**用模拟退火算法和遗传算法解决**

**置换流水车间调度(PFSP)问题**

**vJanGo**

**摘 要:**

置换流水车间问题（Permutation Flow shop Scheduling Problem ，PFSP），是对流水车间调度问题（Flow Shop Scheduling Problem，FSP）的扩展，是一类经典的组合优化NP难问题，也是很多生产调度的简化模型，对于实际生产和理论研究有着重要意义。本文采用多种启发式算法包括模拟退火算法，带NEH启发式算法的遗传算法来解决该问题，对给出的11个用例进行了求解，最终得到了它们的最优解和调度方案。实验表明NEH启发式算法能一定程度提高算法效果，同时表明模拟退火算法在准确性和用时上都更加优异。

**关键词:** 置换流水车间调度问题，模拟退火算法，NEH启发式算法，遗传算法

# **1引言**

**1.1 问题背景**

随着科技的发展，追求更高的生产效率成为现代企业不可避免的问题。在冶金，化工，机械生产等多种工业环境中，如何选择合理的生产调度方式是企业发展的一大问题，有资料显示，制造过程中 95%的时间消耗在非切削过程中[1]，可见企业的生产成本和效益很大程度上依赖于制造过程中的调度技术。置换流水车间问题（Permutation Flow shop Scheduling Problem ，PFSP），是对流水车间调度问题（Flow Shop Scheduling Problem，FSP）的扩展，是一类经典的组合优化NP难问题，也是很多生产调度的简化模型，对于实际生产和理论研究有着重要意义。

**1.2 问题描述与数学建模**

置换流水车间调度问题中，所有工件在各个机器上的调度方式一致，即***n***个工件在***m***台机器上加工，每个工件要进行***m***次加工，一台机器同时只能加工一个工件，一个工件同时也只能被一台机器加工，工件***i***在机器***j***上的加工时间给定，据此求解最大的最小完工时间。

我们设工件序号为***n(N1,N2,N3...Nn)***,机器序号为***m(M1,M****2****,M3...Mm)***，***Tij***表示工件***i***在机器***j***上的加工时间。其中，***Sij***表示工件***i***在机器***j***上的开始时间，给定工件的一个排列为***π=(σ1,σ2,...,σn)***，有如下数学形式：

式***(1.2.1)***说明了目标函数，即求一序列，使得加工时间最小

式***(1.2.2)***说明了各个约束条件，从上到下分别为：

1. 第一个工件的开始加工时间大于等于0
2. 第***i***个工件在机器***j***上的开始时间大于等于机器上一工件的结束时间，即一台机器同时只能加工一个工件
3. 第***j***个机器加工第***i***个工件的时间大于等于上一机器加工该工件的结束时间，即一个工件同时只能被一台机器加工

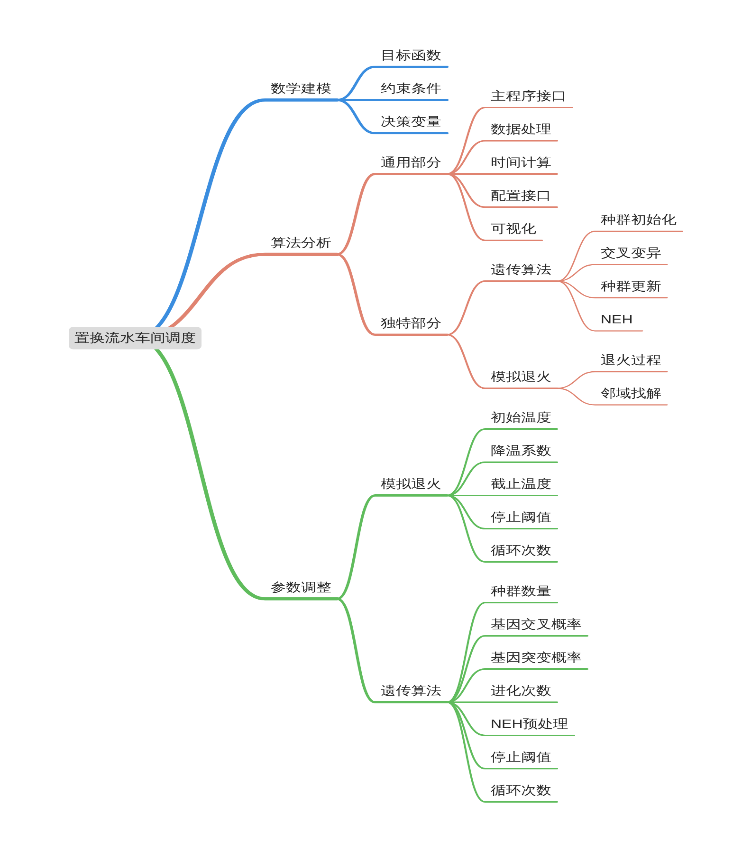
各个符号的数学意义如下表：

**表 1.2.1 数学符号及其含义**

|  |  |  |
| --- | --- | --- |
| 数学符号 |  | 含义 |
|  |  | 工件***i***在机器***j***上的开始时间 |
|  |  | 工件***i***在机器***j***上的加工时间 |
|  |  | 给定的加工顺序中第***i***个工件 |

