



吴贤铭智能工程学院  
SHIEN-MING WU SCHOOL OF  
INTELLIGENT ENGINEERING



## An Automated Assembly Line for the Luban Lock

082100511, Smart Factory

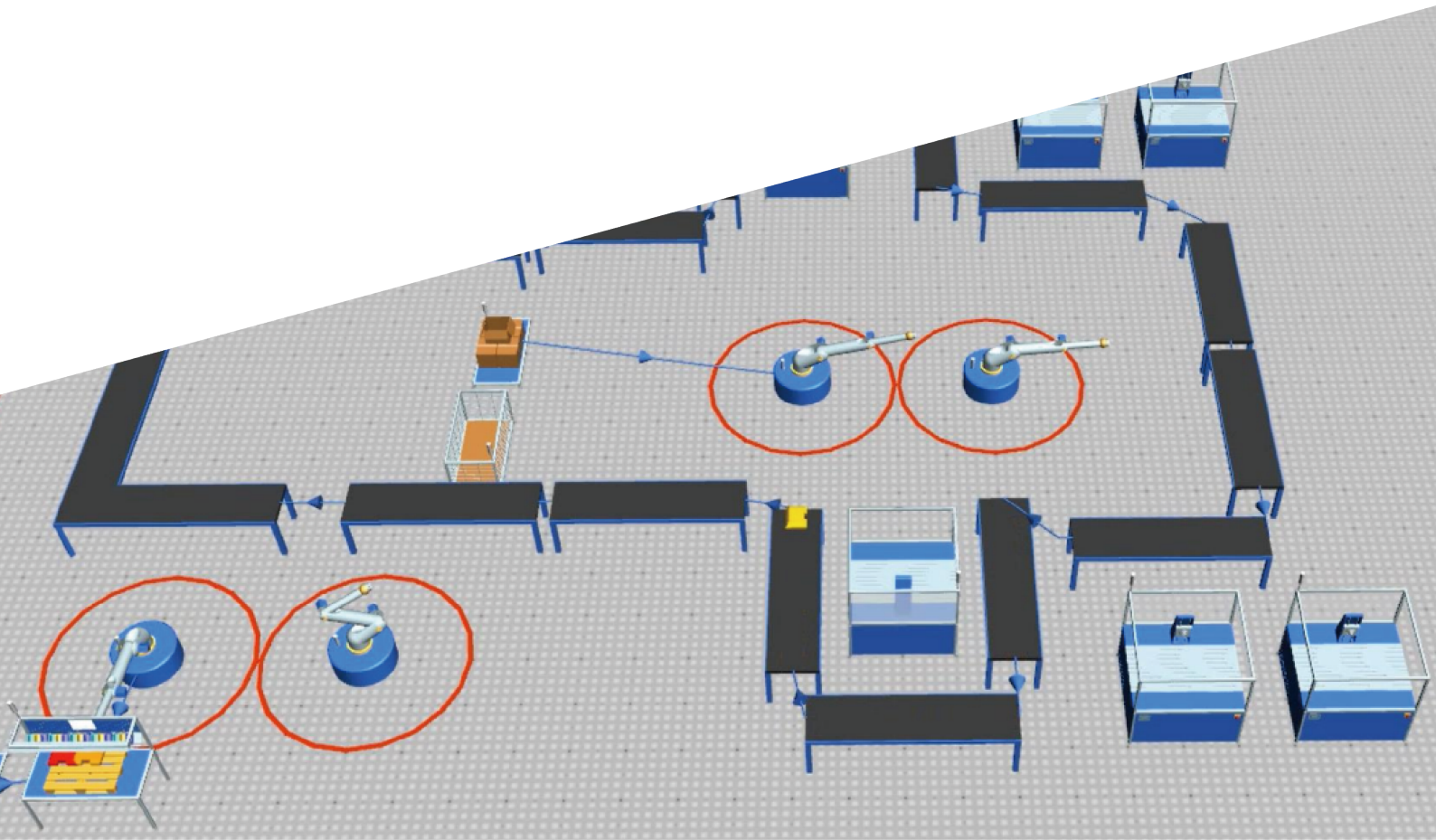
### Group SharpShooters

| Full Name    | Student ID   |
|--------------|--------------|
| Xinlei Zhang | 202030101256 |
| Ze'an He     | 202030020342 |
| Yile Shen    | 202030020205 |
| Juncong Lan  | 202030020113 |
| Yuli Yang    | 202030322149 |
| Jinan Guo    | 201930031052 |

[Github Page](#)

Tutor: Dr.Gang Chen

GZIC, June 30, 2023



# Contents

|           |  |           |
|-----------|--|-----------|
| <b>1</b>  | <b>Introduction</b>  | <b>1</b>  |
| 1.1       | Design Background . . . . .  | 1         |
| 1.2       | Project Description . . . . .  | 1         |
| 1.3       | Project Assignment . . . . .   | 2         |
| <b>2</b>  | <b>Problem Formulation in Mathematics</b>  | <b>2</b>  |
| 2.1       | Mathematical Modeling: Traditional Job Shop Scheduling Problem (JSP) and Flexible Job Shop Scheduling Problem (FJSP) . . . . . | 2         |
| 2.1.1     | Traditional Job Shop Scheduling Problem (JSP) . . . . .  | 2         |
| 2.1.2     | Flexible Job Shop Scheduling Problem (FJSP) . . . . .  | 3         |
| 2.1.3     | Comparison . . . . .   | 4         |
| 2.2       | Assumptions and Clarifications . . . . .   | 5         |
| 2.3       | Mathematical Model . . . . .   | 5         |
| <b>3</b>  | <b>Optimization Method</b>   | <b>7</b>  |
| 3.1       | Optimization Methods for Flexible Job Shop Scheduling Problem . . . . .  | 7         |
| 3.1.1     | Combinatorial Optimization and FJSP . . . . .  | 7         |
| 3.1.2     | Common Algorithms for Combinatorial Optimization and FJSP . . . . .  | 7         |
| 3.2       | MATLAB Optimization Toolbox . . . . .  | 8         |
| 3.3       | MATLAB code . . . . .  | 8         |
| <b>4</b>  | <b>Plant Simulation</b>  | <b>10</b> |
| 4.1       | Software Description . . . . .   | 10        |
| 4.2       | Components . . . . .   | 11        |
| 4.2.1     | Material Flow . . . . .  | 11        |
| 4.2.2     | Information Flow . . . . .   | 13        |
| 4.3       | Modeling . . . . .   | 14        |
| 4.3.1     | Part . . . . .   | 14        |
| 4.3.2     | Conveyor . . . . .   | 15        |
| 4.3.3     | Pick and place robot . . . . .   | 16        |
| 4.3.4     | Processing robot . . . . .   | 16        |
| 4.3.5     | Assembly robot . . . . .   | 17        |
| 4.3.6     | Complete process . . . . .   | 17        |
| <b>5</b>  | <b>Results</b>   | <b>18</b> |
| 5.1       | Plant setting . . . . .  | 18        |
| 5.2       | Simulation results . . . . .   | 19        |
| <b>6</b>  | <b>Conclusion</b>  | <b>20</b> |
| <b>7</b>  | <b>Acknowledgement</b>   | <b>20</b> |
| <b>8</b>  | <b>References</b>  | <b>20</b> |
| <b>9</b>  | <b>Appendix I: Operation Space</b>   | <b>21</b> |
| <b>10</b> | <b>Appendix II: MATLAB Code</b>  | <b>21</b> |

# 1 | Introduction

## 1.1 | Design Background

The Flexible Job Shop Problem (FJSP) is a complex scheduling problem that arises in manufacturing and production environments. It is an extension of the classical Job Shop Problem (JSP) and addresses the need for flexibility in scheduling operations. In a Job Shop, a set of jobs with specific operations needs to be processed on a set of machines. Each job consists of multiple operations that must be performed in a specific order, and each operation requires a certain amount of time to be processed on a particular machine. The objective is to determine the sequence of operations for each job and assign them to machines in such a way that the overall makespan (i.e., the total time required to complete all jobs) is minimized. The Flexible Job Shop Problem introduces the concept of machine flexibility, where each operation can be processed on one of several alternative machines. This allows for more options in scheduling and can be useful when certain machines are busy or unavailable. The FJSP takes into account the availability of machines, the sequence-dependent setup times between operations, and the objective of minimizing the makespan.

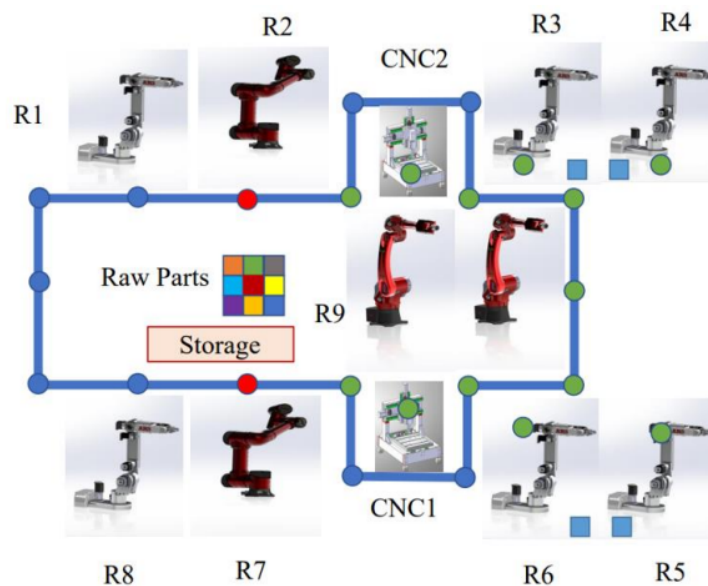
The FJSP is known to be a highly combinatorial and NP-hard problem, meaning that finding the optimal solution becomes exponentially more difficult as the problem size increases. Many heuristic and meta-heuristic algorithms have been developed to tackle the FJSP, including genetic algorithms, tabu search, simulated annealing, and particle swarm optimization, among others.

