基于模拟退火、遗传算法的置换流水车间调度问题求解

张泽渊 (学号: 1120200325)

摘　要: 置换流水车间调度问题（PFSP）旨在优化 n 个工件在多台机器上的加工顺序以最小化总完工时间。PFSP 规定各个机器只能按相同的顺序加工一组工件，不考虑工件在不同机器中加工顺序的改变对总加工时间的影响，把解空间限定在 n 个工件各种排列顺序构成的集合。本文使用模拟退火和遗传算法来求解 PFSP。模拟退火算法通过模拟物理中粒子退火过程，逐步降低系统温度以探索可能的解空间，随机寻找目标函数的全局最优解。遗传算法模拟自然选择和遗传机制，通过选择、交叉和变异操作在解的种群中进行搜索，以期达到优化目标。实验结果显示，遗传算法在问题规模较大时通常能够找到比模拟退火算法更优的解，但算法耗费的时间也更长。模拟退火算法在搜索速度上的表现更为出色，适用于快速寻求近似解的场景。

关键词: 置换流水车间调度问题；模拟退火算法；遗传算法；启发式算法；智能算法

1 引言

1.1 问题背景

流水车间调度问题（Flow Shop Scheduling Problem, FSP）在现代工业生产中是一个关键问题。它涉及在多台机器上对多个工件进行按顺序加工，每台机器同一时间只能加工一个工件，每个工件同一时间只能被一台机器加工，各个工件在每台机器上的加工时间是给定的。问题目标是最小化加工完成所有工件的总时间。FSP 属于 NP-hard 问题，无法通过多项式时间算法求解，因此通常采用启发式算法，如遗传算法、模拟退火算法、蚁群算法等。

根据一组工件能否在不同机器中改变加工顺序的条件，可将 FSP 分为置换流水车间调度问题（PFSP）与非置换流水车间调度问题（NPFSP）两类。PFSP 规定工件在不同机器中的加工顺序必须相同，将解空间限定在 n 个工件各种排列顺序构成的集合。本次求解只针对 PFSP，并且，为了简化计算，规定第一台机器从 0 时刻开始加工工件，这样最后一台机器加工完成最后一个工件的时刻对应的数值就是加工完成全部工件所用的总时长。

1.2 问题数学描述

1.2.1 符号解释

为方便使用数学语言描述置换流水车间调度问题，引入表格 1 中的各个符号。

表格 1 数学符号及含义

符号	属性	含义
m	正整数	加工工件的机器数量
n	正整数	待加工的工件数量
π	排列变量	$\pi = (\pi_1, \pi_2, \dots, \pi_n)$, 成员是工件的编号，成员的顺序代表工件的加工顺序。 例如，$(3,1,2)$表示先加工工件 3，再加工工件 1，再加工工件 2
π_j	正整数	表示π中的第j个成员，代表第j个被加工的工件。 例如，若$\pi = (3,1,2)$，则$\pi_2 = 1$

$C(i,\pi_j)$	函数	第i台机器加工完成工件π_j的完成时刻
t	二维数组	m 行 n 列，成员为t_{ij}，存储各机器加工各工件所需的时间
$t_{i,j}$	正整数	t中第i行第j列的成员，代表第i台机器加工工件j所需时间
x_{ij}	二值变量(0 或 1)	如果工件j安排在π的位置i，那么$x_{ij}=1$，否则$x_{ij}=0$。

1.2.2 数学表述

目标函数：

$$\min \ C(m,\pi_n)$$

约束条件：

$$\text{s.t. } C(1,\pi_1)=t_{1,\pi_1} \tag{1}$$

$$C(1,\pi_j)=C(1,\pi_{j-1})+t_{1,\pi_j} \ \text{ for } j=2,3,\dots,n. \tag{2}$$

$$C(i,\pi_1)=C(i-1,\pi_1)+t_{i,\pi_1} \ \text{for } i=2,3,\dots,m. \tag{3}$$

$$C(i,\pi_j)=max\{C(i,\pi_{j-1}),C(i-1,\pi_j)\}+t_{i,\pi_j} \ \text{ for } i=2,3,\dots,m;j=2,3,\dots,n. \tag{4}$$

$$\sum_{j=1}^{n} x_{ij}=1 \ \text{for } i=1,2,\dots,n. \tag{5}$$

$$\sum_{i=1}^{n} x_{ij}=1 \ \text{for } j=1,2,\dots,n. \tag{6}$$

决策变量：

$$\text{var.} \quad x_{ij}\in\{0,1\} \qquad \text{for } i,j=1,\dots,n$$

$$\pi_i=\sum_{j=1}^{n} j\cdot x_{ij} \qquad \text{for } i=1,2,\dots,n.$$

1.2.3 约束条件含义

约束条件(1)(2)(3)(4)参考了顾瀚、王雷等人发表的《改进人工蜂群算法求解置换流水车间调度问题》中的相关约束条件[1]。

约束条件(1)(2)(3)(4)的递归依赖关系，保证了每个工件在移至下一个机器之前，必须等待前一个机器上的工作完成，确保一个工件不能同时在不同的机器上加工。

约束条件(5)保证π每个位置只能存在一个工件，确保每个机器同时只能加工一个工件。

约束条件(6)保证每个工件只能在π的排列中出现一次，确保每个工件在每台机器上只加工一次。

1.3 问题解决方案

分别使用模拟退火和遗传算法来求解 PFSP，最终结果取两种算法的更优解。

模拟退火算法模拟物理中粒子退火过程，将接受更差的解的概率与系统的温度相结合，通过不断降低温度收敛优化的结果，最终将获得的局部最优解近似为全局最优解。遗传算法模拟自然选择和遗传机制，通过选择、交叉和变异操作在解的种群中进行搜索，通过不断地培育携带更优解的子代寻找全局最优解。

模拟退火算法的搜索速度很快，适用于快速寻求近似解的场景，但由于每次只对一组解进行迭代，解的多样性较差，在问题规模较大时可能无法找到全局最优解。遗传算法耗费的时间较长，但由于每次能够同时对多组解进行迭代，搜索到全局最优解的可能性大大增加，如果不限制种群数量与迭代次数，将会得到比模拟退火算法更好的效果。

2 算法设计

2.1 模拟退火算法

2.1.1 算法思想

模拟退火算法（Simulated Annealing, SA）的核心思想如下：

(1) 从一个初始解开始，设定一个较高的初始温度 T；

(2) 在当前解的邻域中随机选择一个新的解；

(3) 如果新的解比当前解好（目标函数值更优），则接受新解；
如果新的解比当前解差，则以一定概率接受新解，概率与温度和解的劣化程度相关。设新旧两个解目标函数值之差的绝对值是△E，以 exp(-△E/T)的概率接受新的解；

