

# Large-scale production in complex equipment manufacturing

Ziynag Shao  
Supervisor: Peng Guo

# Background

This project focuses on a provider of rail transit maintenance equipment. The company produces 38 kinds of large-scale rail transit operation and maintenance vehicles, each model involves more than 10,000 structural parts, and the materials, priorities, processes and other requirements of different parts are different. In the long-term implementation process of manual production scheduling in its structural branch plant, problems such as low efficiency and poor accuracy are gradually exposed.

- **importance of efficient scheduling**

- reduce production times,
- minimize costs,
- increase overall factory efficiency

- **Limitation of current solutions**

- traditional heuristic algorithms alostuggle to handle complex multi-objective optimization
- only partly optimal solutions can be found in some situation



# Aim

Design a scheduling algorithm and build up its mathematic model to efficiently solve large scale parallel machine scheduling problems. To tackle the challenges of large-scale parallel machine scheduling, particularly in high-end equipment manufacturing industries, this project proposes the design of an scheduling algorithm. The mathematical model underpinning this algorithm is aimed at optimizing several critical aspects of production processes: priority handling, material type management, and the processing numbers of parts. The objective is to minimize the total completion time of scheduled jobs while optimizing the allocation and utilization of resources.

# Assumptions

- Homogeneous parallel machine problem
- Single Operation Per Machine
- Consistent Machine Operation
- Negligible Transport Time
- Exclusion of Assembly Time

# Mathematic model

Stage1 model: to minimise the total completion time using simulated annealing

Parameters and variables	Descriptions
$N$	Set of machines
$J$	Set of parts
$R$	Set of operations
$s_{ijk}$	The start time of operation $i$ for part $j$ on machine $k$
$t_{ijk}$	The process time of operation $i$ for part $j$ on machine $k$
$a_{ijk}$	Binary variable (0 or 1). $a_{ijk} = 1$ if operation $i$ for part $j$ on machine $k$ is valid
$x_{ijk}$	Binary variable (0 or 1). $x_{ij}=1$ if part $J_i$ is processed before part $J_j$
$c_i$	The completion processing time of part $J_i$
$P_k$	The largest work hour of machine $k$
$M$	Very large number
$C_{max}$	Maximum completion time

$$C_{max} > c_i \quad \forall j \in J \quad (1)$$

$$\sum_{j \in J} a_{ijk} = 1 \quad \forall i \in R, \forall k \in N \quad (2)$$

$$(s_{i_1 j_1 k_1}, s_{i_1 j_1 k_1} + t_{i_1 j_1 k_1}) \cap ((s_{i_2 j_2 k_2}, s_{i_2 j_2 k_2} + t_{i_2 j_2 k_2})) = \emptyset \quad (3)$$

$$\sum_{j \in J} \sum_{i \in R} \sum_{k \in N} t_{ijk} a_{ijk} \leq P_k \quad (4)$$

# Mathematic model

## Stage 2 model: 3 objectives

1. Minimize the number of material changes during the processing of parts.
2. Prioritize the processing of parts with a higher priority
3. Prioritize the processing of parts with higher priority within the same project

Parameters and variables	Descriptions
$J$	Set of parts
$p_i$	Processing time of part $J_i$
$s_{ij}$	Binary variable (0 or 1). $s_{ij}=1$ if material of part $J_i$ is different from priority of part $J_j$
$y_i$	Priority of part $J_i$
$u_{ij}$	Binary variable (0 or 1). $u_{ij}=1$ if priority of part $J_i$ is lower than priority of part $J_j$
$q_{ij}$	Binary variable (0 or 1). $q_{ij}=1$ if process number of part $J_i$ is lower than process number of part $J_j$
$x_{ij}$	Binary variable (0 or 1). $x_{ij}=1$ if part $J_i$ is processed before part $J_j$
$c_{ij}$	The completion processing time of part $J_i$
$Z$	The optimize object
$M$	Very large number
$\omega$	Weight of each term in object function

