基于模拟退火算法的流水车间调度问题

Ashen です

摘　要: 流水车间调度问题（Flow Shop Scheduling Problem）是指通过合理安排工件在机器上的加工顺序使得总体加工时间最短，以提高车间作业的效率。本文基于模拟退火算法建立了相应的车间调度模型，并对具体用例进行实验模拟，通过调整参数获得最优的调度时间。同时，本文对实践结果进行详细分析：通过绘制甘特图使得加工流程可视化，验证了算法结果的正确性；本文还通过多参数的对比试验，分析了各个参数对算法效率的影响并找到了适宜的参数设置。最后，总结了整体工作，并分析了不足之处，提出了一系列解决方案。

关键词: 流水车间调度问题；模拟退火算法；Metropolis 准则；Monte Carlo 模拟

1　引言

1.1 问题背景

流水车间调度问题（Flow Shop Scheduling Problem）是指在一个由 m 台机器组成的车间里，有 n 个工件需要在这些机器上进行加工，每个工件需要按照规定的加工顺序依次在各个机器上进行加工，每个机器同一时间只能加工一个工件。要求设计一种作业调度方案，使得所有作业加工完成的时间最短。

流水车间调度问题是一个经典的优化问题，属于 NP-hard 问题，即不存在多项式时间算法能够求解其最优解。因此，目前主要采用的是一些启发式算法或近似算法来求解该问题，如遗传算法、模拟退火算法、蚁群算法等。

在实际生产中，流水车间调度问题是一个非常重要的问题，因为它直接关系到生产效率和生产成本。一个合理的调度方案可以极大地提高生产效率，减少生产成本。

1.2 问题分析

1.2.1 约束

- 有 n 个工件需要在 m 台机器上流水加工
- 每个工件的加工顺序相同，从第一台机器至最后一台机器
- 每个工件均在 0 时刻释放
- 每个工件在每台机器上只加工一次
- 一个工件不能同时在不同的机器上加工
- 每个机器同时只能加工一个工件

1.2.2 数学描述

流水车间问题可描述为：n 个待加工的工件按一定的顺序在 m 台机器上进行加工，设第 i 个工件在第 j 台机器上加工的时间为

$$PT(i, j)，1 \le i \le n，1 \le j \le m$$

这个处理时间是固定且已知的，设第 i 个工件在第 j 台机器上完成加工的总时间为

$$CT(i, j)，1 \le i \le n，1 \le j \le m$$

这个完成时间是未知的，需要通过计算获得，设 Completion_Time 为整个流水线车间的完工时间，以下是具体的数学公式描述

$$\begin{cases} CT(1,1) = PT(1,1) \\ CT(1,j) = CT(1,j-1) + PT(1,j) \\ CT(i,1) = CT(i-1,1) + PT(i,1) \\ CT(i,j) = \max(CT(i-1,j), CT(i,j-1)) + PT(i,j) \end{cases}$$

CT(1, 1) = PT(1, 1) :工件 1 在机器 1 上的完工时间等于自身的处理时间

CT(1, j) = CT(1, j-1) + PT(1, j) :工件 1 在机器 j 上的完工时间等于工件 1 在机器 j-1 上的完工时间加本次的处理时间

CT(i, 1) = CT(i-1, 1) + PT(i, 1) :工件 i 在机器 1 上的完工时间等于工件 i-1 在机器 1 上的完工时间加本次的处理时间

CT(i, j) = max(CT(i-1, j), CT(i, j-1)) + PT(i, j) :工件 i 在机器 j 上的完工时间等于工件 i-1 在机器 j 上的完工时间和工件 i 在机器 j-1 上的完工时间之间的较大值加上本次的处理时间

通过上述的公式可总结出如下的目标函数

Completion_Time = min(CT(n, m))

即整个流水车间的完工时间等于最后一个工件在最后一个机器上的完工时间。

1.3 解决方案

本文基于模拟退火算法对流水车间问题进行求解，模拟退火算法的基本思想是模拟物质从高温到低温的退火过程，通过随机性的跳出局部最优解，以较大的概率接受劣解，从而避免局部最优解陷阱。主要步骤如下：

a. 初始化参数：初始温度、终止温度、降温速率和迭代次数
b. 随机生成一个初始解
c. 进行迭代：每次迭代会根据当前的解产生邻域解，若新解比当前解更优则接受新解，否则将根据 Metropolis 准则以一定概率接受新解
d. 返回最终的解

实验结果表明模拟退火算法的性能会受到参数的影响，如果降温过快，搜索过程会停留在局部最优解，如果降温过慢，搜索过程会浪费大量时间在低温状态下搜索，其优点是可在全局范围内搜索最优解，避免陷入局部最优解陷阱，同时，算法的随机性使得搜索过程不容易陷入死循环。

