合肥学院
HEFEI UNIVERSITY

Optimization Algorithms

2. Structure

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

Institute of Applied Optimization (IAO) | 应用优化研究所
School of Artificial Intelligence and Big Data | 人工智能与大数据学院
Hefei University | 合肥学院
Hefei, Anhui, China | 中国安徽省合肥市

Outline

Introduction

The Structure of Optimization

- So we know roughly what an optimization problem is and that metaheuristics[1–4] are algorithms to solve them.

The Structure of Optimization

- So we know roughly what an optimization problem is and that metaheuristics[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[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[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.

The Structure of Optimization

- So we know roughly what an optimization problem is and that metaheuristics[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 $\mathcal{I}$ 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 $\mathcal{I}$ 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 $\mathbb{Y}$ for the candidate solutions $y \in \mathbb{Y}$, 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 $\mathbb{Y}$ 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

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 $\mathrm{searchOp} : \mathbb{X}^n \mapsto \mathbb{X}$, 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 $\mathrm{searchOp} : \mathbb{X}^n \mapsto \mathbb{X}$, 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 $\mathcal{I}$ 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 $\mathrm{searchOp} : \mathbb{X}^n \mapsto \mathbb{X}$, 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 $\mathrm{searchOp} : \mathbb{X}^n \mapsto \mathbb{X}$, and
  7. a termination criterion.
- Looks complicated, but don't worry..

Components of an Optimization Problem

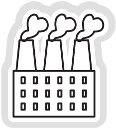
- From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $\mathcal{I}$ to be solved
  2. a data type $\mathbb{Y}$ for the candidate solutions $y \in \mathbb{Y}$, and
  3. an objective function $f : \mathbb{Y} \mapsto \mathbb{R}$.
- 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 $\mathrm{searchOp} : \mathbb{X}^n \mapsto \mathbb{X}$, and
  7. a termination criterion.
- Looks complicated, but don't worry. We will do this one-by-one.

Components of an Optimization Problem

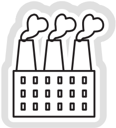
- From the perspective of a programmer, we can say that an optimization problem has the following components:
  1. the input data which specifies the problem instance $\mathcal{I}$ to be solved
  2. a data type $\mathbb{Y}$ for the candidate solutions $y \in \mathbb{Y}$, and
  3. an objective function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

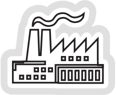
Example Problem: Job Shop Scheduling

Job Shop Problem

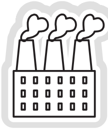


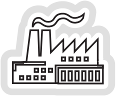
Job Shop Problem



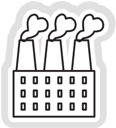


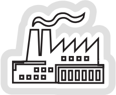
Job Shop Problem



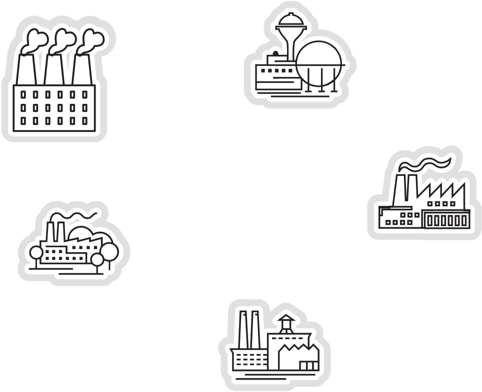


Job Shop Problem

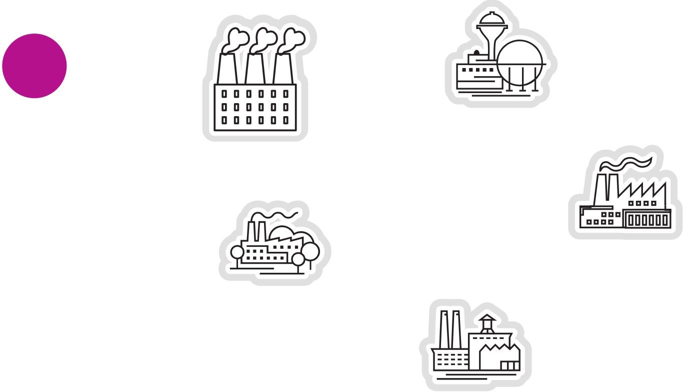




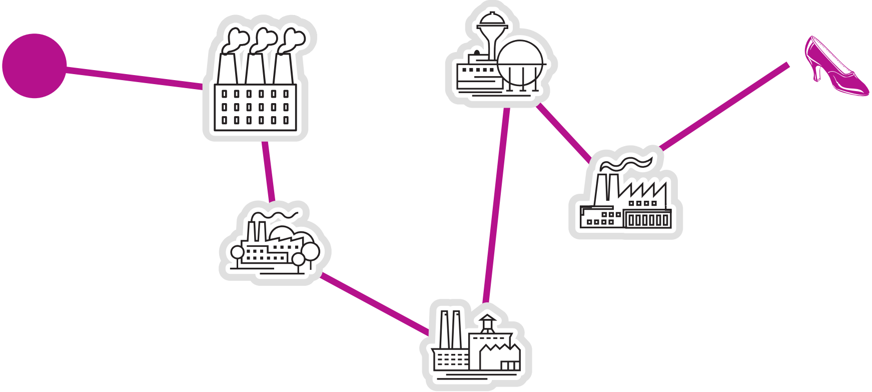

Job Shop Problem



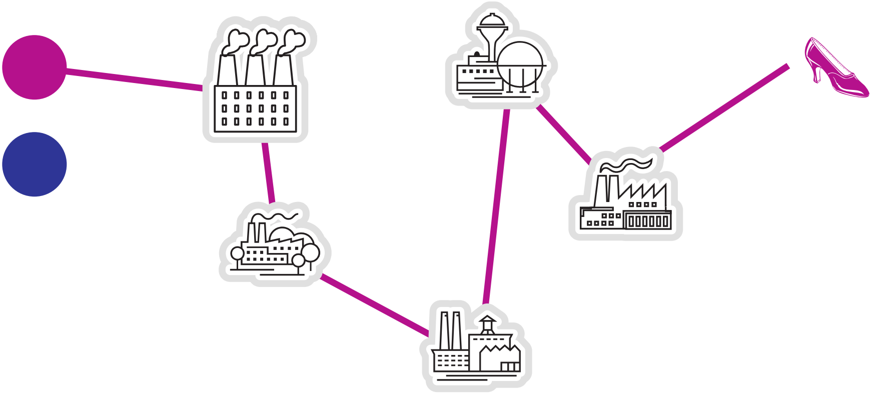
Job Shop Problem



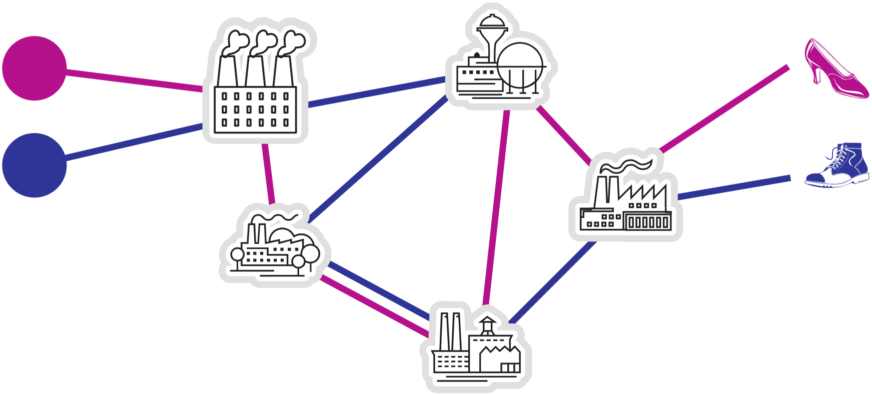
Job Shop Problem



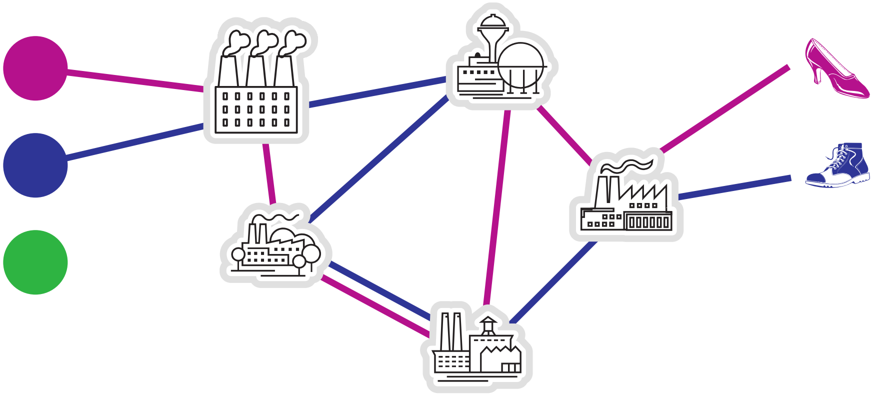
Job Shop Problem



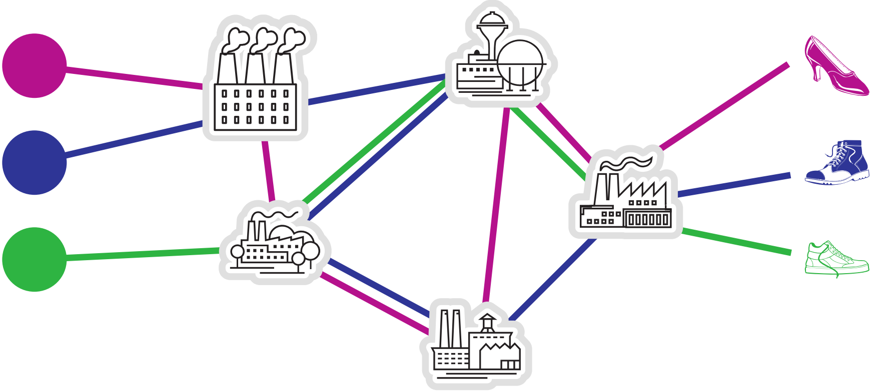
Job Shop Problem



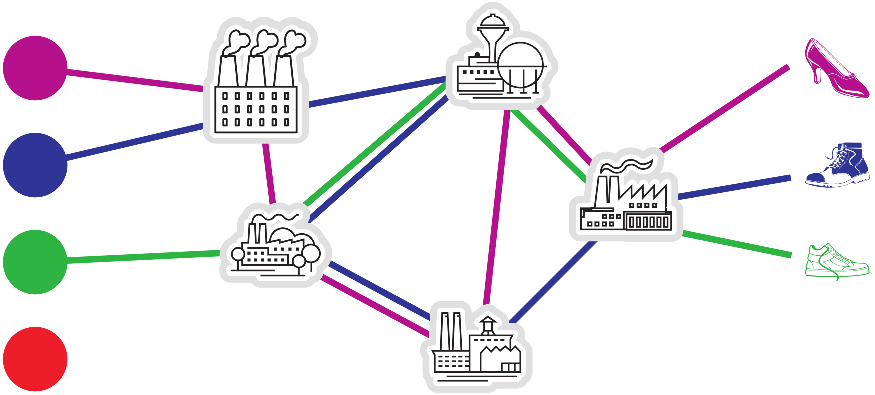
Job Shop Problem



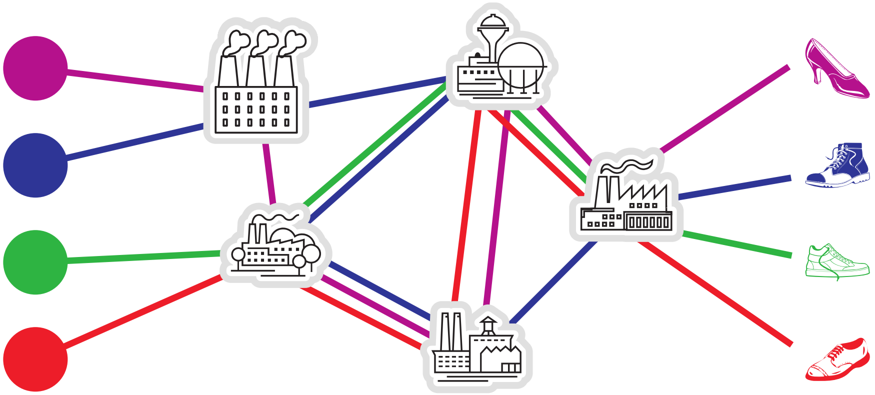
Job Shop Problem



Job Shop Problem



Job Shop Problem



Job Shop Scheduling Problem

- The Job Shop Scheduling Problem (JSSP)[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.

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.

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.

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.
- Also, each job will require a job-specific time at a given machine.

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.
- Also, each job will require a job-specific time at a given machine.
- The goal is to fulfill all tasks as quickly as possible.

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.
- 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."

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.
- 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 $\mathcal{I}$.
- 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`.
- More information about these instances has been collected by van Hoorn[13 14] 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[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[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

number n of jobs

```
++++++++++++++++++++++++
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

number m of machines

number n of jobs

```
+++++++++++++++++++++++
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

number m of machines

number n of jobs

job 0

```
++++++++++++++++++++++++
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

number m of machines

number n of jobs

```
+++++++++++++++++++++++++
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
+++++++++++++++++++++++++
```

job 1

Demo Instance

number m of machines

number n of jobs

```
+++++++++++++++++++++++++
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
+++++++++++++++++++++++++
```

job 2

Demo Instance

number m of machines

number n of jobs

```
+++++++++++++++++++++++++
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
+++++++++++++++++++++++++
```

job 3

Demo Instance

number m of machines

number n of jobs

job 0

```
++++++++++++++++++++++++
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

number m of machines

number n of jobs

```
++++++++++++++++++++++++
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
++++++++++++++++++++++++
```

job 0

Job 0 first needs to be processed by machine 0 for 10 time units

Demo Instance

number m of machines

number n of jobs

job 0

```
+++++++++++++++++++++++++
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
+++++++++++++++++++++++++
```

Job 0 first needs to be processed by machine 0 for 10 time units, it then goes to machine 1 for 20 time units

Demo Instance

number m of machines

number n of jobs

job 0

```
+++++++++++++++++++++++
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
+++++++++++++++++++++++
```

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

Demo Instance

number m of machines

number n of jobs

job 0

```
++++++++++++++++++++++++
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
++++++++++++++++++++++++
```

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

Demo Instance

number m of machines

number n of jobs

job 0

```
+++++++++++++++++++++++++
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
+++++++++++++++++++++++++
```

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.

Demo Instance

number m of machines

number n of jobs

```
+++++++++++++++++++++++++
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
+++++++++++++++++++++++++
```

job 1

Similarly, Job 1 first needs to be processed by machine 1 for 20 time units

Demo Instance

number m of machines

number n of jobs

```
+++++++++++++++++++++++++
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
+++++++++++++++++++++++++
```

job 1

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

Demo Instance

number m of machines

number n of jobs

```
++++++++++++++++++++++++
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
++++++++++++++++++++++++
```

job 1

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

Demo Instance

number m of machines

number n of jobs

job 1

```
++++++++++++++++++++++++
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
++++++++++++++++++++++++
```

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

Demo Instance

number m of machines

number n of jobs

job 1

```
+++++++++++++++++++++++++
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
+++++++++++++++++++++++++
```

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.

Demo Instance

number m of machines

number n of jobs

job 2

```
+++++++++++++++++++++++++
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
+++++++++++++++++++++++++
```

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.

Demo Instance

number m of machines

number n of jobs

```
+++++++++++++++++++++++++
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
+++++++++++++++++++++++++
```

job 3

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.

Demo Instance

number m of machines

number n of jobs

```
+++++++++++++++++++++++++
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
+++++++++++++++++++++++++
```

job 0
job 1
job 2
job 3

Each of the n jobs has m operations, each consisting of a machine index and a time requirement.

Instance `abz7`

Instance `abz7` by Adams et al.[15]

20 jobs

15 machines

```
+++++++++++++++++++++++++++++
Adams, Balas, and Zawack 15 x 20 instance (Table 1, instance 7)
20 15
  2 24  3 12  9 17  4 27  0 21  6 25  8 27  7 26  1 30  5 31 11 18 14 16 13 39 10 19 12 26
  6 30  3 15 12 20 11 19  1 24 13 15 10 28  2 36  5 26  7 15  0 11  8 23 14 20  9 26  4 28
  6 35  0 22 13 23  7 32  2 20  3 12 12 19 10 23  9 17  1 14  5 16 11 29  8 16  4 22 14 22
  9 20  6 29  1 19  7 14 12 33  4 30  0 32  5 21 11 29 10 24 14 25  2 29  3 13  8 20 13 18
 11 23 13 20  1 28  6 32  7 16  5 18  8 24  9 23  3 24 10 34  2 24  0 24 14 28 12 15  4 18
  8 24 11 19 14 21  1 33  7 34  6 35  5 40 10 36  3 23  2 26  4 15  9 28 13 38 12 13  0 25
 13 27  3 30  6 21  8 19 12 12  4 27  2 39  9 13 14 12  5 36 10 21 11 17  1 29  0 17  7 33
  5 27  4 19  6 29  9 20  3 21 10 40  8 14 14 39 13 39  2 27  1 36 12 12 11 37  7 22  0 13
 13 32 11 29  8 24  3 27  5 40  4 21  9 26  0 27 14 27  6 16  2 21 10 13  7 28 12 28  1 32
 12 35  1 11  5 39 14 18  7 23  0 34  3 24 13 11  8 30 11 31  4 15 10 15  2 28  9 26  6 33
 10 28  5 37 12 29  1 31  7 25  8 13 14 14  4 20  3 27  9 25 13 31 11 14  6 25  2 39  0 36
  0 22 11 25  5 28 13 35  4 31  8 21  9 20 14 19  2 29  7 32 10 18  1 18  3 11 12 17  6 15
 12 39  5 32  2 36  8 14  3 28 13 37  0 38  6 20  7 19 11 12 14 22  1 36  4 15  9 32 10 16
  8 28  1 29 14 40 12 23  4 34  5 33  6 27 10 17  0 20  7 28 11 21  2 21 13 20  9 33  3 27
  9 21 14 34  3 30 12 38  0 11 11 16  2 14  5 14  1 34  8 33  4 23 13 40 10 12  6 23  7 27
  9 13 14 40  7 36  4 17  0 13  5 33  8 25 13 24 10 23  3 36  2 29  1 18 11 13  6 33 12 13
  3 25  5 15  2 28 12 40  7 39  1 31  8 35  6 31 11 36  4 12 10 33 14 19  9 16 13 27  0 21
 12 22 10 14  0 12  2 20  5 12  1 18 11 17  8 39 14 31  3 31  7 32  9 20 13 29  4 13  6 26
  5 18 10 30  7 38 14 22 13 15 11 20  9 16  3 17  1 12  2 13 12 40  6 17  8 30  4 38  0 13
  9 31  8 39 12 27  1 14  5 33  3 31 11 22 13 36  0 16  7 11 14 14  4 29  6 28  2 22 10 17

+++++++++++++++++++ EOF ++++++++++++++++++++++++++++++++++++
```

Instance `la24`

Instance `la24` by Lawrence[16].

15 jobs

10 machines

```
+++++++++++++++++++++++++++++
Lawrence 15x10 instance (Table 7, instance 4)
15 10
7  8 9 75 0 72 6 74 4 30 8 43 2 38 5 98 1 26 3 19
6 19 8 73 3 43 0 23 1 85 4 39 5 13 9 26 2 67 7  9
1 50 3 93 5 80 4  7 0 55 2 61 6 57 8 72 9 42 7 46
1 68 7 43 4 99 6 60 5 68 0 91 8 11 3 96 9 11 2 72
7 84 2 34 8 40 5  7 1 70 6 74 3 12 0 43 9 69 4 30
8 60 0 49 4 59 5 72 9 63 1 69 7 99 6 45 3 27 2  9
6 71 2 91 8 65 1 90 9 98 4  8 7 50 0 75 5 37 3 17
8 62 7 90 5 98 3 31 2 91 4 38 9 72 1  9 0 72 6 49
4 35 0 39 9 74 5 25 7 47 3 52 2 63 8 21 6 35 1 80
9 58 0  5 3 50 8 52 1 88 6 20 2 68 5 24 4 53 7 57
7 99 3 91 4 33 5 19 2 18 6 38 0 24 9 35 1 49 8  9
0 68 3 60 2 77 7 10 8 60 5 15 9 72 1 18 6 90 4 18
9 79 1 60 3 56 6 91 2 40 8 86 7 72 0 80 5 89 4 51
4 10 2 92 5 23 6 46 8 40 7 72 3  6 1 23 0 95 9 34
2 24 5 29 9 49 8 55 0 47 6 77 3 77 7  8 1 28 4 48
+++++++++++++++++++++++++++++
```

Instance swv15

Instance swv15 by Storer et al.[17]

50 jobs

10 machines

```
+++++++++++++++++++++++++++++
Storer, Wu, and Vaccari hard 50x10 instance (Table 2, instance 15)
50 10
  2  93  4  40  0   1  3  77  1  77  5  16  9  74  8  11  6  51  7  92
  0  92  4  80  1  76  3  59  2  70  5  86  9  17  6  78  7  30  8  93
  1  44  2  92  3  96  4  77  0  53  9  10  7  49  5  84  8  59  6  14
  1  60  2  19  3  76  0  73  4  85  7  13  8  93  5  68  9  50  6  78
  2  20  0  24  3  41  1   2  4   4  9  44  7  79  8  81  5  16  6  39
  3  41  2  35  1  32  4  18  0  15  8  98  6  29  5  19  7  14  9  26
  1  59  0  45  4  53  3  44  2  98  5  84  6  23  7  45  8  39  9  89
  1  30  4  51  3  25  0  51  2  84  6  60  5  45  7  89  8  25  9  97
  0  47  3  18  2  40  4  62  1  58  5  36  7  93  8  77  9  90  6  15
  3  33  1  68  0  41  4  72  2  20  6  69  7  47  5  22  9  47  8  22
  2  28  1 100  4  20  0  35  3  26  5  24  9  41  6  42  7 100  8  32
  0  65  2  12  4  53  3  93  1  40  8  18  7  23  5  60  6  89  9  53
  0  58  1  60  4  97  3  31  2  50  9  85  5  64  7  38  6  85  8  35
  3  64  0  58  1  49  2  45  4   9  8  49  6  22  5  99  9  15  7   7
  0  10  4  85  3  72  2  37  1  77  5  70  7  45  9   8  6  83  8  57
  4  93  0  87  1  87  2  18  3   4  8  78  5  67  9  20  6  17  7  35
  4  72  0  56  3  57  2  15  1  45  6  41  5  40  9  85  8  32  7  81
  0  36  3  63  4  79  2  32  1   5  6  25  7  86  9  91  5  21  8  35
  2  83  4  29  0   9  1  38  3  73  7  50  9  99  5  18  8  29  6  41
  0 100  3  29  2  60  4  63  1  64  8  71  6  35  5  26  9   9  7  22
  1  81  0  60  3  62  4  48  2  68  7  28  5  69  8  92  6  79  9  10
  0  40  4  80  1  41  2  10  3  68  8  28  9  51  7  33  6  82  5  25
  4  30  2  12  0  35  3  17  1  70  9  29  7  18  8  93  6  94  5  37
  1  36  2  41  3  27  4  36  0  78  7  64  6  88  5  25  9  92  8  66
  2  65  3  27  4  74  0  32  1  40  5  88  8  73  6  92  7  83  9  42
  0  48  1  85  2  92  4  95  3  61  8  72  9  76  5  58  7  11  6  89
  3  84  2  50  0  70  4  24  1  42  9  55  5 100  6  70  7   4  8  68
  0  95  4  41  2  11  3  98  1  85  5  64  6   8  7  26  8   6  9   6
  0  84  2  49  1  17  3  69  4  55  8  75  6  45  9  38  7  59  5  28
  2  48  0  29  4   1  1  64  3  41  5  23  7  64  9  31  6  56  8  12
  2  81  4  25  3  33  0  22  1  50  5  74  9  56  8  33  7  85  6  83
  1  62  4  25  0  21  2  20  3   8  6  36  9   9  5  91  8  90  7  49
  1  43  0  16  2  91  3  96  4  24  5  11  9  91  7  41  8  35  6  66
  1  91  2  20  4  44  0  42  3  87  9  57  6  15  5  38  8  42  7  89
  0  33  3  95  4  68  2  22  1  80  7  53  8  13  9  70  5  22  6  69
  0  15  3  47  1  24  2  31  4  41  8  14  9  28  7  59  5  52  6  39
  2  95  0  42  4   5  1  57  3  67  6  30  9  21  8  70  5   9  7  20
  2  54  0  15  1  20  3  64  4  83  9  40  7   6  5  89  6  91  8  48
  0  22  4  27  1  77  3  25  2  16  8  72  9  61  6  75  7  4   5  19
  3  68  1  82  2  16  0  83  4   2  7  10  8  88  5  41  9  21  6  66
  1  64  0  76  2  85  3  71  4  97  5  97  7   8  6  40  8  70  9  35
  0  94  1  45  2  94  4  84  3  44  8  41  5  30  7  47  6  19  9  22
  2  23  1  10  0  82  3  93  4  90  8  67  7   9  9  18  5  22  6  87
  0  75  2  27  4  97  3   9  1  57  9  14  5  50  7  31  8  62  6  23
  1  42  3  41  2  35  0  75  4  18  9  65  7  38  6  38  8  51  5  56
  4  72  1  63  0  33  2  27  3  41  5  52  7  42  9  10  6  14  8  71
  2  91  1  89  0  44  4  91  3  26  6  49  5  22  8  31  9  69  7   5
  3  42  1  34  0   4  4  34  2  16  6  86  7  25  8  99  5  67  9  25
  4  34  1  93  0  26  3  81  2   9  7  96  8  79  9  68  5  76  6  10
  3  19  1  47  4  13  2  98  0  32  7  12  9  45  6  52  8  49  5  34
+++++++++++++++++++++++++++++
```

Instance `yn4`

Instance `yn4` by Yamada and Nakano[18].

```
    +++++++++++++++++++++++++++++
20 jobs Yamada and Nakano 20x20 instance (Table 4, instance 4)
    20 20 20 machines
    16 34 17 38  0 21  6 15 15 42  8 17  7 41 18 10 10 26 11 24  1 31 19 25 14 31 13 33  4 35  9 30  3 16 12 16  5 30  2 13
     5 41 11 33  6 15 16 38  0 40 14 38  3 37  1 20 13 22  4 34  7 16 17 39  9 15  2 19 10 36 12 39 18 26  8 19 15 39 19 34
    17 34  1 12 16 10  7 47 13 28 15 27  0 19  6 34 19 33 12 40  9 37 14 24  8 15 10 34  2 44  3 37 18 22 11 31  4 39  5 26
     5 48  7 46 16 47 10 45 14 15  8 25  0 34  3 24 12 35 18 15  2 48 13 19 11 10  1 48 17 16 15 28  4 18  6 17  9 44 19 41
    12 47  3 23  9 48 16 45 14 39  6 42  8 32 15 11 13 16  5 14 11 19  1 46 19 10 10 17  7 41  2 47 17 32  4 17  0 21 18 17
    18 14 16 20  1 18 12 14 13 10  6 16  5 24  4 18  0 24 11 18 15 42 19 13  3 23 14 40  9 48  8 12  2 24 10 23  7 45 17 30
     0 27 12 15  4 26 13 19 17 14  5 49  7 16 18 28 16 16  8 20  9 36  2 21 14 30  3 36  1 17 15 22  6 43 11 32 10 23 19 17
     0 32 16 15 17 12  7 46  3 37 18 43 11 40 13 43  9 48  4 36 15 24  8 25  1 33 14 32  5 26  6 37 12 24 10 24  2 15 19 22
    10 34  6 33 15 25  8 46  0 20 18 33  4 19 13 45  2 47  1 32  3 12 11 29 16 29  5 46 12 17  7 48 14 39 17 40 19 41  9 37
    13 26  3 47  5 44  6 49  1 22 17 12 10 28 19 36  9 27  4 25 14 48  7 11 16 49 12 24 11 48  2 19  0 47 18 49  8 46 15 36
    13 23 18 48 14 15  0 42  3 36  8 15  6 32 10 18  1 45 15 23 11 45  2 13 17 21 12 32  7 44  5 25 19 34 16 22  9 11  4 43
    17 37  7 49 15 45  2 28  9 15  8 35 12 29 13 44  1 26  4 25  5 30  3 39  0 15 14 28 18 23  6 42 11 33 16 45 10 10 19 20
     0 10  6 37  3 15 13 13 10 11  2 49  1 28 14 28 15 13  8 29 12 21 16 32 11 21  4 48  5 11 17 26  9 33 18 22  7 21 19 49
    18 38  0 41  4 30 13 43  6 11  2 43 14 27  3 26  9 30 15 19 16 36  1 31 17 47  5 41 10 34  8 40 12 32  7 13 11 18 19 27
     6 24  5 30  7 10 10 35  8 28 16 43 19 12  9 44 15 15  3 15  2 35 18 43  0 38  4 16  1 29 17 40 14 49 13 38 12 16 11 30
     3 48  6 35 13 43  2 37 17 18  5 27  9 27  7 41  1 22 15 28 16 18 10 37 18 48  4 10  8 14 11 18 14 43  0 48 12 12 19 49
     0 13 13 38  7 34  6 42  1 36  5 45 18 24  8 35 14 26 19 30 12 47 16 24 11 47  4 40 10 43  3 16 15 10  2 12  9 39 17 22
    16 30 13 47 19 49  8 20  4 40  3 46 17 21 14 33  6 44  7 23  9 24  0 48 10 43 15 41  2 32  5 29 11 36  1 38 12 47 18 12
    13 10  5 36 12 18 16 48  0 27 14 43 10 46  6 27  7 46 19 35 11 31  2 18  8 24  3 23 17 29 18 14  9 19  1 40 15 38  4 13
     9 45 16 44  0 43 17 31 14 35 13 17 12 42  3 14 18 37 10 39  6 48  7 38 15 26  4 49  2 28 11 35  1 42  5 24  8 44 19 38

   +++++++++++++++++++ EOF ++++++++++++++++++++++++++++++++++++
```

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 $\mathbb{Y}$

- We now know how a problem instance of the JSSP looks like, i.e., the input we get.

Output: Candidate Solutions and Solution Space $\mathbb{Y}$

- 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 $\mathbb{Y}$

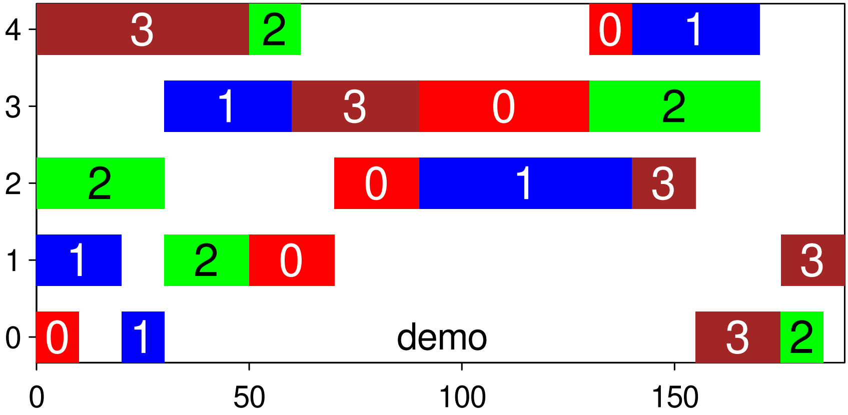
- We now know how a problem instance of the JSSP looks like, i.e., the input we get.
- But what output should we produce?
- In other words, what is a solution for an instance of the JSSP?

Output: Candidate Solutions and Solution Space $\mathbb{Y}$

- 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].

