[FLOW-SHOP 流水线调度问题的贪心算法和元启发式算法研究]

■■■ (学号: ■■■■■■■■■■)

摘　要: FLOW-SHOP 问题是由最佳调度问题引申出的一个 NP-HARD 问题，这意味着即便对于小规模问题，目前最先进的算法仍然无法求解得到最优解。FLOW-SHOP 问题描述了 n 个工件要在 m 台机器上加工，求取在遵循给定工序和“一台机器在同一时间只能加工一个零件”的约束条件下加工所需的最短时间和对应的工件加工顺序的问题。本文针对 FLOW-SHOP 问题提出了三种解决方案。方案一仅仅使用了爬山算法寻找问题较优解，方案二使用了贪心算法中的爬山算法，结合模拟退火算法这种元启发式算法寻找较优解，方案三使用了遗传算法寻找较优解，并尝试结合遗传算法和爬山算法对遗传算法进行优化。最终运行结果显示，爬山算法能够在一定程度上寻找到问题的较优解。相较于爬山算法，基于模拟退火算法的爬山算法有更大概率寻找到问题的较优解，但是时间运行代价更高。遗传算法也能寻找到问题的较优解，并且相对于基于模拟退火算法的爬山算法，其运行的时间代价相近，然而随着工件数和机器数的增多，遗传算法得到的结果略差于方案二。加入爬山算法优化的遗传算法能够更快地收敛，得到比单独使用遗传算法更优的结果，但是运行时间开销大大增加。

关键词: 流水线调度问题，贪心，元启发，爬山，模拟退火，遗传

目录

[FLOW-SHOP 流水线调度问题的贪心算法和元启发式算法研究]....1
1 引言....2
2 算法设计....2
2.1 数据预处理....2
2.2 FLOW-SHOP 时间计算....2
2.3 爬山算法....3
2.3.1 算法简介....3
2.3.2 算法设计....3
2.3.3 算法复杂度分析....4
2.4 基于模拟退火算法的爬山算法....5
2.4.1 算法简介....5
2.4.2 算法设计....5
2.4.3 算法复杂度分析....7
2.5 遗传算法....7
2.5.1 算法介绍....7
2.5.2 算法设计....7
2.5.3 算法改进....9
2.5.4 算法复杂度分析....10
3 实验....10
3.1 实验设置....10
3.2 实验结果 [可包含参数实验、对比实验等，可以组织在不同的小节中]....11
3.2.1 爬山算法....11

3.2.2 基于模拟退火算法的爬山算法 12
3.2.3 遗传算法 14
3.2.4 内嵌爬山算法的改进遗传算法 16
3.2.5 多组参数实验 18
3.2.6 结果综合 所有样例结果结果提交 20
4 总结 21
参考文献: 22

1 引言

现代工业生产环节包括了很多环节，不同生产环节又包括了不同生产节点，它们间的关系紧密复杂，对不同工件的生产顺序的规划安排密切影响着工业生产的整体效率和成本开销，具有重要的理论价值和应用价值。FLOW-SHOP 问题是由最佳调度问题引申出的一个 NP-HARD 问题，是工业生产环节中对不同工件的生产顺序的规划安排问题的简化。

FLOW-SHOP 流水线调度问题可以描述为：n 个工件要在 m 台机器上加工，每个工件需要经过 m 道工序，每道工序要求不同的机器加工。每个工件的加工顺序相同，都是从第一台机器至最后一台机器，每个工件在每台机器上只加工一次。每个机器同时只能加工一个工件，一个工件不能同时在不同的机器上加工，每个工件在不同机器上的加工时间以表格的形式给定。问题的目标是求 n 个工件的加工顺序，使总完工时间达到最小。

本文采用一种贪心算法：爬山算法，尝试寻找 FLOW-SHOP 问题的较优解。同时在贪心算法的基础上，应用了一种元启发式算法：模拟退火算法，跳出局部最优解，寻找更优解。最后，本文使用了另一种元启发式算法——遗传算法尝试解决 FLOW-SHOP 问题，并尝试使用爬山算法对已有的遗传算法进行了优化，加快了原有遗传算法的收敛速度，得到了较好结果，然而运行时间开销大大增加。

本文后续部分组织如下。第 2 节详细陈述使用的方法，第 3 节报告实验结果，第 4 节对讨论本文的方法并总结全文。

2 算法设计

2.1 数据预处理

数据预处理阶段，我们通过按行读入 flowshop-test-10-student.txt 文件的内容，使用 python 中的.strip()和.split()方法将文本文件按行分割，按照原有排布顺序将每一行作为一个 str 类型元素组合成一个 list（本次实验中名为“sents”），方便后续对各个 instance 的读取和处理。接下来通过一次对“sents”的遍历获得每个例子的工件数量和机器数量两个基本信息，并使用每个例子的基本信息再次遍历“sents”，通过 lst = list(map(int, temp.split()))和 lst = lst[1::2]方法将“0 375 1 12 2 142 3 245”转化成可以遍历的 list：[375, 12, 142, 245]，并将不同 instance 的信息以三维 list 的形式存放在 input_data 中（如 input_data[0]即存储了第一个例子的信息，其中为一个二维 list，input_data[0][2][4]为第 3 个零件在第 5 台机器上的加工时间），数据处理工作最终将.txt 文本文件转化为了可遍历的三维 list，方便了后续各种算法的数据读取和数据处理。

2.2 FLOW-SHOP时间计算

定义 flow_shop 函数，输入由数据预处理得到的每个样例的二维 list 和零件处理顺序表（如[0,1,2]（表示有三个零件，处理顺序为 1 号零件->2 号零件->3 号零件），[2,0,1]（表示有三个零件，处理顺序为 3 号零件->1 号零件->2 号零件）），目标是得到该样例按照零件处理顺序表处理零件的加工总时间。

函数定义如下：

首先计算出工件数量 n 和机器数量 m，然后初始化一个二维列表 time 来存储每个工件在每台机器上的完成时间。

接下来，使用动态规划的方法来计算每个工件在每台机器上的完成时间。对于第一台机器，完成时间就是前面所有工件的加工时间之和。对于其他机器，完成时间是前一台机器完成时间和同一台机器前一个工件完成时间的最大值加上当前工件在当前机器上的加工时间。

最后，代码返回最后一个工件在最后一台机器上的完成时间，即所有工件在所有机器上的总完成时间。并返回该流水线按照零件处理顺序表处理零件的加工总时间。

2.3 爬山算法

2.3.1 算法简介

爬山算法（Hill Climbing,HC）是一种贪心算法，它将寻找问题最优解的过程比作现实生活中的爬山过程，山脉的“最高峰”即为我们期望得到的“最优解”。算法开始时，选定某个点（比如“山脚”）开始爬山算法，在临近的点形成的“临近点集”中寻找比当前点更高（更优）的位置。在爬山算法的不同分支中，“首选爬山法”通过选取遍历“临近点集”中找到的第一个高于当前所在位置点的位置点，并使用其更新当前所在位置点，即通过一步爬山“到达了”临近位置的“更高点”;“最陡爬山法”通过选取“临近点集”中的最高点，并使用其更新当前所在位置点，即通过一步爬山“到达了”临近位置的“最高点”。爬山法一般情况下通过判断“临近点集”中是否存在比当前所在位置点高的点来决定爬山的终止，若“临近点集”中所有点都等于或低于当前所在位置点高度，则认为找到了“山顶”，爬山算法结束。

正因为爬山算法的“通过判断‘临近点集’中是否存在比当前所在位置点高的点来决定爬山的终止”的终止特性，爬山算法容易陷入“局部最优解”。

2.3.2 算法设计

我们选取“最陡爬山法”进行算法设计，即每步爬山通过选取“临近点集”中的最高点更新当前位置点。

将零件按照 flowshop-test-10-student.txt 中的初始顺序进行编号，每个用例的第一行为 1 号零件，第二行为 2 号零件，因此类推。算法的处理对象是零件的处理顺序表，如[0,1,2]（表示有三个零件，处理顺序为 1 号零件->2 号零件->3 号零件），[2,0,1]（表示有三个零件，处理顺序为 3 号零件->1 号零件->2 号零件）。