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 算法思路简介

模拟退火算法是一种通用的随机优化算法，用于在一个大的搜索空间中寻找全局最优解或次优解。其基本思路是模拟固体物质从高温状态逐渐冷却至低温状态的物理过程，通过在搜索空间中随机游走来寻找最优解。

具体来说，模拟退火算法从一个初始解开始，以一定的概率接受一个新解，如果新解比当前解更优，则直接接受；如果新解比当前解更差，则以一定的概率接受该新解，这样就有可能跳出局部最优解，继续寻找更优的解。随着算法的迭代，温度逐渐降低，接受劣解的概率也逐渐降低，最终达到稳定状态，得到全局最优解或次优解。

2.2 算法关键操作介绍

2.2.1 初始化参数

需要初始化的参数包括：初始温度、终止温度、降温速率和迭代次数，各个参数的设置会极大地影响算

法的效率，下面做具体分析：

① 初始温度：初始温度越高，接受劣解的概率就越大，搜索空间就越广，但是算法的收敛速度会变慢。因此，初始温度的设置要根据问题的复杂程度和搜索空间的大小来调整，在此问题中设置初始温度 initial_temperature 为 10000

② 终止温度：终止温度越低，算法的搜索深度越深，找到全局最优解的概率更大，但会导致算法降温花费时间过长；终止温度过高可能导致算法过早停止，无法找到全局最优解，终止温度的设置同样需要根据问题的复杂度和搜索空间的大小来调整，在此问题中设置终止温度 stopping_temperature 为 1e-8

③ 降温速率：降温速率越慢，算法的搜索深度越深，可能会找到更优的解，但是计算时间也会增加，在此问题中设置降温速率 cooling_rate 为 0.95

④ 迭代次数：迭代次数越多，算法的搜索深度越深，可能会找到更优的解，但是迭代次数过多会导致算法计算时间过长，甚至陷入局部最优解，在此问题中设置迭代次数 iterations 为 1000

2.2.2 生成初始解

对于 n 个工件的加工顺序，首先需要生成随机加工顺序并生成对应初始解，将此初始解作为当前最优解，以便于之后模拟退火的迭代过程中根据当前解产生邻域解。

2.2.3 迭代与取舍

在模拟退火的迭代过程中，我们需要不断地尝试不同地加工顺序直至最优，本文中使用的迭代方法是每次迭代随机交换两个工件的加工顺序，将新加工顺序所需加工时间与当前最优的加工时间进行比对，根据 Metropolis 准则[1] 以概率来确定是否接受新解，这一迭代过程称为 Monte Carlo 模拟[2] 。

假设前一状态为$x(n)$，系统迭代到下一个状态，变为$x(n+1)$，相应地，加工时间由$E(n)$变为$E(n+1)$。定义系统由$x(n)$变为$x(n+1)$的接收概率为p(probability of acceptance)：

$$p=\begin{cases}1, & E(n+1)\le E(n)\\ \exp\left(-\dfrac{E(n+1)-E(n)}{T}\right), & E(n+1)\ge E(n)\end{cases}$$

当状态转移之后，若新加工顺序所需时间少于当前加工顺序所需时间，则接受新解，将所迭代出的新加工顺序视为最优加工顺序；若新加工顺序所需时间多于当前加工顺序所需时间，就说明系统正偏离全局最优的位置，此时算法不会立刻将新解抛弃，而是进行概率判断是否接受状态转移。每次状态转移后，无论是否接受新解都需要执行降温过程。

在 Monte Carlo 模拟中，我们通常进行多次随机状态转移，每次转移都依据 Metropolis 准则来确定是否接受状态转移。通过这种方式，我们可以在状态空间中搜索具有最大概率分布函数值的状态，从而得到系统的最优解或最优近似解。

