# 最优化大作业

徐帅 (学号: 1320221095)

**摘　要：** 本项目要解决的问题是流水车间调度问题，我采用了遗传算法和模拟退火算法两种优化方法进行求解，最终显示最优调度方案和总完工时间，实验结果相较不错。

**关键词：** 模拟退火算法 遗传算法

## 1　引言

这是一个流水车间调度的问题。在制造过程中，每个工件需要经过多道工序，每道工序需要在不同的机器上加工。每个机器一次只能处理一个工件，且一个工件不能同时在不同的机器上加工。而每个工件在每台机器上的加工时间是已知的。问题的目标是找到 n 个工件在每台机器上的加工顺序，使得总完工时间最小化。

**决策变量：**

$n$：工件的数量。

$m$：机器的数量。

$pij$：表示工件 $i$ 在机器 $j$ 上的加工时间。

**目标函数：**

最小化总完工时间。

min(Cmax)

**约束条件：**

1.每个工件必须按照特定的顺序在不同的机器上完成加工。

$$\forall i \in \{1,2,\ldots,n\}, \quad \sum_{j=1}^{m} p_{ij} = 常数$$

2.同一时间，每台机器只能处理一个工件。

$$\forall j \in \{1,2,\ldots,m\}, \quad \sum_{i=1}^{n} p_{ij} \leq 1$$

3. 加工时间 $pij$ 必须为非负实数。

优化调度问题的目标是找到一种工件的排列顺序，使得总完工时间最小化。这意味着找到一种排列，使得每个工件在每个机器上的加工时间都被最小化，并且所有工件的加工顺序符合其工艺流程的要求。

我的解决方案基于启发式优化算法，包括模拟退火算法和遗传算法。其关键步骤包括初始解生成、邻域解生成（模拟退火算法）或选择、交叉、变异（遗传算法）、新解接受与拒绝、更新状态（温度或种群）、迭代终止条件。模拟退火算法通过降温策略在解空间中搜索最优解，而遗传算法则模拟生物进化过程进行搜索。模拟退火算法适用于小规模问题，而遗传算法适用于大规模问题。

## 2 算法设计

### 2.1 模拟退火算法

**基本思想：** 模拟退火算法通过模拟固体退火过程中晶体结构的形成，利用随机接受次优解的策略逐步降低温度，从而在解空间中搜索最优解。

**关键步骤：**

1.初始解生成： 随机生成初始调度方案。

2.邻域解生成： 通过随机交换两个工件的位置来产生当前解的邻域解。

3.新解接受与拒绝：根据能量差和当前温度，以一定的概率接受比当前解差的邻域解作为新解。

4.温度更新： 使用指数衰减函数更新温度。

5.迭代终止条件：当达到指定的迭代次数或温度降至某个阈值时终止迭代。

实验效果： 模拟退火算法在工件调度问题上通常能够在较短的时间内找到较好的解，尤其适用于小规模问题。但对于大规模问题，可能需要更长的运行时间才能找到较优解。

在模拟退火算法中，通常的邻域方法是通过交换两个工件的位置来生成当前解的邻域解。例如，随机选择两个工件，并交换它们在工件序列中的位置，从而得到一个新的邻域解。这个交换操作可以反复进行，直到生成足够多的邻域解供算法选择。

加工完成时间计算方法是计算给定调度方案的总完工时间。通常情况下，工件的加工顺序和每个工件在每台机器上的加工时间已知。因此，可以根据工件的加工顺序，依次计算每个工件在每台机器上的加工完成时间，然后取其中的最大值作为总完工时间。