算法设计如下：首先生成初始处理顺序：从 1 号零件开始依次处理到用例中的最后一号零件。在爬山算法的每步爬山操作中，首先由当前解生成临近解空间。生成的临近解空间的大小为 10，生成的方法为随机选择处理顺序表中的两个零件并交换位置，即在当前解中随机选择两个零件并交换加工顺序。接下来分别计算解空间中每个方法的加工总用时，并选择解空间中用时最短的方法作为 new_solution 替换现有解 current_solution。当临近解空间中不存在加工总用时短于现有解加工总用时的时候，爬山算法结束，输出当前得到的解作为优解。

算法伪代码如下：

```
定义爬山算法单步爬山函数 climbing_algorithm_step(输入用例矩阵, 当前解)
    临近解空间 = []
    临近解空间中解的时间 = []
    for i in range(10):
        临近解空间[i] = 当前解
        顺序交换函数(临近解空间[i]) # 随机选择两个零件进行加工顺序位置交换，生成临近解空间
        临近解空间中解的时间[i] = flow_shop(输入用例矩阵, 新解[i])
    最短用时 = min(临近解空间中解的时间)
    最短用时对应索引 = 临近解空间中解的时间.index(最短用时)
    新解 = 新解[最短用时对应索引] #使用最陡爬山法
    返回 新解
```

定义爬山算法函数 climbing_algorithm(输入用例矩阵)

最优时间 = [] #记录了当前最优时间的变化规律，方便后续图表绘制和可视化

当前解 = [i for i in range(len(输入用例矩阵))]

最优时间.append(flow_shop(输入用例矩阵，当前解))

新解 = climbing_algorithm_step(输入用例矩阵, 当前解)

当 flow_shop(输入用例矩阵, 新解) < flow_shop(输入用例矩阵, 当前解):

当前解 = 新解

最优时间.添加(flow_shop(输入用例矩阵, 当前解))

新解 = climbing_algorithm_step(输入用例矩阵, 当前解)

最优解 = 当前解

返回 最优时间, 最优解

算法流程图如下：

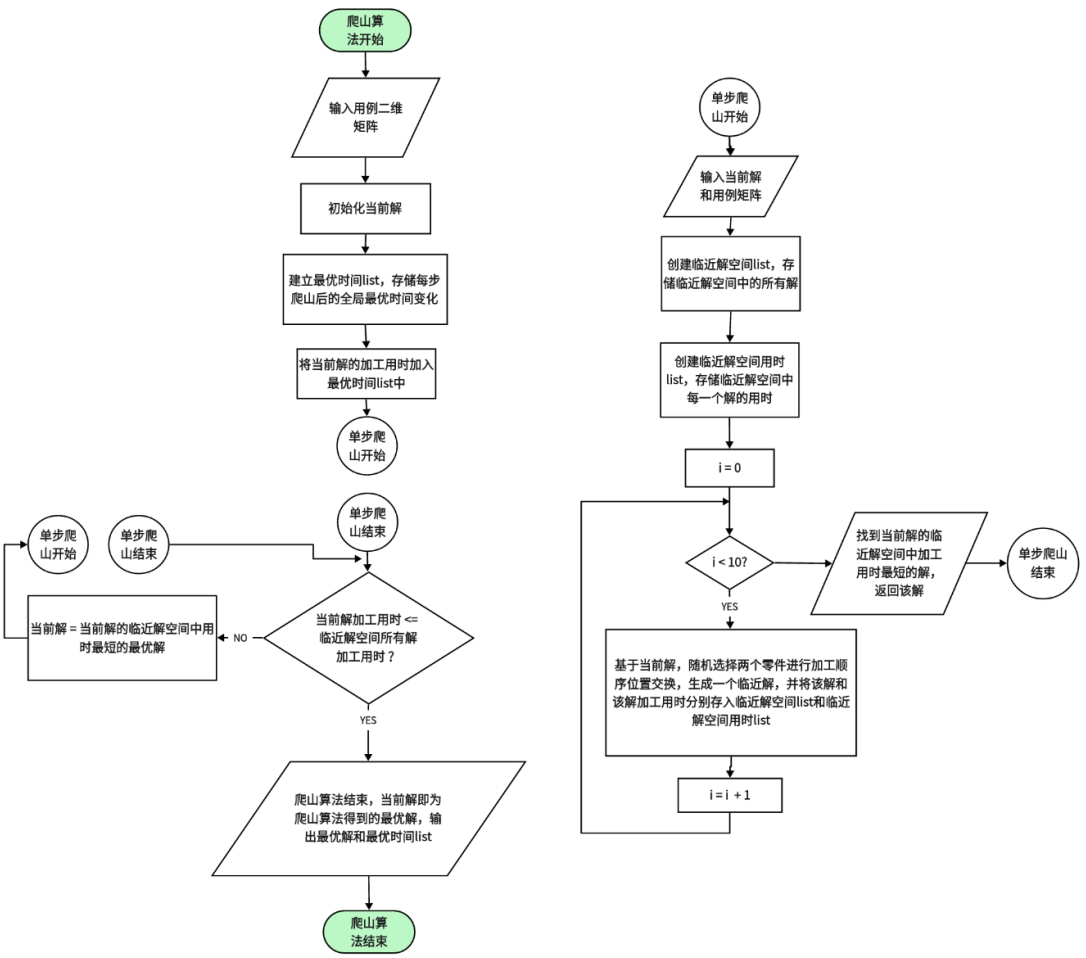


图 1 爬山算法流程图

2.3.3 算法复杂度分析

flow-shop 函数的时间复杂度为 O(nm)，其中 n 为工件数量，m 为机器数量。空间复杂度为 O(nm)。climbing_algorithm_step 函数的时间复杂度为 O(10nm)，其中 n 为工件数量，m 为机器数量。空间复杂度为

O(10n)。

时间复杂度方面，这个函数的时间复杂度取决于循环次数和 flow_shop 函数的时间复杂度。由于循环次数取决于 input 的大小和 climbing_algorithm_step 函数生成新解决方案的效果，因此无法准确估计时间复杂度。但是，由于每次循环中都会调用 flow_shop 函数，因此这个函数的时间复杂度至少与 flow_shop 函数的时间复杂度成正比，即为 O(nm)。

空间复杂度方面，这个函数中使用了几个列表来存储解决方案和完成时间，因此空间复杂度与 input 的大小成正比，即为 O(nm)。

2.4 基于模拟退火算法的爬山算法

2.4.1 算法简介

模拟退火算法（SA）是 80 年代初发展起来的一种随机性组合优化方法。它模拟高温金属降温的热力学过程，并广泛应用于组合优化问题。[2]基于模拟退火算法的爬山算法是对爬山算法的优化，由于爬山算法只能寻找到局部最优解，这使得其最终的输出结果往往与真实最优解大相径庭，而基于模拟退火算法的爬山算法则能使爬山算法在寻找到局部最优解后仍然能以一定概率跳出局部最优解，接受一个相对更劣的解，进而进一步深入搜索新的优解。

算法进行过程中，当前温度决定着接受劣质解的概率。温度越高，接受劣质解决方案的概率也越大。随着温度逐渐降低，搜索过程中接受劣质解决方案的概率也逐渐减少。初始的高温实质上是增加了算法的随机性，使得爬山算法能够寻找到实际最优解的概率大大提升。

基于模拟退火算法的爬山算法在算法主体部分和 2.3 节中介绍的爬山算法相似，但使用模拟退火算法重新定义了算法的结束条件。决定算法停止的条件不再是临近解集和当前解，而是模拟退火算法的温度，退火温度随着爬山算法的进行逐渐降低，当温度低于设定最低温度值时，算法结束，并输出当前得到的最优解作为基于模拟退火算法的爬山算法的优解。

2.4.2 算法设计

算法的处理对象是零件的处理顺序表，如[0,1,2]（表示有三个零件，处理顺序为 1 号零件->2 号零件->3 号零件），[2,0,1]（表示有三个零件，处理顺序为 3 号零件->1 号零件->2 号零件）。

先生成初始当前解，初始处理顺序：从 1 号零件开始依次处理到用例中的最后一号零件，并将当前解作为当前最优解，建立当前解时间 list 和当前最优解时间 list 用于存储每一次模拟退火步骤得到的当前解时间和当前最优解时间，便于时间变化图表绘制和可视化工作。接下来进入模拟退火算法，设定退火温度初始值，温度最低值和冷却速率。从初始温度开始，每一轮退火阶段都将调用单步爬山函数，将当前解的临近解集中的最优解作为新解，并新解与当前解进行比较。若通过单步爬山得到的新解用时小于当前解用时，则用新解更新当前解。若新解用时大于等于当前解用时，则以 e^((当前解用时 – 新解用时)/ 当前温度 的概率接受新解并使用新解更新当前解，若不在该概率区间则不接受新解，当前解不变。