$$minZ = \omega_1 \sum_{i \in J} \sum_{j \in J} x_{ij} s_{ij} + \omega_2 \sum_{i \in J} \sum_{j \in J} x_{ij} u_{ij} + \omega_3 \sum_{i \in J} \sum_{j \in J} x_{ij} q_{ij} \quad (5)$$

$$\sum_{i \in J} x_{ij} = 1 \quad \forall j \in J \quad (6)$$

$$\sum_{j \in J} x_{ji} = 1 \quad \forall i \in J \quad (7)$$

$$c_i \geq c_j + p_i + M(x_{ji} - 1) \quad \forall i \neq j; \quad i, j \in J \quad (8)$$

$$c_i > x_{0i} p_i \quad \forall i \in J \quad (9)$$

$$c_j \geq c_i + M(x_{ji} - 1) \quad \forall i \neq j; \quad i, j \in J \quad (10)$$

$$c_i \geq 0 \quad \forall i \in J \quad (11)$$

$$x_{ij} \in \{0,1\} \quad \forall i \in J, \forall j \in J \quad (12)$$

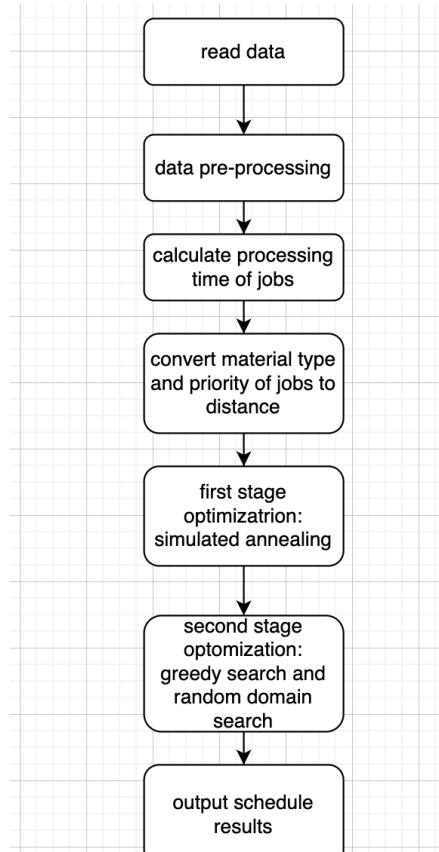
$$s_{ij} \in \{0,1\} \quad \forall i \in J, \forall j \in J \quad (13)$$

