基于启发式、元启发式算法的置换流水车间调度问题

摘　要: 置换流水车间调度问题，是在一定约束条件下安排生产调度，使得加工总时间达到最小的组合优化问题。它是典型的 NP-完全问题，为此本文采用了多种启发式、元启发式算法进行求解，包括爬山算法和基于其思想的模拟退火算法、NEH 启发式算法和利用其启发式信息的改进式蚁群算法，并对模拟退火算法在参数等方面进行了深入的探究。通过对各个算法所得结果进行比较分析，本文给出了各个算法的适用情况。实验结果表明，元启发式算法与启发式算法相比，准确率得到了显著提升。

关键词: 置换流水车间调度问题，登山算法，模拟退火算法，NEH 启发式算法，蚁群算法

一、引言

1.1 问题背景

在现代企业生产过程中，面对生产环节多、连续性强、协作关系复杂、联系紧密的生产特点，如何科学地组织生产调度成为一个值得探讨的问题。生产调度，即在各项生产条件约束下制定加工对象的加工路径与加工时间等方案 [1]，直接关系着企业的生产效益，在工业生产中发挥着举足轻重的作用。置换流水车间调度问题（Permutation Flow shop Scheduling Problem，PFSP）作为其中研究最广的 NP-完全问题的之一，是众多实际调度问题的简化模型，因而具有重要的理论意义和工程价值。

1.2 问题描述

对于置换流水车间调度问题，已知 n 个工件（$N_1, N_2, \cdots, N_n$）要在 m 台机器（$M_1, M_2, \cdots, M_m$）上进行加工，工件 N_i 在机器 M_j 上的加工时间为 $T_{i,j}$ $(i = 1, 2, \cdots, n;\ j = 1, 2, \cdots, m)$，需要确定各工件加工顺序，使得调度总加工时间 E 达到最小。

这是一类典型的组合优化问题，为了更好地对问题进行描述，对加工顺序表示如下：

- 由于每个工件的加工顺序均为从第一台机器至最后一台机器，并且每台机器对每个工件仅加工一次，默认按照机器编号的顺序加工工件，因此每个工件在机器上的加工顺序 $M = \{M_1, M_2, \cdots, M_m\}$
- 由于在每台机器上各工件的加工顺序相同，用 P_i 表示第 i 个步骤加工的工件号，因此在每个机器上工件的加工顺序 $P = \{P_1, P_2, \cdots, P_n\}$，对应工件 $N_1, N_2, \cdots, N_n$ 下标的一个全排列。

设 $S_{i,j}$ 为工件 N_i 在机器 M_j 上的开始时间，用数学语言描述本问题的优化目标及约束条件如下：

1. 优化目标——求使得总加工时间 E 最小的工件加工顺序，即使得最后一个加工的工件加工完成时间最短 $\Rightarrow \underset{P}{argmin}\ E = S_{P_n,m} + T_{P_n,m}$
2. 约束条件——每个工件均在 0 时刻释放，即第一个工件开始加工的时间大于零 $\Rightarrow S_{P_1,1} \geq 0$
3. 约束条件——每个机器只能同时加工一个工件，即下一个工件在该机器上的开始时间大于当前工件在该机器加工的结束时间 $\Rightarrow S_{P_i,j} \geq S_{P_{i-1},j} + T_{P_{i-1},j}$
4. 约束条件——一个工件不能同时在不同的机器上加工，即该工件在下一台机器的开始时间大于该工件在当前机器加工的结束时间 $\Rightarrow S_{i,j} \geq S_{i,j-1} + T_{i,j-1}$

整理上述条件后，得到完整的问题形式化描述如下：

$$
\begin{aligned}
&\text{argmin}_P\ E = S_{P_n,m} + T_{P_n,m} \\
&\text{对于} i = 1,2,\cdots,n,\ j = 1,2,\cdots,m \\
&s.t. \begin{cases} S_{P_1,1} \geq 0 \\ S_{P_i,j} \geq S_{P_{i-1},j} + T_{P_{i-1},j} \\ S_{i,j} \geq S_{i,j-1} + T_{i,j-1} \end{cases}
\end{aligned} \tag{1.1}
$$

其中，每个数学符号的含义见表1-1。

表 1-1 数学符号及对应含义

数学符号	含义
E	总加工时间
P_i	每个机器第 i 个加工的工件下标
$S_{i,j}$	工件 N_i 在机器 M_j 上的开始时间
$T_{i,j}$	工件 N_i 在机器 M_j 上的加工时间

1.3 问题分析

总体而言，解决置换流水车间调度问题的算法大致分为三类：精确方法（Exact Methods）、构造方法（Constructive Methods）和改进方法（Improvement Methods）[2]。精确方法包括分枝定界法、混合整数线性规划等方法，这些方法计算时间过长，只适合车间规模较小的问题。构造方法包括一些比较简单的启发式算法，如 NEH 算法，该方法可以在比较短的时间内得到一个近似的最优解，但是解的质量无法得到较好的保证。改进方法主要为元启发式算法，它基于直观或经验构造，通过将随机算法与局部搜索算法相结合，在可接受的花费下给出可行解，对于置换流水车间调度问题这类 NP-完全问题的求解具有更高的效率。常见的方法包括模拟退火算法、蚁群算法、粒子群算法、遗传算法等。

基于上述求解方法，本文将问题分解成了多个模块进行求解，并选取了多种启发式、元启发式算法进行优化，包括爬山算法、NEH 算法、模拟退火算法、基于 NEH 的改进式蚁群算法，对各个算法的效果进行比较，涉及的主要步骤如下：

1. 问题的形式化描述：对置换流水车间调度问题给出数学语言的描述，得出优化目标与相应的约束条件。
2. 算法的设计：采用模块化的设计思想，将算法分成了通用模块、优化模块，并基于各个算法的原理设计了各模块的实现方案。其中，通用模块包括独立于优化算法的总加工时间的计算，优化模块包括本文所选取的四种算法。
3. 程序的实践：通过 c++ 语言实现算法，运行得到结果，并对结果采用 python 语言进行甘特图的可视化。
4. 多模型对比：根据多个算法得到的运行时间、计算结果，比较各个算法的优劣，给出各个算法的适用条件。
5. 模拟退火算法探究：通过观察退火过程中解的变化进一步理解模拟退火的原理，通过调整初始温度、降温系数、退火内循环次数和模拟退火总次数等参数探究各参数对算法性能的影响。

实验结果表明，元启发式算法与启发式算法相比，准确率得到了显著提升。同时，通过模拟退火的参数实验，证明了实验中对算法在同一温度下增加内循环的改进的正确性。全过程流程图如下：