当前最优解指的是当前整个算法得到的当前全局最优解，在每轮退火操作结束前，检查当前解用时是否小于当前最优解用时。如果当前解用时小于当前最优解用时，说明当前解才是当前最优解，并使用当前解更新当前最优解，否则当前最优解不变化。

每轮退火操作结束后，当前温度按照指定的冷却速率下降，新的当前温度 = 原当前温度*（1 – 冷却速率）。

算法伪代码如下：

```
函数 simulated_annealing(input, temp=1000, cooling_rate=0.00075)
    初始化 当前解 和 当前最优解
    初始化 当前解用时变化列表 和 当前最优解用时变化列表
    while temp > 1 进行退火循环：
        使用单步爬山函数生成新解决方案
```

计算当前解用时和新解用时

if (新解用时 < 当前解用时)

接受新解决方案

else

以 e^((当前解用时 – 新解用时) / 当前温度 的概率接受新解决方案

end if

更新 当前最优解

更新 当前解用时变化列表 和 当前最优解用时变化列表

降低温度

end while

返回 当前解用时变化列表 和 当前最优解用时变化列表 和 当前最优解

算法结束

算法流程图如下：

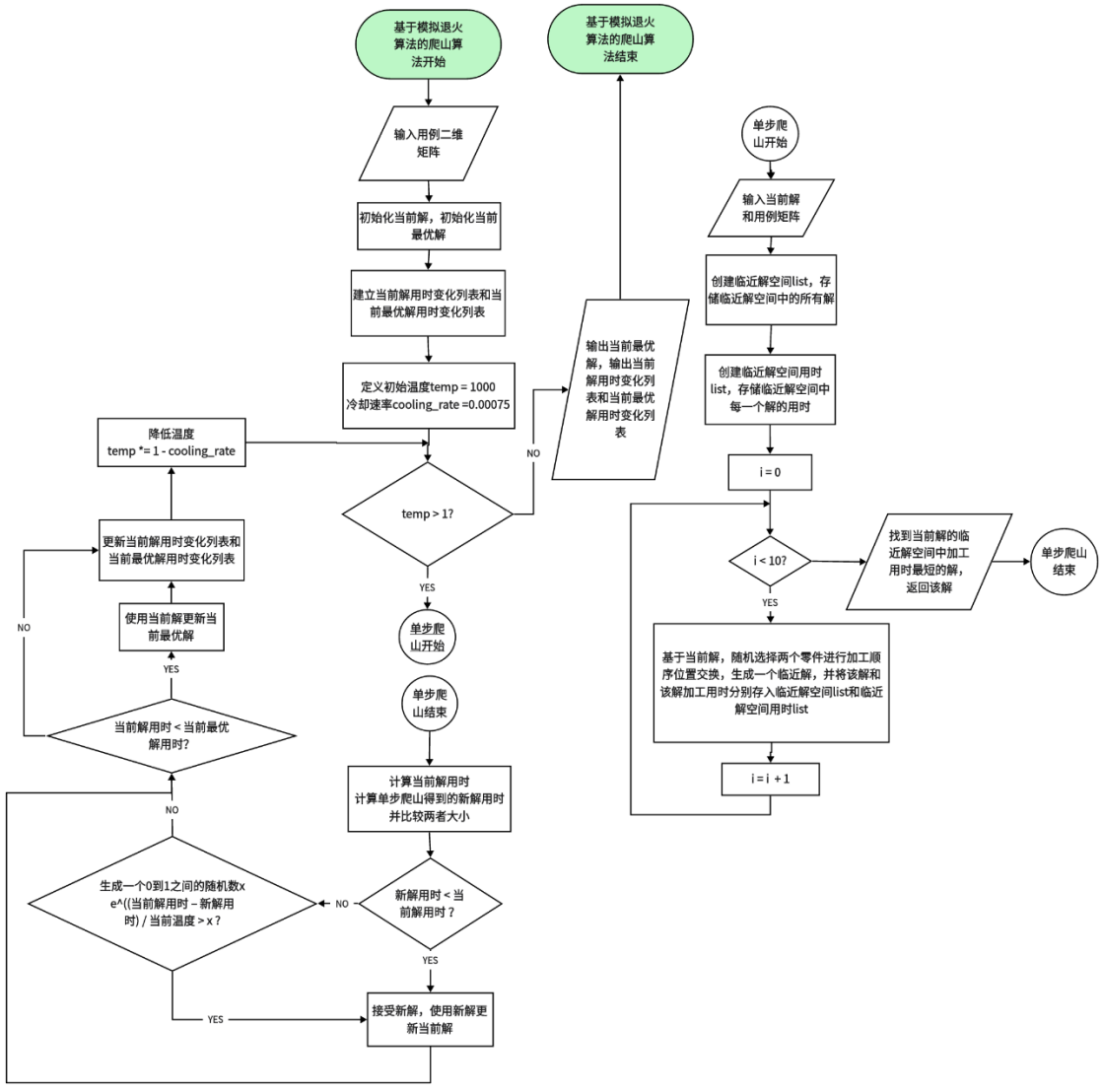


图 2 基于模拟退火算法的爬山算法流程图

2.4.3 算法复杂度分析

函数首先初始化当前解决方案和最佳解决方案，然后进入一个循环，直到温度降到 1 以下。在每次循环中，函数使用 climbing_algorithm_step 函数生成一个新的解决方案，并计算新解决方案和当前解决方案的能量。如果新解决方案的能量小于当前解决方案的能量，则接受新解决方案；否则，根据概率接受新解决方案。最后，函数返回当前时间变量、最佳时间变量和最佳解决方案。

该算法时间复杂度取决于循环次数和 flow_shop 函数的时间复杂度。由于循环次数取决于 temp 和 cooling_rate 的值，因此无法准确估计时间复杂度。但是，由于每次循环中都会调用 flow_shop 函数，因此这个函数的时间复杂度至少与 flow_shop 函数的时间复杂度成正比，即 O(nm)。

空间复杂度方面，模拟退火算法中使用了几个列表来存储解决方案和能量值，空间复杂度与 input 的大小成正比，即 O(nm)。

2.5 遗传算法

2.5.1 算法介绍

遗传算法（GA）由美国的 John holland 于 20 世纪 70 年代提出，它受启发于自然中生物体的遗传进化规律，将解的更新迭代比作种群中基因的遗传，交叉和编译。通过选择，交叉，变异，实现解决方案在遗传过程中一代一代逐步优化，逐步接近实际最优解。

遗传算法的遗传步骤较多，一般情况下包括生成初始种群，计算个体适应度，选择，交叉，变异五个主要步骤。正是由于遗传算法的遗传步骤较多，这使得遗传算法可调节的参数较多，方法选择和搭配的灵活性较大。例如，在选择阶段中，有不同的选择方式：轮盘赌选择，随机选择等等。在选择阶段后，可以加入精英种群筛选法，在选择的开始前通过从现有种群中选择一部分适应度较好的个体直接进入下一代的种群。在交叉阶段，有单点交叉法，改进型单点交叉法，双点交叉法等等。在变异阶段，可以选择不同变异方法：随机选择两点交换，随机选择一段进行 random.shuffle 随机打乱，或是将整段随机打乱……除了方法选择丰富多样外，还有各种可调整的参数：如每一代的种群大小，精英个体比例，交叉率，变异率……我们可以通过对参数和方法的不断调整来尝试寻找我们所需要的更优解。

2.5.2 算法设计

本实验中，为了进一步探究除了模拟退火算法以外的其他元启发式算法寻优的效果，选择遗传算法进行实验。首先使用 random.sample 方法生成初始种群，种群中的个体就是一些随机生成的零件处理顺序表，如[0,1,2]（表示有三个零件，处理顺序为 1 号零件->2 号零件->3 号零件），[2,0,1]（表示有三个零件，处理顺序为 3 号零件->1 号零件->2 号零件）。

接下来进入遗传算法的迭代遗传阶段。

在单步遗传操作中，首先计算当前种群的个体适应度，并选出该种群中适应度最高的作为当代种群中的最优解，记录在记录每代种群最优解变化的 list 中便于后续可视化。接下来进行选择操作，选择方式为轮盘赌选择法，即适应度越高的个体依据适应度比例会得到更高的被选择概率。在进行了初步选择操作后，使用精英种群法将现有种群中的小部分适应度较优个体优先直接遗传进入下一代种群中。

