

基于强化学习的智能 RGV 动态调度策略

摘要

在智能加工系统中，无人驾驶的智能自动引导车 RGV 通过动态调度算法自行调配 8 台数控机床 CNC 的加工流程。在该系统中 RGV 如何根据当前时刻其观测到的环境因素进行决策是求解问题的关键。本文针对该系统加工流程与所给的数据信息用 Python 编程语言搭建了一个可视化的系统仿真环境，基于该仿真环境制定了一套合理的动态调度模型与求解算法并对模型的合理性以及算法的有效性做出了分析。

对于问题一：

在**第一种情况**中，由于系统无随机因素，为简单的静态环境，因此通过分析对系统产出与系统环境因素之间的关系，我们得出了 CNC 启用策略与系统产出的关系，并由此算出待优化的目标函数 $y = n \times \frac{(8 \times 3600 - \text{Init} - \text{Wait})}{\text{Loop}}$ ，进而求解。

在**第二种情况**中，考虑到实践生产中，生产不同成料的系统环境参数不尽相同，状态数量多且复杂。对此我们开创性的使用在**强化学习**^[1]领域中的 **Q-learning 算法**，通过构建不同环境状态下 RGV 动作产生的价值函数 $Q(o, a)$ 来构建 RGV 的动态调度模型 $A(t, s; Q)$ ，然后以最优化作业效率 score 求解目标函数：

$$\operatorname{argmax}_Q \text{ score} = H(A(t, s; Q(o, a)))$$

在**第三种情况**中，系统引入了 CNC 故障机制，系统中的动态因素导致 RGV 随时发生重调度，因此 RGV 的调度模型算法需要对于已知的状态能够作出最优的决策，同时还要有从未知状态中观测学习并修改调度策略的能力。对此我们将 Q-learning 算法中的学习机制改进后引入到 RGV 动态调度模型中，使得调度算法拥有更强的鲁棒性。

对于问题二：

问题二提供了我们 3 组系统环境参数，并要求我们以此检验问题一中得到的算法的效果并提供系统在对对应环境下工作后的物料加工结果。对此我们分别将 3 组环境参数带入仿真环境并用问题一求得的动态调度模型来检验效果。然后我们对结果作了可视化方案发现迭代算法得到很好的收敛，而后我们又通过量化物料平均损耗率来量化 RGV 作业效率，从而分析检验得出 Q-learning 算法模型具有较强的鲁棒性。

关键词： 仿真环境；可视化； Q-learning；强化学习；动态规划

一、 问题重述

某智能加工系统由 8 台计算机数控机床（Computer Number Controller, CNC）、1 辆轨道式自动引导车（Rail Guide Vehicle, RGV）、1 条 RGV 直线轨道、1 条上料传送带、1 条下料传送带等附属设备组成（见图 1）。

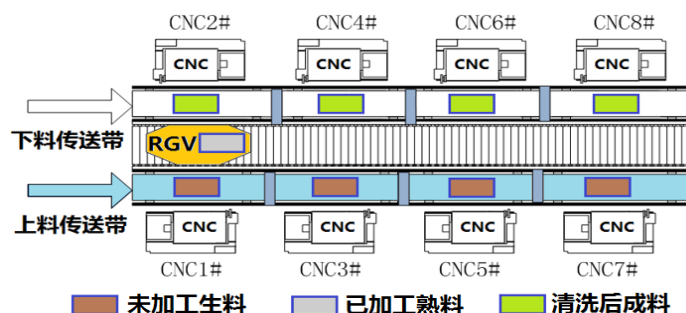


图 1：智能加工系统示意图

在工作正常情况下，当 CNC 空闲时，它会向 RGV 发出需求信号，RGV 在收到某 CNC 的需求信号后，它会根据先前设计好的策略自行确定该 CNC 的上下料作业次序，依次进行上下料操作，其中 RGV 为偶数编号 CNC 一次上下料所需时间要大于为奇数编号 CNC 一次上下料所需时间。在完成上下料后，RGV 需要先对熟料进行清洗操作（熟料清洗后为成料），且在清洗过程中 RGV 不可移动。某 CNC 完成一个物料的加工作业任务后，立刻向 RGV 发出需求信号。如果 RGV 没能即刻到达为其上下料，该 CNC 就会出现等待。

在实际的生产中，该系统可能会面对以下情况：

1. 一道工序的物料加工作业情况：每台 CNC 安装同样的刀具，物料可以在任一台 CNC 上加工完成
2. 两道工序加工作业情况：需要有不同的 CNC 安装不同的刀具依次完成第一和第二道工序。，每台 CNC 只能完成其中的一道工序，且在加工过程中不能更换刀具。
3. CNC 在加工过程中可能发生故障（据统计：故障的发生概率约为 1%）的情况，每次故障排除（人工处理，未完成的物料报废）时间介于 10~20 分钟之间，故障排除后，CNC 为空闲状态。

结合以上三种情况以及系统运转流程，我们可以知道 RGV 上下料必须执行清洗操作，否则无法进行下一步操作。因此该调度算法的核心在于如何根据不同 CNC 机器的优先级以及 RGV 当前位置和历史执行的指令信息，作出最高效的移动策略以及上下料

策略。所以我们要探讨解决的问题为：

1. 对一般问题进行研究，给出 RGV 动态调度模型和相应的求解算法；
2. 利用问题一中的调度算法，结合给定参数，验证 RGV 的调度策略和作业效率。

二、 模型假设

1. 上料传送带传送生料数量充足且传送时间小于 RGV 移动到目标 CNC 的时间；
2. CNC 加工时间远大于 RGV 上下料时间以及 RGV 的单位移动时间；
3. RGV 的清洗时间非常短，因此当仅包含清洗不包含将工料放入下料传送带的动作时，时间消耗量不计；

三、 符号说明

| 符号 | 意义 | 符号 | 意义 |
|----------------|--------------------------------------------------------------------------------------------------------|----------------|--------------------------------------------------------------------|
| i, j | i 表示 RGV 移动前所处位置 j 表示 RGV 移动后所处位置 $i, j \in \{1, 2, 3, 4\}$ ，位置 1 在 CNC1#和 CNC2#正中间，其他位置以此类推 | k | k 为 CNC 编号， $k \in [1, 8] \wedge k \in \mathbb{Z}$ |
| $move_{i,j}$ | 表示 RGV 从位置 i 移动到位置 j 需要的时间，其中 1 号位置为 CNC1 的位置 | $exchange_k$ | 表示给编号为 k 的 CNC 上料需要的时间 $k \in [1, 8] \wedge k \in \mathbb{Z}$ |
| $wash_k$ | 表示清洗编号为 k 的 CNC 的熟料需要的时间 $k \in [1, 8] \wedge k \in \mathbb{Z}$ | $error_k$ | 表示解决编号为 k 的 CNC 故障需要的时间 |
| $process1_k$ | 表示编号为 k 的 CNC 加工完成一个一道工序的物料所需时间 | $process2_k^1$ | 表示编号为 k 的 CNC 加工完成一个两道工序物料的第 1 道工序所需时间 |
| $process2_k^2$ | 表示编号为 k 的 CNC 加工完成一个两道工序物料的第 2 道工序所需时间 | $Episode_k$ | RGV 在学习环境下工作的第 k 个回合 |
| s | 仿真环境的环境状态 Systemstate | o | RGV 从环境状态中观察得到的观察状态 Observation |
| T | 一个回合中环境的时间状态 | a_t | 在 t 时刻 RGV 执行的动作 |
| $Q(o,a)$ | Q-learning 中的价值函数，表示观察状态 o 下动作 a 产生的价值 | $A(s,t;Q)$ | 应用价值函数 Q 的动态调度模型，表示 RGV 在 t 时刻环境 s 下根据 Q 作出的动作决策 |

四、 问题分析

在现代自动化的工业生产模式下，生产过程中出现不确定因素（如机器故障）的频率明显高于传统模式下，这些动态事件的发生大大提高了作业车间对于调度策略的要求。因此，考虑到系统的内在组成约束以及动态事件特征，我们对于一个智能加工系统进行了动态调度问题的研究。

4.1 问题 1

问题一要求我们研究一般问题的 RGV 动态调度模型和求解算法。根据该智能加工系统的组成与作业流程我们知道，RGV 的调度策略与 CNC 的工作机制密切相关，据此我们可以先建立系统的仿真环境模型并量化系统中的各项环境因素，然后根据仿真模型研究系统的动态调度模型与求解算法。求解算法的目标是使系统作业效率即动态调度模型中系统在一个班次内的加工物料数尽可能的多。

对于第一种只有一道加工工序并且不考虑 CNC 故障的情况，我们从系统角度出发考虑影响系统的作业效率的因素有 RGV 的调度策略和 CNC 的启用策略，从这两个方向考虑我们可以进一步分析 RGV 的调度与 CNC 启用数量之间的关系，从而分析该情况下最优的调度算法。

在第二种需要两道加工工序且不考虑 CNC 故障的情况中，每台 CNC 负责的加工工序不同，因此在两道工序加工时间与分布作为自变量引入问题中后，该变量与系统中其余变量的相互作用增加了系统的状态数量。对此我们考虑对强化学习 Q-learning 算法进行改进并以此求解该情况下系统的动态调度模型。该算法通过在仿真模型中建立优化模型挖掘 RGV 动作输入与系统状态之间的关系对照表，并以最大化作业效率为目标求解不同状态下 RGV 不同动作的参数权重表。

对于第三种引入 CNC 故障机制的情况，我们考虑该状况基于现实状况引入随机因素从而增加动态调度模型的鲁棒性。因此我们考虑在仿真环境中引入随机故障因素后再建立优化模型。