$$u_{ij} \in \{0,1\} \quad \forall i \in J, \forall j \in J \quad (14)$$

$$q_{ij} \in \{0,1\} \quad \forall i \in J, \forall j \in J \quad (15)$$

# Implementation of optimization algorithm part

Flow char of general algorithm

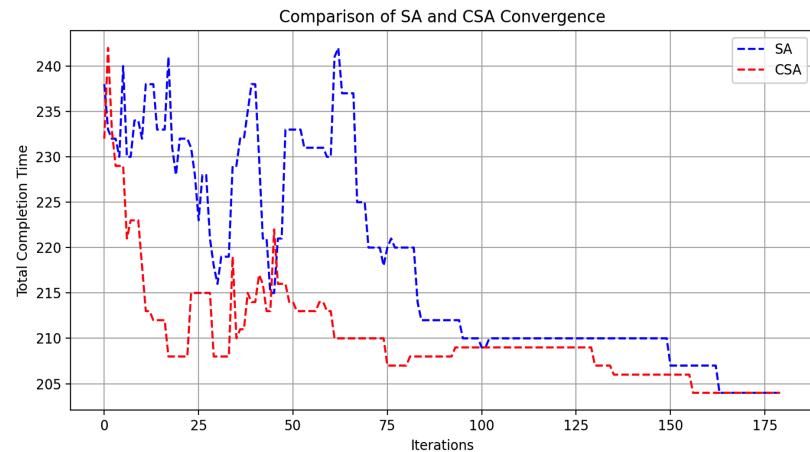


# Implementation of optimization algorithm part

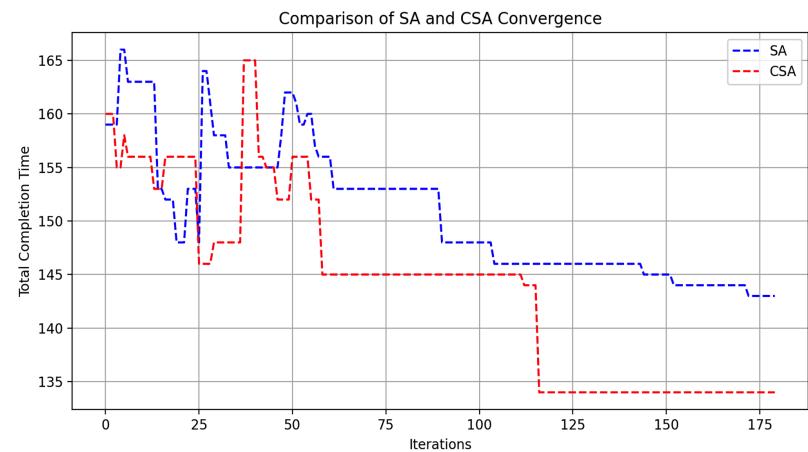
simulate annealing with chaotic operator