在选择阶段进行结束后，开始交叉阶段。

根据现有研究，单点交叉法使子代继承了父代交叉点前的基因，交叉操作只改变了交叉点后的部分基因，而交叉点前的基因一直未发生改变。这样，交叉点前的基因进化只能依靠变异操作进行，而变异的概率是很低的，显然，这种交叉方法的进化对初始种群依赖较大，进化速度缓慢。[1]

所以，为提高寻优质量，我们选择改进的单点交叉法进行交叉操作，它通过随机选择交叉点和随机数来确定交叉段的保留部分。这种方法可以提高寻优质量，减少对初始种群的依赖，加快进化速度。具体操作方法如下：设存在父代 1、父代 2，其操作为：首先随机产生一交叉点；然后产生一个随机数(rand)，确定交叉段的保留部分，具体如下。1）当 rand 小于等于 0.5 时，把两父代交叉点后的基因复制给两子代，然后从父代 2

（或 1）中找出子代 1（或 2）中缺失的基因，并将其按顺序复制给两子代交叉点前的位置。2）当 rand 大于 0.5 时，把两父代交叉点前的基因复制给子代，然后从父代 2（或 1）中找出子代 1（或 2）中缺失的基因，并将其按顺序复制给两子代交叉点后的位置。[1]

最后，进行变异操作，采用的变异方法是：随机选择子代中的两个零件并将交换他们的加工顺序。

上述为每一代遗传中需要进行的单步遗传操作，依据设定的最大遗传代数，当到达最大遗传代数时，输出每一代中最优解的变化情况，最优解最小的一代对应的最优解作为算法全局最优解输出。

算法伪代码如下：

```
def genetic_algorithm(input, population_size, elite_size, max_gen, crossover_rate, mutation_rate):
    初始化种群
    for gen in range(max_gen):
        计算种群中每个个体的适应度
        找到最佳解并记录
        选择操作
        新建一个空种群
        将精英个体加入新种群
        while  新种群未满:
            if  满足交叉概率:
                随机选择两个父代个体
                交叉产生子代个体
            else:
                随机选择一个个体作为子代个体
            变异操作
            将子代个体加入新种群
        更新种群为新种群
    返回最佳解
```

算法流程图如下：

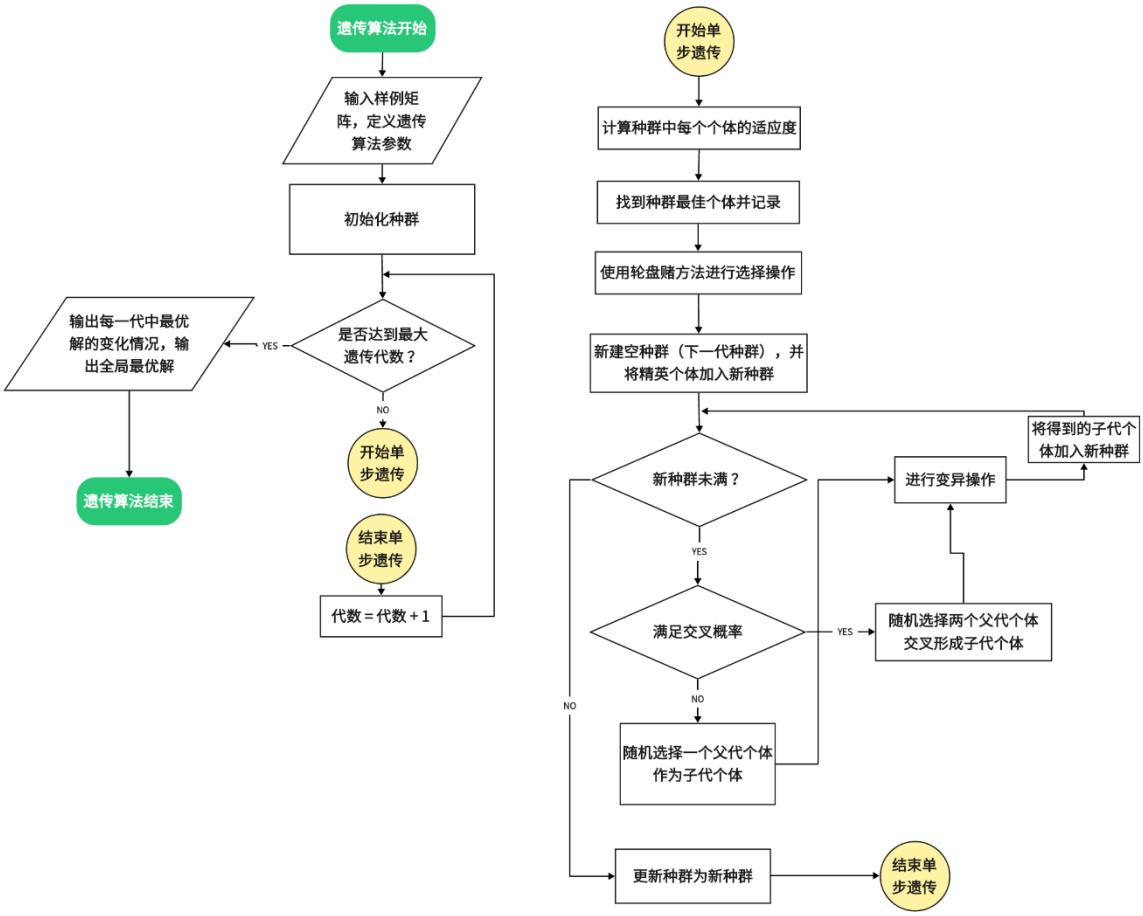


图 3 遗传算法流程图

2.5.3 算法改进

本实验中尝试在原有的遗传算法基础上嵌入爬山算法，以达到加快收敛速度，优化最终结果的效果。爬山算法的嵌入位置为单步遗传操作中变异操作后，将变异得到的子代个体 child 使用单步爬山方法生成一个临近解集，并从邻近解集中选区最大的解作为待选的 new_child，判断若 new_child 方法用时小于 child 用时，则使用 new_child 替换 child，否则 child 保持不变

修改过后的代码的伪代码部分为：

变异操作

爬山算法优化子代个体

将子代个体加入新种群

更新种群为新种群

修改后的代码的部分流程图为：

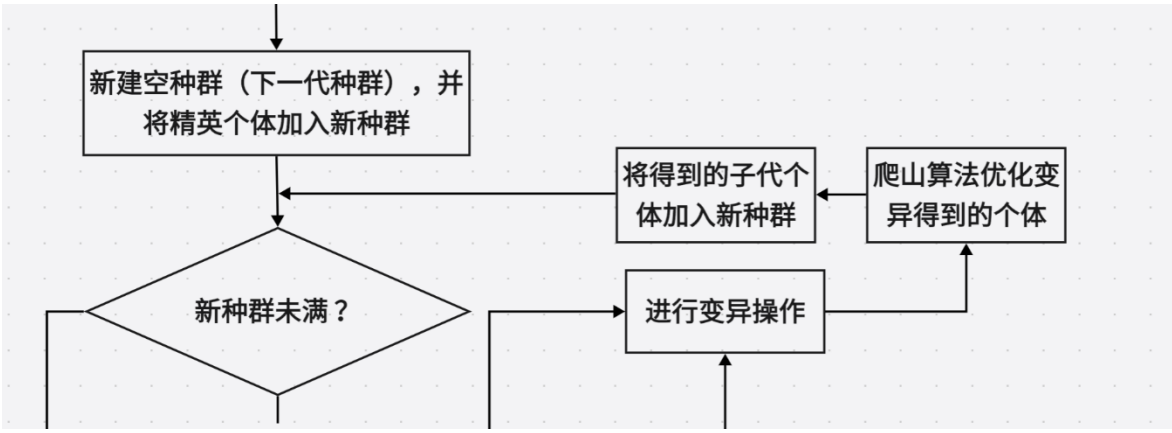


图 4 内嵌爬山算法的改进遗传算法流程图

2.5.4 算法复杂度分析

时间复杂度：由于算法进行了 max_gen 次迭代，每次迭代都需要计算种群中每个个体的适应度，因此计算适应度的总时间复杂度为 O(max_gen * population_size * calculate_fitness)。选择操作、交叉操作和变异操作的时间复杂度取决于它们的具体实现。爬山算法的时间复杂度也取决于它的具体实现。因此，总时间复杂度为 O(max_gen * population_size * (calculate_fitness + selection + crossover + mutation + climbing_algorithm))。