4.2 问题 2

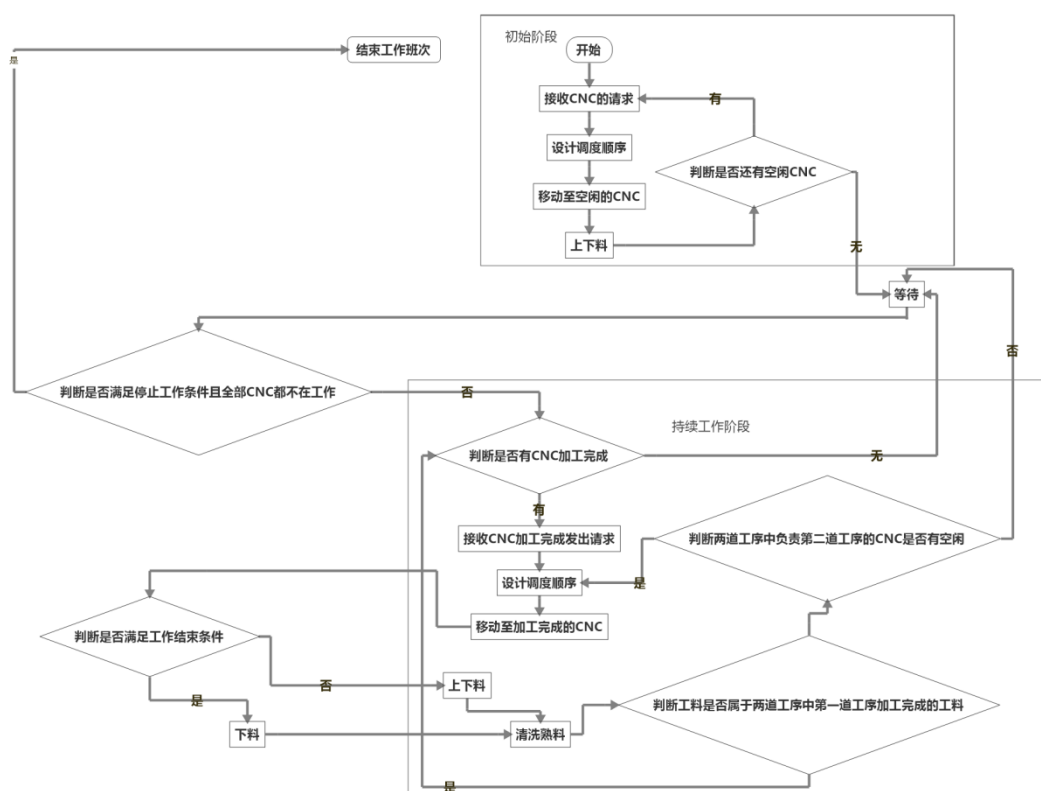
问题二给予我们三组系统作业参数要求我们给出调度模型并评估检验结果。我们将数据代入问题一中的优化模型中求解，输出每组参数在三种情况下的工料编号与上下料时间数据，求出调度策略与作业效率，并对结果进行分析。

五、 模型建立与求解

5.1 建立仿真环境

为了求解智能加工作业系统的动态调度模型，我们首先需要根据提供的智能加工系统作业流程构建 RGV 的基本运作流程图。然后根据流程图我们在虚拟环境中搭建系统加工的仿真环境，并以此为基础求解加工系统的动态调度模型。

首先我们根据智能系统的加工作业流程用 MindManager 构建智能加工系统中 RGV 的基本运作流程图：



藉此我们可以得到 RGV 的基本运行流程：

在一个工作班次的初始阶段，RGV 处于“开始”状态，等待 CNC 发出的加工请求，然后 RGV 会根据发出请求的 CNC 自行设计调度顺序并开始工作。当所有可用于加工的 CNC 都在工作时，RGV 会处于等待状态，等待一台 CNC 加工完成，此时 RGV 进入持续工作状态。

在持续工作状态中的 RGV 接收到加工完成的 CNC 发出请求后，RGV 会自行设计调度顺序，然后按顺序给完成加工的 CNC 执行上下料操作。

当满足工作停止条件时（如接收到工作停止信号），RGV 在 CNC 上执行操作时将不会

再给一道工序或两道工序中负责第一道工序的 CNC 上料,但它依然会获取 CNC 上的熟料,清洗,并视情况放入下料传送带还是放入第二工序的加工 CNC。

在确保所有的 CNC 上都没有工料并且满足工作停止条件时,RGV 将会回到初始位置,进入停止工作状态。

然后我们构建一个包含了 RGV 可以执行的所有动作的对照表(图示为表格中的一部分内容,全表随附件“RGV 动作对照表”):

| 动作编号 | 消耗时间 | 移动步数 | 动作类型 | 操作的CNC编号 |
|------|-----------|------|---------------|----------|
| 0 | 1 | 0 | 等待 | 0 |
| 1 | MOVE1 | 1 | 向右1步 | 0 |
| 2 | MOVE1 | -1 | 向左1步 | 0 |
| 3 | MOVE2 | 2 | 向右2步 | 0 |
| 4 | MOVE2 | -2 | 向左2步 | 0 |
| 5 | MOVE3 | 3 | 向右3步 | 0 |
| 6 | MOVE3 | -3 | 向左3步 | 0 |
| 7 | EXCHANGE1 | 0 | 给1号CNC上料 | 1 |
| 8 | EXCHANGE1 | 0 | 给1号CNC下料 | 1 |
| 9 | EXCHANGE1 | 0 | 给1号CNC下料之后再上料 | 1 |
| 10 | EXCHANGE2 | 0 | 给2号CNC上料 | 2 |
| 11 | EXCHANGE2 | 0 | 给2号CNC下料 | 2 |
| 12 | EXCHANGE2 | 0 | 给2号CNC下料之后再上料 | 2 |
| 13 | EXCHANGE1 | 0 | 给3号CNC上料 | 3 |
| 14 | EXCHANGE1 | 0 | 给3号CNC下料 | 3 |

在上下料操作中,根据智能加工系统作业流程,RGV 一条机械臂上拥有两个机械爪,可以同时完成上料和下料,因此我们考虑 RGV 给 CNC 上料同时下料的操作时间和只上料的时间相同,但奇数编号的 CNC 和偶数编号的 CNC 上下料时间不同。

我们还构建了一个 RGV 和 CNC 所有状态的对照表(全表随附件“RGV 和 CNC 状态对照表”):

| RGV状态表 | | | | | | | | | | |
|-----------|------|------|------|-----|------------------|--------------------|-------------------|------------------------|-----------|----------------|
| 状态\编号 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 移动步数 | 向左3步 | 向左2步 | 向左1步 | 不移动 | 向右1步 | 向右2步 | 向右3步 | | | |
| 当前位置 | | | | | 在第1和第2台CNC之间 | 在第3和第4台CNC之间 | 在第5和第6台CNC之间 | 在第7和第8台CNC之间 | | |
| 正在执行的动作类型 | | | | 等待 | 移动 | 上料 | 下料 | 上下料 | 清洗作业 | 放置下料 |
| 携带的物料类型 | | | | 无 | 只需要一道加工并已被加工后的熟料 | 只需要一道加工并已被加工清洗后的成料 | 需要两道工序并已经过一道加工的熟料 | 需要两道工序并已经过一道加工并被清洗过的熟料 | 二道加工完成的熟料 | 二道加工完成并已清洗过的成料 |

| CNC状态表 | | | | | | | | |
|--------|--------|-----------|------------|--------------------|----|-----------|---|------------------------|
| 状态\编号 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 备注 |
| 工作状态 | 空闲 | 一道工序的加工 | 二道工序的第一道加工 | 二道工序的第二道加工 | 故障 | | | |
| 工作结束时间 | | | | | | | | 无编号，内容一个对应于总时间的结束时间的数值 |
| 台上物料 | 无 | 一道加工熟料 | | 需要第二道加工并已经过一道加工的熟料 | | 二道加工完成的熟料 | | |
| 工作时间 | | | | | | | | 无编号，内容为该CNC工作需要的的时间数值 |
| 工作类型 | 一道工序加工 | 二道工序第一道加工 | 二道工序第二道加工 | | | | | |

然后，我们规定仿真环境的一些额外环境因素（全表随附件“环境参数对照表”）：

| 参数 | 符号 | 说明 |
|----|--------|-----------------------------------------------|
| 时间 | time | 班次剩余的时间，初始值为28800（8个小时），在仿真环境中该数值将随RGV的行动逐步减少 |
| 奖励 | reward | 调度智能获得的奖励，用于引导RGV采取更优的调度策略 |
| 物料 | score | 初始值为0，每完成一个物料的加工便会加1，用于判断动态调度模型的完成效果 |

接着根据智能加工系统作业流程，我们添加一些规则用于完善仿真系统：

1. RGV 在获取 CNC 上的熟料后会即刻进行清洗，然后视工料状况将清洗完后的工料放入下料传送带或者负责第二工序的 CNC。

2. 基于第一条，RGV 身上只能携带一个工料移动，并且该工料只能为经过一道加工需要第二道加工并已被清洗过的工料。

同时，我们规定仿真系统的输入仅为 RGV 的动作，系统的输出为 RGV 的状态，CNC 的状态，以及环境信息。

然后我们根据前面的 RGV 和 CNC 状态对照表分别定义 RGV 和 CNC 的状态列表：

CNC 状态列表 CNCstatus：

$\{(\text{status}_1, \text{overtime}_1, \text{worktime}_1, \text{worktype}_1, \text{materialtype}_1), \dots,$

$(status_8, overtime_8, worktime_8, worktype_8, materialtype_8))\}$

RGV 状态列表 RGVstatus:

$\{move, loc, materialtype\}$

以及根据环境因素对照表定义时间状态 T, 藉此, 我们定义在某一时刻 t 下系统的状态列表 $Systemstate_t$:

$\{Time, RGVstatus, CNCstatus\}$

为了减少算法的负荷, RGV 不需要接收完整的环境信息, 因此我们给 RGV 定义在该时刻 t 下 RGV 的观察状态列表 o_t :

$\{loc, materialtype, (status_1, worktype_1, materialtype_1),$
 $, (status_2, worktype_2, materialtype_2) \dots, materialtype_8)\}$

定义了环境状态与观察状态后, 我们定义一个班次作为仿真系统加工的一个回合 Episode, 在每一回合中, 系统拥有一个初始时间状态 T, 根据该智能系统加工流程我们可以得到 $T = 28800$ 。RGV 在每一回合的开始阶段将可以进行行动决策, 在不违反规则的情况下从 RGV 动作对照表中选择一个动作 a 执行, a 会对对应的环境状态 $Systemstate_t$ 造成影响, 并且该影响包含执行动作 a 所消耗的时间使得时间状态 T 获得更新, 更新公式如下:

$$T = T - atime$$

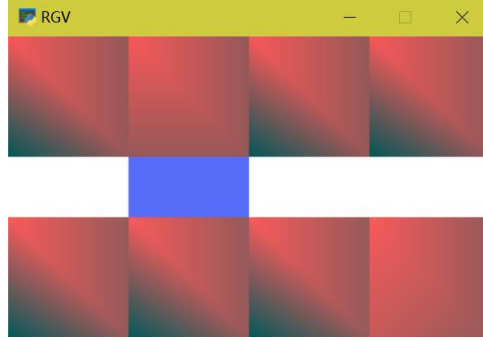
其中 atime 表示执行动作 a 所消耗的时间。

自然, 这其中也包括对 CNC 的状态影响。CNCstatus 中被加工的 CNC 状态将会变为加工状态(或者故障状态), 并且记录加工完成的时间 overtime(或者修理完成的时间), 当时间状态 T 减小至小于 overtime 时, CNC 恢复空闲状态, 并且台上工料状态 Materialtype 转变为熟料(或者无, 在故障的情况下物料作废)