Luban Lock is a traditional Chinese puzzle toy. The Luban Lock consists of a set of interlocking wooden pieces that form a three-dimensional structure. The goal of the puzzle is to disassemble the pieces by manipulating them in a specific way and then reassemble them back into the original structure. The Luban Lock typically consists of several wooden pieces, each with notches, slots, or other interlocking mechanisms. The challenge lies in figuring out the correct sequence of movements and rotations required to unlock and separate the pieces. The solution often involves a series of precise maneuvers and spatial reasoning. The Luban Lock is not only a recreational puzzle but also serves as a metaphor for problem-solving, patience, and craftsmanship in Chinese culture.

Solving the Luban Lock can be an enjoyable and challenging activity that promotes logical thinking, dexterity, and perseverance. It is often seen as an engaging brain teaser or a decorative item that showcases the ingenuity of traditional Chinese craftsmanship.

## 1.2 | Project Description

For this project, the main task is to assemble Luban lock with an automatic assembly line. The Luban lock has nine parts, which need to be assembled in sequence. There are a total of nine robots and two CNC machines on the assembly line. Our target is to reasonably allocate the machines on the assembly line, in order to assemble ten Luban lock in the shortest time.



The functions of each robot on the assembly line are as follows:

- R9: Grab and place items to designated locations.
- R2,R7: Grab and place items to designated locations.
- R3 to R6: Process all parts of Luban lock.
- R1,R8: Assemble the processed parts into Luban lock.
- CNC1,CNC2: Label the assembled Luban lock.

The specific assembly process of a single Luban lock is as follows: R9 puts the parts into one of R3 to R6 for processing, and R9 places the processed parts on the conveyor belt after processing. When the parts are transferred to the designated position, R2 or R7 will place the processed parts on the assembly table of R1 or R8 . After all nine parts are placed, R1 or R8 will assemble the parts. After assembling, R2 or R7 will place the assembled Luban lock on the conveyor belt. When the Luban lock is transferred to the designated position, R9 will place the Luban lock on the workbench of CNC , and CNC will label the Luban lock. After the Luban lock is labeled, R9 will put the Luban lock into the storage.

### 1.3 | Project Assignment

Scheduling Algorithm: XinLei Zhang, YuYang Xie, YuLi Yang;  
 Plant Simulation : JunCong Lan, Ze'An He;  
 Project Report : JiNan Guo, YiLe Shen.

## 2 | Problem Formulation in Mathematics

### 2.1 | Mathematical Modeling: Traditional Job Shop Scheduling Problem (JSP) and Flexible Job Shop Scheduling Problem (FJSP)

The traditional Job Shop Scheduling Problem (JSP) and the Flexible Job Shop Scheduling Problem (FJSP) are two well-known optimization problems in production scheduling. In this subsection, we will describe the mathematical models for each problem, describing their process variables, objective function and constraints in an abstract level.

#### 2.1.1 | Traditional Job Shop Scheduling Problem (JSP)

The JSP involves a set of jobs, each with a predefined sequence of operations that need to be performed. Additionally, there is a set of machines capable of executing specific operations. The objective is to minimize the makespan, which represents the total time required to complete all jobs in the system.

Table 2.1 presents the variables used in the mathematical model for the JSP:

**Objective Function:**

Minimize: Makespan

Here, the makespan can be expressed as the completion time of the last operation of the last job:

$$\text{Makespan} = \max_{j \in J, o \in O_j} \{s[j][o] + p[j][o]\}$$

**Constraints:**

1.  $s[j][o] + p[j][o] \leq s[j][o + 1] \quad \forall j \in J, o \in O_j$
2.  $s[j][o] \geq s[j'][o'] + p[j'][o'] \quad \forall j, j' \in J, o \in O_j, o' \in O_{j'}$
3.  $\sum_{m \in M} x[j][o][m] = 1 \quad \forall j \in J, o \in O_j$
4.  $\sum_{o \in O_j} x[j][o][m] \leq 1 \quad \forall j \in J, m \in M$
5.  $\sum_{j \in J} x[j][o][m] \leq 1 \quad \forall o \in O, m \in M$

| Variable     | Definition   |
|--------------|--|
| $J$          | Set of jobs, where each job $j \in J$ has a predefined sequence of operations to be performed.     |
| $M$          | Set of machines, where each machine $m \in M$ is capable of executing specific operations.         |
| $O$          | Set of operations, where each operation $o \in O$ corresponds to a specific task within a job.     |
| $p[j][o]$    | Processing time required for operation $o$ of job $j$ .  |
| $s[j][o]$    | Start time of operation $o$ of job $j$ .   |
| $C[j][o]$    | Completion time of operation $o$ of job $j$ .  |
| $x[j][o][m]$ | Binary decision variable indicating whether operation $o$ of job $j$ is processed on machine $m$ . |

**Table 2.1:** Variables of the Traditional JSP

Here, Constraint 1 ensures that the start time of each operation precedes the start time of the subsequent operation in the job's sequence. Constraint 2 enforces the precedence relationship between operations of different jobs. Constraint 3 guarantees that each operation is assigned to exactly one machine. Constraint 4 ensures that each machine processes at most one operation at a time. Constraint 5 guarantees that each operation is assigned to at most one machine across all jobs.

### 2.1.2 | Flexible Job Shop Scheduling Problem (FJSP)

The FJSP extends the traditional JSP by introducing additional flexibility in selecting machines for each operation. Each job still has a predefined sequence of operations, but now there is a set of machines available for processing each operation. The objective remains the same: minimizing the makespan. Additionally, the FJSP considers the transfer time between machines, which is a predefined input variable. The mathematical model for the FJSP with transfer time can be formulated as follows:

#### Objective Function:

Minimize: Makespan

Here, the makespan can be expressed as the completion time of the last operation of the last job:

$$\text{Makespan} = \max_{j \in J, o \in O_j} \{s[j][o] + p[j][o]\}$$

#### Constraints:

1.  $s[j][o] + p[j][o] + \sum_{m' \in M[j][o] \setminus \{m\}} t[m'][m] \leq s[j][o+1] \quad \forall j \in J, o \in O_j, m \in M[j][o]$
2.  $s[j][o] \geq s[j'][o'] + p[j'][o'] + \sum_{m' \in M[j'][o'] \setminus \{m\}} t[m'][m] \quad \forall j, j' \in J, o \in O_j, o' \in O_{j'}, m \in M[j][o], m' \in M[j'][o']$
3.  $\sum_{m \in M[j][o]} x[j][o][m] = 1 \quad \forall j \in J, o \in O_j$
4.  $\sum_{o \in O_j} x[j][o][m] \leq 1 \quad \forall j \in J, m \in M$
5.  $\sum_{j \in J} x[j][o][m] \leq 1 \quad \forall o \in O, m \in M$

The variables and their definitions are summarized in Table 2.2.

In addition to the constraints of the JSP, the FJSP introduces Constraint 1, which ensures that the

| Variable     | Definition  |
|--------------|---|
| $J$          | Set of jobs, where each job $j \in J$ has a predefined sequence of operations to be performed.                        |
| $M$          | Set of machines, where each machine $m \in M$ is capable of executing specific operations.                            |
| $O$          | Set of operations, where each operation $o \in O$ corresponds to a specific task within a job.                        |
| $p[j][o]$    | Processing time required for operation $o$ of job $j$ .   |
| $s[j][o]$    | Start time of operation $o$ of job $j$ .  |
| $t[m'][m]$   | Transfer time required to move the processing from machine $m'$ to machine $m$ .                                      |
| $x[j][o][m]$ | Binary decision variable that takes value 1 if operation $o$ of job $j$ is assigned to machine $m$ , and 0 otherwise. |

**Table 2.2:** Variables of the Flexible Job Shop Scheduling Problem (FJSP)

start time of each operation considers both the processing time and the transfer time between machines. Constraint 2 enforces the precedence relationship between operations of different jobs, taking into account the processing time and transfer time as well. Constraint 3 guarantees that each operation is assigned to exactly one machine from the available set of machines. Constraints 4 and 5 are similar to the JSP, ensuring that each machine processes at most one operation at a time, and each operation is assigned to at most one machine across all jobs.

### 2.1.3 | Comparison

The Traditional Job Shop Scheduling Problem (JSP) and the Flexible Job Shop Scheduling Problem (FJSP) can be compared based on several aspects:

#### ■ Machine Assignment:

- JSP: The machines are fixed for each operation and cannot be changed during the scheduling process.
- FJSP: The FJSP allows for a set of machines available for each operation, providing flexibility in machine assignment.

#### ■ Scheduling Flexibility:

- JSP: The JSP has a fixed job and operation sequence that needs to be strictly followed.
- FJSP: The FJSP provides more flexibility in selecting the sequence of operations for each job, allowing for improved scheduling possibilities.