伪代码：

```
function simulated_annealing(machine_times, initial_schedule, initial_temperature, cooling_rate, max_iterations):
    current_schedule = initial_schedule
    current_temperature = initial_temperature
    best_schedule = initial_schedule
    best_schedule_time = evaluate_schedule(initial_schedule, machine_times)
    iteration = 0

    while iteration < max_iterations:
        new_schedule = generate_neighbour(current_schedule)  # 产生新解
        new_schedule_time = evaluate_schedule(new_schedule, machine_times)
        current_schedule_time = evaluate_schedule(current_schedule, machine_times)

        // 计算接受概率
        if new_schedule_time < current_schedule_time:
            accept_probability = 1.0
```

```
        else:
            delta_energy = current_schedule_time - new_schedule_time
            accept_probability = exp(delta_energy / current_temperature)

        // 接受或拒绝新解
        if random() < accept_probability:
            current_schedule = new_schedule

        // 更新最优解
        if new_schedule_time < best_schedule_time:
            best_schedule = new_schedule
            best_schedule_time = new_schedule_time

        // 更新温度
        current_temperature *= cooling_rate
        iteration += 1

    return best_schedule, best_schedule_time
```

**2.2**

**基本思想**：遗传算法是一种模拟生物进化过程的优化算法，通过自然选择、交叉和变异等操作来搜索最优解。在工件调度问题中，遗传算法通过维护一个种群，利用交叉和变异操作产生新的个体，然后通过适应度函数来评估个体的质量，并保留适应度较高的个体。

**关键步骤**：

1. 初始种群生成：随机生成初始种群，每个个体代表一个工件加工顺序。
2. 选择操作：根据适应度函数选择父代个体，通常采用轮盘赌算法。
3. 交叉操作：通过交叉操作产生新的个体，以增加种群的多样性。
4. 变异操作：对新个体进行变异操作，以引入新的基因。
5. 更新种群：将父代和后代合并，并根据适应度保留适应度较高的个体作为下一代种群。
6. 终止条件：当达到指定的迭代次数或满足停止条件时终止算法。

实验效果：遗传算法在工件调度问题上通常能够找到较好的全局解，尤其适用于大规模问题。通过调整参数和优化算子，可以进一步提高算法的性能，获得更好的解决方案。

在遗传算法中，邻域方法通常体现在交叉和变异操作中。交叉操作通过交换两个个体的部分基因来生成新的个体，从而扩展了搜索空间。变异操作则是对个体的基因进行随机变化，以引入新的解。这些操作都可以通过灵活设计来生成相邻的解。

伪代码：

```
function genetic_algorithm(machine_times, population_size, num_parents, num_offsprings, mutation_rate, num_iterations):
    population = generate_initial_population(population_size, num_jobs)

    for _ in range(num_iterations):
```

```
        fitness = [evaluate_schedule(individual, machine_times) for individual in population]

        parents = selection(population, fitness, num_parents)
        offsprings = crossover(parents, num_offsprings)
        offsprings = mutation(offsprings, mutation_rate)

        // 更新种群
        population = parents + offsprings

    // 找到最优解
    best_individual = min(population, key=lambda x: evaluate_schedule(x, machine_times))
    best_time = evaluate_schedule(best_individual, machine_times)

    return best_individual, best_time
```

# 3 实验

## 3.1 实验设置

实验环境：编程语言：python；
python 版本 3.11；
IDE：pycharm；
运行时间：模拟退火：0.23035573959350586 秒。
遗传：1.0250678062438965 秒。
实验参数：
模拟退火算法：
初始温度：根据平均加工时间计算得出。
冷却率：一般设置为 0.99。
最大迭代次数：一般设置为 1000。
遗传算法：
种群大小：50。
父代数量：20。
后代数量：30。
变异率：0.1。
迭代次数： 100。