**1.3 解决方案与实验效果**

本文运用模拟退火算法与遗传算法求解给出的调度例子，使用动态规划算法计算具体调度方案所用的时间，并且将配置文件独立写成接口，便于后续实验的展开。同时，在使用遗传算法求解的过程中，对传统的种群初始化，交叉编译以及更新种群的操作做了创新性的修改，引入了启发式算法NEH，在工件数量增多时，明显提高了最优解的质量和收敛速度。

**** 整体实验效果非常良好，两种方法对于求解结果各有长处，实验对方法中各种参数均进行了细致的调试，最终得到了在不同参数下的实验结果和运行时间，有助于进一步优化调度方案。整体的解决方案如下：

# **2.算法设计**

在设计算法时，本文采用了模块化的思想，便于修改与调试程序。其中，两个算法在数据处理，工件定义，机器定义，动态规划求解时间部分完全一致，称为**通用算法部分**，而在算法的具体实现上各不相同，例如模拟退火的退火过程，遗传算法的种群更新等等，这些统称为**独特算法部分**

## **通用算法**

鉴于数据处理，工件定义等方法较为基础，不涉及算法操作，在此不展开说明，着重分析动态规划与洗牌算法。

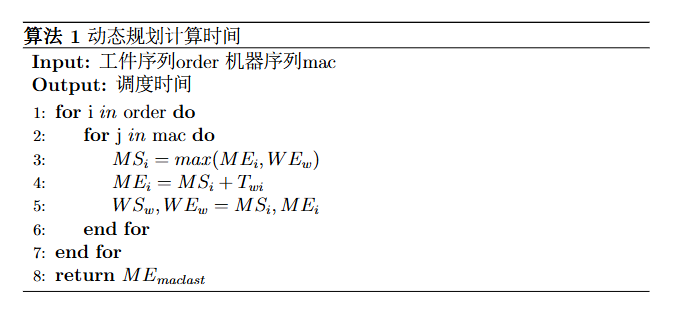
**2.1.1动态规划求解时间**

由**1.2**中的讨论可知，某个机器的开始时间受到该机器本身和要加工的工件共同约束，具体来说，机器要已加工完上一工件，同时要加工的工件已不处于加工状态。由此，我们设机器***i***开始加工时间为，结束时间为，工件***w***开始加工时间为，结束时间为，***Tij***表示工件***i***在机器***j***上的加工时间，有以下结论：

1. 工件1（按调度顺序）的开始时间为0，结束时间为
2. 机器***i***新的开始时间取决于上次机器的结束时间和上次工件的结束时间的最大值
3. 工件新的开始时间为在加工该工件的机器的开始时间
4. 机器***i***的结束时间为
5. 工件的结束时间为机器***i***的结束时间

以上述5点进行递归，我们得到如下公式：

**2.1.2 动态规划算法伪代码**

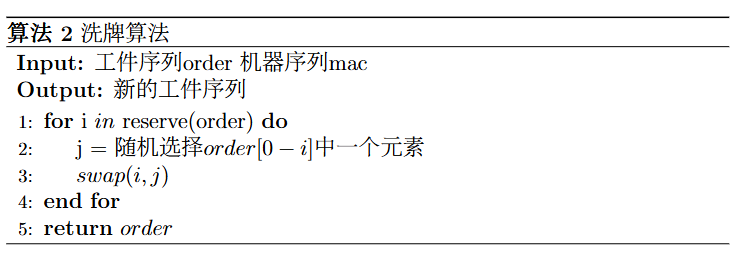


**2.1.3 洗牌算法**

在模拟退火寻找领域解和遗传算法生成初始种群的过程中，我们需要随机地生成给定长度的序列，同时要求序列中没有重复的元素。解决这个问题的方法就是洗牌算法，它通过不断交换一序列中元素***i***与其后面随机一元素，产生一个随机序列，此序列中所有元素被取得的概率可证明是1/n。算法基本过程如下：

1. 逆序遍历数组，假设已经遍历到元素***i***
2. 在元素***i***之前随机寻找到一元素***j***,交换两元素
3. 遍历到第一个元素，结束。

**2.1.4 洗牌算法伪代码**



* 1. **模拟退火算法**

**2.2.1算法简介**

模拟退火算法（simulated annealing，SA）来源于固体退火原理，是一种基于概率的算法。算法思想为：先从一个较高的初始温度出发，逐渐降低温度，直到温度降低到满足热平衡条件为止。在每个温度下，进行n轮搜索，每轮搜索时对旧解添加随机扰动生成新解，并按一定规则接受新解。

