4 50</td>
<td>3 30</td>
<td>2 15</td>
<td>0 20</td>
<td>1 15</td>
</tr>
</tbody>
</table>

\[ y \in Y \]
Demo Example for the Search Space

\[ x \in X \]
\{
1, 2, 0, 1, 2, 3, 1, 2, 0, 3,
0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\[ \gamma : X \mapsto Y \]

A simple demo

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10 1 20 2 20 3 40 4 10</td>
<td></td>
</tr>
<tr>
<td>1 20 0 10 3 30 2 50 4 30</td>
<td></td>
</tr>
<tr>
<td>2 30 1 20 4 12 3 40 0 10</td>
<td></td>
</tr>
<tr>
<td>4 50 3 30 2 15 0 20 1 15</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 1</th>
<th>1 2</th>
<th>3 0</th>
<th>0 1</th>
<th>3 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1</td>
<td>50</td>
<td>100</td>
<td>150</td>
<td></td>
</tr>
</tbody>
</table>
Demo Example for the Search Space

\[ x \in X \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \mapsto Y \]

\[ y \in Y \]

A simple demo

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
</tr>
</tbody>
</table>

+---------------------------------------------+
Demo Example for the Search Space

\[ x \in X \]
\[ \{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\} \]

\[ \gamma : X \rightarrow Y \]

\[ y \in Y \]

A simple demo

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>50</td>
<td>3</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>20</td>
<td>1</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>10</td>
<td>3</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>10</td>
<td>2</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

+++

\[ I \]

\[ 4 \ 5 \]

+++

\[ y \in Y \]

\[ 4 \ 30 \]
$x \in X$
\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

$\gamma : X \mapsto Y$

$y \in Y$

A simple demo

<table>
<thead>
<tr>
<th>machine</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

$\mathcal{I}$

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 10</td>
<td>1 20</td>
</tr>
<tr>
<td>2 20</td>
<td>3 40</td>
</tr>
<tr>
<td>4 10</td>
<td>4 30</td>
</tr>
<tr>
<td>1 20</td>
<td>0 10</td>
</tr>
<tr>
<td>3 30</td>
<td>2 50</td>
</tr>
<tr>
<td>4 50</td>
<td>3 40</td>
</tr>
<tr>
<td>0 20</td>
<td>1 15</td>
</tr>
</tbody>
</table>
Demo Example for the Search Space

\( x \in X \)

\{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1\}

\( \gamma : X \mapsto Y \)

\[ \begin{array}{ccccccccccc}
4 & 5 & 0 & 10 & 1 & 20 & 2 & 20 & 3 & 40 & 4 & 10 \\
1 & 20 & 0 & 10 & 3 & 30 & 2 & 50 & 4 & 30 \\
2 & 30 & 1 & 20 & 4 & 12 & 3 & 40 & 0 & 10 \\
4 & 50 & 3 & 30 & 2 & 15 & 0 & 20 & 1 & 15 \\
\end{array} \]

A simple demo
The Search Space $\mathbb{X}$

- We now have search space $\mathbb{X}$ with which we can easily represent all reasonable Gantt charts.
The Search Space $\mathbb{X}$

- We now have search space $\mathbb{X}$ with which we can easily represent all reasonable Gantt charts.
- As long as our integer strings of length $m \times n$ contain each value in $1 \ldots n$ exactly $m$ times, we will always get feasible Gantt charts by applying our mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$!
The Search Space \( \mathbb{X} \)

- We now have search space \( \mathbb{X} \) with which we can easily represent all reasonable Gantt charts.
- As long as our integer strings of length \( m \times n \) contain each value in \( 1 \ldots n \) exactly \( m \) times, we will always get feasible Gantt charts by applying our mapping \( \gamma : \mathbb{X} \mapsto \mathbb{Y} \)!
- We call this the representation.
The Search Space $\mathbb{X}$

• We now have search space $\mathbb{X}$ with which we can easily represent all reasonable Gantt charts.

• As long as our integer strings of length $m \times n$ contain each value in $1 \ldots n$ exactly $m$ times, we will always get feasible Gantt charts by applying our mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$!

• We call this the representation.

• If necessary, we could also easily add more constraints, such as job-order specific machine setup times, or job/machine specific transport times – they would all go into the mapping $\gamma$. 
An Interface for Representation Mappings in Java

```java
package aitoa.structure;

public interface IRepresentationMapping<X, Y> {
    void map(X x, Y y);
}
```
The JSSP Representation Mapping in Java

```java
package aitoa.examples.jssp;

public class JSSPRepresentationMapping {
    //
    // omitted useless stuff, like member variable "instance"
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
    //
```
package aitoa.examples.jssp;

public class JSSPRepresentationMapping implements IRepresentationMapping<int[]>, JSSPCandidateSolution {  
    // omitted useless stuff, like member variable "instance"
    