#### ■ Complexity:

- JSP: The JSP is a well-studied problem with established solution techniques and algorithms.
- FJSP: The FJSP is an extension of the JSP and is generally considered more complex due to the added flexibility in machine assignment and operation sequencing.

#### ■ Optimization Objective:

- JSP and FJSP: Both problems aim to minimize the makespan, which represents the total time required to complete all jobs.

#### ■ Solution Methods:

- JSP: Various solution methods exist for the JSP, including mathematical programming, heuristic algorithms, and metaheuristic approaches.
- FJSP: The FJSP often requires more advanced solution methods to handle the increased complexity, such as hybrid algorithms or decomposition-based approaches.



## 2.2 | Assumptions and Clarifications

In this subsection, we will give the assumptions we made in our simulation and give reasonable clarifications.

- Assumption 1: The R9 robot can pick the machined part from the workspace of R3, R4, R5, R6 and place it only on the point L8 on the conveyor belt.  
 Clarification: This assumption is made mainly considering that the robot R9 cannot be used to transfer the part between different points on the conveyor belt, or the problem will become more complex to compare the transfer time by the conveyor belt and the robot R9. Besides, since the function of R9 is only pick and place, it's reasonable to assume that robot R9 cannot be used to transfer part between different locations.
- Assumption 2: The R9 robot can only pick the assembled Luban lock from the point L1 on the conveyor belt and place it on the workspace of CNC1, CNC2.  
 Clarification: This assumption is made to simplify the dynamic process of robot R9, since it can move from two different points and different reachable range at different points. With this assumption, the robot R9 must move at the left point and pick the assembled Luban Lock at L1 and place it on CNC1 or CNC2.
- Assumption 3: The nine parts of one Luban lock should be sequentially machined and placed on the workspace of R8 or R1 by R7 or R2. But the sequence for parts from different Luban lock isn't limited.  
 Clarification: This assumption is made mainly considering the difficulties in building the plant in simulation. In real situation, it's also reasonable to machine nine parts of a Luban lock sequentially, since these nine parts differ from each other and should be placed on R1 or R8 at certain sequence.
- Assumption 4: There exists sensors on the conveyor belt, which will stop the part on the conveyor belt when the part reaches its next station.  
 Clarification: This assumption is made to simplify the calculation of transferring time between each machine. It's also reasonable and practical since we can install proximity sensors and RFID equipments to detect whether there is one part at certain point and identify that part. With this assumption, the transferring time between every two machine will be minimized and determined solely by the distance and speed of conveyor belt.
- Assumption 5: No machine will experience the situation of breakdown in our consideration.  
 Clarification: This assumption is made to simplify the problem, although breakdown happens in real plant and the commonly used algorithms are capable of addressing this situation. The situation of breakdown will be considered in future.

## 2.3 | Mathematical Model

In this subsection, we will explain the mathematical model of the FJSP specifically for Luban lock in our problem. At first, we define the input variables, which are Job  $J$ , Machine  $M$ , Operation  $O$ , Processing time  $p$ , Transfer time  $t$ , Precedence Matrix  $U$ , Scheduling variable  $x$ , Start time  $s$  and Completion time  $c$ , which are summarized in Table 2.3

The precedence matrix  $U$  is defined as follows,

$$U = \begin{bmatrix} 0 & 1 & \dots & 1 & 1 \\ 0 & 0 & 1 & \dots & 1 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}_{33 \times 33} \quad (2.1)$$

| Variable     | Quantity Type | Definition  |
|--------------|---------------|---|
| $J$          | Integer       | Set of jobs, where each job $j \in J$ represents producing one Luban lock, $j \in [1, 10]$ .  |
| $M$          | Integer       | Set of machines, where $m \in [1, 11]$ and $M = \{R1, R2, R3, R4, R5, R6, R7, R8, R9, CNC1, CNC2\}$ .   |
| $O$          | Integer       | Set of operations, where $o \in [1, 33]$ . The detailed operation content is summarized in Fig. 9.1.  |
| $p[j][o][m]$ | Continuous    | Processing time required for operation $o$ of job $j$ on machine $m$ . These processing times are defined according to the real-world situation and summarized in Table 5.1.          |
| $s[j][o][m]$ | Continuous    | Starting time of operation $o$ of job $j$ on machine $m$ .  |
| $t[m'][m]$   | Continuous    | Transferring time required to move the processing from machine $m'$ to machine $m$ , which is defined according to the real-world situation and given in Eqn. 2.2.                    |
| $x[j][o][m]$ | Binary        | Binary decision variable that takes value 1 if operation $o$ of job $j$ is assigned to machine $m$ , and 0 otherwise.   |
| $U[o][o']$   | Binary        | Precedence of operations in each job, which takes 1 if the operation $o$ should be scheduled ahead of operation $o'$ , and 0 otherwise. The precedence sequence is given in Eqn. 2.1. |
| $c[j][o][m]$ | Continuous    | Completion time of operation $o$ of job $j$ on machine $m$ .  |

**Table 2.3:** Variables of the Flexible Job Shop Scheduling Problem (FJSP)

Then, the transfer time  $t$  is defined as,

$$t = \begin{bmatrix} 0 & 0.75 & inf & inf & inf & inf & inf & inf & inf & inf & inf \\ 0.75 & 0 & 9 & 9 & 9 & 9 & inf & inf & inf & inf & inf \\ inf & 9 & inf & 0 & inf & inf & 4.5 & inf & 1.5 & inf & inf \\ inf & 9 & inf & inf & 0 & inf & 4.5 & inf & 1.5 & inf & inf \\ inf & 9 & inf & inf & 0 & inf & 4.5 & inf & 1.5 & inf & inf \\ inf & 9 & inf & inf & inf & inf & 4.5 & inf & 1.5 & inf & inf \\ inf & inf & 4.5 & 4.5 & 4.5 & 4.5 & 0 & 0.75 & inf & 5.25 & 5.25 \\ inf & inf & inf & inf & inf & inf & 0.75 & 0 & inf & inf & inf \\ inf & inf & 1.5 & 1.5 & 1.5 & 1.5 & inf & inf & 0 & 0.75 & 0.75 \\ inf & inf & inf & inf & inf & inf & inf & inf & 0.75 & 0 & inf \\ inf & inf & inf & inf & inf & inf & inf & inf & 0.75 & inf & 0 \end{bmatrix}_{11 \times 11} \quad (2.2)$$

In this matrix, 0.75 represents the time for moving the part between adjacent points on the conveyor belt, which equals to the unit length of conveyor belt divided by its speed. 9 represents moving through 12 units length, 4.5 represents 6 units length and 5.25 represents 7 units length. Specially, in the column 9 and row 9, 0.75 and 1.5 represent the time for the operation of R9.

The mathematical model can be formulated as,

$$\begin{aligned} & \min_{x[j][o][m]} \text{makespan} \\ \text{s.t.} & \begin{cases} \text{makespan} \geq c[j][o][m], \text{ for } \forall j, \forall o, \forall m \\ c[j][o][m] == \sum_{m=1}^{11} (s[j][o][m] + p[j][o][m]), \text{ for } \forall j, \forall o \\ \sum_{m=1}^{11} x[j][o][m] == 1, \text{ for } \forall j, \forall o \\ s[j][o1][m] + p[j][o1][m] \leq s[j][o2][m], \text{ for } \forall j, \forall U(o1, o2) = 1, \forall m \\ s[j][o1][m] + p[j][o1][m] + t[m][l] \leq s[j][o2][l], \text{ for } \forall j, \forall U(o1, o2) = 1, \forall m, l \end{cases} \end{aligned}$$



## 3 | Optimization Method

### 3.1 | Optimization Methods for Flexible Job Shop Scheduling Problem

#### 3.1.1 | Combinatorial Optimization and FJSP

Combinatorial optimization is a field of study that deals with finding the best solution from a finite set of possibilities for optimization problems with discrete variables, as illustrated in Fig. 3.1. It involves searching through a large combinatorial space to identify the optimal arrangement or combination of elements that satisfies certain constraints and optimizes a given objective function. The Flexible Job Shop Scheduling Problem (FJSP) falls under the realm of combinatorial optimization due to its discrete decision variables and the combinatorial nature of its solution space.

In the FJSP, the objective is to minimize the makespan by assigning operations to machines in an optimal manner while respecting precedence constraints and resource limitations. The discrete variables in FJSP include the assignment of operations to machines, the sequencing of operations within jobs, and the allocation of time slots for each operation. The solution space of FJSP is combinatorial, as there are multiple possible combinations and permutations of operations and machines to consider.

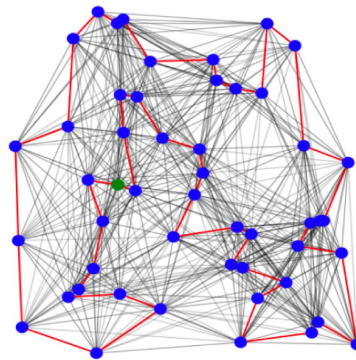


Figure 3.1: Illustration for combinatorial optimization problem

#### 3.1.2 | Common Algorithms for Combinatorial Optimization and FJSP

Several algorithms have been developed to solve combinatorial optimization problems, including the FJSP. Each algorithm has its own strengths and weaknesses, making them suitable for different problem instances and requirements. Here, we review some commonly used algorithms for solving combinatorial optimization problems and the FJSP:

**1. Integer Linear Programming (ILP):** ILP formulations provide a mathematical programming approach to model and solve combinatorial optimization problems. They use binary or integer decision variables to represent the choices and constraints of the problem. ILP solvers employ specialized algorithms, such as branch-and-bound or cutting plane methods, to explore the solution space and find an optimal solution. ILP offers exact solutions but may suffer from scalability issues for large problem instances.