$$x_{n+1} = r \times x_n \times (1 - x_n)$$

20 job with random processing time between 1 and 100 on 5 machines

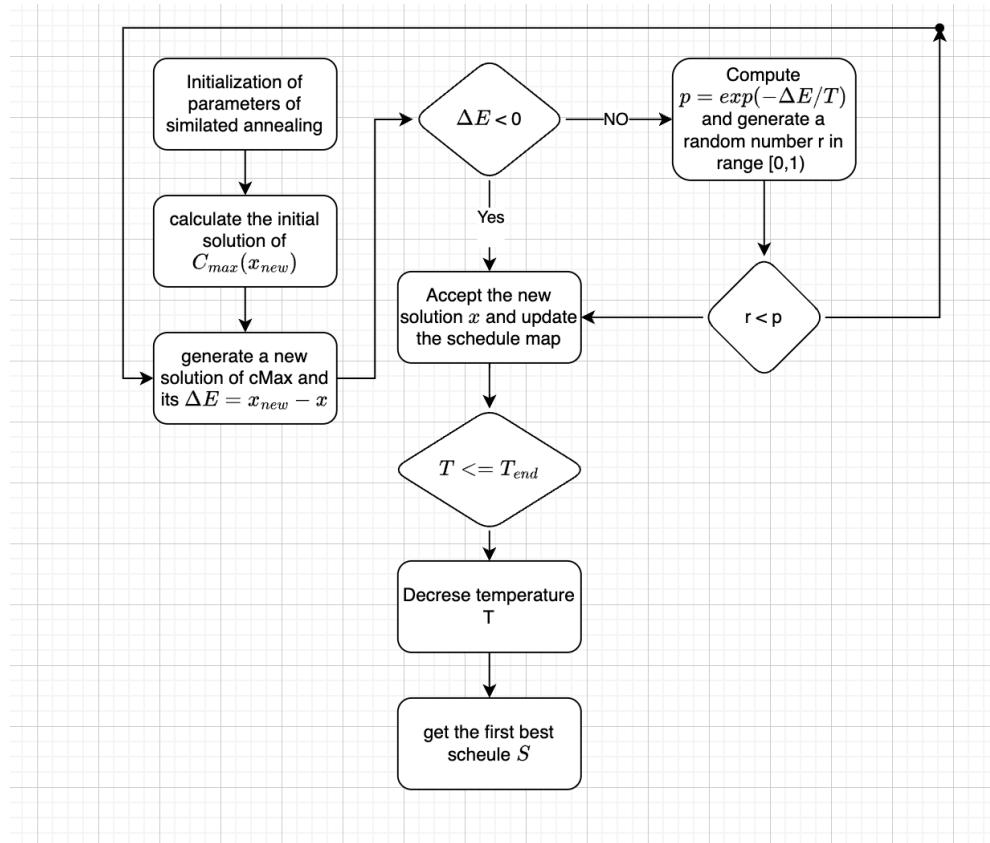


30 job with random processing time between 1 and 100 on 15 machines



# Implementation of optimization algorithm part

flow chart of simulate annealing



# Implementation of optimization algorithm part

## Code of simulate annealing with chaotic operator

```
def simulated_annealing(initial_temp, cooling_rate, max_iterations):
    current_solution = np.random.randint(0, num_machines, size=num_tasks)
    current_cmax, _ = calculate_cmax_and_schedule(current_solution)
    best_solution = current_solution.copy()
    best_cmax = current_cmax
    chaotic_var = np.random.rand()
    for i in range(max_iterations):
        temp = initial_temp * (cooling_rate ** i)
        chaotic_var = logistic_map(chaotic_var)
        new_solution = generate_new_solution(current_solution, chaotic_var)
        new_cmax, _ = calculate_cmax_and_schedule(new_solution)
        energy_diff = new_cmax - current_cmax
        if energy_diff < 0 or np.exp(-energy_diff / temp) > np.random.rand():
            current_solution = new_solution.copy()
            current_cmax = new_cmax

            if new_cmax < best_cmax:
                best_solution = new_solution.copy()
                best_cmax = new_cmax
    best_cmax, best_schedule = calculate_cmax_and_schedule(best_solution)
    return best_solution, best_cmax, best_schedule
```

