基于多种智能优化算法的车间调度问题研究

张荐科（学号：1120200784）

摘要：车间调度问题是指合理安排任务调度次序使得车间任务所需总时长最小的问题。它是典型的 NP 完全问题，常规解法往往不能在合理时间范围内得到较好的结果。登山算法、模拟退火算法、粒子群算法是经典的智能搜索算法，可以有效解决一定规模下的车间调度问题。本文建立了车间调度模型，使用三种优化算法解决模型提出的优化问题。通过实验调整参数最大程度优化算法计算得到的调度总时间以及相应的最佳调度顺序，同时比较了不同算法的搜索特点，发现在较大规模问题中，模拟退火算法和粒子群算法相较于登山算法效果更好，并对粒子群算法的离散化方式进行了讨论分析。

关键词：车间调度问题；模拟退火算法；登山算法；粒子群算法

一. 引言

1.1 问题背景

现代企业生产具有环节众多、关系复杂、连续性强等特点，工厂企业为了适应新形态的供应体系，通过内外资源合理的分配、缩短加工时间以及降低成本等方法来解决许多共性问题。其中一个重要方面——生产调度，就是要对有限的资源进行有效合理配置和优化。人们将这种工业问题需求抽象化建模成为车间调度问题（下简称 JSP），指在工件加工流程、顺序、时间成本要求等约束下，确定最小加工时间和相应的每个工件的加工路径。车间调度问题直接关系着企业的生产成本和效率，在制造行业中发挥着举足轻重的作用，具有重要的理论意义和实际价值。

1.2 问题的形式化描述

车间调度问题可以如下描述：某工厂有 n 个工件，记为 $P_1, P_2, \cdots, P_n$，每个工件都需要在 m 台机器上进行流水加工并且加工顺序可以改变，记 m 个机器为 $M_1, M_2, \cdots, M_m$。同一时间一台机器只能加工一个工件，每个工件需要每台机器加工一次，加工需要一定时间，使用矩阵 $T_{m\times n}$ 记录每个工件在每个机器上的加工时间，$T_{i,j}$ 表示第 i 个工件 P_i 在第 j 台机器 M_j 加工所需要的时间。为了减小问题的搜索空间，人为要求每台机器加

工的零件顺序都一致，用一个长度为 n 的向量 $\vec{o}$ 表示这个顺序。问题所求的是完成所有工件加工的最小时间 t_{min} 以及对应的最优加工顺序 $\vec{o}_{best}$。

这是一类经典的组合优化问题，为了方便描述，总结列举以下编号规则：

- 工件的编号默认为 $1, 2, \cdots, n$，表示 $P_1, P_2, \cdots, P_n$ 工件
- 由于每个工件经历的机器加工次序相同，因此默认此顺序为机器编号顺序，即每个工件依次由 $M_1, M_2, \cdots, M_m$ 加工，用它们的下标 $1, 2, \cdots, m$ 编号表示
- 由于进入机器的工件顺序可变，但所有机器的加工顺序都相同，因此用工件下标组成的向量统一表示这个顺序：$\vec{o} = \{o_1, o_2, \cdots, o_n\}$（是一个 $1, 2, \cdots, n$ 的重排列）

表 1-1 关键符号说明

数学符号	意义
$S_{i,j} \quad (1 \le i \le n, 1 \le j \le m)$	第 j 台机器 M_j 开始加工第 i 个工件 P_i 的时间
$o_k \quad (1 \le k \le n)$	同顺序约束下每个机器第 k 个加工的零件 P_{o_k}
$T_{i,j} \quad (1 \le i \le n, 1 \le j \le m)$	第 j 台机器 M_j 加工第 i 个工件 P_i 所需要的时间

将文字描述转化为数学语言如下：

(1) 约束：一个工件同一时间只能在一个机器上加工 $\Rightarrow S_{o_i,j+1} - S_{o_i,j} \ge T_{o_i,j}$

(2) 约束：每个机器同时只能加工一个工件 $\Rightarrow S_{i,j} - S_{k,j} \ge T_{k,j}$ 或 $S_{k,j} - S_{i,j} \ge T_{i,j}$

(3) 约束：工件 i 在机器 m 上加工需要一定时间 $\Rightarrow S_{o_{i+1},j} - S_{o_i,j} \ge T_{o_i,j}$

(4) 优化目标：最后一个工件完成时间最短 $\Rightarrow t = \min(S_{o_n,m} + T_{o_n,m})$

加入边界条件和下标范围，得到完整的问题形式化描述如下：

$$\min \ t = (S_{o_n,m} + T_{o_n,m})$$

$$s.t. \begin{cases} S_{o_1,1} \ge 0 & \\ S_{o_i,j+1} - S_{o_i,j} \ge T_{o_i,j}, & i = 1, 2, \cdots, n, \quad j = 1, 2, \cdots, m \\ S_{i,j} - S_{k,j} \ge T_{k,j} \text{ 或 } S_{k,j} - S_{i,j} \ge T_{i,j}, & i, k = 1, 2, \cdots, n, \quad i \ne k \quad , j = 1, 2, \cdots, m \\ S_{o_{i+1},j} - S_{o_i,j} \ge T_{o_i,j} & i = 1, 2, \cdots, n-1, \quad j = 1, 2, \cdots, m \end{cases} \tag{1.1}$$

1.3 研究现状与解决方案

对于流水车间调度问题的研究，可以大致分为传统经典方法和智能优化方法两大类。传统的研究方法一般为最优化方法，主要包括分支定界法，动态规划法和拉格朗日

松弛算法。随着社会需求的增长，车间环境愈发，计算难度爆炸式增长，传统的研究方法难以应对这类问题，于是国内外学者开始研究现代智能化算法来解决车间调度问题。

智能化算法是在朴素搜索基础上演化而来的。由于遍历搜索算法复杂度过高，人们开始设计启发式的方法优化搜索过程，往往可得出一类比较好的结果。这些算法模拟自然现象的演变过程，在传统算法难以处理的大规模优化问题中具有自适应、自学习、高效率等优良特点，可以有效利用在流水车间调度问题，如粒子群算法、遗传算法、蚁群算法、禁忌搜索算法、人工免疫算法、微粒群算法、模拟退火算法等等 [1]。

本文基于前人的研究思路，分别采用登山算法、模拟算法和粒子群算法作为问题求解的主体优化算法，并针对问题设计算法的数据读取和处理接口，在参数调试得到较好的实验结果基础上进行算法的优劣比较，主要涉及了以下几个步骤：

(1) **建立模型**：针对加强约束后的流水车间调度问题给出形式化描述，理清变量关系和约束条件。
(2) **设计接口算法**：设计计算一定加工顺序的总加工时间的算法和产生新解的方式，为主体求解算法提供接口。
(3) **设计优化算法**：将经典登山算法、模拟退火算法加以完善，作为问题的主体优化算法。针对适用于连续性优化问题的粒子群算法进行模型改进，使之能够用于求解置换流水车间类型的离散组合优化问题。
(4) **编写程序并试运行**：实现算法并输出计算结果，进行相应的可视化。
(5) **调试参数**：调试每种优化算法的参数，分别得到美中算法较好的结果。
(6) **对比实验**：比较记录不同参数、不同智能优化算法的结果，分析参数对于算法的影响。针对粒子群算法的离散化方式进行实验分析，尝试寻找更好的离散化编码方式。