2.2.4 返回输出

当达到最大迭代次数或者温度达到终止温度后，将最优的加工顺序以及所需的加工时间返回输出。

2.3 算法具体步骤

模拟退火算法的流程图如下：

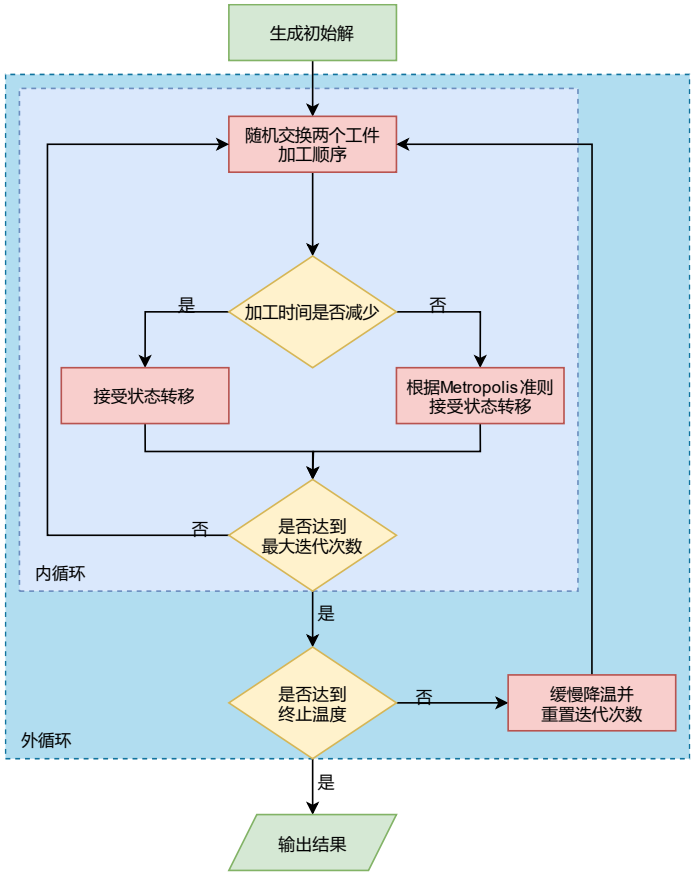

图 2-1 模拟退火算法流程图

其中计算特定加工顺序的加工时间运用到了动态规划的思想，算法伪代码如下：

Input: processing_time[i][j]: Processing time of workpiece j under machine i

schedule[i]: The number of i-th workpiece to be processed

n: The number of workpieces

m: The number of machines

Output: completion_time[n][m]: Total scheduling time

1: for each i ∈ [1, n] do

2: for each j ∈ [1, m] do

3: completion_time[i][j]← processing_time[schedule[i-1]][j-1] + max(completion_time[i-1][j], completion_time[i][j-1])

4: end for

5: end for

6: return completion_time[n][m]

2.4 时空复杂度分析

1）时间复杂度分析

数据读取和处理的时间复杂度为 O(11 * n * m)，其中 n 和 m 分别为工件数和机器数。因为代码只读取了 11 个用例，因此可以简化为 O(n * m)。

calculateCompletionTime 方法中包括两个嵌套的 for 循环，因此时间复杂度为 O(n * m)。

shuffle 方法中包括一个 for 循环和一个随机数生成器，时间复杂度为 O(n)。

模拟退火算法包括一个 while 循环、一个 for 循环和两个随机数生成器，因此时间复杂度为 O(n ^ 2)。

因此，整个代码的时间复杂度为 O(n * m + n ^ 2)。

2）空间复杂度分析

n 和 m 分别占用 O(11) 的空间。

processing_time 列表包含了 11 个 n*m 的二维数组，因此占用 O(11 * n * m) 的空间。

calculateCompletionTime 方法中定义了一个 (n+1) * (m+1) 的 completion_time 二维数组，因此占用 O(n * m) 的空间。

shuffle 方法中只使用了常数级别的额外空间。

模拟退火算法中定义了 current_schedule 数组和 new_schedule 数组，每个数组占用 O(n) 的空间，因此总共占用 O(n) 的空间。

因此，整个代码的空间复杂度为 O(11 * n * m + n)。

3 实验

3.1 实验设置

3.1.1 实验环境

操作系统：Windows 10 家庭中文版 64 位

处理器：AMD Ryzen 9 5900HX with Radeon Graphics

内存：16GB

显卡：NVIDIA GeForce RTX 3060 Laptop GPU

编译器：IntelliJ IDEA Community Edition 2022.1.4

JDK 版本：Oracle OpenJDK version 18.0.2

3.1.2 参数设置和运行时间

表 3-1 模拟退火算法参数设置和运行时间

用例	初始温度	终止温度	循环次数	降温速率	运行时间(ms)
0	1000	1e-8	1000	0.95	314
1	1000	1e-8	1000	0.95	180
2	1000	1e-8	1000	0.95	213
3	1000	1e-8	1000	0.95	251
4	1000	1e-8	1000	0.95	274
5	1000	1e-8	1000	0.95	212
6	1000	1e-8	1000	0.95	425
7	1000	1e-8	1000	0.95	470
8	1000	1e-8	1000	0.95	322
9	1000	1e-8	1000	0.95	469
10	1000	1e-8	1000	0.95	740