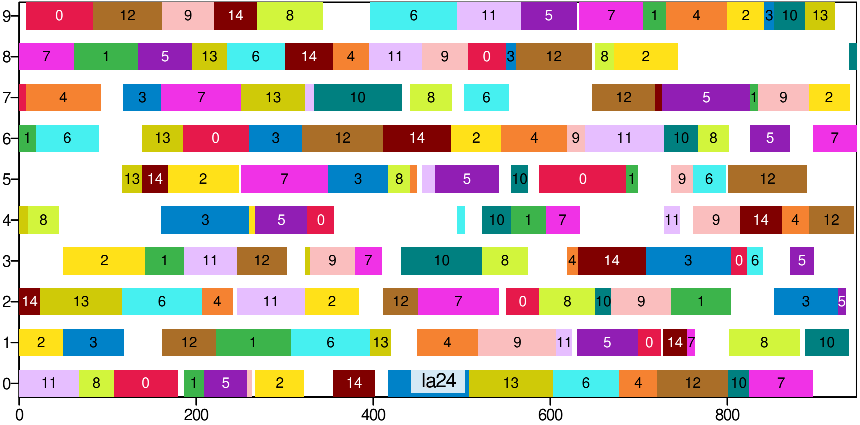
Output: Candidate Solutions and Solution Space $\mathbb{Y}$

one possible solution for the `demo` instance, illustrated as Gantt chart



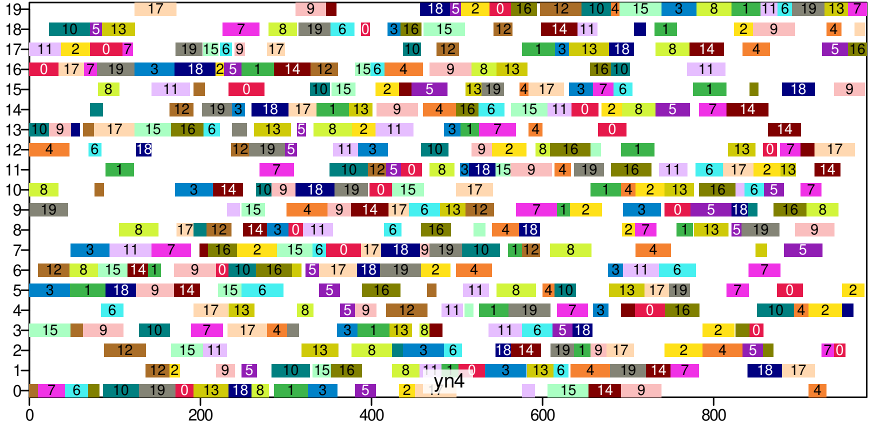
Output: Candidate Solutions and Solution Space $\mathbb{Y}$

one possible solution for the `la24` instance, illustrated as Gantt chart



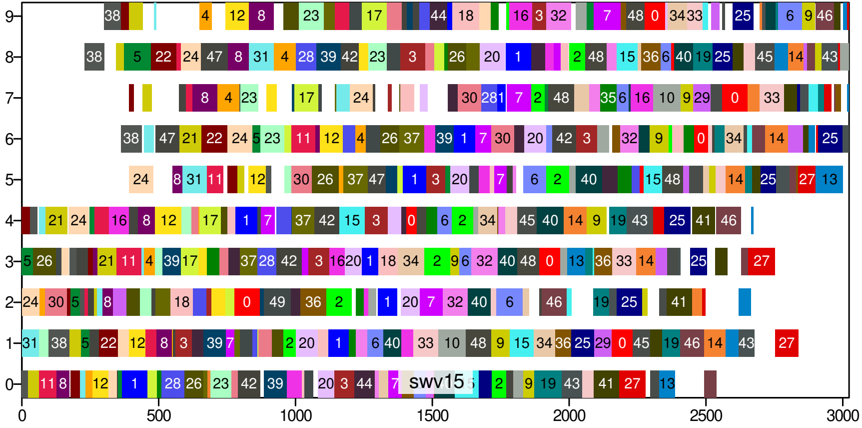
Output: Candidate Solutions and Solution Space $\mathbb{Y}$

one possible solution for the `yn4` instance, illustrated as Gantt chart



Output: Candidate Solutions and Solution Space $\mathbb{Y}$

one possible solution for the `swv15` instance, illustrated as Gantt chart



Output: Candidate Solutions and Solution Space $\mathbb{Y}$

- 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.

Output: Candidate Solutions and Solution Space $\mathbb{Y}$

- 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.

Output: Candidate Solutions and Solution Space $\mathbb{Y}$

- 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. */
}
```

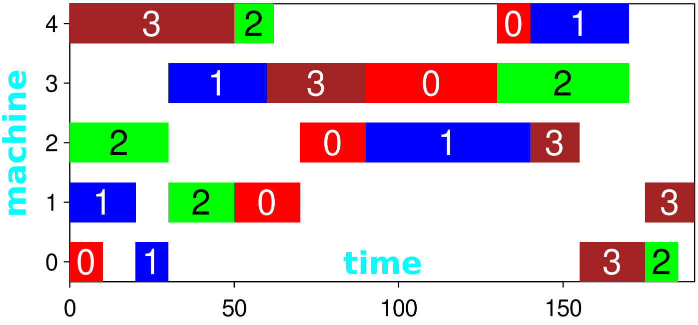
As Java Class

- 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 $3n$.

```
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. */
}
```

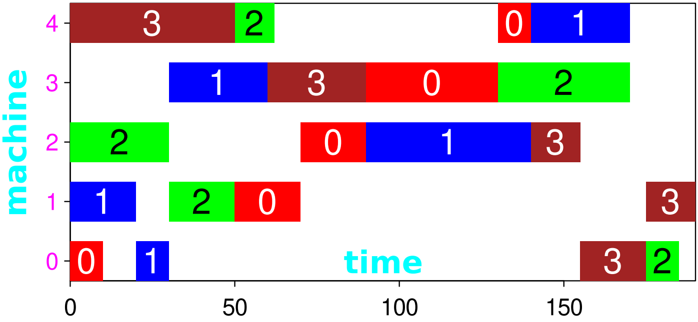
As Java Class

```
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}
}
```



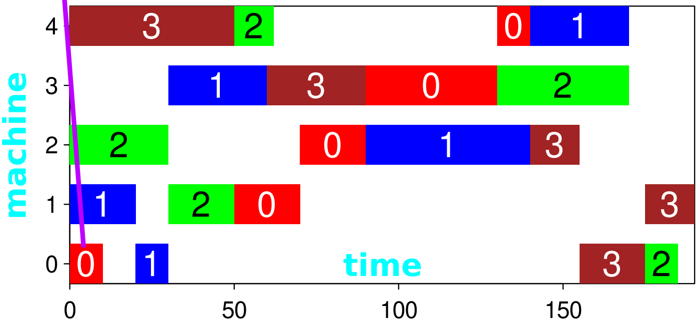
As Java Class

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



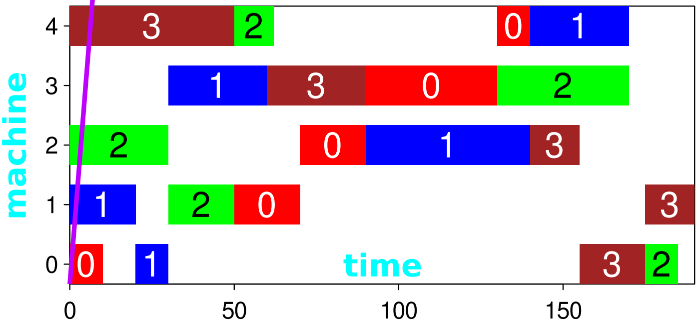
As Java Class

```
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}
}
```



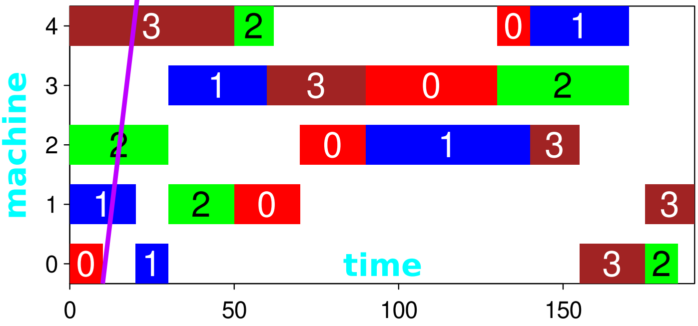
As Java Class

```
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}
}
```



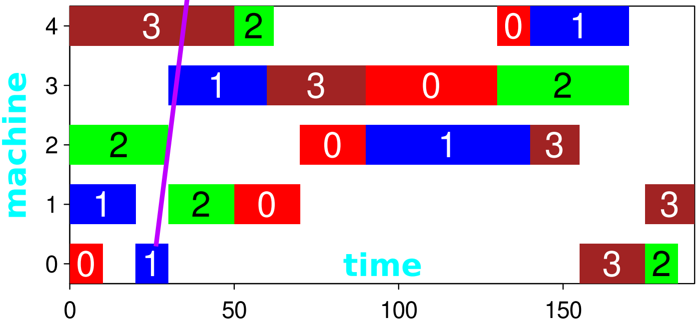
As Java Class

```
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}
}
```



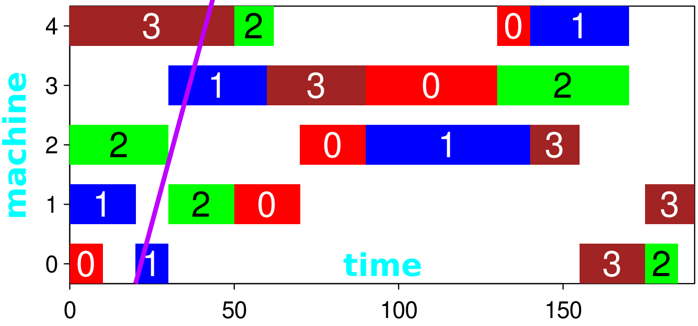
As Java Class

```
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}
}
```



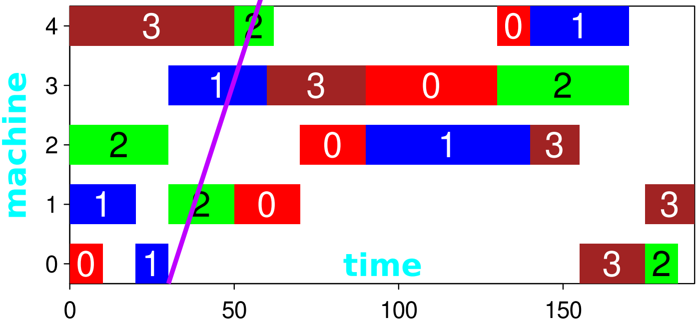
As Java Class

```
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}
}
```



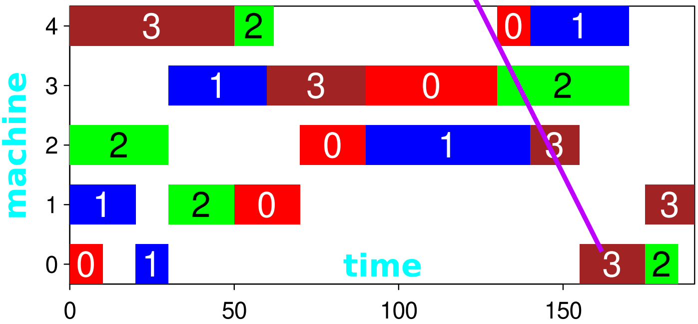
As Java Class

```
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}
}
```



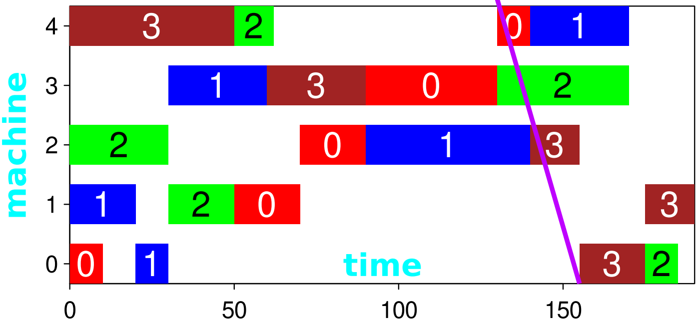
As Java Class

```
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}
}
```



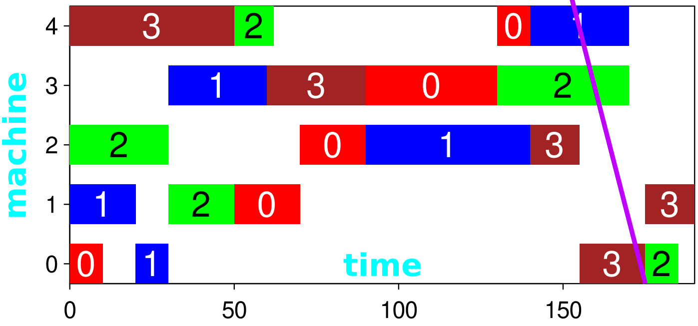
As Java Class

```
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}
}
```



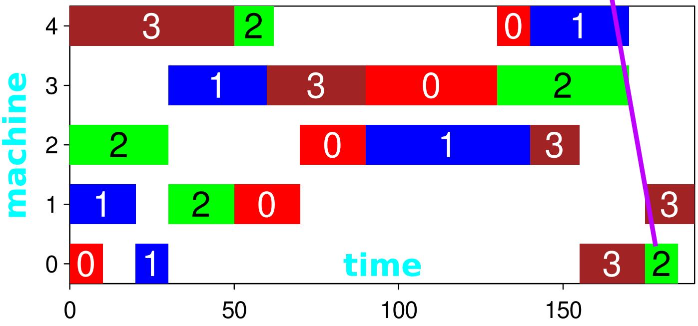
As Java Class

```
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}
}
```



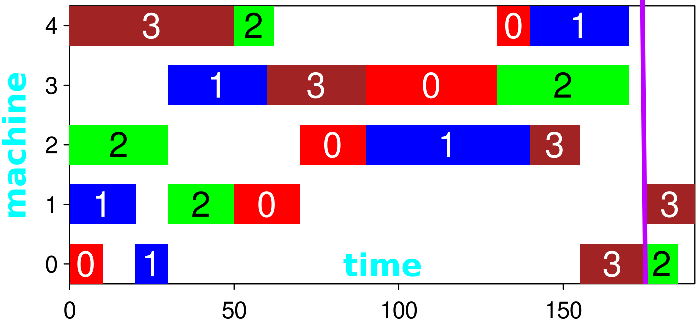
As Java Class

```
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}
}
```



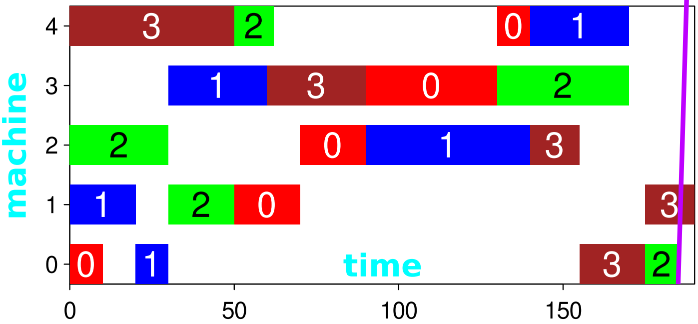
As Java Class

```
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}
}
```



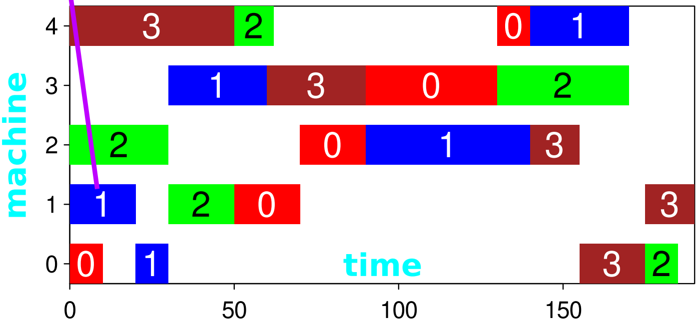
As Java Class

```
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}
}
```



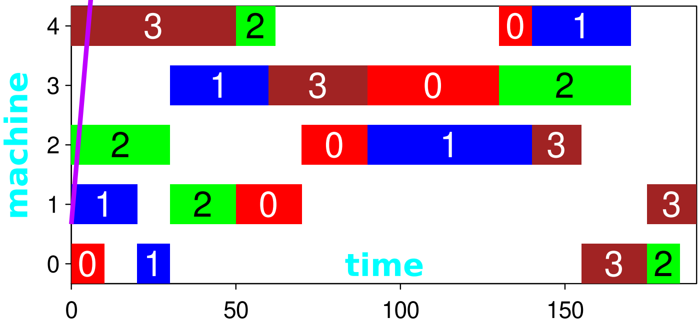
As Java Class

```
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}
}
```



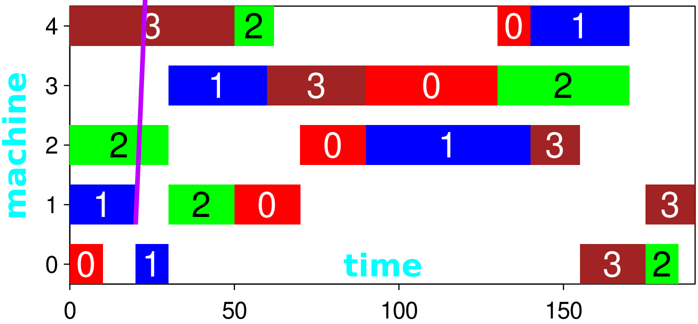
As Java Class

```
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}
}
```



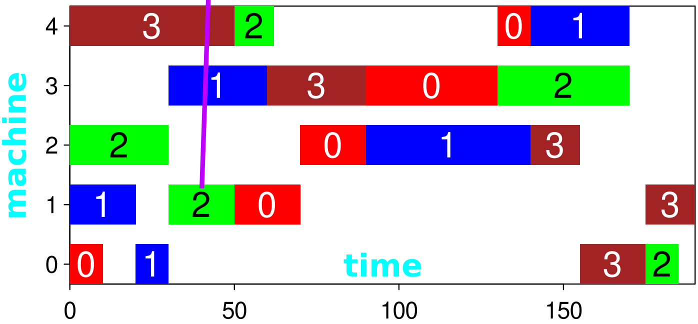
As Java Class

```
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}
}
```



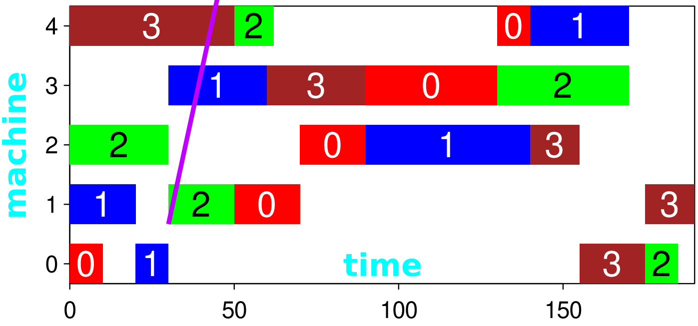
As Java Class

```
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}
}
```



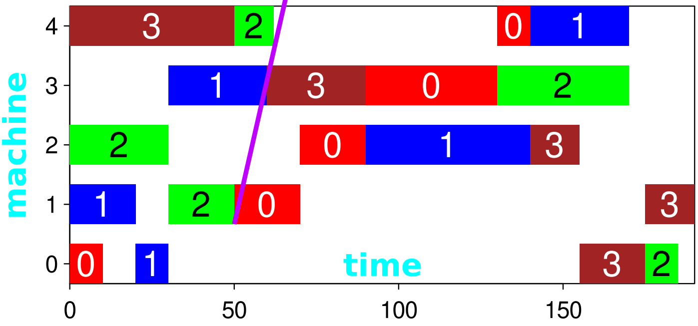
As Java Class

```
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}
}
```



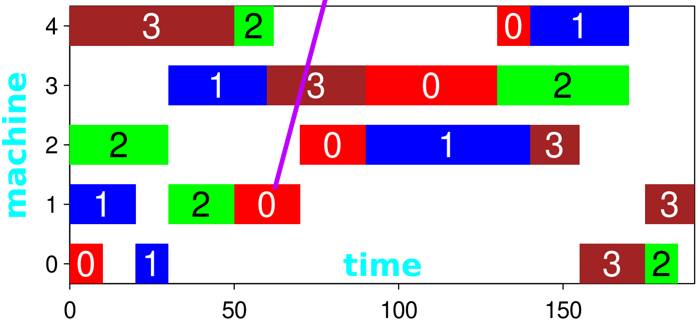
As Java Class

```
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}
}
```



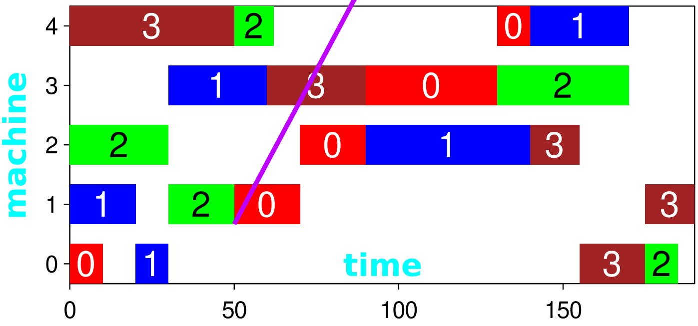
As Java Class

```
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}
}
```



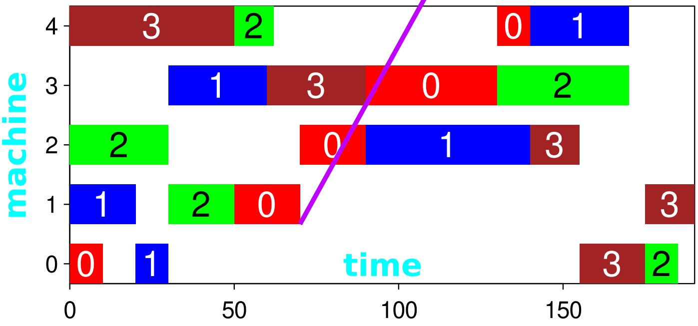
As Java Class

```
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}
}
```



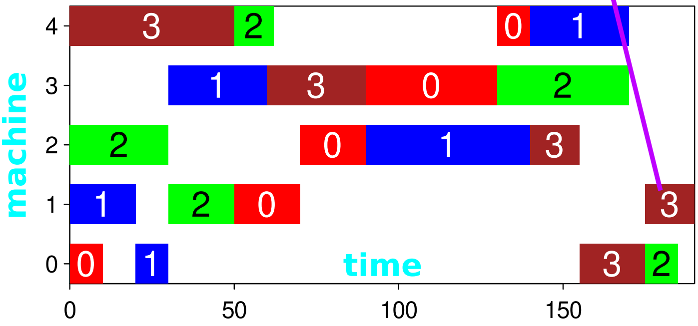
As Java Class

```
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}
}
```



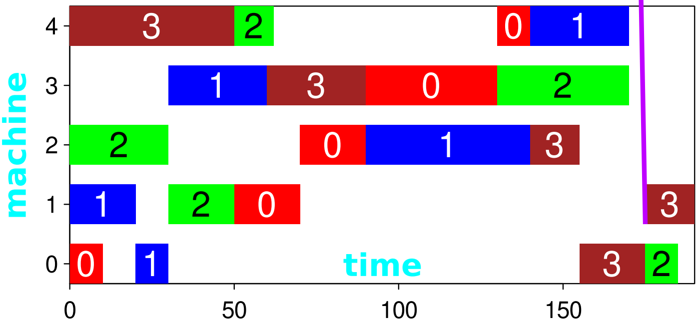
As Java Class

```
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}
}
```



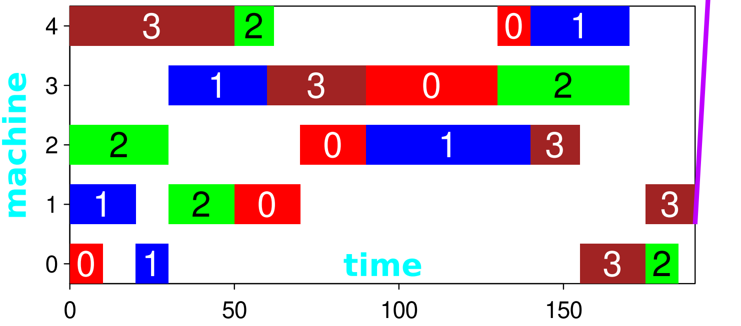
As Java Class