最终，论文得出了一组很好的测试数据的优化结果 (使用 SA 算法的结果作为本实验对所给 11 个用例的最终计算结果)。同时，在对比实验中探索了算法参数对结果的影响程度和定性原因，发现在模拟退火算法中加入同温度多循环会大大提升优化效果，总体上好于传统的外层增大退火次数等改进方法。另外，论文发现随着问题规模的增大，模拟退火算法和粒子群算法的效果将远超过登山算法。最后，针对粒子群算法，实验发现在小规模问题上 PSO 算法并不能保证很好的优化结果，但是在更大规模问题上相较于其他算法具有更高的时间效率，能够在同等时间下获得更好的优化结果。

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。

二. 算法设计

2.1 思路简介

由于 JSP 是 NPC 问题，难以找到常规的多项式时间复杂度算法加以求解，因此求解的思路核心便是搜索。搜索可以细分为很多类型，除了暴力的遍历搜索外，有很多智能搜索算法可以加速搜索过程以提高搜索效率。搜索算法本质是就是对解空间的扫描，不同的算法有着不同的跳转查找方式。为了找到合适的解决方案，实验中使用登山算法、模拟退火算法作为主要的搜索模块。另一方面，为了让搜索算法能够应用在车间调度问题中，还应该设计多个配套的计算模块，主要用于为搜索算法提供领域解生成和优化目标结果的计算。它们将作为求解算法主体的搜索算法的接口，提供数据更新和结果评估。

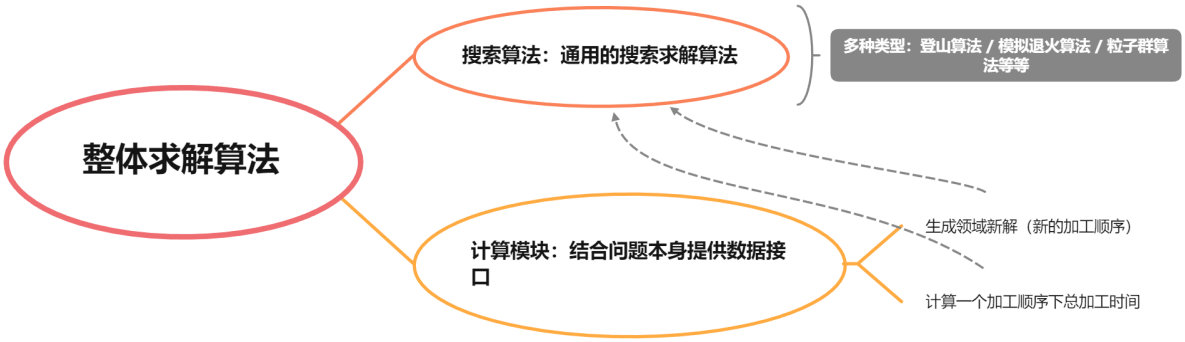

图 2-1 整体算法结构

综上，解决 JSP 问题的整体算法将由图2-1中的几个模块组合形成，其中搜索模块可以用各种智能搜索算法替代。2.3、2.4节将主要介绍登山算法和模拟退火算法，2.5节将介绍常规的粒子群算法以及能够用于解决离散化问题的算法改进。

2.2 基本计算模块

这一部分中需要设计算法完成智能搜索算法需要的两个模型接口，其一是**根据已有解随机生成一个领域新解**，即由一个已有加工顺序生成一个新的顺序，另一个则是**根据解计算结果**，也就是根据新的加工顺序计算完成加工的时长。

(1) **随机生成领域新解算法**思想是随机交换原有顺序向量中两个元素的位置作为新的加工顺序。这个操作可以很容易地通过 C++ 中 random 函数实现，直接对数组进行操作即可。

Algorithm 1 Generate new neighborhood solutions randomly

Input: n: Number of artifacts, ans_old: original solution vector (array of length N)

Output: ans_new: new neighborhood solution vector

```
a=rand()%n, b=rand()%n
while a == b do
    b=rand()%n
end while
for i = 0 to n − 1 do
    ans_new[i]=ans_old[i]
end for
swap(ans_new[a],ans_old[b])// change the order
return answer_new
```

时间复杂度计算：复制操作时间复杂度 $O(n)$, 寻找两个不同的随机数时间复杂度可以如下计算：

$$P(\text{循环 k 次才能找到与 a 不同的 b}) = \frac{1}{n^{k-1}} \cdot \frac{n-1}{n} = \frac{n-1}{n^k}$$

$$\begin{aligned} E(k) &= \sum_{k=1}^{\infty} k \cdot \frac{n-1}{n^k} \\ &= n(n-1)\sum_{k=1}^{\infty} \frac{k}{n^{k+1}} = n(n-1)\frac{d}{dx}(\sum_{k=1}^{\infty} \int \frac{k}{x^{k+1}} dx)|_{x=n} = n(n-1)\frac{d}{dx}(\sum_{k=1}^{\infty} -\frac{1}{x^k})|_{x=n} \\ &= n(n-1)\frac{d}{dx}(-\frac{1}{x-1})|_{x=n} = \frac{n}{n-1} \sim O(1) \end{aligned}$$

所以生成新解的总时间复杂度为 $O(n)$。空间复杂度为存储新解所使用的空间，亦为 $O(n)$。

(2) 根据已有**加工顺序计算总加工时长**则需要对整个加工过程进行模拟，具体计算思路如下。

考虑1.2节中给出的约束条件1.1，由于所有机器都以相同的顺序依次加工各个零件，为了让总时长最小，应该避免可加工的工件和对应的机器同时停歇的情况，并尽早地开始加工。因此使得总时间最小的调度方案一定满足以下特点：(a) 一个机器 (M_j) 开始加工一个工件 (P_{o_i}) 的条件为该机器 (M_j) 已完成上一个工件 ($P_{o_{i-1}}$) 的加工并且该工件 (P_{o_i}) 已在上一台机器 (M_{j-1}) 完成加工；(b) 第一台机器在工件释放开始时就开始加工第一个工件。基于以上分析，约束条件的实现可以通过以下**动态规划**的转移函数实现，为了方便代码实现和结果的输出，使用每台机器的工件加工结束时间矩阵记录加工过

程。（与上文中给出的 $S_{i,j}$ 的效果完全相同，有：$F_{i,j} = S_{o_i,j} + T_{o_i,j}$）

表 2-1 补充符号说明

数学符号	意义
$S_{i,j}$ $(1 \le i \le n, 1 \le j \le m)$	第 j 台机器 M_j 开始加工第 i 个工件 P_i 的时间
$F_{i,j}$ $(1 \le i \le n, 1 \le j \le m)$	第 j 台机器 M_j 结束加工调度顺序中第 i 个工件即 P_{o_i} 的时间

根据上述分析给出**计算总时间算法的转移函数：**

$$F_{i,j} = \begin{cases} \max(F_{i,j-1} + F_{i-1,j}) + T_{o_i,j} & , i = 1,2,\cdots n, \quad j = 1,2,\cdots,m \\ 0 & , i = 0 \text{ 或 } j = 0 \end{cases} \tag{2.1}$$