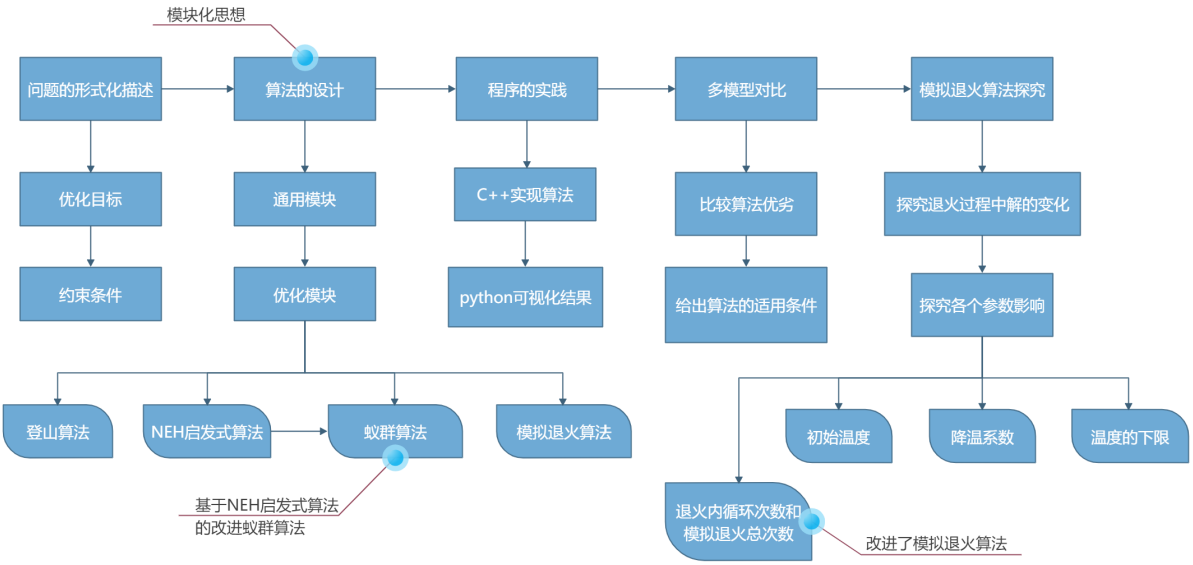


图 1-1 全过程流程图

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

对于算法的设计，在实验中采用了**模块化**的方法——将问题的求解过程分为通用模块和优化模块，二者相互独立。其中，通用模块为总加工时间的计算的方式，可以为各个优化算法所共用；由于置换车间调度问题为 NP-完全问题，优化模块由启发式算法、元启发式算法构成，包括启发式算法 NEH 算法、爬山算法和实现了随机算法与局部搜索算法相结合的元启发式算法模拟退火算法和改进的蚁群算法。

2.1 通用模块——计算总加工时间的算法设计

2.1.1 算法思路简介

为了确定不同调度顺序所需要的最少加工时间，根据该问题重叠子问题和最优子结构的性质，考虑采用**动态规划算法**。动态规划算法的基本思想是将原问题分解为相对较为简单的子问题并记录其解，自底而上进行计算，直到得到最大的子问题，即可求得原问题的解。

2.1.2 算法流程

(1) 算法关键操作

该问题中，最小的子问题为在每个机器上第一个工件加工的结束时间和每个工件在第一个机器上加工的结束时间。根据式1.1中的约束条件，可以知道为了实现加工时间最短，每个工件在每台机器上的完成时间必然满足以下特点：

- 在第一台机器上加工的第一个工件，完成时间为加工时间本身；

- 在第一台机器上加工的其余工件，完成时间为上一个加工工件的结束时间加上加工时间；
- 在其余机器上第一个加工的工件，完成时间为该工件在上一个机器的完成时间加上加工时间；
- 在其余机器上其余工件的完成时间的计算需要满足下面两个约束条件：一方面，工件需要在前一道工序完成之后才能开始加工，另一方面，工件需要在上一个工件在该机器上完成加工后才能开始加工。因此，加工完成时间为上述约束条件得到的开始时间最大值加上该工件本身的加工时间。

最后一个工件在最后一台机器上的加工时间，即加工的总时间。设 $E_{i,j}$ 为第 i 个工件在第 j 个机器上加工的完成时间，将上述自然语言符号化，可以得出工件加工的完成时间：

$$\begin{aligned} &E_{P_1,1} = T_{P_1,1} \\ &E_{P_i,1} = E_{P_{i-1},1} + T_{P_i,1},\ i = 2,3,\cdots,n, \\ &E_{P_1,j} = E_{P_1,j-1} + T_{P_1,j}, j = 2,3,\cdots,m \\ &E_{P_i,j} = max(E_{P_{i-1},j}, E_{P_i,j-1}) + T_{P_i,j}, i = 2,3,\cdots,n,\ j = 2,3,\cdots,m \end{aligned} \tag{2.1}$$

(2) 算法伪代码

Algorithm 1 Dynamic Programming

Input: 各元件在各机器上的加工时间 $time$，工件数量 n，机器数量 m，加工顺序 p

Output: $sum[p[n]][m]$，工件加工的完成时间

$sum[p[1]][1] = time[p[1]][1]$
for $i = 2$ **to** n **do**
    $sum[p[i]][1] = sum[p[i-1]][1] + time[p[i]][1]$
end for
for $j = 2$ **to** m **do**
    $sum[p[1]][j] = sum[p[1]][j-1] + time[p[1]][j]$
    for $i = 2$ **to** n **do**
        $sum[p[i]][j] = max(sum[p[i-1]][j], sum[p[i]][j-1]) + time[p[i]][j]$
    end for
end for
return $sum[p[n]][m]$

(3) 算法核心部分复杂度分析

上述算法中含有两重循环，内层循环次数为工件数量 n，外层循环次数为机器数量 m，因此时间复杂度为 $O(mn)$。由于需要用 $n \times m$ 维的 sum 数组记录中间结果，空间复杂度也为 $O(mn)$。

2.2 优化模块——爬山算法设计

2.2.1 算法思路简介

局部搜索算法是求解最优化问题时常用的启发式算法，它可以缩小解空间的大小，在短时间内得最优解。爬山算法是以局部搜索算法为框架的算法中最基础的方法，在每次迭代过程中，该算法从邻域中选取最优解并对当前解进行更新，直到满足迭代终止条件。但是该算法对初始状态依赖性较强，容易陷入局部最优。

为了尽可能缩小该算法的缺点，本文对之稍作改进，采用随机重启的方法，其基本思想如下：

1. 生成初始解：选取一个初始解，开始爬山搜索。
2. 定义邻域和候选解：根据问题的性质，定义解的邻域，并生成候选解方案。
3. 确定新解：从候选解中选取最优解，如果最优解优于当前解，则将其作为新解。
4. 迭代：重复 2-3，直到满足终止条件。
5. 随机重启：重复 1-4 指定次数，每次随机选取一个初始状态开始新一轮的爬山搜索。