## 3.2 模拟退火算法参数实验

```
initial_temperature=1000, stopping_temperature=10, cooling_rate=0.95, iterations=100
Instance 0:
Optimal Schedule: [7, 2, 4, 3, 0, 1, 10, 8, 6, 5, 9]
Total Completion Time: 7048
Instance 1:
Optimal Schedule: [3, 1, 0, 4, 2]
```

```
    Total Completion Time: 6297
    Instance 2:
    Optimal Schedule: [3, 7, 5, 4, 1, 6, 2, 0]
    Total Completion Time: 5072
    Instance 3:
    Optimal Schedule: [6, 4, 7, 9, 8, 3, 0, 5, 2, 1]
    Total Completion Time: 6676
    Instance 4:
    Optimal Schedule: [2, 7, 10, 5, 12, 8, 4, 13, 11, 0, 9, 6, 1, 14, 3]
    Total Completion Time: 9437
    Instance 5:
    Optimal Schedule: [3, 2, 1, 0, 5, 7, 6, 4]
    Total Completion Time: 7066
    Instance 6:
    Optimal Schedule: [1, 10, 14, 8, 7, 12, 11, 9, 0, 15, 17, 6, 3, 16, 2, 5, 13, 4]
    Total Completion Time: 1573
    Instance 7:
    Optimal Schedule: [4, 12, 17, 11, 13, 6, 3, 9, 8, 14, 1, 15, 16, 0, 10, 7, 2, 5]
    Total Completion Time: 2016
    Instance 8:
    Optimal Schedule: [4, 3, 14, 12, 2, 0, 9, 11, 13, 15, 1, 6, 8, 5, 7, 10]
    Total Completion Time: 1018
    Instance 9:
    Optimal Schedule: [15, 14, 12, 5, 1, 13, 9, 8, 10, 0, 4, 7, 11, 6, 3, 2]
    Total Completion Time: 2018
    Instance 10:
    Optimal Schedule: [33, 7, 38, 24, 36, 10, 0, 8, 30, 34, 4, 11, 19, 18, 21, 27, 29, 20, 3, 15,
14, 16, 28, 9, 5, 13, 37, 17, 2, 26, 39, 6, 23, 12, 1, 25, 31, 35, 22, 32]
    Total Completion Time: 2941

    initial_temperature=100, stopping_temperature=100, cooling_rate=0.95, iterations=100
    Instance 0:
    Optimal Schedule: [3, 8, 1, 4, 9, 5, 6, 2, 7, 0, 10]
    Total Completion Time: 9608
    Instance 1:
    Optimal Schedule: [0, 4, 3, 2, 1]
    Total Completion Time: 7508
    Instance 2:
    Optimal Schedule: [2, 6, 3, 0, 5, 1, 7, 4]
    Total Completion Time: 7267
    Instance 3:
    Optimal Schedule: [6, 3, 2, 9, 1, 4, 7, 8, 5, 0]
    Total Completion Time: 8639
```

```
    Instance 4:
    Optimal Schedule: [3, 4, 0, 6, 13, 9, 7, 14, 10, 5, 2, 12, 8, 11, 1]
    Total Completion Time: 12298
    Instance 5:
    Optimal Schedule: [2, 6, 5, 3, 0, 1, 7, 4]
    Total Completion Time: 8777
    Instance 6:
    Optimal Schedule: [0, 3, 5, 1, 10, 9, 8, 16, 14, 6, 15, 17, 2, 12, 11, 4, 7, 13]
    Total Completion Time: 1834
    Instance 7:
    Optimal Schedule: [16, 13, 11, 9, 7, 10, 1, 5, 17, 12, 6, 3, 0, 8, 4, 14, 15, 2]
    Total Completion Time: 2324
    Instance 8:
    Optimal Schedule: [6, 15, 4, 0, 9, 12, 8, 2, 14, 13, 5, 11, 7, 3, 10, 1]
    Total Completion Time: 1207
    Instance 9:
    Optimal Schedule: [11, 2, 13, 10, 5, 12, 0, 3, 9, 1, 14, 4, 8, 6, 7, 15]
    Total Completion Time: 2088
    Instance 10:
    Optimal Schedule: [12, 6, 10, 5, 3, 23, 30, 11, 1, 14, 25, 37, 24, 0, 31, 4, 17, 34, 32, 20,
27, 8, 21, 9, 35, 7, 36, 29, 15, 16, 39, 2, 33, 38, 18, 22, 28, 13, 19, 26]
    Total Completion Time: 3420

    initial_temperature=1000, stopping_temperature=10, cooling_rate=0.4, iterations=10
    Instance 0:
    Optimal Schedule: [2, 0, 3, 7, 6, 4, 9, 10, 8, 1, 5]
    Total Completion Time: 7699
    Instance 1:
    Optimal Schedule: [3, 1, 0, 4, 2]
    Total Completion Time: 6297
    Instance 2:
    Optimal Schedule: [7, 3, 0, 1, 4, 2, 5, 6]
    Total Completion Time: 5636
    Instance 3:
    Optimal Schedule: [7, 8, 0, 6, 4, 3, 5, 9, 1, 2]
    Total Completion Time: 7444
    Instance 4:
    Optimal Schedule: [2, 10, 13, 4, 12, 5, 11, 3, 8, 14, 0, 6, 7, 9, 1]
    Total Completion Time: 9913
    Instance 5:
    Optimal Schedule: [2, 0, 7, 1, 5, 6, 4, 3]
    Total Completion Time: 7122
    Instance 6:
```