根据转移函数，给出算法 2 如下所示。根据循环次数，可以判断该算法的时间和空间复杂度都是 $O(mn)$。

Algorithm 2 Calculate the total processing time

Input: order: Processing order, T: processing time data table , n: number of workpiece, m: number of machine

Output: Total processing time

for $i = 0$ to $n-1$ **do**
  for $j = 0$ to $m-1$ **do**
    if $i == 0$ and $j == 0$ **then**
      $F(i,j) = 0$
    else if $i == 0$ **then**
      $F(i,j) = F(i,j-1) + T(order[i],j)$
    else if $j == 0$ **then**
      $F(i,j) = F(i-1,j) + T(order[i],j)$
    else
      $F(i,j) = max(F(i-1,j), F(i,j-1)) + T(order[i],j)$
    end if
  end for
end for
return $F(n-1, m-1)$

2.3 登山算法

登山算法又称爬山算法 (Hill Climbing)，是一种局部搜索算法，本质上利用了最优极值点附近如 “山峰” 一样的特点，让搜索沿着 “山坡” 一路向上进行快速搜索，从而避免暴力遍历的盲目性。具体一点来讲，算法每次从当前解的临近解空间中选择一个最优解作为当前解，让遍历目标在目标函数不断增加的方向上连续移动以找到问题的最佳极值点。当它达到一个峰值，没有邻居有更高的值时，它就会终止。当然，这种贪心策略带来的无回溯也导致了算法很容易陷入局部最优，很可能没有找到比它更好的其他更优解，这样的局限性在2.4节提到的模拟退火算法中得到了很大程度的改善。

本实验中采用的登山算法的关键操作流程包含以下三个部分：

- **产生随机初始解**：对于每个机器上的工件加工顺序随机排序，可以使用 random.shuffle 方法直接打乱顺序数组。
- **寻找邻域最优解**：遍历当前解的邻域，寻找临近解空间的最优解，注意：这里的生成新解算法不能直接使用 2.2 节给出的随机生成邻域新解的算法，因为登山算法的每一步是向确定的最优方向寻找，而不是随机选择邻域解进行比较。
- **迭代更新条件**：设置内外两层循环，内层循环为常规的登山步骤，用于一步步逼近局部最优解，只有当迭代达到设置的次数上限或者已经到达局部最优，迭代才终止；外层循环用于多次尝试产生不同的初始解，增大找到更好的局部最优解或全局最优解的可能。

Algorithm 3 登山算法 Hill-Climbing

Input: iter_num: 迭代次数

Output: 搜索到的最优解

for $i = 0$ to iter_num **do**
    随机生成一个初始解
    while 循环达到搜索次数上限或者解没有进行更新 **do**
        遍历解邻域所有新解，选择最优的作为新解
    end while
    更新并记录最优解
end for
return 最优解

登山算法的时间复杂度主要由循环体内的选择邻域最优解决定，记最外层的随机初

始解尝试次数$iter_num$为C_{out}，内层更新搜索次数上限为C_{in-max}，每次邻域搜索需要尝试C_n^2次不同的顺序交换，并计算其对应的解，因此单次循环需要$O(C_n^2 \times mn)$~$O(mn^3)$的时间。中层循环次数由初始生成解距离局部最优位置距离和C_{in-max}决定，前者大小无法估计，因此取后者作为时间复杂度的上限进行估计，故登山算法总时间复杂度小于$O(C_{out} \times C_{in-max} \times mn^3)$。

考虑算法的空间复杂度，由于循环的过程需要原地更新存储长度为n的解向量，并且计算总时间使用的大小为$m \times n$的矩阵和其他个别临时变量，每次循环都进行原地更新，因此总空间复杂度为$O((m+1)n)$。

2.4 模拟退火算法

模拟退火算法 (Simulated Annealing) 是一种基于概率的算法，它模拟固体退火过程，从一个初始的高温度出发，在温度降低的同时以一定概率跳脱当前寻找的解，避免陷入局部最优从而寻找目标函数的全局最优解，当温度降低并趋于稳定，得到的解也逐渐趋于稳定，逐渐收敛在可能的全局最优的位置。与登山搜索的按照结果上升搜索不同，模拟退火算法因为有概率跳脱的机制，能够更好地避免登山搜索陷入局部最优的弊端。

本文采用的模拟退火算法的关键操作流程包含以下几个部分：

- **产生随机初始解**：对于每个机器上的工件加工顺序随机排序，可以使用 random.shuffle 方法直接打乱顺序数组。
- **多层搜索循环**：最外层设置比较小的循环，用于多次尝试不同的初始解，防止初始生成位置使搜索陷入局部最优；中间层为正常的温度衰减循环，实现跳脱概率由高到低的搜索过程；最内层为用于提高同一退火温度下的搜索次数，一定程度增大搜索量，提高搜索精度。
- **新解生成**：随机选择某两个工件，记录它们的编号然后交换原解中这两个工件的顺序作为新的领域解。
- **优化目标计算**：计算新解对应的优化目标的值，用于判断是否保留或跳脱当前解。
- **局部跳脱**：设置判断条件，按照退火温度的指数概率选择新的非更优解。