    // end abridged class
The JSSP Representation Mapping in Java

```java
package aitoa.examples.jssp;

public class JSSPRepresentationMapping implements IRepresentationMapping<int[], JSSPCandidateSolution> {
    // omitted useless stuff, like member variable "instance"
    public void map(int[] x, JSSPCandidateSolution y) {
        // omitted function implementation
    }
} // end abridged class
```
The JSSP Representation Mapping in Java

```java
package aitoa.examples.jssp;

public class JSSPRepresentationMapping
    implements IRepresentationMapping<int[], JSSPCandidateSolution> {
    // omitted useless stuff, like member variable "instance"
    public void map(int[] x, JSSPCandidateSolution y) {
        int[] machineState = new int[this.instance.m]; // These variables can be member
        int[] machineTime = new int[this.instance.m]; // variables.
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
        //
    } // end function
} // end abridged class
```
The JSSP Representation Mapping in Java

```java
package aitoa.examples.jssp;

public class JSSPRepresentationMapping implements IRepresentationMapping<int[], JSSPCandidateSolution> {

    // omitted useless stuff, like member variable "instance"

    public void map(int[] x, JSSPCandidateSolution y) {

        int[] machineState = new int[this.instance.m]; // These variables can be member
        int[] machineTime = new int[this.instance.m]; // variables that only need to be
        int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid
        int[] jobTime = new int[this.instance.n]; // allocating them each time.

    } // end function

} // end abridged class
```
The JSSP Representation Mapping in Java

```java
package aitoa.examples.jssp;

public class JSSPRepresentationMapping
    implements IRepresentationMapping<int[], JSSPCandidateSolution> {
    // omitted useless stuff, like member variable "instance"
    public void map(int[] x, JSSPCandidateSolution y) {
        int[] machineState = new int[this.instance.m]; // These variables can be member
        int[] machineTime = new int[this.instance.m]; // variables that only need to be
        int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid
        int[] jobTime = new int[this.instance.n]; // allocating them each time.

        for (int nextJob : x) { // iterate over job IDs in x
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            //
            // end iteration over job IDs
        } // end function
    } // end abridged class
```
package aitoa.examples.jssp;

public class JSSPRepresentationMapping implements IRepresentationMapping<int[], JSSPCandidateSolution> {
    // omitted useless stuff, like member variable "instance"
    public void map(int[] x, JSSPCandidateSolution y) {
        int[] machineState = new int[this.instance.m]; // These variables can be member
        int[] machineTime = new int[this.instance.m]; // variables that only need to be
        int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid
        int[] jobTime = new int[this.instance.n]; // allocating them each time.

        for (int nextJob : x) { // iterate over job IDs in x
            int[] jobSteps = this.instance.jobs[nextJob]; // get the operations of the job
            //
            //
            //
            //
            //
            //
            //
            //

        } // end iteration over job IDs
    } // end function
} // end abridged class
The JSSP Representation Mapping in Java

```java
package aitoa.examples.jssp;

public class JSSPRepresentationMapping
    implements IRepresentationMapping<int[], JSSPCandidateSolution> {
    // omitted useless stuff, like member variable "instance"
    public void map(int[] x, JSSPCandidateSolution y) {
        int[] machineState = new int[this.instance.m]; // These variables can be member
        int[] machineTime = new int[this.instance.m]; // variables that only need to be
        int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid
        int[] jobTime = new int[this.instance.n]; // allocating them each time.

        for (int nextJob : x) { // iterate over job IDs in x
            int[] jobSteps = this.instance.jobs[nextJob]; // get the operations of the job
            int jobStep = (jobState[nextJob]++) << 1; // 2*(increased job step index)
        } // end iteration over job IDs
    } // end function
} // end abridged class
```
package aitoa.examples.jssp;

public class JSSPRepresentationMapping
    implements IRepresentationMapping<int[], JSSPCandidateSolution> {
    // omitted useless stuff, like member variable "instance"
    public void map(int[] x, JSSPCandidateSolution y) {
        int[] machineState = new int[this.instance.m]; // These variables can be member
        int[] machineTime = new int[this.instance.m]; // variables that only need to be
        int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid
        int[] jobTime = new int[this.instance.n]; // allocating them each time.

        for (int nextJob : x) { // iterate over job IDs in x
            int[] jobSteps = this.instance.jobs[nextJob]; // get the operations of the job
            int jobStep = (jobState[nextJob]++) << 1; // 2*(increased job step index)
            int machine = jobSteps[jobStep]; // get the machine to use
        }
    }
} // end abridged class
The JSSP Representation Mapping in Java

```java
package aitoa.examples.jssp;

public class JSSPRepresentationMapping
    implements IRepresentationMapping<int[], JSSPCandidateSolution> {
    // omitted useless stuff, like member variable "instance"
    public void map(int[] x, JSSPCandidateSolution y) {
        int[] machineState = new int[this.instance.m]; // These variables can be member
        int[] machineTime = new int[this.instance.m]; // variables that only need to be
        int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid
        int[] jobTime = new int[this.instance.n]; // allocating them each time.

        for (int nextJob : x) { // iterate over job IDs in x
            int[] jobSteps = this.instance.jobs[nextJob]; // get the operations of the job
            int jobStep = (jobState[nextJob]++) << 1; // 2*(increased job step index)
            int machine = jobSteps[jobStep]; // get the machine to use

            int start = Math.max(machineTime[machine], jobTime[nextJob]);
            //
            //
            //
            //
        } // end iteration over job IDs
    } // end function
} // end abridged class
```
The JSSP Representation Mapping in Java

```java
package aitoa.examples.jssp;

public class JSSPRepresentationMapping implements IRepresentationMapping<int[], JSSPCandidateSolution> {
    // omitted useless stuff, like member variable "instance"

    public void map(int[] x, JSSPCandidateSolution y) {
        int[] machineState = new int[this.instance.m]; // These variables can be member
        int[] machineTime = new int[this.instance.m]; // variables that only need to be
        int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid
        int[] jobTime = new int[this.instance.n]; // allocating them each time.

        for (int nextJob : x) { // iterate over job IDs in x
            int[] jobSteps = this.instance.jobs[nextJob]; // get the operations of the job
            int jobStep = (jobState[nextJob]++) << 1; // 2*(increased job step index)
            int machine = jobSteps[jobStep]; // get the machine to use

            int start = Math.max(machineTime[machine], jobTime[nextJob]);
            int end = start + jobSteps[jobStep + 1]; // begin + operation time
        } // end iteration over job IDs
    } // end function
} // end abridged class
```
package aitoa.examples.jssp;

public class JSSPRepresentationMapping implements IRepresentationMapping<int[], JSSPCandidateSolution> {
    // omitted useless stuff, like member variable "instance"
    public void map(int[] x, JSSPCandidateSolution y) {
        int[] machineState = new int[this.instance.m]; // These variables can be member
        int[] machineTime = new int[this.instance.m]; // variables that only need to be
        int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid
        int[] jobTime = new int[this.instance.n]; // allocating them each time.