空间复杂度：算法需要存储初始种群和新种群，因此空间复杂度为 O(2 * population_size * size_of_individual + (calculate_fitness + selection + crossover + mutation + climbing_algorithm))。

3 实验

3.1 实验设置

实验环境：使用软件：Visual Studio Code，python 3.10.9 环境，jupyter notebook 编辑器

实验输入：flowshop-test-10-student.txt，代码中已提供预处理方法，确保.ipynb 代码与 flowshop-test-10-student.txt 位于同一文件夹路径中即可运行代码。

算法输入：样例的二维列表。不同 instance 的信息以三维 list 的形式存放在 input_data 中，如 input_data[0]即存储了第一个例子的信息，通过调整 input_instance = input_data[x]中的 x 即可调整输入算法的每个样例

实验外引库：实验使用 matplotlib 库进行图表绘制和可视化工作，需要提前下载并配置到 python 环境中

实验参数：（一）爬山算法：无可调整参数

（二）基于模拟退火算法的爬山算法：退火初始温度，冷却速率，终止温度

退火初始温度默认值为 1000，冷却速率默认值为 0.00075，终止温度默认值为 1

（三）遗传算法：每代种群大小，精英个体数量，最大遗传代数，交叉率，变异率

每代种群大小默认值为 200

精英个体数量默认值为种群大小*0.2 = 200*0.2 = 40

最大遗传代数默认值为 800

（普通遗传算法最大遗传代数设置为 400~800 合适，对于内嵌爬山算法的遗传算法的改进算法，考虑到较快的收敛速度和较大的时间开销，最大遗传代数设置为 100~400 合适）

交叉率默认值为 0.8

变异率默认值为 0.03

（四）内嵌爬山算法的改进遗传算法：同遗传算法

时间计算：使用 python 中的 time 库计算每次算法运行使用的时间

3.2 实验结果 [可包含参数实验、对比实验等，可以组织在不同的小节中]

说明：本次实验结果展示基于 instance 0 和 instance10，即样例 0（示例）和样例 10

选择两个样例的原因：选择示例（即 instance0）是为了方便和已经给定的正确答案进行对比对照

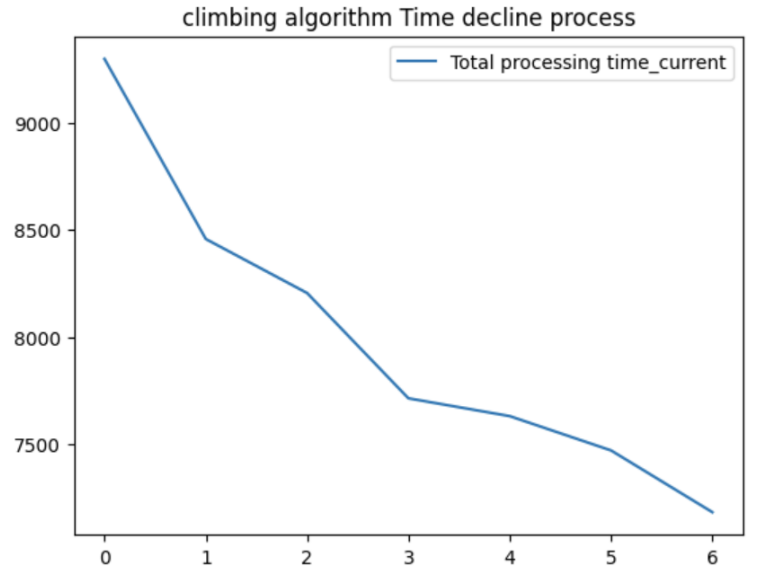
选择 instance10 的原因为：考虑到 instance0 样例过于简单，①一者不同算法的处理时间也相近，不能体现不同算法的处理效率区别 ②二者简单样例难以体现算法的整体时间下降过程，以及其中的波动现象，而复杂样例则体现明显

在 3.2.5 节中，将展示.txt 文本文件文件中所有样例的运行结果，作为最终提交的结果。

3.2.1 爬山算法

①

instance 0（样例)运行结果:



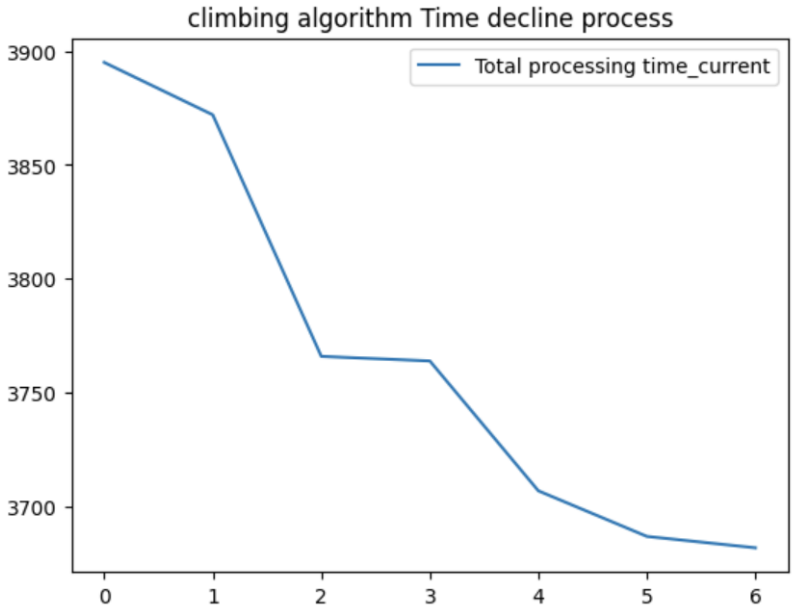

```
instance 0 's best soltion time is 7184
instance 0 's best soltion schedule is [7, 2, 10, 3, 4, 0, 6, 5, 8, 9, 1]
```

图 5 爬山算法运行 instance0 结果

instance 0 运行时间：0.001s

②

instance 10 运行结果:



instance 10 's best soltion time is 3587

instance 10 's best soltion schedule is [29, 13, 2, 3, 6, 27, 39, 25, 8, 19, 10, 11, 12, 16, 14, 15, 1, 7, 4, 9, 20, 21, 22, 45, 24, 17, 26, 5, 28, 35, 30, 47, 32, 38, 34, 0, 36, 37, 33, 18, 40, 46, 42, 43, 44, 23, 41, 31, 48, 49]

图 6 爬山算法运行 instance10 结果

instance 10 运行时间：0.022s

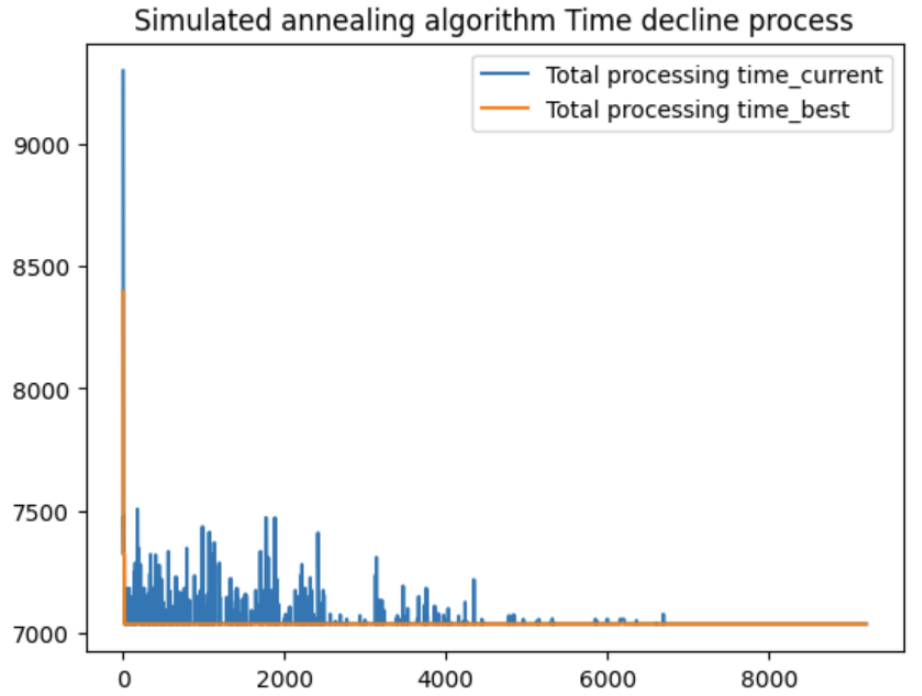
Total processing time_current 表示每轮爬山后当前最优解值，爬山算法在当临近解空间中不存在加工总用时短于现有解加工总用时的时候，算法结束，输出当前得到的解作为优解。