为了更清晰地表明论文采用的模拟退火的多层循环设置，使用图2-2详细地表述算法：

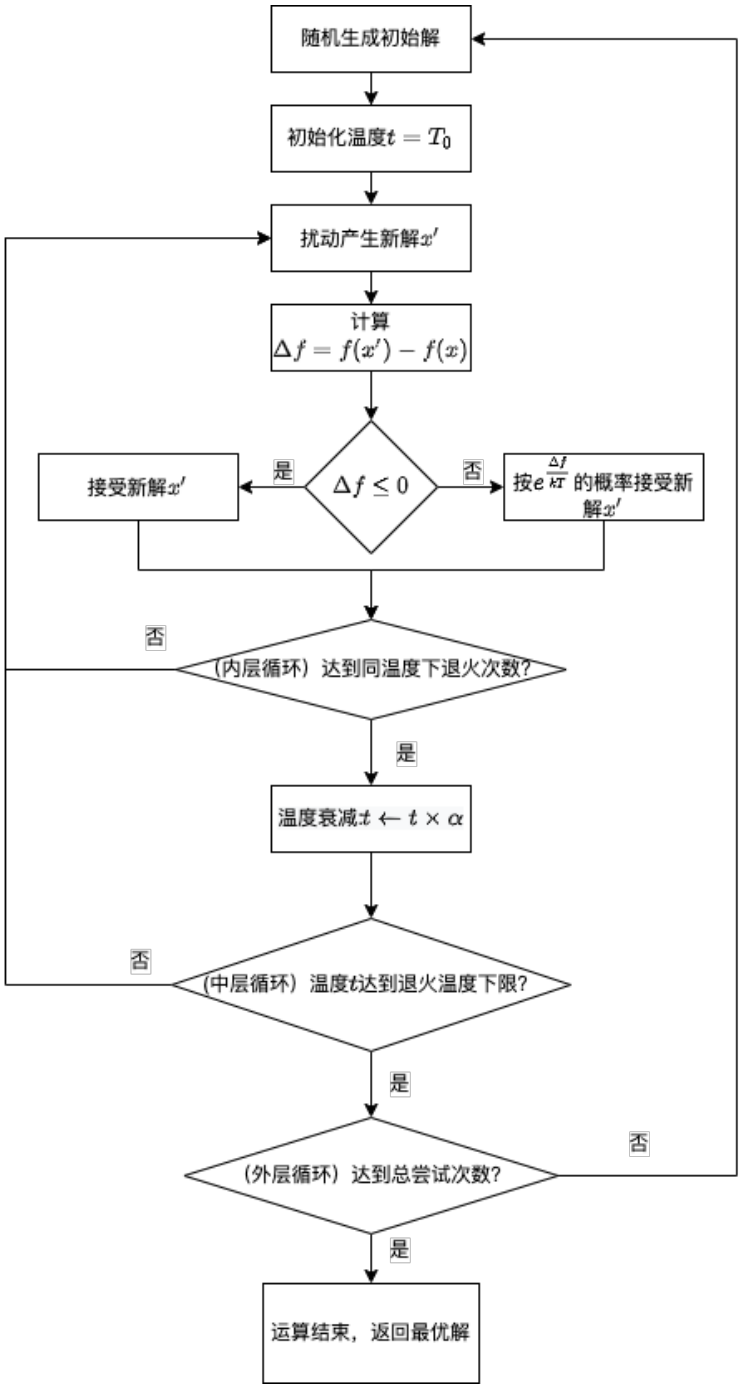

图 2-2 SA 算法流程图

该算法的时间复杂度主要由三重循环的次数决定，假设最外层循环次数为 C_{out}、中间层循环次数为 C_{mid}（实际上 C_{mid} 由退火初始温度 T_0，下限温度 T_d，温度衰减系数 α 决定：$C_{mid} = log_{\alpha}\frac{T_d}{T_0}$），最内层同温度循环 C_{in}，那么总搜索次数即为 $C_{out} \times C_{mid} \times C_{in}$，每次循环都需要进行一次生成解和求总加工时间，这两者的时间复杂度之和为 $O((m+1)n)$，因此模拟退火算法的总时间复杂度为：$O(C_{out} \times C_{mid} \times C_{in} \times (m+1)n) = O(C_{out} \times log_{\alpha}\frac{T_d}{T_0} \times C_{in} \times (m+1)n)$。

考虑算法的空间复杂度，由于循环的过程需要存储长度为 n 的新解向量，计算总时间时使用了大小为 $m \times n$ 的矩阵和其他常数个变量，并且在每次循环时都进行原地更新，因此总空间复杂度为 $O((m+1)n)$。

2.5 粒子群算法

解决如置换车间调度问题的 PSO 算法需要对原算法进行离散化的改进，在2.5.1节中介绍基本的粒子群算法及相关关键步骤原理，在2.5.2节中介绍本文采用的解的离散化编码方式，并给出了可用于置换流水车间问题求解的完整 PSO 算法流程图及复杂度分析。

2.5.1 连续问题的 PSO 算法

粒子群算法 (Particle Swarm Optimization, PSO) 是一种模拟自然界中鸟类觅食行为的进化算法 [4]。算法生成一组随机遍布于解空间的粒子 (称为种群)，种群中的粒子将按照一定的规则，在每次迭代中更改速度位置搜索最优解。将每个粒子的位置延伸为 N 维空间，第 i 个粒子在 N 维空间的位置表示为向量 $X_i = (x_1, x_2, \cdots, x_N)$，飞行速度表示为矢量 $V_i = (v_1, v_2, \cdots, v_N)$. 每个粒子都有一个由目标函数决定的适应值，并且记录了自身从迭代开始到目前为止发现的最好位置 pbest 和现在的位置 X_i。同时，每个粒子还会根据目前整个群体中所有粒子发现的最好位置 gbest 更新自己的下一步位置和速度。整个过程可以看作是粒子在种群经验和自身经验不断“进步”的过程。

具体来说，粒子群算法的关键操作包含以下几个部分：

- **产生随机初始粒子**：设置一个给定的上下限，根据上下限随机生成每个粒子的位置和速度向量。
- **迭代更新粒子/粒子群最优值**：给定循环次数，计算每个粒子的局部最优 pbest，并由此选择全局最优 gbest。
- **粒子群更新**：按照公式2.2更新每个粒子的位置和速度向量 [5]。

惯性因子: ω,　　学习因子: c_1, c_2

$$
\begin{aligned}
V_i^{k+1} &= \omega \times V_i^k + c_1 \times rand(0,1) \times (pbest_i^k - X_i^k) + c_2 \times rand(0,1) \times (gbest^k - X_i^k) \\
X_i^{k+1} &= X_i^k + V_i^{k+1}
\end{aligned}
\tag{2.2}
$$

其中对 V 的更新中第一项 $\omega \times V_i^k$ 表示迭代中自身速度的衰减，使粒子具有一定记忆性，一般取 $\omega = 1$；第二项 $c_1 \times rand(0,1) \times (pbest_i^k - X_i^k)$ 称为“自我认知部分”，c_1 的值表现了粒子对自身搜索的依赖性，如果 c_1 太低，那么所有粒子将会趋同一个粒

子, 失去种群多样性, 从而无法跳脱局部最优; 第三项 $c_2 \times rand(0,1) \times (gbest^k - X_i^k)$ 称为 “社会经验部分”，c_2 的值表现了粒子对群体的依赖性，如果 c_2 太低，那么所有粒子将独自运行，失去社会共享，导致收敛极慢。

- **粒子边界限制**：将更新后位置越界的粒子重新随机初始化，并设置粒子速度的上限，加速收敛速度。

一般来说，粒子群算法可以在连续的解空间中高效地寻找全局最优解，并且并不需要目标函数具有可导性。这一点使得 PSO 也可以用于离散型问题的求解。针对不同的离散型优化问题，人们设计了多样的 DPSO 算法对离散解进行编解码，同时对更新函数2.2进行了一定修改。针对本文研究的车间调度问题，2.5.2节中给出了一种简单的离散化方式，后续实验中证明了该方法的有效性并将其与其他方法进行了讨论分析。

2.5.2 针对车间调度的 DPSO 算法设计

考虑到常规 PSO 算法是在连续的解空间 $X_i \in \mathbb{R}^n$ 进行搜索的 (每一步的位置速度均可取连续的值)，而在置换流水车间调度问题中，解空间是一组大小为 $n!$ 的离散值，即 X_i 中的元素是 $1, 2, \cdots, n$ 的重排列。

为了能够使解决连续优化问题的 PSO 算法应用于车间调度的离散问题, 可以采用一种最直观的离散值到连续值映射的方式进行处理 (在当前更多的 DPSO 研究中，人们通常将映射处理方法称为“编码”)。本文采用的编码方法思想如下: 将原本 PSO 算法的每个维度的坐标表示这个维度所代表工件的优先级概率，其值越大，表示这个维度代表的工件约先加工。比如假设 $gbest = (5.2, 3.3, 7.4, 2.6, -5.9)$, 那么它对应的车间调度解 $o_{best} = (3, 1, 2, 4, 5)$