(4) 逐步降低温度 T，一般以 T← α T 的策略降温，α 是一个 0 到 1 之间的常数；

(5) 重复(2)(3)(4)，直到温度降到某个阈值或到达最大迭代次数时，停止搜索。

2.1.2 关键操作

2.1.2.1 邻域定义

在置换流水车间调度问题中，某加工排列顺序的邻域定义：有两个工件加工位置与其相反，其他工件加工位置与其相同的排列顺序。

在搜索当前排列顺序的邻域时，随机选择当前排列中两个工件的位置进行交换得到一个新的排列顺序，这个新的排列就是旧排列的一个邻域排列。

2.1.2.2 加工完成时间计算方法

加工完成时间使用动态规划方法生成。

设：

(1) 机器数量为 m，工件数为 n；

(2) 调度顺序存储在一维数组 sort[n]中，sort[i-1]表示第 i 个被加工的工件；

(3) 各机器加工各工件所需的时间存储在二维数组 T[m][n]中，T[i-1][j-1]表示第 i 台机器加工工件 j 所需要的时间；

(4) 各机器加工完成各工件时的时刻存储在二维数组 ans[m+1][n+1]中，数组各成员都初始化为 0，ans[i][j]表示第 i 台机器加工完成工件 j 时所处的时刻。

由于数组各成员都初始化为 0，当 i 和 j 分别为 1 时，max(ans[i-1][j],ans[i][j-1])分别为 max(0,ans[i][j-1])和 max(ans[i-1][j],0)，可将 PFSP 数学表达中的约束条件(1)(2)(3)(4)合并为约束条件(4)，于是使用转移方程：ans[i][j]= T[i-1][sort[j-1]]+max(ans[i-1][j],ans[i][j-1])计算出的 ans[m][n]即为所求。

算法伪代码：

Algorithm 1 Calculation of Makespan

Input: sort[n], T[m][n]
Output: ans[m][n]
1: **for** each i ∈ [1, n] **do**
2: **for** each j ∈ [1, m] **do**
3: ans[i][j]← T[i-1][sort[j-1]]+**max**(ans[i-1][j],ans[i][j-1])
4: **end for**
5: **end for**
6: **return** ans[i][j]

2.1.2.3 初始解生成

(1) 初始化加工顺序为 0,1,…，n-1（设索引从 0 开始）；

(2) 使用 C++的库函数 shuffle 函数将加工顺序随机打乱，将打乱后的顺序作为初始解。

2.1.2.4 温度降低方式

使用 T←αT 的策略降温，温度不能降低太快，控制 α 在 0.80 到 0.99 之间

2.1.2.5 接受准则

无条件接受更优解。

有条件接受更差解，根据 Metropolis 准则，若在温度 T 下，新的排列顺序对应的完工时间比旧的排列顺序对应的完工时间多出△E，则可以根据概率$e^{-\Delta E/T}$接受这个更差的排列顺序。

2.1.3 算法流程图（见图 1）

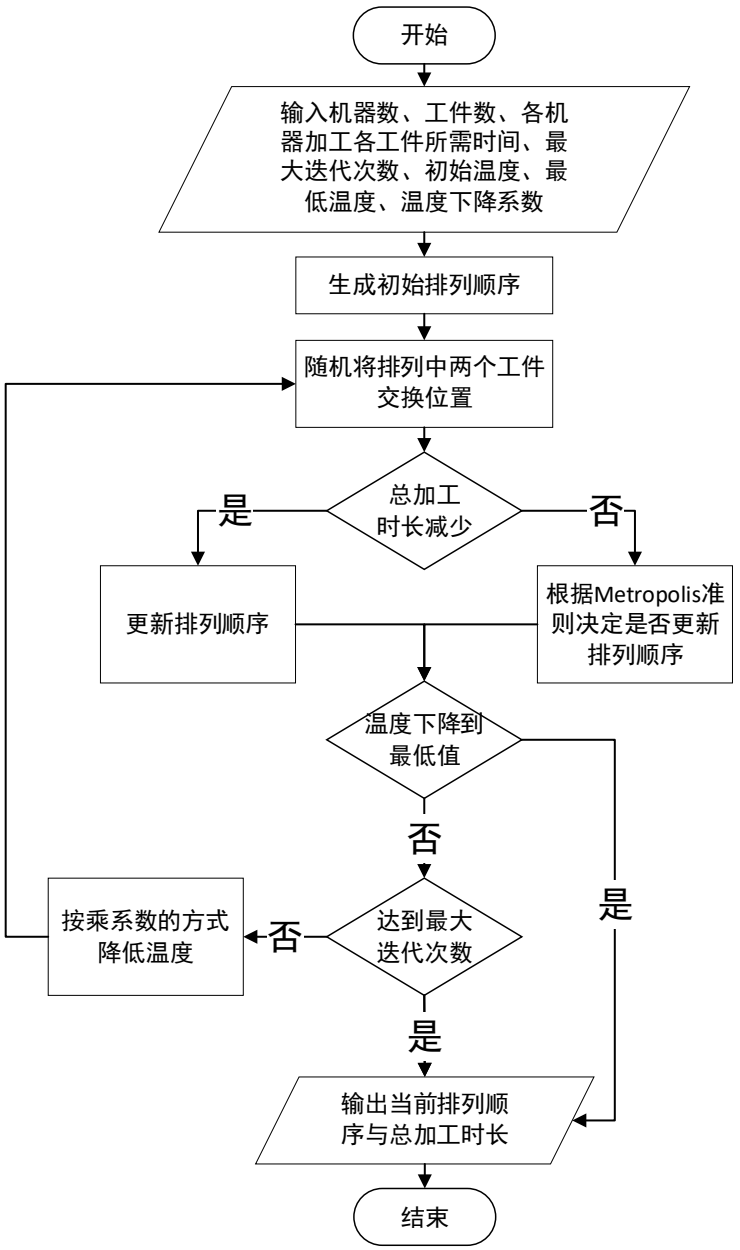

图 1 模拟退火算法流程图

2.2 遗传算法

2.2.1 算法思想

遗传算法（Genetic Algorithm, GA）的核心思想如下：

(1) 将每个可能的解表示为一个个体（染色体），通常用二进制、数字排列等编码方式；
(2) 生成一个随机的初始种群，每个个体代表一个可能的解；
(3) 设计适应度函数，用于评估当前种群每个个体的优劣程度，适应度函数反映了个体解的质量；
(4) 根据个体的适应度值选择个体进行繁殖，常用的选择方法包括轮盘赌选择、排序选择等；
(5) 对选中的个体进行交叉操作生成新的个体（子代），交叉操作模拟生物遗传中的基因重组过程，常用的交叉方法有单点交叉、两点交叉和均匀交叉等；
(6) 对新生成的个体进行变异操作，以一定的概率改变个体的某些基因，增加种群的多样性，避免陷入局部最优；
(7) 用新生成的个体（子代）替换当前种群，替换策略可以是完全替换或部分替换；
(8) 重复(3)到(7)，最后根据设定的终止条件（如达到最大代数）停止进化过程。