```
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

```
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

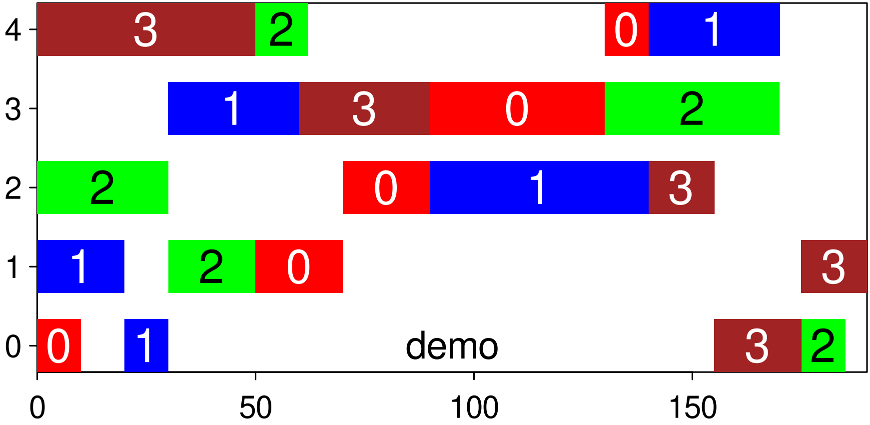
- So we have identified what the possible solutions to our problems are and know how to store them in a data structure.
- How do we rate the quality of a solution?
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if it allows us to complete 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 $y_1 \in \mathbb{Y}$ 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.

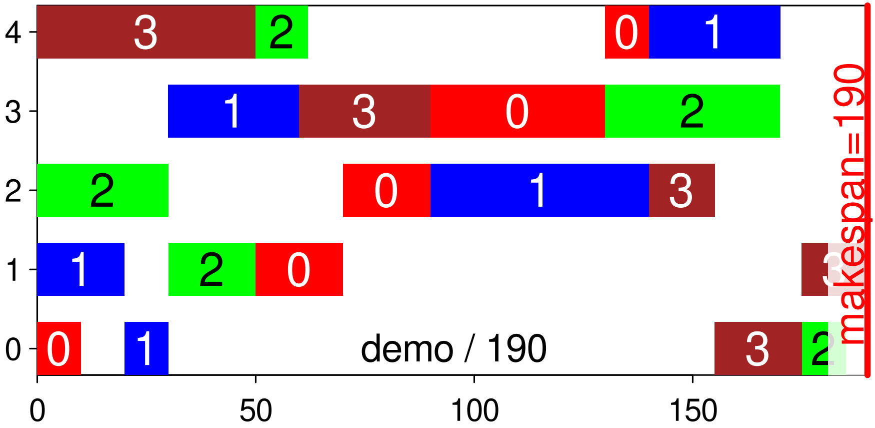
Solution Quality

- 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.



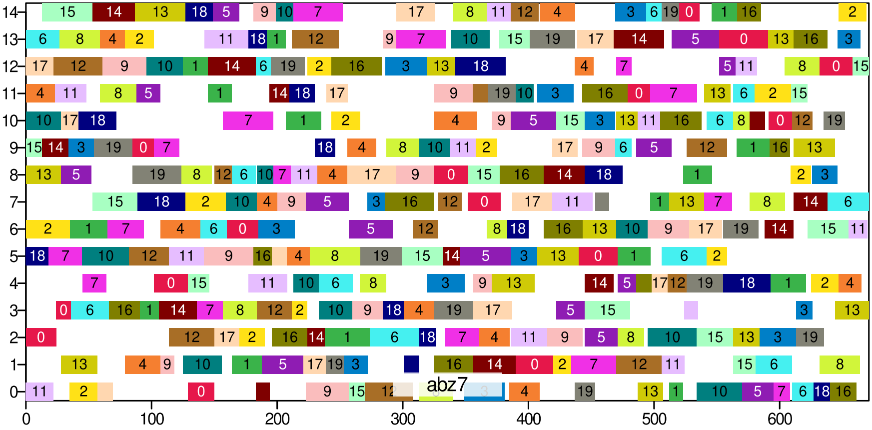
Solution Quality

- 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.



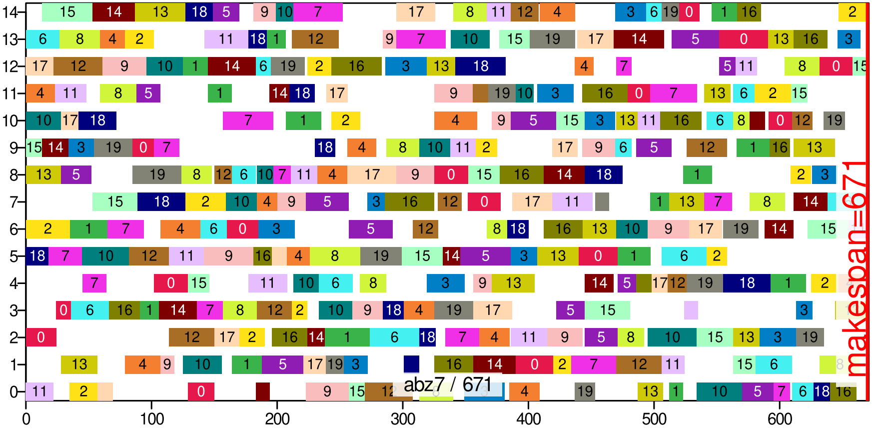
Solution Quality

- 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.



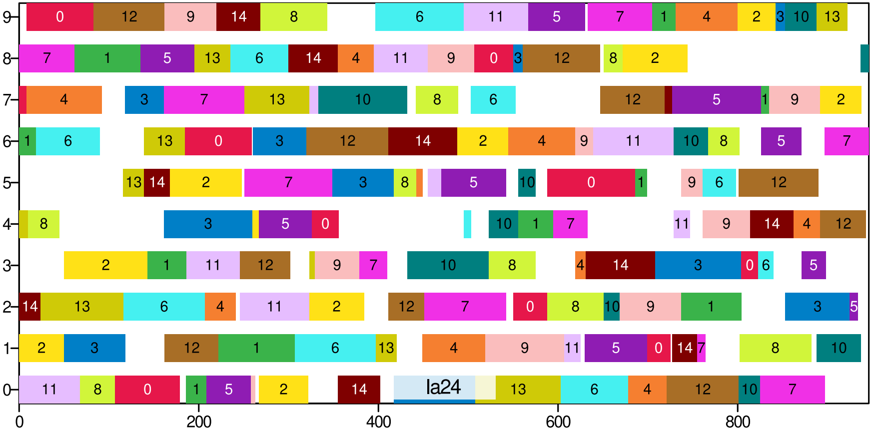
Solution Quality

- 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.



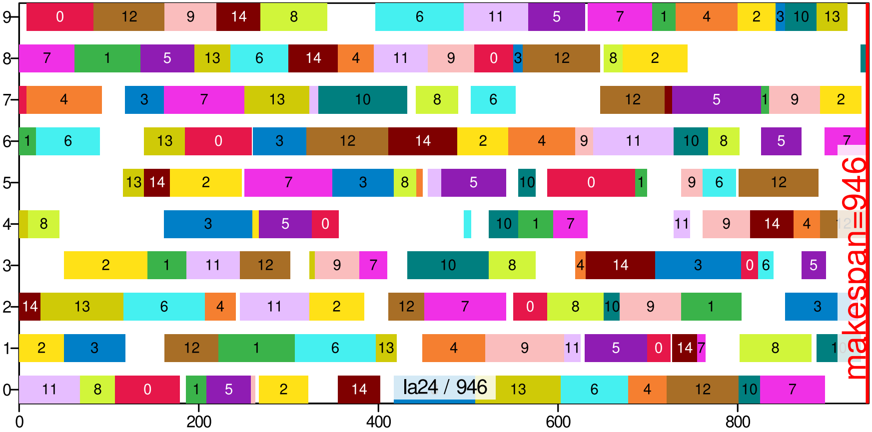
Solution Quality

- 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.



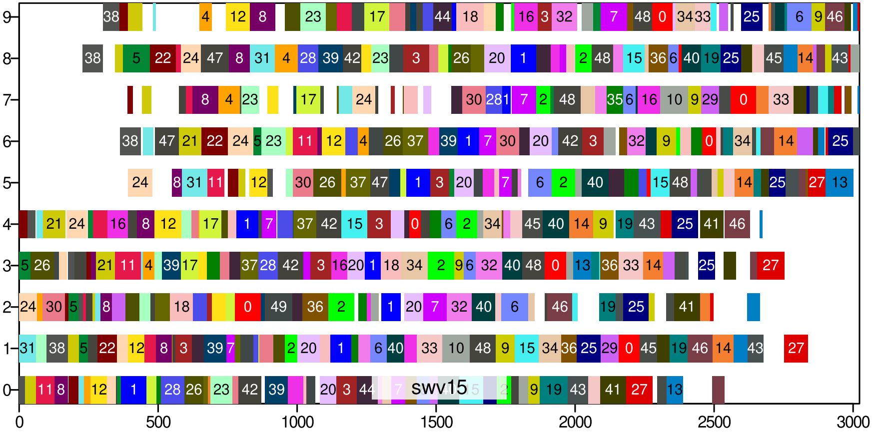
Solution Quality

- 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.



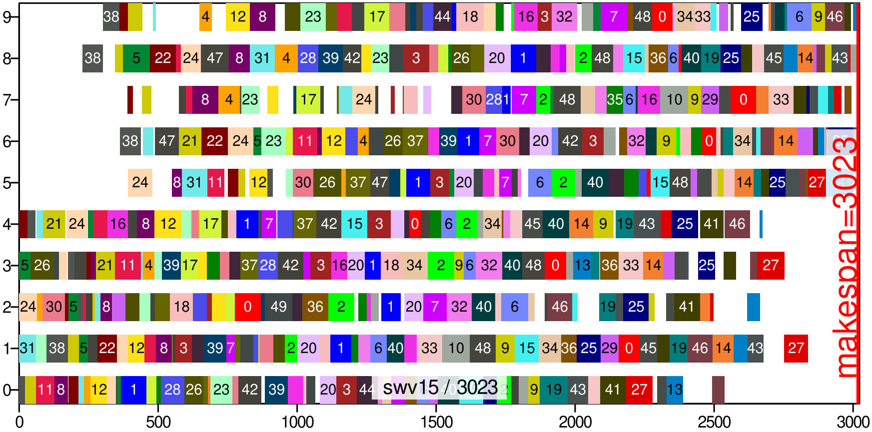
Solution Quality

- 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.



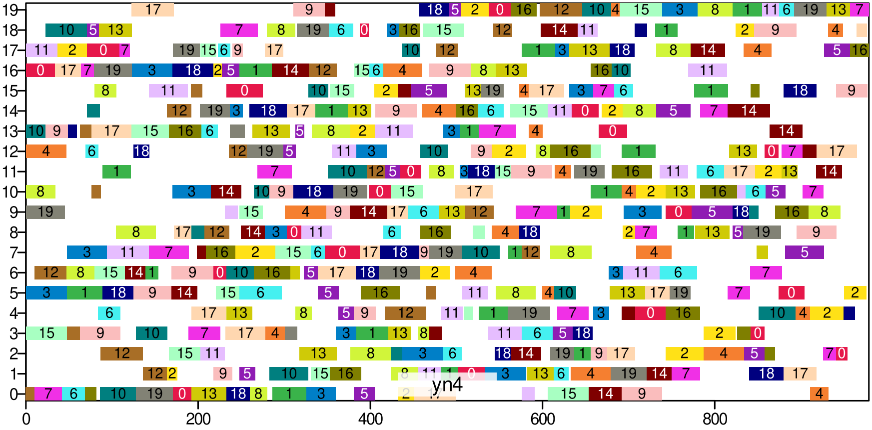
Solution Quality

- 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.



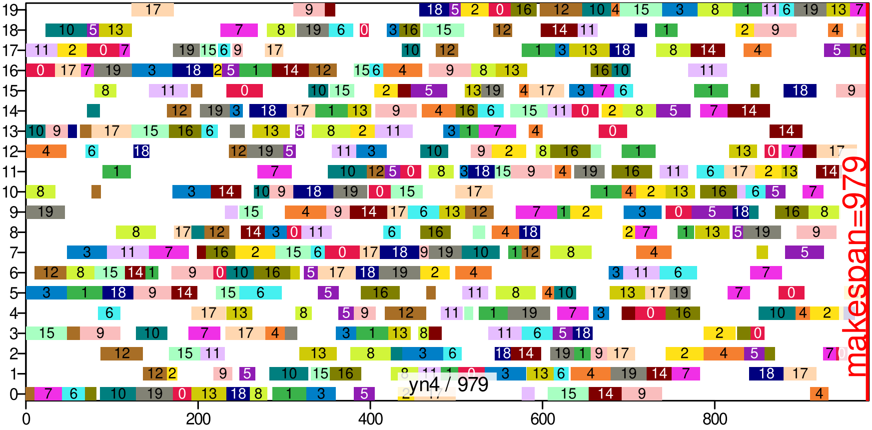
Solution Quality

- 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.



Solution Quality

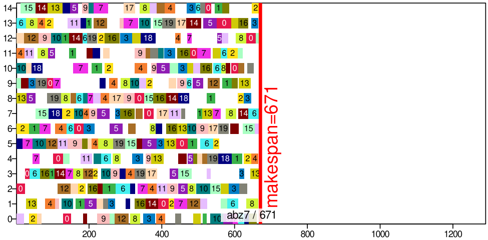
- So we have identified what the possible solutions to our problems are and know how to store them in a data structure.
- How do we rate the quality of a solution?
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if it allows us to complete 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.

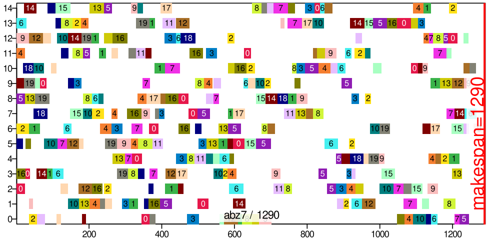
Solution Quality

- 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

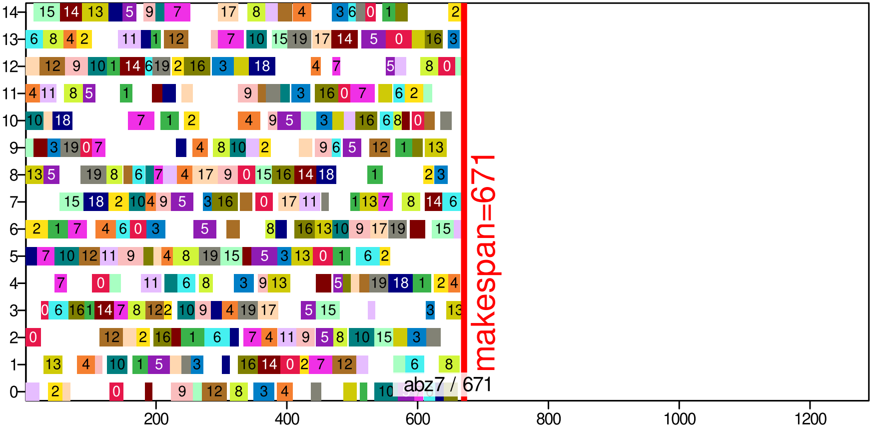
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$.





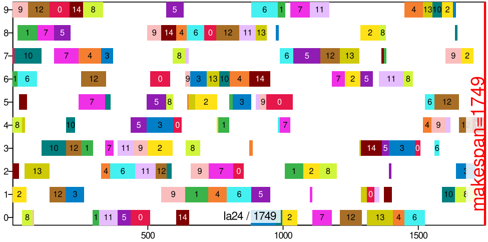
Solution Quality

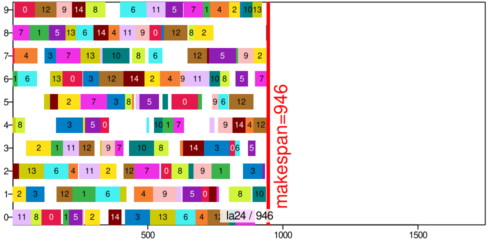
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$.



Solution Quality

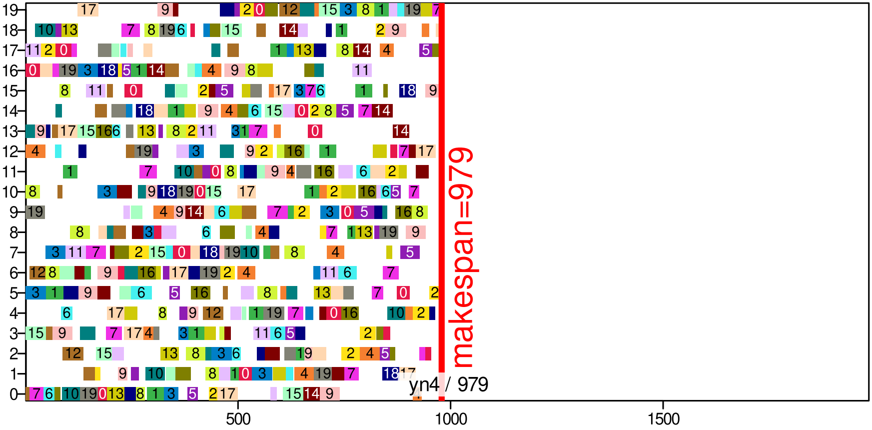
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$.





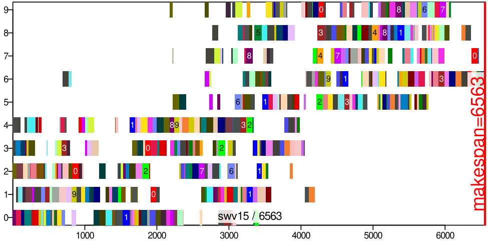
Solution Quality

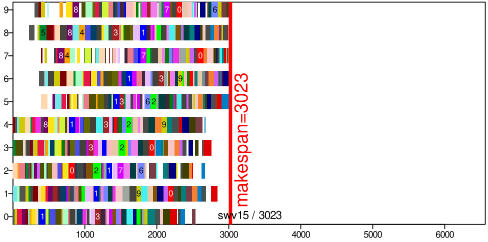
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$.



Solution Quality

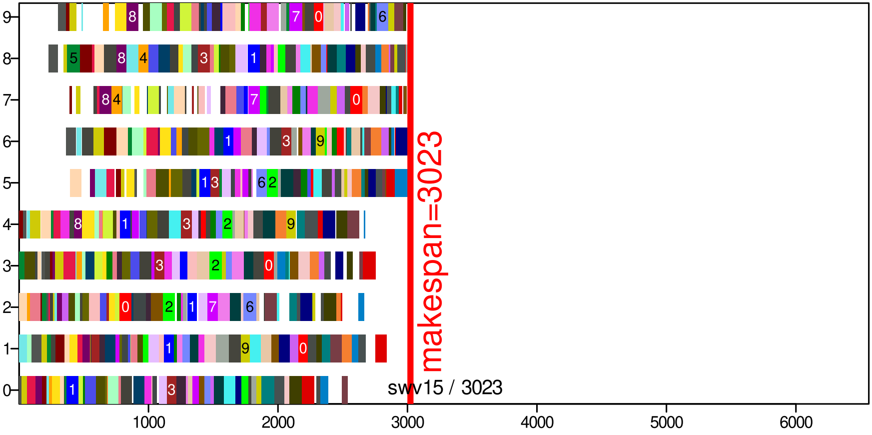
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$.





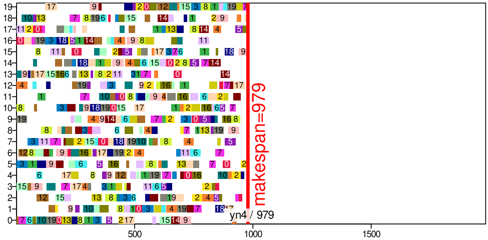
Solution Quality

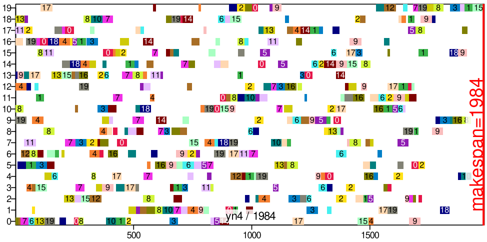
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$.



Solution Quality

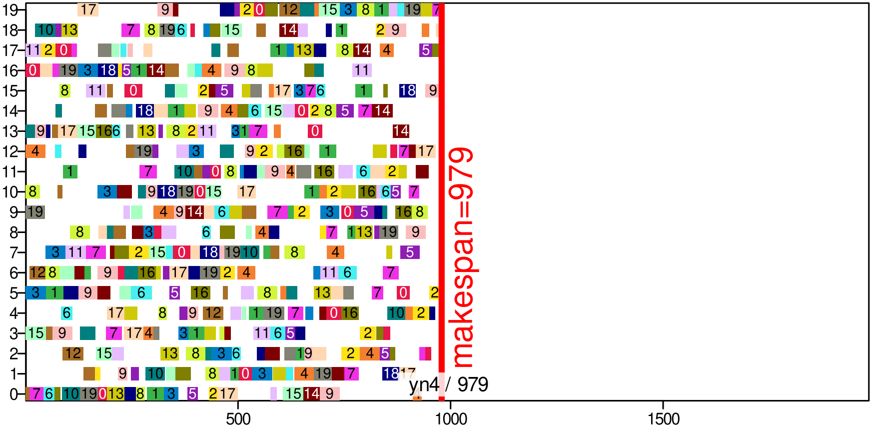
- A Gantt chart $y_1 \in \mathbb{Y}$ is a better solution to our problem than another chart $y_2 \in \mathbb{Y}$ if $f(y_1) < f(y_2)$.





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 $f(y_1) < f(y_2)$.



An Interface for Objective Functions in Java

```java
package aitoa.structure;

public interface IObjectiveFunction<Y> {

  double evaluate(Y y);

}
```

The JSSP Objective Function in Java

```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. */

//
//
//
//
//
//
//
//
//
//
}
```

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. */

//
//
//
//
//
//
//
//
//
//
}
```

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) {
//
//
//
//
//
//
//
//
  }
}
```

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
//
//
//
//
//
//
//
  }
}
```

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
//
//
//
//
//
//
    return makespan;
  }
}
```

The JSSP Objective Function in 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) {
//
//
//
//
    }
    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;
  }
}
```

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];
      if (end > makespan) { // this machine ends later
        makespan = end; // remember biggest end time
      }
    }
    return makespan;
  }
}
```

The Global Optimum $y^\star$ in $\mathbb{Y}$

- There must be at least one globally optimal solution $y^\star$.

The Global Optimum $y^\star$ in $\mathbb{Y}$

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

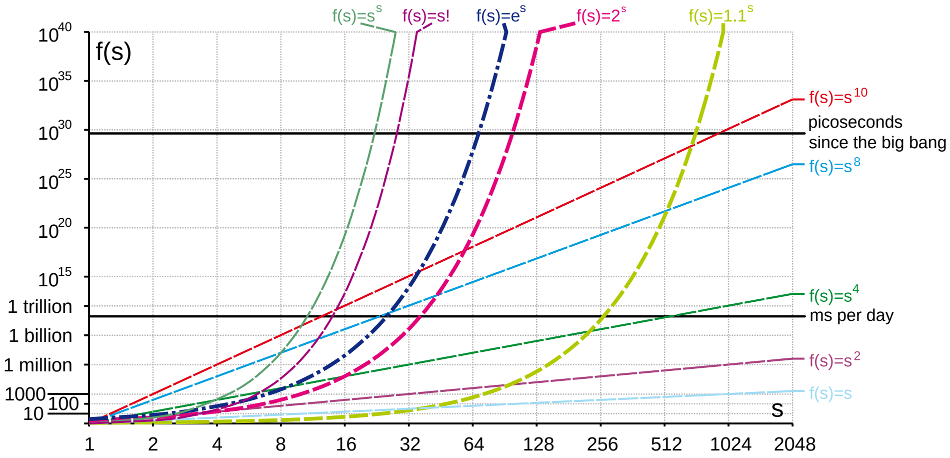
The Global Optimum $y^\star$ in $\mathbb{Y}$

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

The Global Optimum $y^\star$ in $\mathbb{Y}$

- There must be at least one globally optimal solution $y^\star$ for which $f(y^\star) \leq f(y)\;\forall y \in \mathbb{Y}$ holds.
- How do we find such a solution?
- We know the problem is $\mathcal{NP}$-hard[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 $y^\star$ in $\mathbb{Y}$



The Global Optimum $y^\star$ in $\mathbb{Y}$

- There must be at least one globally optimal solution $y^\star$ for which $f(y^\star) \leq f(y)\ \forall y \in \mathbb{Y}$ holds.
- How do we find such a solution?
- We know the problem is $\mathcal{NP}$-hard[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 $y^\star$ in $\mathbb{Y}$

- There must be at least one globally optimal solution $y^\star$ for which $f(y^\star) \leq f(y)\ \forall y \in \mathbb{Y}$ holds.
- How do we find such a solution?
- We know the problem is $\mathcal{NP}$-hard[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.
- What we can always do is search in $\mathbb{Y}$ and hope to get as close to $y^\star$ within reasonable time as possible.

The Global Optimum $y^{\star}$ in $\mathbb{Y}$

- There must be at least one globally optimal solution $y^{\star}$ for which $f(y^{\star}) \leq f(y)\ \forall y \in \mathbb{Y}$ holds.
- How do we find such a solution?
- We know the problem is $\mathcal{NP}$-hard[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.
- What we can always do is search in $\mathbb{Y}$ and hope to get as close to $y^{\star}$ 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 $\mathbb{Y}$?

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*.

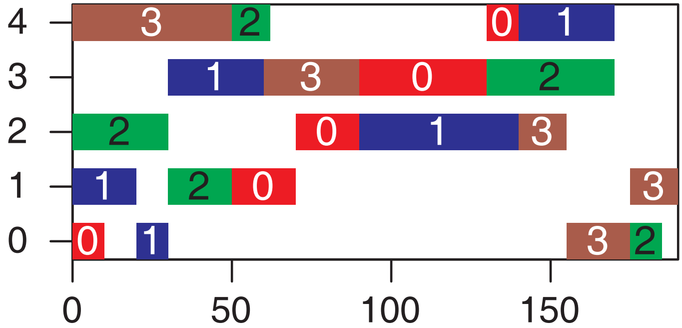
Feasibility of Solutions

- 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

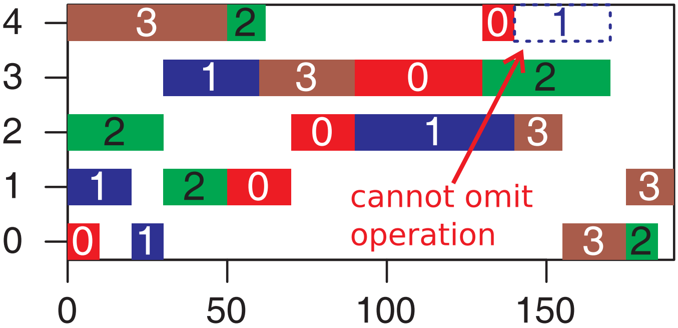
Feasibility of Solutions

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

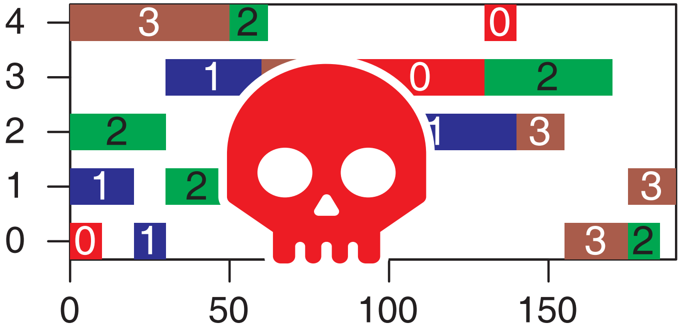
Feasibility of Solutions



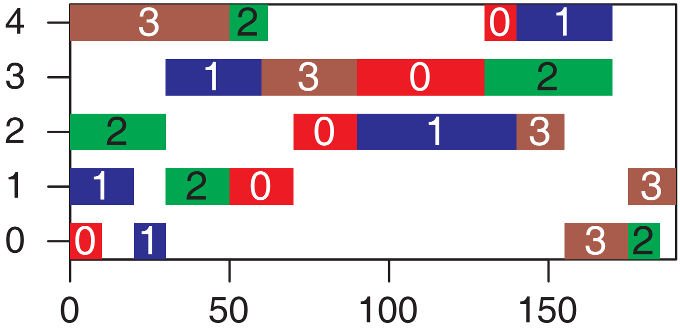
Feasibility of Solutions



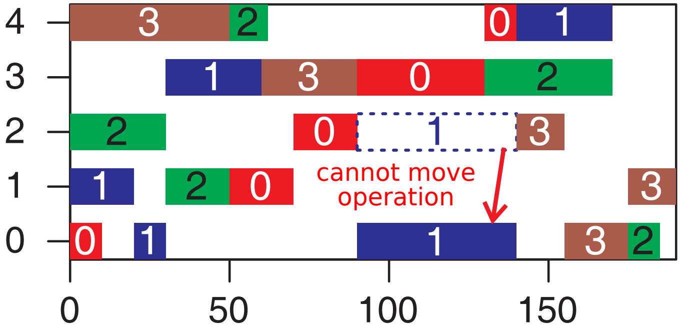
Feasibility of Solutions



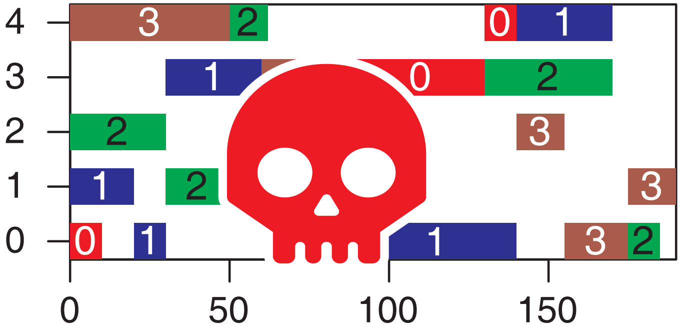
Feasibility of Solutions



Feasibility of Solutions



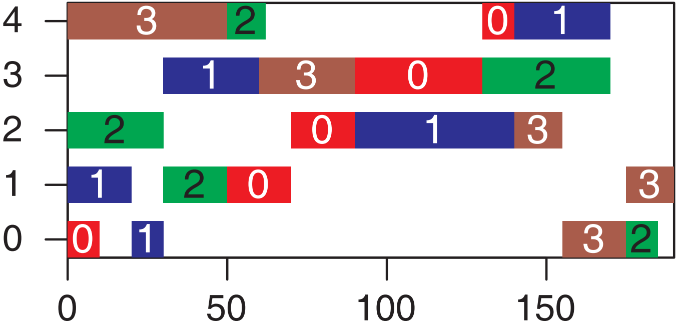
Feasibility of Solutions



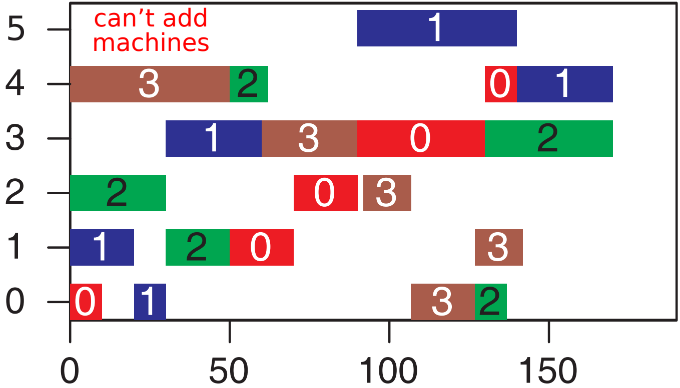
Feasibility of Solutions

- 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

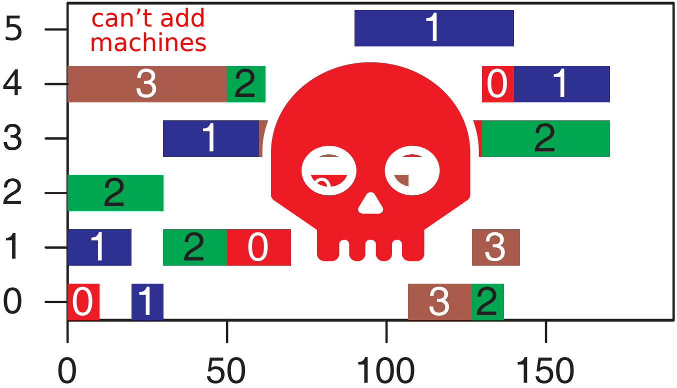
Feasibility of Solutions



Feasibility of Solutions



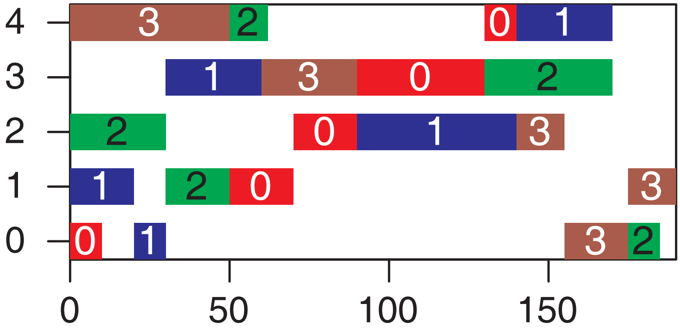
Feasibility of Solutions



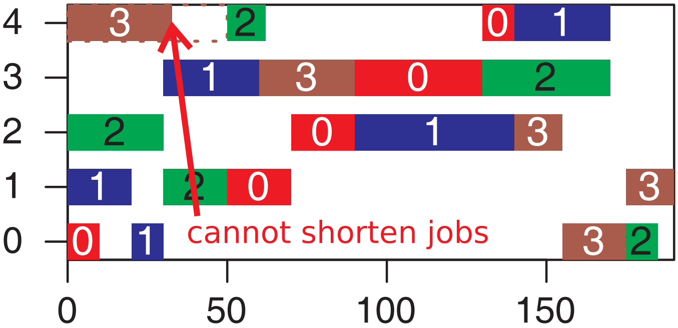
Feasibility of Solutions

- 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

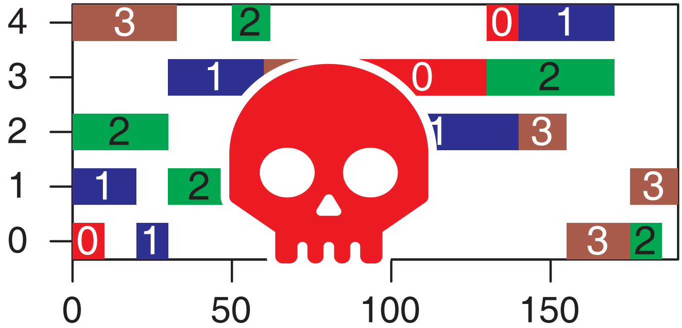
Feasibility of Solutions



Feasibility of Solutions



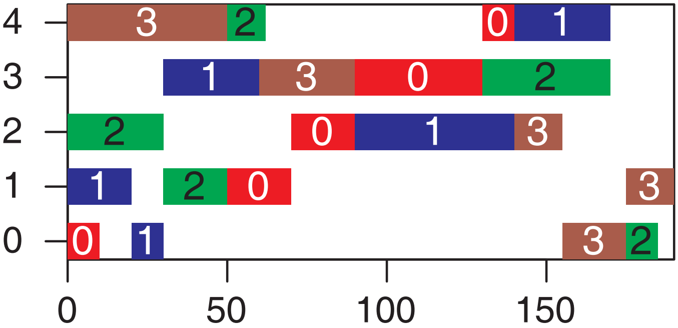
Feasibility of Solutions



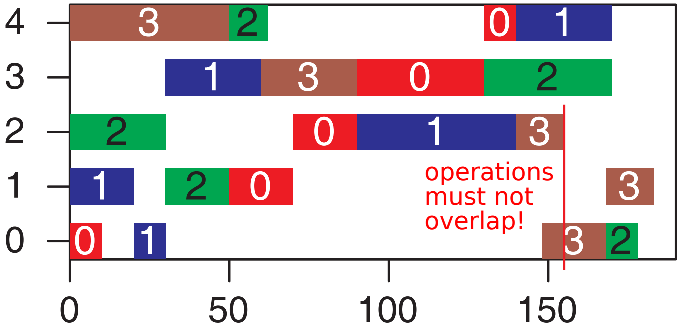
Feasibility of Solutions

- 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

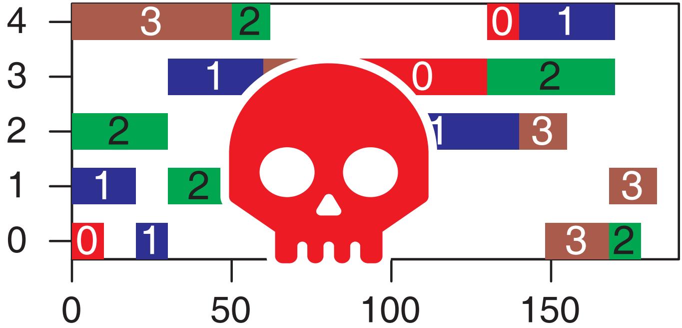
Feasibility of Solutions



Feasibility of Solutions



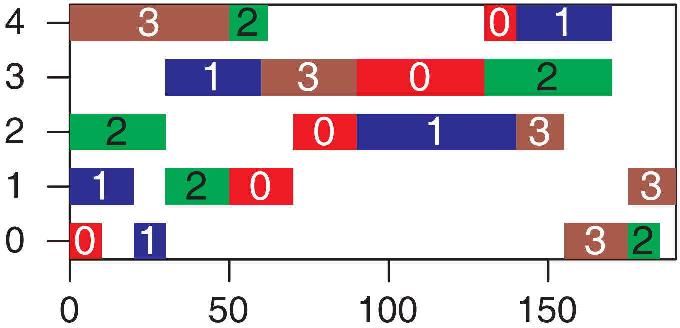
Feasibility of Solutions



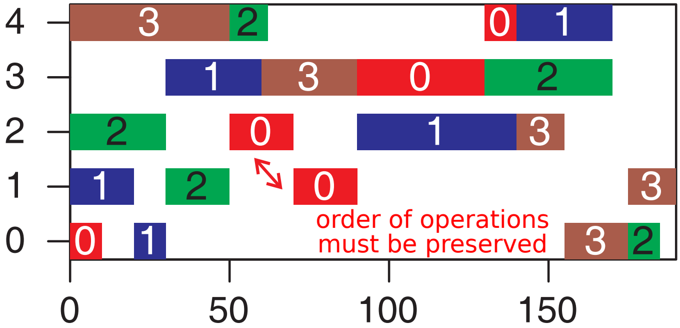
Feasibility of Solutions

- 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.

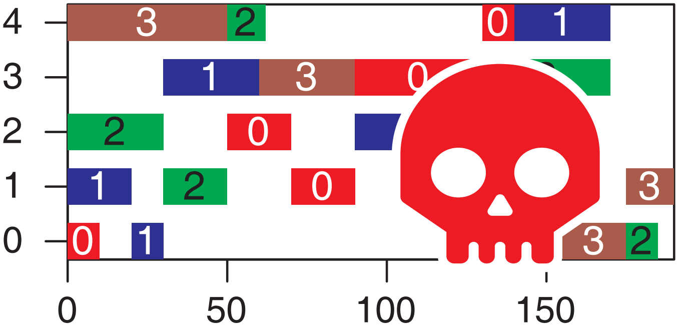
Feasibility of Solutions



Feasibility of Solutions



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.
- Only a Gantt chart obeying all of these constraints is feasible, i.e., can be implemented in practice.

Hardships when Searching in $\mathbb{Y}$

- So how do we search in the space of Gantt charts?

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.

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!

Hardships when Searching in $\mathbb{Y}$

```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
++++++++++++++++++++++++
```

Hardships when Searching in $\mathbb{Y}$

```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10    1  20
0  10    1  20
++++++++++++++++++++++++
```

Hardships when Searching in $\mathbb{Y}$

```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
++++++++++++++++++++++++
```

Hardships when Searching in $\mathbb{Y}$

```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
++++++++++++++++++++++++
```

Hardships when Searching in $\mathbb{Y}$

```
job 0
+++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
+++++++++++++++++++++++++
```

Hardships when Searching in $\mathbb{Y}$

```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
++++++++++++++++++++++++
```

job 0

job 1

Hardships when Searching in $\mathbb{Y}$

```
job 0
job 1
+++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
+++++++++++++++++++++++
```

M0: Job 0, Job 1

Hardships when Searching in $\mathbb{Y}$

```
+++++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
+++++++++++++++++++++++++++
```

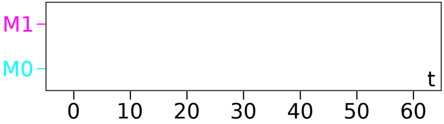
job 0
job 1

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

Hardships when Searching in $\mathbb{Y}$

++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2 2
job 0 | 0 10 1 20
job 1 | 0 10 1 20
++++++++++++++++++++++++

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

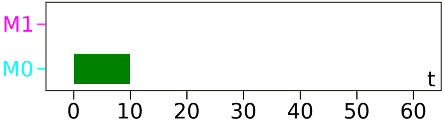


Hardships when Searching in $\mathbb{Y}$

job 0
job 1

```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
++++++++++++++++++++++++
```

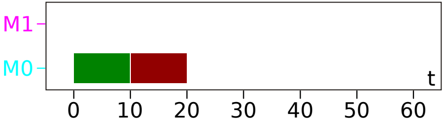
M0: Job 0, Job 1; M1: Job 0, Job 1



Hardships when Searching in $\mathbb{Y}$

```
      ++++++++++++++++++++++++
      instance A with 2 jobs and 2 machines
      2  2
job 0 0  10   1  20
job 1 0  10   1  20
      ++++++++++++++++++++++++
```

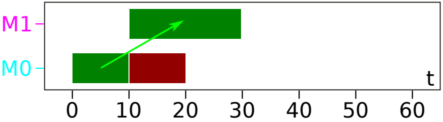
M0: Job 0, Job 1; M1: Job 0, Job 1



Hardships when Searching in $\mathbb{Y}$

```
       +++++++++++++++++++++++
       instance A with 2 jobs and 2 machines
       2  2
job 0  0  10   1  20
job 1  0  10   1  20
       +++++++++++++++++++++++
```

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

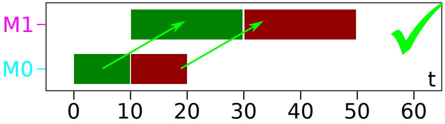


Hardships when Searching in $\mathbb{Y}$

```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10    1  20
0  10    1  20
++++++++++++++++++++++++
```

job 0
job 1

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

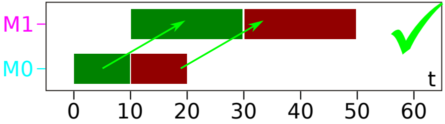


Hardships when Searching in $\mathbb{Y}$