这种编码方法实际上就是对 PSO 算法的解进行降序排序，并去其排序后元素在原来向量中的位置的索引组成的向量作为车间调度的解。begin 根据上述离散化方法，给出最终的用于解决车间调度问题的离散化 PSO 算法如下:

Algorithm 4 针对车间调度问题的粒子群算法

Input: N: 种群粒子数量, n: 坐标维度 (即工件数量), X_{max}/X_{min}: 解空间上下限, ω: 惯性因子, c_1, c_2: 学习因子

Output: 搜索到的最优加工顺序

for $i = 0$ to N **do**
    根据上下限随机初始化每个粒子的位置和速度
end for
while 循环达到搜索次数上限或者 gbest 的目标值小于一定值 **do**
    for $i = 0$ to N **do**
        将粒子 i 的坐标 X_i 按降序索引转化为解顺序 o_i
        根据 o_i 计算加工时间作为该粒子的适应值
        更新 pbest
    end for
    遍历所有粒子，选择最优适应值粒子位置作为 gbest
    for $i = 0$ to N **do**
        根据公式2.2更新粒子的速度和位置
    end for
end while
return 最优解的降序索引 o_{best}

该算法主要包括两层循环，外层使循环搜索次数 C，内层是粒子数量 N，每一次更新操作都需要对粒子的每一维度进行计算，复杂度为 $O(n)$, 转化为最优解时需要对向量进行排序，复杂度为 $O(nlogn)$, 计算适应值需要 O(mn), 因此总时间复杂度为 $O(CNmn^2log(n))$。在 n 逐渐变大时，可以看到 PSO 算法的时间复杂度上涨会显著慢于模拟退火算法，具有更高的时间效率。

相应地，PSO 算法需要记录全部 N 个粒子的两个 n 维向量 (速度和位置)，在计算加工时间时需要开辟 $O(mn)$ 的临时空间，因此总空间复杂度为 $O(mn + Nn)$。

三. 实验

3.1 实验设置

(1) 实验环境：（MacOs/Windows 11 操作系统）

- 算法实现：Visual Studio Code 开发环境、C++ 语言
- 绘图实现：Visual Studio Code 开发环境、Python 语言、Matplotlib 及 Numpy 库

(2) 通用参数：

- 最大工件数量：50；最大机器数量：50
- 运行时间需求：单个用例运行时间不超过 5 分钟

(3) 登山算法实验参数：

- 最外层循环（多次尝试不同初始解次数）C_{out}: 2500
- 内层更新搜索次数上限 C_{in-max}: 10000

(4) 模拟退火算法实验参数：为了得到最好解，针对不同用例设置不同参数

表 3-1 模拟退火算法实验参数

用例	初始温度 T_0	结束温度 T_d	衰减系数 α	外层循环次数 C_{out}	内层循环次数 C_{in}
0	1e4	1e-7	0.99	1	1000
1	1e4	1e-7	0.99	1	1000
2	1e4	1e-7	0.99	1	1000
3	1e4	1e-7	0.99	1	1000
4	1e4	1e-7	0.99	1	1000
5	1e4	1e-7	0.99	1	1000
6	2e5	1e-7	0.99	1	1000
7	2e5	1e-6	0.998	1	3000
8	1e4	1e-4	0.99	1	1000
9	1e7	1e-4	0.998	200	30
10	1e4	1e-7	0.99	1	1000

(5) 粒子群算法实验参数：主要作为算法效果对比，没有进行详细调参，故采用统一参数。

- 最外层循环（多次尝试不同初始解次数）C: 10000
- 粒子数量 N: 50
- 坐标上下界 X_{max}: 100, X_{min}: -100
- 参数 $\omega = c_1 = c_2$: 1

3.2 实验结果

3.2.1 运行结果

对实验涉及的 11 个用例分别采用模拟退火算法 (SA)、登山算法 (HC)、粒子群算法 (PSO) 调试各自参数至满足计算时间需求的最佳状态（表3-1已经给出对应结果所需要的全部参数），得到最终 11 个用例的运行结果见表3-2。另外，为了更加直观地展示算法计算最优解的加工顺序，在图3-1中画出部分用例最优结果的加工时序甘特图。

表 3-2 全部实验结果

用例	SA 结果	SA 运行时间/s	HC 结果	HC 运行时间/s	PSO 结果	PSO 运行时间/s
0	**7038**	1.35	7038	0.30	7038	4.18
1	**8366**	1.69	8457	0.18	8366	3.00
2	**7166**	1.13	7166	0.47	7166	4.45
3	**7312**	1.46	7312	0.52	7312	4.30
4	**8003**	1.19	8003	0.43	8003	5.48
5	**7720**	1.53	7720	0.29	7720	3.53
6	**1431**	5.72	1431	9.12	1437	8.44
7	**1950**	127.80	1964	13.07	1973	8.42
8	**1109**	1.56	1109	3.25	1150	8.43
9	**1902**	243.16	1918	15.50	1908	9.08
10	**3277**	11.15	3277	153.41	3321	20.18

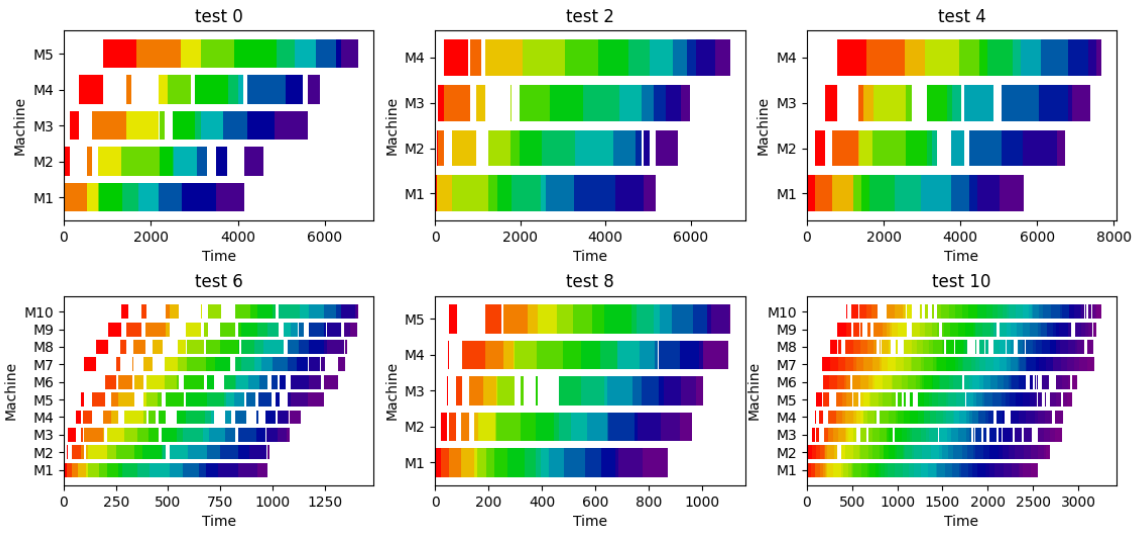

图 3-1 部分用例最优调度时序图

观察表3-2，SA 算法结果（最小完工时间）都要低于登山算法并且差距比较明显。可以大致确定 SA 的优化效果要显著好于 HC 算法，这说明前者使用概率跳脱避免陷入局部最优的启发式策略是十分有效的。为了进一步说明参数变化对 SA 优化策略效果的影响，在3.2.3-3.2.6节中采用参数对比实验进行各类参数影响的详细讨论。兼顾快速计算不同参数的实验结果和搜索难度，本文采用两组耗时和数据规模均适中的用例。3.2.3-3.2.6节中的对比实验取用用例 6 和 10 作为测试数据（其机器数量和工件数量均较多并且 SA 算法运行时间相对适中），用例参考的最佳调度时间分别为 1431、3277。对比实验中进行单变量试验，除对比的参数外，所有控制变量的默认值如下（之后不再赘述）：