3.2 实验结果

3.2.1 客观结果

为保证实验结果的准确性，所有用例均测试多次，但由于部分用例最优加工顺序不唯一，因此随机挑选其中一种加工顺序制表，最终得到各用例的最优加工顺序如下表 3-2 所示

表 3-2 各测试用例最优加工时间和最工顺序

用例	工件数	机器数	加工时间	加工顺序
0	11	5	7038	[7, 2, 4, 3, 10, 1, 6, 9, 5, 0, 8]
1	5	8	6269	[3, 1, 0, 4, 2]
2	10	4	5977	[5, 2, 3, 9, 0, 1, 8, 6, 7, 4]
3	12	5	7321	[8, 9, 7, 11, 4, 3, 6, 2, 10, 1, 5, 0]
4	15	4	9231	[2, 12, 10, 11, 13, 0, 6, 8, 5, 14, 4, 7, 3, 1, 9]
5	9	6	7498	[3, 1, 0, 2, 6, 4, 8, 7, 5]
6	19	10	1376	[1, 13, 8, 11, 15, 3, 12, 9, 18, 10, 7, 2, 4, 14, 0, 17, 16, 6, 5]
7	19	15	1910	[4, 13, 7, 3, 18, 12, 17, 11, 15, 6, 1, 14, 0, 10, 8, 9, 16, 2, 5]
8	18	5	1006	[11, 4, 5, 1, 0, 8, 10, 14, 13, 7, 9, 16, 2, 6, 12, 3, 15, 17]
9	19	15	1912	[7, 8, 18, 16, 12, 17, 1, 14, 3, 2, 11, 13, 10, 9, 6, 15, 0, 5, 4]
10	40	10	2766	[7, 8, 34, 33, 36, 20, 38, 19, 17, 11, 3, 2, 12, 4, 29, 21, 9, 35, 13, 6, 28, 32, 39, 0, 1, 30, 31, 37, 14, 5, 10, 27, 23, 15, 24, 16, 26, 22, 18, 25]