```
+++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10    1  20
0  10    1  20
+++++++++++++++++++++++++
```

job 0
job 1

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



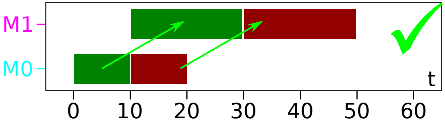
M0: Job 0, Job 1

Hardships when Searching in $\mathbb{Y}$

```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
++++++++++++++++++++++++
```

job 0
job 1

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

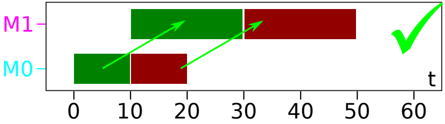


M0: Job 0, Job 1; M1: Job 1, Job 0

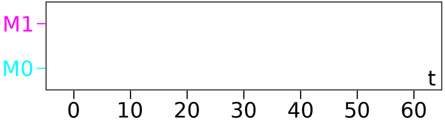
Hardships when Searching in $\mathbb{Y}$

```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
job 0   0  10    1  20
job 1   0  10    1  20
++++++++++++++++++++++++
```

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



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

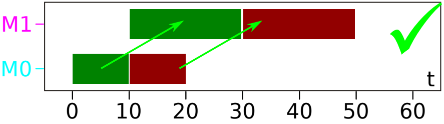


Hardships when Searching in $\mathbb{Y}$

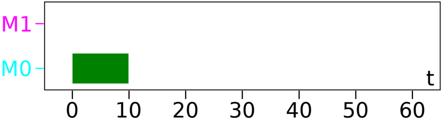
```
+++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10    1  20
0  10    1  20
+++++++++++++++++++++++++
```

job 0
job 1

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



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

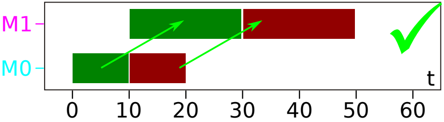


Hardships when Searching in $\mathbb{Y}$

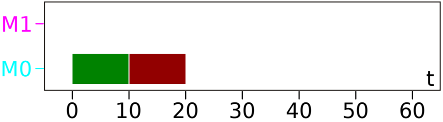
job 0
job 1

```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
++++++++++++++++++++++++
```

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



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

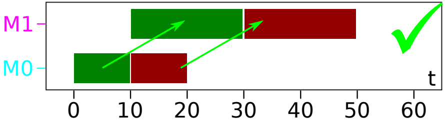


Hardships when Searching in $\mathbb{Y}$

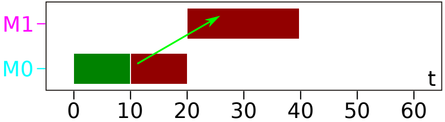
```
+++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10    1  20
0  10    1  20
+++++++++++++++++++++++++
```

job 0
job 1

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



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

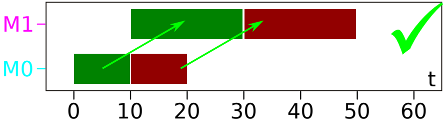


Hardships when Searching in $\mathbb{Y}$

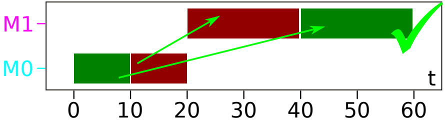
```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10    1  20
0  10    1  20
++++++++++++++++++++++++
```

job 0
job 1

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



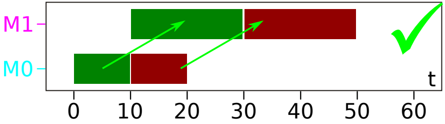
M0: Job 0, Job 1; M1: Job 1, Job 0



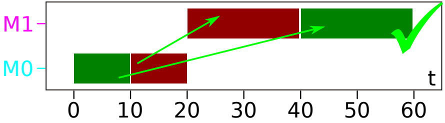
Hardships when Searching in $\mathbb{Y}$

```
        ++++++++++++++++++++++++
        instance A with 2 jobs and 2 machines
        2  2
job 0   0  10   1  20
job 1   0  10   1  20
        ++++++++++++++++++++++++
```

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



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



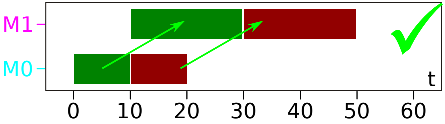
M0: Job 1, Job 0

Hardships when Searching in $\mathbb{Y}$

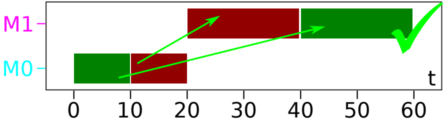
```
+++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10    1  20
0  10    1  20
+++++++++++++++++++++++++
```

job 0
job 1

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



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



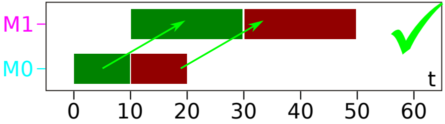
M0: Job 1, Job 0; M1: Job 0, Job 1

Hardships when Searching in $\mathbb{Y}$

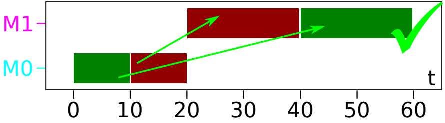
```
job 0
job 1
```

+++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2 2
job 0 — 0 10 1 20
job 1 — 0 10 1 20
+++++++++++++++++++++++

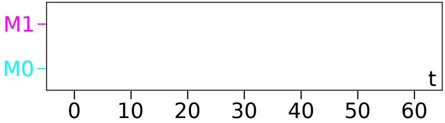
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

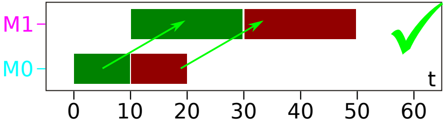


Hardships when Searching in $\mathbb{Y}$

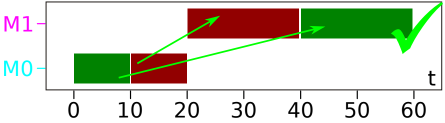
```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
++++++++++++++++++++++++
```

job 0
job 1

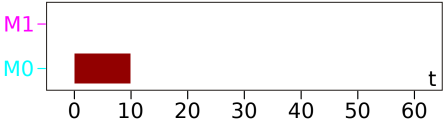
M0: Job 0, Job 1; M1: Job 0, Job 1



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



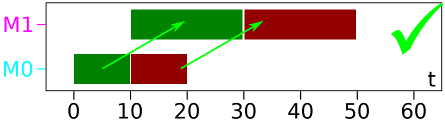
M0: Job 1, Job 0; M1: Job 0, Job 1



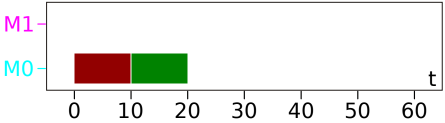
Hardships when Searching in $\mathbb{Y}$

```
+++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
job 0   0  10    1  20
job 1   0  10    1  20
+++++++++++++++++++++++++
```

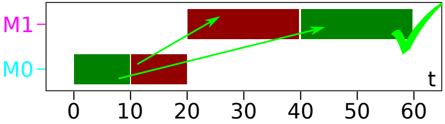
M0: Job 0, Job 1; M1: Job 0, Job 1



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



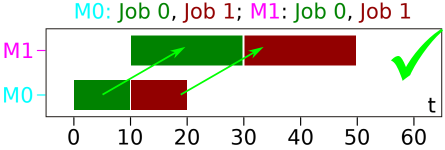
M0: Job 0, Job 1; M1: Job 1, Job 0

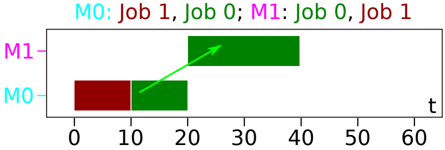


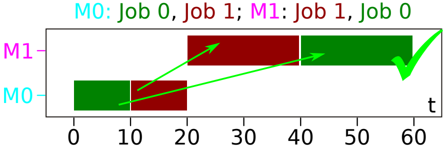
Hardships when Searching in $\mathbb{Y}$

```
+++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
+++++++++++++++++++++++
```

job 0
job 1







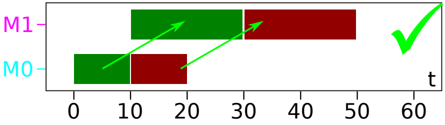
Hardships when Searching in $\mathbb{Y}$

```
++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2 2
0 10  1 20
0 10  1 20
++++++++++++++++++++++++
```

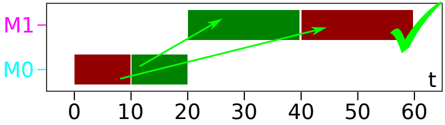
job 0 — 0 10 1 20

job 1 — 0 10 1 20

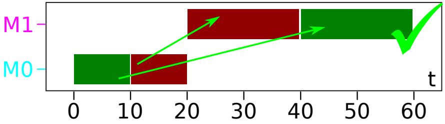
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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



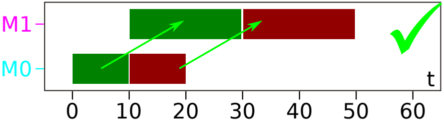
Hardships when Searching in $\mathbb{Y}$

```
++++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
++++++++++++++++++++++++++
```

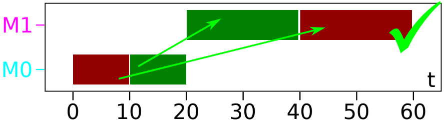
job 0

job 1

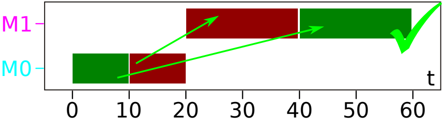
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

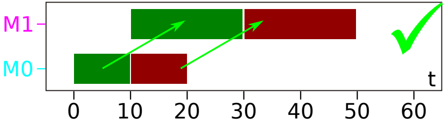


M0: Job 1, Job 0

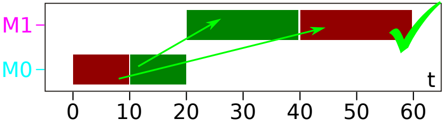
Hardships when Searching in $\mathbb{Y}$

```
          ++++++++++++++++++++++++
          instance A with 2 jobs and 2 machines
          2  2
job 0     0  10    1  20
job 1     0  10    1  20
          ++++++++++++++++++++++++
```

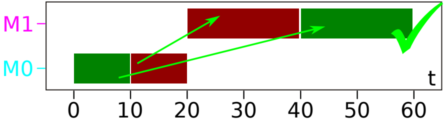
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

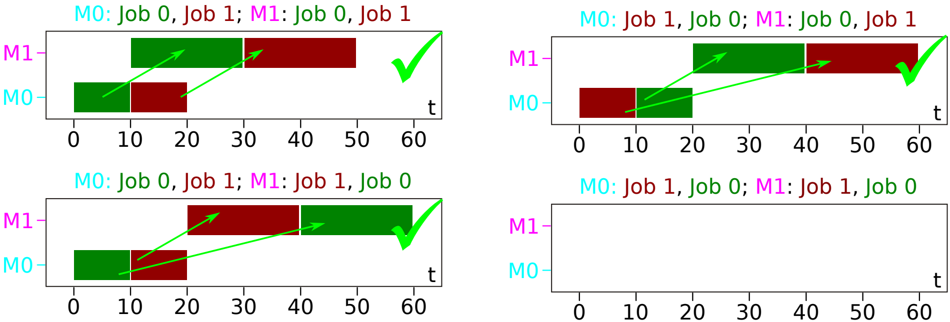


M0: Job 1, Job 0; M1: Job 1, Job 0

Hardships when Searching in $\mathbb{Y}$

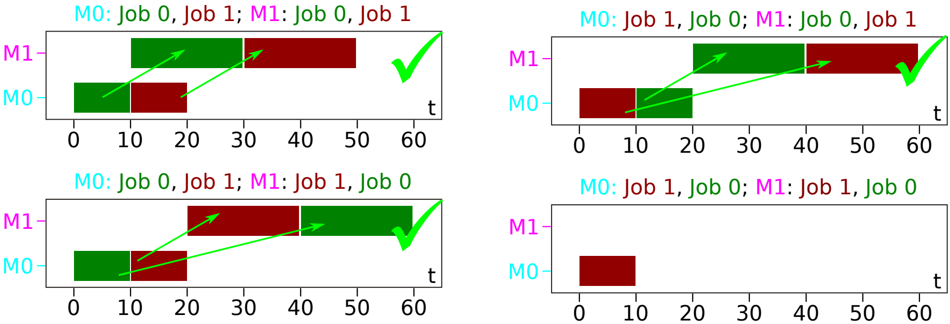
```
++++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
++++++++++++++++++++++++++
```

job 0
job 1



Hardships when Searching in $\mathbb{Y}$

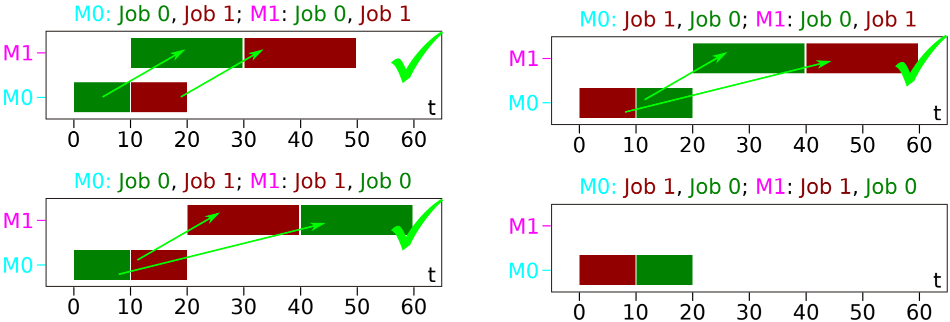
```
        ++++++++++++++++++++++++
        instance A with 2 jobs and 2 machines
        2  2
job 0   0  10    1  20
job 1   0  10    1  20
        ++++++++++++++++++++++++
```



Hardships when Searching in $\mathbb{Y}$

```
+++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
+++++++++++++++++++++++++
```

job 0
job 1

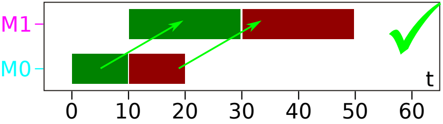


Hardships when Searching in $\mathbb{Y}$

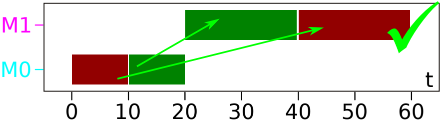
```
+++++++++++++++++++++++++
instance A with 2 jobs and 2 machines
2  2
0  10   1  20
0  10   1  20
+++++++++++++++++++++++++
```

job 0
job 1

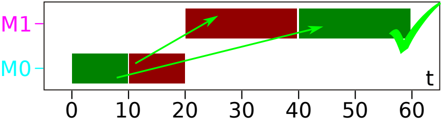
M0: Job 0, Job 1; M1: Job 0, Job 1



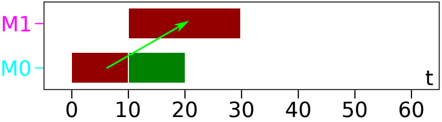
M0: Job 1, Job 0; M1: Job 0, Job 1



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



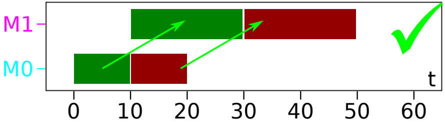
M0: Job 1, Job 0; M1: Job 1, Job 0



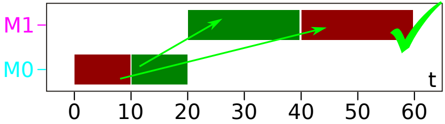
Hardships when Searching in $\mathbb{Y}$

	+++++++++++++++++++++++
	instance A with 2 jobs and 2 machines
	2 2
job 0	0 10 1 20
job 1	0 10 1 20
	+++++++++++++++++++++++

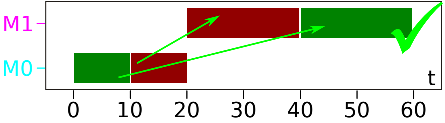
M0: Job 0, Job 1; M1: Job 0, Job 1



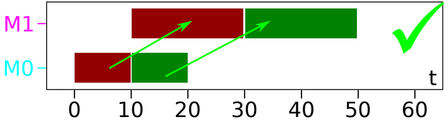
M0: Job 1, Job 0; M1: Job 0, Job 1



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



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



Hardships when Searching in $\mathbb{Y}$

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

Hardships when Searching in $\mathbb{Y}$

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

Hardships when Searching in $\mathbb{Y}$

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

Hardships when Searching in $\mathbb{Y}$

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

Hardships when Searching in $\mathbb{Y}$

job 0

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

Hardships when Searching in $\mathbb{Y}$

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

job 0
job 1

Hardships when Searching in $\mathbb{Y}$

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

job 0
job 1

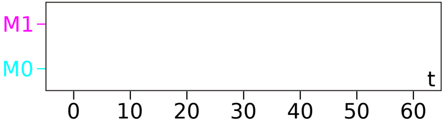
M0: Job 0, Job 1; M1: Job 0, Job 1

Hardships when Searching in $\mathbb{Y}$

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

job 0: 0 10 1 20
job 1: 1 20 0 10

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

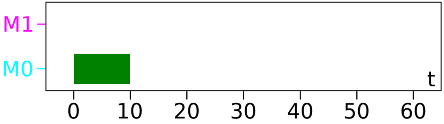


Hardships when Searching in $\mathbb{Y}$

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

job 0
job 1

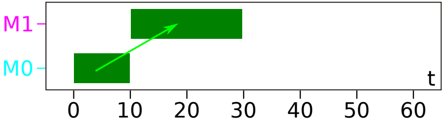
M0: Job 0, Job 1; M1: Job 0, Job 1



Hardships when Searching in $\mathbb{Y}$

```
        ++++++++++++++++++++++++
        instance B with 2 jobs and 2 machines
        2  2
job 0   0  10   1  20
job 1   1  20   0  10
        ++++++++++++++++++++++++
```

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



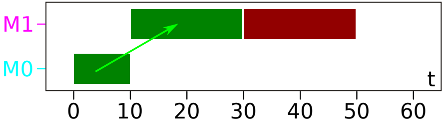
Hardships when Searching in $\mathbb{Y}$

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

job 0

job 1

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

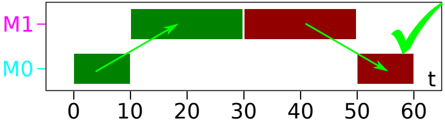


Hardships when Searching in $\mathbb{Y}$

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

job 0
job 1

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

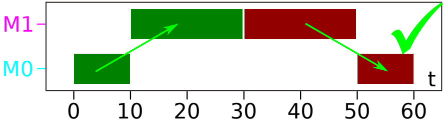


Hardships when Searching in $\mathbb{Y}$

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

job 0
job 1

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



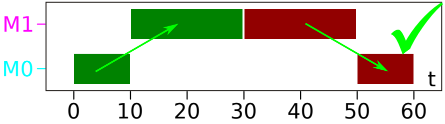
M0: Job 0, Job 1; M1: Job 1, Job 0

Hardships when Searching in $\mathbb{Y}$

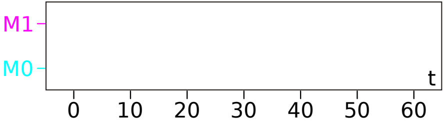
```
+++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10    1  20
1  20    0  10
+++++++++++++++++++++++++
```

job 0
job 1

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

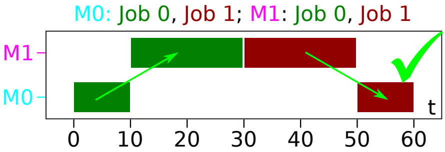


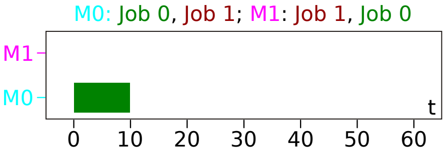
M0: Job 0, Job 1; M1: Job 1, Job 0



Hardships when Searching in $\mathbb{Y}$

```
        +++++++++++++++++++++++++
        instance B with 2 jobs and 2 machines
        2  2
job 0   0  10    1  20
job 1   1  20    0  10
        +++++++++++++++++++++++++
```

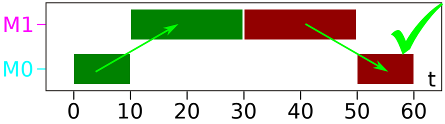




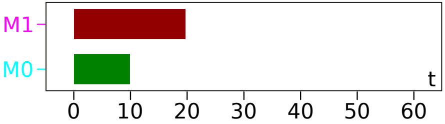
Hardships when Searching in $\mathbb{Y}$

```
+++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
job 0   0  10    1  20
job 1   1  20    0  10
+++++++++++++++++++++++
```

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



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

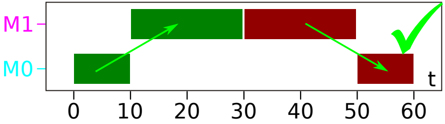


Hardships when Searching in $\mathbb{Y}$

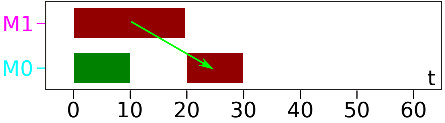
```
+++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10   1  20
1  20   0  10
+++++++++++++++++++++++++
```

job 0: 0 10 1 20
job 1: 1 20 0 10

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



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

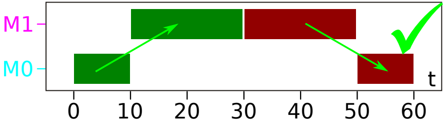


Hardships when Searching in $\mathbb{Y}$

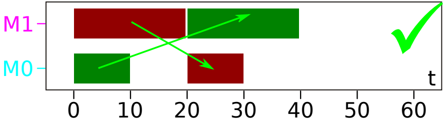
```
+++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10    1  20
1  20    0  10
+++++++++++++++++++++++++
```

job 0
job 1

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



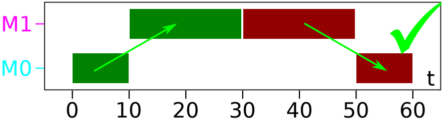
M0: Job 0, Job 1; M1: Job 1, Job 0



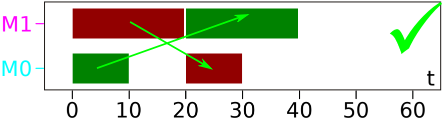
Hardships when Searching in $\mathbb{Y}$

```
+++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10   1  20      job 0
1  20   0  10      job 1
+++++++++++++++++++++++
```

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



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



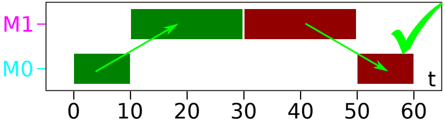
M0: Job 1, Job 0; M1: Job 0, Job 1

Hardships when Searching in $\mathbb{Y}$

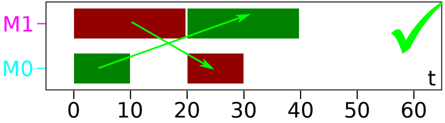
```
++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10   1  20
1  20   0  10
++++++++++++++++++++++++
```

job 0
job 1

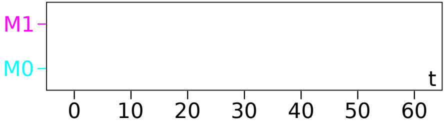
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

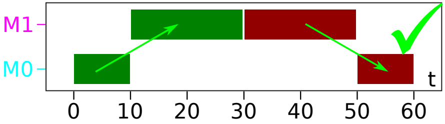


Hardships when Searching in $\mathbb{Y}$

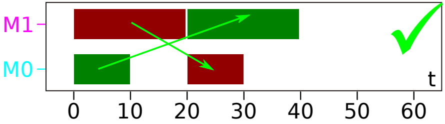
```
++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10   1  20
1  20   0  10
++++++++++++++++++++++++
```

job 0 | job 1

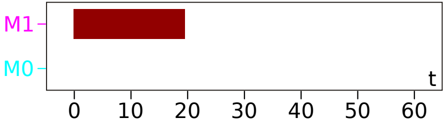
M0: Job 0, Job 1; M1: Job 0, Job 1



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



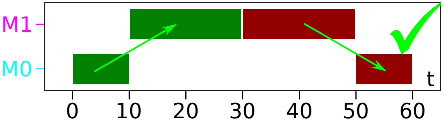
M0: Job 1, Job 0; M1: Job 0, Job 1



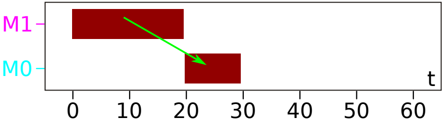
Hardships when Searching in $\mathbb{Y}$

```
++++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
job 0   0  10   1  20
job 1   1  20   0  10
++++++++++++++++++++++++++
```

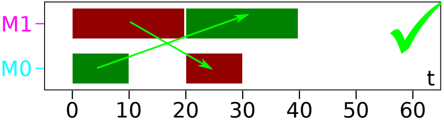
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

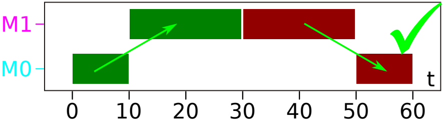


Hardships when Searching in $\mathbb{Y}$

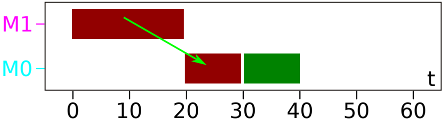
```
++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10   1  20
1  20   0  10
++++++++++++++++++++++++
```

job 0
job 1

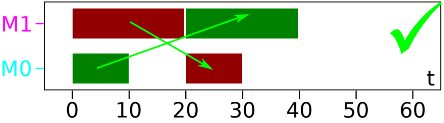
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

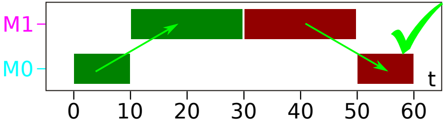


Hardships when Searching in $\mathbb{Y}$

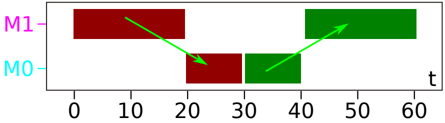
```
++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10   1  20
1  20   0  10
++++++++++++++++++++++++
```

job 0
job 1

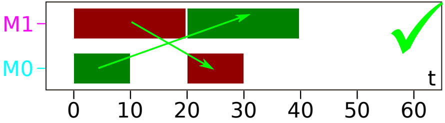
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

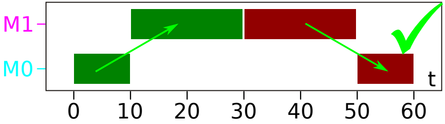


Hardships when Searching in $\mathbb{Y}$

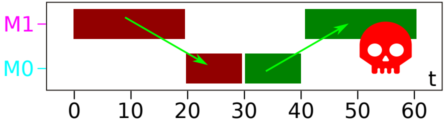
```
+++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10   1  20
1  20   0  10
+++++++++++++++++++++++++
```

job 0: 0 10 1 20
job 1: 1 20 0 10

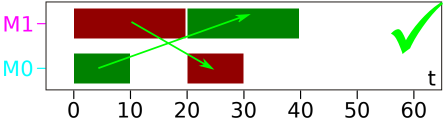
M0: Job 0, Job 1; M1: Job 0, Job 1



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



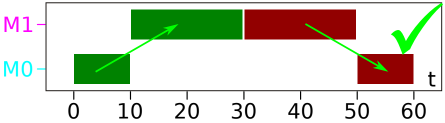
M0: Job 0, Job 1; M1: Job 1, Job 0



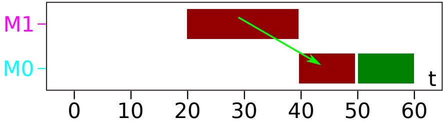
Hardships when Searching in $\mathbb{Y}$

```
      +++++++++++++++++++++++++
      instance B with 2 jobs and 2 machines
      2  2
job 0 0  10   1  20
job 1 1  20   0  10
      +++++++++++++++++++++++++
```

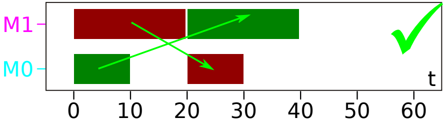
M0: Job 0, Job 1; M1: Job 0, Job 1



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



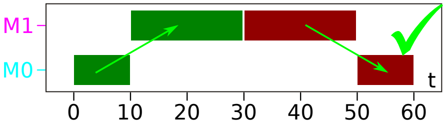
M0: Job 0, Job 1; M1: Job 1, Job 0



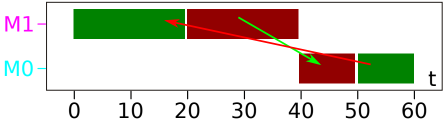
Hardships when Searching in $\mathbb{Y}$

```
        ++++++++++++++++++++++++
        instance B with 2 jobs and 2 machines
        2  2
job 0   0  10   1  20
job 1   1  20   0  10
        ++++++++++++++++++++++++
```

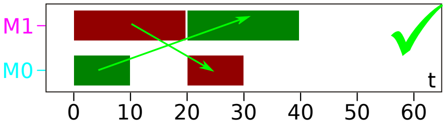
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

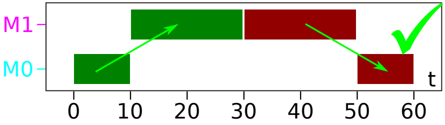


Hardships when Searching in $\mathbb{Y}$

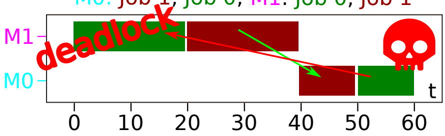
```
+++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10   1  20
1  20   0  10
+++++++++++++++++++++++
```

job 0
job 1

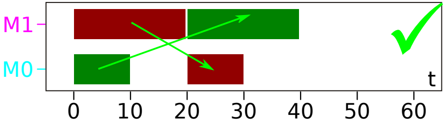
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

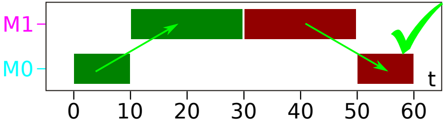


Hardships when Searching in $\mathbb{Y}$

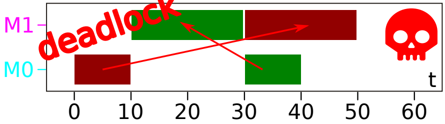
```
+++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10   1  20
1  20   0  10
+++++++++++++++++++++++++
```

job 0
job 1

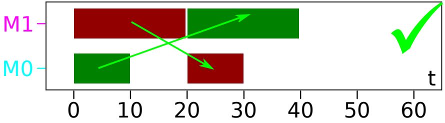
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

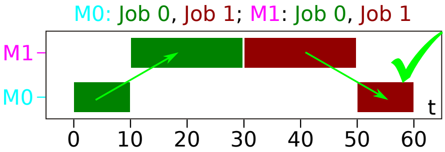


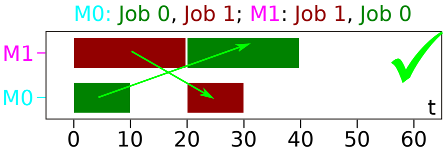
Hardships when Searching in $\mathbb{Y}$

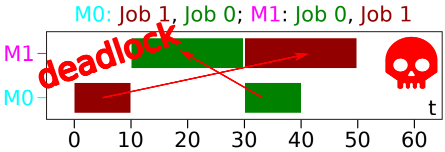
```
+++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2 2
0 10  1 20
1 20  0 10
+++++++++++++++++++++++++
```

job 0
job 1

Machine 0 should begin by doing job 1.





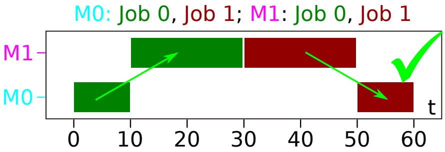


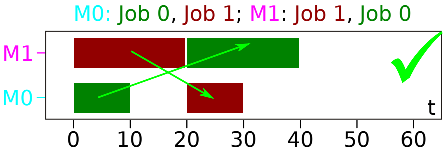
Hardships when Searching in $\mathbb{Y}$

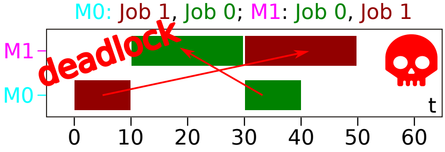
```
+++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10   1  20
1  20   0  10
+++++++++++++++++++++++++
```

job 0
job 1

Machine 0 should begin by doing job 1.
Job 1 can only start on machine 0 after it has been finished on machine 1.







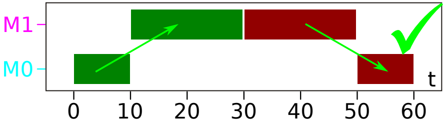
Hardships when Searching in $\mathbb{Y}$

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

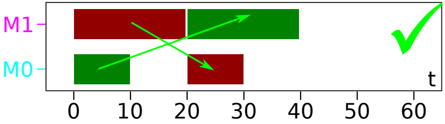
job 0
job 1

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.

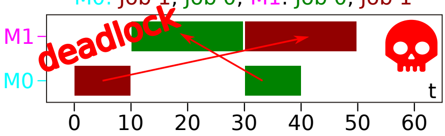
M0: Job 0, Job 1; M1: Job 0, Job 1



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



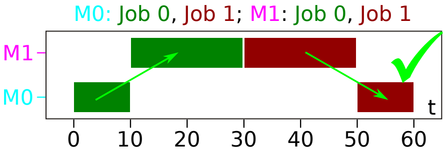
M0: Job 1, Job 0; M1: Job 0, Job 1

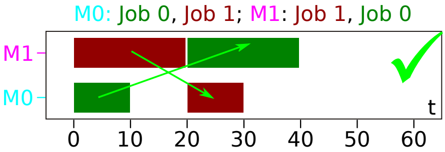


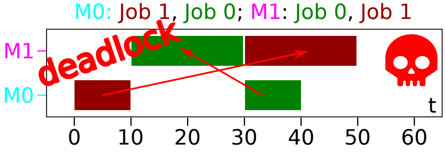
Hardships when Searching in $\mathbb{Y}$