**2. Metaheuristics:** Metaheuristic algorithms, such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Simulated Annealing (SA), are iterative optimization techniques that use heuristics to search through the solution space. These algorithms explore the space by iteratively generating and improving candidate solutions. Metaheuristics are suitable for finding near-optimal solutions and handling complex, large-scale combinatorial optimization problems. However, they do not guarantee finding the global optimum.

**3. Heuristic Search Algorithms:** Heuristic search algorithms, including Depth-First Search (DFS), Breadth-First Search (BFS), and A\* algorithm, systematically explore the solution space by evaluating promising paths or solutions based on heuristics. These algorithms are particularly effective for problems with a defined search tree or graph structure. They may find an optimal solution if properly guided by appropriate heuristics, but their performance heavily depends on the problem structure and quality of heuristics.

**4. Constraint Programming (CP):** CP is a declarative programming approach that models and solves combinatorial optimization problems using constraints and variables. CP solvers iteratively search for

feasible solutions by enforcing constraints and making local improvements. CP is suitable for problems with complex constraints and can handle both discrete and continuous variables. However, it may struggle with large problem instances due to the search complexity.

Each algorithm has its advantages and disadvantages, which are summarized in Table 3.1. The choice of algorithm depends on the problem size, complexity, available computational resources, and the trade-off between solution quality and computational time.

| Algorithm                        | Pros                                   | Cons                                     |
|----------------------------------|--|--|
| Integer Linear Programming (ILP) | Provides exact solutions               | Scalability issues for large instances   |
| Metaheuristics                   | Suitable for large-scale problems      | No guarantee of global optimality        |
| Heuristic Search Algorithms      | Effective for defined search structure | Performance depends on problem structure |
| Constraint Programming (CP)      | Handles complex constraints            | Search complexity for large instances    |

**Table 3.1:** Pros and cons of commonly used algorithms

### 3.2 | MATLAB Optimization Toolbox

The MATLAB Optimization Toolbox, an extensive collection of functions and algorithms, can be effectively utilized to solve complex optimization problems. In the context of the Flexible Job Shop Scheduling Problem (FJSP), the Optimization Toolbox offers a wide range of optimization techniques that can be leveraged to find optimal schedules and minimize the makespan.

By formulating the FJSP as a mathematical optimization problem, you can employ the Optimization Toolbox's capabilities to define the objective function, set constraints, and optimize the schedule. The choice of specific optimization algorithms from the toolbox will depend on the characteristics of the problem and the specific requirements.

For example, you can utilize the toolbox's linear programming functions to solve FJSP instances with linear objective functions and constraints. Alternatively, for more complex FJSP instances involving nonlinear objective functions or constraints, you can apply nonlinear optimization algorithms provided by the toolbox.

The Optimization Toolbox also enables the incorporation of additional features specific to the FJSP. For instance, you can include custom transfer time between machines as input variables and constraints, allowing for a more accurate representation of the scheduling problem.

By harnessing the power of the MATLAB Optimization Toolbox, you can efficiently explore the solution space of the FJSP and identify schedules that optimize the makespan, ultimately improving the efficiency and productivity of the job shop environment.

**Citation:** MathWorks. MATLAB Optimization Toolbox. Accessed June 30, 2023. <https://www.mathworks.com/products/optimization.html>.

### 3.3 | MATLAB code

The pseudocode in the Algorithm 1 shows the structure of the MATLAB script used to solve the FJSP problem. The complete MATLAB script can be found in the Appendix I, or downloaded in the page. Next, we will explain the pseudocode line by line.

- First, the problem data is defined, including the processing times (`processingTimes`), the availability of machines for each operation (`machineAvailability`), and the transfer times between operations (`transferTimes`).
- The necessary variables and optimization problem object, `problem`, are created. These variables include the start times of operations (`startTimes`), the makespan (overall completion time, `makespan`), and the completion times of each operation (`completionTimes`).
- The objective sense of the `problem` is set to `minimize`, indicating that the goal is to minimize the makespan.

**Algorithm 1** Matlab Pseudocode

---

```

1: Define problem data
2: Initialize processingTimes matrix
3: Initialize U matrix
4: Initialize transferTimes matrix
5: Create a binary variable  $x_{\text{scheduling}}$  to represent operation assignment
6: Create an optimization problem object 'problem'
7: Create variables: startTimes, makespan, completionTimes
8: Set the objective sense of 'problem' as 'minimize'
9: Add constraints for completion times of each operation
10: for each job  $j$  do
11:   for each operation  $o$  do
12:     Add constraint:  $\sum_m (\text{startTimes}_{jmo} + \text{processingTimes}_{jmo}) = \text{completionTimes}_{jo}$ 
13: Add constraint for makespan
14: for each job  $j$  do
15:   Add constraint:  $\text{makespan} \geq \max\{\text{completionTimes}_{j1}, \text{completionTimes}_{j2}, \dots, \text{completionTimes}_{jm}\}$ 
16: Add constraints for operation assignment and scheduling
17: for each job  $j$  do
18:   for each operation  $o$  do
19:     Add constraint:  $\sum_m x_{\text{scheduling}_{jmo}} = 1$ 
20: for each job  $j$  do
21:   for each operation  $o_1$  do
22:     for each operation  $o_2$  do
23:       if  $U_{o_1 o_2} == 1$  then
24:         Add constraint:  $\text{startTimes}_{jmo_1} + \text{processingTimes}_{jmo_1} \leq \text{startTimes}_{jmo_2}$ 
25: Set the objective function of 'problem' as 'makespan'
26: Choose a suitable solver and its options
27: Solve the optimization problem using the chosen solver and options
28: Print the solution, objective value, and solver status
29: Retrieve the solution values for  $x_{\text{scheduling}}$ , startTimes, makespan, completionTimes, etc.

```

---

- Constraints are added to ensure the correct completion times for each operation. For each job and operation, the sum of the start time and processing time of all machines assigned to that operation should equal the completion time.
- A constraint is added to guarantee that the makespan is greater than or equal to the maximum completion time among all operations for each job.
- Constraints are added to ensure that each operation is assigned to exactly one machine. For each job and operation, the sum of the binary scheduling variables representing the assignment should equal 1.
- Constraints are added to enforce the scheduling order between operations based on the availability of machines. For each pair of operations with a transfer time between them (indicated by the binary matrix  $U$ ), if the transfer is allowed ( $U = 1$ ), the completion time of the first operation plus its processing time should be less than or equal to the start time of the second operation.
- The objective function of the problem is set as the makespan, indicating that the objective is to minimize the overall completion time.
- A suitable solver and its options are chosen to solve the optimization problem.
- The optimization problem is solved using the chosen solver and options.
- The solution, objective value, and other relevant information are obtained and can be further analyzed or utilized as needed.

## 4 | Plant Simulation

### 4.1 | Software Description

To build the model, we use Plant Simulation software. Tecnomatix Plant Simulation is a powerful software tool used for modeling, simulating, and optimizing complex manufacturing and logistics systems. It is part of the Tecnomatix suite of digital manufacturing solutions developed by Siemens Digital Industries Software.

Plant Simulation provides a comprehensive platform for creating virtual models of production systems, including factories, assembly lines, warehouses, and material flow networks. It allows engineers, planners, and decision-makers to analyze and optimize various aspects of the system, such as throughput, cycle times, resource utilization, and material handling.

With Plant Simulation, users can visually design their production systems using a drag-and-drop interface that includes a rich library of pre-built objects and components. These objects represent different entities, such as machines, conveyors, operators, and storage areas, which can be combined and configured to build a detailed simulation model.

Once the model is created, Plant Simulation enables users to define the behavior and interactions of the various components using a powerful set of modeling and programming tools. This allows for the creation of dynamic simulations that accurately represent the real-world system, including factors like variability, breakdowns, maintenance schedules, and production rules.

Simulation scenarios can be run and analyzed to gain insights into system performance, identify bottlenecks, optimize resource allocation, and evaluate alternative production strategies. Users can perform "what-if" analyses to test different scenarios, evaluate the impact of changes, and make informed decisions to improve productivity, efficiency, and overall system performance.

In addition to its simulation capabilities, Plant Simulation offers features for statistical analysis, visualization, and reporting. It provides tools for generating detailed reports, charts, and animations, which can be used to communicate findings, present results, and facilitate collaboration among stakeholders.

The key uses of the software are:

- **Production System Design:** Plant Simulation allows engineers and planners to design and optimize production systems before their physical implementation. It helps in determining the layout, capacity, and configuration of manufacturing facilities, including the arrangement of machines, workstations, and material handling systems.
- **Process Optimization:** The software enables users to analyze and optimize production processes to improve efficiency and reduce cycle times. It helps identify bottlenecks, optimize resource allocation, and streamline workflow to increase productivity and throughput.
- **Production Planning and Scheduling:** Plant Simulation supports production planning and scheduling activities by simulating different scenarios and evaluating their impact on system performance. It helps in determining optimal production sequences, shift schedules, and resource utilization to meet production targets and minimize downtime.