算法流程如下：

1. **初始化**：设定初始温度 和初始解 ，并计算 的目标函数值。

2. **循环开始**：对当前温度*T*，执行以下步骤：

- **从领域获得新解**：在当前解 ***X***的领域中搜索，得到新的解 ***X'***。

- **计算目标函数差**：计算新解 ***X'*** 和当前解***X***的目标函数值之差。

- **接受判断**：如果 ，则接受新***X'***；否则，按照概率接受新解

- **降温**：更新温度 *T* 为，其中是温度衰减系数。

- **平衡判断**：在当前温度下重复上述步骤直到系统达到热平衡。

3. **循环结束**：如果温度低于预设的停止阈值或其他停止条件满足，算法终止。

这个过程通过逐渐降低温度和接受概率来避免陷入局部最优解，从而有可能找到全局最优解。

**2.2.2 关键操作**

要将模拟退火算法运用到此问题中，我们需要将之前的数学模型抽象到算法上。

1.**初始化：**初始温度由配置文件给定，初始解取顺序序列（1，2，3，4……）

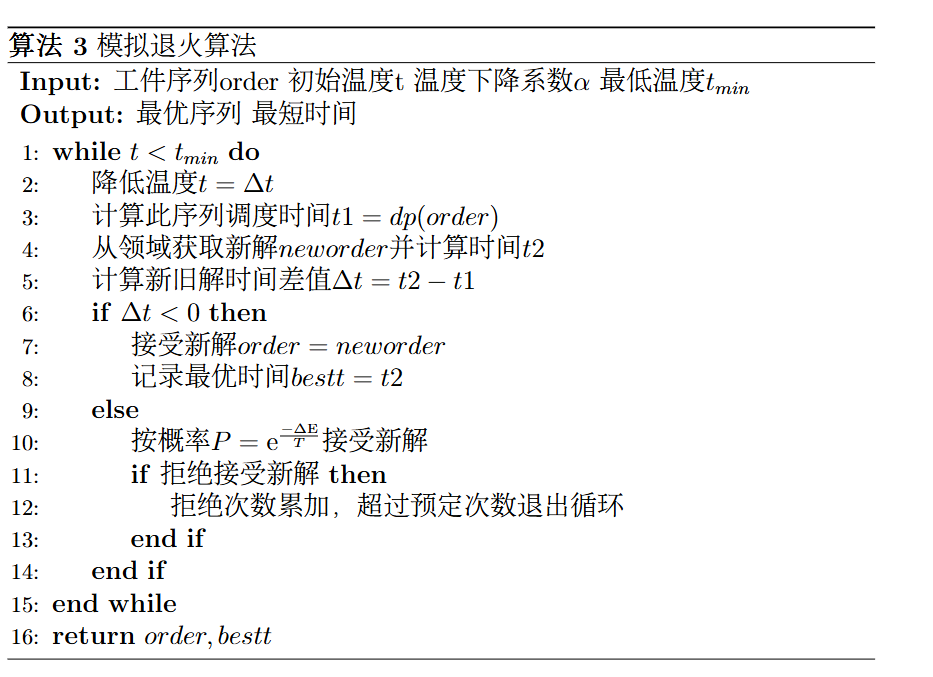
2.**从领域获得新解：**按照**2.1.3**中的洗牌算法得到新解

3.**计算目标函数差：**调用**2.1.1**中的算法，计算新解与旧解的时间差值**Δ**

4.**接受判断：**若**Δ**<0,接受新解，并更新最优解为新解，否则按概率接受新解。

其余过程同**2.2.1**。

**2.2.3 算法伪代码**



* 1. **遗传算法**

**2.3.1 算法简介**

遗传算法（Genetic Algorithm, GA）是一种模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，用于解决优化和搜索问题。它通过构建一个模拟环境，允许候选解“个体”通过适应度评价进行“生存竞争”，适应度高的解有更高的繁殖机会。通过这种机制，遗传算法寻求在给定的问题空间内找到最优或可行的解。算法流程如下：