        for (int nextJob : x) { // iterate over job IDs in x
            int[] jobSteps = this.instance.jobs[nextJob]; // get the operations of the job
            int jobStep = (jobState[nextJob]++) << 1; // 2*(increased job step index)
            int machine = jobSteps[jobStep]; // get the machine to use

            int start = Math.max(machineTime[machine], jobTime[nextJob]);
            int end = start + jobSteps[jobStep + 1]; // begin + operation time
        }
    }
    // end function
} // end abridged class
package aitoa.examples.jssp;

public class JSSPRepresentationMapping implements IRepresentationMapping<int[], JSSPCandidateSolution> {
   // omitted useless stuff, like member variable "instance"
   public void map(int[] x, JSSPCandidateSolution y) {
      int[] machineState = new int[this.instance.m]; // These variables can be member
      int[] machineTime = new int[this.instance.m]; // variables that only need to be
      int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid
      int[] jobTime = new int[this.instance.n]; // allocating them each time.

      for (int nextJob : x) { // iterate over job IDs in x
         int[] jobSteps = this.instance.jobs[nextJob]; // get the operations of the job
         int jobStep = (jobState[nextJob]++) << 1; // 2*(increased job step index)
         int machine = jobSteps[jobStep]; // get the machine to use

         int start = Math.max(machineTime[machine], jobTime[nextJob]);
         int end = start + jobSteps[jobStep + 1]; // begin + operation time

         int[] schedule = y.schedule[machine]; // get list of tasks for machine
         //
         //
      } // end iteration over job IDs
   } // end function
} // end abridged class
package aitoa.examples.jssp;

public class JSSPRepresentationMapping implements IRepresentationMapping<br> {<br>    // omitted useless stuff, like member variable "instance"
    public void map(int[] x, JSSPCandidateSolution y) {<br>        int[] machineState = new int[this.instance.m]; // These variables can be member<br>        int[] machineTime = new int[this.instance.m]; // variables that only need to be<br>        int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid<br>        int[] jobTime = new int[this.instance.n]; // allocating them each time.<br>
        for (int nextJob : x) { // iterate over job IDs in x<br>            int[] jobSteps = this.instance.jobs[nextJob]; // get the operations of the job<br>            int jobStep = (jobState[nextJob]++) << 1; // 2*(increased job step index)<br>            int machine = jobSteps[jobStep]; // get the machine to use<br>
            int start = Math.max(machineTime[machine], jobTime[nextJob]);<br>            int end = start + jobSteps[jobStep + 1]; // begin + operation time<br>            jobTime[nextJob] = machineTime[machine] = end;<br>
            int[] schedule = y.schedule[machine]; // get list of tasks for machine<br>            schedule[machineState[machine]++] = nextJob; // store job<br>
        } // end iteration over job IDs<br>    } // end function<br>} // end abridged class
The JSSP Representation Mapping in Java

```java
package aitoa.examples.jssp;

public class JSSPRepresentationMapping implements IRepresentationMapping<int[], JSSPCandidateSolution> {
    // omitted useless stuff, like member variable "instance"
    public void map(int[] x, JSSPCandidateSolution y) {
        int[] machineState = new int[this.instance.m]; // These variables can be member
        int[] machineTime = new int[this.instance.m]; // variables that only need to be
        int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid
        int[] jobTime = new int[this.instance.n]; // allocating them each time.

        for (int nextJob : x) { // iterate over job IDs in x
            int[] jobSteps = this.instance.jobs[nextJob]; // get the operations of the job
            int jobStep = (jobState[nextJob]++) << 1; // 2*(increased job step index)
            int machine = jobSteps[jobStep]; // get the machine to use

            int start = Math.max(machineTime[machine], jobTime[nextJob]);
            int end = start + jobSteps[jobStep + 1]; // begin + operation time

            int[] schedule = y.schedule[machine]; // get list of tasks for machine
            schedule[machineState[machine]++] = nextJob; // store job
            schedule[machineState[machine]++] = start; // store start time
        } // end iteration over job IDs
    } // end function
} // end abridged class
```
package aitoa.examples.jssp;

public class JSSPRepresentationMapping
    implements IRepresentationMapping<int[], JSSPCandidateSolution> {
    // omitted useless stuff, like member variable "instance"
    public void map(int[] x, JSSPCandidateSolution y) {
        int[] machineState = new int[this.instance.m]; // These variables can be member
        int[] machineTime = new int[this.instance.m]; // variables that only need to be
        int[] jobState = new int[this.instance.n]; // filled with 0. Then we avoid
        int[] jobTime = new int[this.instance.n]; // allocating them each time.