```
job 0
job 1
+++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10    1  20
1  20    0  10
+++++++++++++++++++++++++
```

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.





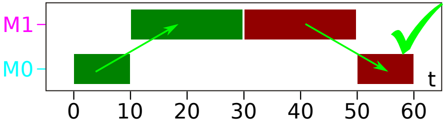


Hardships when Searching in $\mathbb{Y}$

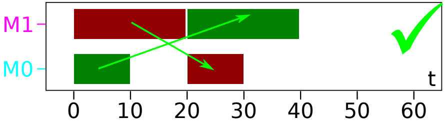
```
          ++++++++++++++++++++++++
          instance B with 2 jobs and 2 machines
          2  2
job 0     0  10   1  20
job 1     1  20   0  10
          ++++++++++++++++++++++++
```

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.

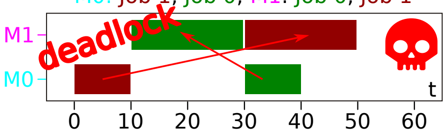
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

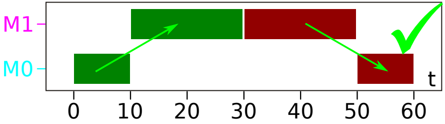


Hardships when Searching in $\mathbb{Y}$

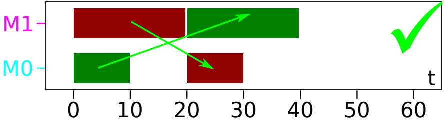
```
++++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2 2
0 10   1 20      (job 0)
1 20   0 10      (job 1)
++++++++++++++++++++++++++
```

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.

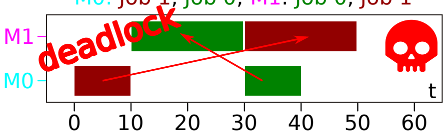
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

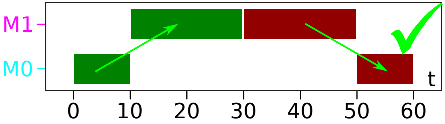


Hardships when Searching in $\mathbb{Y}$

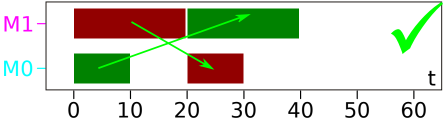
```
+++++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2 2
job 0   0 10  1 20
job 1   1 20  0 10
+++++++++++++++++++++++++
```

A cyclic blockage has appeared: no job can be executed on any machine if we follow this schedule.

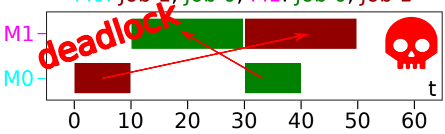
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

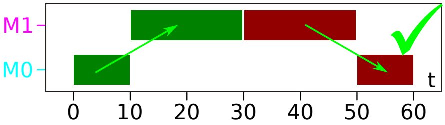


Hardships when Searching in $\mathbb{Y}$

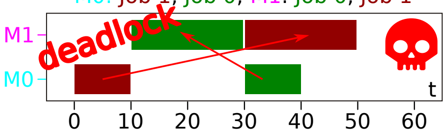
```
      +++++++++++++++++++++++++
      instance B with 2 jobs and 2 machines
      2  2
job 0 0  10   1  20
job 1 1  20   0  10
      +++++++++++++++++++++++++
```

A cyclic blockage has appeared: no job can be executed on any machine if we follow this schedule. This is called a deadlock.

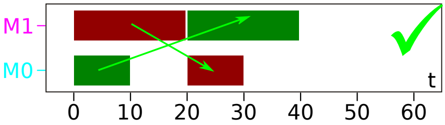
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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



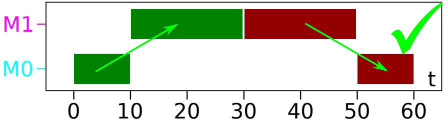
Hardships when Searching in $\mathbb{Y}$

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

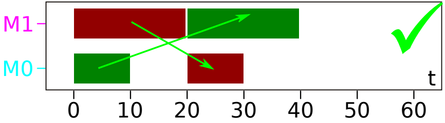
job 0
job 1

This is called a deadlock. The schedule is infeasible, because it cannot be executed or written down without breaking the precedence constraint.

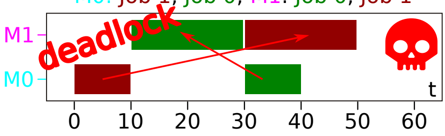
M0: Job 0, Job 1; M1: Job 0, Job 1



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



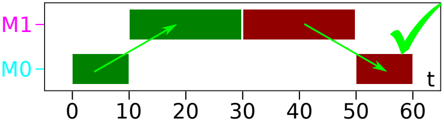
M0: Job 1, Job 0; M1: Job 0, Job 1



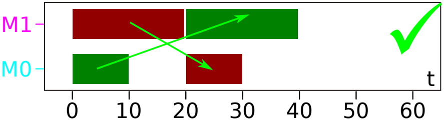
Hardships when Searching in $\mathbb{Y}$

```
    ++++++++++++++++++++++++++
    instance B with 2 jobs and 2 machines
    2  2
job 0  0  10   1  20
job 1  1  20   0  10
    ++++++++++++++++++++++++++
```

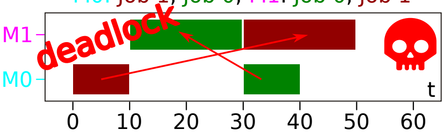
M0: Job 0, Job 1; M1: Job 0, Job 1



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



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

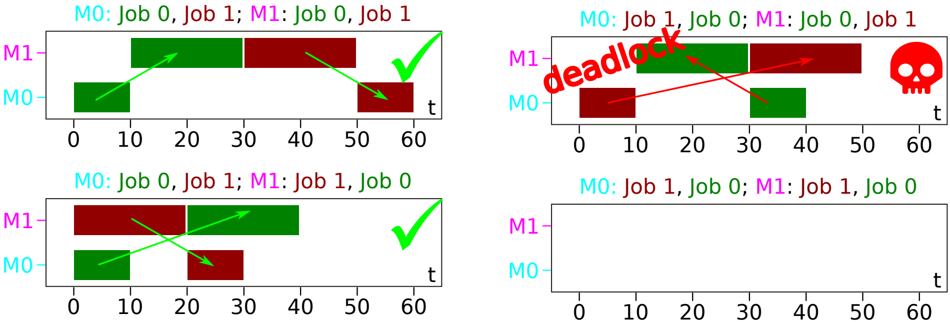


M0: Job 1, Job 0; M1: Job 1, Job 0

Hardships when Searching in $\mathbb{Y}$

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

job 0
job 1

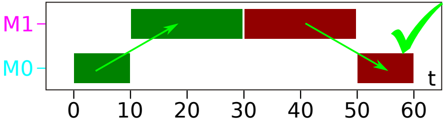


Hardships when Searching in $\mathbb{Y}$

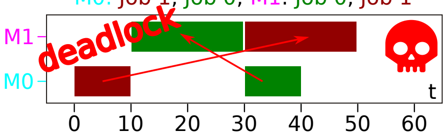
job 0
job 1

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

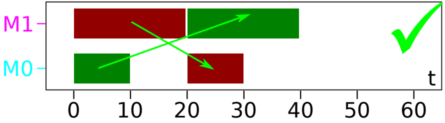
M0: Job 0, Job 1; M1: Job 0, Job 1



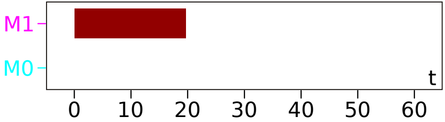
M0: Job 1, Job 0; M1: Job 0, Job 1



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



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

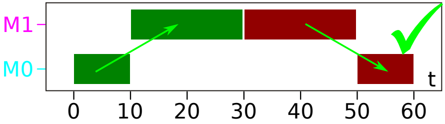


Hardships when Searching in $\mathbb{Y}$

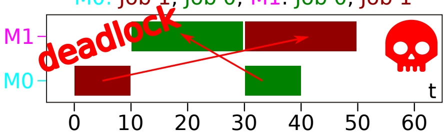
```
+++++++++++++++++++++++
instance B with 2 jobs and 2 machines
2  2
0  10   1  20
1  20   0  10
+++++++++++++++++++++++
```

job 0
job 1

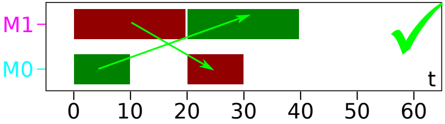
M0: Job 0, Job 1; M1: Job 0, Job 1



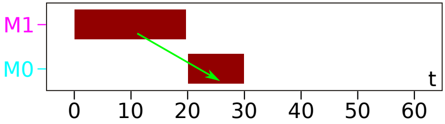
M0: Job 1, Job 0; M1: Job 0, Job 1



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



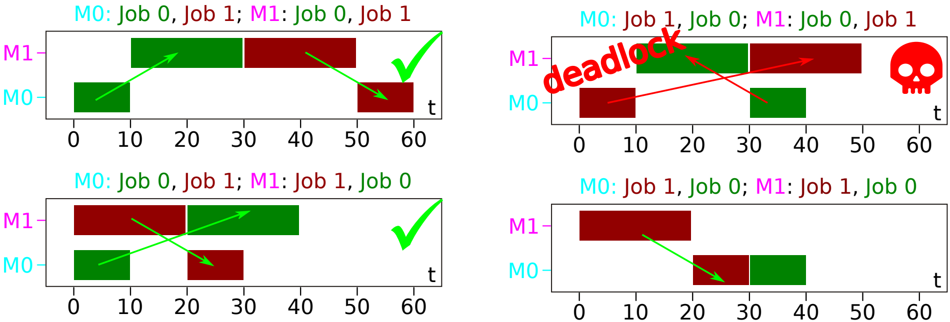
M0: Job 1, Job 0; M1: Job 1, Job 0



Hardships when Searching in $\mathbb{Y}$

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

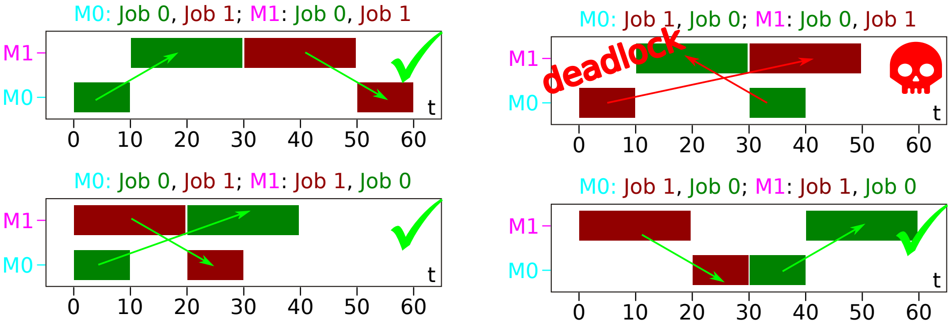
job 0
job 1



Hardships when Searching in $\mathbb{Y}$

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

job 0
job 1



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 $\mathbb{Y}$

- 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. . .

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 $\mathbb{Y}$

- 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 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 easy 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 $\mathbb{X}$

- The solution space $\mathbb{Y}$ is complicated and constrained.

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...

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.

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.

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.

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 $\mathbb{X}$.

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 $\mathbb{Y}$.

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 $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ for the JSSP look like?

One Search Space $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ for the JSSP look like?
- Let us revisit the demo problem instance.

One Search Space $\mathbb{X}$ for the JSSP

number m of machines

number n of jobs

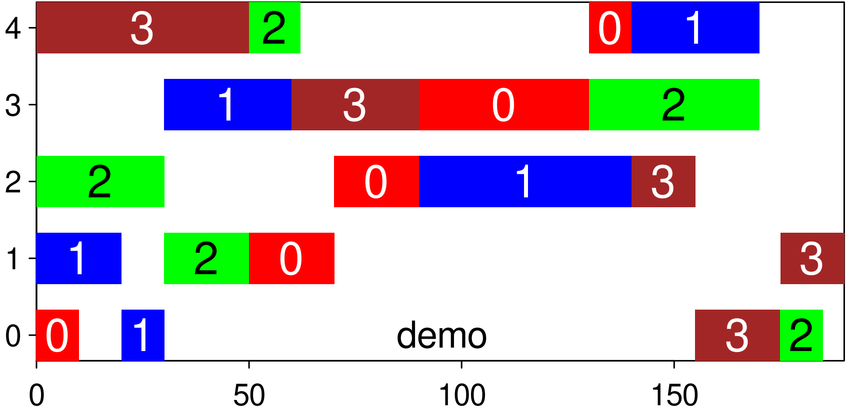
```
+++++++++++++++++++++++++
A simple demo
4 5
0 10  1 20  2 20  3 40  4 10
1 20  0 10  3 30  2 50  4 30
2 30  1 20  4 12  3 40  0 10
4 50  3 30  2 15  0 20  1 15
+++++++++++++++++++++++++
```

job 0
job 1
job 2
job 3

Each of the n jobs has m operations, each consisting of a machine index and a time requirement.

This is information that we have, which does not need to be stored in the elements $x \in \mathbb{X}$.

One Search Space $\mathbb{X}$ for the JSSP



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 $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $m * n = 20$ operation one ID, a number in $0 \dots 19$.

One Search Space $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $m * n = 20$ operation one ID, a number in $0 \dots 19$.
- Then, a linear string containing a permutation of these IDs could denote the exact processing order of the operations.

One Search Space $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $m * n = 20$ operation one ID, a number in $0 \dots 19$.
- 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 $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $m * n = 20$ 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 $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $m * n = 20$ operation one ID, a number in $0 \dots 19$.
- How can we use a linear encoding without deadlocks?

One Search Space $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $m * n = 20$ 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 $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $m * n = 20$ 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 $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $m * n = 20$ operation one ID, a number in $0 \dots 19$.
- 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.

One Search Space $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $m * n = 20$ operation one ID, a number in $0 \dots 19$.
- 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]

One Search Space $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $m * n = 20$ operation one ID, a number in $0 \dots 19$.
- 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.

One Search Space $\mathbb{X}$ for the JSSP

- So how could a simple search space $\mathbb{X}$ 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 $m * n = 20$ operation one ID, a number in $0 \dots 19$.
- 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

$\boldsymbol{x} \in \mathbb{X}$

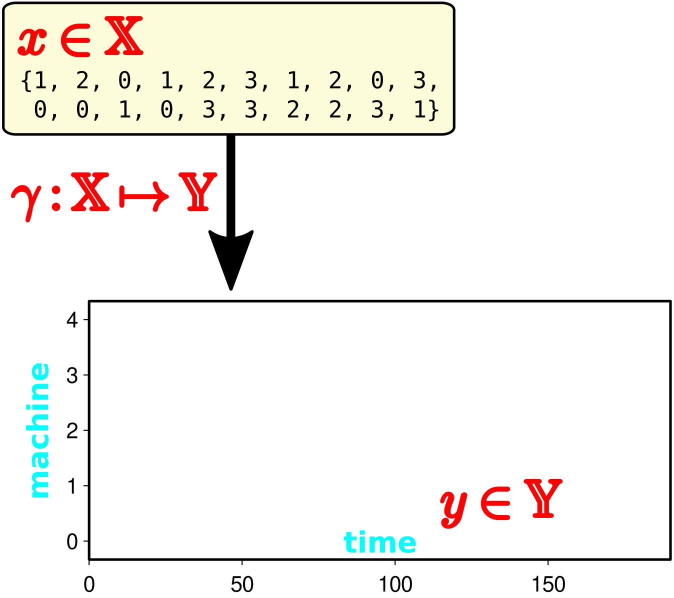
`{1, 2, 0, 1, 2, 3, 1, 2, 0, 3, 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}`

Demo Example for the Search Space

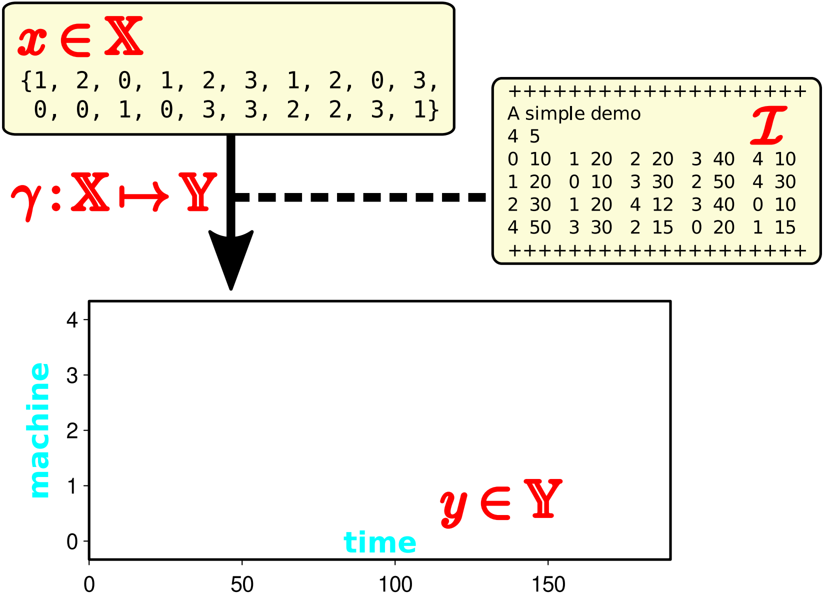
$x \in \mathbb{X}$