**1. 种群初始化**：随机生成一个初始种群，种群中的每个个体代表一个可能的解。

**2. 评估：**计算每个个体的适应度，适应度通常是个体质量的量化指标。

**3. 选择：**根据个体的适应度进行选择，适应度高的个体有更大的机会被选中参与下一代的繁殖。

**4. 交叉：**随机选择一对个体作为父母，按照交叉概率交换它们的部分基因，产生新的个体。

**5. 变异：**以一定的变异概率修改个体的部分基因，以维持种群的多样性。

**6. 进化：**用新生成的个体替换掉原种群中的一部分个体，通常是替换掉适应度较低的个体。

**7. 终止条件判断：**如果达到预设的终止条件（如迭代次数、适应度阈值等），则停止算法；否则，返回步骤 2继续迭代。

**2.3.2 关键操作**

要将遗传算法运用到此问题中，我们需要将之前的数学模型抽象到算法上。

1.种群初始化：根据配置文件给出的种群大小确定种群规模，若没有指定，令种群规模为工件数量的平方。

2.产生父本群：首先选择要生成子代的父本，父本数量由随机数决定。每个父本对有两个父本

3.确定父本对：每个父本对中的父本由二元竞争办法产生，具体来说，从初始种群随机选取两个个体，选择其中适应度（即工件序列的调度时间）较小的。

4.交叉操作：按照交叉概率将父本群中的每对父本进行交叉操作产生子代，具体操作如下：

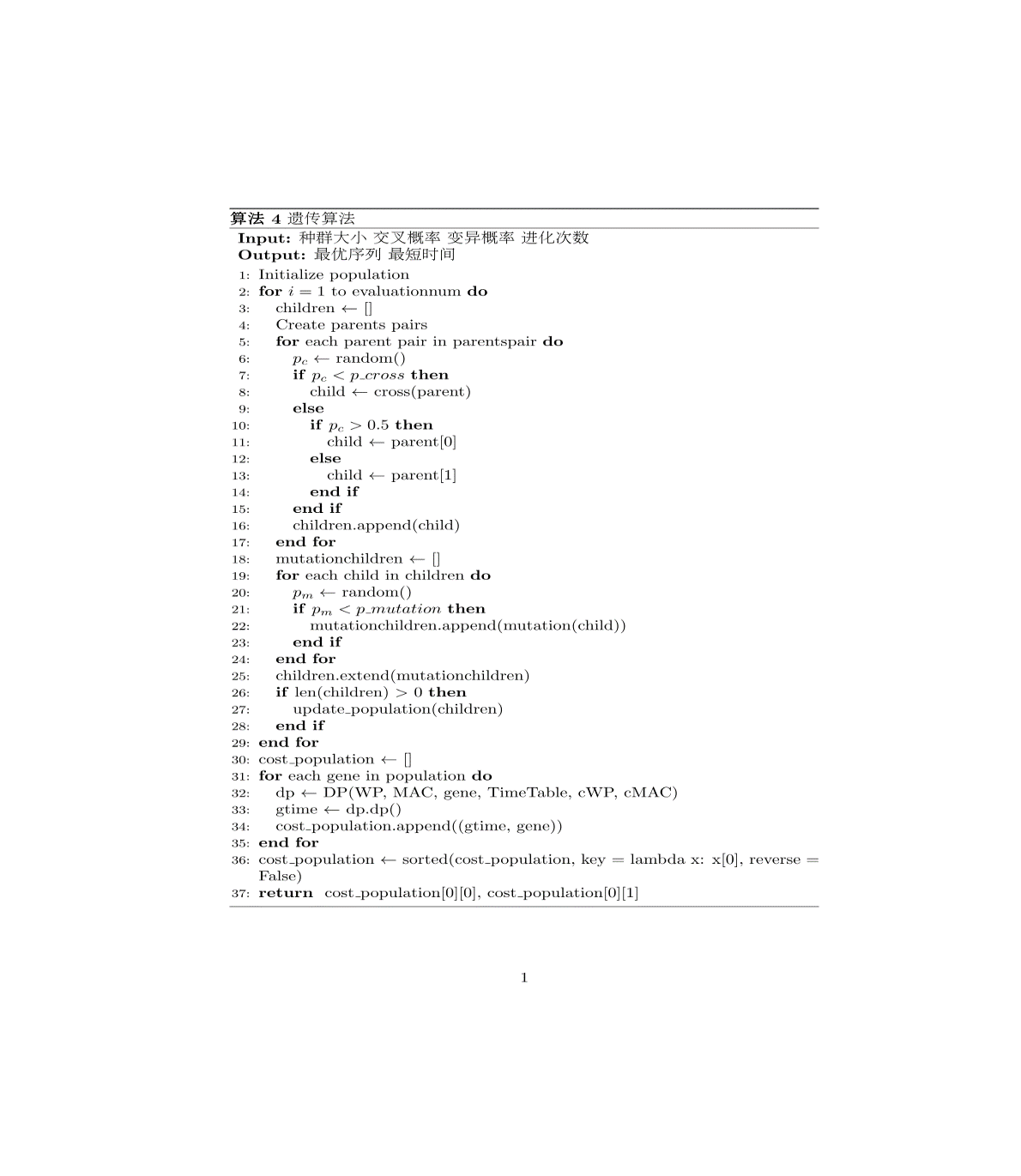
* 选择两个父本，定义两点p1，p2将基因分为三段
* 将父本1中间段复制给子代
* 从父本2中选择不重复的元素到子代中

通过上述交叉操作，父本中的部分优良基因得以传承，同时父本2的片段增加了种群的多样性

5.变异操作：按照变异概率选择一个子代，随机交换其中两个基因，视作变异。同时将该子代和变异后的子代都加入子代群中，防止变异将优良基因丢失。

6.更新种群：计算当前种群所有个体的适应度（调度时间），按逆序排序种群。再计算子代群所有个体的适应度，按顺序排序。将子代群中第一个不在原种群且适应度更好的个体加入种群，删除不良个体。

**2.3.3 算法伪代码**



* 1. **算法创新与优化**

对于遗传算法，初始种群是随机产生的，那么种群的质量和多样性往往不可控，这可能会影响到结果收敛的速度，尤其是在工件数量明显增多的情况下。因此，本文在种群初始化的时候，加入了启发式算法NEH[2]用于搜寻一个较优的初始解来减小收敛速度。