然后根据 $Systemstate_t$, RGV 将会观察到状态 o_t , 并将 o_t 的结果输入动态调度模型, 并根据动态调度模型提供的结果进行下一次行动决策。

根据以上仿真环境构造计划, 我们用 Python 编程语言在系统平台 Windows 10 上搭建了该仿真系统, 完善了一些逻辑上的细节, 并做了一个简单的可视化界面用于观察算法运行状况以便引导我们对算法作出调整。

可视化信息输出实例:



（蓝色方块表示 RGV，红色方块表示 CNC，CNC 左下角的绿色标记用于标识 CNC 是否空闲，有标记为空闲）

文字信息输出实例：

```
获得reward: -5 采用动作: 2 RGV位置 [3] 剩余时间: 26250
RGV携带工料: [4]
RGV状态 [(-1, 3, 0, 4)] CNC状态 [(2, 26134, 400, 1, 3) (0, 26124, 378, 2, 0) (0, 26900, 400, 1, 3)
(0, 26889, 378, 2, 5) (0, 26358, 400, 1, 3) (0, 26433, 378, 2, 5)
(2, 25978, 400, 1, 3) (0, 26262, 378, 2, 5)]
```

（CNC 状态中的 8 个元组分别对应编号 1-8 的 CNC 的状态）

5.2 问题一

5.2.1 模型分析

我们首先定义模型的最终优化目标：

班次内加工物料数 score：班次内加工物料数是指在一个班次（8 小时内）智能加工系统所能完成加工的物料总数，该指标对作业效率拥有最直接的影响，我们的调度策略的最终目标即为尽可能地使该指标的数值变大。

然后基于仿真环境我们对于智能加工作业系统的三种情况分别进行分析：

(1) 情况一

根据智能加工系统作业流程，我们知道，在情况一中：

1. 所有的 CNC 都拥有相同的功能与加工时间
2. 最先开始加工的 CNC 必定会最先完成加工操作
3. 奇数编号的 CNC 的上料时间比偶数的 CNC 的上料时间快
4. 移动的单位越多需要的时间越多

在第一种只有一道加工工序且不考虑 CNC 故障的情况中，系统不受随机因素影响，此时调度模型为静态模型。其次，我们可以推导出影响系统产出的主要原因在与当 RGV 在给某 CNC 完成上下料操作和清洗操作后，应采取静止等待至其他 CNC 加工完成，还是

前往空闲的 CNC 的策略选择上。这个决策等价于 CNC 的启用策略,我们可以通过构造 CNC 启用台数关于系统产出成料的函数并求出使其产出成料最大的 CNC 启用台数,由此得出最佳的调度模型。

(2) 情况二

在第二种情况下,所有物料的加工都需要经过两道工序,一台 CNC 只能安装一种刀具并负责两道工序中的其中一道工序的加工,在该种情况下 CNC 的刀片分布影响 RGV 的调度策略,对于该智能加工系统,我们需要将智能加工作业参数作为自变量首先求得在对应参数下最优的 CNC 刀片排布,然后才能基于该排布求解最优的 RGV 调度策略。

但在实际计算时,要获取最优的 CNC 刀片排布需要知道在不同排布下执行最优 RGV 调度策略的作业效率,因此我们首先求解在固定 CNC 刀片排布的情况下 RGV 的调度策略。

对此问题,我们首先假设 CNC 拥有刀片排布 Arrangement。

$$\text{Arrangement} = \{k_1, k_2, \dots, k_8\}$$

其中 k_n 表示第 n 台 CNC 的刀片类型。

在该排布下,我们有 RGV 动态调度模型 scheduler 指引 RGV 在第 n 回合 Episode_n 下以最大化系统作业目标 score 进行动作决策 a 。

接着我们定义获取奖励 r , 每当 RGV 采取的一个动作有利于 score 增加, RGV 就会获得奖励 r_{action} , RGV 在每一个观察状态 o 下采取的一个动作 a 会产生的价值为 q , q 越大该动作 a 获得的奖励 r 越大,同理 q 越小或者为负则 r 越小或者为负,负数的 r 代表对 RGV 执行了不利于增加 score 的运动的惩罚。

据此,我们定义 $Q(o, a)^{[2]}$ 为在观察状态 o 下采取动作 a 产生的值为 q 的价值函数,即有:

$$q = Q(o, a) = Q(\text{Observation}, \text{action})$$

我们定义列表 $Q\text{-table}$, 当 $Q\text{-table}$ 为最大长度时, 其内容如下:

$$\begin{aligned} & \{(o, aq_1, aq_2, aq_3 \dots, aq_{\text{maxactions}})_1, \\ & (o, aq_1, aq_2, aq_3 \dots, aq_{\text{maxactions}})_2, \\ & \dots, \\ & (o, aq_1, aq_2, aq_3 \dots, aq_{\text{maxactions}})_{\text{maxstates}}\} \end{aligned}$$

在 $Q\text{-table}$ 中每一个 o 对应一个观察状态, 表中所有的 o 都是唯一的。 aq_n 表示在观察状态 o 中取第 n 个动作 a_n 所能够产生的价值 q_n 。其中 maxactions 表示所有动作的数量, maxstates 表示所有观察状态的数量。我们定义 $Q\text{-table}(o_t, a_n)$ 为取 $Q\text{-table}$ 中状态 o_t 动

作 a_n 的q值。

因此得到价值函数 $Q(o, a)$:

$$Q(o, a) = Q - \text{table}(o, a)$$

藉此，我们构建 RGV 的动态调度模型 $A(t, s; Q)$ ，其内容为:

1. 在 t 时刻，模型根据环境状态 s 提取观察状态 o_t
 2. 计算 $a_{max} = \operatorname{argmax}_{a_n} Q(o_t, a_n)$ ，其中 n 属于区间 $[0, 32]$
 3. 返还并执行动作 a_{max}
-

为了求解最优的动态调度模型，我们构建目标函数 $H(A)$:

$$\operatorname{argmax}_Q \text{score} = H(A(t, s; Q(o, a)))$$

H 为 RGV 在一个班次内受动态调度模型 A 驱动能够加工的物料数。

此时，对于第 n 个排布 Arrangement_n 我们可以通过我们在后面给出详细求解内容的 Q-learning 算法求出它的最优价值函数 Q_n ，从而得到在该价值函数 Q_n 下 RGV 所能产生的最优工作量 score_n ，然后根据 score_n 判断该刀片排布的有效性。

(3) 情况三

对于情况三，我们在仿真环境中引入 CNC 的故障机制:

当 CNC 开始加工时，进行条件判断:

生成区间在 $[0, 1]$ 间的随机数 random ，if $\text{random} > 0.99$:

CNC 状态转换为故障;

CNC 上的物料状态为空;

CNC 变为空闲的时间=当前时间状态 T -CNC 故障修理时间;

else:

CNC 正常工作;

然后，我们再求解在该环境下的价值函数 $Q(o, a)$ 。但是由于该环境不同于情况一和情况二的静态环境，该环境下的随机事件使得环境引入了动态因素。因此我们在将 Q 作为价值函数引入构建动态调度模型时，不同于情况二，我们不将价值函数 Q 作为动态调度模型 A 选取动作的唯一决策，我们将求解价值函数 $Q(o, a)$ 的求解算法 Q-learning 也

引入动态调度模型，使得 RGV 在遇到突发情况（CNC 故障）时，能够根据得到的环境状态随时更新价值函数 Q 。

5.2.2 模型建立与求解

（1）情况一

在情况一的分析中，我们分析了影响系统产出与 RGV 上下料时间、RGV 移动时间、RGV 清洗时间、CNC 加工时间的关系，进而得出了“CNC 的启用策略影响系统产出”这一结论。因此我们意图建立的调度模型的建立在我们的系统要启用多少台 CNC 基础上。

由于奇数编号的 CNC 的上料时间比偶数的 CNC 的上料时间快，当系统处于初始状态（所有 CNC 处于空闲状态），我们选择优先为奇数编号上料，可以使得 CNC 更早的开始加工，减少了 CNC 空闲的时间。因此我们选择以编号升序作为上下料顺序。由此我们可以给出该系统的调度模型：

设我们的调度系统启用编号为 1-n 的 CNC，其中 $n \in \{1, 2, 3, 4, 5, 6, 7, 8\}$

- 1 初始状态 RGV 处于初始位置（CNC1# 和 CNC2# 之间），其余 CNC 处于空闲状态
- 2 系统初始化阶段：
 - 2.1 RGV 依次为 CNC1#, CNC2#, CNC3#, ……CNCn# 上料
 - 2.2 RGV 返回初始位置
 - 2.3 若 CNC1#已完成加工，则系统进入循环作业阶段，否则 RGV 进入等待状态，直至 CNC1#完成加工后，系统进入循环作业阶段
- 3 循环作业阶段：（循环至班次时间结束）
 - 3.1 RGV 依次为 CNC1#, CNC2#, CNC3#, ……CNCn# 完成上下料以及熟料清洗工作并回到初始位置
 - 3.2 若此时 CNC1#尚未完成加工，则 RGV 等待至 CNC1#加工完成

在求解我们要启用多少台 CNC 之前，我们先做以下定义：

在上面的调度算法中，我们称 RGV 完成 2.1、2.2 这两个步骤的时间为系统初始化间（Init），称步骤 2.3 中 RGV 静止等待的时间为等待时间（Wait）为，在系统进入循环作业阶段后，RGV 每回到一次初始位置所需要的时间称之为循环时间（Loop）。

我们设系统八小时内产出为 y 个成料，可以得到：

$$\text{Init} = \lceil \frac{n}{2} \rceil \times \text{exchange}_1 + \lfloor \frac{n}{2} \rfloor \times \text{exchange}_2 + (n-1)\text{move}_{1,2} + \text{move}_{n,1}$$

$$\text{Wait} = \max(\text{process1}_1 - \text{Init}, 0)$$

$$\text{Loop} = \text{Init} + n \times \text{wash}_1 + \max(\text{process1}_1 - \text{Init} - n \times \text{wash}_1, 0)$$

$$y = n \times \frac{(8 \times 3600 - \text{Init} - \text{Wait})}{\text{Loop}}$$

我们希望通过选取适合的 n 使得系统在一个班次的产出达到最大，因此我们的优化目标为：

$$\underset{n}{\operatorname{argmax}} y, n \in \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$\text{s.t. } \text{exchange}_1 > 0, \text{ exchange}_2 > 0, \text{ wash}_1 > 0$$

$$\text{move}_{1,2} > 0, \quad \text{move}_{n,1} > 0, \quad \text{process1}_1 > 0$$