# Implementation of optimization algorithm

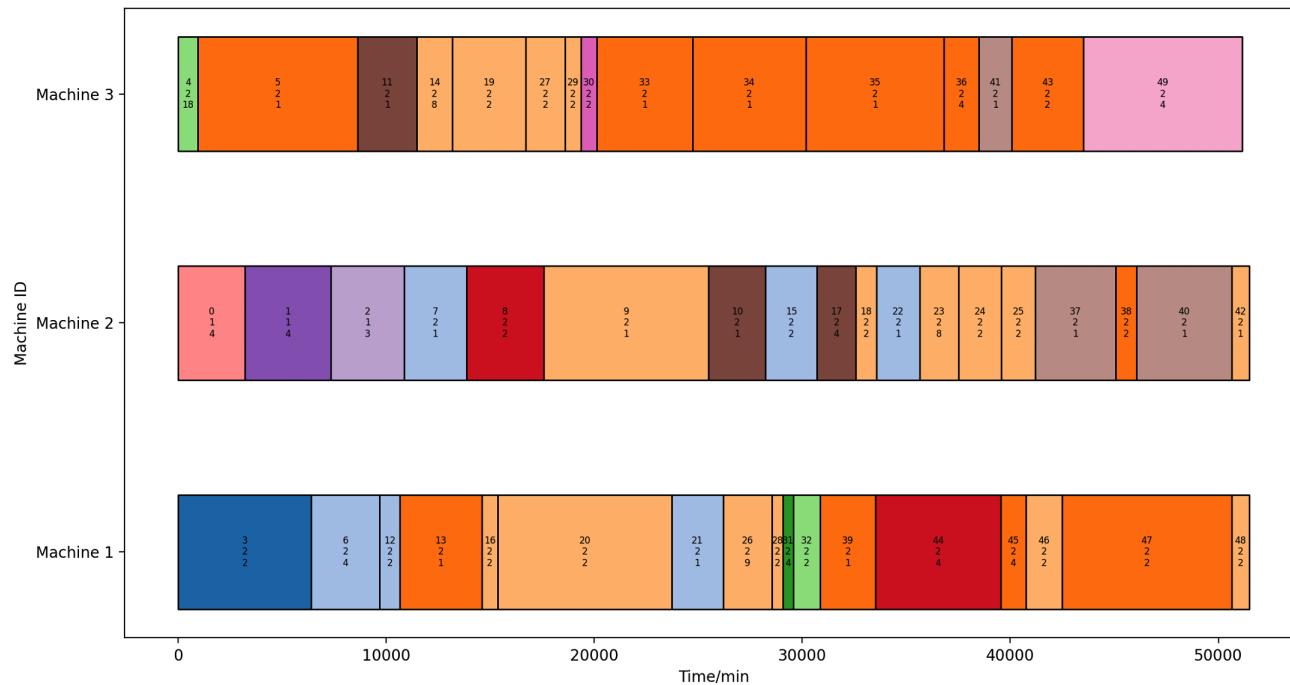
data used for test

Priority	Part code	Processing quantity	Part specification(thickness*width*length)	Material
1	20025203	4	45*455*80*	m15/Q235B
1	20024524	4	20*250*420*	m22/Q355B
1	20028407	3	38*218*218*	m28/Q355B
2	20005273	2	25*450*290*	m25/45
2	29001843	18	40*110*110*	mQ355B/20
2	23108228	1	40*280*350*	mQ355D/20
2	23108229	4	160*150*70*	mQ355D/16
2	23108230	1	16*475*200*	mQ355D/16
2	23108232	2	25*105*725*	mQ355D/25
2	23108235	1	54*272*275*	mQ355D/14
2	23108236	1	12*136*855*	mQ355D/12
2	23108237	1	12*136*885*	mQ355D/12
2	23108238	2	16*380*80*	mQ355D/16
2	23108239	1	20*265*380*	mQ355D/20
2	23108240	8	54*230*70*	mQ355D/14
2	23108241	2	16*110*717*	mQ355D/16
2	23108242	2	14*131*215*	mQ355D/14
2	23108243	4	12*110*717*	mQ355D/12
2	23108244	2	14*131*278*	mQ355D/14
2	23108245	2	54*131*254*	mQ355D/14
2	23108246	2	54*110*717*	mQ355D/14
2	23108247	1	16*110*717*	mQ355D/16
2	23108248	1	56*272*70*	mQ355D/16
2	23108249	8	64*210*70*	mQ355D/14
2	23108250	2	54*280*70*	mQ355D/14
2	23108251	2	54*217*70*	mQ355D/14
2	23108252	9	54*318*70*	mQ355D/14
2	23108253	2	54*256*70*	mQ355D/14
2	23114185	2	54*80*60*	mQ355D/14
2	23114186	2	54*120*60*	mQ355D/14
2	23114187	2	60*110*60*	mQ355D/10
2	23114201	4	22*60*60*	mQ355D/22

# Implementation of optimization algorithm part

## Gantt chart of schedule results of simulate annealing

top: id of the part, middle: priority, bottom: processing quantity



# Implementation of optimization algorithm part

preprocess of data to reduce solving time and computational load

$$d_{material} = \begin{cases} 0 & (material_i = material_j) \\ 1 & (material_i \neq material_j) \end{cases}$$

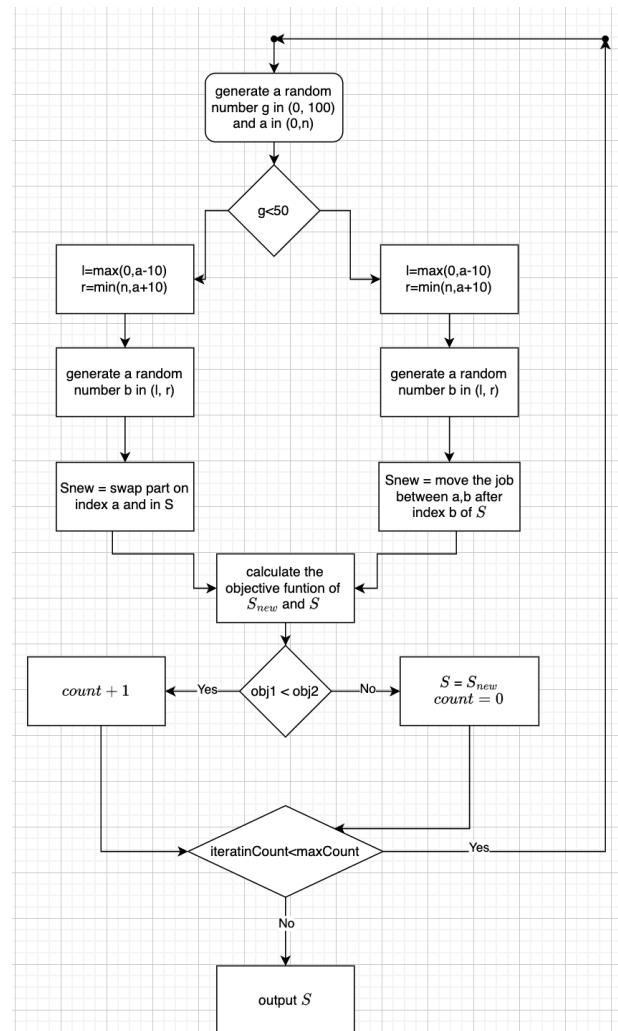
$$d_{priority} = \begin{cases} 0 & (priority_i = priority_j) \\ 1 & (priority_i < priority_j) \\ 2 & (priority_i > priority_j \text{ and } priority_i > priority_j + 1) \\ 3 & (priority_i > priority_j + 2) \end{cases}$$