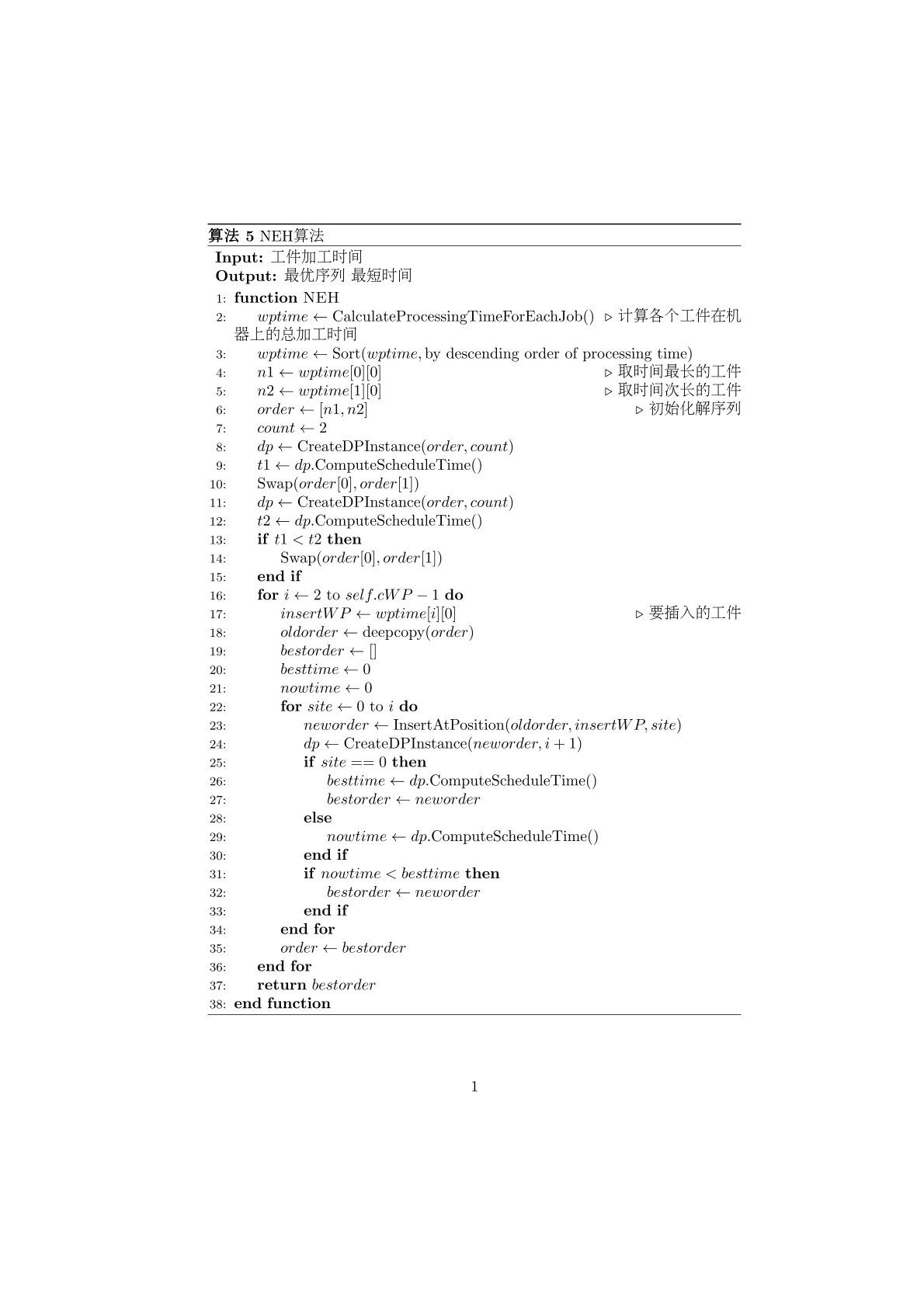
NEH算法的基本思想是：

1.计算每个工件的总加工时间，以此将工件按逆序排列。

2.取序列中前两个工件，计算两种调度方案的最短时间，取时间最短的加入调度序列***π***。

3.从序列第三个工件开始，遍历每个工件***i***，将其插入到调度序列***π***中,取生成的***i***个序列中调度时间最短的序列作为新的调度序列，直到遍历完成，算法结束。

下面给出NEH算法的伪代码：



在程序中，可以自定义是否NEH算法，这点在后文中将会讨论。

**3.仿真实验**

## **3.1 实验设置**

3.1.1 **实验环境：**

* **操作系统：**Windows11操作系统
* **硬件配置：**AMD RYZEN 7 + NVIDIA RTX3060
* **实验语言：**Python 3.10
* **辅助库：**直接安装requirements.txt即可

**3.1.2 运行实验：**

实验采用文件中的**data.txt**数据，取其中的11个例子作为实验对象。可以直接在主目录下运行各个算法包中的**Main.py**文件进行实验。

用户可以修改**Config.py**文件，来选择计算哪个例子，应用什么参数，以找到最适合的解。本文中最佳的参数详见**3.2.4**

考虑到随工件和机器数量增长，算法的复杂度陡然上升，我们取三个合适又较不费时的例子作为低，中，高工件和机器数量的代表来探究算法在不同情形下的效果。这两个例子分别为**data.txt**中的用例1，4，6。实验中单位都为秒s。