**Figure 4.1:** Tecnomatix Plant Simulation 2302 software

## 4.2 | Components

### 4.2.1 | Material Flow

In the modeling process, we totally use 8 parts in material flow, which are “Connector”, “Source”, “Part”, “Container”, “Station”, “AssemblyStation”, “PickAndPlace”, “Conveyor”, and “Drain”.

| Name      | Icon  | Effects   |
|-----------|---|---|
| Connector |    | <ul style="list-style-type: none"> <li>■ <b>Material Flow Connection:</b> The primary function of the Connector object is to establish a material flow connection between two objects within the same framework.</li> <li>■ <b>Interface Connection:</b> The Connector object can also link an object with the interface of a framework. The interface represents the connection point between different frameworks or modules within the simulation model.</li> <li>■ <b>Direction and Routing:</b> Connectors can be configured to define the direction and routing of material flow between objects. Users can specify the path that entities will follow, determining the sequence of objects or processes they will pass through.</li> </ul> |
| Source    |  | <ul style="list-style-type: none"> <li>■ <b>Entity Generation:</b> The Source object generates entities according to a specified arrival pattern or rate. Users can define the inter-arrival time between entities, allowing for constant or time-varying arrivals.</li> <li>■ <b>Quantity and Capacity:</b> Users can configure the Source object to generate a specific quantity of entities at the start of the simulation or continuously throughout the simulation run. The capacity of the Source object can be limited .</li> </ul>  |
| Part      |  | <ul style="list-style-type: none"> <li>■ <b>Representation of Physical Objects:</b> The Part object represents tangible items within the simulation model. Each Part object corresponds to a specific entity or item that moves through the production system.</li> <li>■ <b>Attributes and Characteristics:</b> Parts can be assigned attributes and characteristics to represent their specific properties. These attributes can be used to define the behavior and constraints of the parts within the simulation model.</li> <li>■ <b>Statistical Analysis:</b> The Part object can be used to collect statistical data about part flow, cycle times, processing times, and other performance metrics.</li> </ul>                             |

| Name             | Icon  | Effects   |
|------------------|---|---|
| Container        |    | <ul style="list-style-type: none"> <li>■ <b>Storage and Transport:</b> The primary function of the Container object is to provide a designated space for storing entities. It represents physical storage units that hold items within the simulation model. Containers can be stationary or mobile, enabling the transport of entities between different locations within the system.</li> <li>■ <b>Capacity and Constraints:</b> Containers have a defined capacity that determines the maximum number of entities they can hold.</li> <li>■ <b>Loading and Unloading:</b> The Container object enables the loading and unloading of entities into and out of the storage units.</li> </ul> |
| Station          |  | <ul style="list-style-type: none"> <li>■ <b>Processing and Operation:</b> Stations simulate the processing or operation that occurs at a specific location within the production system. Entities, such as parts or products, move to the station for processing, assembly, inspection.</li> <li>■ <b>Statistical Analysis:</b> The Station object can collect statistical data related to processing times, idle times, utilization rates, and other performance metrics.</li> <li>■ <b>Processing Time and Variability:</b> Each station has a defined processing time for performing the associated operation on entities.</li> </ul>  |
| Assembly Station |  | <ul style="list-style-type: none"> <li>■ <b>Assembly Workstation Representation:</b> The Assembly Station object serves as a representation of a physical workstation dedicated to assembly operations within the production system. It simulates the processes involved in assembling components or parts to create finished products.</li> <li>■ <b>Assembly Time and Variability:</b> Each Assembly Station has a defined assembly time, representing the time taken to perform the assembly operations on entities.</li> </ul>  |
| Drain            |  | <ul style="list-style-type: none"> <li>■ <b>Entity Removal:</b> The Drain object is responsible for removing entities from the simulation model. It represents the point where entities exit the system, indicating the end of their lifecycle within the production process.</li> <li>■ <b>Entity Statistics:</b> The Drain object can collect statistical data related to the entities that pass through it.</li> </ul>   |






| Name           | Icon   | Effects  |
|----------------|--|--|
| Pick And Place |   | <ul style="list-style-type: none"> <li>■ <b>Pick and Place Locations:</b> The PickAndPlace object defines the source and destination locations for the pick-and-place operations. Entities are picked up from the source location and placed at the designated destination location.</li> <li>■ <b>Timing and Synchronization:</b> PickAndPlace enables the definition of timing and synchronization parameters for the pick-and-place operations. Users can specify the time taken for picking and placing entities, as well as any delays or synchronization requirements between multiple PickAndPlace objects or other processes within the system.</li> <li>■ <b>Path Planning and Collision Avoidance:</b> PickAndPlace supports path planning and collision avoidance functionalities to ensure safe and efficient movement of entities.</li> </ul> |
| Conveyor       |  | <ul style="list-style-type: none"> <li>■ <b>Material Transportation:</b> The Conveyor object simulates the movement of entities along a predefined path within the production system. It represents the conveyance systems.</li> <li>■ <b>Routing and Paths:</b> Conveyors can have multiple routing options and paths within the simulation model.</li> <li>■ <b>Speed and Capacity:</b> The Conveyor object supports the specification of conveyor speed and capacity parameters. Users can set the speed of conveyors to simulate realistic transportation rates.</li> </ul>  |

Table 4.1: Material Flow

#### 4.2.2 | Information Flow

In addition, we use “List”, “Variable” and “Method” in information flow.

| Name | Icon  | Effects   |
|------|---|---|
| List |  | <ul style="list-style-type: none"> <li>■ <b>Entity or Data Storage:</b> The List object allows for the storage of entities or data elements within a collection. It provides a container to hold and organize these items, facilitating efficient retrieval, modification, and analysis.</li> <li>■ <b>Iteration and Looping:</b> Lists can be iterated over using loops or iteration methods, allowing users to access and process each item within the list.</li> </ul> |



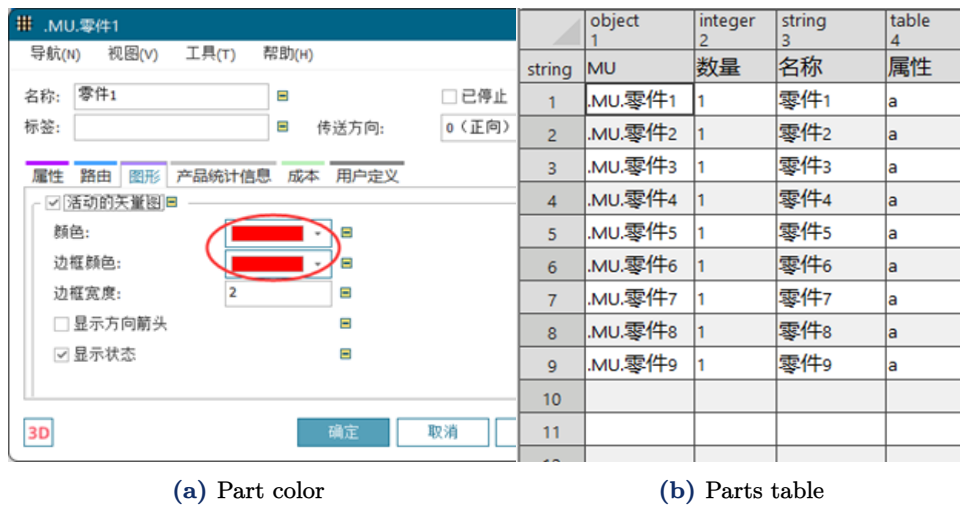
| Name     | Icon  | Effects   |
|----------|---|---|
| Method   |    | <ul style="list-style-type: none"> <li>■ Custom Procedures: Methods allow users to define custom procedures or algorithms tailored to their specific simulation needs. They encapsulate a series of instructions, operations, or calculations that can be reused and invoked at different points within the simulation model.</li> <li>■ Data Manipulation: Methods provide a mechanism for manipulating and processing data within the simulation model. They can accept input parameters, perform calculations, modify variables, update entity attributes, or interact with other simulation objects to influence system behavior.</li> <li>■ Algorithmic Complexity: Methods support the implementation of complex algorithms or decision-making processes within the simulation model. Users can define conditions, loops, branching logic, or other control structures to handle intricate simulation scenarios and model realistic system behavior.</li> </ul>   |
| Variable |  | <ul style="list-style-type: none"> <li>■ Data Storage: The Variable object serves as a container for storing data within the simulation model. It can hold various types of information, including numerical values, Boolean values, strings, or custom-defined data types.</li> <li>■ Dynamic Data Management: Variables enable the manipulation and management of dynamic data. Users can perform operations such as assignment, arithmetic calculations, logical operations, or string manipulations on the values stored in variables.</li> <li>■ Data Tracking and Tracing: Variables allow for tracking and tracing the changes in data values throughout the simulation model. Users can monitor the values of variables at specific points in the model or during specific events, enabling analysis and understanding of system behavior.</li> <li>■ Data Sharing and Communication: Variables facilitate data sharing and communication between different components or objects within the simulation model.</li> </ul> |

Table 4.2: Information Flow

## 4.3 | Modeling

### 4.3.1 | Part

In the production process of LuBan locks, there are nine parts assembled in a specific order. To distinguish between different parts, we use color variations for identification. To ensure the order and stability of part supply, we use a data table to control the generation of parts from the material source in the correct sequence.



(a) Part color

(b) Parts table

**Figure 4.2:** Modify the color of parts and adjust the production sequence of parts.

Furthermore, to avoid blockages and excessive accumulation, we employ a dynamic adjustment of the discharge interval for the material source. By using “Method” to modify the discharge intervals based on production demands and pacing, we ensure a smooth and continuous supply of parts.