code of local greedy search

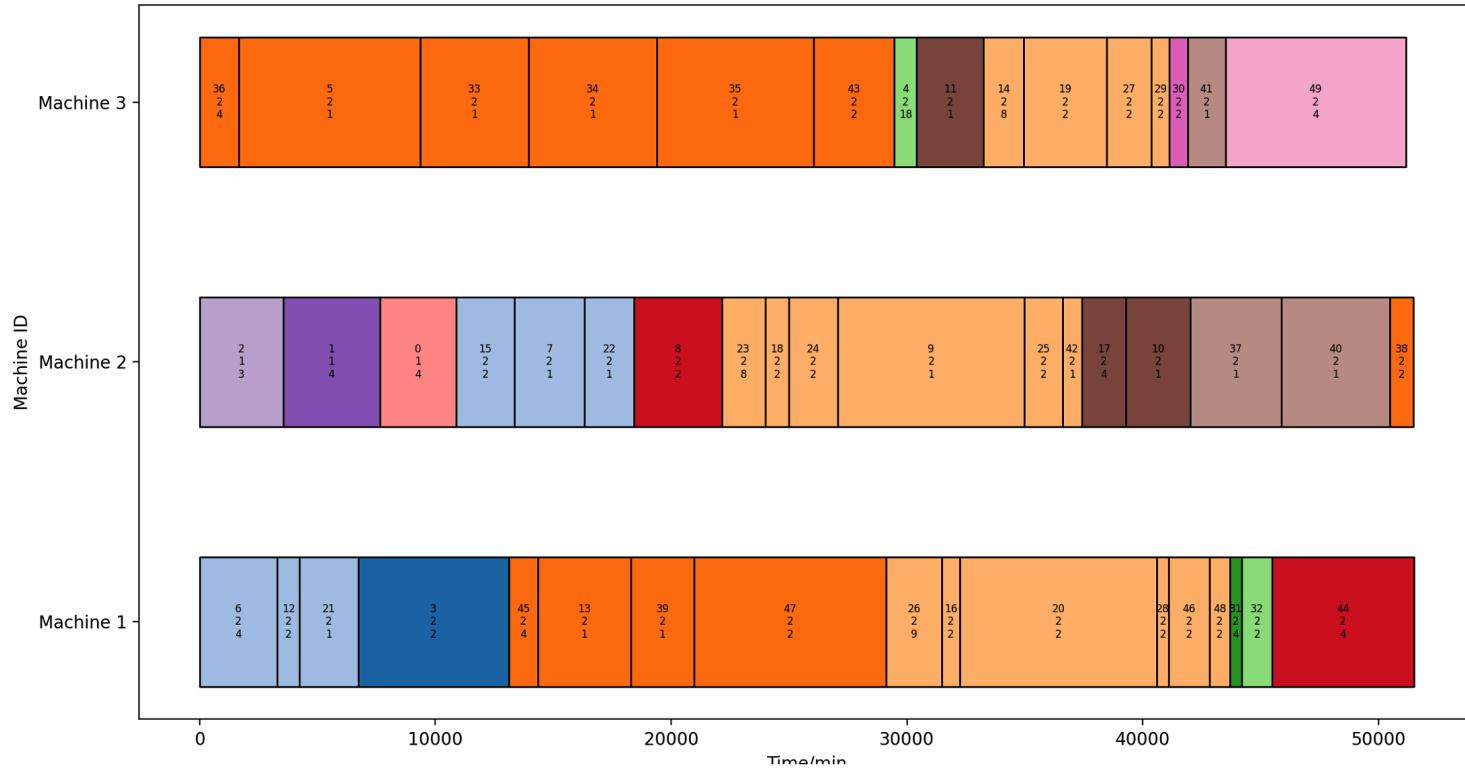
```
 1 def local_greedy_search(distance):
 2     l = len(distance)
 3     x1, x2 = np.unravel_index(np.argmin(distance, axis=None), distance.shape)
 4     sorted = [x1, x2]
 5     min_a = None
 6     min_b = None
 7     used = [False for _ in range(l)]
 8     used[x1] = True
 9     used[x2] = True
10
11    for i in range(l-2):
12        min_disa = 10
13        min_disb = 10
14        for j in range(l):
15            if used[j] or j == sorted[0]:
16                continue
17            if distance[j][sorted[0]] < min_disa:
18                min_disa = distance[j, sorted[0]]
19                min_a = j
20        for j in range(l):
21            if used[j] or sorted[-1] == j:
22                continue
23            if distance[sorted[-1]][j] < min_disb:
24                min_disb = distance[sorted[-1]][j]
25                min_b = j
26        if min_disa < min_disb:
27            sorted.insert(0, min_a)
28            used[min_a] = True
29        else:
30            sorted.append(min_b)
31            used[min_b] = True
32
33    return sorted
```