## **模拟退火算法参数实验**

为探究单一变量对实验结果的影响，我们要控制其它变量不变。在此定义默认的初始变量（**并非最优参数**），这些变量在探究其它变量的影响时对解的效果和时间影响较小：

**表3.2.1 变量默认值**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 变量 |  |  |  | 默认值 |  |
|  | 初始温度 |  |  |  | 10000 |  |
|  | 下降系数 |  |  |  | 0.9 |  |
|  | 截止温度  退火次数 |  |  |  | 0.01  1 |  |

**3.2.1 初始温度对实验结果的影响**

初始温度通常选用温度较高的值，这样有助于深度挖掘领域的解，将退火过程进行的较为完整，有助于算法收敛，但温度过高就会导致算法时间过长，得不偿失。因此我们进行多次实验，探究在各个大小的用例下初始温度对实验结果的影响，控制其它变量不变保持初始温度变化。得到以下实验结果：

**表3.2.2 初始温度对实验结果的影响**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 初始温度 | 用例0结果 | 用例0时间 | 用例4结果 | 用例4时间 | 用例6结果 | 用例6时间 |
| 1 | 7038 | 0.03 | 9533 | 0.03 | 1428 | 0.09 |
| 10 | 7038 | 0.09 | 9431 | 0.09 | 1402 | 0.24 |
| 100 | 7038 | 0.17 | 9431 | 0.16 | 1411 | 0.44 |
| 1000 | 7038 | 0.27 | 9436 | 0.25 | 1389 | 0.67 |
| 10000 | 7038 | 0.39 | 9431 | 0.36 | 1435 | 0.94 |
| 100000 | 7038 | 0.52 | 9431 | 0.49 | 1442 | 1.28 |
| 1000000 | 7038 | 0.71 | 9431 | 0.64 | 1440 | 1.76 |

从上表可以看出，在工件和机器数量较低的情况下，初始温度对结果的影响有限，这是因为收敛速度较快，并不需要深度运算。而当情况较为复杂时，温度过低不能算法进行的深度不够，不能挖掘到最优解，过高又会延长搜寻时间，还会导致解从最优解中跳出，得不偿失。

**3.2.2 下降系数对实验结果的影响**

该系数同初始温度一样，也会影响算法的收敛速度和挖掘深度，具体情况我们依然通过以下三个例子说明：

**表3.2.3 下降系数对实验结果的影响**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 下降系数 | 用例0结果 | 用例0时间 | 用例4结果 | 用例4时间 | 用例6结果 | 用例6时间 |
| 0.9 | 7074 | 0.01 | 10424 | 0.01 | 1460 | 0.03 |
| 0.99 | 7038 | 0.09 | 9907 | 0.12 | 1403 | 0.34 |
| 0.999 | 7038 | 1.00 | 9431 | 1.09 | 1371 | 3.09 |
| 0.9999 | 7038 | 9.42 | 9431 | 10.59 | 1376 | 29.29 |
| 0.99999 | 7038 | 96.32 | 9431 | 100.76 | 1336 | >200 |
| 0.999999 | 7038 | >200 | 9431 | >200 | 1335 | >2000 |

从上表可以看出，下降系数是对实验结果和时间影响最大的系数，下降系数每增大十倍，时间就会爆发式增长。在数量少的时候，增大下降系数对实验结果机会没有影响，但数量增加时，增大下降系数明显能取得更优解，但代价是花费更多的时间。因此，选取合适的下降系数至关重要。

**3.2.3 截止温度对实验结果的影响**

该系数同之前的一样，会影响算法的收敛速度和挖掘深度，具体情况我们依然通过以下三个例子说明：

**表3.2.4 截止温度对实验结果的影响**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 下降系数 | 用例0结果 | 用例0时间 | 用例4结果 | 用例4时间 | 用例6结果 | 用例6时间 |
| 0.1 | 7162 | 0.01 | 10424 | 0.01 | 1488 | 0.02 |
| 0.01 | 7652 | 0.02 | 9854 | 0.02 | 1484 | 0.05 |
| 0.001 | 7038 | 0.03 | 9666 | 0.03 | 1494 | 0.09 |
| 0.0001 | 7038 | 0.05 | 9642 | 0.05 | 1557 | 0.13 |
| 0.00001 | 7038 | 0.09 | 9882 | 0.06 | 1472 | 0.17 |
| 0.000001 | 7038 | 0.10 | 9882 | 0.08 | 1451 | 0.23 |

由上表可以看出，截止温度对于算法时间的影响不大，但对最优解的结果影响较大，其影响效果类似于初始温度，都是过低时无法达到最优，而过高又跳出最优解。选择一个合适的截止温度既能提升算法准确度，又基本不影响算法运行的时间，可谓是两全其美。

**3.2.4 最优参数**

在多次实验上测试多种用例，最终得出以下最优参数表，在各个用例上的平均时间和解的质量最好。

**表3.2.5 模拟退火算法参数**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 变量 |  |  |  | 默认值 |  |
|  | 初始温度 |  |  |  | 10000 |  |
|  | 下降系数 |  |  |  | 0.999 |  |
|  | 截止温度 |  |  |  | 0.001 |  |