Optimal Schedule: [0, 17, 12, 5, 3, 6, 9, 4, 7, 14, 13, 2, 16, 10, 15, 1, 8, 11]
Total Completion Time: 1730
Instance 7:
Optimal Schedule: [11, 15, 12, 4, 0, 6, 17, 14, 5, 3, 8, 13, 10, 2, 1, 9, 16, 7]
Total Completion Time: 2208
Instance 8:
Optimal Schedule: [14, 2, 12, 9, 0, 11, 10, 5, 3, 8, 4, 1, 13, 6, 15, 7]
Total Completion Time: 1076
Instance 9:
Optimal Schedule: [5, 9, 0, 12, 13, 3, 1, 14, 11, 10, 6, 8, 15, 7, 4, 2]
Total Completion Time: 2081
Instance 10:
Optimal Schedule: [30, 25, 24, 18, 5, 11, 8, 38, 3, 9, 32, 7, 35, 17, 28, 0, 21, 2, 39, 10, 4, 12, 33, 19, 31, 22, 29, 37, 16, 6, 34, 36, 1, 15, 27, 14, 20, 23, 13, 26]
Total Completion Time: 3169

initial_temperature（初始温度）：初始温度决定了算法开始时的搜索范围。较高的初始温度会使算法更容易接受次优解，从而增加了搜索空间，有利于避免陷入局部最优解。但是，如果初始温度设置得过高，可能会导致算法长时间在高温下搜索，浪费计算资源。

stopping_temperature（终止温度）：终止温度是算法停止迭代的条件之一。当温度降低到终止温度以下时，算法停止迭代。较低的终止温度可以更好地保证算法在搜索空间中找到全局最优解，但也可能导致算法提前停止，从而无法达到最优解。因此，终止温度的选择应该合理平衡算法的收敛速度和最终结果的质量。

cooling_rate（降温速率）：降温速率控制了温度的下降速度，即算法在搜索过程中温度的减少速率。较慢的降温速率意味着算法会在搜索空间中更多地进行随机搜索，从而更容易接受次优解，有利于全局搜索。

iterations（迭代次数）：迭代次数决定了算法在每个温度下进行的搜索次数。较多的迭代次数可以使算法在每个温度下进行更深入的搜索，从而有更大的可能性找到更优的解。但是，过多的迭代次数会增加算法的计算成本和时间开销。

### 3.3 遗传算法参数实验

population_size = 50；num_parents = 20；num_offsprings = 30；mutation_rate = 0.1 ；num_iterations = 100；

实例 0
最优调度方案: [0, 8, 4, 6, 8, 8, 0, 0, 8]
总完工时间: 6814
实例 1
最优调度方案: [9, 6, 10, 6, 6, 1, 9, 0, 14, 0, 8, 4, 6, 14, 1]
总完工时间: 7226
实例 2
最优调度方案: [2, 2, 0, 0, 6, 0, 0]
总完工时间: 5526
实例 3
最优调度方案: [4, 8, 8, 8, 8, 8, 0, 8, 8]