由于函数 y 不连续，无法通过求导，但是由于 n 的取值只有 8 个，因此我们可以尝试代入 n 所有可能的取值以及系统的其他参数（RGV 上下料时间、RGV 移动时间、RGV 清洗时间、CNC 加工时间）按照上式进行计算，比较出使目标函数 y 去最大值的 n 。即可得到系统应当启用编号 1- n 的 CNC 进行加工的策略。

（2）情况二

根据模型分析，为了求解动态调度模型 $A(t, s; Q)$ ，我们构建目标函数：

$$\underset{Q}{\operatorname{argmax}} \text{score} = H(A(t, s; Q(o, a)))$$

求解目标函数，便是求解在当前环境下，对于每一个观察状态 o 的每一个可执行动作 a 的价值 a_q ，即求解价值函数 $Q(o, a)$ ，使得在动态调度模型 $A(t, s; Q(o, a))$ 驱动下 score 的值最大。

我们基于 $Q - \text{learning}^{[3]}$ 算法实现对指定环境参数下的价值函数 $Q(o, a)$ 求解，该求解算法如下：

1. 定义学习率 α ，奖励衰减 γ ，贪婪度 ϵ 为可人工调整的自变量。
2. 初始化内容为空的 Q -table。
3. 对于每一回合 Episode:

在 t 时刻：

- 3.1 RGV 从环境状态 Systemstate 中提取观察状态 o 。

3.2 RGV 检查 Q-table, 如果当前观察状态 o 在 Q-table 中不存在则创建该 o 对应的行, 并将该 o 对应的 aq 全部初始化为 0.

3.3 生成一个区间在 $[0,1]$ 间的随机数 $random$,

if $random > \varepsilon$:

RGV 检查可选的动作列表并从中随机选择一个动作 a_t ;

else:

RGV 从 Q-table 中找到最大的 aq , 该 aq 可能不止一个, RGV 从中随机选择一个对应的 a_t 作为动作决策。

3.4 RGV 执行动作 a_t , 环境状态更新, RGV 从新环境状态中观察获得新观察状态 o_{new} , 并计算获得奖励 $r^{[4]}$

3.5 检查新观察状态 o_{new} 是否在 Q-table 中, 如果没有则为新观察状态 o_{new} 创建一行。

3.6 将 o_{new} 带入 Q-table, 查找 o_{new} 中该动作 a_t 的对应的价值 aq_{new}

3.7 如果下一时刻 $t+1$ 本回合 Episode 尚未结束, 求得 Q_{target} (其中 $\max_{action_{t+1}} \{Q(o_{new}, action_{t+1})\}$ 指使得 Q-table 中 o_{new} 对应的最大 aq 的值。):

$$Q_{target} = r + \gamma * \max_{action_{t+1}} \{Q(o_{new}, action_{t+1})\}$$

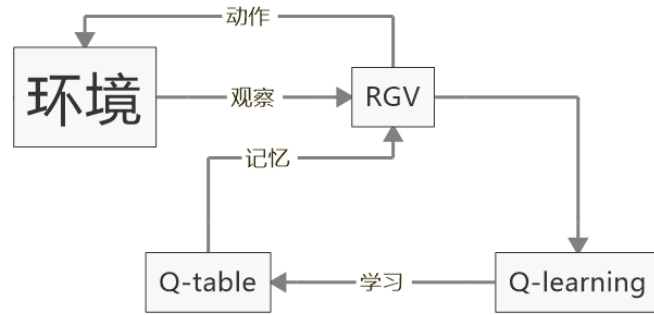
否则:

$$Q_{target} = r$$

3.8 更新 Q-table 中 o 对应行中 a_t 对应的 aq :

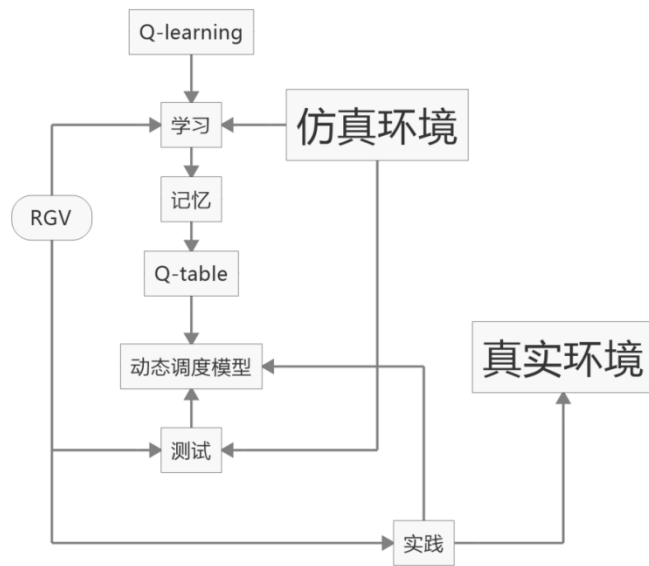
$$aq = aq + \alpha * (Q_{target} - Q(o, a_t))$$

此即为 Q-table 的更新算法 Q-learning, 该算法让 RGV 不断在仿真环境中尝试与学习, 并将学习到的内容存储在记忆 Q-table 中。



通过 Q-learning 算法,我们在任一静态环境任一刀片排布下可以求解 RGV 对于每一个观察状态的价值函数 $Q(o,a)$, 然后我们根据价值函数 Q 得到模型分析中所述的 RGV 的动态调度模型的记忆库 Q-table 实现动态调度。

整个系统的工作示意图:



总而言之, RGV 将先在仿真环境中通过 Q-learning 算法不断迭代并以此更新自己的记忆库 Q-table, 然后在测试环境与真实环境中运行时, RGV 将通过动态调度模型调用记忆库 Q-table, 从而实现动态智能调度。

然后, 根据 CNC 的数量, 我们求得刀片排布 Arrangement 的种数 Arrangementsum:

$$\text{Arrangementsum} = C_8^1 + C_8^2 + \cdots + C_8^7 = 254$$

刀片排布的数量较多, 但是我们通过分析得出, 若第一道工序所需时间与第二道工序所需时间要长, 为了使两阶段的生产速度相匹配, 第二道工序所用的刀片数量不应超

过第一道工序所用刀片数量。基于该策略，我们对搜索空间进行剪枝，将搜索空间缩小为 162 种刀片排布可能，提高了 36.2% 的搜索效率。同时为了能够在搜索最佳刀片排布时减少时间损耗，我们调整 Q-learning 在每个 Episode 的结束阶段检查当前获取的 score，根据记录的前面 3 个 Episode 的 score 来衡量是否还有必要进行接下来的 Episode：

在第 k 个 Episode $_k$ 的结束阶段，RGV 获得 score $_k$ 。

定义停机判断值 stop：

$$\text{stop} = |\text{score}_k - \frac{(\text{score}_{k-1} + \text{score}_{k-2} + \text{score}_{k-3} + \dots + \text{score}_{k-n})}{n}|$$

其中 n 作为自变量表示取当前 Episode 之前的多少个 Episode。

对于 stop，我们判断当 $\text{stop} < \text{limitchange}$ 时，系统停止该 Arrangement 上的动态模型求解，转向求解下一个 Arrangement 的动态模型。

对于所有的刀片排布 Arrangement，我们求出其对应的动态调度模型在一个 Episode 中所能获取的加工物料数 score，然后取 score 值最大的 Arrangement 作为本组环境参数下的最优刀片排布。

(3) 情况三

我们首先根据系统加工作业流程设定动态因素的引入机制，即 CNC 的故障机制。CNC 的故障机制为在加工过程中有 1% 的几率发生故障，修理所需时间为 10 到 20 分钟。藉此，我们设定 FixTime 为区间 10 到 20 分钟之间的一个随机值，然后设置当 RGV 给某台 CNC 上料时，有 1% 的几率判定该 CNC 将会发生故障，若判定成功，该 CNC 的故障修理时间为 FixTime 加上可能的加工时间 WorkTime，其中 WorkTime 为区间 0 到 CNC 加工完成时间中的一个随机值。

在引入动态因素后，假设我们已经根据情况二所述的求解算法得到了在该环境模拟状况下的价值函数 $Q(o, a)$ ，接下来我们将 Q 引入新的动态调度模型 $A(t, s; Q)$ ：

1. 在 t 时刻，根据环境状态 s 提取观察状态 s
2. 计算 $a_{\max} = \operatorname{argmax}_{a_n} Q(o_t, a_n)$ ，其中 n 属于区间 $[0, 32]$
4. 将 a_{\max} 与环境作用获得新观察状态 o_{t+1} 输入 Q-learning 算法更新 Q-table
5. 执行动作 a_{\max}

尽管我们引入的 Q-Learning 算法更新 Q-table 的过程，但我们没有将 Q-learning 中包含随机选择的动作选择算法也一并引入，因为在实际调用动态调度模型时，我们希望 RGV 执行动作决策时更加的谨慎而不像在仿真环境中训练那样敢于大胆的尝试。

5.3 问题二

5.3.1 模型分析

问题二提供了我们三组系统作业参数数据表，要求我们在三组系统作业参数数据表上对应题述的三种情况分别给出应用了问题一的动态调度模型与求解算法后，在一个班次中随着时间推移系统加工的物料的状况。

为了解决这个问题，我们首先需要将物料的加工状态作为环境状态加入，因为在问题一中我们不需要追踪具体物料的活动流程即能满足算法的构建，而此时，我们需要有追踪物料加工状态的数据才能实现问题二的解答。

然后，我们将数据表中提供的参数输入到仿真环境中，并根据仿真环境求解三组数据对应不同状况的动态调度模型。

最后，我们基于仿真环境建立测试环境，这其中的改动包括增加我们需要的物料加工状况的输出，调整动态调度模型实际执行时的一些策略。然后，我们让 RGV 在三组环境参数对应的测试环境中各运行一个班次，从而得到物料的加工状态。

5.3.2 模型的建立与求解

根据模型分析，我们首先需要将物料追踪状况引入仿真环境。对此，我们考虑对同一个班次中出现的所有物料从 1 开始进行编号，设上料传送带传输本班次的第 n 个物料 $object_n$ ，该物料具有如下状态（全表随附件“物料状态表”）：

| 物料状态表 | |
|--------------|-------------------------------------------------------------|
| 状态 | 说明 |
| 物料编号 | 在一个班次中该物料产生的序号 |
| 上料时间（故障开始时间） | 在一个班次中物料从上料传送带传过来的时刻（若上料时 CNC 发生故障则修改为 CNC 开始发生故障的时刻） |
| 下料时间（故障结束时间） | 在一个班次中物料被 RGV 放至下料传送带的时刻（若之前上料时 CNC 发生故障则修改为 CNC 开始发生故障的时刻） |
| 故障对应 CNC | 若上料时 CNC 发生故障，此处便为对应 CNC 的编号，否则为 0 |

此外，我们在 RGV 和 CNC 状态中也各自增加了“物料编号”状态，以便在任意时刻追踪物料在环境中的位置。

然后我们创建一个加工完成队列和一个故障队列，用于记录一个班次中加工完成的物料信息和发生故障的物料信息。

接着，**对于情况一：**

我们使用问题一中针对情况一的目标函数求解方法求解情况一的动态调度模型。然后将题述三组环境参数输入测试环境并应用动态调度模型，得到情况一的结果 Case1.