        for (int nextJob : x) { // iterate over job IDs in x
            int[] jobSteps = this.instance.jobs[nextJob]; // get the operations of the job
            int jobStep = (jobState[nextJob]++) << 1; // 2*(increased job step index)
            int machine = jobSteps[jobStep]; // get the machine to use

            int start = Math.max(machineTime[machine], jobTime[nextJob]);
            int end = start + jobSteps[jobStep + 1]; // begin + operation time

            int[] schedule = y.schedule[machine]; // get list of tasks for machine
            schedule[machineState[machine]++] = nextJob; // store job
            schedule[machineState[machine]++] = start; // store start time
            schedule[machineState[machine]++] = end; // store end time
        } // end iteration over job IDs
    } // end function
} // end abridged class
Number of Possible Solutions
Number of Solutions: Size of $\mathcal{Y}$

• OK, we want to solve a JSSP instance
Number of Solutions: Size of $\Upsilon$

- OK, we want to solve a JSSP instance
- How many possible candidate solutions are there?
Number of Solutions: Size of $\mathcal{Y}$

- OK, we want to solve a JSSP instance
- How many possible candidate solutions are there?
- If we allow arbitrary useless waiting times between jobs, then we could create arbitrarily many different valid Gantt charts for any problem instance.
Number of Solutions: Size of $Y$

- OK, we want to solve a JSSP instance
- How many possible candidate solutions are there?
- If we allow arbitrary useless waiting times between jobs, then we could create arbitrarily many different valid Gantt charts for any problem instance.
- Let us assume that no time is wasted by waiting unnecessarily – which is what our search space representation does, too.
Number of Solutions: Size of $\mathbb{Y}$

- OK, we want to solve a JSSP instance
- How many possible candidate solutions are there?
- If we allow arbitrary useless waiting times between jobs, then we could create arbitrarily many different valid Gantt charts for any problem instance.
- Let us assume that no time is wasted by waiting unnecessarily – which is what our search space representation does, too.
- If there was only 1 machine, then we would have $n! = 1 \times 2 \times 3 \times 4 \times 5 \times \ldots \times n$ possible ways to arrange the $n$ jobs.
OK, we want to solve a JSSP instance
How many possible candidate solutions are there?
If we allow arbitrary useless waiting times between jobs, then we could create arbitrarily many different valid Gantt charts for any problem instance.
Let us assume that no time is wasted by waiting unnecessarily – which is what our search space representation does, too.
If there was only 1 machine, then we would have
\[ n! = 1 \times 2 \times 3 \times 4 \times 5 \times \ldots \times n \] possible ways to arrange the \( n \) jobs.
If there are 2 machines, this gives us \((n!) \times (n!) = (n!)^2\) choices.
Number of Solutions: Size of $\mathcal{Y}$

- OK, we want to solve a JSSP instance
- How many possible candidate solutions are there?
- If we allow arbitrary useless waiting times between jobs, then we could create arbitrarily many different valid Gantt charts for any problem instance.
- Let us assume that no time is wasted by waiting unnecessarily – which is what our search space representation does, too.
- If there was only 1 machine, then we would have $n! = 1 \times 2 \times 3 \times 4 \times 5 \times \ldots \times n$ possible ways to arrange the $n$ jobs.
- If there are 2 machines, this gives us $(n!) \times (n!) = (n!)^2$ choices.
- For three machines, we are at $(n!)^3$. 
Number of Solutions: Size of $\mathbb{Y}$

- OK, we want to solve a JSSP instance
- How many possible candidate solutions are there?
- If we allow arbitrary useless waiting times between jobs, then we could create arbitrarily many different valid Gantt charts for any problem instance.
- Let us assume that no time is wasted by waiting unnecessarily – which is what our search space representation does, too.
- If there was only 1 machine, then we would have $n! = 1 \times 2 \times 3 \times 4 \times 5 \times \ldots \times n$ possible ways to arrange the $n$ jobs.
- If there are 2 machines, this gives us $(n!) \times (n!) = (n!)^2$ choices.
- For $m$ machines, we are at $(n!)^m$ possible solutions.
Number of Solutions: Size of $\mathcal{Y}$

- OK, we want to solve a JSSP instance
- How many possible candidate solutions are there?
- If we allow arbitrary useless waiting times between jobs, then we could create arbitrarily many different valid Gantt charts for any problem instance.
- Let us assume that no time is wasted by waiting unnecessarily – which is what our search space representation does, too.
- If there was only 1 machine, then we would have $n! = 1 \times 2 \times 3 \times 4 \times 5 \times \ldots \times n$ possible ways to arrange the $n$ jobs.
- If there are 2 machines, this gives us $(n!) \times (n!) = (n!)^2$ choices.
- For $m$ machines, we are at $(n!)^m$ possible solutions.
- But some may be wrong, i.e., contain deadlocks!
### Number of Solutions: Size of $\mathbb{Y}$

| name | $n$ | $m$ | $\min(\#\text{feasible})$ | $|\mathbb{Y}|$ |
|------|-----|-----|-----------------------------|--------------|
|      | 2   | 2   | 3                           | 4            |
Number of Solutions: Size of $\mathbb{Y}$

| name | $n$ | $m$ | $\text{min}(\# \text{feasible})$ | $|\mathbb{Y}|$ |
|------|-----|-----|-------------------------------|------------|
|      | 2   | 2   | 3                            | 4          |
|      | 2   | 3   | 4                            | 8          |
### Number of Solutions: Size of $\mathbb{Y}$

| name | $n$ | $m$ | $\min(\#\text{feasible})$ | $|\mathbb{Y}|$ |
|------|----|----|----------------|--------|
| 2    | 2  | 3  | 3              | 4      |
| 2    | 3  | 4  | 4              | 8      |
| 2    | 4  | 5  | 5              | 16     |
### Number of Solutions: Size of $\mathbb{Y}$