```
{1, 2, 0, 1, 2, 3, 1, 2, 0, 3,
 0, 0, 1, 0, 3, 3, 2, 2, 3, 1}
```

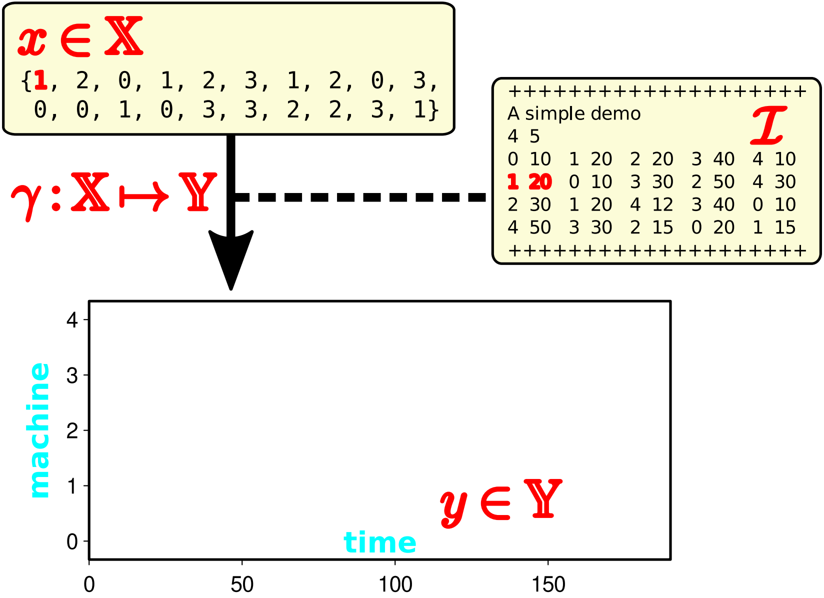
Demo Example for the Search Space



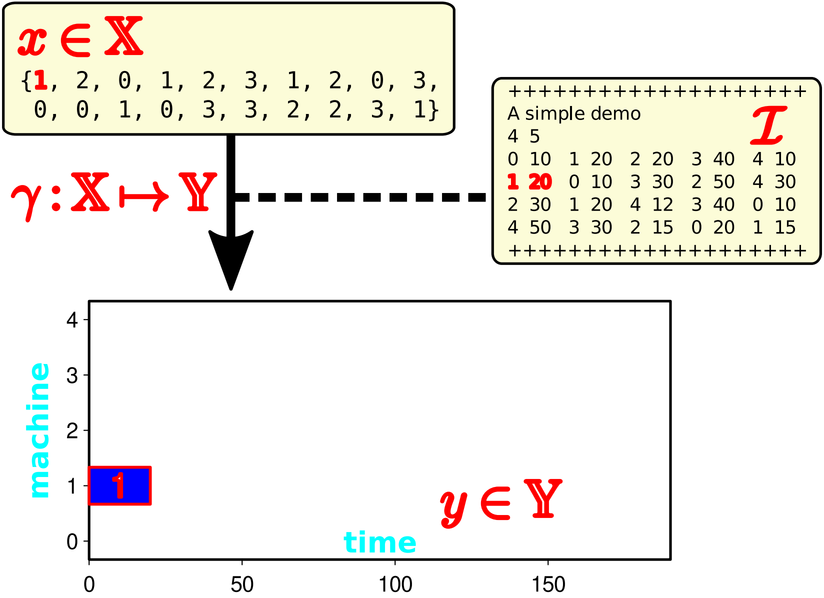
Demo Example for the Search Space



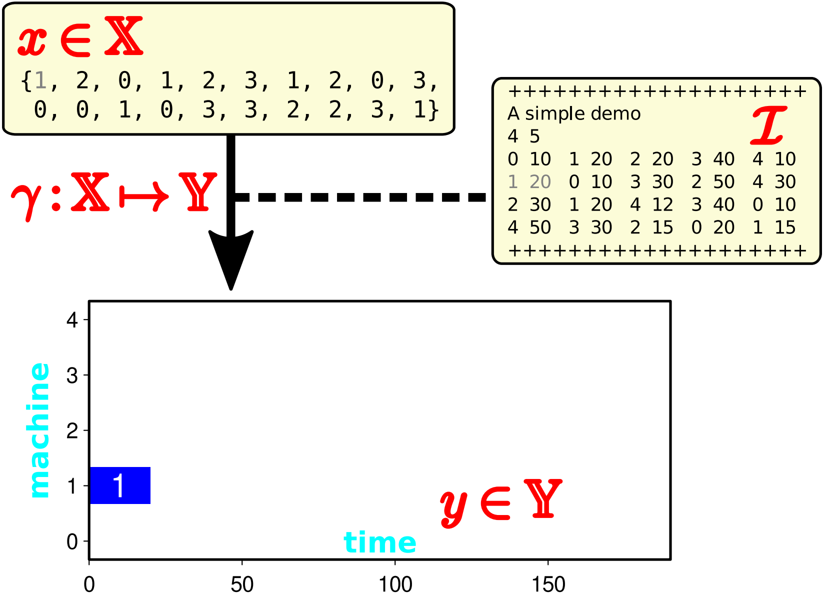
Demo Example for the Search Space



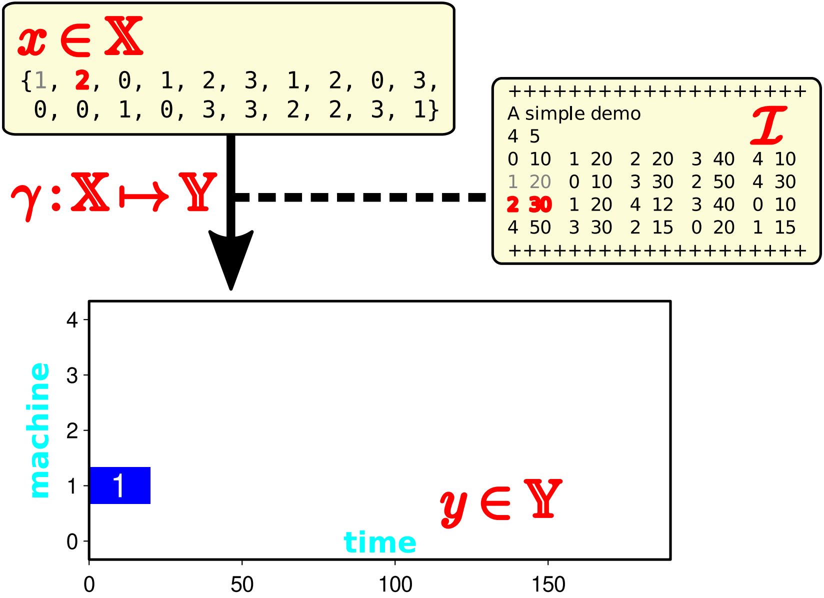
Demo Example for the Search Space



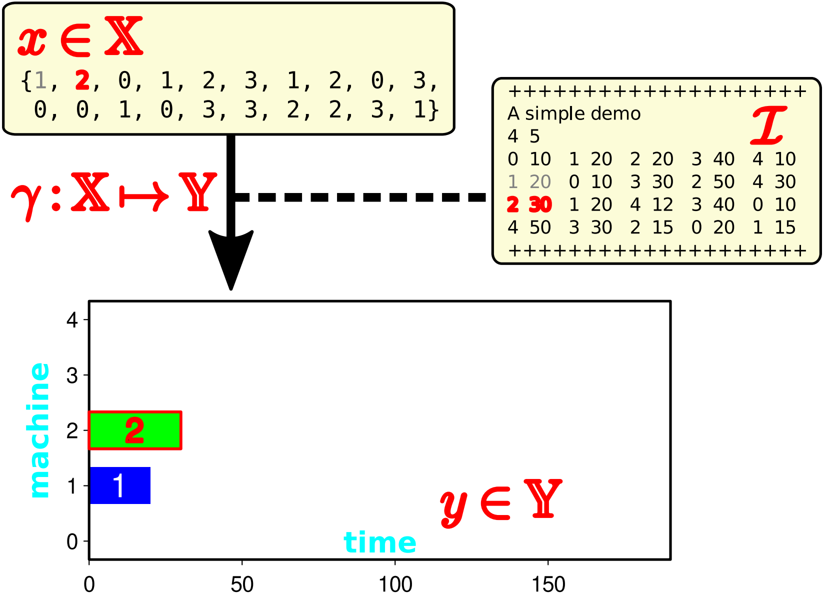
Demo Example for the Search Space



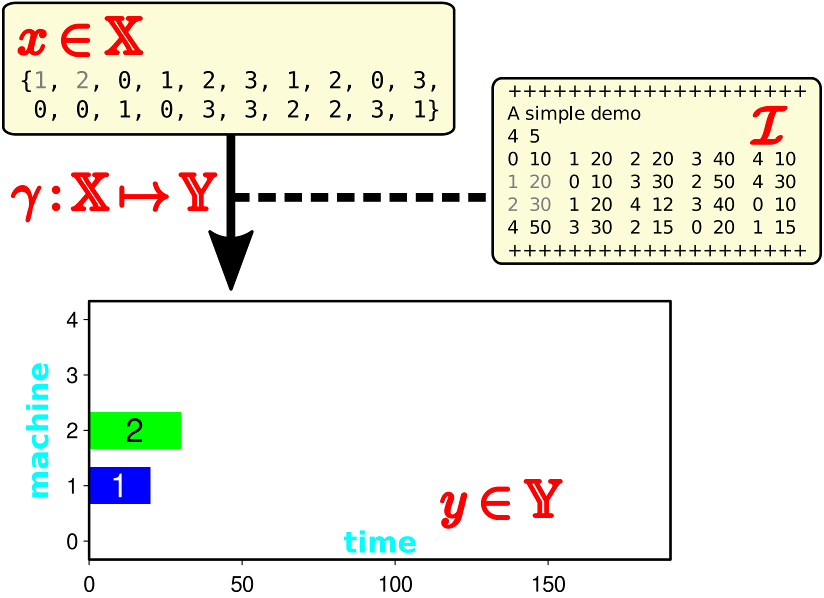
Demo Example for the Search Space



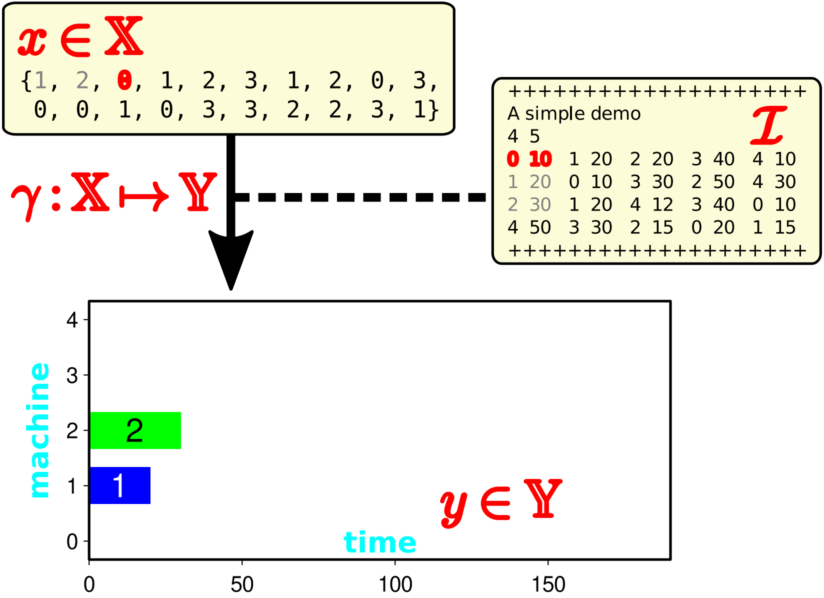
Demo Example for the Search Space



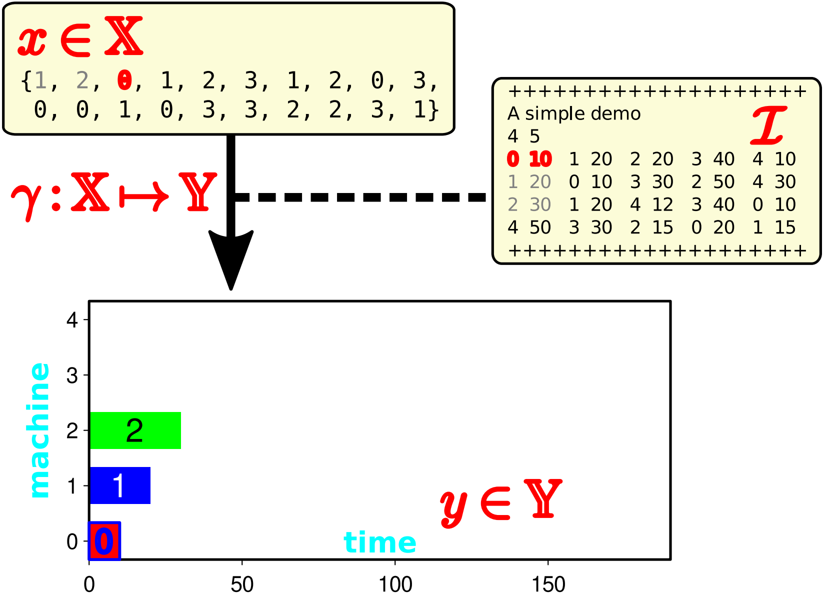
Demo Example for the Search Space



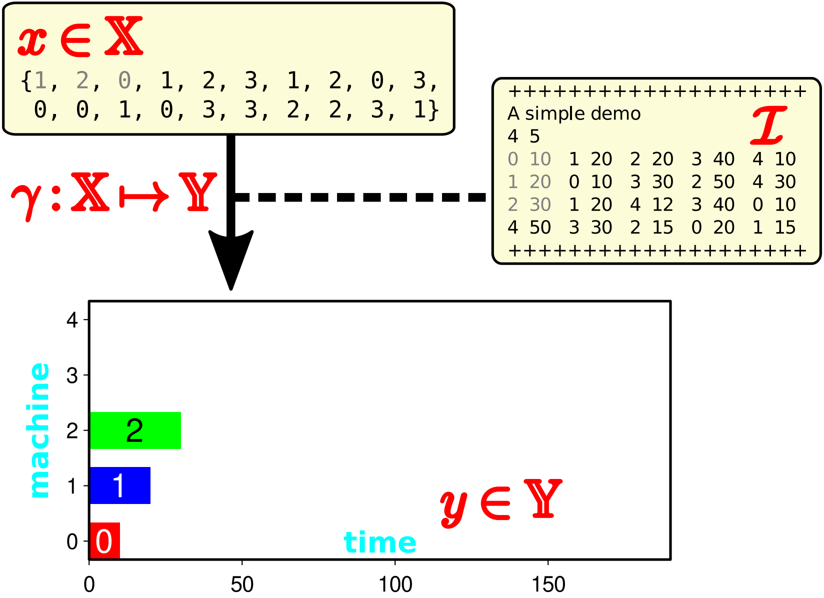
Demo Example for the Search Space



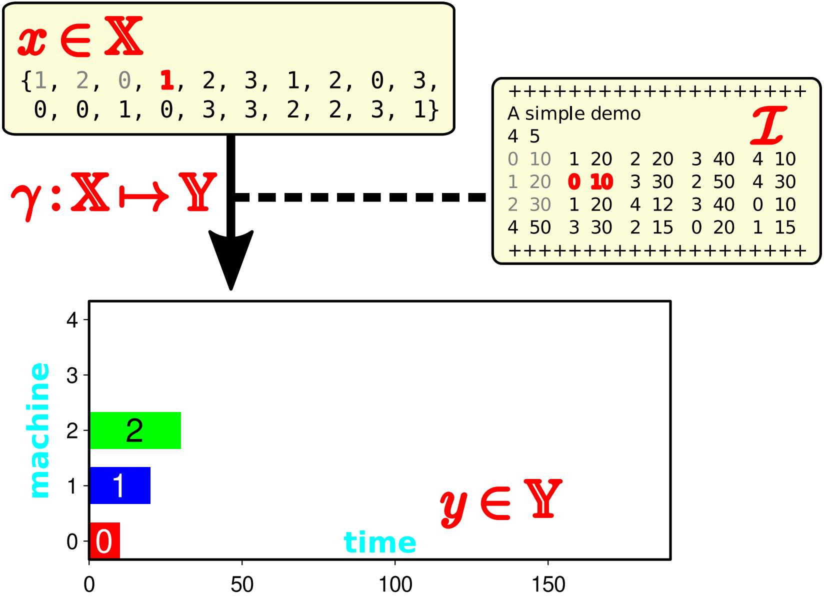
Demo Example for the Search Space



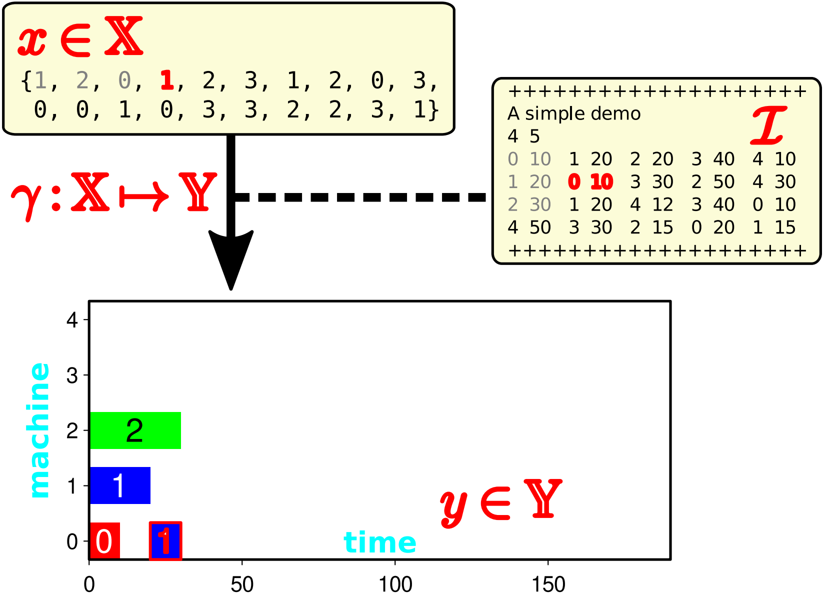
Demo Example for the Search Space



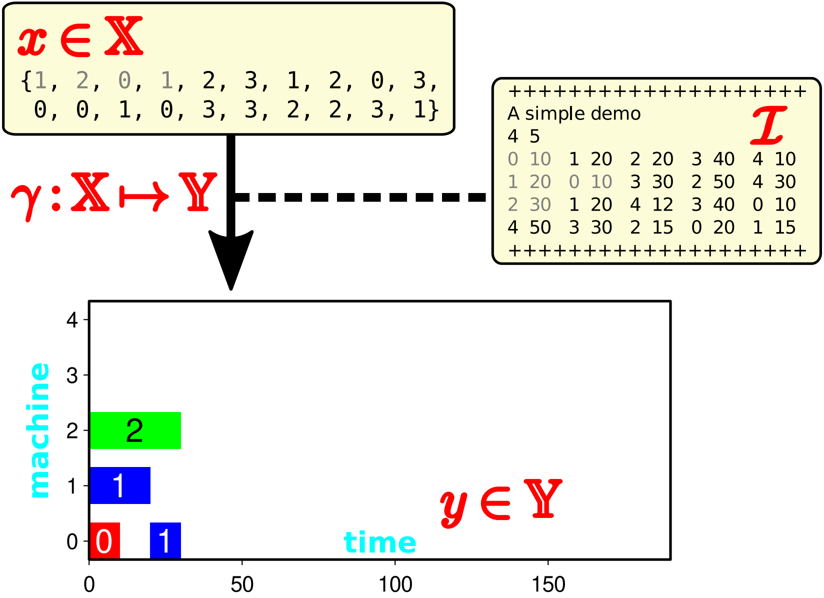
Demo Example for the Search Space



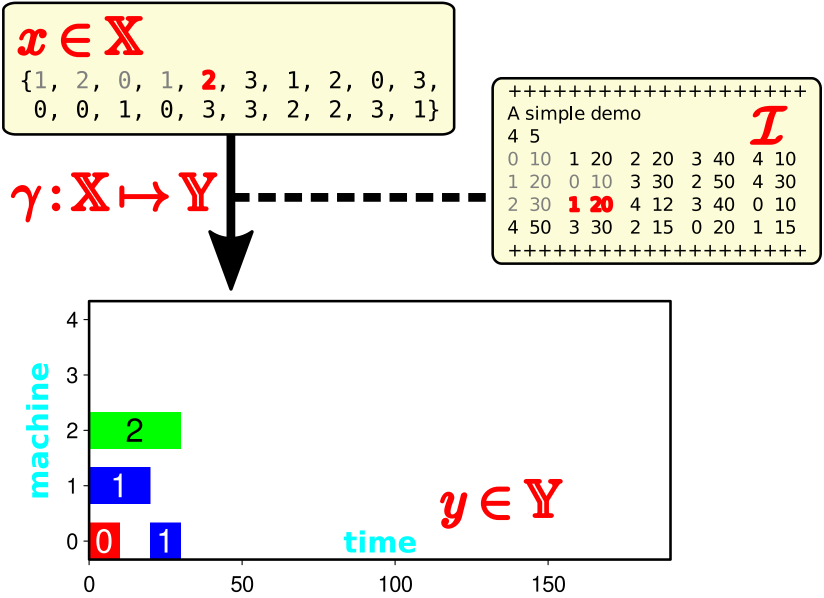
Demo Example for the Search Space



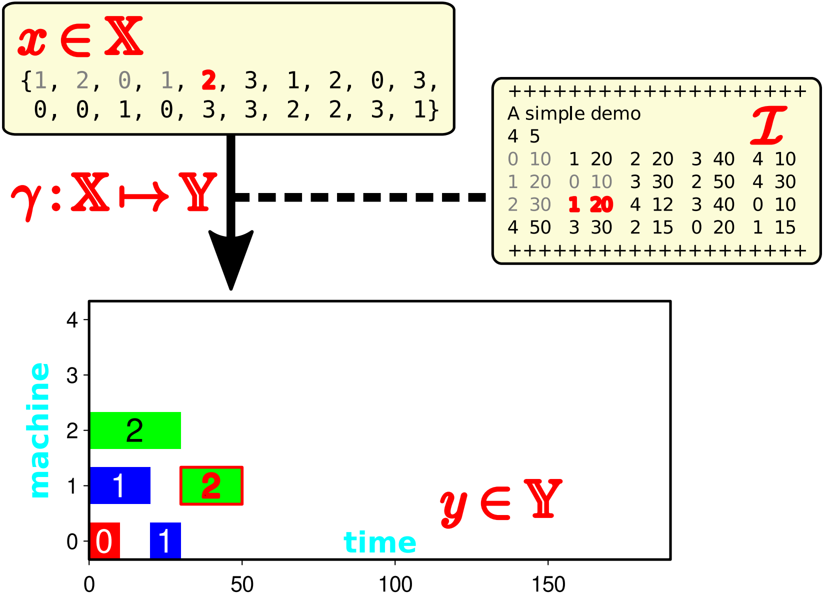
Demo Example for the Search Space



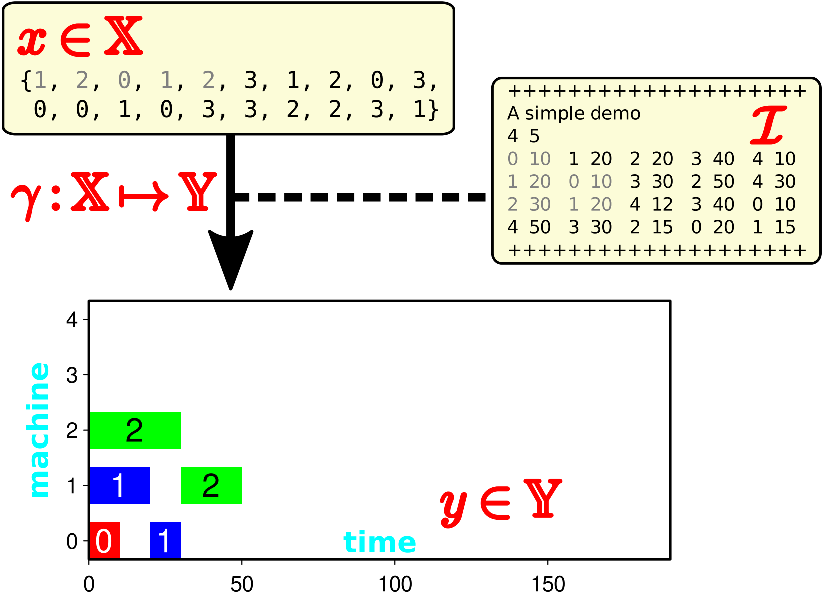
Demo Example for the Search Space



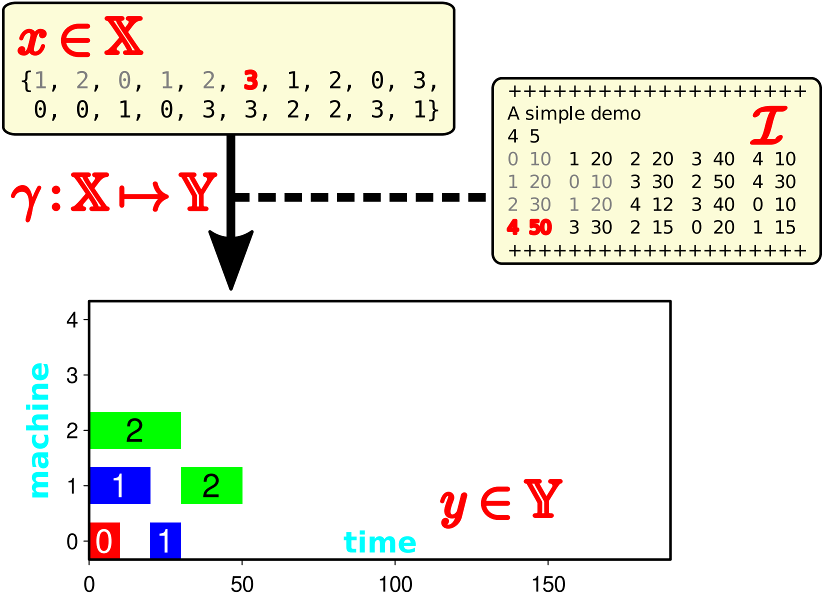
Demo Example for the Search Space



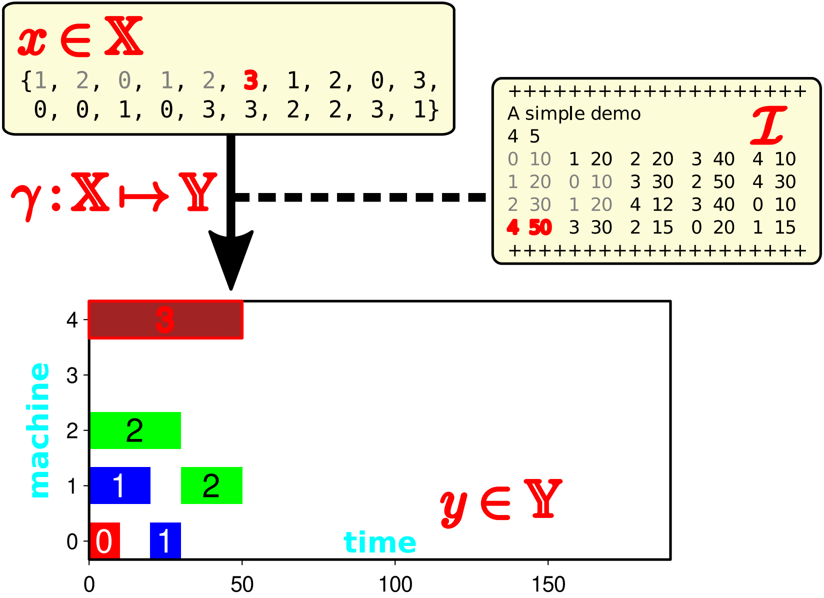
Demo Example for the Search Space



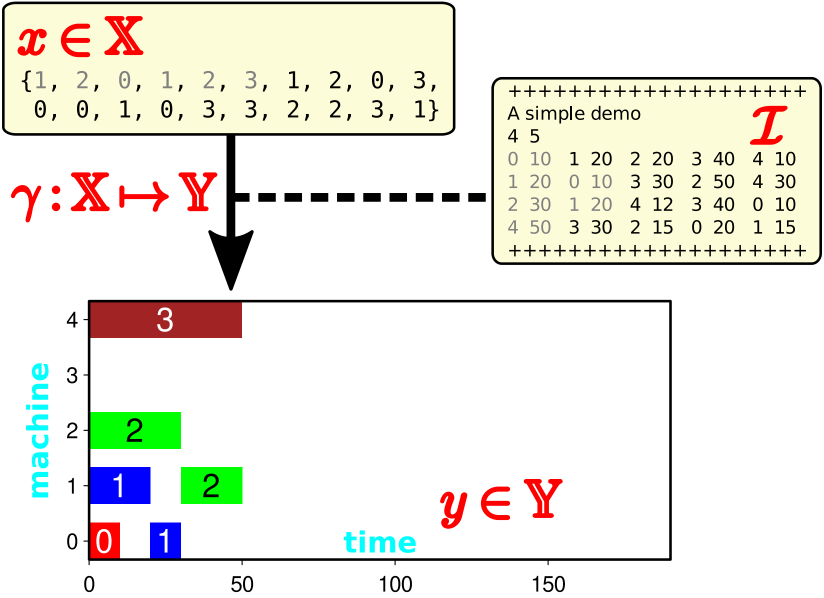
Demo Example for the Search Space



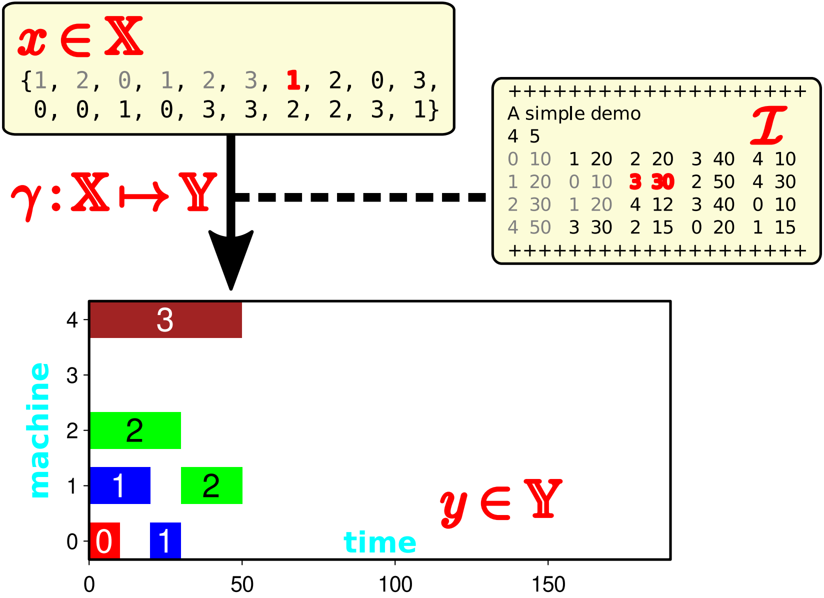
Demo Example for the Search Space



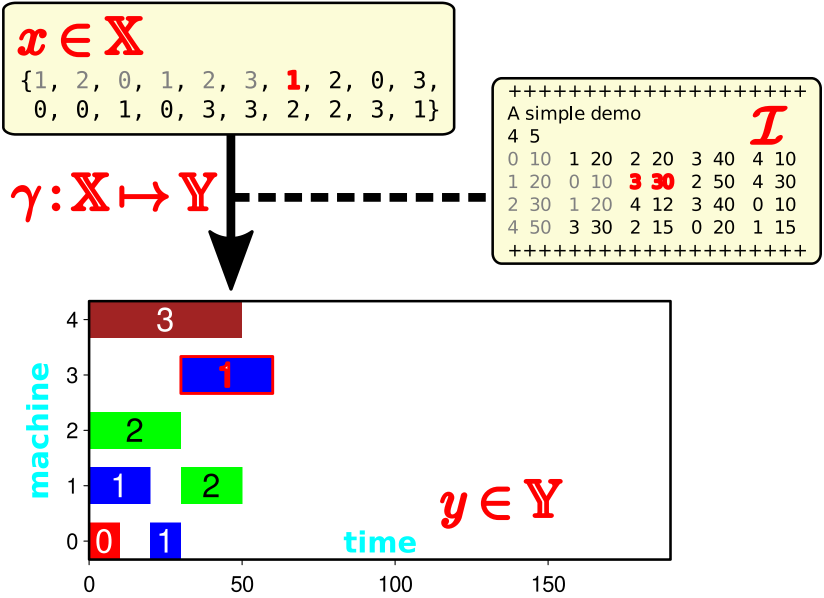
Demo Example for the Search Space



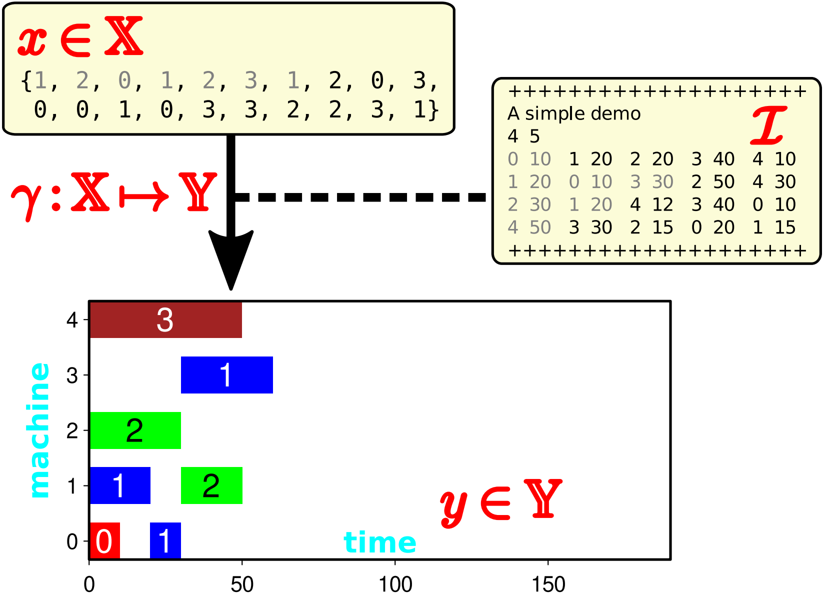
Demo Example for the Search Space



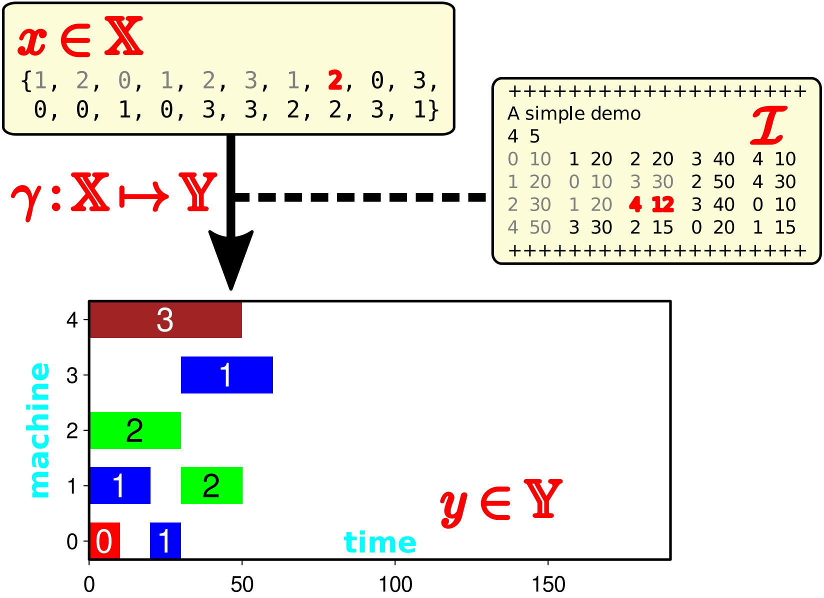
Demo Example for the Search Space



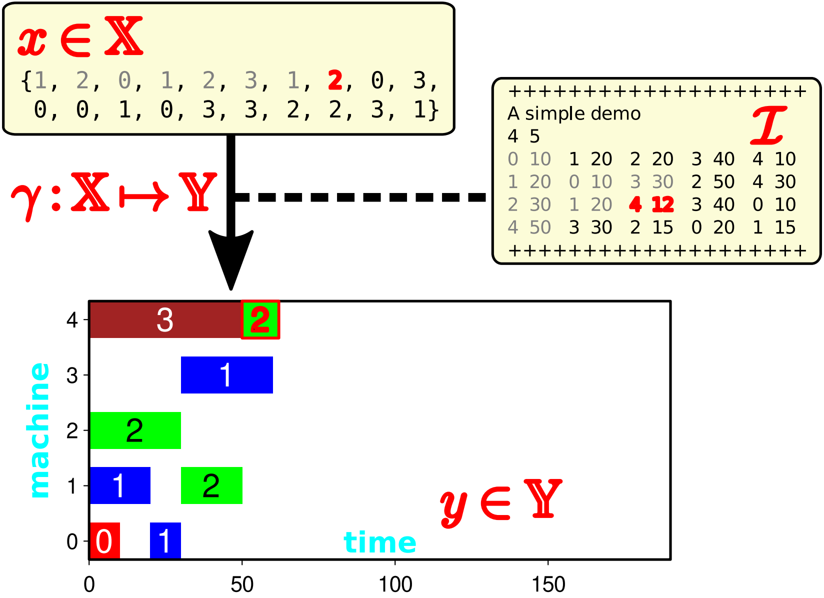
Demo Example for the Search Space



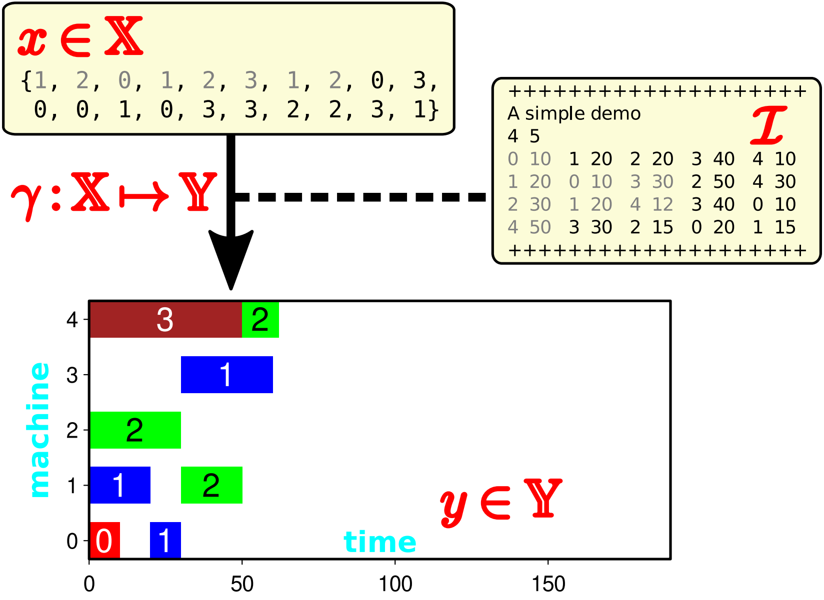
Demo Example for the Search Space



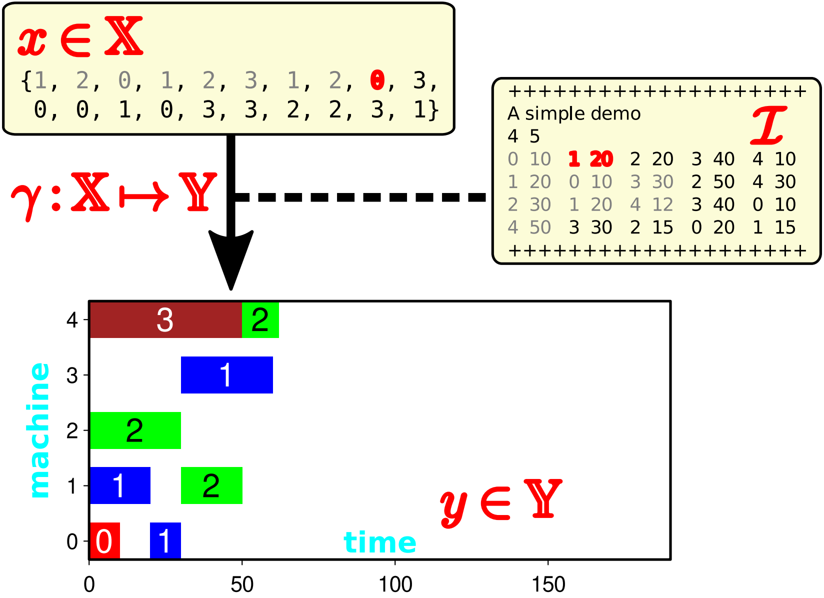
Demo Example for the Search Space



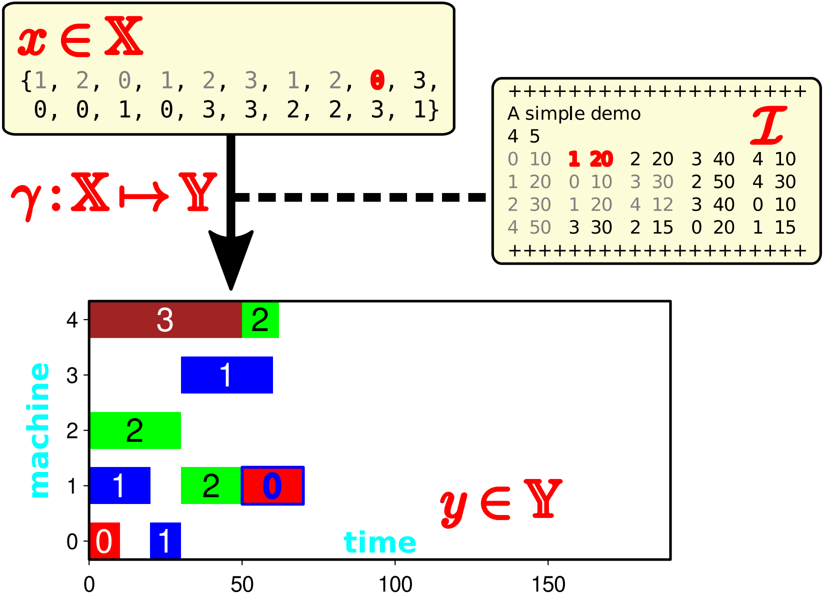
Demo Example for the Search Space



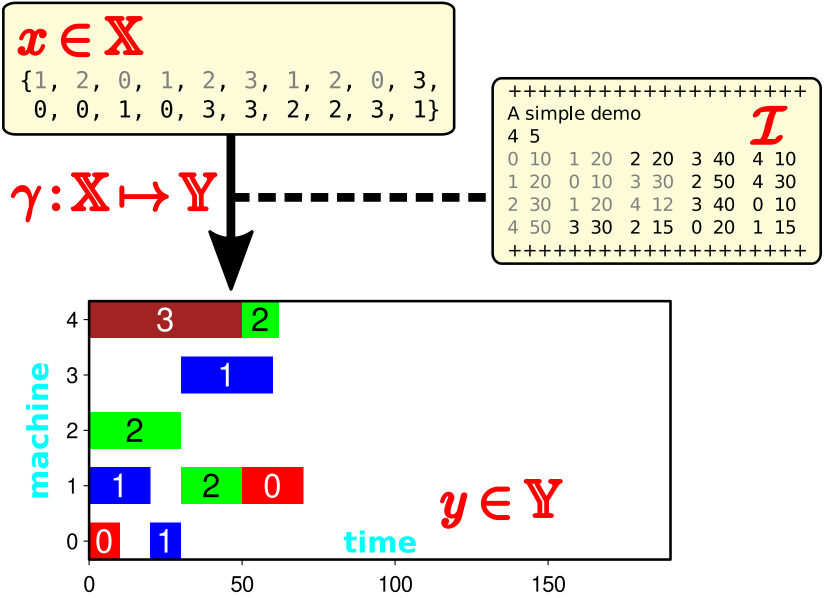
Demo Example for the Search Space



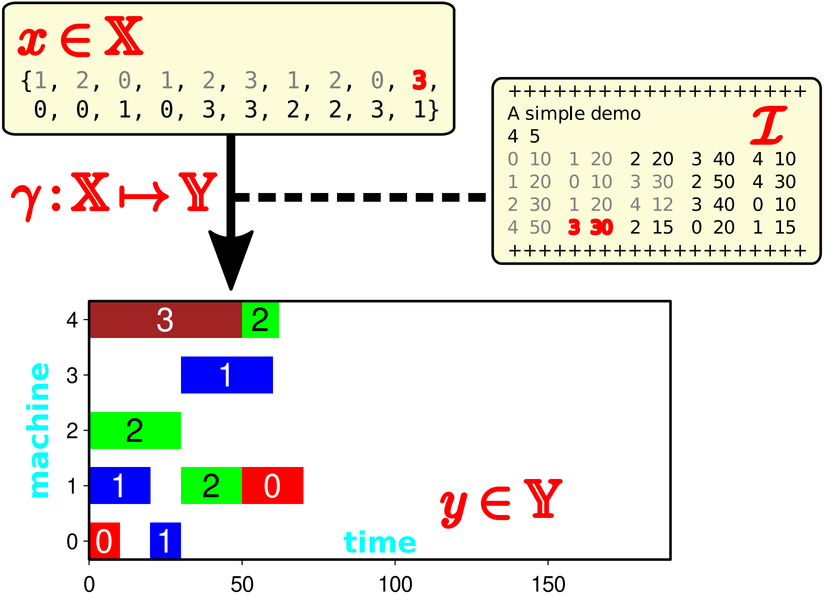
Demo Example for the Search Space



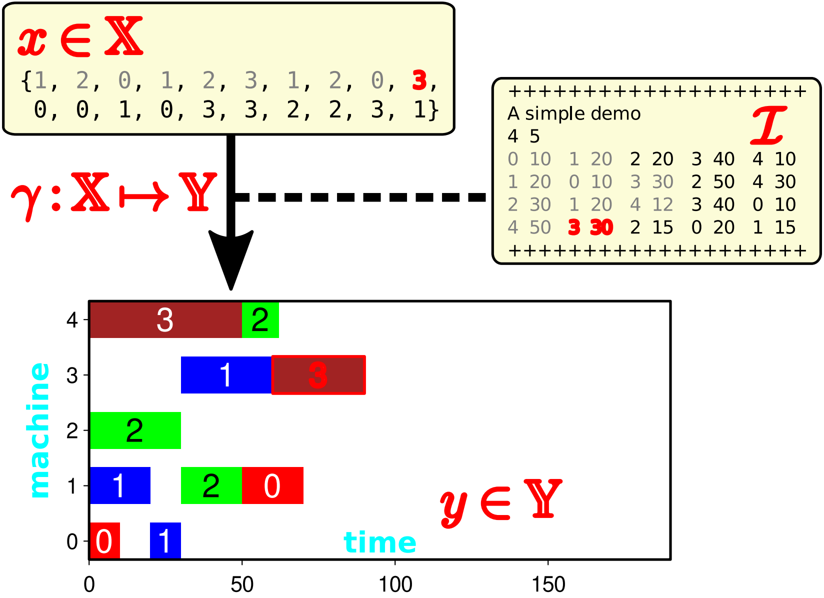
Demo Example for the Search Space



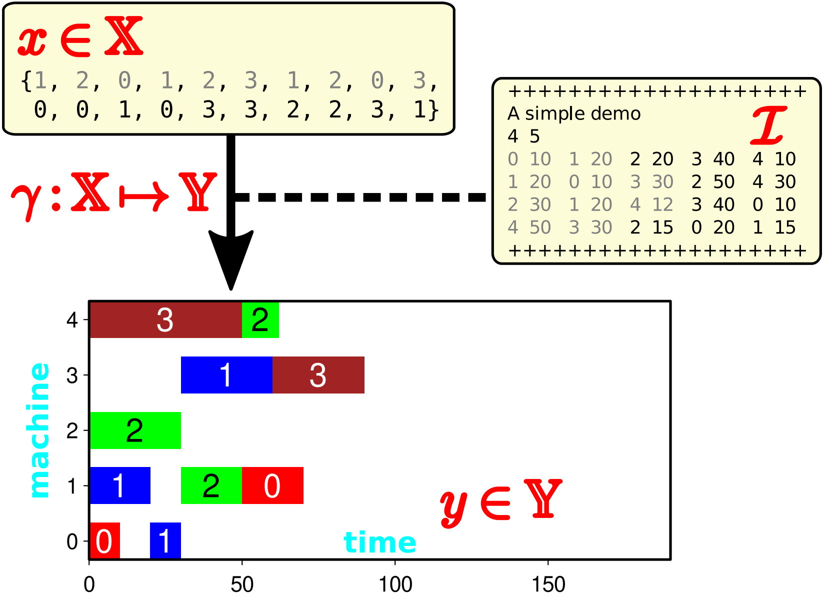
Demo Example for the Search Space



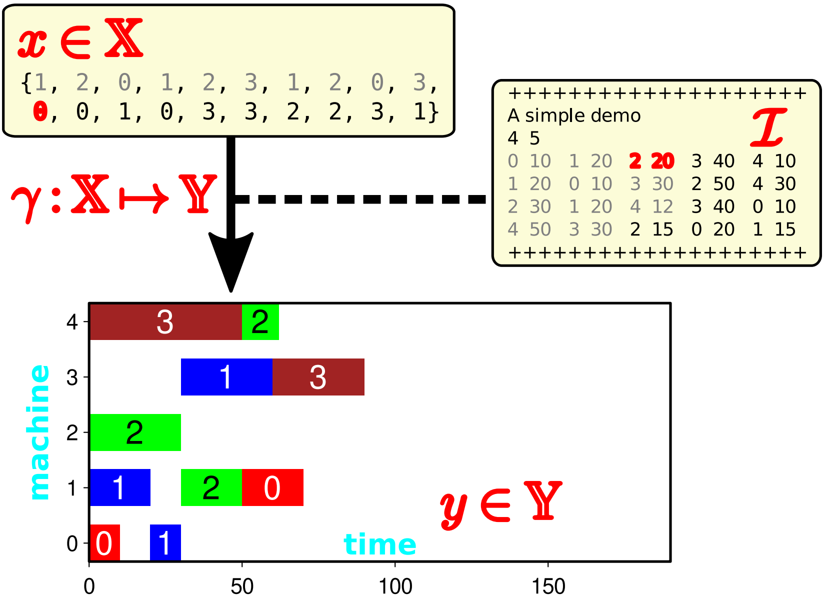
Demo Example for the Search Space



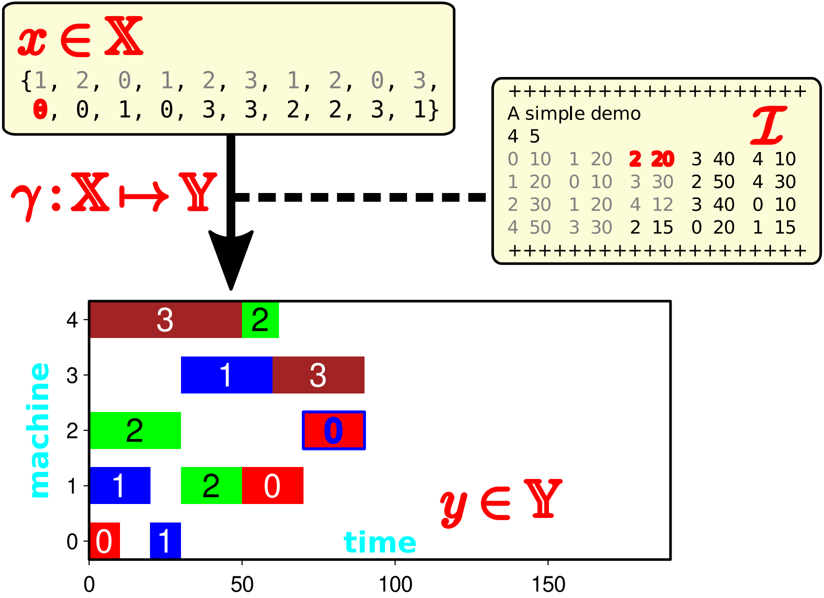
Demo Example for the Search Space



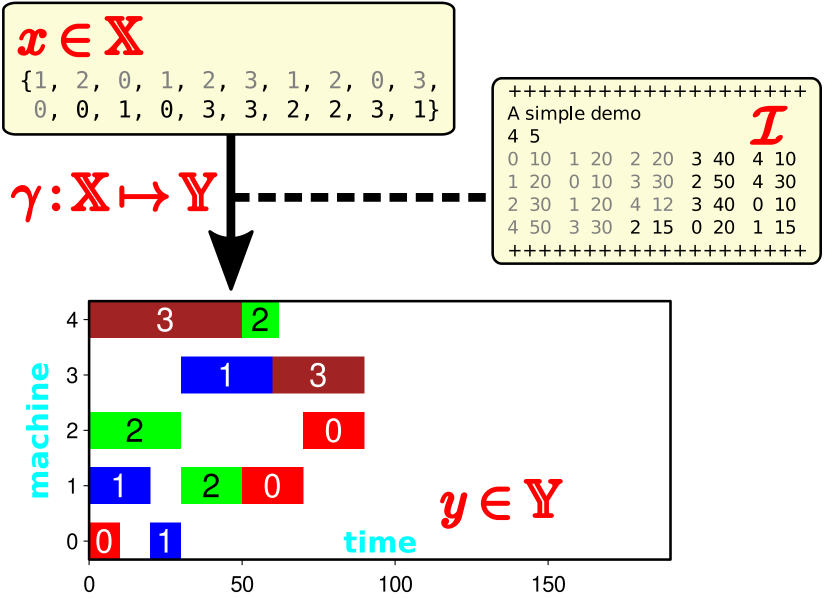
Demo Example for the Search Space



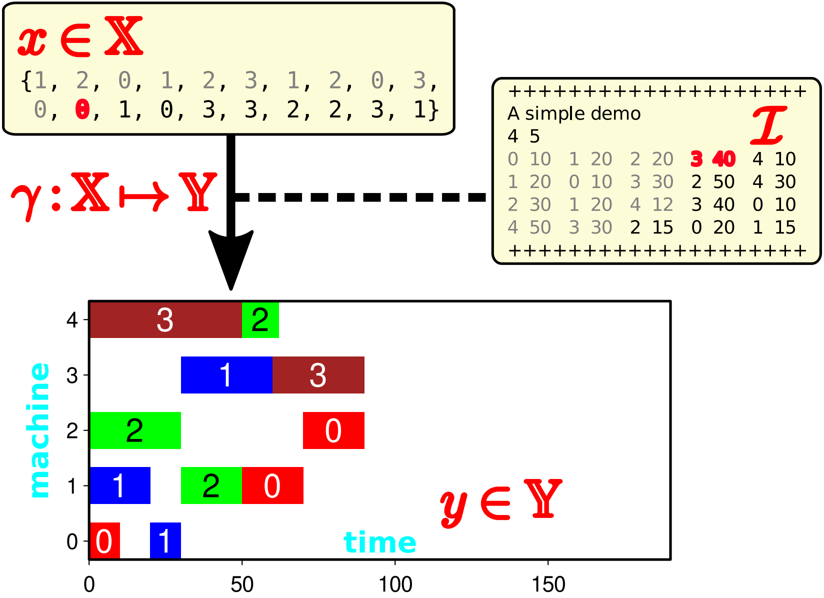