对于情况二：

我们对于三组环境参数分别列出所有适用于该参数的刀片分布，然后对每一个刀片分布应用问题一中对于情况二所采用的 Q-learning 算法，求得三组参数对应的最优刀片分布及其动态调度模型。然后将三组参数对应的刀片分布与其动态调度模型输入测试环境，得到情况二的结果 Case2

对于情况三：

我们在仿真环境中引入随机误差，然后对于一道工序的状况我们应用情况二中所应用的 Q-learning 算法求解动态调度模型，对于二道工序的状况我们列出所有适用的刀片分布后再对应每个刀片分布用 Q-learning 算法求解动态调度模型，并根据问题一中情况三的动态调度模型构建方法调整动态调度模型以适应动态环境。最后，我们将参数对应的刀片分布和动态调度模型输入测试环境，得到情况三的结果 Case3.

（模型验证结果 Case1，Case2，Case3 已放入支撑材料中）

此外，我们还得到三组参数在三种情况下动态调度模型的作业效率（全表随附

件“作业效率表”):

| 作业效率 | | |
|------|------------|--------------|
| 参数组 | 情况 | 一班次内 加工物料 |
| 第一组 | 一道工序 | 404 |
| | 两道工序 | 236 |
| | 有故障可能的一道工序 | 343 |
| | 有故障可能的两道工序 | 187 |
| 第二组 | 一道工序 | 360 |
| | 两道工序 | 156 |
| | 有故障可能的一道工序 | 293 |
| | 有故障可能的两道工序 | 166 |
| 第三组 | 一道工序 | 416 |
| | 两道工序 | 205 |
| | 有故障可能的一道工序 | 343 |
| | 有故障可能的两道工序 | 203 |

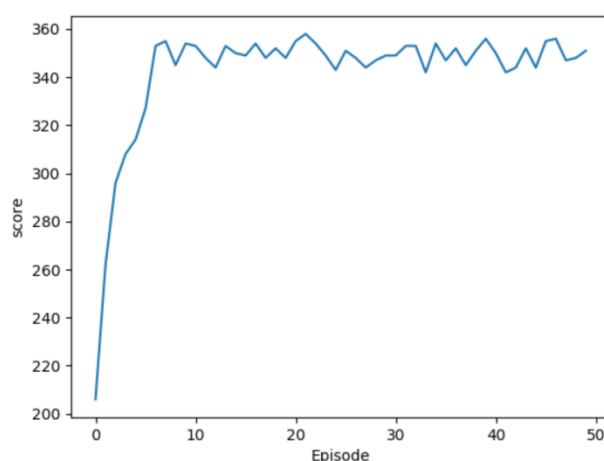
5.4 模型的检验与结果分析

我们首先搭建了一个智能加工系统的仿真环境。然后,在求解情况一的调度模型时,我们根据我们推导出的目标函数在三种系统环境参数设置下进行求解,均得出启用所用 CNC 的为最佳策略。经分析,这是因为三组系统环境参数中,单个 CNC 加工的时间比比 RGV 循环时间还要长,因此只有启用所有的 CNC 才可以提高 CNC 的负载。

通过检验模拟系统多个循环的作业流水记录,我们发现系统在进入循环作业阶段后,所有 CNC 均处于满负载状态,此时系统的生产速度达到最高,因此在该情况下,该调度模型的调度十分高效。

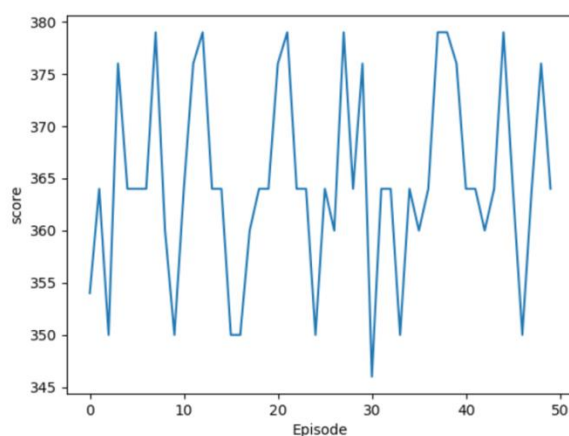
而对于情况二和情况三,我们分别构建了 Q-learning 算法求解的静态环境下的动态调度模型和动态环境下的动态调度模型。为了观察算法的效果与收敛水平,我们将题附智能加工系统作业参数数据表的第一组参数带入求解算法中,并绘制出 RGV 在 50 个回合 Episode 中每一回合加工的物料数量 score 的折线图。

下图为 RGV 在第一种情况的加工环境中学习的 score-Episode 折线图。



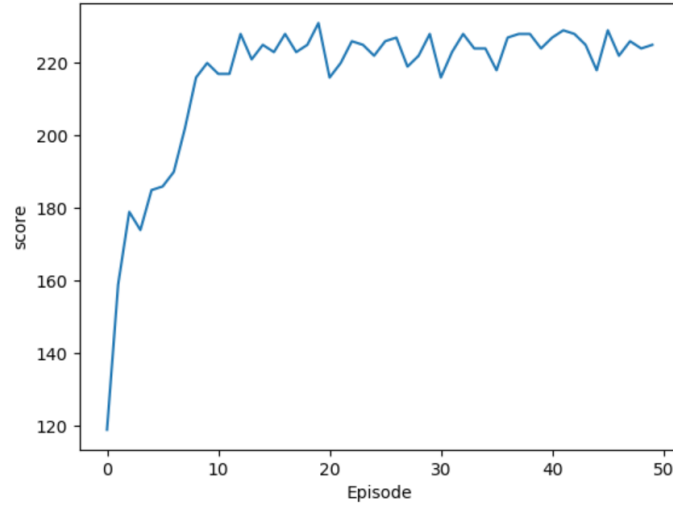
我们看到在几个回合的迭代后算法便已收敛，收敛迅速源于第一种情况环境比较简单，求解算法没有持续收敛至最佳是因为这是一个静态环境，而我们设置了参数 ϵ 让 RGV 在学习过程能够大胆的尝试不同的做法，因此 RGV 总是会因为尝试一些并不那么好的动作使得 score 降低。而在测试环境中，我们取消该参数让 RGV 保守的只选择最优的动作执行：

下图为 RGV 在第一种情况的加工环境中测试的 score-Episode 折线图。



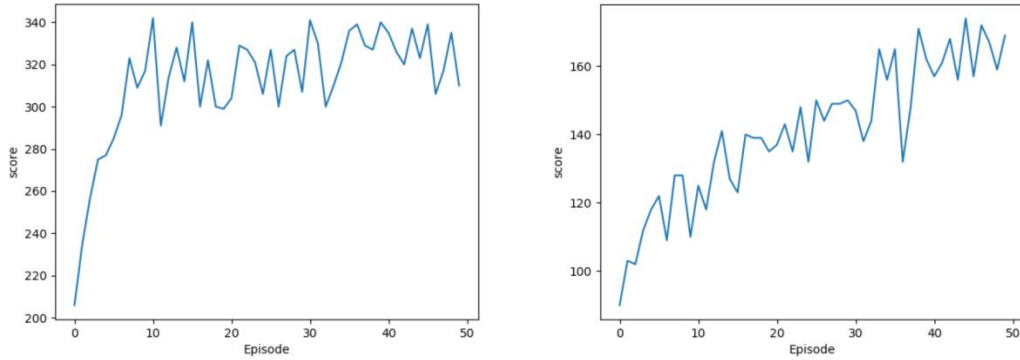
得分在区间[350,380]波动，因为 RGV 在选择动作时尽管会选择价值最大的动作，但在同一观察状态下价值最大的动作可能不止一个。

接着我们假设所有的物料需要两道工序，并以第一组参数作为环境输入，随机选择一组刀片排序作为 Arrangement，我们得到在该 Arrangement 下 RGV 的 score-Episode 折线图：



环境变得复杂后算法的收敛需要更多的 Episode，并且由于物料需要 2 道加工，能够得到的 score 也随之下降。

而对于情况三，我们得到一道工序和两道工序的 Episode-score 折线图：



左为一道工序，右为二道工序，可以发现引入故障机制的时候动态调度模型的结果会变得更加不稳定，收敛后的相邻回合间的 score 相差最大为 50，而前面两种情况相邻回合的 score 差值最大不超过 20，并且在有两道工序的情况下引入故障机制会使得算法需要更多的回合去收敛。

基于我们在三种情况下应用求解算法 Q-learning 的结果，我们可以得知我们的求解算法在题述所有情况下都能够根据更新公式：

$$Q_{target} = r + \gamma * \max_{action_{t+1}} \{Q(o_{new}, action_{t+1})\}$$

求解优化问题：

$$\operatorname{argmax}_Q \text{ score} = A(t, s; Q(o, a))$$

然后我们定义c为 RGV 的平均物料损耗量，即在一个班次中 RGV 处理每个物料时 RGV

的等待加移动所消耗的时间，对于该变量，我们有：

$$c \approx \frac{28800}{score} - WASH - \frac{EXCHANGE1 + EXCHANGE2}{2}$$

其中score表示该班次中 RGV 加工的物料数，WASH 表示 RGV 清洗作业消耗的时间，EXCHANGE1 表示 RGV 为奇数编号 CNC 上下料的时间，EXCHANGE2 表示 RGV 为偶数编号 CNC 上下料的时间。可以发现，在同样的环境下，c 与 RGV 的作业效率呈负相关，c 越接近 0，RGV 的作业效率越高。

然后我们计算得到测试环境输入三组参数与三种情况下的c列表：

| RGV工作效率 | | |
|---------|------------|------------|
| 参数组 | 情况 | RGV平均物料损耗率 |
| 第一组 | 一道工序 | 16.78713 |
| | 两道工序 | 67.5339 |
| | 有故障可能的一道工序 | 29.46501 |
| | 有故障可能的两道工序 | 99.5107 |
| 第二组 | 一道工序 | 17.5 |
| | 两道工序 | 122.1154 |
| | 有故障可能的一道工序 | 35.79352 |
| | 有故障可能的两道工序 | 110.994 |
| 第三组 | 一道工序 | 14.73077 |
| | 两道工序 | 85.9878 |
| | 有故障可能的一道工序 | 29.46501 |
| | 有故障可能的两道工序 | 87.37192 |