| name | $n$ | $m$ | $\min(\#\text{feasible})$ | $|\mathbb{Y}|$ |
|------|----|----|---------------------|--------|
| 2    | 2  | 2  | 3                   | 4      |
| 2    | 3  | 3  | 4                   | 8      |
| 2    | 4  | 4  | 5                   | 16     |
| 2    | 5  | 5  | 6                   | 32     |
## Number of Solutions: Size of $\mathbb{Y}$

| name | $n$ | $m$ | min(#feasible) | $|\mathbb{Y}|$ |
|------|-----|-----|---------------|-------------|
| 2    | 2   |     | 3             | 4           |
| 2    | 3   |     | 4             | 8           |
| 2    | 4   |     | 5             | 16          |
| 2    | 5   |     | 6             | 32          |
| 3    | 2   |     | 22            | 36          |
| 3    | 3   |     | 63            | 216         |
| 3    | 4   |     | 147           | 1'296       |
| 3    | 5   |     | 317           | 7'776       |
| 4    | 2   |     | 244           | 576         |
| 4    | 3   |     | 1'630         | 13'824      |
| 4    | 4   |     | 7'451         | 331'776     |
## Number of Solutions: Size of $\mathbb{Y}$

| name   | $n$ | $m$ | min(#feasible) | $|\mathbb{Y}|$ |
|--------|-----|-----|----------------|--------|
| 2      | 2   | 2   | 3              | 4      |
| 2      | 3   | 4   | 8              |        |
| 2      | 4   | 5   | 16             |        |
| 2      | 5   | 6   | 32             |        |
| 3      | 2   | 22  | 36             |        |
| 3      | 3   | 63  | 216            |        |
| 3      | 4   | 147 | 1'296          |        |
| 3      | 5   | 317 | 7'776          |        |
| 4      | 2   | 244 | 576            |        |
| 4      | 3   | 1'630 | 13'824       |        |
| 4      | 4   | 7'451 | 331'776      |        |
| demo   | 4   | 5   | 7'962'624      |        |
| la24   | 15  | 10  | $\approx 1.462 \times 10^{121}$ | |
| abz7   | 20  | 15  | $\approx 6.193 \times 10^{275}$ | |
| yn4    | 20  | 20  | $\approx 5.278 \times 10^{367}$ | |
| swv15  | 50  | 10  | $\approx 6.772 \times 10^{644}$ | |
Size of Search Space $X$

• Our search space $X$ is not the same as the solution space $Y$. 
Size of Search Space $X$

- Our search space $X$ is not the same as the solution space $Y$.
- How many points are in our representations of the solution space?
### Size of Search Space

| name   | \( n \) | \( m \) | \(| \mathcal{Y} | \) | \(| \mathcal{X} | \) |
|--------|--------|--------|--------|--------|
| 3 2    | 36     | 90     |
| 3 3    | 216    | 1'680  |
| 3 4    | 1'296  | 34'650 |
| 3 5    | 7'776  | 756'756|
| 4 2    | 576    | 2'520  |
| 4 3    | 13'824 | 369'600|
| 4 4    | 331'776| 63'063'000|
| 5 2    | 14'400 | 113'400|
| 5 3    | 1'728'000 | 168'168'000|
| 5 4    | 207'360'000 | 305'540'235'000|
| 5 5    | 24'883'200'000 | 623'360'743'125'120|
| demo 4 5 | 7'962'624 | 11'732'745'024 |
| la24 15 10 | \( \approx 1.462 \times 10^{121} \) | \( \approx 2.293 \times 10^{164} \) |
| abz7 20 15 | \( \approx 6.193 \times 10^{275} \) | \( \approx 1.432 \times 10^{372} \) |
| yn4 20 20 | \( \approx 5.278 \times 10^{367} \) | \( \approx 1.213 \times 10^{501} \) |
| swv15 50 10 | \( \approx 6.772 \times 10^{644} \) | \( \approx 1.254 \times 10^{806} \) |
Size of Search Space $|X|$
Size of Search Space $\mathbb{X}$

- Our search space $\mathbb{X}$ is not the same as the solution space $\mathbb{Y}$.
- How many points are in our representations of the solution space?
- Both $\mathbb{X}$ and $\mathbb{Y}$ are very big for any relevant problem size.
Size of Search Space $X$

- Our search space $X$ is not the same as the solution space $Y$.
- How many points are in our representations of the solution space?
- Both $X$ and $Y$ are very big for any relevant problem size.
- $X$ is bigger, we pay with size for the simplicity and the avoidance of infeasible solutions.
Search Operators
Search Operators

• Another general structure element needed by many optimization algorithms are search operators.
Search Operators

• Another general structure element needed by many optimization algorithms are search operators.

Definition

An $k$-ary search operator $\text{searchOp} : \mathbb{X}^k \rightarrow \mathbb{X}$ is a left-total relation which accepts $k$ points in the search space $\mathbb{X}$ as input and returns one point in the search space as output.
Search Operators

• Another general structure element needed by many optimization algorithms are search operators.

**Definition**

Search Operator

An $k$-ary search operator $\text{searchOp} : \mathbb{X}^k \mapsto \mathbb{X}$ is a left-total relation which accepts $k$ points in the search space $\mathbb{X}$ as input and returns one point in the search space as output.

• Based on their arity $k$, we can distinguish the following most common operator types:
Search Operators

• Another general structure element needed by many optimization algorithms are search operators.

Definition

An \( k \)-ary search operator \( \text{searchOp} : \mathbb{X}^k \mapsto \mathbb{X} \) is a left-total relation which accepts \( k \) points in the search space \( \mathbb{X} \) as input and returns one point in the search space as output.