表 3-3 对比实验默认参数表

默认用例	初始温度 T_0	结束温度 T_d	衰减系数 α	外层循环次数 C_{out}	内层循环次数 C_{in}
6	2e5	1e-7	0.99	1	1000
10	1e4	1e-7	0.99	1	1000

3.2.2 登山算法参数实验：外层循环次数影响

登山算法的参数量很小，可调部分只有外层循环次数 C_{out} 和内层迭代次数上限 C_{in-max}，但是实际上单次登山搜索只在解空间的非常小的区域内进行，一般来说几十次就已经到达局部最优了，因此 C_{in-max} 的设置对于算法效果影响微乎其微。想要避免初始解的随机性，就需要更改 C_{out} 多次尝试不同的邻域来获得更优的解。登山算法的时间复杂度上界为 $O(C_{out} \times C_{in-max} \times mn^3)$，与工件数量的个数三次方成正比（图3-2中给出了 11 个用例的问题规模与运行时间的关系图），因此 C_{out} 的提高会极大地影响算法的搜索时间。为了考察循环次数对结果的影响，进行对比实验，表3-4中给出了用例 6 在不同 C_{out} 条件下得到的结果及运行时间。

表 3-4 外层循环次数对 HC 的影响

外层循环次数 C_{out}	1	5	10	100	500	1000	5000	10000
运行结果	1546	1505	1451	1441	1436	1431	1431	1431
运行时间/s	0.01	0.02	0.04	0.36	1.83	3.56	18.28	36.98

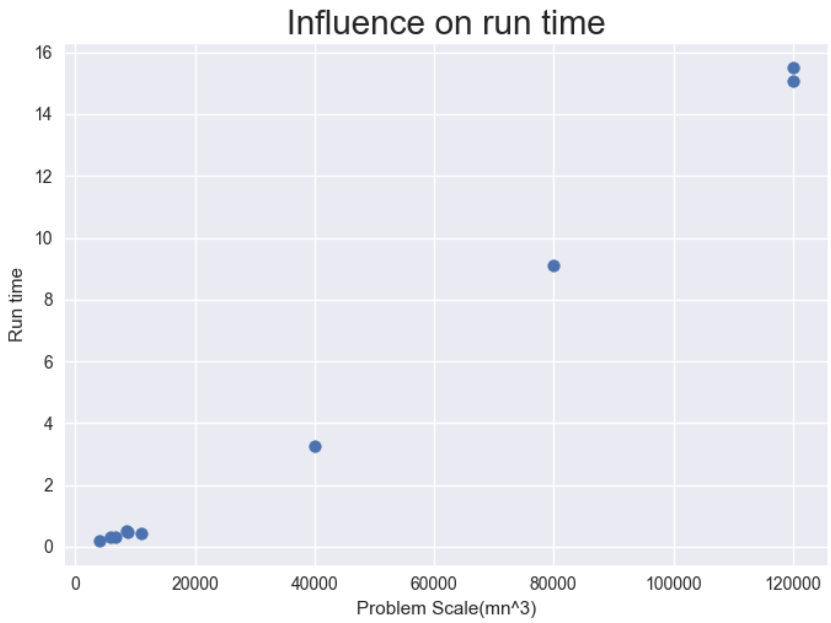

图 3-2 登山算法运行时间与数据规模的关系

根据图3-2和表3-4的结果，可以验证运行时间与 C_{out} 及问题规模的高次成正比关系，对于所用的测试用例 6，当循环次数达到 1000 以上时便可以稳定确定最优解，初始情况带来的随机性影响已经可以忽略不计。在一般情况下，可以首先根据时间需求和问题规模尝试用低循环次数预估最大循环次数上限，然后从上限开始依次降低，以确定低时间消耗下可以取得的最优计算结果。

3.2.3 模拟退火算法：初始温度的影响

初始温度一般是一个比较高的值，在退火初期，高的温度使得算法更容易跳脱出当前局域（因为跳脱概率随退火温度负相关）。因此高的初始温度可以使得搜索过程更随机化，与登山的定向优化差别更大，更有可能找到最优解。但同时过高的初始温度一方面会增大退火循环次数，速度过慢，不利于解的收敛，容易耗费大量时间进行广度上的“跳脱”而非合理的定向优化。为了定量考察初始温度的影响，对不同的初始温度值进行测试，结果如表3-5。

由表可知，在其他参数保持不变的情况下，当退火初始温度降低时，可能达不到最优调度时间；当退火初始温度达到一定值后继续升高时，算法仍能达到最佳时间，但是运行时间更长。对于用例 6，1e5 左右是一个比较合适的初始温度取值区间。

表 3-5 初始温度对 SA 结果的影响

初始温度 T_0	0.1	1	10	100	1000	1e4	1e5	2e5
用例 6：计算结果	1523	1447	1432	1433	1438	1433	1431	1431
用例 6；运算时间/s	2.71	3.25	3.83	4.23	4.78	5.19	5.73	5.95
用例 10：计算结果	3288	4007	3280	3277	3277	3277	3277	3277
用例 10；运算时间/s	5.89	6.89	7.86	8.89	9.93	11.01	12.02	12.37

总体来说，选择初始温度可以根据问题的需求具体尝试确定，温度设置过低会导致无法跳脱局域，温度过高搜索范围过大效率下降。因此在使用模拟退火算法求解时要根据问题的规模、需求、解空间特点具体选择合适的中间值作为初始退火温度。

3.2.4 模拟退火算法：衰减系数的影响

衰减系数时模拟退火过程中温度的退火率，决定了温度的衰减速度和中层循环的次数，反映了退火搜索过程的精细程度。不同衰减系数的运行结果和运行时间如表3-6所示。

表 3-6 衰减系数对 SA 结果的影响

衰减系数 α	0.94	0.95	0.96	0.97	0.99	0.995
用例 6：计算结果	1456	1450	1441	1434	1431	1431
用例 6；运算时间/s	0.94	1.13	1.40	1.88	5.91	11.82
用例 10：计算结果	3286	3294	3277	3277	3277	3277
用例 10；运算时间/s	2.05	2.47	3.09	4.16	12.51	25.22

由上表可知，在其他参数保持不变的情况下，当退火率降低时，总运行时间大幅度减少，但算法的优化效果也衰退明显；当退火率达到 0.99 继续升高时，运行时间显著增加。在这两个例子中，0.99 是一个比较合适的退火率取值。事实上，退火率影响的是退火次数：退火率越高，退火温度下降越慢，退火所需时间越长。因此，若退火率设置过低，可能因为搜索次数不够而达不到最优解；若退火率设置过高，退火温度下降到稳定值所需的时间过长，运行时间会显著增加。在模拟退火时，对于不同规模的输入需要选取合适的衰减系数，兼顾效率和准确度。