根据智能系统加工流程，我们知道在有故障的一道工序的情况下，RGV 的损耗率会略微增大但不会相差无故障情况太大。根据上表我们得出三组参数之间用静态模型的一道工序与应用 Q-learning 算法的有故障机制的一道工序 RGV 平均物料损耗率之间的差值为 12.67788586，18.29351536，14.73424535，可以发现应用 Q-learning 算法的动态调度模型与直接求解最优化静态调度模型之间差距很小，而 Q-learning 算法相较于求解静态调度模型拥有更广泛的适用性与应变能力，说明 Q-learning 模型具有较强的鲁棒能力。

六、 模型评价

6.1 模型的评价

6.1.1 模型的优点：

1. 我们用 Python 编程语言建立了能够模拟三种情况的环境仿真模型，使得求

解的动态调度模型可以即时的在虚拟环境中测试，并且也可以方便地构建基于环境的动态调度模型求解算法。

2. 我们引入了流行的强化学习中的动态模型求解算法 Q-learning 并对其作出了适应问题的改进，使得智能加工系统能够在不同环境不同状况下都拥有较强的应变能力，算法也拥有更强的鲁棒能力。

3. 针对不同情况我们应用了不同的调度模型和求解算法，对情况一的简单静态环境我们用静态调度模型搭配静态求解算法求解，对情况二的复杂静态环境，我们用静态调度模型搭配动态求解算法 Q-learning 求解，对情况三的动态环境，我们用引入及时学习机制的动态调度模型搭配动态求解算法求解。

4. 我们可视化了仿真环境的状态、流程与动态效果，有助于获取仿真环境的信息与搭建动态调度模型。

5. 我们使用 Python 作为主要的编程语言，Python 的高级语言特性使得构建仿真环境时我们能够更加地专注于系统的运行机制而不是复杂的底层细节。

6.1.2.模型的缺点及改进：

1. 模型在求解两道工序的最优刀片分布时密集的计算对机器的处理性能要求较高，计算时间较长。

2. 在求解最佳刀片分布时，收集更多的环境参数信息在不同刀片分布下的作业效率可以通过线性规划更快速地求解最优刀片分布。

3. 模型中供人工调整的超参数可以根据智能系统的需求做调整使得动态调度模型更加符合期望，如减小奖励衰减率来使 RGV 执行动作决策时更加的保守从而减少 RGV 的功耗。

4. 可以根据系统的细节设计更加合适的价值奖励机制来提升算法效果。

6.2 模型的推广

1. 本文采用的仿真环境模型建立方案，由动作与状态互相作用的仿真环境设计方法能够用于模拟多种有限状态的复杂环境。

2. 本文构建动态调度模型求解算法时采用的价值函数机制通过引入智能体的奖惩来驱使智能体决策，该方法通过奖惩参数的设计，能够让智能体针对不同的复杂环境寻找最优的决策。

3. 本文采用的 Q-learning 算法能够用于各种环境下智能体的动作规划，包括

机器智能作业，游戏 AI 设计，自动驾驶等，当今强化学习在各领域应用的大部分主流算法都是该算法结合深度学习算法的改进方法。

参考文献

- [1] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction[M]. USA: MIT, 2017.
- [2] Richard Bellman. A Markovian Decision Process[D]. India Univ., 1957
- [3] 孙晟. 基于强化学习的动态单机调度研究[D]. 上海交通大学, 2007.
- [4] T Behrens. An Interface for Agent-Environment Interaction[D]. Springer Berlin Heidelberg, 2010

附录

附录一：RGV 动作对照表

| 动作编号 | 消耗时间 | 移动步数 | 动作类型 | 操作的CNC编号 | 备注 |
|------|-----------|------|---------------|----------|-----------------------------------------------------------------------------------------------|
| 0 | 1 | 0 | 等待 | 0 | 操作的CNC编号在代码中为-1至7便于编程操作，此处标识为0-8更易于理解 |
| 1 | MOVE1 | 1 | 向右1步 | 0 | |
| 2 | MOVE1 | -1 | 向左1步 | 0 | |
| 3 | MOVE2 | 2 | 向右2步 | 0 | |
| 4 | MOVE2 | -2 | 向左2步 | 0 | |
| 5 | MOVE3 | 3 | 向右3步 | 0 | |
| 6 | MOVE3 | -3 | 向左3步 | 0 | |
| 7 | EXCHANGE1 | 0 | 给1号CNC上料 | 1 | |
| 8 | EXCHANGE1 | 0 | 给1号CNC下料 | 1 | |
| 9 | EXCHANGE1 | 0 | 给1号CNC下料之后再上料 | 1 | |
| 10 | EXCHANGE2 | 0 | 给2号CNC上料 | 2 | |
| 11 | EXCHANGE2 | 0 | 给2号CNC下料 | 2 | |
| 12 | EXCHANGE2 | 0 | 给2号CNC下料之后再上料 | 2 | |
| 13 | EXCHANGE1 | 0 | 给3号CNC上料 | 3 | |
| 14 | EXCHANGE1 | 0 | 给3号CNC下料 | 3 | |
| 15 | EXCHANGE1 | 0 | 给3号CNC下料之后再上料 | 3 | |
| 16 | EXCHANGE2 | 0 | 给4号CNC上料 | 4 | |
| 17 | EXCHANGE2 | 0 | 给4号CNC下料 | 4 | |
| 18 | EXCHANGE2 | 0 | 给4号CNC下料之后再上料 | 4 | |
| 19 | EXCHANGE1 | 0 | 给5号CNC上料 | 5 | |
| 20 | EXCHANGE1 | 0 | 给5号CNC下料 | 5 | |
| 21 | EXCHANGE1 | 0 | 给5号CNC下料之后再上料 | 5 | |
| 22 | EXCHANGE2 | 0 | 给6号CNC上料 | 6 | |
| 23 | EXCHANGE2 | 0 | 给6号CNC下料 | 6 | |
| 24 | EXCHANGE2 | 0 | 给6号CNC下料之后再上料 | 6 | |
| 25 | EXCHANGE1 | 0 | 给7号CNC上料 | 7 | |
| 26 | EXCHANGE1 | 0 | 给7号CNC下料 | 7 | |
| 27 | EXCHANGE1 | 0 | 给7号CNC下料之后再上料 | 7 | |
| 28 | EXCHANGE2 | 0 | 给8号CNC上料 | 8 | |
| 29 | EXCHANGE2 | 0 | 给8号CNC下料 | 8 | |
| 30 | EXCHANGE2 | 0 | 给8号CNC下料之后再上料 | 8 | |
| 31 | 0 | 0 | 清洗物料 | 0 | 根据智能加工系统流程，清洗时间极短，远小于放置物料到下料传送带的时间，因此我们将清洗时间也合并至放置物料时间中，并且对于两道工序中经过第一道工序的物料只有极短的清洗时间，因此可以忽略不计 |
| 32 | PUT | 0 | 放置物料到下料传送带 | 0 | |

附录二：RGV 和 CNC 状态表

| RGV状态表 | | | | | | | | | | |
|-----------|------|------|------|-----|------------------|--------------------|-------------------|------------------------|-----------|----------------|
| 状态 \ 编号 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 移动步数 | 向左3步 | 向左2步 | 向左1步 | 不移动 | 向右1步 | 向右2步 | 向右3步 | | | |
| 当前位置 | | | | | 在第1和第2台CNC之间 | 在第3和第4台CNC之间 | 在第5和第6台CNC之间 | 在第7和第8台CNC之间 | | |
| 正在执行的动作类型 | | | | 等待 | 移动 | 上料 | 下料 | 上下料 | 清洗作业 | 放置下料 |
| 携带的物料类型 | | | | 无 | 只需要一道加工并已被加工后的熟料 | 只需要一道加工并已被加工清洗后的成料 | 需要两道工序并已经过一道加工的熟料 | 需要两道工序并已经过一道加工并被清洗过的熟料 | 二道加工完成的熟料 | 二道加工完成并已清洗过的成料 |

| CNC状态表 | | | | | | | | |
|---------|--------|-----------|------------|--------------------|----|-----------|---|------------------------|
| 状态 \ 编号 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 备注 |
| 工作状态 | 空闲 | 一道工序的加工 | 二道工序的第一道加工 | 二道工序的第二道加工 | 故障 | | | |
| 工作结束时间 | | | | | | | | 无编号，内容一个对应于总时间的结束时间的数值 |
| 台上物料 | 无 | 一道加工熟料 | | 需要第二道加工并已经过一道加工的熟料 | | 二道加工完成的熟料 | | |
| 工作时间 | | | | | | | | 无编号，内容为该CNC工作需要的时间数值 |
| 工作类型 | 一道工序加工 | 二道工序第一道加工 | 二道工序第二道加工 | | | | | |

附录三：环境参数对照表

| 参数 | 符号 | 说明 |
|----|--------|-----------------------------------------------|
| 时间 | time | 班次剩余的时间，初始值为28800（8个小时），在仿真环境中该数值将随RGV的行动逐步减少 |
| 奖励 | reward | 调度智能获得的奖励，用于引导RGV采取更优的调度策略 |
| 物料 | score | 初始值为0，每完成一个物料的加工便会加1，用于判断动态调度模型的完成效果 |

附录四：物料状态表

| 物料状态表 | |
|--------------|-------------------------------------------------------|
| 状态 | 说明 |
| 物料编号 | 在一个班次中该物料产生的序号 |
| 上料时间（故障开始时间） | 在一个班次中物料从上料传送带传过来的时刻（若上料时CNC发生故障则修改为CNC开始发生故障的时刻） |
| 下料时间（故障结束时间） | 在一个班次中物料被RGV放至下料传送带的时刻（若之前上料时CNC发生故障则修改为CNC开始发生故障的时刻） |
| 故障对应CNC | 若上料时CNC发生故障，此处便为对应CNC的编号，否则为0 |

附录五：作业效率表

| 作业效率 | | |
|------|------------|----------|
| 参数组 | 情况 | 一班次内加工物料 |
| 第一组 | 一道工序 | 404 |
| | 两道工序 | 236 |
| | 有故障可能的一道工序 | 343 |
| | 有故障可能的两道工序 | 187 |
| 第二组 | 一道工序 | 360 |
| | 两道工序 | 156 |
| | 有故障可能的一道工序 | 293 |
| | 有故障可能的两道工序 | 166 |
| 第三组 | 一道工序 | 416 |
| | 两道工序 | 205 |
| | 有故障可能的一道工序 | 343 |
| | 有故障可能的两道工序 | 203 |

| RGV工作效率 | | |
|---------|------------|------------|
| 参数组 | 情况 | RGV平均物料损耗率 |
| 第一组 | 一道工序 | 16.78713 |
| | 两道工序 | 67.5339 |
| | 有故障可能的一道工序 | 29.46501 |
| | 有故障可能的两道工序 | 99.5107 |
| 第二组 | 一道工序 | 17.5 |
| | 两道工序 | 122.1154 |
| | 有故障可能的一道工序 | 35.79352 |
| | 有故障可能的两道工序 | 110.994 |
| 第三组 | 一道工序 | 14.73077 |
| | 两道工序 | 85.9878 |
| | 有故障可能的一道工序 | 29.46501 |
| | 有故障可能的两道工序 | 87.37192 |