- Based on their arity \( k \), we can distinguish the following most common operator types:
  - nullary operators \( (k = 0) \) generate one (random) point in \( \mathbb{X} \).
Search Operators

- Another general structure element needed by many optimization algorithms are search operators.

**Definition**

A $k$-ary search operator $\text{searchOp} : X^k \rightarrow X$ is a left-total relation which accepts $k$ points in the search space $X$ as input and returns one point in the search space as output.

```java
package aitoa.structure;

public interface INullarySearchOperator<X> {
    void apply(X dest, Random random);
}
```
Search Operators

- Another general structure element needed by many optimization algorithms are search operators.

**Definition**

A \( k \)-ary search operator \( \text{searchOp} : X^k \rightarrow X \) is a left-total relation which accepts \( k \) points in the search space \( X \) as input and returns one point in the search space as output.

- Based on their arity \( k \), we can distinguish the following most common operator types:
  - nullary operators (\( k = 0 \)) generate one (random) point in \( X \).
  - unary operators (\( k = 1 \)) take one point from \( X \) as input and return another (similar) point.
Search Operators

- Another general structure element needed by many optimization algorithms are search operators.

**Definition**

Search Operator

An \( k \)-ary search operator \( \text{searchOp} : \mathbb{X}^k \rightarrow \mathbb{X} \) is a left-total relation which accepts \( k \) points in the search space \( \mathbb{X} \) as input and returns one point in the search space as output.

```java
package aitoa.structure;

public interface IUnarySearchOperator<X> {

    void apply(X x, X dest, Random random);
}
```
Search Operators

- Another general structure element needed by many optimization algorithms are search operators.

**Definition**

A \( k \)-ary search operator \( \text{searchOp} : \mathbb{X}^k \rightarrow \mathbb{X} \) is a left-total relation which accepts \( k \) points in the search space \( \mathbb{X} \) as input and returns one point in the search space as output.

- Based on their arity \( k \), we can distinguish the following most common operator types:
  - nullary operators \((k = 0)\) generate one (random) point in \( \mathbb{X} \).
  - unary operators \((k = 1)\) take one point from \( \mathbb{X} \) as input and return another (similar) point.
  - binary operators \((k = 2)\) take two points from \( \mathbb{X} \) as input and return another point which should be similar to both.
Search Operators

- Another general structure element needed by many optimization algorithms are search operators.

Definition

Search Operator

An $k$-ary search operator $\text{searchOp} : \mathbb{X}^k \mapsto \mathbb{X}$ is a left-total relation which accepts $k$ points in the search space $\mathbb{X}$ as input and returns one point in the search space as output.

```java
class IBinarySearchOperator<X> { 
    void apply(X x0, X x1, X dest, Random random); 
}
```
Search Operators

• Another general structure element needed by many optimization algorithms are search operators.

Definition

An \( k \)-ary search operator \( \text{searchOp} : \mathbb{X}^k \rightarrow \mathbb{X} \) is a left-total relation which accepts \( k \) points in the search space \( \mathbb{X} \) as input and returns one point in the search space as output.

• Based on their arity \( k \), we can distinguish the following most common operator types:
  • nullary operators \( (k = 0) \) generate one (random) point in \( \mathbb{X} \).
  • unary operators \( (k = 1) \) take one point from \( \mathbb{X} \) as input and return another (similar) point.
  • binary operators \( (k = 2) \) take two points from \( \mathbb{X} \) as input and return another point which should be similar to both.

• We will discuss concrete implementations of the operators later.
Termination
Searching and Stopping

• Eventually, we will have a program that uses the search operators efficiently to find elements in the set $\mathbb{X}$ which correspond to good solutions in $\mathbb{Y}$. 
Searching and Stopping

• Eventually, we will have a program that uses the search operators efficiently to find elements in the set $\mathcal{X}$ which correspond to good solutions in $\mathcal{Y}$.
• How long should it run?
Searching and Stopping

• Eventually, we will have a program that uses the search operators efficiently to find elements in the set $X$ which correspond to good solutions in $Y$.

• How long should it run?

• When can it stop?
Searching and Stopping

• Eventually, we will have a program that uses the search operators efficiently to find elements in the set $X$ which correspond to good solutions in $Y$.

• How long should it run?

• When can it stop?

• This is called the termination criterion.
When to stop?

• We assume that a human operator receives the job information, enters them into a computer (as JSSP instance), and then goes to drink a coffee.
When to stop?

• We assume that a human operator receives the job information, enters them into a computer (as JSSP instance), and then goes to drink a coffee.
• Can we solve larger, hard JSSPs with such huge numbers of potential solutions until she comes back?
When to stop?

- We assume that a human operator receives the job information, enters them into a computer (as JSSP instance), and then goes to drink a coffee.
- Can we solve larger, hard JSSPs with such huge numbers of potential solutions until she comes back?
- Probably not.
When to stop?

- We assume that a human operator receives the job information, enters them into a computer (as JSSP instance), and then goes to drink a coffee.
- Can we solve larger, hard JSSPs with such huge numbers of potential solutions until she comes back?
- Probably not.
- The best algorithms guaranteeing to find the optimal solution for our JSSPs may need a runtime growing exponential with $m$ and $n$.\textsuperscript{6,25}
When to stop?

• We assume that a human operator receives the job information, enters them into a computer (as JSSP instance), and then goes to drink a coffee.
• Can we solve larger, hard JSSPs with such huge numbers of potential solutions until she comes back?
• Probably not.
• The best algorithms guaranteeing to find the optimal solution for our JSSPs may need a runtime growing exponential with \(m\) and \(n\).\(^6\)\(^{25}\)
• Even algorithms that just guarantee to be a constant factor worse than the optimum (like, 1% longer, 10 times longer... ) cannot faster on the JSSP in the worst case!\(^{26-28}\)
When to stop?