总完工时间: 7308
实例 4
最优调度方案: [2, 4, 2, 4, 6, 2, 4]
总完工时间: 6280
实例 5
最优调度方案: [2, 2, 4, 4, 4, 3, 2, 0, 2, 2, 2]
总完工时间: 9308
实例 6
最优调度方案: [14, 8, 0, 13, 1, 2, 6, 10, 6, 1, 14, 4, 0, 12, 8, 1, 14, 14, 8]
总完工时间: 1017
实例 7
最优调度方案: [27, 9, 12, 18, 0, 27, 28, 2, 23, 0, 4, 4, 9, 18, 13, 19, 14, 16, 20, 16, 0, 6, 21, 16, 25, 23, 13, 11, 18]
总完工时间: 1312
实例 8
最优调度方案: [6, 6, 6, 6, 1, 2, 6, 6, 2]
总完工时间: 755
实例 9
最优调度方案: [25, 9, 23, 12, 10, 4, 24, 6, 28, 17, 11, 7, 8, 21, 6, 2, 15, 27, 28, 22, 2, 2, 20, 24, 15, 13, 8, 13, 11]
总完工时间: 1152
实例 10
最优调度方案: [0, 1, 14, 14, 10, 11, 15, 0, 10, 3, 16, 1, 14, 15, 8, 14, 12, 0, 10]
总完工时间: 994
代码运行时间： 1.199134349822998 秒

population_size = 500; num_parents = 200; num_offsprings = 300; mutation_rate = 0.1 ; num_iterations = 100;
实例 0
最优调度方案: [8, 6, 8, 8, 8, 8, 8, 6, 6]
总完工时间: 8223
实例 1
最优调度方案: [2, 4, 6, 4, 4, 0, 6, 4, 8, 6, 6, 4, 2, 6, 6]
总完工时间: 10296
实例 2
最优调度方案: [0, 6, 0, 6, 6, 6, 0]
总完工时间: 6606
实例 3
最优调度方案: [2, 2, 2, 2, 2, 4, 2, 4, 0]
总完工时间: 6852
实例 4
最优调度方案: [2, 4, 2, 2, 2, 4, 6]
总完工时间: 6160

实例 5
最优调度方案: [6, 6, 0, 0, 0, 0, 0, 0, 0, 8, 0]
总完工时间: 9707
实例 6
最优调度方案: [2, 18, 5, 14, 8, 18, 8, 17, 4, 2, 16, 18, 6, 18, 18, 2, 8, 2, 4]
总完工时间: 1303
实例 7
最优调度方案: [2, 22, 21, 18, 22, 0, 28, 12, 28, 4, 2, 20, 13, 28, 4, 18, 28, 8, 6, 25, 2, 6, 22, 10, 4, 24, 25, 5, 28]
总完工时间: 1808
实例 8
最优调度方案: [2, 6, 6, 2, 0, 2, 2, 8, 8]
总完工时间: 719
实例 9
最优调度方案: [16, 2, 10, 16, 4, 28, 24, 13, 4, 2, 19, 25, 25, 2, 0, 4, 28, 27, 2, 26, 13, 16, 16, 10, 0, 20, 0, 14, 0]
总完工时间: 2046
实例 10
最优调度方案: [12, 18, 12, 10, 4, 2, 10, 4, 4, 2, 12, 18, 12, 2, 16, 9, 18, 2, 16]
总完工时间: 1373

population_size = 50; num_parents = 20; num_offsprings = 30; mutation_rate = 0.9 ; num_iterations = 1000;
实例 0
最优调度方案: [8, 8, 8, 8, 8, 8, 8, 8, 8]
总完工时间: 8991
实例 1
最优调度方案: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
总完工时间: 11835
实例 2
最优调度方案: [2, 2, 2, 2, 2, 2, 2]
总完工时间: 6272
实例 3
最优调度方案: [6, 6, 6, 6, 6, 6, 6, 6, 6]
总完工时间: 8064
实例 4
最优调度方案: [6, 6, 6, 6, 6, 6, 6]
总完工时间: 6993
实例 5
最优调度方案: [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
总完工时间: 10956
实例 6
最优调度方案: [8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]