各用例的甘特图表示如下，其中同一种颜色表示同一个工件

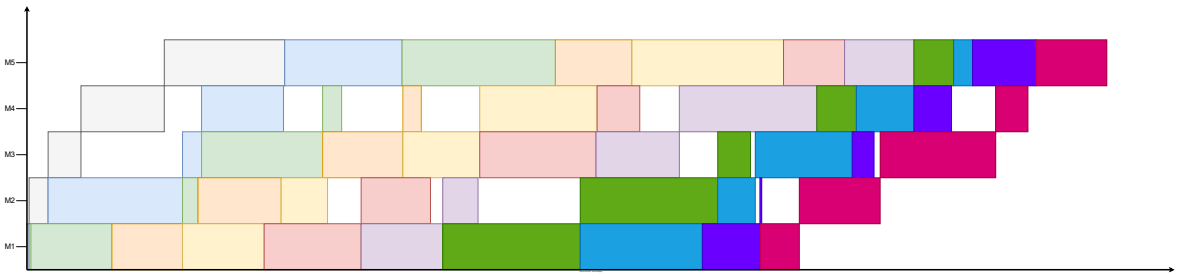

图 3-1 用例 0 甘特图

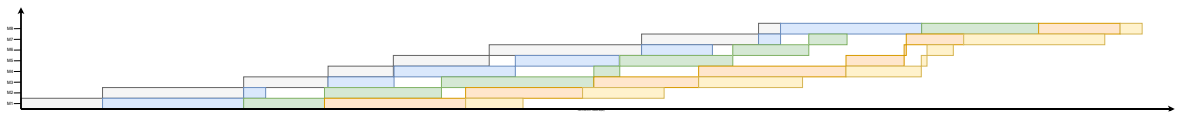

图 3-2 用例 1 甘特图

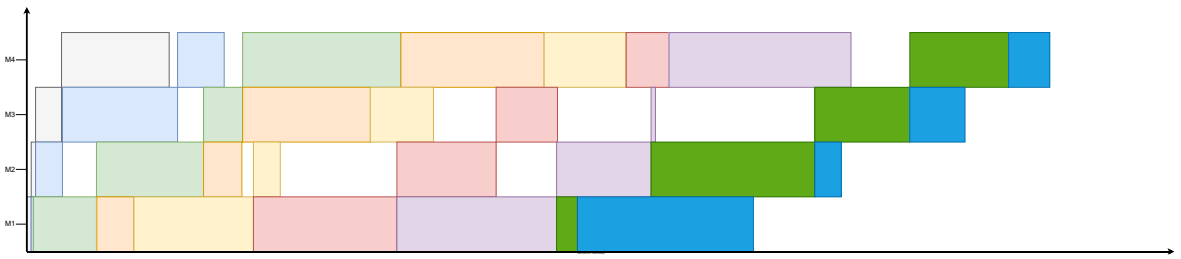

图 3-3 用例 2 甘特图

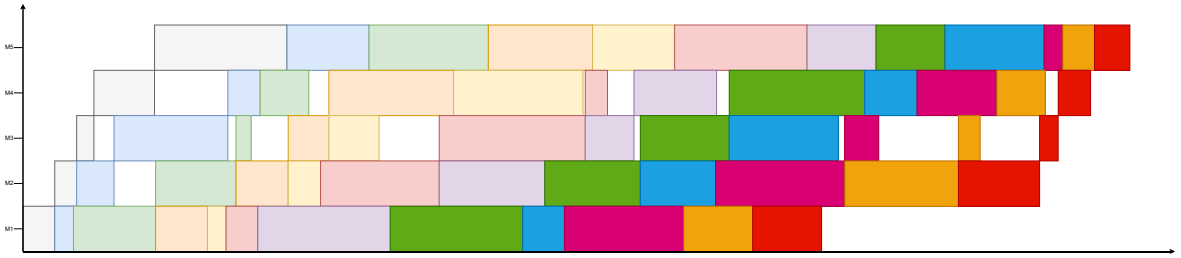

图 3-4 用例 3 甘特图

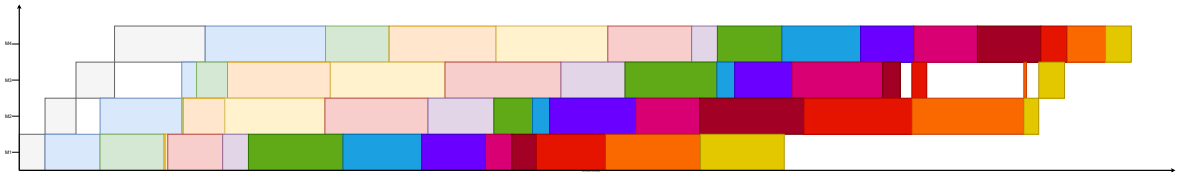

图 3-5 用例 4 甘特图

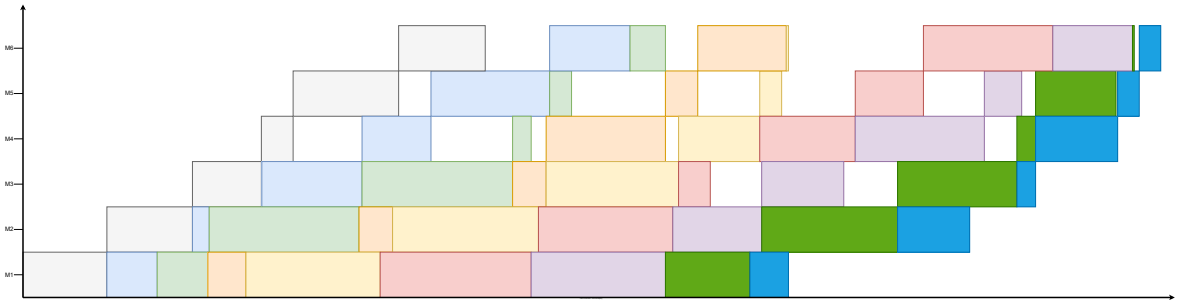

图 3-6 用例 5 甘特图

3.2.2 对比试验

1) 改变初始温度，其他参数不变

其他参数设置为：终止温度 1e-8，降温速率 0.95，循环次数 1000

表 3-3 仅改变初始温度的加工时间和运行时间

初始温度	用例 0 加工时间	用例 0 运行时间 (ms)	用例 4 加工时间	用例 4 运行时间 (ms)	用例 8 加工时间	用例 8 运行时间 (ms)
1000	7038	326	9231	279	1010	328
5000	7038	342	9231	287	1005	340
10000	7038	355	9231	291	1006	341
50000	7038	358	9231	325	1024	378
100000	7038	360	9231	329	1006	381
500000	7038	368	9231	329	1024	387
1000000	7038	369	9231	344	1011	398
5000000	7038	407	9231	355	1012	416

由表可知，在其他参数不变的前提下，初始温度过低或过高时都可能找不到最优解，其中初始温度如果设置过高会导致运行时间增加并且无法找到最优解。

分析其原因，当初始温度设置过高时，搜索过程中的震荡幅度会变大，算法可能会陷入局部最优解或者停滞不前，并且使搜索范围过大，搜索时间过长；当温度设置过低时，算法的收敛速度会变慢，可能会在局部最优解处停滞不前而无法找到全局最优解；总结来说，初始温度主要影响了算法的搜索范围和跳出局部最优解的能力，因此在设置初始温度时需要尝试多次对比实验以找到最适合的初始温度。

2) 改变终止温度，其他参数不变

其他参数设置为：初始温度 1000，降温速率 0.95，循环次数 1000

表 3-4 仅改变终止温度的加工时间和运行时间

终止温度	用例 0 加工时间	用例 0 运行时间 (ms)	用例 4 加工时间	用例 4 运行时间 (ms)	用例 8 加工时间	用例 8 运行时间 (ms)
1e-5	7038	249	9231	196	1024	228
1e-6	7038	279	9231	220	1006	264
1e-7	7038	326	9231	243	1010	289
1e-8	7038	358	9231	277	1024	327
1e-9	7038	331	9231	292	1006	356
1e-10	7038	356	9231	326	1006	387