- We assume that a human operator receives the job information, enters them into a computer (as JSSP instance), and then goes to drink a coffee.
- Can we solve larger, hard JSSPs with such huge numbers of potential solutions until she comes back?
- Probably not.
- The best algorithms guaranteeing to find the optimal solution for our JSSPs may need a runtime growing exponential with $m$ and $n$.\textsuperscript{6}\textsuperscript{25}
- Even algorithms that just guarantee to be a constant factor worse than the optimum (like, 1% longer, 10 times longer...) cannot faster on the JSSP in the worst case!\textsuperscript{26–28}
- So?
When to stop?

- We assume that a human operator receives the job information, enters them into a computer (as JSSP instance), and then goes to drink a coffee.
- Can we solve larger, hard JSSPs with such huge numbers of potential solutions until she comes back?
- Probably not.
- The best algorithms guaranteeing to find the optimal solution for our JSSPs may need a runtime growing exponential with $m$ and $n$. Even algorithms that just guarantee to be a constant factor worse than the optimum (like, 1% longer, 10 times longer...) cannot faster on the JSSP in the worst case!
- So? ... The operator drinks a coffee.
When to stop?

- We assume that a human operator receives the job information, enters them into a computer (as JSSP instance), and then goes to drink a coffee.
- Can we solve larger, hard JSSPs with such huge numbers of potential solutions until she comes back?
- Probably not.
- The best algorithms guaranteeing to find the optimal solution for our JSSPs may need a runtime growing exponential with \( m \) and \( n \).\(^{6,25}\)
- Even algorithms that just guarantee to be a constant factor worse than the optimum (like, 1% longer, 10 times longer...) cannot faster on the JSSP in the worst case!\(^{26-28}\)
- So? ... The operator drinks a coffee. ... We have about three minutes.
When to stop?

- We assume that a human operator receives the job information, enters them into a computer (as JSSP instance), and then goes to drink a coffee.
- Can we solve larger, hard JSSPs with such huge numbers of potential solutions until she comes back?
- Probably not.
- The best algorithms guaranteeing to find the optimal solution for our JSSPs may need a runtime growing exponential with $m$ and $n$.\textsuperscript{6,25}
- Even algorithms that just guarantee to be a constant factor worse than the optimum (like, 1\% longer, 10 times longer...) cannot faster on the JSSP in the worst case!\textsuperscript{26–28}
- So? ... The operator drinks a coffee. ... We have about three minutes. ... Let's look for the algorithm implementation that can give us the best solution quality within that time window.
When to stop?

• We assume that a human operator receives the job information, enters them into a computer (as JSSP instance), and then goes to drink a coffee.
• Can we solve larger, hard JSSPs with such huge numbers of potential solutions until she comes back?
• Probably not.
• The best algorithms guaranteeing to find the optimal solution for our JSSPs may need a runtime growing exponential with \( m \) and \( n \).\(^6_{25}\)
• Even algorithms that just guarantee to be a constant factor worse than the optimum (like, 1% longer, 10 times longer...) cannot faster on the JSSP in the worst case!\(^{26–28}\)
• So? ... The operator drinks a coffee. ... We have about three minutes. ... Let’s look for the algorithm implementation that can give us the best solution quality within that time window.
• This is the termination criterion we will use on our JSSP example problem in this lecture.
When to stop?

• We assume that a human operator receives the job information, enters them into a computer (as JSSP instance), and then goes to drink a coffee.
• Can we solve larger, hard JSSPs with such huge numbers of potential solutions until she comes back?
• Probably not.
• The best algorithms guaranteeing to find the optimal solution for our JSSPs may need a runtime growing exponential with $m$ and $n$.\textsuperscript{6,25}
• Even algorithms that just guarantee to be a constant factor worse than the optimum (like, 1% longer, 10 times longer...) cannot faster on the JSSP in the worst case!\textsuperscript{26–28}
• So? ... The operator drinks a coffee. ... We have a about three minutes. ... Let’s look for the algorithm implementation that can give us the best solution quality within that time window.
• This is the termination criterion we will use on our JSSP example problem in this lecture.
• Obviously, in other scenarios, there might be vastly different criteria...
Summary
Summary

- This was the most complicated lesson in this course!
Summary

- This was the most complicated lesson in this course!
- Thank you for sticking with me during this.
Summary

• This was the most complicated lesson in this course!
• Thank you for sticking with me during this.
• What we have learned is the most basic process when attacking any optimization problem!
Summary

• This was the most complicated lesson in this course!
• Thank you for sticking with me during this.
• What we have learned is the most basic process when attacking any optimization problem:
  1. Understand how the scenario / input data is defined!
Summary

• This was the most complicated lesson in this course!
• Thank you for sticking with me during this.
• What we have learned is the most basic process when attacking any optimization problem:
  1. Understand how the scenario / input data is defined!
  2. Make a data structure $Y$ for the solutions, which can contain all the information that the end user needs and considers as a full solution to the problem!
Summary

• This was the most complicated lesson in this course!
• Thank you for sticking with me during this.
• What we have learned is the most basic process when attacking any optimization problem:
  1. Understand how the scenario / input data is defined!
  2. Make a data structure $\mathbb{Y}$ for the solutions, which can contain all the information that the end user needs and considers as a full solution to the problem!
  3. Define the objective function $f$, which rates how good a solution is!
Summary