3.2.5 模拟退火算法：退火结束温度的影响

退火的结束温度下限决定了在低温度情况下搜索的次数，这时算法的搜索与登山算法很相似，由于温度比较低，新的解一般都是当前解的邻域较优解。因此，结束温度越

低，迭代结果会更容易地收敛在当前邻域里的最优位置，但同时温度过低也会导致大量搜索是无效的，造成计算浪费。为了确定退火温度的影响，仍然控制变量使用用例 6 和 10 进行实验，结果见表3-7。

表 3-7 退火结束温度对 SA 结果的影响

退火结束温度 T_d	1000	500	100	0.1	0.01	1e-4	1e-5	1e-7
用例 6：计算结果	1649	1651	1610	1434	1431	1431	1434	1431
用例 6；运算时间/s	1.08	1.23	1.57	3.01	3.41	4.40	4.88	5.89
用例 10：计算结果	3678	3515	3590	3277	3277	3277	3277	3277
用例 10；运算时间/s	1.10	1.42	2.13	5.14	6.12	8.13	9.08	11.10

表3-7的结果显示当结束温度 T_d 过大时，解的偏差会非常大，因此一般情形中不应将结束温度设置过高。同时，可以看到尽管 T_d 降到了很低的值，优化的结果仍然会有些许的偏差，比如用例 6 中当 $T_d = 1e-5$ 时反而结果增大了。本质上这是因为退火结束温度仅仅控制了内层循环的迭代次数和收敛效果，无法避免初始随机生成解与最优解的过大距离影响（因为温度低时算法无法跳脱局部最优）。因此一方面 T_d 的控制需要与温度上限 T_0 共同调整，前者负责调整后期收敛能力，后者负责控制前期跳脱能力，另一方面要在最外层设置初始化不同解进行多次退火尝试，避免初始解位置偏差导致算法的结果具有过强随机性。

3.2.6 模拟退火算法：内外层循环效果对比

内外层循环的设计时本文采用的与常规 SA 算法不同的地方，内层循环 C_{in} 用于控制同一温度下的搜索次数，外层循环 C_{out} 用于多次尝试 SA 初始化避免随机初始解对结果的影响。显然，增大总体搜索次数结果一定会更优，因此搜索结果一定随着 C_{in}、C_{out} 的增大而增大。但是如果在一定搜索更新次数的限制下，如何分配 C_{in} 和 C_{out} 则不是一件显然的事情。

为了探索内外层循环的特点，对比实验中除了控制其他参数外，还增加了整体循环次数的约束，即 $C_{in} \times C_{out} = Const$，根据上文中给出的默认参数，实验中的约束 $Const$ 为 1000。

表 3-8 内外层循环配比对 SA 结果的影响

循环比 C_{out}/C_{in}	1/1000	2/500	5/200	10/100	20/50	50/20	100/10	500/2	1000/1
用例 6 结果	1431	1431	1452	1437	1431	1434	1444	1434	1434
用例 6 时间/s	5.82	5.75	5.76	5.67	5.71	5.69	5.68	5.67	5.68
用例 10 结果	3277	3277	3277	3327	3296	3302	3280	3287	3277
用例 10 时间/s	11.02	10.97	11.00	11.02	11.02	11.02	11.10	10.99	11.03

就用例 6 和 10 的结果来看，内层循环的效果要比外层循环好一些。内层循环是同温度下的邻域随机搜索，相当于在保证退火温度情况下增大搜索次数，使得每次的更新更趋于领域最优；外层循环则是重新初始化，尝试多次并行的模拟退火以避免初始解带来的随机性。从原理上看，这两者的作用是不相交的，各自都能弥补算法本身的一些缺陷，它们为调整参数提供了更多的选择，能够避免只使用其中一种循环导致次数过多仍然难以得到最优解的问题。在实际应用中需要根据具体的运行时间需求分配内外层循环的次数，以在合理时间范围内求得更好的优化结果。

3.2.7 SA 与 HC 搜索特点对比

在这部分中，选取用例 6 作为测试数据追踪模拟退火算法和登山算法搜索时解的变化过程以更好地对比理解两种搜索算法的特点。画出曲线图如图3-3所示。(为避免随机性，登山算法画出五次不同的搜索路线；模拟退火算法每 5000 次画一个点)

根据3-3，可以看出 SA 算法中搜索前期解的跳跃性很明显，而后期解的突变逐渐减小趋于平稳，这是因为退火温度初期高后期低的特点使得指数跳脱概率也随之相应变化。除了大量的浮动之外，可以看到 SA 算法的搜索过程主体上呈现下降趋势，只不过在局部会有比较大的波动。这种现象可以视为算法对于不同邻域中的试错尝试，在确定了较好的局部位置后，进行局域的搜索收敛于最终的最优解。相比之下，HC 算法则呈现出单一的单调递减的形式，但是由于算法仅仅在初始解的邻域内进行搜索，能够优化的范围有限并且有很强的随机性，造成了搜索过程快速达到终止的现象，优化的结果因而远远差于模拟退火算法。

从图3-3和表3-2中的实验结果对比来看，模拟退火算法在解空间较大的情况下能够获得相较登山算法更好的结果，搜索效率更高并且能够通过算法本身避免初始解位置带来的随机性。但是如果问题规模过小或者能够确定问题是凸优化（只有一个极值点即全局最优），那么登山算法可以更高效地完成搜索任务。综合来说，在车间调度问题规模较大的情况下，SA 算法在时间效率和结果效果上都显著好于 HC 算法。

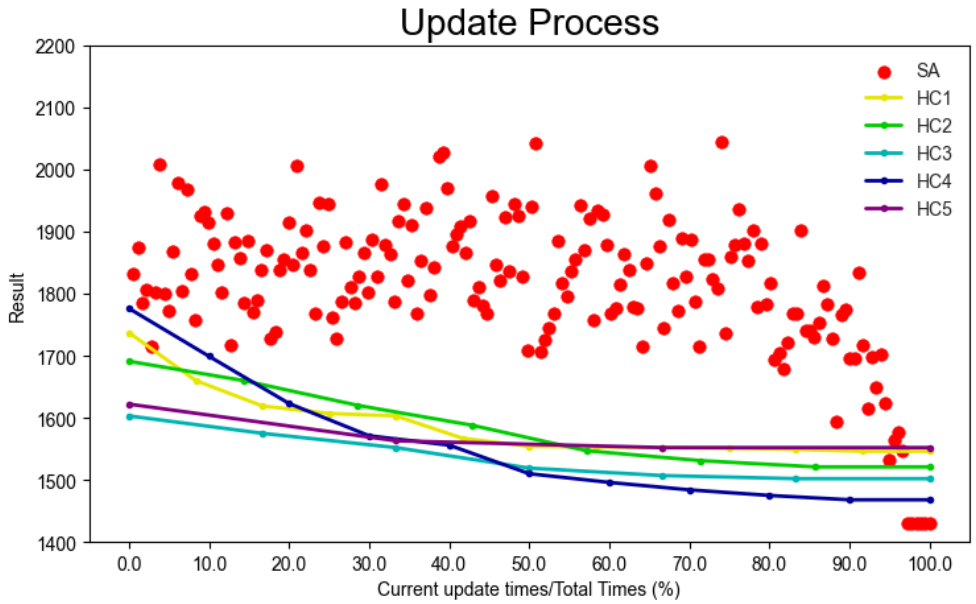

图 3-3 两种算法的搜索过程对比

3.2.8 SA 与 PSO 优劣对比

本实验主要从优化效果和时间消耗两个角度对两种算法进行比较，取相同参数条件下 (与运行结果里的最优参数不同)11 个用例的运行结果和运行时间进行绘图，得到图3-4。