2.2.2 关键操作

李小缤等人提出了一种求解置换流水车间调度问题的改进遗传算法[2]，本文对其中一些操作的思路进行了借鉴。

2.2.2.1 初始种群生成

初始种群通常由随机生成的个体解组成，每个解代表一个可能的工件处理顺序。

每个解的生成过程如下：

(1) 初始化加工顺序为 0,1,…，n-1（设索引从 0 开始）；
(2) 使用 C++的库函数 shuffle 函数将加工顺序随机打乱，将打乱后的顺序作为初始解。

2.2.2.2 适应度计算

针对 PFSP，设定适应度为解的完工时间（makespan）的倒数。完工时间越短，适应度越高。适应度计算被用在轮盘赌选择策略中以便选择出进行基因交叉的父代。

2.2.2.3 选择操作

选择操作基于种群中个体的适应度计算,本文使用轮盘赌方法从当前种群中选出将参与产生后代的个体，具体的步骤如下：

(1) 计算种群中所有个体适应度的总和，将总和用作归一化个体适应度的基数，使得个体的选择概率等于其适应度与总适应度之比；
(2) 为每个个体计算累积的选择概率（相当于在轮盘上划分每个个体对应的扇区大小）；
(3) 使用随机数决定每次选择落在哪个扇区，从而选择出对应的个体；
(4) 将选出的个体作为下一代的父代。

2.2.2.4 交叉操作

本次实验采用单点交叉，单点交叉的第一步是随机选择一个位置作为交叉点，这个点将调度顺序分为两部分。

根据交叉点，将两个父代的部分排列进行交换，从而形成新的后代。为了便于编写程序，一次只返回一个子代。父代 1 的前半部分作为子代的前半部分，父代 2 按顺序填充父代 1 的前半部分没有出现的数字到子代的后半部分，形成完整的子代。

2.2.2.5 变异操作

首先设置一个变异概率，之后遍历种群中的每个个体，每次生成一个 0 到 1 之间的随机数，如果这个随机数小于变异概率，就对当前的个体执行变异操作。

对个体进行变异操作时，随机选择当前个体对应排列中两个工件的位置进行交换，将得到的新排列作为变异的结果。

2.2.2.6 替代策略

采用精英保留策略，设置父代的留存率为 a%，首先对父代个体进行评估，将父代个体按适应度从高到低的顺序排列，保留父代前 a%的个体到下一代，剩余的位置用由交叉和突变产生的子代填充，最终形成新一代种群，填充时要保持每一代种群的个体总数不变。

2.2.3 算法流程图（见图 2）

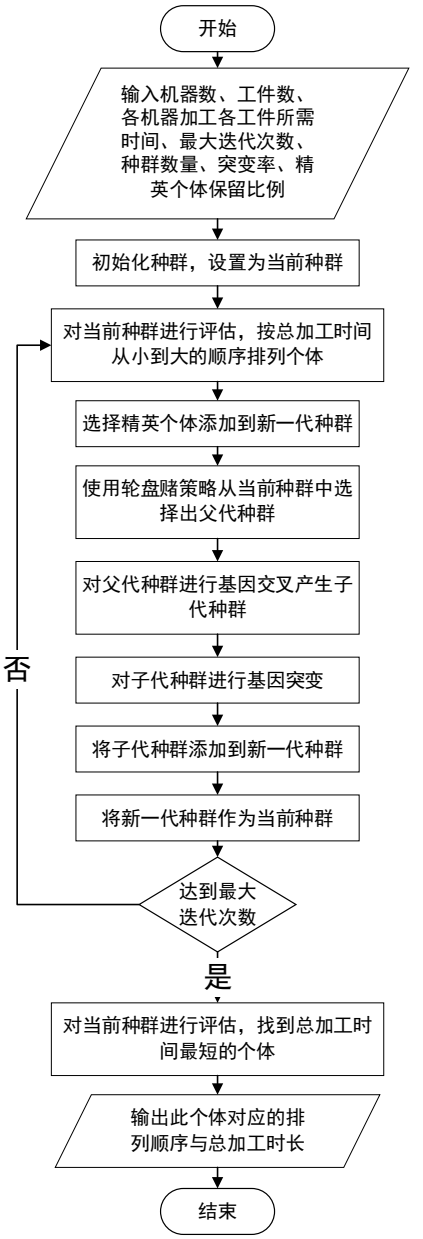

图 2 遗传算法流程图

3 实验

3.1 实验设置

在本次实验中，PFSP 求解程序使用 C++语言编写，详见 PFSP_Optimization_Experiment.cpp 文件，甘特图绘制程序使用 Python 语言编写，详见 GanttChart 文件夹下的 DrawGanttChart.py 文件。

推荐的实验环境如下：

操作系统：Windows10、Windows11

编程语言：C++、Python

环境依赖：MinGW-w64（或其他 GNU 编译器套件），Python 3.8 及以上版本

编辑器 ：VSCode

执行代码前需要将命令终端切换到代码文件所在的目录，并直接编译运行即可。PFSP 求解程序默认按次序对 11 个测试用例进行测试，可在代码对应位置切换为对单个用例进行测试，甘特图绘制程序需要输入测试用例序号和相应的调度顺序。

注意事项：字符的编码格式不同可能导致代码中的中文变成乱码，遇此情况请切换到 UTF-8 编码格式。

3.2 模拟退火算法参数实验

模拟退火算法共四个参数可供调整，分别是：

(1) 初始温度 temp ；

(2) 最低温度 endTemp ；

(3) 降温百分比 coolingRate ；

(4) 最大迭代次数 maxIterations 。

3.2.1 起始温度参数对照

保持 endTemp = 0.1； coolingRate = 0.95； maxIterations = 10000

改变起始温度，分别对测试用例 3、6、9 进行测试，每个用例进行 10 次实验，记录加工时长的最小值、平均值、最大值，记录格式为：最小值/平均值/最大值，测试结果见表格 2。

表格 2 起始温度参数对照实验

编号	temp=10	temp=100	temp=1000	temp=10000	temp=100000
3	6664/6917.1/7310	6655/6902.8/7278	6666/6857.8/7135	6655/6801/7123	6655/6752.3/7012
6	1461/1507.7/1548	1463/1494.2/1557	1429/1490.2/1560	1410/1486.6/1561	1444/1481.2/1512
9	1839/1891.3/2013	1855/1882.8/1910	1848/1890.8/1931	1808/1858.9/1947	1851/1876/1904

分析实验结果，平均值随着起始温度的提高在整体上取得了更优的结果；最小值随着起始温度从 10 上升到 10000 时整体上不断减小，但随着起始温度从 10000 上升到 100000 时反而得到了更差的结果；最大值在各测试用例中的变化趋势有所不同。

可以看出，对于规模较小的测试用例 3，起始温度的提升带来了较好的影响，但对于规模较大的测试用例 6、9，温度并不是越高越好，当温度过高时会导致算法选取很多较差的邻域解，最终得到的结果也会相应变差。因此，应对选择一个较高的起始温度，但是也要控制起始温度不要过于巨大。