• This was the most complicated lesson in this course!
• Thank you for sticking with me during this.
• What we have learned is the most basic process when attacking any optimization problem:
  1. Understand how the scenario / input data is defined!
  2. Make a data structure $Y$ for the solutions, which can contain all the information that the end user needs and considers as a full solution to the problem!
  3. Define the objective function $f$, which rates how good a solution is!
  4. Is $Y$ easy to understand and to process by an algorithm?
Summary

• This was the most complicated lesson in this course!
• Thank you for sticking with me during this.
• What we have learned is the most basic process when attacking any optimization problem:
  1. Understand how the scenario / input data is defined!
  2. Make a data structure $Y$ for the solutions, which can contain all the information that the end user needs and considers as a full solution to the problem!
  3. Define the objective function $f$, which rates how good a solution is!
  4. Is $Y$ easy to understand and to process by an algorithm? If yes: cool.
Summary

• This was the most complicated lesson in this course!
• Thank you for sticking with me during this.
• What we have learned is the most basic process when attacking any optimization problem:
  1. Understand how the scenario / input data is defined!
  2. Make a data structure $\mathbb{Y}$ for the solutions, which can contain all the information that the end user needs and considers as a full solution to the problem!
  3. Define the objective function $f$, which rates how good a solution is!
  4. Is $\mathbb{Y}$ easy to understand and to process by an algorithm? If yes: cool. If no: define a simple data structure $\mathbb{X}$ and a translation $\gamma$ from $\mathbb{X}$ to $\mathbb{Y}$!
Summary

- This was the most complicated lesson in this course!
- Thank you for sticking with me during this.
- What we have learned is the most basic process when attacking any optimization problem:
  1. Understand how the scenario / input data is defined!
  2. Make a data structure $\mathbb{Y}$ for the solutions, which can contain all the information that the end user needs and considers as a full solution to the problem!
  3. Define the objective function $f$, which rates how good a solution is!
  4. Is $\mathbb{Y}$ easy to understand and to process by an algorithm? If yes: cool. If no: define a simple data structure $\mathbb{X}$ and a translation $\gamma$ from $\mathbb{X}$ to $\mathbb{Y}$!
  5. Understand when we need to stop the search!
Summary

• This was the most complicated lesson in this course!
• Thank you for sticking with me during this.
• What we have learned is the most basic process when attacking any optimization problem:
  1. Understand how the scenario / input data is defined!
  2. Make a data structure $Y$ for the solutions, which can contain all the information that the end user needs and considers as a full solution to the problem!
  3. Define the objective function $f$, which rates how good a solution is!
  4. Is $Y$ easy to understand and to process by an algorithm? If yes: cool. If no: define a simple data structure $X$ and a translation $\gamma$ from $X$ to $Y$!
  5. Understand when we need to stop the search!
• If we have this, we can directly use most of the algorithms in the rest of the lecture (almost) as-is.
Summary

- We now have the basic tools to search and find solutions for the JSSP.
Summary

- We now have the basic tools to search and find solutions for the JSSP.
- Many other problems are similar and can be represented in a similar way.
Summary

• We now have the basic tools to search and find solutions for the JSSP.
• Many other problems are similar and can be represented in a similar way.
• The key is often to translate the complicated task to work with a complex data structure $\mathbb{Y}$ (e.g., Gantt diagram with many constraints) to a simpler scenario where I only need to deal with a basic data structure $\mathbb{X}$ (like a list of integer numbers with few constraints) by putting the “complicated” rules into a mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$. 
Summary

• We now have the basic tools to search and find solutions for the JSSP.
• Many other problems are similar and can be represented in a similar way.
• The key is often to translate the complicated task to work with a complex data structure $\mathbb{Y}$ (e.g., Gantt diagram with many constraints) to a simpler scenario where I only need to deal with a basic data structure $\mathbb{X}$ (like a list of integer numbers with few constraints) by putting the “complicated” rules into a mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$.
• If I can do that, then from now on I do not need to worry about $\mathbb{Y}$ and its rules any more – I only need to work with $\mathbb{X}$, which is easier to understand and to program.
Summary

- We now have the basic tools to search and find solutions for the JSSP.
- Many other problems are similar and can be represented in a similar way.
- The key is often to translate the complicated task to work with a complex data structure $\mathbb{Y}$ (e.g., Gantt diagram with many constraints) to a simpler scenario where I only need to deal with a basic data structure $\mathbb{X}$ (like a list of integer numbers with few constraints) by putting the “complicated” rules into a mapping $\gamma : \mathbb{X} \mapsto \mathbb{Y}$.
- If I can do that, then from now on I do not need to worry about $\mathbb{Y}$ and its rules any more – I only need to work with $\mathbb{X}$, which is easier to understand and to program.
- Let us now try to solve the JSSP using metaheuristics that search inside $\mathbb{X}$ (and thus can find solutions in $\mathbb{Y}$).
Summary

• We now have the basic tools to search and find solutions for the JSSP.
• Many other problems are similar and can be represented in a similar way.
• The key is often to translate the complicated task to work with a complex data structure $\mathbb{Y}$ (e.g., Gantt diagram with many constraints) to a simpler scenario where I only need to deal with a basic data structure $\mathbb{X}$ (like a list of integer numbers with few constraints) by putting the “complicated” rules into a mapping $\gamma: \mathbb{X} \mapsto \mathbb{Y}$.
• If I can do that, then from now on I do not need to worry about $\mathbb{Y}$ and its rules any more – I only need to work with $\mathbb{X}$, which is easier to understand and to program.
• Let us now try to solve the JSSP using metaheuristics that search inside $\mathbb{X}$ (and thus can find solutions in $\mathbb{Y}$ within 3 minutes).
谢谢
Thank you
References I


References II