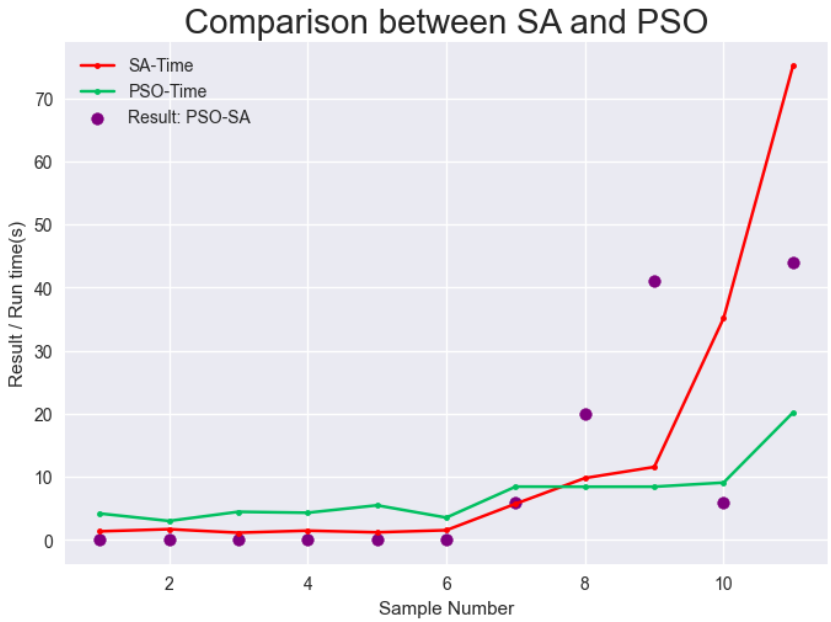

图 3-4 SA 和 PSO 的效果及效率比较

可以看到当用例的规模上涨时，模拟退火算法的时间消耗将大幅度增长，而粒子群

算法的时间增长则不明显，可以推测在更大规模的问题中，粒子群算法在时间上具有模拟退火算法无法比拟的优势。另一方面从结果上来看，result(紫色) 呈现的时 PSO 与 SA 结果之差，可以看到 PSO 在前几个用例中能够达到模拟退火算法的优化效果，而在后面几个用例中则会略逊于模拟退火算法，但是这个差值相比于总的结果来说都在可以接受的范围内。综合来看，本文采用的 PSO 算法能够进行有效的车间调度优化，在大规模问题中，PSO 算法能够以准确度换取时间，用更低的时间消耗计算结果，这在一定程度上证明了粒子群优化的理论正确性和有效性。当然，由于本问采用的离散化方法比较简单，在真正工业领域中可以设计更加复杂的 DPSO 算法，兼顾时间成本和准确度多个方面。

四. 结论

4.1 工作内容总结

本文采用登山算法、模拟退火算法和粒子群算法来解决车间调度问题。首先，从当前流水车间调度问题的研究现状出发，阐述当前研究的主要解决思路，其中包括经典算法和智能优化算法。其次，本文基于登山算法和模拟退火算法的基本原理设计了主体求解算法的流程，并根据算法需求设计了基于动态规划的结果计算函数和生成解函数。针对经典的登山和模拟退火算法进行部分改进，更改和增加不同的循环策略以获得更好的测试用例的优化结果，对粒子群算法进行离散化处理，设计新的“编码”方式使之能够用于解决车间调度问题。在给定的 11 个样例的输入下，表3-2的第一列给出了本问所得最好最佳优化值。最后，本文设计了系列对比实验进行算法分析比较。通过分析算法的时间复杂度和理论分析对参数的选择进行估计，利用参数实验分析参数对于算法的影响，采用定性分析和数据结果结合的方式对比三种优化算法的特点，分析了各者间的优劣异同。

4.2 方法讨论与分析

本文使用的方法主要分为以下两个方面，分别进行展开讨论：

(1) 算法：主要是用登山算法、模拟退火算法和粒子群算法

这几种算法都是经典的智能优化算法，但是各有优劣。登山算法在问题规模很小时发挥较好，而模拟退火对于相对较大规模的问题效果更好。但是对于更大规模的问题而言，模拟退火算法的搜索效率也很低，难以在工业或真正实际中加一应用，可以采用基于粒子群算法的 DPSO 算法进行处理，例如文献 [4] 给出了一

种使用了“改进优先操作交叉策略”(IPOX) 的多目标优化粒子群算法。就本文的研究探索，算法改进可以首先通过调整参数以适应不同数据空间的特点，其次可以根据多种不同的智能优化算法取长补短，比如将模拟退火算法和粒子群算法结合使用，根据问题需求设计新的融合算法。

另外，由于问题规模的限制，本文求解的 JSP 问题实际上是简化版本，增加了每个机器加工顺序相同的约束，使得原本大小为 $O(m \times n!)$ 解空间减小为 $O(n!)$，对算法的搜索能力、时空复杂度要求降低了很多。去掉这个约束后，全局最优的结果可能会更好，但是解空间的规模会极大提升。如何设计相应的算法解决全解空间的 JSP 问题仍是我们需要探索思考的问题。

(2) 分析方法：结果讨论分析方法

本文采用理论分析与控制变量对比实验结合方式进行分析。由于用例数据样本比较小，计算机算力有限，在比较实验中没有选择规模较大的作为测试样本，这可能造成一定的偶然性。为了解决这种实验数据的偶然因素，一方面应该增加测试的用例数量，比如一组参数使用多组数据样本进行结果计算；另一方面选择规模更大的作为实验数据，这样可以保证解的搜索过程足够复杂，对搜索过程的分析更具有普适性。

参考文献

[1] 孙丙坤. 带批生产约束的混合流水车间调度优化研究 [D]. 河南: 郑州大学,2021.

[2] 简亮. 基于改进遗传模拟退火算法的生产调度研究 [D]. 郑州航空工业管理学院,2017.

[3] 尹丹. 多目标粒子群优化算法的改进与应用 [D]. 江苏: 扬州大学,2021.

[4] 吕媛媛, 樊坤, 瞿华, 等. 多目标粒子群算法求解混合多处理机任务作业车间调度问题研究 [J]. 小型微型计算机系统,2022,43(1):218-224.

[5] 崔航浩, 张春江, 李新宇. 基于带随机网络的多种群粒子群优化算法求解多资源受限柔性作业车间调度问题 [J]. 重庆大学学报,2022,45(4):56-66.