3.2.2 最低温度参数对照

保持 temp=10000; coolingRate = 0.95; maxIterations = 10000

改变最低温度，分别对测试用例 2、5、10 进行测试，每个用例进行 10 次实验，记录加工时长的最小值、平均值、最大值，记录格式为：最小值/平均值/最大值，测试结果见表格 3。

表格 3 最低温度参数对照实验

编号	endTemp=10	endTemp=1e-1	endTemp=1e-5	endTemp=1e-10	endTemp=1e-20	endTemp=1e-30
2	5066/5452.7/5826	5066/5067.6/5077	5066/5067/5071	5066/5066/5066	5066/5066/5066	5066/5066/5066
5	6991/7148.6/7628	7044/7081.2/7252	6961/7036.2/7055	7021/7053.1/7118	7044/7047.4/7072	7021/7045.1/7072
10	3009/3098/3258	2908/2970.9/3073	2863/2918.9/2969	2848/2882.6/2933	2793/2836.6/2893	2791/2837.2/2889

整体而言，最低温度越低，解的质量越好。对于用例 2，最低温度降低到 1e-10 后，10 次实验的结果完全一致，都给出 5066 的解，这个解在用例 2 的所有结果中是最好的。但对于用例 5，当最低温度下降时，并没有获得更优的解，因为当最低温度趋于 0 时，接受更差的邻域的概率极低，这就会导致如果没有在温度较高时找到全局最优解，当温度趋于 0 时最后就会陷入到局部最优解。但从整体上看，设置一个更小的最低温度对寻找更优解是起到积极作用的。

3.2.3 降温百分比参数对照

保持 temp=10000; endTemp = 1e-30; maxIterations = 10000

改变降温百分比，分别对测试用例 1、7、8 进行测试，每个用例进行 10 次实验，记录加工时长的最小值、平均值、最大值，记录格式为：最小值/平均值/最大值，测试结果见表格 4。

表格 4 降温百分比参数对照实验

编号	coolingRate =0.99	coolingRate =0.95	coolingRate =0.9	coolingRate =0.8	coolingRate =0.7	coolingRate =0.6
1	6269/6308.6/6467	6269/6341/6530	6269/6314.9/6530	6269/6295.1/6467	6269/6347.3/6530	6269/6347.3/6530
7	1850/1877.7/1914	1850/1881.9/1910	1890/1912.5/1945	1891/1923.4/1971	1883/1938.2/1975	1916/1969/2015
8	935/949.3/964	936/963.9/990	961/973.5/989	958/973.4/1002	969/989.5/1002	975/1003.1/1042

整体而言，降低降温百分比会导致实验结果变差，降温百分比较低会导致温度下降较快，从一个局部最优解跳转到另一个局部最优解的可能降低，不利于寻找到全局最优解。应设置一个较高的降温百分比以便找到更优的解。

3.2.4 最大迭代次数参数对照

保持 temp=10000; endTemp = 1e-30; coolingRate = 0.99

改变最大迭代次数，分别对测试用例 0、4、10 进行测试，每个用例进行 10 次实验，记录加工时长的最小值、平均值、最大值，记录格式为：最小值/平均值/最大值，测试结果见表格 5。

表格 5 最大迭代次数参数对照实验

编号	maxIterations =10	maxIterations =100	maxIterations =1000	maxIterations =10000	maxIterations =100000
0	7932/8602.3/9526	8243/8915.9/10541	7038/7038/7038	7038/7038/7038	7038/7038/7038
4	10264/11324.9/11910	10603/11130/11594	9431/9482.7/9882	9431/9431/9431	9431/9431/9431

10	3282/3384.2/3486	3147/3273.7/3391	2841/2885.8/2981	2776/2781.2/2801	2776/2783/2815

最大迭代次数的作用与最低温度起到的作用基本相同，设置最大迭代次数的目的是防止模型的运行时间过长，也可以不设置最大迭代次数仅用最低温度来控制模型的结束。整体来看，最大迭代次数的提高会带来更优的解，但最大迭代次数提高到一个过大的值时已经失去了意义，因为此时算法会根据最低温度停止迭代。分析用例 0、4 可以发现，当迭代次数到达一定的值时，因为温度已经足够低，算法跳转到其他全局最优解的可能性也逐渐趋于 0，因此后面历次求解将得到相同的解。

3.3 遗传算法参数实验

遗传算法共四个参数可供调整，分别是：

(1) 种群大小 populationSize；

(2) 迭代次数 generations；

(3) 突变率 mutationRate；

(4) 精英留存率 eliteRate 。

3.3.1 种群大小参数对照

保持 generations = 200; mutationRate = 0.3; eliteRate = 0.4

改变种群大小，分别对测试用例 3、6、9 进行测试，每个用例进行 10 次实验，记录加工时长的最小值、平均值、最大值，记录格式为：最小值/平均值/最大值，测试结果见表格 6。

表格 6 种群大小参数对照实验

编号	populationSize =5	populationSize =10	populationSize =50	populationSize =100	populationSize =500	populationSize =1000
3	6666/6883.6/7310	6666/6666/6666	6655/6664.9/6666	6655/6664.9/6666	6655/6659.4/6666	6655/6657.2/6666
6	1434/1505.6/1531	1463/1491.6/1555	1379/1415.5/1447	1386/1419.5/1451	1374/1386.6/1403	1369/1384.9/1412
9	1831/1883.2/1926	1826/1858.8/1899	1766/1824.8/1883	1766/1798.6/1828	1746/1763.8/1795	1748/1768/1790

种群大小直接决定了解的多样性（即解的广度），从整体上看，种群越大求得的实验结果越好，但是种群的增大也会导致计算时间变长，因此取 500 左右的种群大小即可。如果时间允许，可以设置一个更大的种群大小用于求取潜在的更优解。

3.3.2 迭代次数参数对照

保持 populationSize = 300; mutationRate = 0.3; eliteRate = 0.4

改变迭代次数，分别对测试用例 2、5、10 进行测试，每个用例进行 10 次实验，记录加工时长的最小值、平均值、最大值，记录格式为：最小值/平均值/最大值，测试结果见表格 7。

表格 7 迭代次数参数对照实验

编号	generations=10	generations=50	generations=100	generations=500	generations=1000
2	5066/5067.6/5077	5066/5066/5066	5066/5066/5066	5066/5066/5066	5066/5066/5066
5	7044/7060.5/7089	7021/7021/7021	6961/6998.6/7044	6961/6967/6991	6961/6961/6961
10	3103/3144.9/3186	2854/2886/2932	2790/2841.3/2889	2776/2778.4/2782	2776/2777.5/2782

从实验结果可以看到，无论是最大值、最小值还是平均值，迭代次数的增加都会带来更好的实验结果，从算法原理上分析，每次实验都会保留一定的最优解，这就保证了至少实验不会随着迭代次数的增加得到更差的值，同时迭代次数的增加也会让算法在解空间中探索到更大的范围，有助于得到更优的结果。因此，在条件允许的情况下，迭代次数应取一个较大的值。