爬山算法每次运行得到的 best solution time 变化波动较大，所以本次实验通过多次实验取平均值位置的 best solution time 及其对应的 best solution 作为结果输出

3.2.2 基于模拟退火算法的爬山算法

①

instance 0 运行结果：



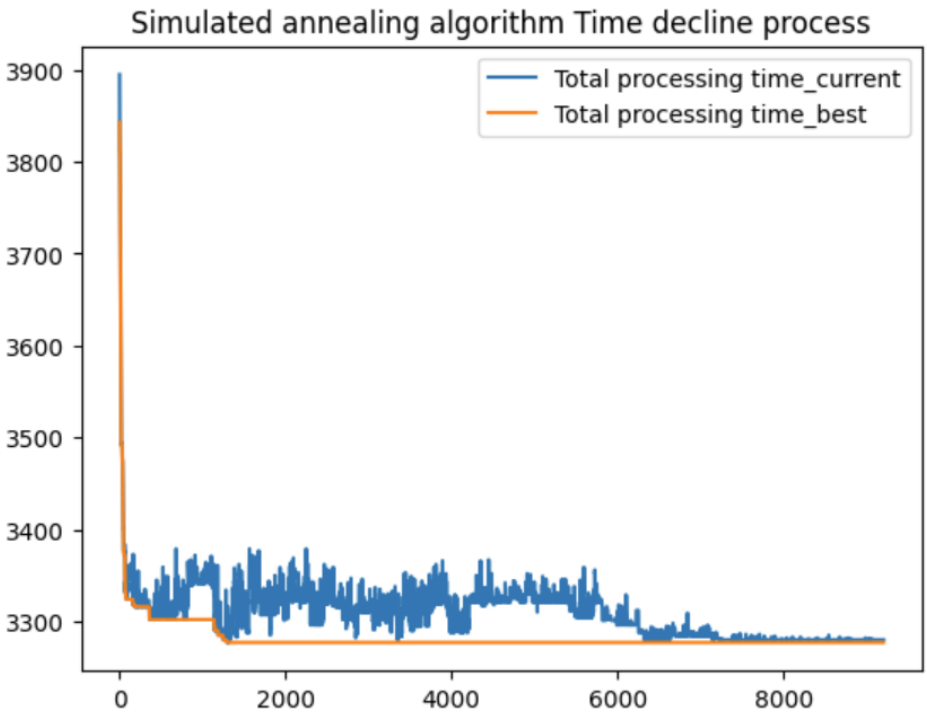

```
instance 0 's best soltion time is 7038
instance 0 's best soltion schedule is [7, 0, 2, 4, 6, 3, 10, 1, 8, 9, 5]
```

图 7 基于模拟退火算法的爬山算法的 instan0 运行结果

instance 0 运行时间：3.819s

②

instance 10 运行结果：



instance 10 's best soltion time is 3277

instance 10 's best soltion schedule is [12, 13, 39, 34, 49, 41, 25, 24, 1, 8, 7, 36, 32, 2, 18, 45, 16, 33, 0, 29, 4, 11, 35, 43, 20, 5, 10, 40, 26, 9, 17, 27, 44, 38, 28, 22, 3, 46, 37, 42, 14, 21, 19, 31, 48, 6, 15, 47, 23, 30]

图 8 基于模拟退火算法的爬山算法的 instance10 运行结果

instance 10 运行时间：25.969s

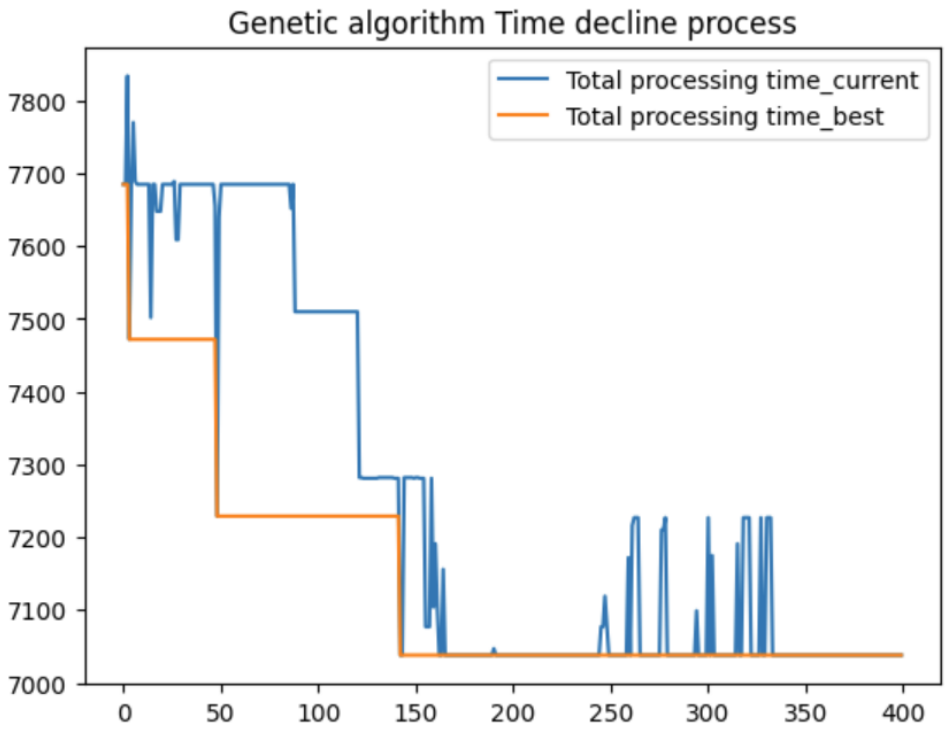
Total processing time_current 表示每轮模拟退火算法后的当前解用时，它的频繁波动源于模拟退火算法以一定概率接受劣质解的特性，但随着退火的进行，接受劣质解的概率越来越低，波动幅度和频率也越来越低。Total processing time_best 表示目前为止得到的最优方法的用时（单调下降，每轮退火后只有存在更优解时才会更新）

多轮实验显示，相比较于爬山算法，基于模拟退火算法的爬山算法的输出结果时间稳定得多，对于样例 0，经过 10 次重复实验，HC+SA 算法的输出最优结果时间始终为 7038（但是 best solution schedule 变化，因为不同的 schedule 可能得到的总加工时间相同，最优方法不唯一），即便对于复杂一些的样例 10，HC+SA 算法得到的输出结果也稳定在 3277 附近，波动值不超过±5。

3.2.3 遗传算法

①

instance 0 运行结果：



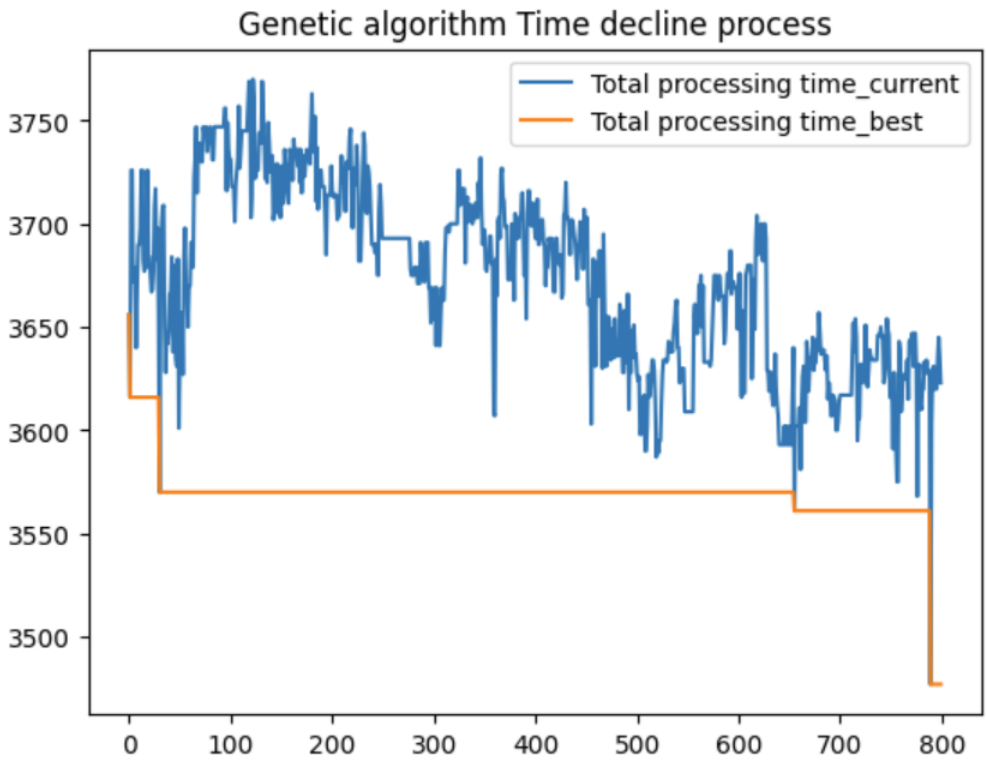

```
instance 0 's best soltion time is 7038
instance 0 's best soltion time is 7038.0
instance 0 's best soltion schedule is [7, 2, 0, 4, 3, 6, 8, 10, 5, 1, 9]
```