2.2.2 算法流程

(1) 算法关键操作

为解决车间调度问题，下文从初始解生成、邻域和候选解确定、确定新解、迭代终止条件四个方面对算法进行设计。

1. 初始解的生成：对工件的加工顺序进行随机排序。
2. 新解的生成：遍历当前解的所有邻域（C_n^2 种交换两个工件加工顺序的方式），选取其中的最优解。需要注意的是，这里新解生成的方式不采用基础模块的算法，原因在上文已给出。
3. 新解的确定：如果交换后的总调度时间比当前调度时间短，便对当前解进行更新。
4. 迭代的终止条件：如果当前解没有更新，则已得到该处的局部最优解，迭代终止。

(2) 算法伪代码

Algorithm 2 Hill Climbing

Input: 各元件在各机器上的加工时间 $time$

Output: 最短调度时间 $BestTime$

$BestTime = \infty$
for $i = 1$ **to** s **do**
  $current$ = 随机的初始化调度顺序
  for $i = 1$ **to** t **do**
    $next$= 当前调度顺序下交换两个工件顺序之后的最优解
    if $VALUE(next) < VALUE(current)$ **then**
      $current = next$
    else
      $break$
    end if
  end for
  if $BestTime > current$ **then**
    $BestTime = current$
  end if
end for
return $BestTime$

(3) 算法核心部分复杂度分析

爬山算法的核心部分为加工时间计算函数与邻域最优解选取函数。其中，外循环是随机重启的迭代次数 s，内循环是同一次爬山中最大的迭代次数 t，选取邻域最优解需要对邻域遍历 $O(C_n^2) \sim O(n^2)$ 次，每次需要 $O(mn)$（见计算总加工时间的算法设计部分）的时间来计算每个新解的加工时间，因此，内循环内部的时间复杂度为 $O(n^2 \cdot mn) = O(mn^3)$。由于如果内循环中的解不再更新，则迭代终止，$t$ 是最坏条件下的内循环迭代次数，因此爬山算法的最坏时间复杂度为 $O(stmn^3)$。空间复杂度主要由加工时间计算函数决定，为 $O(mn)$（见计算总加工时间的算法设计部分）

2.3 优化模块——模拟退火算法设计

2.3.1 算法思路简介

由于爬山算法易陷入局部最优而难以跳出，模拟退火算法 [3] 对其做出了改进。模拟退火的思想来源于固体退火原理，是基于 Monte-Carlo 迭代求解策略的一种随机寻优算法。它模拟了固体退火原理，从一个较高的温度出发，随着温度的下降，根据概率突跳特性在解空间中搜索目标函数的全局最优解，使得温度趋于稳定时的解趋近于全局最优。

与爬山算法只向更优值点更新的策略不同，模拟退火算法允许当前值在一定概率下向更劣值更新，随着时间的推移，更新概率逐渐变小，下山移动的步长也逐渐变小。由于算法可能会在一定概率下向更劣值更新，为了提高解的质量，对传统算法稍加改进，采用有记忆的模拟退火算法 [4]，其主要思想如下：

1. 选取初始温度 T_0，降温系数 T_k，在可行解空间中随机选取初始解 $current_a$，最优解 $best_a = current_a$，内循环次数 L，温度下限 min。
2. 随机变化，在可行解空间中产生新解 new_a。
3. 计算新解与最优解的距离，$\Delta t = Value(new_a) - Value(best_a)$，其中 Value 为加工时间的计算函数。
4. 如果 $\Delta t < 0$，接受新解 new_a 作为最优解 $best_a$。
5. 计算新解与当前解的距离，$\Delta t = Value(new_a) - Value(current_a)$，其中 Value 为加工时间的计算函数。
6. 如果 $\Delta t < 0$，接受新解 new_a 作为当前解 $current_a$；否则，以指数概率 $exp(\frac{-\Delta t}{T})$ 接受（根据 Metropolis 准则）。
7. 如果新解 new_a 不被接受，则回溯至上一状态，即变化前的解。
8. 重复执行步骤 2-7 L 次。
9. 利用降温系数对温度进行更新，$T = T \cdot T_k$。
10. 重复步骤 2-9，直到温度趋近于 0。

该方法在提高解的质量的同时，也一定程度上降低了算法的效率，增大了时间与空间的开销。

2.3.2 算法流程

(1) 算法关键操作介绍

在模拟退火算法中，关键操作为步骤 2 中新解的生成、状态的回溯、结束条件与内外循环的设计。

在状态回溯的方面，用 new_a 存储每次随机生成的新解，如果接受新解，则将 new_a 赋给 $current_a$；在结束条件与内外循环设计的方面，设置外循环为温度衰减的控制循环，内循环为同一温度下固定次数的 for 循环，以增加迭代次数、提高解的质量，当温度降低到温度下限时，跳出循环；在新解生成的方式方面，采用随机交换两个工件加工顺序的方法，算法的伪代码如下：