3.3.3 突变率参数对照

保持 populationSize = 200; generations = 300; eliteRate = 0.4

改变突变率，分别对测试用例 1、7、8 进行测试，每个用例进行 10 次实验，记录加工时长的最小值、平均值、最大值，记录格式为：最小值/平均值/最大值，测试结果见表格 8。

表格 8 突变率参数对照实验

编号	mutationRate =0.01	mutationRate =0.1	mutationRate =0.3	mutationRate =0.5	mutationRate =0.7	mutationRate =0.9
1	6269/6319.4/6332	6269/6269/6269	6269/6269/6269	6269/6269/6269	6269/6269/6269	6269/6269/6269
7	1875/1914.1/1955	1866/1880.2/1922	1861/1875.3/1891	1859/1867.1/1874	1859/1868.1/1875	1860/1865.4/1872
8	952/981.1/1014	938/953.3/965	932/942.8/961	932/940.8/961	932/943.1/952	932/939.9/952

突变率的提高有助于获得更好的实验结果。当突变率从一个较小值 0.01 变为 0.1 时，实验结果从三方面均得到了极大的提高。但此后继续提高突变率所获得的优化效果差别不是很大，保持突变率在 0.3 左右便能达到一个较好的优化效果。

3.3.4 精英留存率参数对照

保持 populationSize = 200; generations = 300; mutationRate = 0.3

改变精英留存率，分别对测试用例 0、4、10 进行测试，每个用例进行 10 次实验，记录加工时长的最小值、平均值、最大值，记录格式为：最小值/平均值/最大值，测试结果见表格 9。

表格 9 精英留存率参数对照实验

编号	eliteRate=0	eliteRate=0.1	eliteRate =0.3	eliteRate =0.5	eliteRate =0.7	eliteRate =0.9
0	7038/7335.4/7689	7038/7038/7038	7038/7038/7038	7038/7038/7038	7038/7038/7038	7038/7038/7038
4	10230/10389.4/10445	9431/9431.5/9436	9431/9431.5/9436	9431/9431/9431	9431/9440/9511	9431/9572.9/9882
10	2991/3105.9/3261	2779/2829.2/2886	2779/2806.3/2880	2776/2783.5/2809	2785/2813/2880	2796/2853.5/2895

将精英留存率从 0 提高到 0.1 时，实验效果在三个维度均获得了极大的提升，得到了更好的解，但继续提升时所得的结果差别不大，甚至在从 0.5 提高到 0.9 时实验结果越来越差了，从算法原理分析，精英留存率过高代表每次产生的子代数量就会减少，不利于解的多样性提高。精英留存率为 0 则会导致算法会丢失父代中的更优解，无法保证每次迭代后实验结果不会变差。因此，精英留存率保持在小于 0.5 但大于 0 的一个较小值更有利于得到更好的实验效果。

3.4 算法对比

3.4.1 算法最优加工完成时间与程序运行时间比较

十个例子下的模拟退火算法与遗传算法的实验对比结果见表格 10。

实验参数：

SA：initial temp: 10000, endTemp: 1e-30, coolingRate: 0.99, maxIterations: 100000

GA：populationSize: 500, generations: 500, mutation rate: 0.5, eliteRate: 0.4

表格 10 模拟退火算法与遗传算法实验对比结果

编号	模拟退火算法 最优加工完成时间	模拟退火算法程序 平均运行时间(ms)	遗传算法 最优加工完成时间	遗传算法程序 平均运行时间(ms)
1	6269	27.2	6269	1571.1
2	5066	32.9	5066	1806.9
3	6655	40.3	6655	2106.3
4	9431	58.3	9431	2756.7
5	7044	35.2	6961	1917.8
6	1360	81.1	1335	3573.1
7	1846	96.1	1848	4062.1
8	932	63.8	932	2988.5
9	1765	89.7	1746	3700.4
10	2776	169.3	2776	7090.6

对于测试用例 1、2、3、4、8、10，模拟退火算法与遗传算法得到的最优加工完成时间相同。

对于测试用例 7，模拟退火算法优于遗传算法，得到的最优加工完成时间更小。

对于测试用例 5、6、9，遗传算法优于模拟退火算法，得到的最优加工完成时间更小。

上面的算法对比结果，是对每个测试用例进行 10 次重复实验后得到的，最优加工完成时间取每 10 次试验后得到的最小值，程序平均运行时间(ms)取每 10 次试验后得到的平均值。

由于算法中若干过程的随机性，多次运行算法得到的优化结果并不会完全相同，两种算法都有可能在某次实验中随机产生出全局最优解，因此以上的对比实验结果仅供参考。

3.4.2 最优调度方案

最终十个例子的最优调度方案见表格 11，工件的编号是从 1 开始的。

表格 11 各样例最优调度方案

编号	最优调度方案	总加工完成时间
1	4, 2, 1, 5, 3	6269
2	4, 8, 6, 5, 2, 3, 7, 1	5066
3	7, 8, 6, 10, 9, 5, 4, 1, 3, 2	6655
4	3, 6, 13, 5, 11, 12, 14, 10, 8, 9, 15, 1, 2, 7, 4	9431
5	2, 1, 3, 6, 8, 7, 5, 4	6961
6	2, 13, 8, 11, 10, 15, 12, 9, 18, 3, 1, 17, 14, 4, 7, 6, 16, 5	1335
7	11, 7, 4, 13, 5, 12, 18, 2, 14, 10, 17, 16, 15, 1, 8, 9, 3, 6	1846
8	15, 4, 5, 13, 2, 10, 1, 8, 12, 7, 3, 14, 9, 6, 11, 16	932
9	12, 3, 16, 11, 15, 14, 9, 1, 2, 10, 8, 5, 6, 7, 13, 4	1746
10	8, 9, 31, 40, 35, 34, 27, 2, 5, 3, 6, 7, 37, 21, 22, 20, 16, 39, 17, 4, 12, 10, 25, 38, 1, 36, 24, 32, 15, 29, 30, 18, 13, 28, 11, 14, 33, 23, 19, 26	2776