总完工时间: 1900

实例 7

最优调度方案: [6, 12, 12, 6, 6, 6, 24, 24, 6, 24, 6, 6, 6, 24, 12, 24, 12, 24, 6, 24, 24, 6, 24, 6, 6, 6, 24, 24, 6]

总完工时间: 2432

实例 8

最优调度方案: [2, 2, 2, 2, 2, 2, 2, 2, 2]

总完工时间: 864

实例 9

最优调度方案: [0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 0, 0, 0, 24, 0, 0]

总完工时间: 2644

实例 10

最优调度方案: [10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10]

总完工时间: 1843

population_size（种群大小）：种群大小决定了每一代的候选解数量。较大的种群大小会增加搜索空间，有助于更广泛地探索解空间。但同时也会增加计算成本。

num_parents（父代数量）：父代数量决定了每一代中用于产生后代的候选解数量。较大的父代数量意味着更多的解可以作为父代参与繁衍后代，从而增加了搜索空间。但过多的父代数量可能会增加算法的计算成本。

num_offsprings（后代数量）：后代数量决定了每一代中生成的新候选解数量。较多的后代数量可以增加搜索空间，有助于更广泛地探索解空间。但过多的后代数量可能会增加算法的计算成本。

mutation_rate（变异率）：变异率决定了在产生后代过程中进行变异的概率。较高的变异率意味着更多的解可能发生变化，有助于增加搜索空间。但过高的变异率可能导致搜索过于随机，降低了算法的搜索效率。

num_iterations（迭代次数）：迭代次数决定了算法运行的总体时长。较多的迭代次数可以增加算法的搜索深度，有助于更好地探索解空间，但也会增加计算成本。

### 3.4 算法对比

**模拟退火：**

Instance 0:

Optimal Schedule: [7, 3, 0, 4, 2, 6, 10, 8, 1, 5, 9]

Total Completion Time: 7051

Instance 1:

Optimal Schedule: [3, 1, 0, 4, 2]

Total Completion Time: 6297

Instance 2:

Optimal Schedule: [3, 7, 1, 0, 4, 5, 6, 2]

Total Completion Time: 5077

Instance 3:

Optimal Schedule: [6, 4, 7, 9, 5, 3, 8, 0, 2, 1]

Total Completion Time: 6676

Instance 4:
Optimal Schedule: [2, 5, 11, 12, 8, 13, 9, 14, 4, 6, 10, 7, 0, 1, 3]
Total Completion Time: 9448
Instance 5:
Optimal Schedule: [3, 0, 2, 7, 1, 4, 6, 5]
Total Completion Time: 7059
Instance 6:
Optimal Schedule: [14, 12, 11, 2, 10, 3, 1, 9, 13, 6, 7, 8, 17, 4, 0, 16, 15, 5]
Total Completion Time: 1502
Instance 7:
Optimal Schedule: [10, 6, 4, 17, 12, 16, 1, 9, 13, 8, 11, 3, 0, 14, 15, 7, 2, 5]
Total Completion Time: 2013
Instance 8:
Optimal Schedule: [4, 3, 2, 0, 12, 11, 7, 14, 6, 8, 13, 1, 9, 15, 5, 10]
Total Completion Time: 975
Instance 9:
Optimal Schedule: [11, 15, 10, 9, 14, 4, 7, 2, 1, 12, 0, 8, 13, 6, 5, 3]
Total Completion Time: 1952
Instance 10:
Optimal Schedule: [36, 38, 26, 9, 34, 8, 33, 12, 37, 0, 10, 4, 27, 35, 6, 11, 24, 39, 21, 7, 30, 2, 28, 5, 17, 23, 14, 18, 1, 15, 13, 16, 32, 29, 3, 19, 20, 31, 25, 22]
Total Completion Time: 2974