在算法中，最关键的一点就是步骤 6 中得到新解为当前值的更劣解时，以随温度降低而递减的指数概率接受策略转移，跳出了局部最优解。

(2) 算法流程图

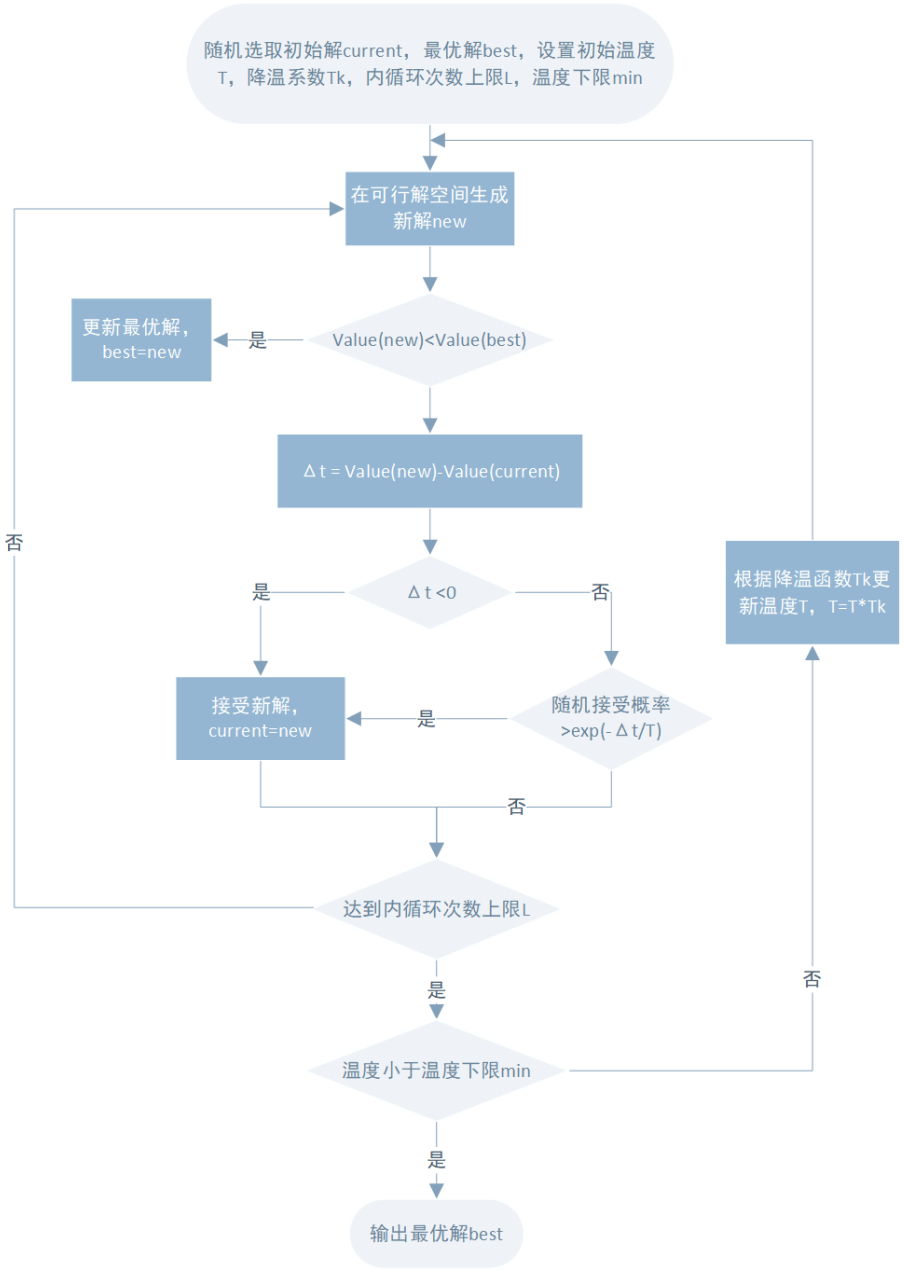


图 2-1 模拟退火算法流程图

(3) 算法核心部分复杂度分析

模拟退火算法的核心部分为加工时间计算函数与模拟退火算法函数。其中，外循环是温度衰减的控制循环，可根据初始温度和温度衰减策略得出循环次数 S 次（$S = \log_{T_k} \frac{min}{T}$）；内循环为同一温度下的循环次数 L，每次循环均需计算调度的最短时间，计算过程中用于存储加工时间的变量为程序主要占用的空间大小。

对于新解生成的复杂度，由于需要 n 维的数组存储新解，算法的空间复杂度为 $O(n)$。生成新解算法

的时间复杂度的计算由三部分组成：生成两个不同随机数、复制当前解、交换生成新解。其中，复制当前解的过程含有一重循环，时间复杂度为 $O(n)$，交换生成新解的时间复杂度为 $O(1)$，运用计算概率、期望，通过级数求和的思想计算生成两个不同随机数的时间复杂度为 $O(1)$。因此，生成新解算法的时间复杂度为 $O(n)$。

综合上述判断，结合计算总加工时间的算法设计中对调度时间计算的分析，可以得到模拟退火总时间复杂度为 $O(SLmn)$，空间复杂度为 $O(mn)$。

2.4 优化模块——NEH 启发式算法设计

2.4.1 算法思路简介

NEH 启发式算法 [5] 由 Nawaz、Enscore 和 Ham 共同提出，是解决置换流水车间调度问题最有效的启发式算法之一，可以快速地构造出最优解。该算法采用赋予总加工时间越长的工件在排列插入时越大的优先权的基本思想，每次插入一个工件，通过构造最优的局部解构造出工件整体的加工顺序。

2.4.2 算法流程

NEH 算法的基本流程如下：

1. 计算每个工件在所有机器上的总加工时间 $TP_i = \sum_{j=1}^{m} T_{i,j},\ i \in \{1, 2, \cdots, n\}$。把工件按照 TP_i 非增的顺序排列，得到初始排列 $S = S(1), S(2), \cdots, S(n)$。
2. 从 S 中取出排在前两位的工件 $S(1), S(2)$，将二者排序，得到两个部分调度 $\{S(1), S(2)\}$ 和 $\{S(2), S(1)\}$。计算两个部分调度中的完工时间，取较小的排列作为当前调度，得到新的调度排列 $S' = \{S'(1), S'(2)\}$。令 $i = 3$。
3. 从 S 中取出第 i 个工件 $S(i)$，对 S' 进行所有可能的插入，得到 i 个部分排列，选取完工时间最短的部分排列作为当前调度 S'。
4. 令 $i = i + 1$，如果 $i \leq n$，则继续执行步骤 3，否则输出调度 S' 作为最终调度，算法结束。

分析算法的基本流程，步骤 1 计算总加工时间的时间复杂度为 $O(mn)$，冒泡排序的时间复杂度为 $O(n^2)$，步骤 3 插入过程的时间复杂度为 $O(n^2 \cdot mn)$，算法整体时间复杂度为 $O(mn^3)$。算法占用的空间主要为对总加工时间、工件调度的存储，空间复杂度为 $O(n)$。

2.5 优化模块——基于 NEH 启发式算法的改进式蚁群算法

2.6 算法思路简介

受到蚂蚁寻找食物行为的启发，意大利学者 Dorigo、Maniezzo 和 Colorni 等人提出了蚁群算法 [6]。在蚂蚁寻找食物时，它会在走过的路径上释放一种被称为信息素的物质，影响其他蚂蚁的路径选择。当一条路径上走过的蚂蚁越多，留在该路径上信息素越多，后来的蚂蚁选择该条路径的概率越高。通过信息素这种媒介，蚂蚁群最终会走上一条并未意识的从蚂蚁巢通向食物的最短路。标准蚁群算法的流程如下：

1. 将最优化问题转化为在某个图上的搜索问题。
2. 在图上放置一批蚂蚁，蚂蚁基于各条路径上的信息素浓度和本地的启发信息能见度，构造出一个路径。
3. 每只蚂蚁根据各自的解更新所经过路上的信息素浓度。
4. 重复步骤 2、3，直至终止条件满足。

刘延风在《基于蚁群优化的置换流水车间调度算法》中对蚁群算法进行了改进 [7]，将 NEH 启发信息与蚁群算法相结合，充分发挥了启发信息对蚁群的引导作用，取得了优良的结果。本文采用刘延风的改进算法，对基于 NEH 启发式算法的改进式蚁群算法进行了实践。