图 9 遗传算法的 instance0 运行结果（400gens）

instance 0 运行时间：2.439s

②

instance 10 运行结果：



instance 10 's best soltion time is 3477

instance 10 's best soltion time is 3477.0000000000005

instance 10 's best soltion schedule is [18, 9, 13, 16, 23, 41, 43, 10, 1, 31, 17, 48, 35, 39, 8, 14, 7, 5, 38, 12, 49, 26, 11, 21, 28, 37, 33, 42, 20, 47, 25, 34, 24, 6, 45, 36, 22, 2, 40, 3, 30, 4, 0, 32, 19, 46, 29, 15, 44, 27]

图 10 遗传算法的 instance10 运行结果（800gens）

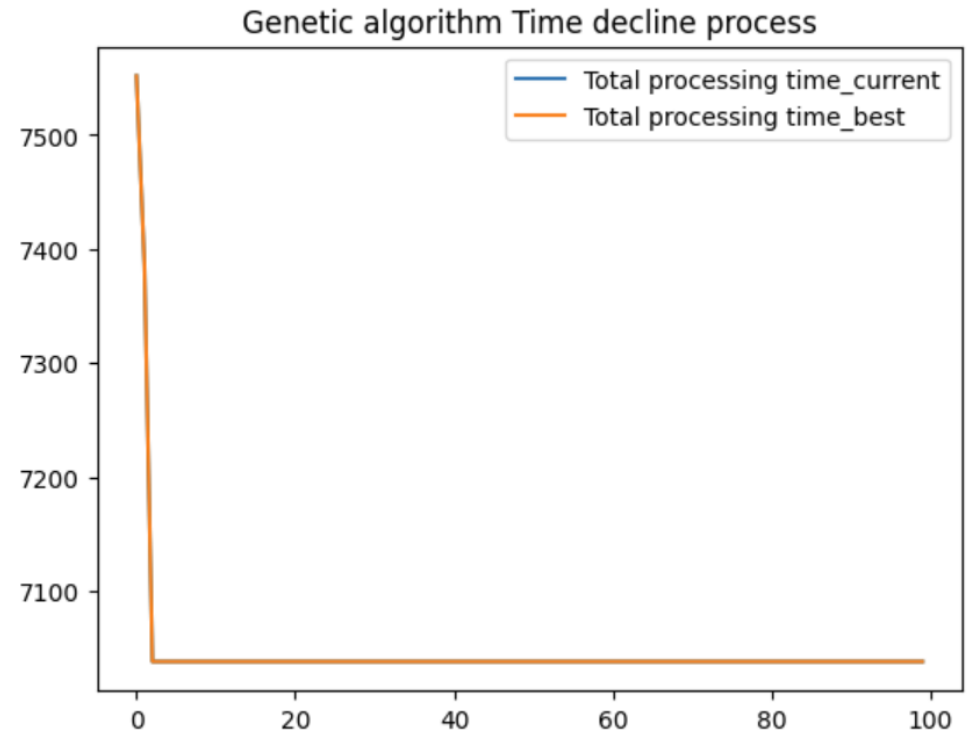
instance 10 运行时间：27.861s

遗传算法总结与改进的遗传算法总结综合在 3.2.4 小节中。

3.2.4 内嵌爬山算法的改进遗传算法

①

instance 0 运行结果：



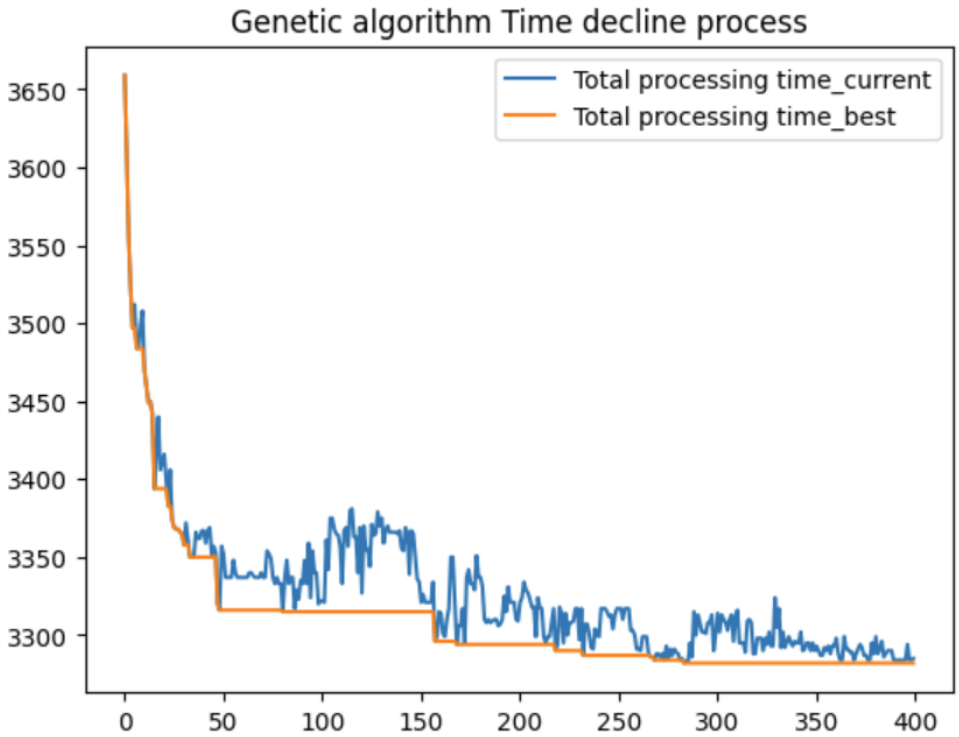

```
instance 0 's best soltion time is 7038
instance 0 's best soltion time is 7038.0
instance 0 's best soltion schedule is [7, 4, 1, 0, 2, 10, 3, 8, 9, 6, 5]
```

图 11 改进的遗传算法的 instance0 运行结果（100gens）

instance 0 运行时间：5.967s

②

instance 10 运行结果：



instance 10 's best soltion time is 3282

instance 10 's best soltion time is 3282.0

instance 10 's best soltion schedule is [12, 13, 39, 49, 43, 2, 38, 41, 34, 35, 32, 33, 24, 4, 25, 5, 16, 29, 37, 44, 40, 20, 1, 18, 10, 8, 17, 26, 0, 6, 7, 36, 42, 28, 21, 22, 19, 23, 45, 11, 9, 14, 46, 3, 48, 27, 31, 47, 15, 30]

图 12 改进的遗传算法的 instance10 运行结果（400gens）

instance 10 运行时间：152.155s

遗传算法运行时间受遗传代数选择和每代种群大小影响较大，所以实验中根据不同样例的解的收敛速度，选择不同的最大遗传代数。

Total processing time_current 表示每代遗传算法后的当前代中最优个体用时，它的频繁波动源于遗传算法解的选择、交叉和突变。Total processing time_best 表示目前为止得到的全局最优方法的用时（单调下降，每代遗传后只有存在更优解时才会更新）