## implementation of random neighbor search

```
● ● ●
1 def random_neighborhood_search(sorted):
2     l = len(sorted)
3     re_sorted = np.zeros(l)
4     maxCount = 1000
5     count = 0
6     for i in range(maxCount):
7         g = randi(0, 100)
8         a = randi(0, l)
9         if g < 50:
10             s1 = 10
11             left = max(0, a-s1)
12             right = min(l, a + s1)
13             b = a
14             while (a == b):
15                 b = randi(left, right)
16             re_sorted = sorted
17             re_sorted[a], re_sorted[b] = re_sorted[b], re_sorted[a]
18         else:
19             s2 = 10
20             left = max(0, a-s2)
21             right = min(l, a+s2)
22             b = a
23             while (a == b):
24                 b = randi(left, right)
25             # remove the part from index a to b then insert them to the end
26             re_sorted = sorted
27             removed = re_sorted[a:b]
28             re_sorted = re_sorted[:a] + re_sorted[b:]
29             re_sorted += removed
30
31     # calculate minZ of sorted and resorted obj1 obj2
32     m1 = [material[j] for j in sorted]
33     p1 = [priority[j] for j in sorted]
34     q1 = [processing_quantity[j] for j in sorted]
35     m2 = [material[j] for j in re_sorted]
36     p2 = [priority[j] for j in re_sorted]
37     q2 = [processing_quantity[j] for j in re_sorted]
38     obj1 = obj(m1, p1, q1)
39     obj2 = obj(m2, p2, q2)
40     if (obj2 < obj1):
41         sorted = re_sorted,
42         count = 0
43     else:
44         count = count + 1
45     return sorted
```



## results of second stage optimization



# Summary

## Key Achievements

- Development of a two-stage optimization algorithm and reach the optimization requirement
- Integration of chaotic simulated annealing for improved convergence
- Implementation of local greedy and random neighborhood searches
- Performance evaluation with Gantt charts

## Future Work

- Incorporation of dynamic scheduling capabilities
- Development of a user-friendly software interface
- Make the algorithm more scaleable to suit different situations

END

thanks for listening