遗传算法：

实例 0
最优调度方案: [2, 8, 0, 6, 8, 8, 8, 6, 6]
总完工时间: 7081
实例 1
最优调度方案: [2, 8, 6, 8, 7, 12, 6, 10, 10, 8, 0, 6, 0, 0, 6]
总完工时间: 7864
实例 2
最优调度方案: [0, 6, 6, 6, 6, 6, 0]
总完工时间: 6728
实例 3
最优调度方案: [6, 8, 6, 6, 6, 6, 2, 6, 6]
总完工时间: 7360
实例 4
最优调度方案: [4, 6, 6, 6, 2, 6, 2]
总完工时间: 6328
实例 5
最优调度方案: [0, 3, 8, 6, 4, 8, 4, 0, 0, 4, 4]
总完工时间: 6997
实例 6

最优调度方案：[10, 17, 2, 1, 16, 17, 10, 2, 16, 4, 14, 6, 10, 2, 16, 12, 10, 10, 2]
总完工时间：1303
实例 7
最优调度方案：[25, 0, 16, 26, 14, 18, 6, 25, 21, 3, 17, 1, 12, 6, 23, 6, 6, 16, 28, 1, 25, 12, 12, 18, 28, 18, 17, 28, 25]
总完工时间：1394
实例 8
最优调度方案：[2, 4, 4, 2, 2, 2, 2, 0, 4]
总完工时间：734
实例 9
最优调度方案：[7, 28, 24, 28, 12, 10, 26, 24, 6, 18, 7, 24, 27, 22, 22, 25, 27, 7, 14, 17, 24, 3, 21, 18, 21, 14, 4, 18, 24]
总完工时间：1364
实例 10
最优调度方案：[10, 18, 2, 6, 14, 2, 11, 4, 6, 11, 2, 11, 1, 2, 10, 10, 1, 0, 2]
总完工时间：1118

对于一些实例（例如实例 0、1、2、3、5），模拟退火算法给出了相对较好的解，总完工时间相对较低。然而，对于一些更大规模的实例（例如实例 4），模拟退火算法的表现并不理想，总完工时间较高。反之遗传算法效果较好。

模拟退火算法通常适用于解空间较小、局部搜索空间较为复杂的问题，而遗传算法适用于解空间较大、全局搜索空间较为复杂的问题。

### 3.5 算法改进

详情请在 github 搜索 optimization-course-task
https://github.com/Xuanshen818/optimization-course-task.git
目前还未公开还在私密，我会在交完作业之后公开的。

## 4 总结

本文主要介绍了两种不同的算法（模拟退火算法和遗传算法）来解决工件调度问题，并通过具体的例子对它们的性能进行了比较。

在比较两种算法的性能时，发现它们在不同的实例上表现不同。模拟退火算法在一些较小规模的实例上表现较好，但对于更大规模的实例可能表现不佳；而遗传算法则在某些实例上能够给出更好的解，但也有一些实例表现较差。

对于模拟退火算法，初始温度、终止温度、冷却速率和迭代次数等参数的选择可能会影响算法的性能，需要进行更精细的调节。对于遗传算法，种群大小、父代数量、后代数量、变异率和迭代次数等参数的选择可能会影响算法的性能，需要进行更合理的设置。

可以尝试使用其他优化算法来求解工件调度问题，如禁忌搜索、粒子群算法等，以便综合比较不同算法的性能。需要进一步研究工件调度问题的特点和约束条件，针对不同的情况设计更加有效的算法和策略。

结合启发式方法和元启发式方法，设计更加高效的混合优化算法，如混合模拟退火和遗传算法，以充分利用各自的优点，提高求解效率和质量。

**参考文献**:

[1] Kirkpatrick, Scott, Gelatt Jr, Charles D., and Vecchi, Mario P. "Optimization by Simulated Annealing." Science 220.4598 (1983): 671-680.

[2] Pinedo, Michael L. "Scheduling: Theory, Algorithms, and Systems." Springer Science & Business Media, 2012.