样例 1 最优调度方案对应的甘特图见图 3：

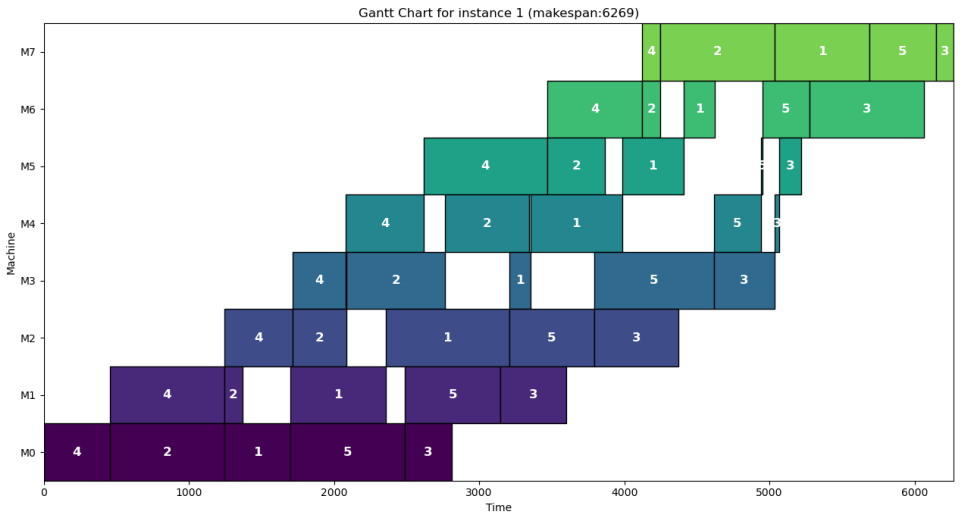

图 3 样例 1 调度方案

样例 2 最优调度方案对应的甘特图见图 4：

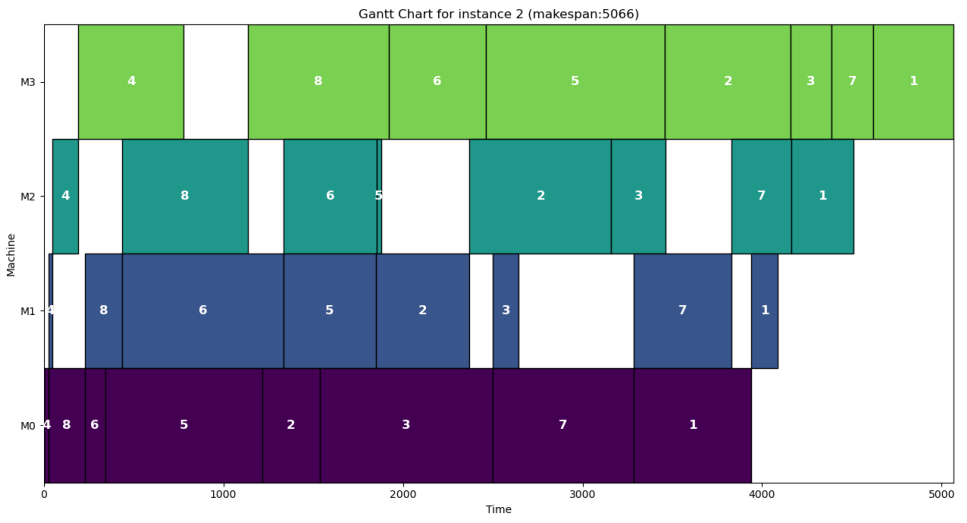

图 4 样例 2 调度方案

样例 3 最优调度方案对应的甘特图见图 5：

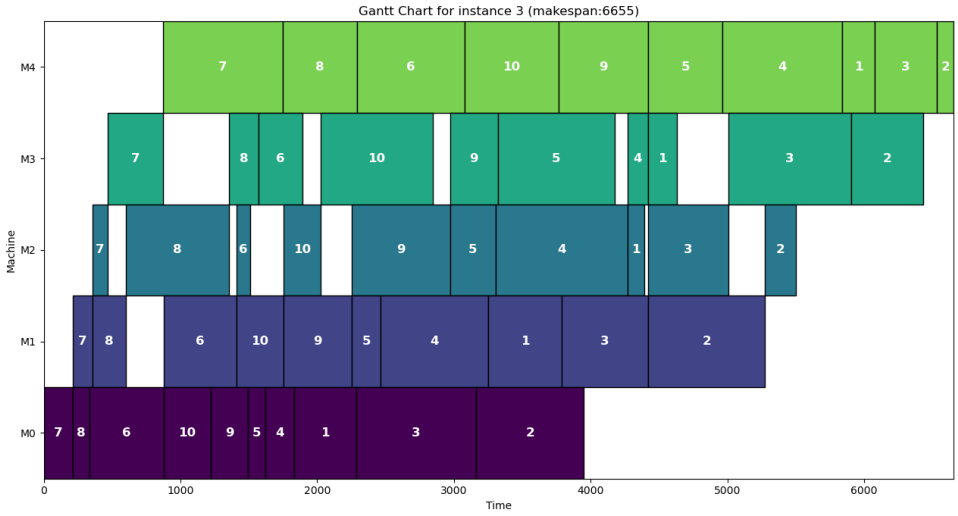

图 5 样例 3 调度方案

样例 4 最优调度方案对应的甘特图见图 6：

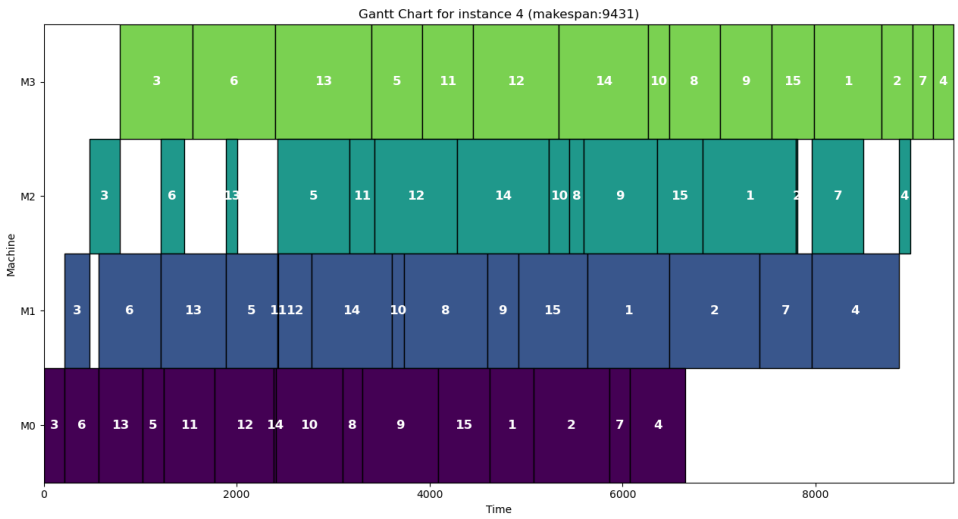

图 6 样例 4 调度方案

样例 5 最优调度方案对应的甘特图见图 7：

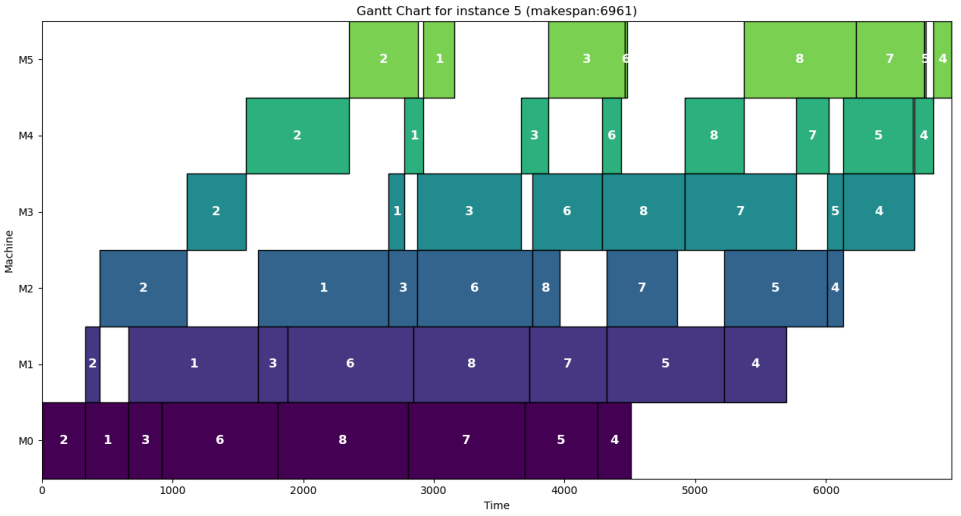

图 7 样例 5 调度方案

样例 6 最优调度方案对应的甘特图见图 8：

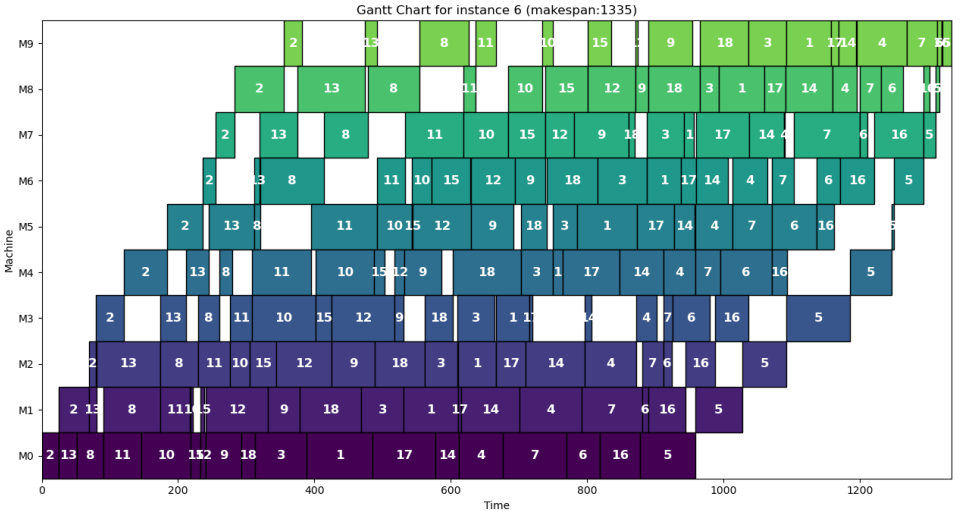

图 8 样例 6 调度方案

样例 7 最优调度方案对应的甘特图见图 9：

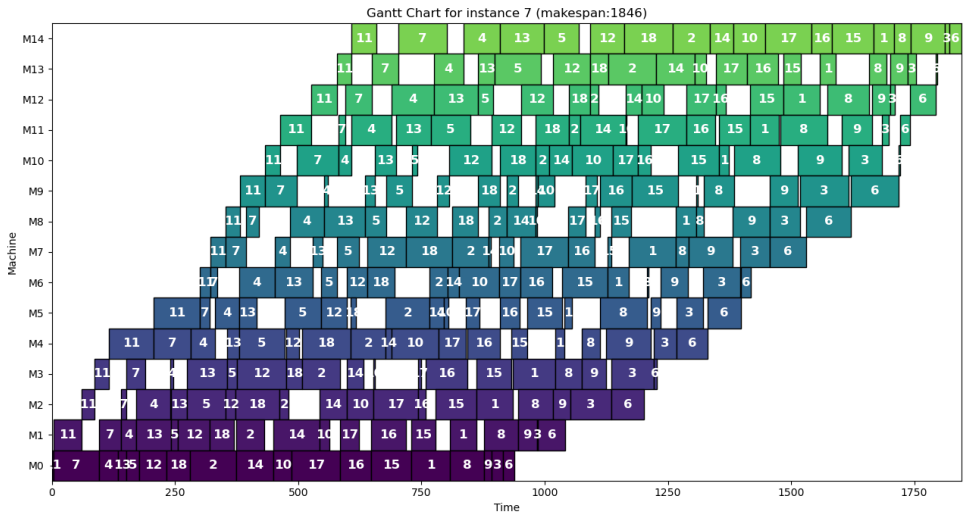

图 9 样例 7 调度方案

样例 8 最优调度方案对应的甘特图见图 10：

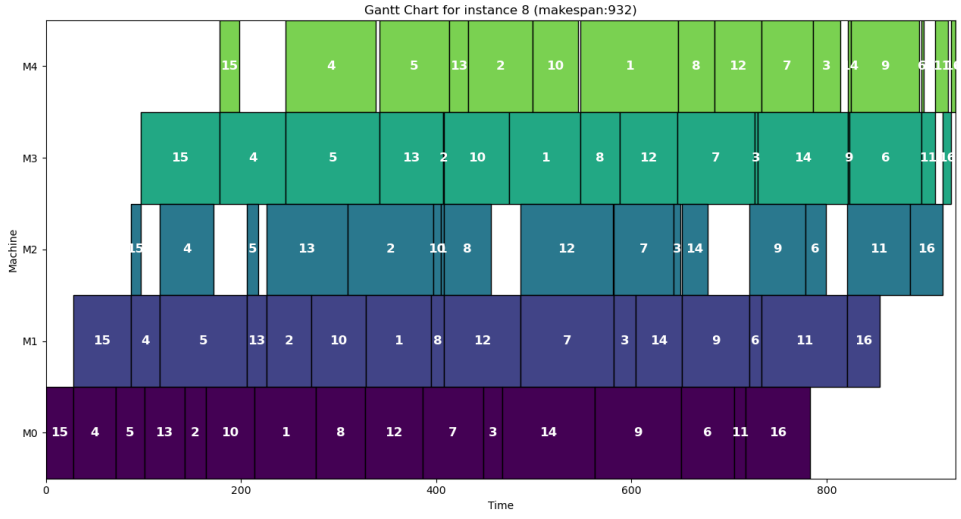

图 10 样例 8 调度方案

样例 9 最优调度方案对应的甘特图见图 11：

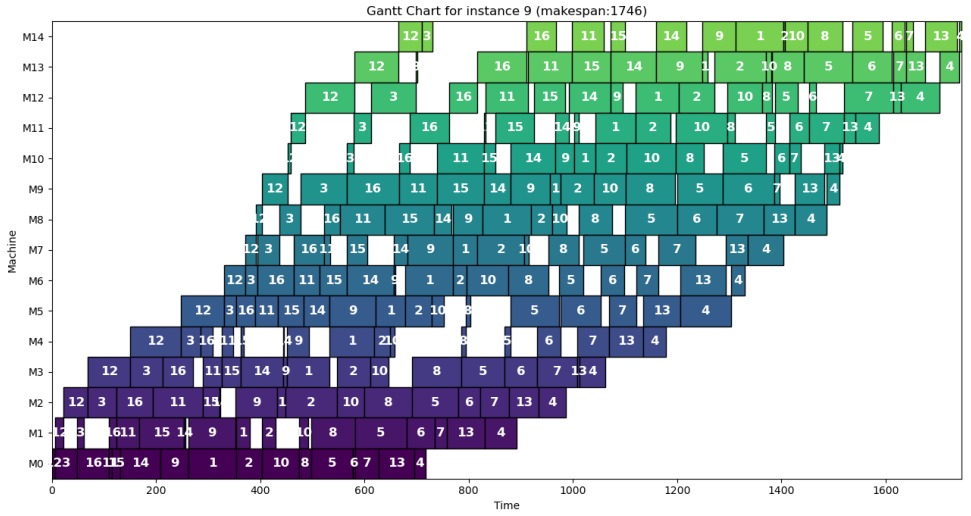

图 11 样例 9 调度方案

样例 10 最优调度方案对应的甘特图见图 12：

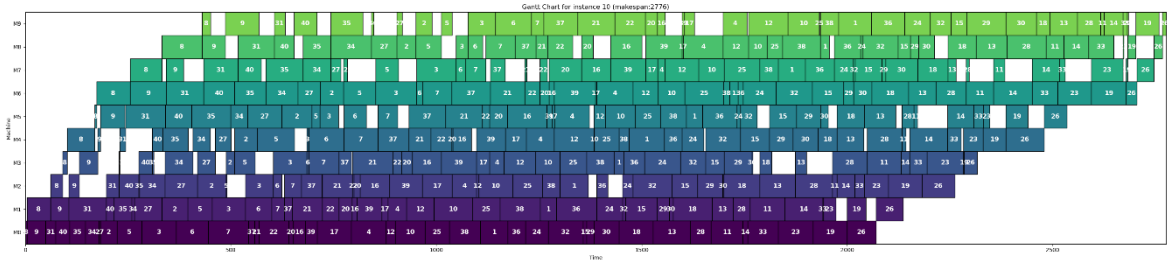

图 12 样例 10 调度方案

3.5 算法改进