由表可知，在其他参数不变的前提下，终止温度设置过高或过低都可能找不到最优解，其中，如果终止温度设置过高，算法可能会在搜索空间中随机游走而无法达到最优解，并且算法需要更长的运行时间才能达到终止状态；如果终止温度设置过低，算法可能会在搜索空间中过早地停止，无法充分探索搜索空间，从而导致算法停留在局部最优解而无法达到全局最优解。

综上，终止温度在模拟退火算法中是控制算法搜索空间深度的重要参数，为了获得算法的最佳性能，以充分探索搜索空间并找到最优解，需要通过实验来确定最佳的终止温度。

3) 改变循环次数，其他参数不变

其他参数设置为：初始温度 1000，终止温度 1e-10，降温速率 0.95

表 3-5 仅改变循环次数的加工时间和运行时间

循环次数	用例 0 加工时间	用例 0 运行时间 (ms)	用例 4 加工时间	用例 4 运行时间 (ms)	用例 8 加工时间	用例 8 运行时间 (ms)
100	7038	61	9231	51	1006	39
500	7038	217	9231	162	1009	206
1000	7038	370	9231	331	1005	395
5000	7038	1452	9231	1579	1006	1864

由表可知，在其他参数不变的前提下，循环次数设置过高或过低都可能找不到最优解，其中，如果循环次数设置过高，算法可能需要更长的时间才能达到中止状态，这会大大降低算法的效率；如果循环次数过低，算法可能会在搜索空间中过早地停止，最终导致算法停留在局部最优解而无法达到全局最优解。

综上，循环次数是控制算法搜索空间广度的另一个重要参数，为了获得算法的最佳性能，需要对用例进行多次实验以找到最适当的参数设置。

4) 改变降温速率，其他参数不变

其他参数设置为：初始温度 1000，终止温度 1e-8，循环次数 1000

降温速率	用例 0 加工时间	用例 0 运行时间 (ms)	用例 4 加工时间	用例 4 运行时间 (ms)	用例 8 加工时间	用例 8 运行时间 (ms)
0.7	7038	68	9231	48	1012	48
0.8	7038	95	9231	68	1010	72
0.9	7038	179	9231	136	1010	159
0.95	7038	312	9231	268	1006	322

0.99	7038	1265	9231	1362	1010	1693

由表可知，在其他参数不变的前提下，降温速率设置过高或过低都可能找不到最优解，其中，如果降温速率设置过高，模拟退火的过程需要花费更多时间才能达到终止状态，并且算法的稳定性会降低，这意味着在不同的运行中，算法可能会产生不同的结果；如果降温速率设置过低，算法在解空间中的搜索过程会进行得不彻底，这会导致算法可能找不到全局最优解。

综上，降温速率是控制算法搜索空间深度的一个重要参数，需要注意的是，降温速率的最佳值与具体问题的性质有关，因此在不同的问题上可能需要进行不同的参数设置。

4 总结

4.1 总结工作

本文通过模拟退火算法来解决流水车间调度问题，并通过控制变量原则调整参数，最终分析出了各个参数对于算法性能的影响。本文首先从流水车间调度问题出发，阐述了问题的研究背景，并对问题进行分析与数学抽象；接着，设计了基于模拟退火算法的解决流水车间调度问题的具体算法，其中包括了基于动态规划思想的加工时间算法；最后通过多组对比试验，观测出不同的参数对算法性能的影响，并分析阐述普适性规律。

4.2 未来展望

不足之处：

算法的时间复杂度高，在处理大规模问题时需要花费大量时间找到最优解，并且在运行过程中经常陷入局部最优解。

解决方案：

1) 参数调整：可以采用实验设计或启发式方法来确定最佳参数，以提高算法的效率和准确性。

2) 增加终止条件：可以增加提前终止算法的条件，当算法产出的解在较多次的循环中都没有更新，说明此时可能已经收敛到最优解，可以提前终止算法。

3) 初始解：可以使用其他算法来生成来生成更好的初始解。

参考文献:

[1] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller. Equations of state calculations by fast computing machines. [J]. *J. Chem. Phys.*, 1953, 21(6): 1087–1092.

[2] Metropolis, N. and Ulam, S. The Monte Carlo method. [J]. *J. American Statist. Assoc.*, 1949, 44: 335–341.

[3] Banterle, M., Grazian, C., Lee, A., and Robert, C. P. Accelerating Metropolis-Hastings algorithms by Delayed Acceptance. [J]. *ArXiv e-prints.*, 2015, 1503.00996.