We also implement control at the outlet of the material source to modify the destination attribute of each part, which facilitates subsequent recognition.

```

number:=number+1
if number mod 4 =1
    @.destination=R3
    ?.interval=2.5
elseif number mod 4 =2
    @.destination=R4
elseif number mod 4 =3
    @.destination=R5
elseif number mod 4 =0
    @.destination=R6
    ?.interval=33
end
if in3<4
    @.move(R9_P1)
end

```

**Figure 4.3:** Parts destination and discharge intervals

These measures enable us to achieve accuracy, sequence, and stability in the supply of parts during the production process of LuBan locks, ensuring the overall smooth operation of the production flow.

#### 4.3.2 | Conveyor

In the production line of LuBan locks, conveyors play a vital role as a key component for transporting parts to different nodes. Additionally, it is important for the robots to be able to place parts at various nodes on the conveyor. However, in the simulation software, the robotic arms are only capable of placing parts at the starting segment of the conveyor. Therefore, we have designed a multi-segment conveyor system where each segment serves as a node, enabling the placement and retrieval of parts at different nodes. To ensure accurate control and operation of parts on the conveyor, we utilize sensors for recognition. Sensors are installed at nodes L14 and L15 of the conveyor, which detect the attributes of the parts, such as their destination, as they pass through these nodes and trigger corresponding actions.

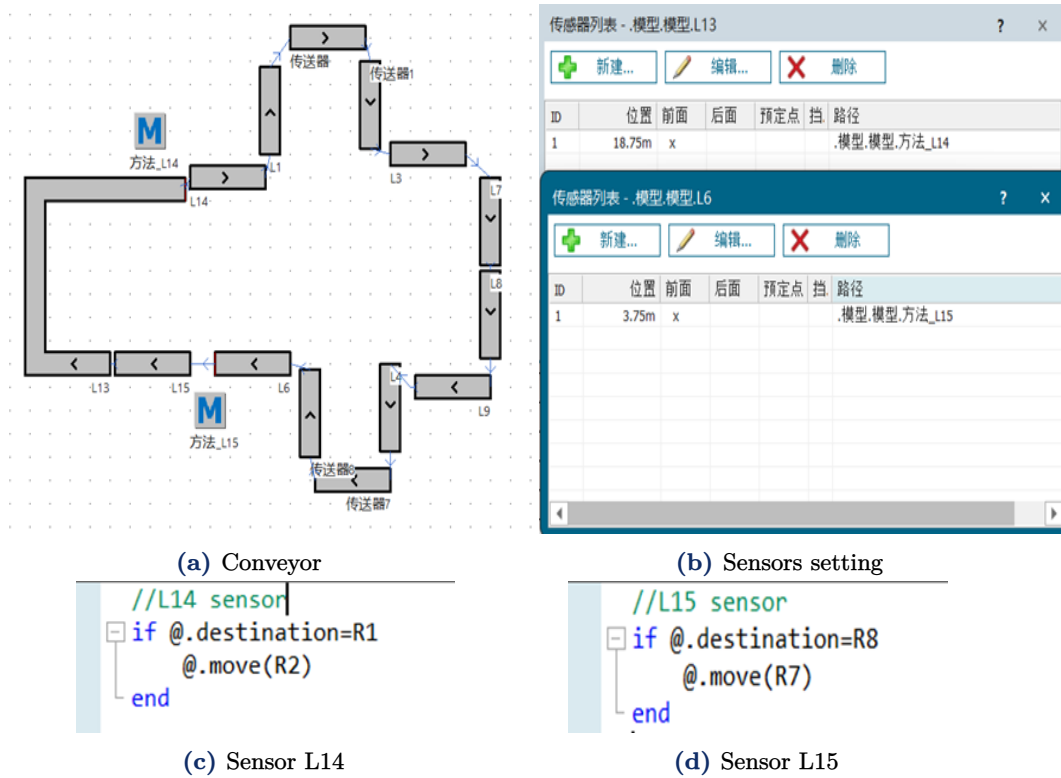


Figure 4.4: Conveyor and associated Settings

### 4.3.3 | Pick and place robot

In the production flow of LuBan locks, the robot R9 plays a crucial role. According to the design requirements, the robot R9 should be capable of moving between positions P1 and P2, with a required time of 0.75 seconds for each movement. However, in the simulation software, we are unable to directly implement the movement functionality of the robot. Therefore, to simulate the functionality of robot R9, we utilize two separate robots placed at P1 and P2 respectively. The time interval for transporting a part from P1 to P2 is set to 0.75 seconds.

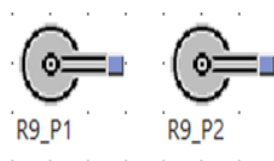


Figure 4.5: R9

Determining the appropriate node for placing each part is crucial for all robots. We use "Method" to identify the destination information of each part, allowing us to adjust the placement of parts by the robots at the respective nodes. This intelligent mechanism ensures accurate part placement during the production process, thereby enhancing the efficiency of the entire production flow.

### 4.3.4 | Processing robot

we use processing stations to simulate the process of manufacturing and labeling the parts. By adjusting the processing time at the stations, we can simulate the actual time required for processing and labeling. Additionally, at the exit of the processing stations, we use "method" to modify the destination attribute of the parts to facilitate subsequent assembly and other processes.

```

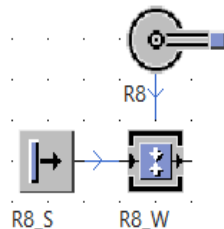
//method of R9 at P2
if @.destination=R3
  R9_P2.setDestination( @.destination )
elseif @.destination=R4
  R9_P2.setDestination( @.destination )
elseif @.destination=R5
  R9_P2.setDestination( @.destination )
elseif @.destination=R6
  R9_P2.setDestination( @.destination )
elseif @.destination=R8 or @.destination=R1
  R9_P2.setDestination( L8)
end

```

**Figure 4.6:** An example of robot placement code

#### 4.3.5 | Assembly robot

A LuBan lock is assembled by sequentially combining nine components. To simulate the assembly process, we utilize a material source and an assembly station, along with a robot mimicking either Robot R1 or R8.



**Figure 4.7:** Assembly robot R8

The specific assembly process is as follows:

- The material source produces a container and places it onto the assembly station.
- The robot receives parts from Robot R2 or R7 and places them onto the container (repeated nine times).
- Once the container is filled with the nine parts, the assembly station initiates the assembly process.
- After a certain assembly time, the assembly of the LuBan lock is completed. Before the assembled LuBan lock leaves the assembly station, we use “Method” to modify its destination attribute.

#### 4.3.6 | Complete process

The model includes a material source, robots, a conveyor belt, and assembly and labeling processes. The complete workflow of the entire model is as follows:

- The material source produces unprocessed parts and sets their destination as the processing stations (R3-6). Every 2.5 seconds, robot R9\_P1 grabs a part from the material source and transfers it to R9\_P2 to simulate the movement process. R9\_P2 places the part into the processing station (R3-6) for processing. If all processing stations are occupied, the production time interval for the material source is adjusted to 33 seconds. After processing, the processing station modifies the destination of the part to the assembly station (R1 or R8). Then, R9\_P2 retrieves the part from the processing station and places it on conveyor belt node L8. This process is repeated nine times to handle nine parts.
- Sensors on the conveyor belt identify the destination of the parts at nodes L14 or L15. If the destination is R2, robot R1 places the part on the assembly station of R2. If the destination is R8, robot R7 places the part on the assembly station of R8. When the assembly station is filled with nine parts, the assembly process of the Luban lock begins. Once assembly is complete, the assembly

station modifies the destination of the lock to CNC1 or CNC2. Then, robot R2 or R8 places the lock back on the conveyor belt at nodes L14 or L15.

- When the Luban lock reaches nodes L1 or L4 on the conveyor belt, robot R9.P1 removes the lock from the conveyor belt and places it on the labeling station, where it waits for labeling. After the labeling process, R9.P1 takes the labeled lock and places it at the output.