对模拟退火算法进行改进：传统的模拟退火算法只能对 1 个解进行迭代，这样每次搜索的范围较小，容易被局限在局部最优解，可以将模拟退火算法扩展到同时对 n 个解进行迭代，最终输出 n 个解中的最优解作为优化结果。这样做可以保证优化完成时，所得结果至少不会比只对 1 个解进行迭代取得的结果更差，有利于得到更好的实验结果。n 为正整数，当 n=1 时退化为传统的模拟退火算法。下面取 n=10 进行实验，同时对 10 个解进行迭代，最终输出 10 个解中的最优解。在算法中用参数 numSchedules 表示 n。

针对 10 组测试样例，对每个样例分别运行 10 次传统的模拟退火算法、改进的模拟退火算法和遗传算法，得到 10 个例子下的三种算法的实验对比结果，结果见表格 12。

实验参数：

SA：initial temp: 10000, endTemp: 1e-30, coolingRate: 0.99, maxIterations: 100000

GA：populationSize: 500, generations: 500, mutation rate: 0.5, eliteRate: 0.4

iSA：initial temp: 10000, endTemp: 1e-30, coolingRate: 0.99, maxIterations: 100000, numSchedules: 10

表格 12 模拟退火、改进的模拟退火、遗传算法实验对比结果

编号	模拟退火算法 最优加工完成时间	改进的模拟退火算法 最优加工完成时间	遗传算法 最优加工完成时间
1	6269	6269	6269
2	5066	5066	5066
3	6655	6655	6655
4	9431	9431	9431
5	7021	6961	6961
6	1372	1342	1346
7	1858	1855	1842
8	932	932	932
9	1763	1746	1749
10	2776	2776	2776

对于测试用例 1、2、3、4、8、10，三种算法得到的最优加工完成时间相同。

对于测试用例 5，改进的模拟退火算法得到的最优加工完成时间优于传统的模拟退火算法得到的最优加工完成时间，与遗传算法最优加工完成时间相同。

对于测试用例 6、7、9，改进的模拟退火算法得到的最优加工完成时间优于传统的模拟退火算法得到的最优加工完成时间，但比遗传算法最优加工完成时间差。