## **遗传算法参数实验**

同模拟退火算法实验，为探究单一变量对实验结果的影响，我们要控制其它变量不变。在此定义默认的初始变量（**并非最优参数**），这些变量在探究其它变量的影响时对解的效果和时间影响较小：

**其中，种群数量取-1代表不预置参数，取工件数量的平方作为种群数量。**

**表3.3.1 变量默认值**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 变量 |  |  |  | 默认值 |  |
|  | 种群数量 |  |  |  | -1 |  |
|  | 基因交叉概率 |  |  |  | 0.9 |  |
|  | 基因突变概率  种群进化次数  算法循环次数 |  |  |  | 0.9  10  1 |  |

**3.3.1 种群数量对实验结果的影响**

种群数量决定了初始解空间的大小，对基因多样性有着一定影响，太小的数量不足以产生足够的样本，太大的数量会造成运行时间的增加，我们采用上述三个例子，进行如下实验分析：

**表3.3.2 种群数量对实验结果的影响**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 种群数量 | 用例0结果 | 用例0时间 | 用例4结果 | 用例4时间 | 用例6结果 | 用例6时间 |
| -1(121) | 7378 | 0.51 | 9436 | 1.29 | 1489 | 4.88 |
| 100 | 7281 | 0.54 | 9500 | 0.47 | 1512 | 1.87 |
| 1000 | 7038 | 5.39 | 9818 | 6.08 | 1494 | 18.71 |
| 10000 | 7038 | 55.38 | 9431 | 72.72 | 1446 | 239.77 |

由上表可以看出，种群数量太少，算法结果不太理想。反之，数量太多又使得计算时间明显上升。选择默认值即前文所提到的工件数的平方，既能得到较优的解，也能大大缩减算法的计算时间。

**3.3.2 基因交叉概率对实验结果的影响**

基因交叉概率决定了种群的多样性，较低的概率父本容易直接复制给子代，使得新解不容易出现，算法很可能陷入局部最优解。而太高的交叉概率又不能很好的保留父本优秀的基因，使得算法收敛速度降低。为了探究此概率的影响，我们进行如下实验分析：

**表3.3.3 基因交叉概率对实验结果的影响**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 交叉概率 | 用例0结果 | 用例0时间 | 用例4结果 | 用例4时间 | 用例6结果 | 用例6时间 |
| 0.85 | 7472 | 0.47 | 9500 | 1.24 | 1489 | 4.16 |
| 0.9 | 7378 | 0.50 | 9436 | 1.23 | 1489 | 4.25 |
| 0.95 | 7358 | 0.48 | 9436 | 1.44 | 1500 | 4.59 |
| 1 | 7270 | 0.48 | 9436 | 1.28 | 1489 | 4.19 |

由上表可知，交叉概率对于算法的计算时间几乎没有影响。总体来说，越大的交叉概率能产生越优的解，说明基因多样化的同时没由丢失原来优秀的基因，这在一定程度上证明了算法交叉操作里三段交叉法对优良片段传承的有效性

**3.3.3 基因突变对实验结果的影响**

该变量的影响同**3.3.2**，我们进行如下实验分析：

表**3.3.4 基因突变对实验结果的影响**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 突变概率 | 用例0结果 | 用例0时间 | 用例4结果 | 用例4时间 | 用例6结果 | 用例6时间 |
| 0.85 | 7184 | 0.63 | 9854 | 1.05 | 1487 | 3.98 |
| 0.9 | 7378 | 0.48 | 9436 | 1.34 | 1489 | 4.32 |
| 0.95 | 7123 | 0.56 | 9436 | 1.37 | 1478 | 4.88 |
| 1 | 7094 | 0.66 | 9436 | 1.13 | 1485 | 3.89 |

该测试得到的结论与**3.3.2**相似，越大的突变概率可能会得到越优的解，同时对算法运行时间几乎没有影响。

**3.3.4 种群进化次数对实验结果的影响**

直觉上来说，种群进化的次数越多，种群中的优良个体就越多。但进化到一定次数后，算法可能会陷入部分解无法跳出，此时再进化种群，只会浪费时间和资源。为探究这种说法是否合理，我们进行如下实验分析：

**表3.3.5 种群进化次数对实验结果的影响**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 进化次数 | 用例0结果 | 用例0时间 | 用例4结果 | 用例4时间 | 用例6结果 | 用例6时间 |
| 10 | 7378 | 0.51 | 9436 | 1.46 | 1489 | 4.35 |
| 50 | 7038 | 8.36 | 9487 | 18.37 | 1468 | 62.09 |
| 100 | 7038 | 29.54 | 9730 | 63.75 | 1480 | 233.28 |
| 150 | 7038 | 64.36 | 9481 | 133.74 | 1475 | 513.99 |

由上表可以看出，实验结果符合直觉，在中等进化次数时，实验结果和运行时间都较优秀，而在较高的进化次数下运行时间大大提高，算法解的质量并没有太大提升，在较低的进化次数下实验结果不太稳定。