附录六：Python 源程序

1. RGV 学习主程序 train.py

```
from enviroment import Env
from rl import QLearning
import pandas as pd
```

```
# 设置全局变量
```

```
MAX_EPISoDES = 20
```

```
def update():
```

```
    score_record=[]
```

```
    for episode in range(MAX_EPISoDES):
```

```
        # 初始化 state 的观测值
```

```
        o = env.reset()
```

```
    while True:
```

```
        # 更新可视化环境
```

```
        env.render(o)
```

```
        # RL 大脑根据 state 的观测值挑选 a
```

```
        a = RL.choose_a_QL(o)
```

```
        # 探索者在环境中实施这个 a, 并得到环境返回的下一个 state 观测值,
reward 和 done (是否是掉下地狱或者升上天堂)
```

```
        o_, reward, done = env.step(a)
```

```
        # RL 从这个序列 (state, a, reward, state_) 中学习
```

```
        RL.learn(o, a, reward, o_)
```

```

        # 将下一个 state 的值传到下一次循环，并更新环境参数
        if o_ != 'terminal':
            o = o_
        env.update(o)

        print("获得 reward:",str(reward),"采用动作:",a,"RGV 位置",o[1]['loc',"剩余时间:",env.time)
        print("RGV 携带工料:",o[1]['material_type'])
        print("RGV 状态",str(o[1]),"CNC 状态",str(o[2]))

        print("加工物料:",str(env.score))

        # 回合结束
        if done:
            print("本回合加工物料:",str(env.score))
            score_record.append(env.score)

        pd.DataFrame({'score':score_record}).to_csv('result/record/param3case3_1_record.csv',index=False)

        break

    # 结束并关闭窗口
    print('over')
    env.destroy(RL,path='result/q_result_p3c3_1.csv')

if __name__ == "__main__":
    # 定义环境 env 和 RL 方式
    env = Env()
    RL = QLearning(as=list(range(env.n_as)))

    # 开始可视化环境 env
    update()

```

2. RGV 测试主程序 predict.py

```

from enviroment import Env
from rl import QLearning
import pandas as pd

# 设置全局变量
MAX_EPISoDES = 1

def material_list_output(work_list,error_list,path1,path2):
    num=[]
    up_time1=[]

```

```

down_time1=[]
cnc1=[]
cnc2=[]
up_time2=[]
down_time2=[]
for material in work_list:
    num.append(material.num)
    up_time1.append(28800-material.up_time1)
    down_time1.append(28800-material.down_time1)
    cnc1.append(material.cnc1)
    up_time2.append(28800-material.up_time2)
    down_time2.append(28800-material.down_time2)
    cnc2.append(material.cnc2)
work=pd.DataFrame({'num':num,
                   'up_time1':up_time1,
                   'up_time2':up_time2,
                   'down_time1':down_time1,
                   'down_time2':down_time2,
                   'cnc1':cnc1,
                   'cnc2':cnc2
                   })

num=[]
error_start=[]
error_end=[]
error_cnc=[]
for material in error_list:
    num.append(material.num)
    error_start.append(28800-material.error_start)
    error_end.append(28800-material.error_end)
    error_cnc.append(material.error_cnc)
error=pd.DataFrame({
    'num':num,
    'error_start':error_start,
    'error_end':error_end,
    'error_cnc':error_cnc
})
work.to_csv(path1,index=False)
error.to_csv(path2,index=False)

```

```

def update():
    score_record=[]
    for episode in range(MAX_EPISODES):
        # 初始化 state 的观测值
        o = env.reset()

```

```

while True:
    # 更新可视化环境
    env.render(o)

    # RL 大脑根据 state 的观测值挑选 a
    a = RL.choose_a_static(o)

    # 探索者在环境中实施这个 a, 并得到环境返回的下一个 state 观测值,
    # reward 和 done (是否是掉下地狱或者升上天堂)
    o_, reward, done = env.step(a)

    # RL 从这个序列 (state, a, reward, state_) 中学习
    RL.learn(o, a, reward, o_)

    # 将下一个 state 的值传到下一次循环, 并更新环境参数
    if o_ != 'terminal':
        o = o_
        env.update(o)

    print("获得 reward:",str(reward),"采用动作:",a,"RGV 位置",o[1]['loc',"剩余时
间:",env.time)
    print("RGV 携带工料:",o[1]['material_type'])
    print("RGV 状态",str(o[1]),"CNC 状态",str(o[2]))

    print("加工物料:",str(env.score))

    # 回合结束
    if done:
        print("本回合加工物料:",str(env.score))
        score_record.append(env.score)

    pd.DataFrame({'score':score_record}).to_csv('result/record/param3case3_1_predict_record.c
sv',index=False)

    material_list_output(env.work_list,env.error_list,
                        path1='result/result/p3c3_1_work_result.csv',
                        path2='result/result/p3c3_1_error_result.csv')

    break

# 结束并关闭窗口
print('over')
env.destroy(RL,path=r'result\q_result_p3c3_1_predict.csv')

if __name__ == "__main__":

```

```

# 定义环境 env 和 RL 方式
env = Env()
RL = QLearning(as=list(range(env.n_as)),path='result/q_result_p3c3_1.csv')

# 开始可视化环境 env
update()

3. 仿真环境 environment.py
import tkinter as tk
import numpy as np
import time
import sys
import pygame

ToTAL_TIME = 28800 # 8 个小时时间
PUT = 25 # RGV 清洗放置作业时间
MoVE1 = 18 # RGV 移动一个单位的时间
MoVE2 = 32 # RGV 移动两个单位的时间
MoVE3 = 46 # RGV 移动三个单位的时间
CNC_1 = 545 # CNC 加工完成一道工序时间
CNC_2_1 = 455 # CNC 加工完成两道工序中第一道工序时间
CNC_2_2 = 182 # CNC 加工完成两道工序中第二道工序时间
EXCHANGE1 = 27 # RGV 为奇数编号 CNC 完成一次上下料时间
EXCHANGE2 = 32 # RGV 为偶数编号 CNC 完成一次上下料时间

CNC_STATUS = [0, 0, 0, 0, 0, 0, 0, 0] # CNC 初始工作状态
CNC_LoC = [1, 2, 3, 4, 5, 6, 7, 8] # CNC 位置标识序列
CNC_WoRK_TYPE = [0,0,0,0,0,0,0,0] # CNC 工种 0,1,2 对应一道工序，二道工序第一道，
二道工序第二道
CNC_WoRK_TIME_TYPE=[CNC_1,CNC_2_1,CNC_2_2]

RGV_INITIAL_LoC=1

RGV_HEIGHT=50
RGV_WIDTH=100
CNC_HEIGHT=100
CNC_WIDTH=100

ERRoR_RATE=0.01
FIX_TIME=600

class RGVa(object):
    def __init__(self,time,move,a_type,cnc_num):
        """

```


time rgv 完成这个动作所消耗的时间
 move rgv 的行动步数 -3 - +3
 a_type 对应 0, 1, 2, 3, 4, 5, 6 分别表示 等待, 移动, 上料, 下料, 上下料, 清洗作业, 放置下料

```
'''
self.consume_time=time
self.move=move
self.a_type=a_type
self.cnc_num=cnc_num
```

class Env():

```
viewer = None # viewer 默认为空
score=0
time=ToTAL_TIME #总共的时间, 在回合结束前不断消耗
material_num=1 #工料编号
work_list=[] #加工成功的工料列表
error_list=[] #故障的工料列表
material_dict={} #工料表
a_space = [
    RGVa(1, 0, 0, -1),
    RGVa(MoVE1, 1, 1, -1),
    RGVa(MoVE1, -1, 1, -1),
    RGVa(MoVE2, 2, 1, -1),
    RGVa(MoVE2, -2, 1, -1),
    RGVa(MoVE3, 3, 1, -1),
    RGVa(MoVE3, -3, 1, -1), # 6
    RGVa(EXCHANGE1, 0, 2, 0),
    RGVa(EXCHANGE1, 0, 3, 0),
    RGVa(EXCHANGE1, 0, 4, 0),
    RGVa(EXCHANGE2, 0, 2, 1),
    RGVa(EXCHANGE2, 0, 3, 1),
    RGVa(EXCHANGE2, 0, 4, 1),
    RGVa(EXCHANGE1, 0, 2, 2),
    RGVa(EXCHANGE1, 0, 3, 2),
    RGVa(EXCHANGE1, 0, 4, 2),
    RGVa(EXCHANGE2, 0, 2, 3),
    RGVa(EXCHANGE2, 0, 3, 3),
    RGVa(EXCHANGE2, 0, 4, 3),
    RGVa(EXCHANGE1, 0, 2, 4),
    RGVa(EXCHANGE1, 0, 3, 4),
    RGVa(EXCHANGE1, 0, 4, 4),
    RGVa(EXCHANGE2, 0, 2, 5),
    RGVa(EXCHANGE2, 0, 3, 5),
```

```

    RGVa(EXCHANGE2, 0, 4, 5),
    RGVa(EXCHANGE1, 0, 2, 6),
    RGVa(EXCHANGE1, 0, 3, 6),
    RGVa(EXCHANGE1, 0, 4, 6),
    RGVa(EXCHANGE2, 0, 2, 7),
    RGVa(EXCHANGE2, 0, 3, 7),
    RGVa(EXCHANGE2, 0, 4, 7),
    RGVa(0, 0, 5, -1),
    RGVa(PUT, 0, 6, -1)
]

def __init__(self):
    self.rgv_info=RGVenv().rgv_info
    self.cnc_info=CNCenv().cnc_info
    self.n_as=len(self.a_space)

def update(self,o):
    self.time=o[0]
    self.rgv_info=o[1]
    self.cnc_info=o[2]
    self.material_num=o[3]
    self.work_list=o[4]
    self.error_list=o[5]
    self.material_dict = o[6]

def step(self, rgv_a):
    FIX_TIME=np.random.randint(600,1200)#随机产生一个修理时间

    rgv_a=self.a_space[rgv_a]

    done=False
    r=0

    rgv_info=self.rgv_info.copy()
    time=self.time
    cnc_info=self.cnc_info.copy()
    material_num=self.material_num
    work_list=self.work_list.copy()
    error_list=self.error_list.copy()
    material_dict=self.material_dict.copy()

    consume=rgv_a.consume_time #rgv 完成这个动作所消耗的时间
    move=rgv_a.move #rgv 的行动步数 -3 - +3
    work=rgv_a.a_type # a_type 对应 0, 1, 2, 3, 4, 5, 6 分别表示 等待, 移动, 上