Demo Example for the Search Space



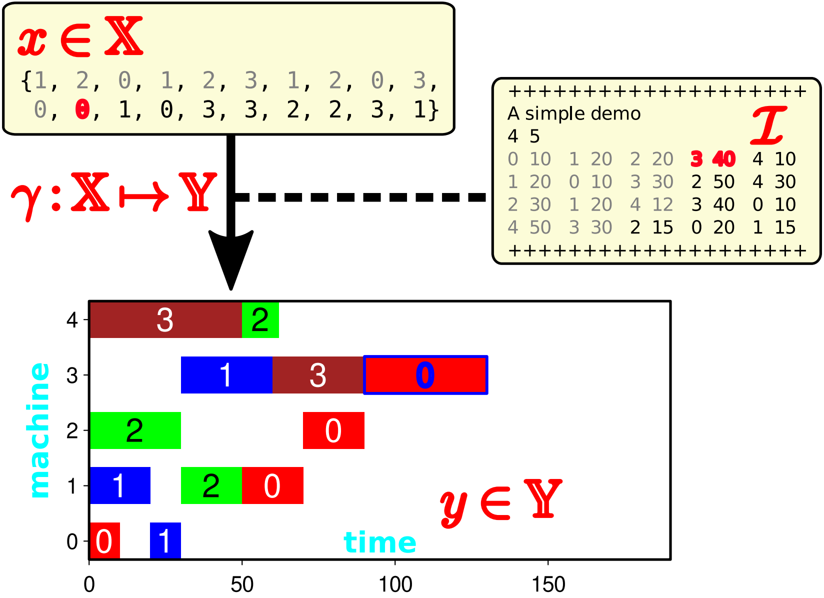
Demo Example for the Search Space



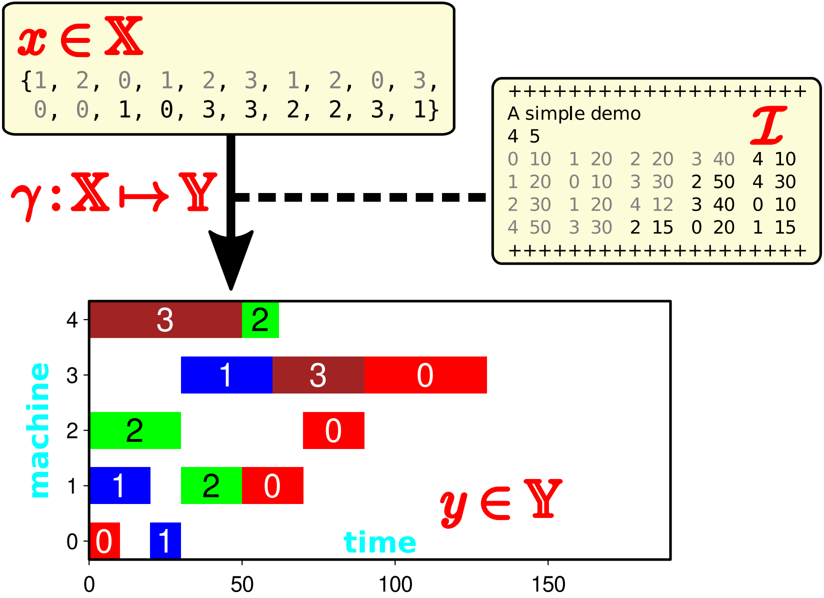
Demo Example for the Search Space



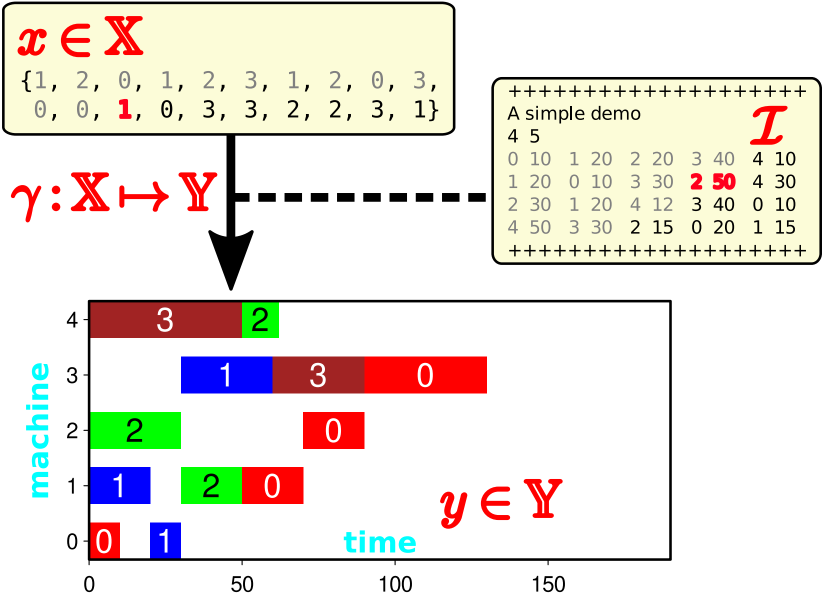
Demo Example for the Search Space



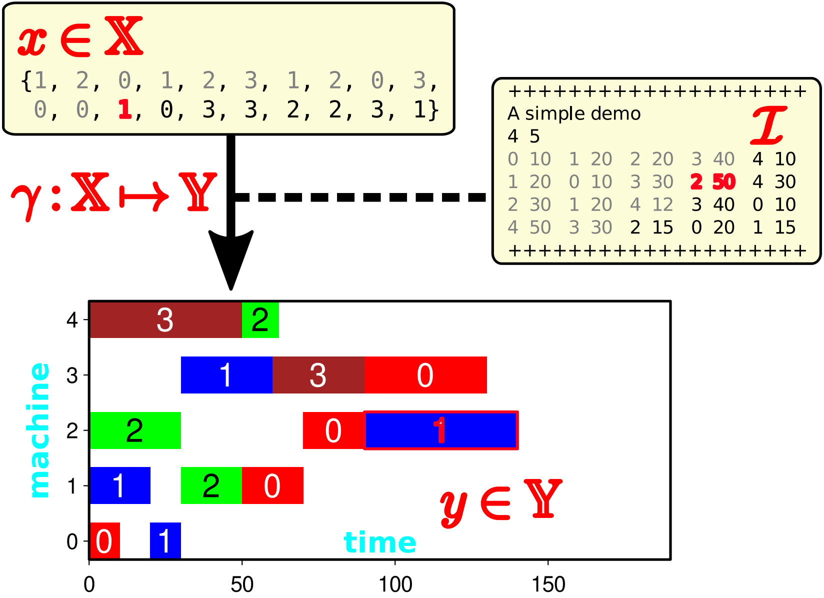
Demo Example for the Search Space



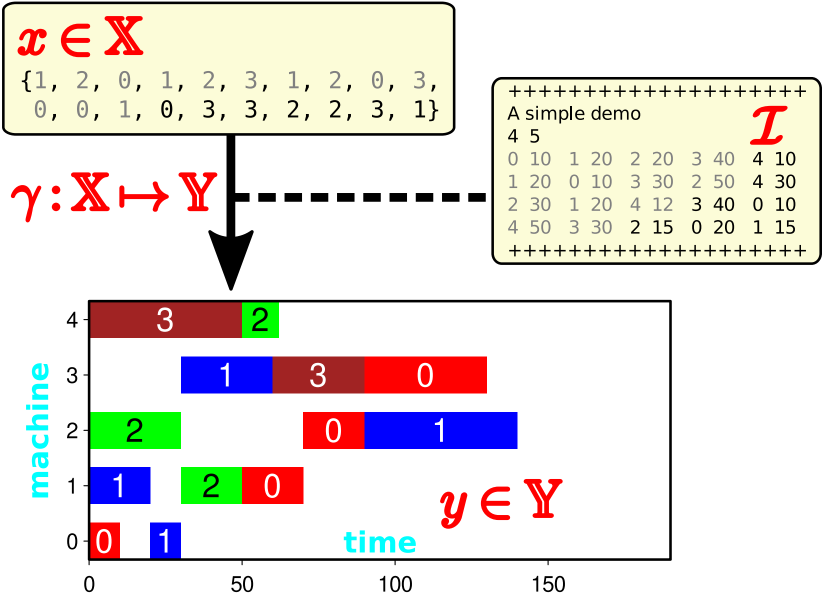
Demo Example for the Search Space



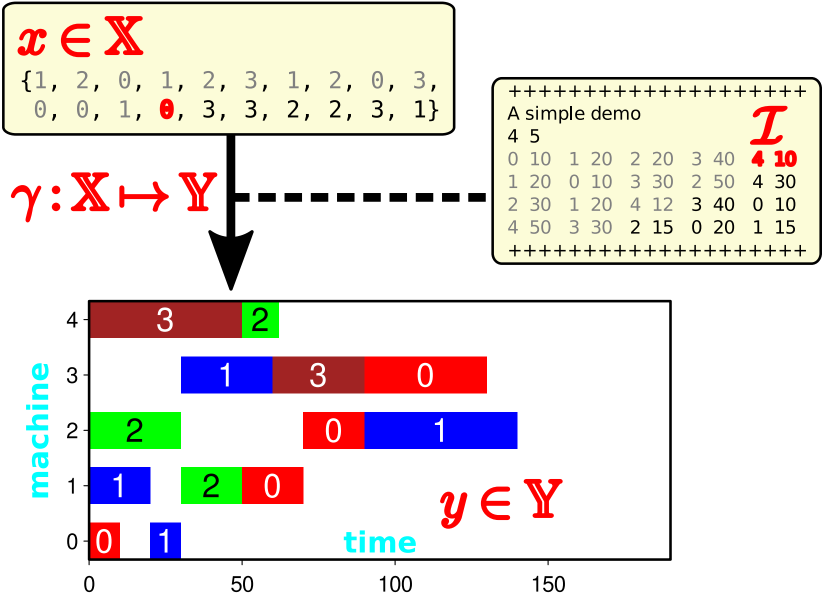
Demo Example for the Search Space



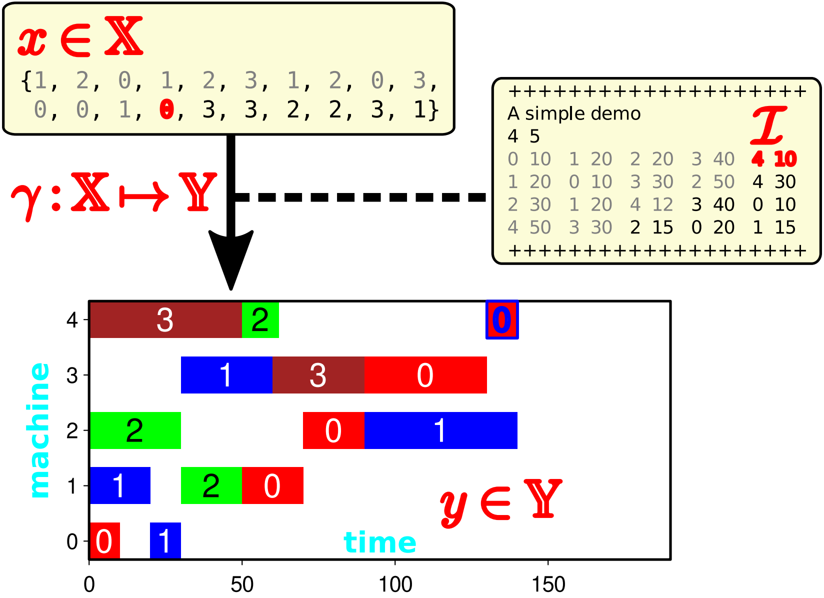
Demo Example for the Search Space



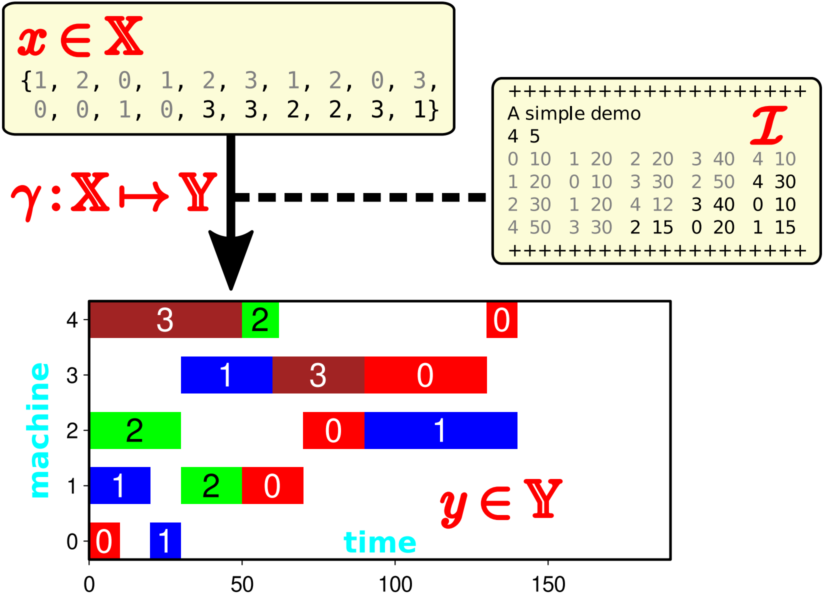
Demo Example for the Search Space



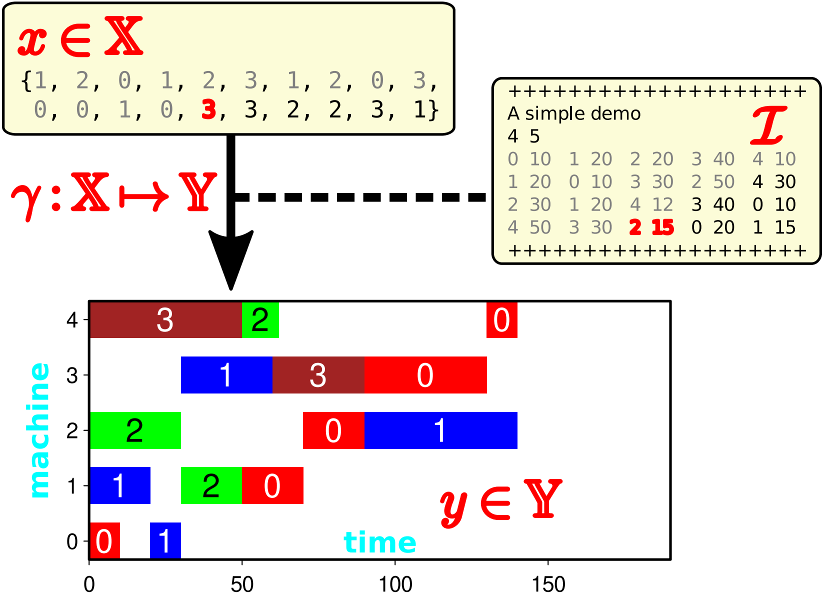
Demo Example for the Search Space



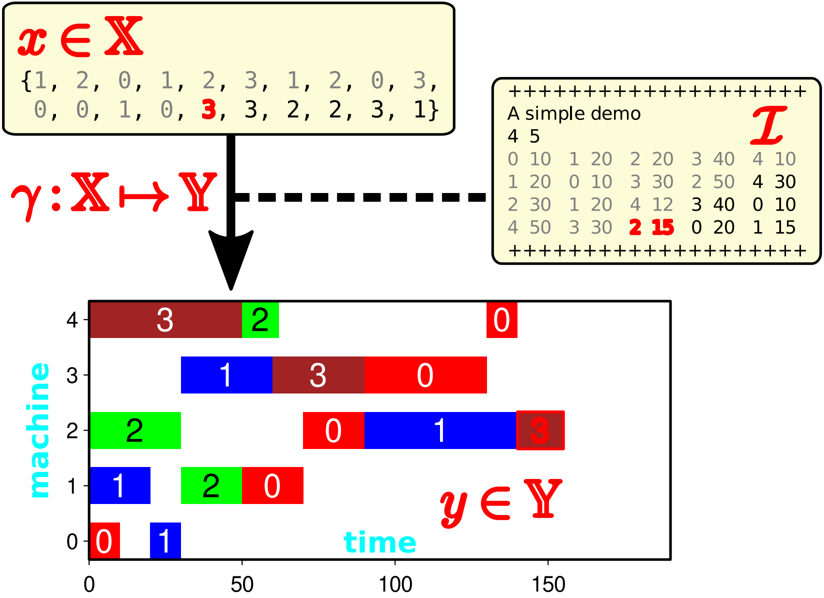
Demo Example for the Search Space



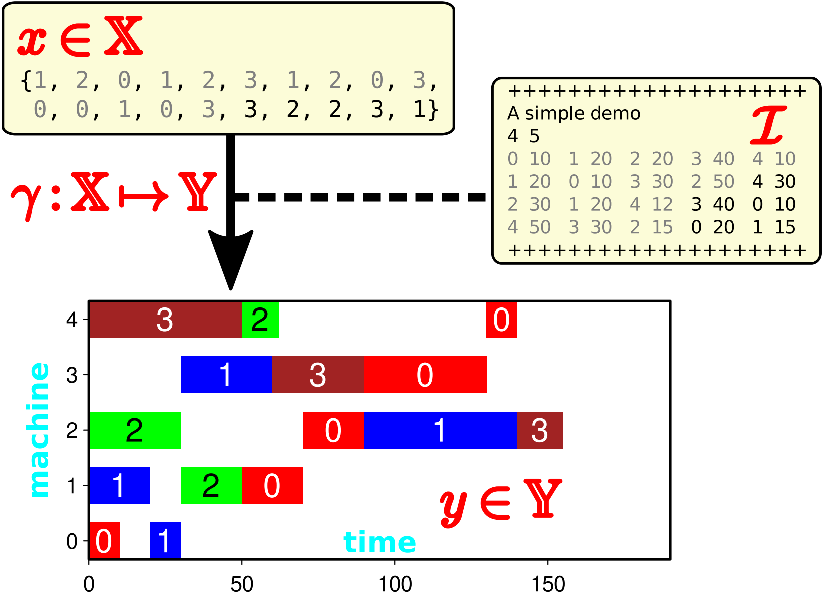
Demo Example for the Search Space



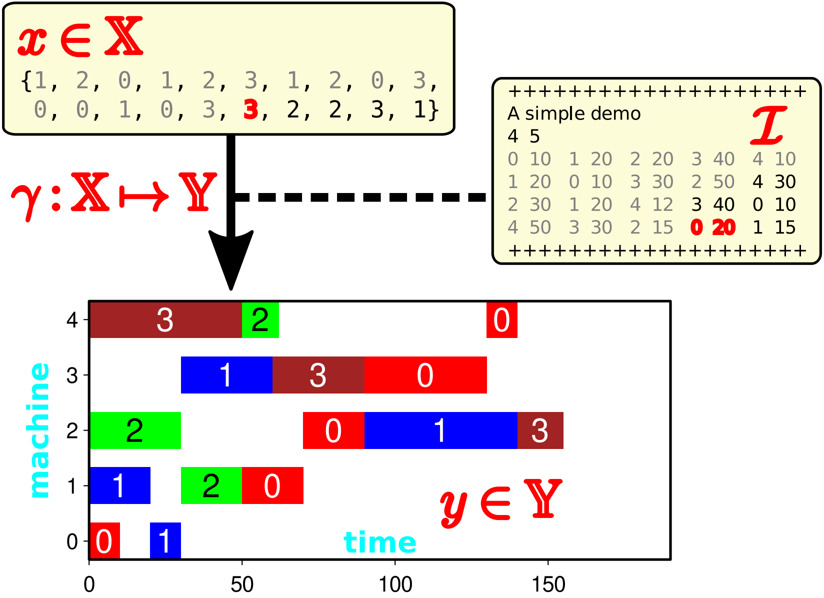
Demo Example for the Search Space



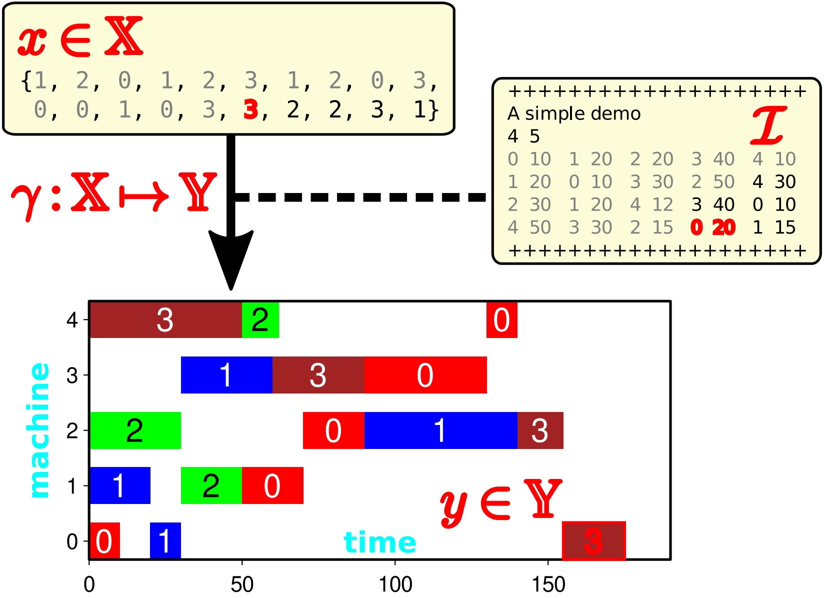
Demo Example for the Search Space



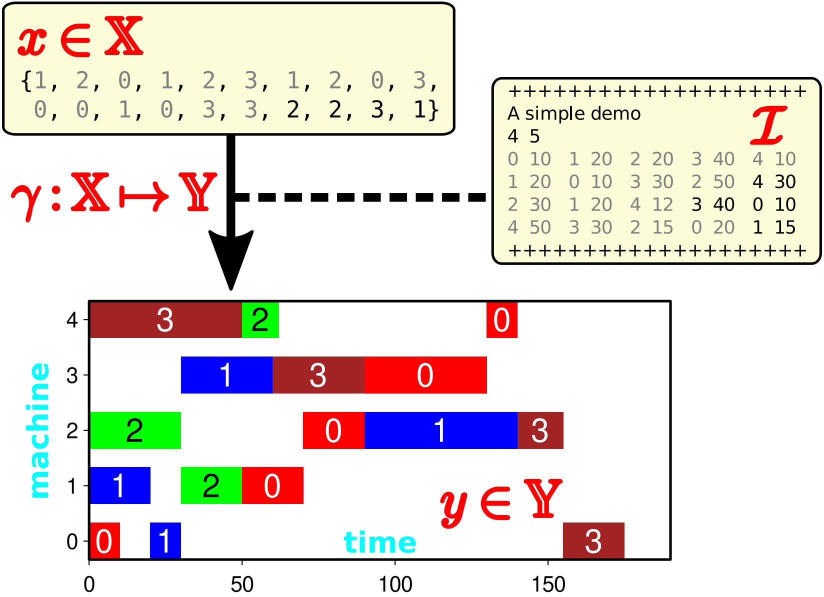
Demo Example for the Search Space



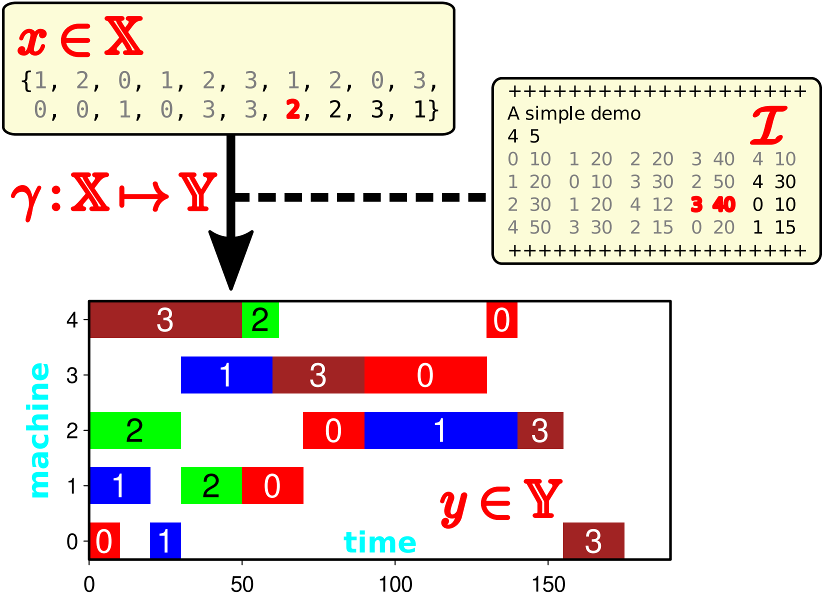
Demo Example for the Search Space



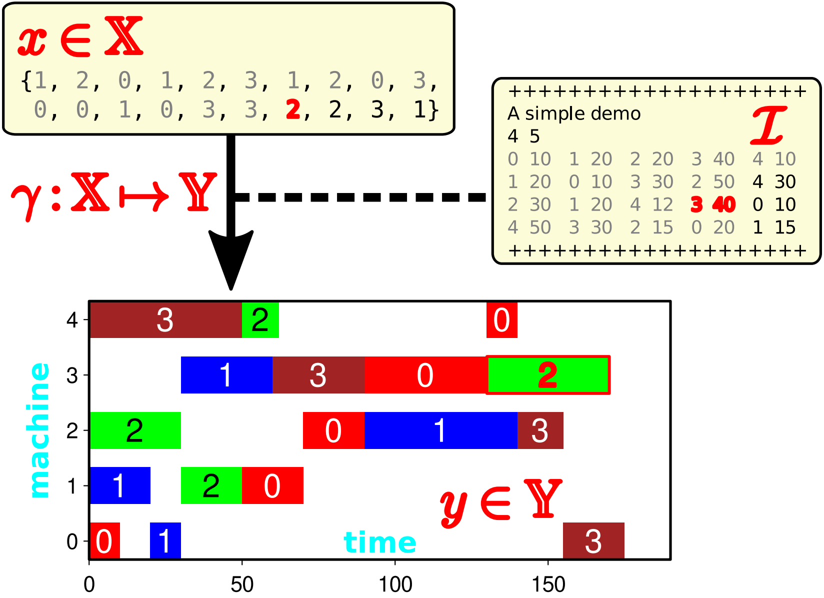
Demo Example for the Search Space



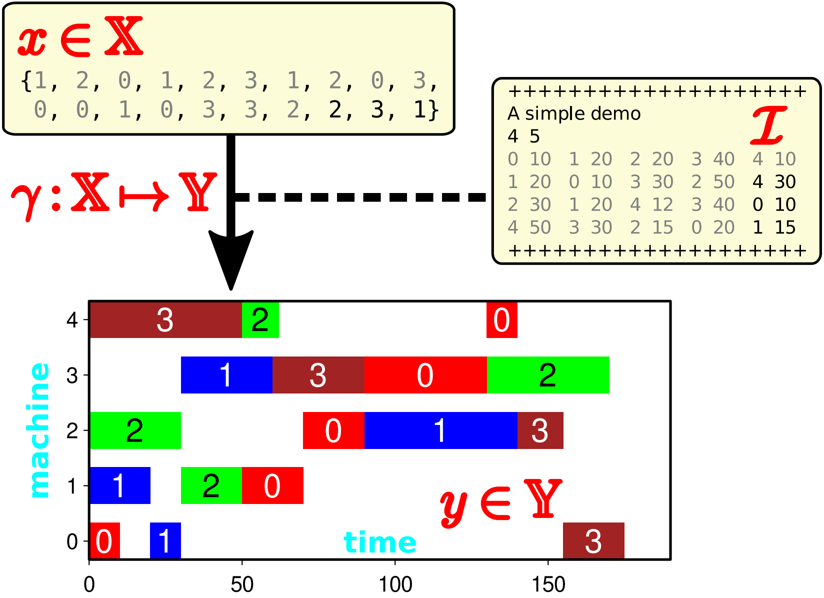
Demo Example for the Search Space



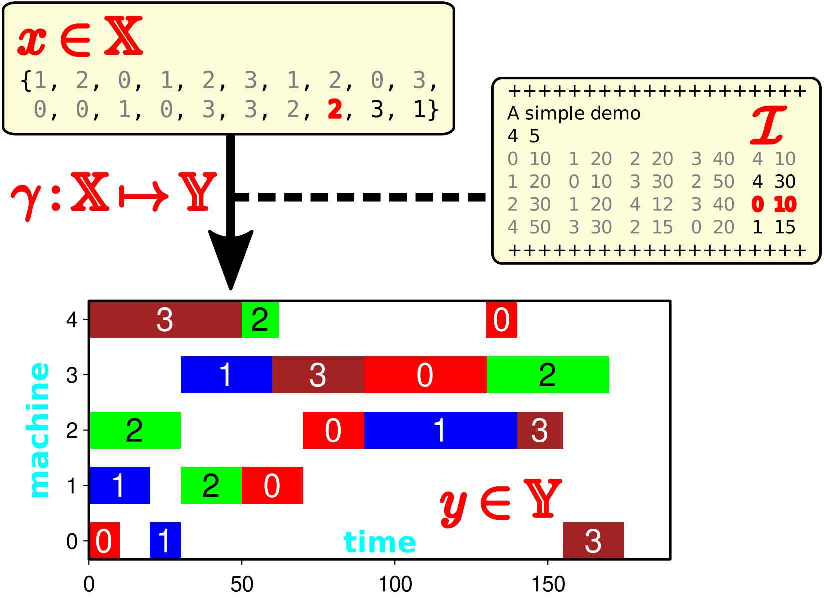
Demo Example for the Search Space



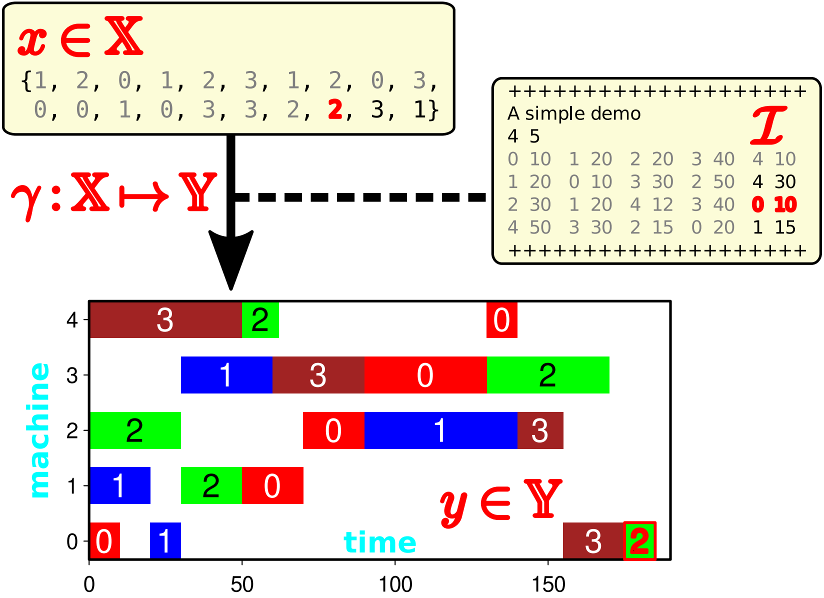
Demo Example for the Search Space



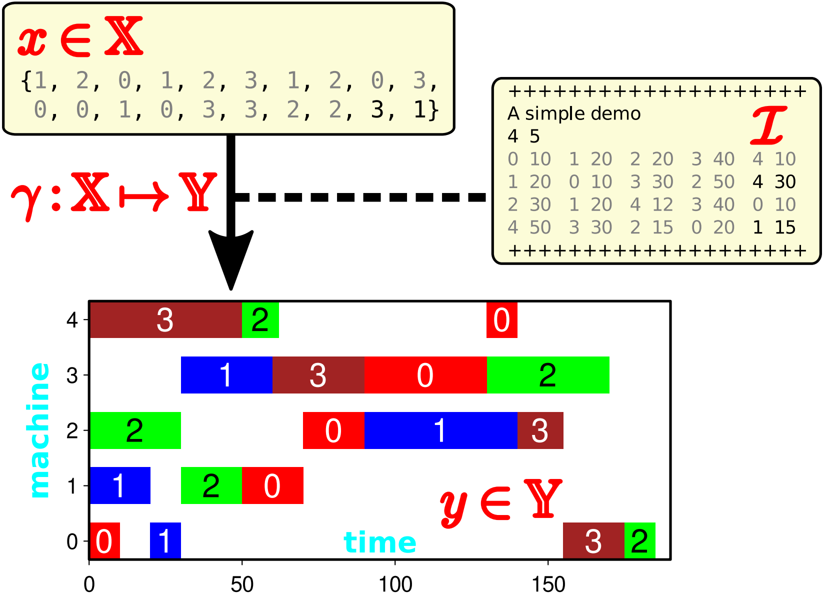
Demo Example for the Search Space



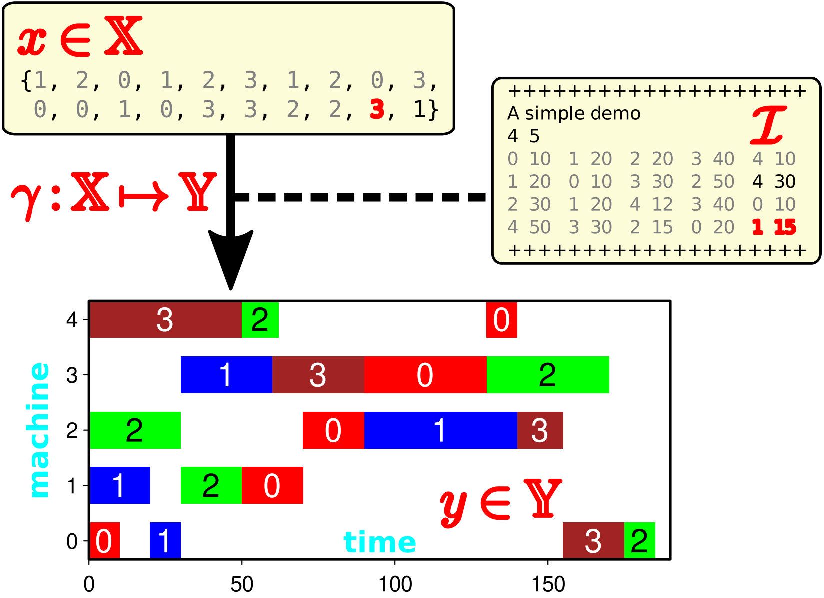
Demo Example for the Search Space



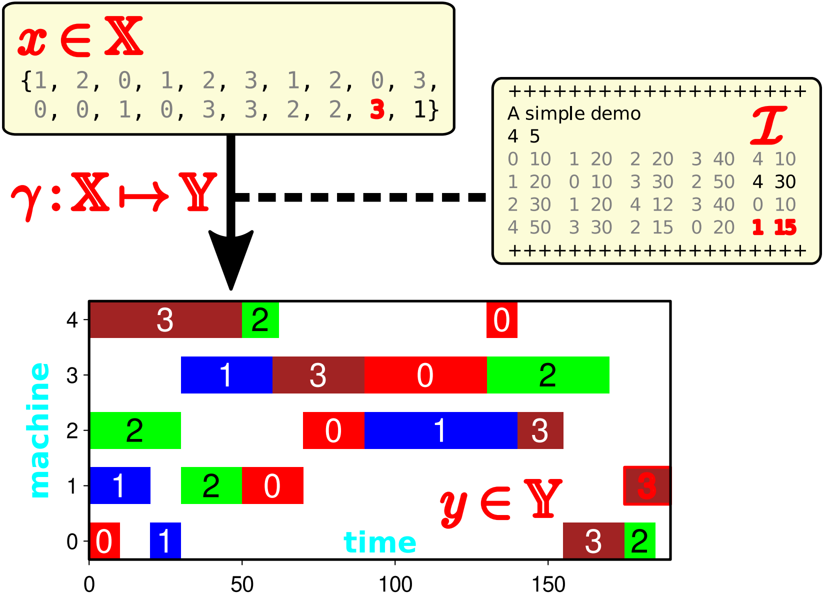
Demo Example for the Search Space



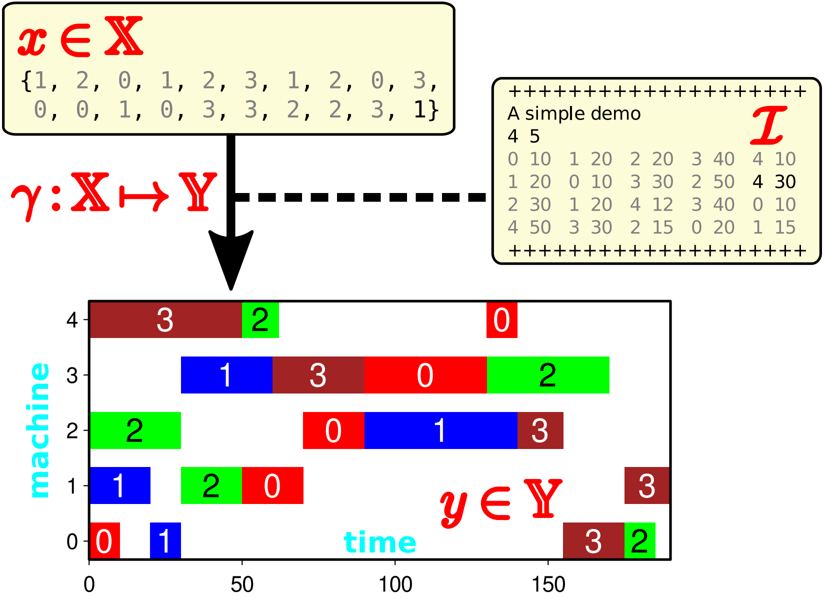
Demo Example for the Search Space



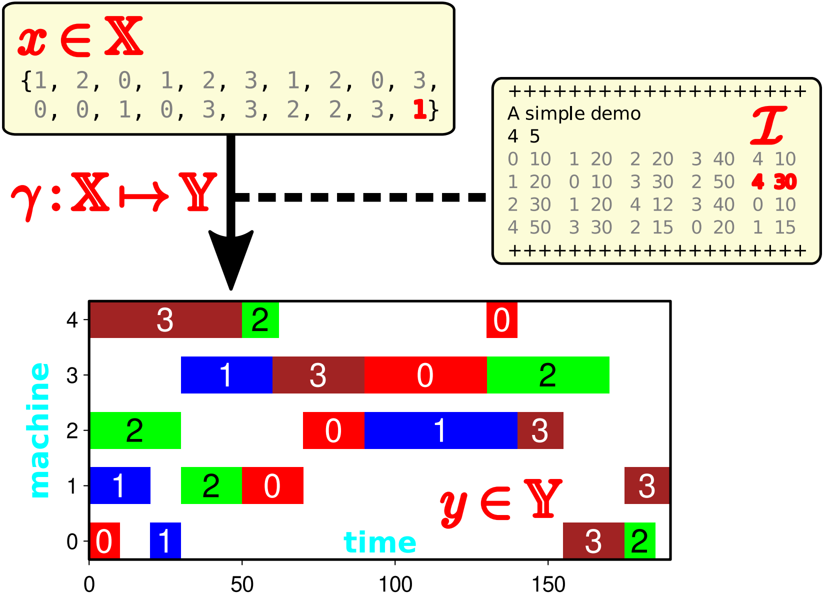
Demo Example for the Search Space



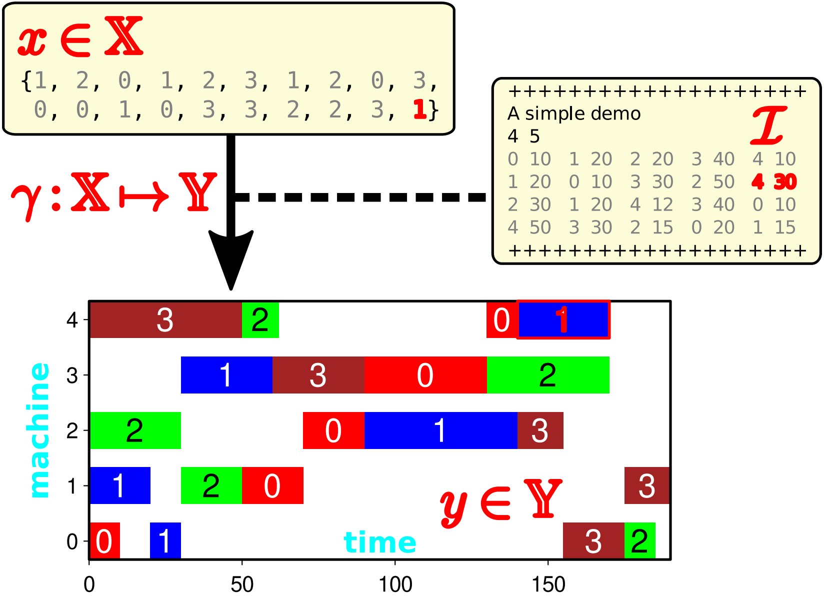
Demo Example for the Search Space



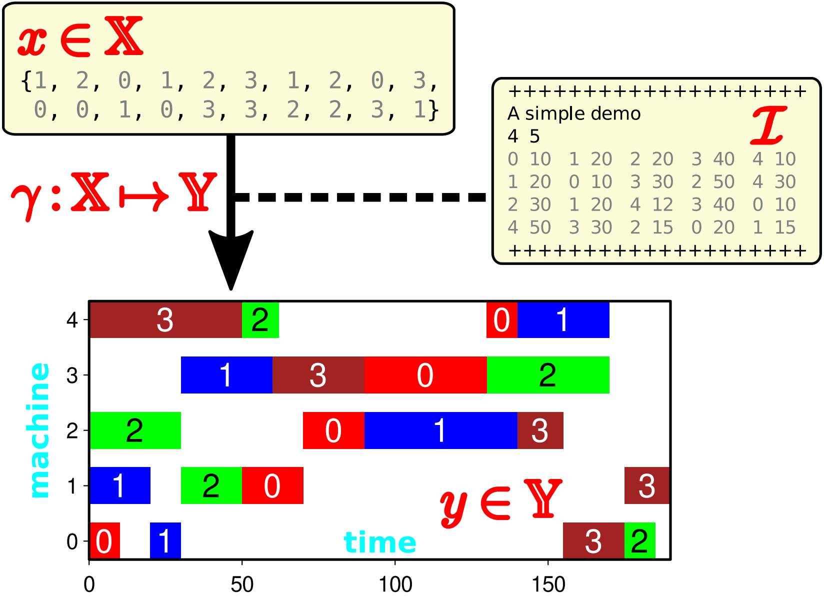
Demo Example for the Search Space



Demo Example for the Search Space



Demo Example for the Search Space



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 $m * n$ 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 $m * n$ contain each value in $1 \dots n$ 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 $m * n$ 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 γ.

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"
//
//
//
//
//

//
//
//
//

//
//
//

//
//
//
//
//
//
} // 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"
//
//
//
//
//

//
//
//
//

//
//
//

//
//
//
//
//
//
} // end abridged class
```