附录 A：考虑无等待约束的改进算法

无等待流水车间调度问题（No-Wait Flow Shop Scheduling Problem）是指在一个由 m 台机器组成的车间里，有 n 个工件需要在这些机器上进行加工，每个工件需要按照规定的加工顺序依次在各个机器上进行加工，每个机器同一时间只能加工一个工件，每个工件在一个机器上完成加工后，必须直接进入下一个机器进行加工，不需要等待其他工件的加工完成，要求设计一种作业调度方案，使得所有作业加工完成的时间最短。

A.1 算法改进

加上无等待约束后，问题的解空间并没有发生改变，但加工时间的计算方法有所改变，因此整个算法需要改动的只有重新设计加工时间的计算算法，我们以用例 1 为例子进行分析

$$\begin{pmatrix} 5 & 7 & 3 & 4 \\ 2 & 8 & 9 & 10 \\ 4 & 1 & 5 & 6 \\ 6 & 3 & 2 & 4 \\ 10 & 15 & 11 & 20 \\ 11 & 16 & 12 & 5 \end{pmatrix}$$

我们先绘制出其加工过程的甘特图以便对算法进行分析改进

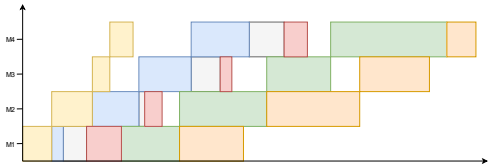

图 A-0-1 不满足无等待约束的用例 1 甘特图

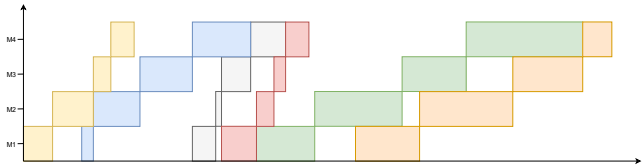

图 A-0-2 满足无等待约束的用例 1 甘特图

由甘特图可知，加上无等待约束后，每个工件在每台机器上的加工过程是紧密相连的，类似于两块多边形拼接在一起，前后两个工件完整的加工流程一定会产生一个连接处，只要能找出这个连接处就能通过动态规划的思想得到这个工件在每台机器上的完工时间，所以我们的目标就变成了找到前后两个工件加工流程的连接处。

仔细观察甘特图后可以发现连接处都有两个特征（设分析的工件为 i）：

1) 工件 i 在机器 j 上的处理时间大于工件 i-1 在机器 j+1 上的处理时间，即

processing_time[i][j] > processing_time[i-1][j+1]

2) 工件 i 在机器 j 上的处理时间到工件 i 在机器 m-1 上的处理时间的和大于工件 i-1 在机器 j+1 上的处理时间到工件 i-1 在机器 m 上的处理时间的和，即

$$\sum_{j}^{m-1} processing\_time[i][j] > \sum_{j+1}^{m} processing\_time[i-1][j]$$

算法为代码如下：

Input: processing_time[i][j]: Processing time of workpiece j under machine i

schedule[i]: The number of i-th workpiece to be processed

n: The number of workpieces

m: The number of machines

Output: completion_time[n][m]: Total scheduling time

1: for each i ∈ [1, n] do

2: completion_time[1][i]← completion_time[1][i-1] + processing_time[schedule[0]][i-1]

3: end for

4: for each i ∈ [2, n] do

5: for each j ∈ [1, m-1] do

6: for each k ∈ [j, m-1] do

7: sum_up← sum_up + processing_time[schedule[i-2]][k]

8: sum_down← sum_down + processing_time[schedule[i-1]][k-1]

9: end for

10: if sum_down > sum_up and processing_time[schedule[i-1]][j-1] > processing_time[schedule[i-1]][j] then

11: colum← j

```
        break
    colum← m
end for
completion_time[i][colum]← completion_time[i-1][colum] + processing_time[schedule[i-1]][colum-1]
for each j ∈ [colum+1, m] do
    completion_time[i][j]← completion_time[i][j-1] + processing_time[schedule[i-1]][j-1]
endfor
for each j ∈ [1, colum-1] do
    completion_time[i][j]← completion_time[i][j+1] - processing_time[schedule[i-1]][j-1]
end for
end for
return completion_time[n][m]
```

A.2 实验结果

为保证实验结果的准确性，所有用例均测试多次，最终得到各用例的最优加工顺序如下表 3-2 所示

表 A-1 各测试用例最优加工时间和加工顺序

用例	工件数	机器数	加工时间	加工顺序
1	6	4	74	[1, 4, 5, 2, 3, 0]
2	5	4	129	[4, 0, 3, 2, 1]
3	7	7	6795	[0, 3, 1, 5, 2, 4, 6]
4	8	8	7485	[5, 7, 1, 6, 3, 0, 2, 4]