2.7 算法流程

首先，将置换流水车间调度问题中工件调度顺序的选取转化为图的搜索问题，得到表示如下：

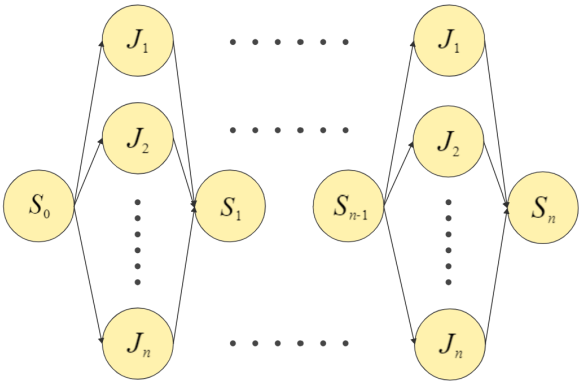


图 2-2 蚁群算法图表示

其中，S_0、S_1 到 S_n 为状态结点，分别表示初始状态、确定一个工件后、确定两个工件后和确定完所有工件调度顺序的状态，J_1 到 J_n 为任务节点，代表待选择的工件。

对算法进行初始化，根据 NEH 算法计算出每个工件在所有机器上的总加工时间 $TP_i = \sum_{j=1}^{m} T_{i,j}$，$i \in \{1, 2, \cdots, n\}$，并根据算法得到的工件排列的启发式结果得到对应的完工时间 $makespan$。设置蚂蚁个数 na，迭代次数 $count = 0$，最大迭代次数 $Ncmax$。定义 τ_{ij} 为把工件 i 放在状态 j 的信息素，设置其初始值为 $\tau_0 = 1/(n \cdot makespan)$，使得能够构成较好解的工件在对于的状态能够以更大的概率被选中。定义能见度 $\eta_i = TP_i$，使得蚁群算法能够利用 NEH 算法的优良启发式信息构造出较好的结果 [8]。

对于置换流水车间调度问题，所有蚂蚁均被放在 S_0 状态处。每个任务节点被设置了信息素信息，蚂蚁由一个状态转移到另一个状态，选取任务节点时，依照转移概率公式计算可能到达该任务节点的概率。第 k 只蚂蚁在第 j 个状态选择工件 i 的概率公式为

$$p_{ij}^{k} = \begin{cases} \frac{[\tau_{ij}]^{\alpha}[\eta_i]^{\beta}}{\sum_{u \in \text{allowed}_k}[\tau_{uj}]^{\alpha}[\eta_u]^{\beta}}, & \text{工件} i \text{ 从未被第} k \text{ 只蚂蚁选中} \\ 0, & \text{其他} \end{cases}$$

其中，allowed_k 表示没有被蚂蚁 k 选到过的工件。

对于蚁群构造的部分解，根据 NEH 算法的步骤 2 到步骤 4，进行局部优化；对于每只蚂蚁构造的完整解，将第 i 个位置（i 的范围为 0 到 $n-2$）的工件依次插入 $i+1$ 至 $n-1$ 位置，进行插入式局部寻优。

对于每只蚂蚁，根据自身路径所得解的相关信息依据如下公式进行信息素的更新：

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{k=1}^{na} \Delta\tau_{ij}^{k}$$

其中，ρ 为挥发系数，范围为 0 到 1，$1-\rho$ 表示在每次更新信息素时剩余的信息素占原有信息素的比

例，以加快收敛。$\Delta\tau_{ij}^{k}$ 选取了第 k 只蚂蚁所构造的解得到的完工时间的倒数，完工时间越少，则对应路径上信息素增量越多。

该算法全过程的流程图如下：

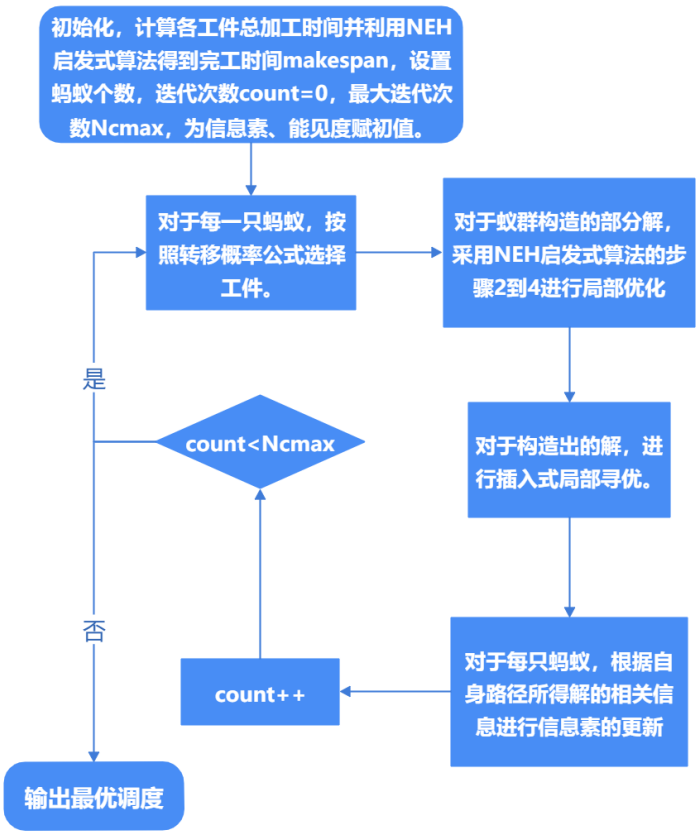


图 2-3 蚁群算法流程图

该算法时间复杂度由初始化、工件选择、插入式局部寻优、信息素更新四部分构成。初始化部分，结合 NEH 启发式算法的时间复杂度分析，makespan 计算的时间复杂度为 $O(mn^3)$，信息素初始化需要遍历 n 个状态的 n 个任务节点，时间复杂度也为 $O(n^2)$；工件选择部分，需要重复 $Ncmax$ 次，每次具有 na 只蚂蚁分别进行选择，选择的时间复杂度为 $O(n^2)$，依照 NEH 启发式算法进行局部优化的复杂度为 $O(mn^3)$，因此工件选择部分的时间复杂度为 $O(Ncmax \cdot na \cdot m \cdot n^5)$；插入式局部寻优算法需要两层循环，时间复杂度为 $O(n^2)$；信息素更新时间复杂度同信息素初始化，为 $O(n^2)$。因此，基于 NEH 的改进式模拟退火算法由于结合了 NEH 算法，整体时间复杂度较大，为 $O(Ncmax \cdot na \cdot m \cdot n^5)$。该算法的空间利用主要为信息素、能见度的占用，空间复杂度为 $O(n^2)$。

三、实验

3.1 实验设置

(1) 实验环境

1. 系统：Windows10 操作系统
2. 语言：C++、python 语言环境
3. 平台：

- 算法部分：Visual Studio Code、Microsoft Visual Studio 2022 等 C++ 开发环境。
- 绘图部分：Pycharm 2021 等 python 开发环境。