**3.3.5 最优参数**

在多次实验上测试多种用例，最终得出以下最优参数表，在各个用例上的平均时间和解的质量最好。

**表3.3.6 遗传算法参数**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 变量 |  |  |  | 默认值 |  |
|  | 种群数量 |  |  |  | -1 |  |
|  | 交叉概率 |  |  |  | 1 |  |
|  | 变异概率  进化次数  循环次数 |  |  |  | 1  50  10 |  |

## **算法对比**

考虑到算法的随机性，我们对每个算法取十次循环，下面是十次循环的对比，单位为s:

**表3.4.1 SA与GA的结果对比**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 用例 |  | 算法 |  | 运行结果 |  | 运行时间 |  |  |
|  | 0 |  | SA |  | 7038 |  | 10.63 |  |  |
|  |  | GA |  | 7038 |  | 8.36 |  |  |
|  | 1 |  | SA |  | 6269 |  | 8.06 |  |  |
|  |  |  | GA |  | 6269 |  | 11.62 |  |  |
|  | 2 |  | SA |  | 5066 |  | 6.51 |  |  |
|  |  |  | GA |  | 5066 |  | 25.55 |  |  |
|  | 3  4  5  6  7  8  9  10 |  | SA  GA  SA  GA  SA  GA  SA  GA  SA  GA  SA  GA  SA  GA  SA  GA |  | 6655  6666  9431  9431  6961  7044  1335  1442  1839  1887  932  986  1749  1799  2776  2884 |  | 9.56  56.67  11.13  179.2  9.53  35.2  32.3  817.9  62.7  1047.9  17.6  288.5  46.33  771.9  90.4  >2000 |  |  |

## **算法改进**

如前文所说，本文在遗传算法求解种群时设计了NEH启发式算法来得到较优解。经多次实验测试，启用NEH 算法在工件机器数量多的情况下确实能减少收敛速度，提高准确度，但很容易陷入某一局部最优解而无法更新，综合来看，启用该算法后，解的质量有所提高，但仍然比不上模拟退火算法，运算时间亦是，故在此不在展开叙述此算法实验。

**4.最优调度方案**

实验得到最优调度方案如下表：

**表4.1 最优调度方案**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 用例 |  |  | 序列 |  |  | 时间 |  |  |
|  | 0 | [7, 0, 2, 4, 1, 10, 6, 3, 8, 5, 9]  [3, 1, 0, 4, 2]  [3, 7, 5, 4, 1, 0, 6, 2]  [6, 7, 5, 9, 4, 8, 3, 0, 2, 1]  [2, 12, 5, 11, 4, 7, 8, 9, 13, 0, 1, 10, 6, 14, 3]  [1, 0, 2, 5, 7, 6, 4, 3]  [1, 12, 7, 10, 9, 14, 11, 8, 17, 2, 0, 16, 13, 3, 6, 5, 15, 4]  [9,12,3,16,10,4,13,6,11,8,17, 14,1,15,0,7,2,5]  [14, 3, 4, 1, 0, 7, 9, 12, 11, 6, 2, 13, 8, 5, 10, 15]  [11, 15, 10, 14, 13, 8, 0, 1, 9, 5, 4, 6, 12, 7, 3, 2]  [7, 8, 29, 39, 19, 34, 36, 3, 33, 30, 27, 26, 13, 28, 0, 31, 23, 38, 35, 20, 21, 2, 12, 15, 4, 9, 5, 14, 11, 1, 37, 10, 16, 17, 24, 6, 22, 32, 18, 25] | | | | | 7038 |  |  |
|  | 1 | 6269 |  |  |
|  | 2 | 5066 |  |  |
|  | 3 | 6655 |  |  |
|  | 4 | 9431 |  |  |
|  | 5 | 6961 |  |  |
|  | 6  7  8  9  10 | 1335  1839  932  1749  2776 |  |  |

# **5总结**

本文用六千多字详细介绍了如何用模拟退火和遗传算法求解置换流水车间调度问题(PFSP)，其中，为确定最优解和适合的参数，进行了多个变量的单一对照实验，最终为两个算法求得了该用例适宜的参数，并得到了每个用例的最优解和调度序列。

本文亦有许多不足之处：遗传算法的求解时间普遍较长，在面对工件和机器数量多的用例上表现得也不尽人意，据实验结果推测是种群在进化到某一阶段总会停滞，并且陷入局部最优解中。尽管尝试了使用

NEH启发式算法解决该问题，但结果与模拟退火仍有差距。在种群进化到某一程度后丢弃部分个体再重新加入初始化种群或许对解决这一问题有所帮助。然而限于个人能力和精力有限，未能再对此展开详细测试。

参考文献:

[1] 赵文成, 赵红. 基于产业价值链的我国制造业竞争战略研究. 中国科学工程, 2008, 10(9): 54-59

[2] 王书婷. 基于遗传变邻域算法的置换流水车间调度问题研究[D]. 湖北:华中科技大学,2013. DOI:10.7666/d.D413934.