可以看出改进的模拟退火算法确实获得了比传统的模拟退火算法更好的效果。

4 总结

本文针对 PFSP 问题，分别使用了模拟退火算法、遗传算法进行了求解，并尝试对传统的模拟退火算法进行优化。文章讲解了模拟退火算法、遗传算法的思想和关键操作，给出了两种算法的流程图，并根据算法的实现思路进行编程，针对 10 个测试样例进行了实际求解，给出了对应的最优调度方案和调度甘特图。分析实验结果发现，模拟退火算法求解的速度相对更快，并且在很多测试样例中都获得了不弱于遗传算法的实验结果，并且有时可能得到比遗传算法更优的结果。遗传算法求解速度较慢，但是对解空间的搜索更为全面，获得全局最优解的概率比模拟退火算法获得全局最优解的概率更大，在解空间巨大时更能体现出优势。

传统的模拟退火算法的缺点在于，每次只能对一个解进行迭代，如果全局最优解在温度较高时被搜索到，后续有很大概率被舍弃掉。为了解决这一问题，可以同时设置多个搜索进程，同时对多个解进行迭代。此外，还可以尝试寻找更好的邻域定义的方式、尝试不同的降温策略，都有可能获得更好的效果。

传统的遗传算法的缺点在于，由于种群规模较大，算法运行的时间较长，并且不同的选择策略、交叉策略、变异策略对最终的结果有很大影响。本文在最开始设计遗传算法时，新种群全部由新产生的个体组成，得到的结果很差，后来引入精英策略，每次都保留一部分最优的个体，最后达到了更好的效果。相应的，可以尝试使用动态的变异概率、不同的编码与交叉策略、不同的选择策略，都有可能会达到比传统的遗传算法更好的优化结果。此外，为了降低算法运行的时间，由于在后期时种群的最优个体已经基本不发生变化，可以提早结束迭代，通过尝试在不影响最优解的前提下向下调整种群大小与迭代次数，可以得到一个相对更短的程序运行时间。

参考文献:

[1] 顾瀚,王雷,蔡劲草,等.改进人工蜂群算法求解置换流水车间调度问题[J].井冈山大学学报(自然科学版),2023,44(04):82-89.

[2] 李小缤,白焰,耿林霄.求解置换流水车间调度问题的改进遗传算法[J].计算机应用,2013,33(12):3576-3579.