(2) 输入格式

1. 算法部分：与给定 flowshop-test-10-student.txt 的用例格式相同。
2. 绘图部分：每绘制一幅甘特图，需运行一次。输入为算法部分输出文件 result.txt 中的一个用例与 flowshop-test-10-student.txt 中的对应用例，均不包含每个用例开始的序号名、工件数等信息以及用例之间的分隔符。

(3) 参数设置与运行时间

1. 爬山算法：

   - 外层循环（随机重启的次数）：50000
   - 内层循环（同一次登山搜索的上限）：1000
   - 运行时间（所有用例）：50 秒以内

2. 模拟退火算法：为了能够兼顾解的质量和运行时间，对各个用例分别进行调参，得到较好的运行参数如下：

表 3-1 模拟退火算法参数及运行时间

序号	内循环次数	初始温度	降温系数	温度下限	运行时间（s）
0	100	1000	0.99	1×10^{-7}	1.675
1	1000	5000	0.95	1×10^{-7}	0.77
2	1000	10000	0.97	1×10^{-7}	1.101
3	2000	20000	0.99	1×10^{-7}	1.655
4	1000	10000	0.99	1×10^{-7}	1.514
5	1000	10000	0.99	1×10^{-7}	1.615
6	1000	150000	0.99	1×10^{-7}	6.089
7	3000	150000	0.998	1×10^{-6}	133.256
8	1000	10000	0.99	1×10^{-7}	2.698
9	3000	150000	0.997	1×10^{-7}	93.341
10	1000	10000	0.99	1×10^{-7}	13.349

3. 基于 NEH 启发式算法的改进式蚁群算法

   - $\alpha = 1, \beta = 2$：分别控制已有信息素信息和启发式信息对蚂蚁选择的影响
   - $\rho = 0.3$：挥发系数，用于信息素的更新
   - $na = 1.5 \cdot$ 工件数：蚂蚁个数，在一次搜索中并行进行的蚂蚁。
   - $Ncmax$：迭代次数，实验中各个用例依次为 200、500、5000、5000、3000、5000、5000、5000、5000、5000、5000 个
   - 运行时间：工件数较少时不超过 1 秒，工件数较大时可达几分钟

分别采用优化模块的各个算法，调试参数以得到最优解，得到运行结果见表3-8。

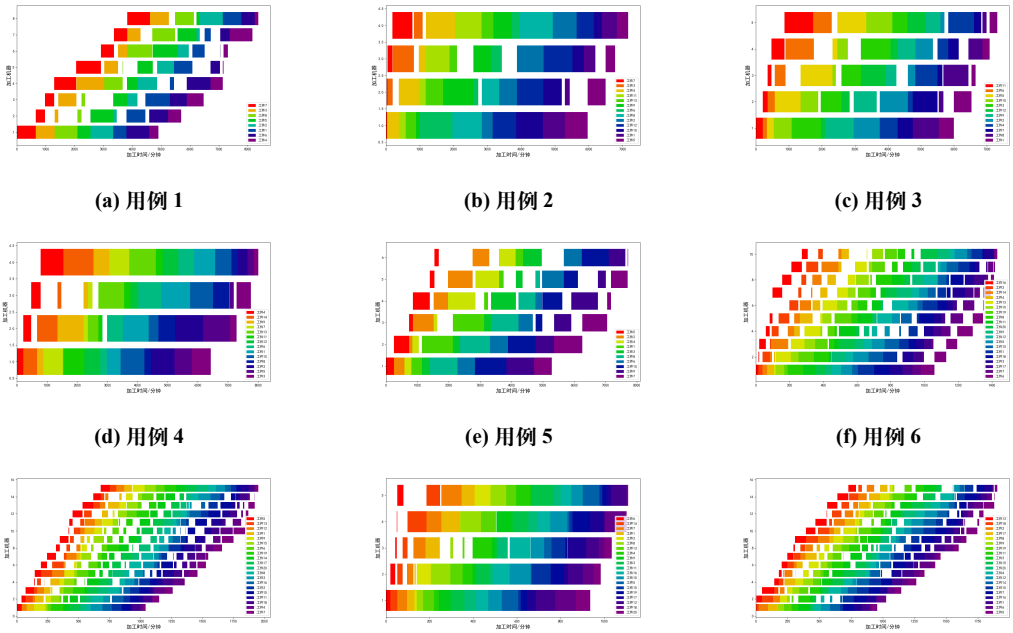
表 3-2 各个算法结果、运行时间

序号	模拟退火算法结果	运行时间（s）	爬山算法结果	运行时间（s）	NEH 算法结果	运行时间（s）	蚁群算法结果	运行时间（s）
0	**7038**	1.675	7038	0.297	7038	<0.001	7038	0.083
1	**8366**	0.77	8366	0.897	8564	<0.001	8366	0.323
2	**7166**	1.101	7280	0.165	7376	<0.001	7166	2.496
3	**7312**	1.655	8160	0.318	7399	0.001	7312	3.285
4	**8003**	1.514	8233	0.189	8003	<0.001	8003	0.384
5	**7720**	1.615	8394	0.437	7835	<0.001	7720	2.819
6	**1431**	6.089	1629	4.503	1550	0.001	1431	38.554
7	**1950**	133.256	2075	17.739	2013	0.001	2047	104.832
8	**1109**	2.698	1135	0.56	1132	<0.001	1109	5.506
9	**1902**	93.341	2012	17.33	2019	0.001	1902	101.622
10	**3277**	13.349	3467	12.402	3313	0.01	3463	225.477

从表3-8中可以发现，模拟退火算法、蚁群算法等元启发式算法结果显著优于 NEH、爬山算法等启发式算法，由于元启发式算法将随机算法与局部搜索算法相结合，跳出了局部最优解，能够得到较好的结果。另一方面，元启发式算法运行时间较启发式算法更长，时间上最大可相差 200 秒，差距较为显著。相较于爬山算法，NEH 启发式算法在保证速度的同时，结果整体上优于爬山算法；模拟退火算法在部分实验的结果上优于蚁群算法，可能是没有对蚁群算法进行精细调参的缘故。本文采用的基于 NEH 启发式信息改进的蚁群算法在工件数较多的情况时明显变缓：对于 50 个工件、10 台机器的用例 10，蚁群算法的速度远低于其余算法，原因在于蚁群算法复杂度较高，会随着工件数的增加显著增加。

因此，对于对解的要求不高，并且需要快速计算时，可采用 NEH 启发式算法；对于工件规模不大，特别是机器数较多工件数较少的情况，可以采用蚁群算法；对于工件规模较大的情况，采用模拟退火算法或其他元启发式算法。