根据结果图表我们可以发现，原始的遗传算法解的波动幅度较大，收敛速度较慢，需要得到更优解往往需要更大的原始种群大小和更多的最大遗传代数。而内嵌爬山算法的改进遗传算法解的波动幅度较小，收敛速度明显快于原始遗传算法，且改进遗传算法最终得到的解优于原始遗传算法得到的最终解。

改进遗传算法的缺陷在于由于每一代中内嵌了爬山算法以优化子代，时间开销大大增加。尤其随着遗传代数的增加，程序运行时间增长速度较快。对于小规模问题，由于所需最大遗传代数较少，这种时间开销影响不大，但是对于大规模问题，当所需遗传代数大大增加时，时间开销的问题就更加突出，需要酌情考虑是否使用优化。

3.2.5 多组参数实验

表 1 基于模拟退火算法的爬山算法参数实验

退火初始温度	退火 冷却速率	instance 0 运行结果	运行时间 （秒）	instance 10 运行结果	运行时间 （秒）
300	0.00075	7038	3.139	3277	20.978
1000	0.00075	7038	3.758	3277	24.987
3000	0.00075	7038	4.380	3277	29.621
1000	0.0001	7038	28.264	3277	192.906
1000	0.001	7038	2.900	3280	19.176
1000	0.01	7038	0.287	3295	1.895

参数实验对退火初始温度与退火冷却速率两个参数都进行了控制变量法实验。

结果显示，退火初始温度的变化对最终运行结果的影响较小，不会引起最终运行结果的大幅度改变；同时不同的退火温度的程序运行时间相近，说明退火温度对程序运行时间影响也较小。

退火冷却速率对最终运行结果的影响较大，同时决定了程序运行时间。冷却速率的变化与运行时间的变化成比例关系，当冷却速率增加为 10 倍时，程序运行时间也相应减少为约 1/10。越快的冷却速率一方面大大减少了程序运行时间，另一方面却使最终结果更劣，这是因为越快的退火速率会使温度快速下降，使得在模拟退火算法程序运行初始阶段，接受劣质解的概率减小，将更快陷入局部最优解中，更难找到全局实际最优解。

表 2 遗传算法参数实验

每代种群大小	精英个体比例	最大遗传代数	交叉率	突变率	instance 0 运行结果	运行时间 （秒）	instance 10 运行结果	运行时间 （秒）
200	0.2	800	0.8	0.03	7038	4.804	3545	27.810
100	0.2	800	0.8	0.03	7038	2.184	3582	13.978
50	0.2	800	0.8	0.03	7038	0.987	3701	6.855
200	0.4	800	0.8	0.03	7038	5.057	3457	27.282
200	0.6	800	0.8	0.03	7038	5.263	3499	28.914
200	0.2	800	0.6	0.03	7038	4.960	3500	28.274
200	0.2	800	1	0.03	7038	4.998	3482	29.537
200	0.2	800	0.8	0.1	7038	4.898	3489	28.565
200	0.2	800	0.8	0.5	7038	5.103	3443	28.724

表 3 改进遗传算法参数实验

每代种群大小	精英个体比例	最大遗传代数	交叉率	突变率	instance 0 运行结果	运行时间（秒）	instance 10 运行结果	运行时间（秒）
200	0.2	400	0.8	0.03	7038	23.539	3280	151.626
100	0.2	400	0.8	0.03	7038	11.583	3291	76.151
50	0.2	400	0.8	0.03	7038	5.710	3309	37.959

遗传算法参数实验对初始种群大小，精英个体比例，交叉率，突变率进行了控制变量实验。改进遗传算法参数实验对初始种群大小进行了控制变量实验。

实验结果显示：每代种群大小对程序运行结果和运行时间的影响较大。较小的每代种群大小能显著减少程序运行时间，但是寻找到优解的概率减小，得到的输出解往往会劣于较大的种群大小的到的解。

精英个体比例对运行时间几乎无影响，对程序运行结果有一定影响，过大的精英个体比例会使输出结果更劣。

不同的交叉率对运行时间几乎无影响，对程序运行结果有一定影响，较大的交叉率会使输出结果更优。

不同的突变率对运行时间几乎无影响，对程序运行结果有一定影响，过大的突变率会使每代的最优解变化剧烈，在图像上显示为剧烈波动的曲线，虽然通过增加了随机性丰富了子代个体，然而对于寻找到所需的最优解未必有很大帮助。

3.2.6 结果综合 所有样例结果结果提交

表 4 算法结果与运行时间

算法	instance 0 运行结果	运行时间（秒）	instance 10 运行结果	运行时间（秒）
HC 爬山算法	7184	0.1	3587	0.2
HC+SA 基于模拟退火算法的爬山算法	7038	3.8	3277	25.3
GA 遗传算法	7038	2.6	3477	28.5
GA+HC 内嵌爬山算法的改进遗传算法	7038	6.1	3282	151.4

综合四种算法，我们发现，基于模拟退火算法的爬山算法时间开销位于可接受范围内，且算法输出的最终解效果较优，所以我们最终选择 HC+SA 算法来处理.txt 文本文件中的所有案例，并将输出作为最终用于评分的提交解。

表 5 HC+SA 算法 所有样例运行时间

样例	instance 0 运行结果
instance 0	7038
instance 1	8366
instance 2	7166

instance 3	7312
instance 4	8003
instance 5	7720
instance 6	1438
instance 7	1973
instance 8	1109
instance 9	1927
instance 10	3277

4 总结

文章针对于 FLOW-SHOP 流水线调度问题提出了一种贪心算法和两种元启发式算法作为解决方案：爬山算法、基于模拟退火算法的爬山算法、遗传算法，并尝试内嵌爬山算法得到改进遗传算法以获得更优的解。实验结果显示：爬山算法能够在一定程度上寻找到问题的较优解。相较于爬山算法，基于模拟退火算法的爬山算法有更大概率寻找到问题的较优解，但是时间运行代价更高。遗传算法也能寻找到问题的较优解，并且相对于基于模拟退火算法的爬山算法，其运行的时间代价相近，然而随着工件数和机器数的增多，遗传算法得到的结果略差于方案二。加入爬山算法优化的遗传算法能够更快地收敛，得到比单独使用遗传算法更优的结果，但是运行时间开销大大增加。

本实验有不少考虑欠妥之处，列举如下：

（一）遗传算法的算法设计还可以进一步改进

实验结果显示，尽管尝试使用了内嵌爬山算法来改进现有的遗传算法，这种改进方法虽然能有效优化最终解，然而时间开销大，当问题复杂度进一步升高时，巨大的时间开销将使得这种方法不再适用。

正如遗传算法的算法介绍部分所述，遗传算法可调整的部分多，从算法结构，到遗传每一步函数都可以选择不同的方案，尝试不同的组合。而本实验中除了尝试改进交叉函数方法外，并未通过修改变异的方法或是修改选择的方法以尝试更多的遗传算法的组合，最终得到的输出结果也劣于基于模拟退火算法的爬山算法，仍需进一步改进。

可能的解决方案：花更多时间尝试不同的交叉，变异，选择方法的搭配组合，如交叉方式可以选择单点交叉，改进单点交叉，双点交叉，多点交叉等等；变异方式可以选择随机选择两点交换位置，随机选择多点交换位置，随机选择部分打乱顺序等等；选择方式可以选择轮盘赌选择，锦标赛选择，截断选择，蒙特卡洛选择等等。通过尝试不同的搭配组合可能能找到相对于当前组合的更优解。

（二）同一算法，针对不同规模的 FLOW-SHOP 问题可以使用不同的参数，以提高运行效率

实验过程中发现，对于规模不同的 flow-shop 问题，无论是使用模拟退火算法还是遗传算法求解，解的收敛速度都有很大的不同。对于规模较小的问题，使用较快的退火速率，较小的种群数量和较少的遗传代数也能收敛得到较优解，如果参数设定不正确，将导致解在已经收敛后程序仍在运行，造成不必要的时间开销。

可能的解决方案：因此，同一算法，针对不同规模的 FLOW-SHOP 问题可以尝试使用不同的参数，以提

高运行效率，在实际应用中可以将问题分类为几种不同规模，并将预先训练好的针对不同规模问题的参数对应应用，将大大减少时间开销，提高程序运行效率。

参考文献:

[1] 张博凡,黄宗南.基于变形遗传算法交叉算子的 Flow-Shop 问题求解[J].*制造业自动化*,2011,33(19):27-29+46.

[2] 高鹰,谢胜利.基于模拟退火的粒子群优化算法[J].*计算机工程与应用*,2004(01):47-50.