The JSSP Representation Mapping in 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) {
//
//
//
//

//
//
//
//

//
//
//

//
//
//
//
//
  } // end function
} // end abridged class
```

The JSSP Representation Mapping in 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

```
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

```
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
```

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
//
//

//
//
//

//
//
//
//
    } // 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
```

The JSSP Representation Mapping in 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

//
//
//

//
//
//
//
    } // end iteration over job IDs
  } // end function
} // end abridged class
```

The JSSP Representation Mapping in 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

```
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
```

The JSSP Representation Mapping in 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
      jobTime[nextJob] = machineTime[machine] = end;

//
//
//
//
    } // end iteration over job IDs
  } // end function
} // end abridged class
```

The JSSP Representation Mapping in 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
      jobTime[nextJob] = machineTime[machine] = end;

      int[] schedule = y.schedule[machine]; // get list of tasks for machine
//
//
//
    } // 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
      jobTime[nextJob] = machineTime[machine] = end;

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

The JSSP Representation Mapping in 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
      jobTime[nextJob] = machineTime[machine] = end;

      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
```

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
      jobTime[nextJob] = machineTime[machine] = end;

      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 $\mathbb{Y}$

- OK, we want to solve a JSSP instance

Number of Solutions: Size of $\mathbb{Y}$

- OK, we want to solve a JSSP instance
- How many possible candidate solutions are there?

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.

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.

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 $n! = 1 * 2 * 3 * 4 * 5 * \ldots * n$ possible ways to arrange the n jobs.

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 $n! = 1 * 2 * 3 * 4 * 5 * \ldots * n$ possible ways to arrange the n jobs.
- If there are 2 machines, this gives us $(n!) * (n!) = (n!)^2$ choices.

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 $n! = 1 * 2 * 3 * 4 * 5 * \ldots * n$ possible ways to arrange the n jobs.
- If there are 2 machines, this gives us $(n!) * (n!) = (n!)^2$ 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 $n! = 1 * 2 * 3 * 4 * 5 * \ldots * n$ possible ways to arrange the n jobs.
- If there are 2 machines, this gives us $(n!) * (n!) = (n!)^2$ choices.
- For m machines, we are at $(n!)^m$ possible solutions.

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 $n! = 1 * 2 * 3 * 4 * 5 * \ldots * n$ possible ways to arrange the n jobs.
- If there are 2 machines, this gives us $(n!) * (n!) = (n!)^2$ 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	$\min(\#\text{feasible})$	$\lvert\mathbb{Y}\rvert$
	2	2	3	4

Number of Solutions: Size of $\mathbb{Y}$

name	n	m	$\min(\#\text{feasible})$	$\lvert\mathbb{Y}\rvert$
	2	2	3	4
	2	3	4	8

Number of Solutions: Size of $\mathbb{Y}$

name	n	m	$\min(\#\text{feasible})$	$\lvert\mathbb{Y}\rvert$
	2	2	3	4
	2	3	4	8
	2	4	5	16

Number of Solutions: Size of $\mathbb{Y}$

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

Number of Solutions: Size of $\mathbb{Y}$

name	n	m	$\min(\#\text{feasible})$	$\lvert\mathbb{Y}\rvert$
	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	$\min(\#\text{feasible})$	$\lvert\mathbb{Y}\rvert$
	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		$\approx 1.462*10^{121}$
`abz7`	20	15		$\approx 6.193*10^{275}$
`yn4`	20	20		$\approx 5.278*10^{367}$
`swv15`	50	10		$\approx 6.772*10^{644}$

Size of Search Space $\mathbb{X}$

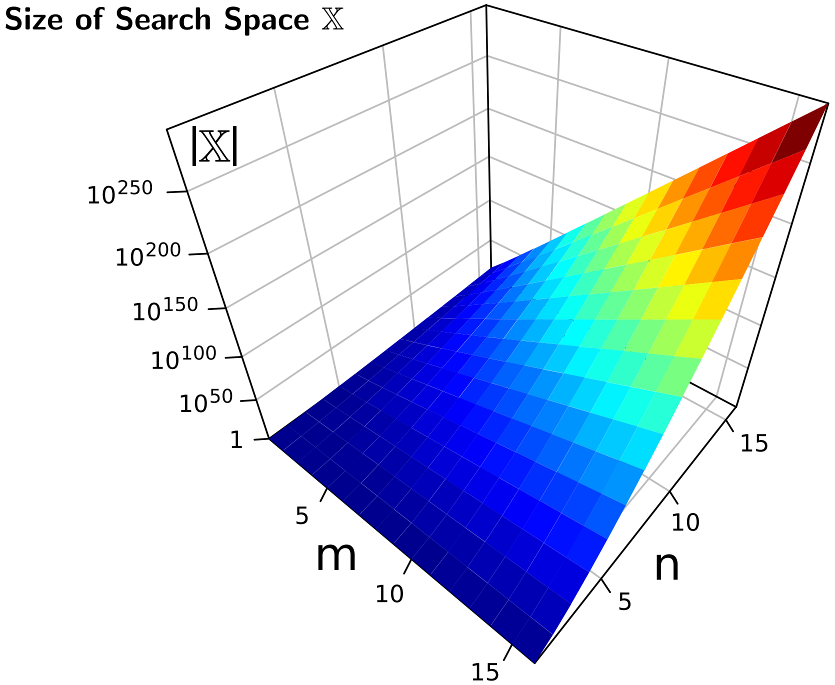
- Our search space $\mathbb{X}$ is not the same as the solution space $\mathbb{Y}$.

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?

Size of Search Space $\mathbb{X}$

name	n	m	$\lvert\mathbb{Y}\rvert$	$\lvert\mathbb{X}\rvert$
	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	$\approx 1.462{*}10^{121}$	$\approx 2.293{*}10^{164}$
`abz7`	20	15	$\approx 6.193{*}10^{275}$	$\approx 1.432{*}10^{372}$
`yn4`	20	20	$\approx 5.278{*}10^{367}$	$\approx 1.213{*}10^{501}$
`swv15`	50	10	$\approx 6.772{*}10^{644}$	$\approx 1.254{*}10^{806}$



Size of Search Space $\mathbb{X}$
$|\mathbb{X}|$
10^{250}
10^{200}
10^{150}
10^{100}
10^{50}
1
5
m
10
15
15
10
n
5

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 $\mathbb{X}$

- 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.
- $\mathbb{X}$ 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

Search OperatorAn k-ary search operator $\mathrm{searchOp} : \mathbb{X}^k \mapsto \mathbb{X}$ 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.

Search Operators

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

Definition

Search OperatorAn k-ary search operator $\mathrm{searchOp} : \mathbb{X}^k \mapsto \mathbb{X}$ 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

Search OperatorAn k-ary search operator $\mathrm{searchOp} : \mathbb{X}^k \mapsto \mathbb{X}$ 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

Search OperatorAn k-ary search operator $\mathrm{searchOp} : \mathbb{X}^k \mapsto \mathbb{X}$ 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.

```
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

Search OperatorAn k-ary search operator $\mathrm{searchOp} : \mathbb{X}^k \mapsto \mathbb{X}$ 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 $\mathbb{X}$.
  - unary operators $(k = 1)$ take one point from $\mathbb{X}$ as input and return another (similar) point.

Search Operators

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

Definition

Search OperatorAn k-ary search operator $\mathrm{searchOp} : \mathbb{X}^k \mapsto \mathbb{X}$ 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.

```
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

Search OperatorAn k-ary search operator $\mathrm{searchOp} : \mathbb{X}^k \mapsto \mathbb{X}$ 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 $\mathbb{X}$.
  - 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 OperatorAn k-ary search operator $\mathrm{searchOp} : \mathbb{X}^k \mapsto \mathbb{X}$ 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
package aitoa.structure;

public interface 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

Search OperatorAn k-ary search operator $\mathrm{searchOp} : \mathbb{X}^k \mapsto \mathbb{X}$ 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 $\mathbb{X}$.
  - 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 $\mathbb{X}$ which correspond to good solutions in $\mathbb{Y}$.
- 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 $\mathbb{X}$ which correspond to good solutions in $\mathbb{Y}$.
- 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 $\mathbb{X}$ which correspond to good solutions in $\mathbb{Y}$.
- 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.[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.[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?

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.

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 a 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.[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 a 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 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.

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 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 $\mathbb{Y}$ for the solutions, which can contain all the information that the end user needs and considers as a full solution to the problem!

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 $\mathbb{Y}$ 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?

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.

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 γ 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 γ 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 $\mathbb{Y}$ 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 γ from $\mathbb{X}$ to $\mathbb{Y}$!
  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

1. Thomas Weise. *An Introduction to Optimization Algorithms*. Institute of Applied Optimization (IAO) [应用优化研究所] of the School of Artificial Intelligence and Big Data [人工智能与大数据学院] of Hefei University [合肥学院], Hefei [合肥市], Anhui [安徽省], China [中国], 2018–2020. URL http://thomasweise.github.io/aitoa/.
2. Thomas Weise. *Global Optimization Algorithms – Theory and Application*. it-weise.de (self-published), Germany, 2009. URL http://www.it-weise.de/projects/book.pdf.
3. Fred Glover and Gary A. Kochenberger, editors. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science (ISOR)*. Springer Netherlands, Dordrecht, Netherlands, 2003. ISBN 0-306-48056-5. doi:10.1007/b101874.
4. Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, Berlin/Heidelberg, 2nd edition, 2004. ISBN 3-540-22494-7.
5. Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and Alexander Hendrik George Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5: 287–326, 1979. doi:10.1016/S0167-5060(08)70356-X.
6. Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In Stephen C. Graves, Alexander Hendrik George Rinnooy Kan, and Paul H. Zipkin, editors, *Handbook of Operations Research and Management Science*, volume IV: Production Planning and Inventory, chapter 9, pages 445–522. North-Holland Scientific Publishers Ltd., Amsterdam, The Netherlands, 1993. doi:10.1016/S0927-0507(05)80189-6.
7. Eugene Leighton Lawler. Recent results in the theory of machine scheduling. In AAchim Bachem, Bernhard Korte, and Martin Grötschel, editors, *Math Programming: The State of the Art*, chapter 8, pages 202–234. Springer-Verlag, Bonn/New York, 1982. ISBN 978-3-642-68876-8. doi:10.1007/978-3-642-68874-4_9.
8. Éric D. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research (EJOR)*, 64(2): 278–285, January 1993. doi:10.1016/0377-2217(93)90182-M.
9. Jacek Błażewicz, Wolfgang Domschke, and Erwin Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research (EJOR)*, 93:1–33, August 1996. doi:10.1016/0377-2217(95)00362-2. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.159.1650&type=pdf.
10. Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations. The IBM Research Symposia Series.*, pages 85–103. Springer, Boston, MA, USA, 1972. ISBN 978-1-4684-2003-6. doi:10.1007/978-1-4684-2001-2_9.
11. Stephen Arthur Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC'71), May 3–5, 1971, Shaker Heights, OH, USA*, pages 151–158, New York, NY, USA, 1971. ACM. doi:10.1145/800157.805047.

References II

12. John Edward Beasley. Or-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society (JORS)*, 41:1069–1072, November 1990. doi:10.1057/jors.1990.166.
13. Jelke Jeroen van Hoorn. Job shop instances and solutions, 2015. URL `http://jobshop.jjvh.nl`.
14. Jelke Jeroen van Hoorn. The current state of bounds on benchmark instances of the job-shop scheduling problem. *Journal of Scheduling*, 21:127–128, feb 2018. doi:10.1007/s10951-017-0547-8.
15. Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988. doi:10.1287/mnsc.34.3.391.
16. Stephen R. Lawrence. *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*. PhD thesis, Graduate School of Industrial Administration (GSIA), Carnegie-Mellon University, Pittsburgh, PA, USA, 1984.
17. Robert H. Storer, S. David Wu, and Renzo Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):1495–1509, 1992. doi:10.1287/mnsc.38.10.1495.
18. Takeshi Yamada and Ryohei Nakano. A genetic algorithm applicable to large-scale job-shop instances. In Reinhard Männer and Bernard Manderick, editors, *Proceedings of Parallel Problem Solving from Nature 2 (PPSN II), September 28–30, 1992, Brussels, Belgium*, pages 281–290, Amsterdam, The Netherlands, 1992. Elsevier.
19. James M. Wilson. Gantt charts: A centenary appreciation. *European Journal of Operational Research (EJOR)*, 149: 430–437, September 2003. doi:10.1016/S0377-2217(02)00769-5. URL `http://www-public.imtbs-tsp.eu/~gibson/Teaching/Teaching-ReadingMaterial/Wilson03.pdf`.
20. Robert Klein. *Scheduling of Resource-Constrained Projects*, volume 10 of *Operations Research/Computer Science Interfaces Series*. Springer US, New York, NY, USA, 2000. ISBN 978-0-7923-8637-7. doi:10.1007/978-1-4615-4629-0.
21. Mitsuo Gen, Yasuhiro Tsujimura, and Erika Kubota. Solving job-shop scheduling problems by genetic algorithm. In *Humans, Information and Technology: Proceedings of the 1994 IEEE International Conference on Systems, Man and Cybernetics, October 2–5, 1994, San Antonio, TX, USA*, volume 2. IEEE, 1994. ISBN 0-7803-2129-4. doi:10.1109/ICSMC.1994.400072. URL `http://read.pudn.com/downloads151/doc/658565/00400072.pdf`.
22. Christian Bierwirth. A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations-Research-Spektrum (OR Spectrum)*, 17:87–92, June 1995. doi:10.1007/BF01719250. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.7392&type=pdf`.

References III

23. Christian Bierwirth, Dirk C. Mattfeld, and Herbert Kopfer. On permutation representations for scheduling problems. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (PPSN IV), September 22–24, 1996, Berlin, Germany*, volume 1141/1996 of *Lecture Notes in Computer Science (LNCS)*, pages 310–318, Berlin, Germany, 1996. Springer-Verlag GmbH. ISBN 3-540-61723-X. doi:10.1007/3-540-61723-X_995. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.8377&type=pdf.
24. Guoyong Shi, Hitoshi Iima, and Nobuo Sannomiya. New encoding scheme for solving job shop problems by genetic algorithm. In *Proceedings of the 35th IEEE Conference on Decision and Control (CDC'96), December 11–13, 1996, Kobe, Japan*, volume 4, pages 4395–4400. IEEE, 1997. ISBN 0-7803-3590-2. doi:10.1109/CDC.1996.577484.
25. Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. A review of machine scheduling: Complexity, algorithms and approximability. In Ding-Zhu Du and Panos M. Pardalos, editors, *Handbook of Combinatorial Optimization*, pages 1493–1641. Springer-Verlag US, Boston, MA, USA, 1998. ISBN 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_25. also pages 21–169 in volume 3/3 by Kluwer Academic Publishers.
26. David Paul Williamson, Leslie A. Hall, J. A. Hoogeveen, Cor A. J. Hurkens, Jan Karel Lenstra, Sergey Vasil'evich Sevast'janov, and David B. Shmoys. Short shop schedules. *Operations Research*, 45(2):288–294, March–April 1997. doi:10.1287/opre.45.2.288.
27. Klaus Jansen, Monaldo Mastrolilli, and Roberto Solis-Oba. Approximation schemes for job shop scheduling problems with controllable processing times. *European Journal of Operational Research (EJOR)*, 167(2):297–319, December 2005. doi:10.1016/j.ejor.2004.03.025. URL http://people.idsia.ch/~monaldo/papers/EJOR-varJsp-05.pdf.
28. Monaldo Mastrolilli and Ola Svensson. Hardness of approximating flow and job shop scheduling problems. *Journal of the ACM (JACM)*, 58(5):20:1–20:32, October 2011. doi:10.1145/2027216.2027218. URL http://theory.epfl.ch/osven/Ola%20Svensson_publications/JACM11.pdf.