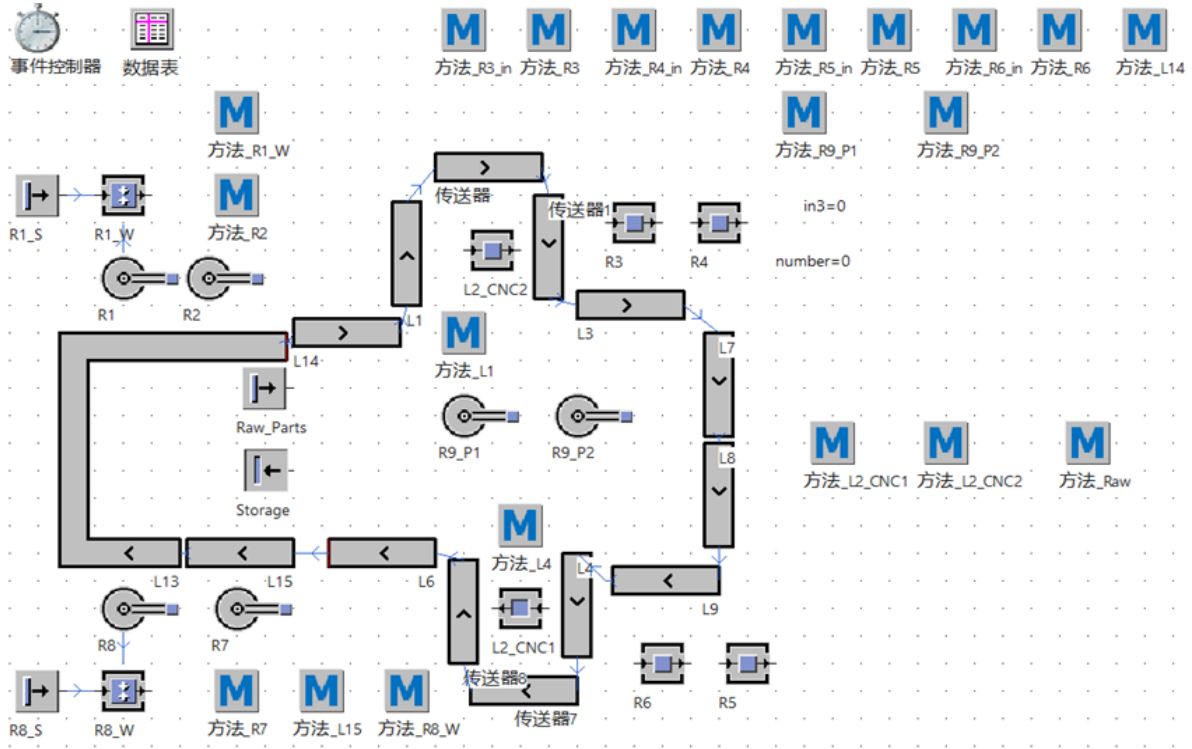


Figure 4.8: The complete model

Through the above process, we are able to simulate the operation of the Luban lock production assembly line, ensuring the sequential processing, assembly, and labeling of parts, and ultimately obtaining fully assembled Luban locks.

## 5 | Results

### 5.1 | Plant setting

Table 5.1: Machine Process Time

| Process                             | Time(s) |
|-------------------------------------|---------|
| R3-R6 processing                    | 30      |
| R1&R8 assembling                    | 20      |
| CNC1&CNC2 labeling                  | 5       |
| R9 move between two position        | 0.75    |
| R9 pick part from raw parts         | 0.25    |
| R9 place part to R3-R6              | 0.25    |
| R9 pick and place to other position | 0.75    |
| Other robot pick and place part     | 0.75    |



## 5.2 | Simulation results

In simulation, we simulate the worst case for this problem, which takes the longest makespan to produce 10 Luban locks at first, served as the baseline for the optimization. In the worst case, we schedule the whole process as operation by operation and job by job, which means that we only produce one Luban lock and machine one part of that Luban lock at one time. The baseline is denoted by the Type 1 in the results, shown in Fig. 5.1 and 5.2. Then, based on the intuition and experience gained during the project, we use the heuristic algorithm to increase the efficiency of machining machine and assembly machine to optimize the result, denoted by Type 2 and Type 3 respectively. The optimization algorithm proposed in Section 3 failed to provide the well scheduled plan (we've run the program beyond 24h in PC and it's still running), due to the curse of dimensionality and the complexity of this combinatorial FJSP problem. But we believe the result of Type 2 and Type 2 are also near optimal schedule.

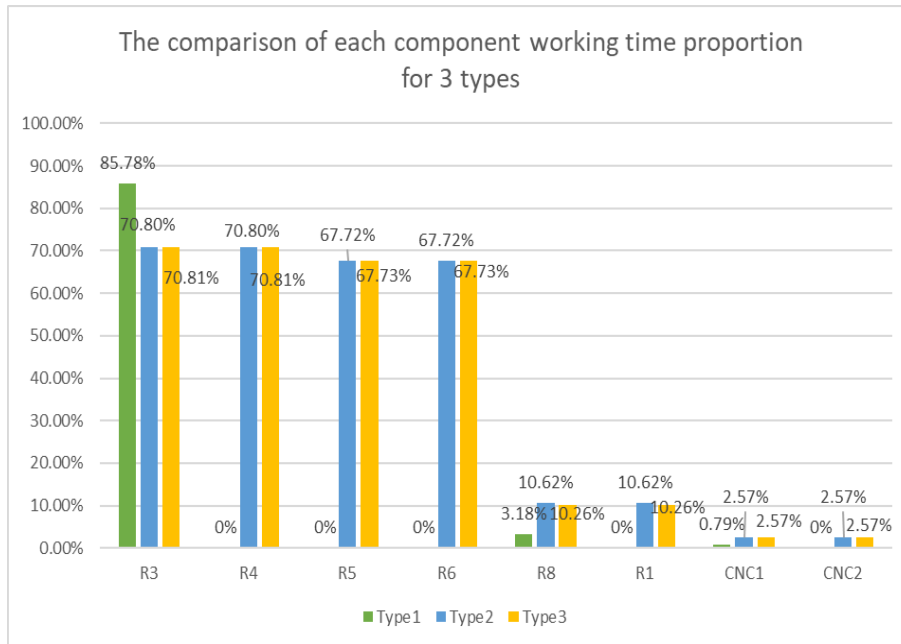


Figure 5.1: The comparison of each component working time proportion for 3 types

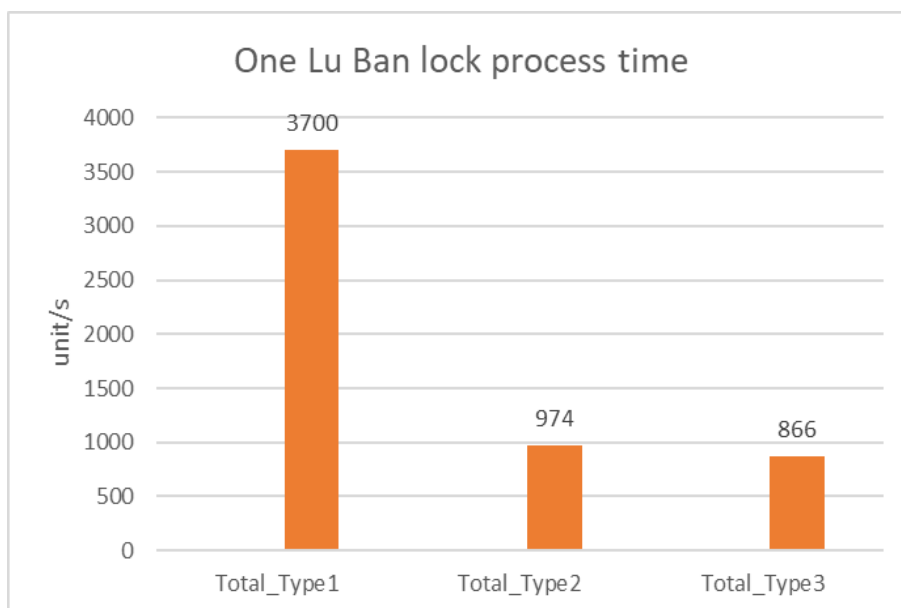


Figure 5.2: One Lu Ban lock process time

## 6 | Conclusion

We have designed three types of working mode using heuristic strategies mentioned before. The worst case, we only machine one part at one time. And we improve it by making four parts processed together, which increases the production rate and the time utilization. This method makes great contribution to the processing time for 10 Luban locks from 3700s to 974s. And we further improve the working mode by making the assembly machines work together which saves 104 seconds more. So that we found that there is little affect by making this modification. This observation indicates the bottle neck of the plant layout is the number of processing machines: R3-R6. To further improve the plant's processing rate, we may need to add more processing machine like R3-R6 or increase the processing speed of the machine to reduce processing time.

Finished this project, we've progressed a lot, not only our personal skills in many aspects, but also the team coherence. Better performance in future could be expected.

## 7 | Acknowledgement

All work of this project is finished by the team *SharpShooters*, supported by SHIEN-MING WU SCHOOL OF INTELLIGENT ENGINEERING, Dr. Gang Chen and all people who provided valuable assistance. Lots of online open-source websites provides useful materials, e.g. CSDN[1], Stack Overflow[4]. The photos of this report are mainly from Google Images[3]. The mathematical modeling process is mainly from S. Gaiardelli[2].

## 8 | References

- [1] CSDN. Csdn. <https://www.csdn.net/>, 1999.
- [2] Sebastiano Gaiardelli, Damiano Carra, Stefano Spellini, and Franco Fummi. On the impact of transport times in flexible job shop scheduling problems. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2022.
- [3] Google Images. Google images. <https://images.google.com/>, 2001.
- [4] Stack Overflow. Stack overflow. <https://stackoverflow.com/?products>, 2008.