为了更好地展示各个用例的最优调度顺序，采用甘特图对结果可视化，如下图所示。



(a) 用例 1 **(b) 用例 2** **(c) 用例 3**

(d) 用例 4 **(e) 用例 5** **(f) 用例 6**

(g) 用例 7 **(h) 用例 8** **(i) 用例 9**

图 3-1 各用例最优调度对应的甘特图

3.2 模拟退火算法深入探究

下文中，对模拟退火算法进行了深入探究。首先，为了具体观察退火过程中解的变化，记录过程中当前解的调度时间，绘制折线图如图3-2所示。

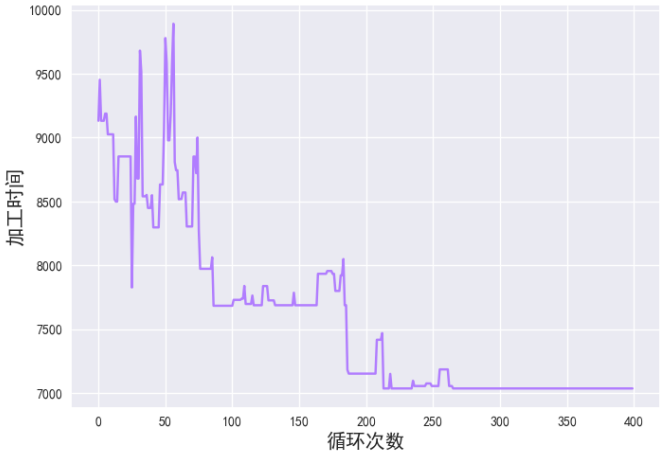


图 3-2 模拟退火中解的变化

从图中可以看出，随着循环次数的增加与模拟退火的进行，加工时间震荡下降，直至趋于稳定：不断震荡是因为按照一定概率向更劣的策略转移，逐渐下降是因为随着温度的下降，向更劣策略转移的概率不断降低。该过程与模拟退火的原理相符合。

为了进一步分析模拟退火算法中各个参数对实验结果、运行时间的影响，采用控制变量的方法调整用例 1、用例 2 中的参数，通过重复实验来对比分析，判断当前参数的设置是否合理 [9]。各个用例的参数默认值如下：

表 3-3 各用例的默认参数

序号	内循环次数	初始温度	降温系数	温度下限
1	1000	5000	0.95	1×10^{-7}
2	1000	10000	0.97	1×10^{-7}

3.2.1 改变初始温度，其余参数不变

从默认参数可以看出，初始温度通常选取较高的值。从定性的角度而言，较高的初始温度对解主要有两方面的影响：一方面，较高的初始温度使得当前解劣于最优解时接受当前解的概率更大，增大了搜索空间（即增大了 exploration），相较寻找局部最优解的登山算法更有可能得到全局最优解；另一方面，初始温度过高，容易在整个搜索空间进行大范围的搜索而非小范围的优化（即增大 exploration 的另一方面是阻碍了 exploitation），耗费时间过长并且不利于解的收敛。因此，为了定量地探究初始温度对结果的影响，选取不同的初始温度对各个用例进行测试：

表 3-4 不同初始温度的实验结果

初始温度	用例 1 的结果	用例 1 的运行时间（s）	用例 2 的结果	用例 2 的运行时间（s）
1×10^{-1}	9048	0.078	8253	0.56
1×10^{0}	8865	0.789	8026	0.894
1×10^{1}	8457	0.895	7795	1.155
1×10^{2}	8410	2.147	7638	2.375
5×10^{2}	8409	2.392	7293	2.399
1×10^{3}	8409	2.234	7226	1.501
5×10^{3}	8366	2.336	7226	2.471
1×10^{4}	8366	2.766	7166	3.133
1×10^{5}	8366	3.268	7166	3.779
1×10^{6}	8366	2.967	7166	3.053

分析实验结果可以发现，在其他参数保持不变的情况下，初始温度较低时，总加工时间相对较长，难以达到最优解；当初始温度达到一定值后，基本稳定在最优解，并且运行时间随温度的增大而增大。对于温度过大对结果影响不大的原因，考虑是实验中在同一温度下多次迭代，循环次数较多，使得结果相对来说具有较高的稳定性。

因此，初始温度设置过低，算法更容易陷入局部最优难以跳出，初始温度设置过高，影响主要在于延长计算时间，降低算法效率。实际选取初始温度时，需要根据问题的特点选取合适的中间值。对于实验中的用例 1，选取 5000 作为初始温度；对于用例 2，选取 10000 作为初始温度。

3.2.2 改变降温系数，其余参数不变

降温系数控制了每次退火时温度衰减的幅度，决定了外层循环的次数和退火的精细程度。选取不同的降温系数，对各个用例进行测试：

表 3-5 不同降温系数的实验结果

降温系数	用例 1 的结果	用例 1 的运行时间（s）	用例 2 的结果	用例 2 的运行时间（s）
0.9	8410	0.212	7290	0.264
0.91	8409	0.269	7290	0.429
0.93	8366	0.324	7281	0.371
0.95	8366	0.506	7166	0.572
0.97	8366	0.708	7166	0.765
0.99	8366	2.694	7166	2.429

分析实验结果，在其他参数不变时，当降温系数增加，所得结果越来越接近于最优解；当降温系数增加到一定程度，结果保持最优，并且运行时间越来越长。实际上，降温系数较低时，温度衰减较快，外层循环次数较少，搜索次数较少，难以得到最优解；降温系数较高时，退火较为精细，外层循环次数较大，容易得到最优解。然后，温度系数过高（如表中 0.999），运行时间会显著增大。因此，选取降温系数时需要综合考量，在保证结果精确度的时候尽可能降低运行时间，保证算法的效率。对于实验中的用例 1、用

例 2，分别选取 0.95、0.97 作为降温系数，在增加随机算法的精确度的同时保证运行效率。

3.2.3 改变温度的下限，其余参数不变

温度的下限决定了模拟退火的总次数，尤其是温度较低时搜索的次数——温度较低时，接受劣于当前解的解的概率大幅减少，算法主要在一局部解空间内深挖。因此，温度下限越低，越容易收敛到最优解，但温度过低时，类似降温系数过高的情况，会出现解已经收敛至最优解却仍然不停搜索的情况，造成计算资源的浪费。通过改变温度的下限，对其影响进行定量分析：

表 3-6 不同温度的下限的实验结果

温度的下限	用例 1 的结果	用例 1 的运行时间（s）	用例 2 的结果	用例 2 的运行时间（s）
1×10^{-3}	8409	0.012	7204	0.043
1×10^{-2}	8424	0.026	7373	0.066
1×10^{0}	8366	0.064	7287	0.126
1×10^{-3}	8409	0.145	7376	0.258
1×10^{-5}	8366	0.149	7166	0.329
1×10^{-7}	8366	0.151	7166	0.35
1×10^{-9}	8366	0.186	7166	0.427