各用例的甘特图表示如下，其中同一种颜色表示同一个工件

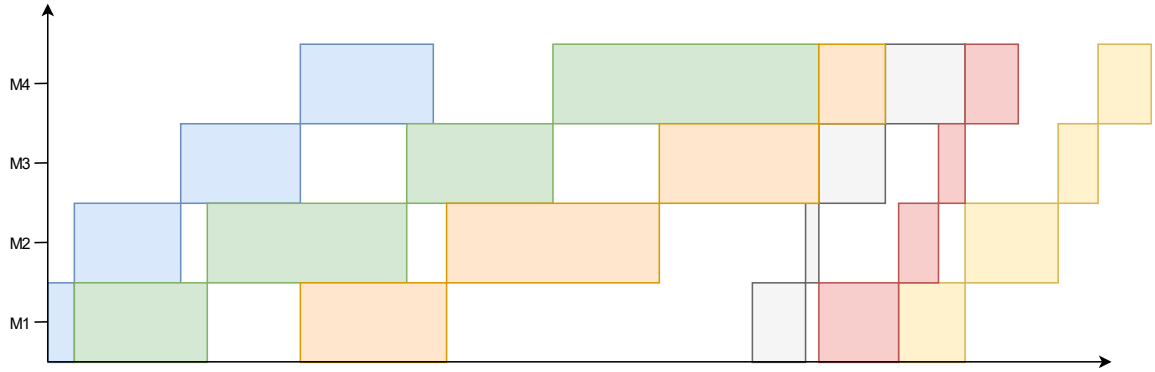

图 A-3 用例 1 甘特图

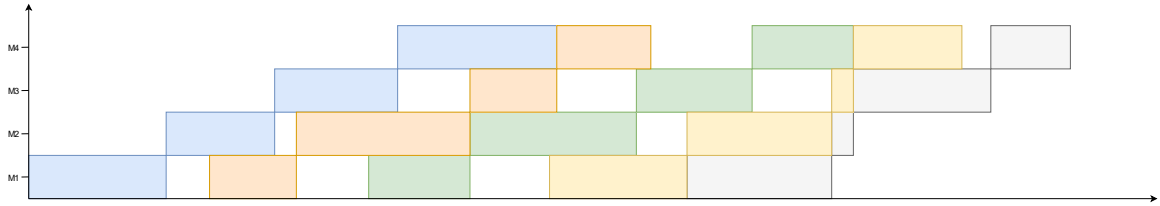

图 A-4 用例 2 甘特图

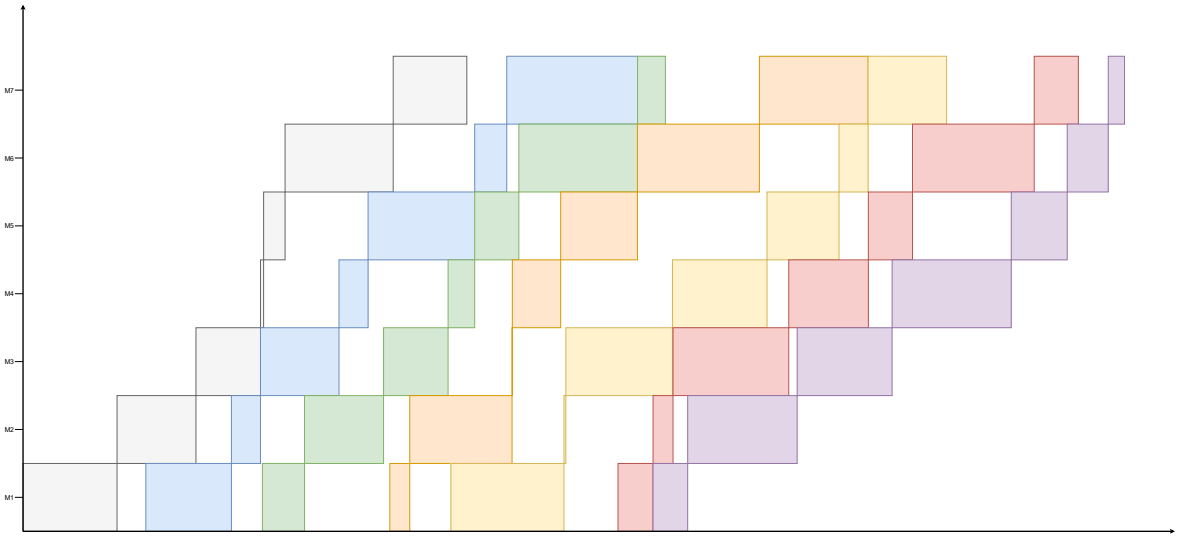

图 A-5 用例 3 甘特图