## 9 | Appendix I: Operation Space

| Task, $t_{ij}$   | Executable Machine |
|--|--------------------|
| $j \in [1,9]$ , R9 pick the $j^{th}$ raw part and place it to any one of R3, R4, R5 and R6 | R9                 |
| $j \in [10,18]$ , primary process of $(j - 9)^{th}$ raw part                               | R3, R4, R5, R6     |
| $j \in [19,27]$ , pick the $(j - 18)^{th}$ part after primary process and place it to L8   | R9                 |
| $j = 28$ , pick the part and place it to R1 or R8  | R2, R7             |
| $j = 29$ , assembly the placed nine parts into a Luban Lock                                | R1, R8             |
| $j = 30$ , pick the assembled Luban Lock and place it to L14 or L15                        | R2, R7             |
| $j = 31$ , pick the assembled Luban Lock at L1 and place it to CNC1 or CNC2                | R9                 |
| $j = 32$ , place a label on the assembled Luban Lock                                       | CNC1, CNC2         |
| $j = 33$ , pick the completed Luban Lock and place it to Storage                           | R9                 |

Figure 9.1: Operation Space Table

## 10 | Appendix II: MATLAB Code

```

1 % Define problem data
2 numJobs = 10;
3 numMachines = 11;
4 numOperations = 33;
5
6 processingTimes = ones(numJobs, numMachines, numOperations);
7 processingTimes(1, :, 1) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 1.5, 100000, 100000];
8 processingTimes(1, :, 2) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 1.5, 100000, 100000];
9 processingTimes(1, :, 3) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 1.5, 100000, 100000];
10 processingTimes(1, :, 4) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 1.5, 100000, 100000];
11 processingTimes(1, :, 5) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 1.5, 100000, 100000];
12 processingTimes(1, :, 6) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 1.5, 100000, 100000];
13 processingTimes(1, :, 7) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 1.5, 100000, 100000];
14 processingTimes(1, :, 8) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 1.5, 100000, 100000];
15 processingTimes(1, :, 9) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 1.5, 100000, 100000];
16 processingTimes(1, :, 10) = [100000, 100000, 30, 30, 30, 30, 100000, 100000, 100000, 100000, 100000];
17 processingTimes(1, :, 11) = [100000, 100000, 30, 30, 30, 30, 100000, 100000, 100000, 100000, 100000];
18 processingTimes(1, :, 12) = [100000, 100000, 30, 30, 30, 30, 100000, 100000, 100000, 100000, 100000];
19 processingTimes(1, :, 13) = [100000, 100000, 30, 30, 30, 30, 100000, 100000, 100000, 100000, 100000];
20 processingTimes(1, :, 14) = [100000, 100000, 30, 30, 30, 30, 100000, 100000, 100000, 100000, 100000];
21 processingTimes(1, :, 15) = [100000, 100000, 30, 30, 30, 30, 100000, 100000, 100000, 100000, 100000];
22 processingTimes(1, :, 16) = [100000, 100000, 30, 30, 30, 30, 100000, 100000, 100000, 100000, 100000];
23 processingTimes(1, :, 17) = [100000, 100000, 30, 30, 30, 30, 100000, 100000, 100000, 100000, 100000];
24 processingTimes(1, :, 18) = [100000, 100000, 30, 30, 30, 30, 100000, 100000, 100000, 100000, 100000];
25 processingTimes(1, :, 19) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 0.75, 100000, 100000];
26 processingTimes(1, :, 20) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 0.75, 100000, 100000];
27 processingTimes(1, :, 21) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 0.75, 100000, 100000];
28 processingTimes(1, :, 22) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 0.75, 100000, 100000];
29 processingTimes(1, :, 23) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 0.75, 100000, 100000];
30 processingTimes(1, :, 24) = [100000, 100000, 100000, 100000, 100000, 100000, 100000, 100000, 0.75, 100000, 100000];

```

```

31 processingTimes(1, :, 25) = [100000,100000,100000,100000,100000,100000,100000,100000,100000,0.75,100000,100000];
32 processingTimes(1, :, 26) = [100000,100000,100000,100000,100000,100000,100000,100000,100000,0.75,100000,100000];
33 processingTimes(1, :, 27) = [100000,100000,100000,100000,100000,100000,100000,100000,100000,0.75,100000,100000];
34 processingTimes(1, :, 28) = [100000,0.75,100000,100000,100000,100000,100000,0.75,100000,100000,100000,100000];
35 processingTimes(1, :, 29) = [20,100000,100000,100000,100000,100000,100000,20,100000,100000,100000];
36 processingTimes(1, :, 30) = [100000,0.75,100000,100000,100000,100000,0.75,100000,100000,100000,100000];
37 processingTimes(1, :, 31) = [100000,100000,100000,100000,100000,100000,100000,100000,100000,0.75,100000,100000];
38 processingTimes(1, :, 32) = [100000,100000,100000,100000,100000,100000,100000,100000,100000,100000,5,5];
39 processingTimes(1, :, 33) = [100000,100000,100000,100000,100000,100000,100000,100000,100000,0.75,100000,100000];
40
41 processingTimes(2, :, :) = processingTimes(1, :, :);
42 processingTimes(3, :, :) = processingTimes(1, :, :);
43 processingTimes(4, :, :) = processingTimes(1, :, :);
44 processingTimes(5, :, :) = processingTimes(1, :, :);
45 processingTimes(6, :, :) = processingTimes(1, :, :);
46 processingTimes(7, :, :) = processingTimes(1, :, :);
47 processingTimes(8, :, :) = processingTimes(1, :, :);
48 processingTimes(9, :, :) = processingTimes(1, :, :);
49 processingTimes(10, :, :) = processingTimes(1, :, :);
50
51
52 % Sequence of operations in a job if U(o1,o2) == 1, o1 should be scheduled
53 % before o2
54 U = triu(ones(numOperations,numOperations) - diag(ones(numOperations,1)));
55
56 % transferTimes = ones(numMachines,numMachines);
57 transferTimes = [0,0.75,100000,100000,100000,100000,100000,100000,100000,100000,100000,100000,100000;
58 0.75,0,9,9,9,9,100000,100000,100000,100000,100000;
59 100000,9,100000,0,100000,100000,6*0.75,100000,1.5,100000,100000;
60 100000,9,100000,100000,0,100000,6*0.75,100000,1.5,100000,100000;
61 100000,9,100000,100000,100000,0,6*0.75,100000,1.5,100000,100000;
62 100000,9,100000,100000,100000,100000,6*0.75,100000,1.5,100000,100000;
63 100000,100000,6*0.75,6*0.75,6*0.75,6*0.75,0,0.75,100000,7*0.75,7*0.75;
64 100000,100000,100000,100000,100000,100000,0.75,0,100000,100000,100000;
65 100000,100000,1.5,1.5,1.5,1.5,100000,100000,0,0.75,0.75;
66 100000,100000,100000,100000,100000,100000,100000,100000,0.75,0,100000;
67 100000,100000,100000,100000,100000,100000,100000,100000,0.75,100000,0];
68
69
70 % Create a binary variable indicating if an operation is assigned to a machine
71 x_sch = optimvar('binvar', numJobs, numMachines, ...
72 numOperations, 'Type', 'integer', 'LowerBound', 0, 'UpperBound', 1);
73
74 % Create the MILP optimization problem
75 problem = optimproblem('ObjectiveSense', 'minimize');
76
77 % Create variables for start times of each operation
78 startTimes = optimvar('startTimes', numJobs, numMachines, numOperations, 'LowerBound', ...
79 0, 'Type', 'integer');
80 % Create variable for the final makespan
81 makespan = optimvar('makespan', 'LowerBound', 0, 'Type', 'integer');
82
83 % Create variables for the completion times of each operation
84 completionTimes = optimvar('completionTimes', numJobs, numOperations, 'LowerBound', ...
85 0, 'Type', 'continuous');
86 for j = 1:numJobs
87     for o = 1:numOperations
88         Problem.Constraints(sprintf('CompletionTimeofJob%dOperation%d', j, o)) = ...
89             sum(startTimes(j, :, o) + processingTimes(j, :, o)) == completionTimes(j, o);
90     end
91 end
92
93 % Add a constraint for makespan
94 for j = 1:numJobs
95     for o = 1:numOperations
96         problem.Constraints.(sprintf('MaxCompletionTimeJob%d', j)) = makespan >= completionTimes(j, o);
97     end
98 end
99
100 % Define the constraints
101
102 % Each operation of a job must be assigned to exactly one machine
103 for j = 1:numJobs

```

```

104     for o = 1:numOperations
105         problem.Constraints.(sprintf('Job%dOperation%d', j, o)) = sum(x_sch(j, :, o)) == 1;
106     end
107 end
108
109 % Each machine can process one task at a time
110 for m = 1:numMachines
111     for j1 = 1:numJobs
112         %for j2 = 1:numJobs
113             for o1 = 1:numOperations
114                 for o2 = 1:numOperations
115                     if U(o1,o2) == 1
116                         problem.Constraints.('EachMachineOneTaskAtATime') =...
117                             startTimes(j1,m,o1) + processingTimes(j,m,o1) <= startTimes(j1,m,o2);
118                     end
119                 end
120             end
121         %end
122     end
123 end
124
125 % Sequence constraint: Operation o2 must be scheduled after operation o1 within each job
126 for j = 1:numJobs
127     for m = 1:numMachines
128         for l = 1:numMachines
129             for o1 = 1:numOperations
130                 for o2 = 1:numOperations
131                     if U(o1,o2) == 1
132                         problem.Constraints.(sprintf('Job%dMachine%dOperation%dBeforOperation%d',...
133                             j, m, o1, o2)) = ...
134                             startTimes(j, m, o1) + processingTimes(j, m, o1) + transferTimes(m, l) <=...
135                             startTimes(j, l, o2);
136                     end
137                 end
138             end
139         end
140     end
141 end
142
143 % Define the objective function
144 problem.Objective = makespan;
145
146 % Solve the MINP problem
147 solver = 'ga'; % Solver for integer programming problems
148 options = optimoptions(solver, 'Display', 'final');
149 [solution, fval, ex_temptflag, output] = solve(problem, 'options', options);
150
151 % Display the optimal schedule and makespan
152 if ex_temptflag == 1
153     disp('Optimal schedule:');
154     disp(solution.binvar);
155     disp('Makespan:');
156     disp(fval);
157 else
158     disp('Failed to find an optimal solution.');
```