根据表格中的结果，在其他参数保持不变的时候，温度过高时，所得结果距离最优解较远，随着温度的降低，总加工时间震荡下降，最终得到最优解。一方面，温度下限对结果的影响具有一定的随机性，跟随机生成的初始解关系更大；另一方面，随着温度下限越低，运行时间整体上越来越长。在确定温度下限时需要结合初始温度综合考虑，由初始温度控制前期的探索，由温度下限控制后期的深挖。对于本实验中的用例 1、用例 2，选取 1×10^{-7} 作为温度的下限。

3.2.4 改变退火内循环次数和模拟退火总次数，其余参数不变

在模拟退火的改进方案中，通常通过重复进行多次模拟退火实验，取最优解作为算法的解，以提高解的稳定性。然而在实验中发现，增大同一温度下迭代的次数较增加模拟退火的次数具有更好的效果，考虑是增加同一温度下迭代次数使得进行一次模拟退火的总搜索次数显著增加，增加了算法深挖的程度，提高了整体的稳定性。因此，在本实验中采用的算法中，舍去了重复进行模拟退火的循环，增加了同一温度下重复进行的循环。为了定量探究二者对结果的影响，本小节分别改变模拟退火次数和内循环次数，对二者的影响进行对比分析。

首先，改变内循环次数，重复进行 3 次模拟退火实验，取最优结果如下：

表 3-7 不同内循环次数的实验结果

内循环次数	用例 1 的结果	用例 1 的运行时间（s）	用例 2 的结果	用例 2 的运行时间（s）
1	8570	0.001	8042	0.001
1×10^1	8617	0.003	7571	0.004
5×10^1	8420	0.012	7585	0.02
1×10^2	8473	0.003	7376	0.044
5×10^2	8366	0.141	7166	0.239
1×10^3	8366	0.273	7166	0.441
5×10^3	8366	1.45	7166	2.204
1×10^4	8366	2.815	7166	4.116

可以发现，随着内循环次数的增加，结果由开始的随机波动逐渐变得稳定。同时，当内循环次数大于 1000 次后，在重复进行 3 次实验时每次得到的结果均为最优解，具有较好的稳定性，因此，用例 1、2 均选取 1000 次作为内循环的次数。随着内循环次数的增大，运行的结果和结果的稳定性明显变优。

接着，设置内循环次数为 1，重复进行多次模拟退火实验，探究模拟退火的次数对结果的影响：

表 3-8 不同模拟退火次数的实验结果

模拟退火次数	用例 1 的结果	用例 1 的运行时间（s）	用例 2 的结果	用例 2 的运行时间（s）
1	8393	0.001	8014	0.001
1×10^1	8675	0.001	7794	0.001
1×10^2	8825	0.001	8061	0.001
1×10^3	8711	0.001	8027	0.001
1×10^4	8868	0.002	8278	0.004
1×10^5	8665	0.019	7844	0.037
1×10^7	8703	2.171	7940	3.861
1×10^8	8834	21.299	7730	26.33

从实验结果中可以发现，重复多次进行模拟退火对结果并没有过大的影响，需要结合初始温度、降温系数、最低温度等因素综合调参。相比之下，增大同一温度下的循环次数对能够给结果带来显著的影响：当内循环次数大到一定程度，能够保证绝大多数情况下进行模拟退火时得到的都是最优解。本文所采用的在同一温度下重复进行搜索而非增加模拟退火次数的方法，创新了模拟退火算法，并且通过了实验的证实。

四、总结

本文采用了多种启发式、元启发式算法来解决置换流水车间调度问题，包括爬山算法、模拟退火算法、NEH 启发式算法、蚁群算法。其中，对模拟退火算法稍加改进，采用了有记忆的模拟退火算法，并在实验的过程中引入了在同一温度下增加内循环的思想，对算法进一步改进；对于蚁群算法，采用了在 NEH 启发算法基础上的改进蚁群算法，在初始化时充分利用 NEH 算法的启发式信息、在搜索新解时也采用 NEH 算法进行局部优化、插入式局部寻优，提高了解的质量。

本文首先从问题形式化的角度出发，对置换流水车间问题进行数学语言描述，明确了问题的优化目标、约束条件。其次，根据该问题已有的研究现状，从精确方法、构造方法和改进方法三类解决方案中选取了后两种中的部分算法进行尝试。根据各个算法的思想，将解决方案模块化，分成了通用模块、优化模块，并基于原理设计了各模块的实现方案。通过对参数的调整，得到了给定用例的最优结果，并对结果进行了甘特图的可视化。接着，通过对各个算法结果、运行时间的比较，得出了各个算法的适用条件，并对模拟退火算法在初始温度、降温系数、退火内循环次数和模拟退火总次数等方面进行了探究，并通过实验结果进一步证明了对算法在同一温度下增加内循环的改进的正确性。

然而，本文的算法仍有不足之处，特别是当算法规模过大时无法在短时间内取得最优解，可以考虑如下解决方法：修改参数、修改随机解的生成规则，以得到更可靠的结果；融合多种算法的优势以取长补短，如将模拟退火算法和遗传算法相结合。本文基于 NEH 算法的改进式蚁群算法就是融合算法的一个很好的例子，然而限于个人精力并未对该算法进行性能方面深入的评估，考虑对各个参数进行精细的调参，将结果与模拟退火算法进行进一步对比。

参考文献

[1] 谢丽芳, 费跃农. 一种基于模拟退火算法的作业车间调度算法 [J]. 中国制造业信息化,2006(09):50-53.

[2] 刘莹, 谷文祥, 李向涛. 置换流水线车间调度问题的研究 [J]. 计算机科学,2013,40(11):1-7+22.

[3] Kirkpatricks S, Vecchi M P. Optimization by simulated annealing[M]. San Francisco, Cal. , USA: Morgan Kaufmann Publishers Inc., 1987.

[4] 卢宇婷, 林禹攸, 彭乔姿, 王颖. 模拟退火算法改进综述及参数探究 [J]. 大学数学,2015,31(06):96-103.

[5] Nawaz M, Enscore E, Ham I. A heuristic algorithm for the machine n-job flow shop sequencing problem[J]. Omega, 1983, 11:11-95.

[6] Colorni A，Doigo M . Heuristics from nature for hard combinatorial optimization problems[J]. International Trans Operational Research, 1996(3):1-21.

[7] 刘延风, 刘三阳. 基于蚁群优化的置换流水车间调度算法 [J]. 系统工程与电子技术,2008(09):1690-1692.

[8] 杨文进. 置换流水车间调度问题上的蚁群算法研究 [D]. 湖南大学,2010.