```

料，下料，上下料，清洗作业,放置下料

```
rgv_info['loc']=rgv_info['loc']+move
rgv_info['move']=move

time=time-consume
r-=consume*0.01

for cnc in range(8):
    if cnc_info['status'][cnc]==0:
        r-=1
    if cnc_info['over_time'][cnc]>=time:
        cnc_info['status'][cnc]=0

if work==6:
    material = material_dict[int(rgv_info['material_num'])]
    if rgv_info['material_type']==2:
        rgv_info['material_type']=0
    elif rgv_info['material_type']==6:
        rgv_info['material_type']=0
    self.score+=1
    print(material_dict)
    work_list.append(material) #将工料添加进完成列表
    material_dict.pop(int(rgv_info['material_num'])) #将工料从工料表中删除
    rgv_info['material_num']=0 #清除 RGV 上的工料
    r=10
elif work==2: # 上料
    rgv_info['material_type'] = 0
    cnc=rgv_a.cnc_num
    print('cnc 编号:',cnc+1)

    if cnc_info['work_type'][cnc]==0:# 如果是需要一道工序的生料
        cnc_info['status'][cnc]=1
        cnc_info['material_type'][cnc]=1
        # 增加工料
        material=Material(material_num,time+consume,0)
        material.cnc1=cnc+1
        material_dict[material.num]=material
        cnc_info['material_num'][cnc] = material.num
        material_num+=1
    elif cnc_info['work_type'][cnc]==1:# 如果是需要两道工序的生料
        cnc_info['status'][cnc] = 2
        cnc_info['material_type'][cnc] = 3
```

```

# 增加工料
material = Material(material_num, time + consume, 1)
material.cnc1 = cnc + 1
material_dict[material.num] = material
cnc_info['material_num'][cnc] = material.num
material_num += 1

elif cnc_info['work_type'][cnc]==2: # 如果是需要两道工序的熟料
    cnc_info['status'][cnc] = 3
    cnc_info['material_type'][cnc] = 5
    #转移工料
    material_dict[int(rgv_info['material_num'])].up_time2=time+consume
    material_dict[int(rgv_info['material_num'])].cnc2 = cnc + 1
    cnc_info['material_num'][cnc]=int(rgv_info['material_num'])
    rgv_info['material_num']=0

if np.random.uniform() < ERRoR_RATE: # 如果发生了故障
    cnc_info['material_type'][cnc] = 0
    cnc_info['status'][cnc] = 4
    #记录故障工料
    material=material_dict[int(cnc_info['material_num'][cnc])]
    material.error_cnc=cnc+1

if cnc_info['status'][cnc]!=4:
    cnc_info['over_time'][cnc]=time-cnc_info['work_time'][cnc] #标识 cnc 工作
    完成的时间点
else:

    cnc_info['over_time'][cnc]=time-FIX_TIME-np.random.randint(0,cnc_info['work_time'][cnc
    ]) #标识故障修理完的时间点
    material = material_dict[int(cnc_info['material_num'][cnc])]

    material.error_end=time-FIX_TIME-np.random.randint(0,cnc_info['work_time'][cnc]) #在工
    料上记录故障修理完成时间点
    material.error_start=time+consume
    error_list.append(material) #将工料添加至故障工料队列
    material_dict.pop(int(cnc_info['material_num'][cnc]))
    r=1
elif work==4: #上下料
    cnc = rgv_a.cnc_num

    cache=int(cnc_info['material_num'][cnc]) #缓存工料

if cnc_info['work_type'][cnc] == 0: # 如果是需要一道工序的生料

```

```

cnc_info['status'][cnc] = 1
rgv_info['material_type'] = 1 #取回完成一道工序的熟料
cnc_info['material_type'][cnc] = 1
# 增加工料
material = Material(material_num, time + consume, 0)
material.cnc1 = cnc + 1
material_dict[material.num] = material
cnc_info['material_num'][cnc] = material.num
material_num += 1
material_dict[cache].down_time1 = time + consume

elif cnc_info['work_type'][cnc] == 1: # 如果是需要两道工序的生料
    cnc_info['status'][cnc] = 2
    rgv_info['material_type'] = 3 #取回完成一道工序需要第二道工序的熟料
    cnc_info['material_type'][cnc] = 3
    # 增加工料
    material = Material(material_num, time + consume, 1)
    material.cnc1 = cnc + 1
    material_dict[material.num] = material
    cnc_info['material_num'][cnc] = material.num
    material_num += 1
    material_dict[cache].down_time1 = time + consume

elif rgv_info['material_type'] == 4 and cnc_info['work_type'][cnc] == 2: # 如果
    是需要两道工序并且被清洗过后的熟料
    cnc_info['status'][cnc] = 3
    rgv_info['material_type'] = 5 #取回完成两道工序的熟料
    cnc_info['material_type'][cnc] = 5
    # 转移工料
    material_dict[int(rgv_info['material_num'])].up_time2 = time + consume
    material_dict[int(rgv_info['material_num'])].cnc2 = cnc + 1
    cnc_info['material_num'][cnc] = int(rgv_info['material_num'])
    material_dict[cache].down_time2 = time + consume

# 转移工料
rgv_info['material_num'] = cache

if np.random.uniform() < ERROR_RATE: # 如果发生了故障
    cnc_info['material_type'][cnc] = 0
    cnc_info['status'][cnc] = 4
    # 记录故障工料
    material = material_dict[int(cnc_info['material_num'][cnc])]
    material.error_cnc = cnc + 1

```

```

        if cnc_info['status'][cnc] != 4:
            cnc_info['over_time'][cnc] = time - cnc_info['work_time'][cnc] # 标识 cnc
工作完成的时间点
        else:
            cnc_info['over_time'][cnc] = time -
FIX_TIME-np.random.randint(0,cnc_info['work_time'][cnc]) # 标识故障修理完的时间点
            material = material_dict[int(cnc_info['material_num'][cnc])]
            material.error_end = time - FIX_TIME
-np.random.randint(0,cnc_info['work_time'][cnc]) # 在工料上记录故障修理完成时间点
            material.error_start = time + consume
            error_list.append(material) # 将工料添加至故障工料队列
            material_dict.pop(int(cnc_info['material_num'][cnc]))

        r=3
    elif work==5:
        if rgv_info['material_type']==1:
            rgv_info['material_type']=2
        elif rgv_info['material_type']==3:
            rgv_info['material_type']=4
        elif rgv_info['material_type']==5:
            rgv_info['material_type']=6
        r=5

    if time<=0:
        done=True
        s_='terminal'
    else:
        s_=[time,rgv_info,cnc_info,material_num,work_list,error_list,material_dict]

    return s_,r,done

```

```

def reset(self):
    self.rgv_info['move'] = 0
    self.rgv_info['loc'] = RGV_INITIAL_LoC
    self.rgv_info['material_type'] = 0
    # 携带的物料编号，0 表示没有携带
    self.rgv_info['material_num'] = 0
    self.time=ToTAL_TIME
    self.score=0
    self.cnc_info['status'] = 0
    self.cnc_info['over_time'] = 0
    # material_type 取值 0, 1, 2, 3, 4, 5, 6 表示 无工料，熟料，成料，需要第二
道加工并已经过一道加工的熟料，需要第二道加工并已经过一道加工并且清洗过后的熟料，
二道加工完成的熟料，二道加工并且被清洗过后的成料

```

```

self.cnc_info['material_type'] = 0
# 携带的物料编号，0 表示没有携带
self.cnc_info['material_num'] = 0
self.material_num = 1
self.work_list = []
self.error_list = []
self.material_dict={}
for cnc, work_type in zip(range(8), CNC_WoRK_TYPE):
    self.cnc_info['work_time'][cnc] = CNC_WoRK_TIME_TYPE[work_type]
    self.cnc_info['work_type'][cnc] = work_type

return
[self.time,self.rgv_info,self.cnc_info,self.material_num,self.work_list,self.error_list]

def render(self,o):
    if self.viewer is None: # 若没有 viewer 则生成一个
        self.viewer = Viewer(rgv_info=self.rgv_info,cnc_info=self.cnc_info)
    self.viewer.render(o) # 使用 viewer 中的 render 功能

def sample_a(self):
    a=RGVa(10,1,1,-1)
    return a

def destroy(self,RL,path):
    RL.q_table.to_csv(path)

class RGVEnv(object):
    state_dim=4
    a_dim=4

    def __init__(self):
        self.rgv_info=np.zeros(
            1, dtype=[('move',np.int32),
                      ('loc',np.int32),
                      ('material_type',np.int32),
                      ('material_num',np.int32)]
        )
        # move 取值-3,-2,-1,0,1,2,3
        self.rgv_info['move']=0
        # loc 取值 1,2,3,4
        self.rgv_info['loc']=RGV_INITIAL_LoC
        # material_type 取值 0, 1, 2, 3, 4, 5, 6 表示 无工料，熟料，成料，需要第二
        # 道加工并已经过一道加工的熟料，需要第二道加工并已经过一道加工并且清洗过后的熟料，

```

二道加工完成的熟料，二道加工并且被清洗过后的成料

```
self.rgv_info['material_type'] = 0
#携带的物料编号，0 表示没有携带
self.rgv_info['material_num']=0
```

class CNCenv(object):

```
def __init__(self):
    self.cnc_info=np.zeros(
        8,dtype=[('status',np.int32),
                ('over_time',np.int32),
                ('work_time',np.int32),
                ('work_type',np.int32),
                ('material_type',np.int32),
                ('material_num',np.int32)]
    )
    #0,1,2,3,4 空闲，加工，加工 1，加工 2，故障
    self.cnc_info['status']=0 #所有 cnc 起初都处于空闲状态
    self.cnc_info['over_time']=0
    # material_type 取值 0，1，2，3，4，5，6 表示 无工料，熟料，成料，需要第二
    道加工并已经过一道加工的熟料，需要第二道加工并已经过一道加工并且清洗过后的熟料，
    二道加工完成的熟料，二道加工并且被清洗过后的成料
    self.cnc_info['material_type']=0
    # 携带的物料编号，0 表示没有携带
    self.cnc_info['material_num'] = 0
    for cnc,work_type in zip(range(8),CNC_WoRK_TYPE): #初始化所有 cnc 的工种
        self.cnc_info['work_time'][cnc]=CNC_WoRK_TIME_TYPE[work_type]
        self.cnc_info['work_type'][cnc] = work_type
```

class Material(object):

```
def __init__(self,num,up_time,type):
    self.num=num
    self.up_time1=up_time
    self.down_time1=0
    self.up_time2=0
    self.down_time2=0
    self.cnc1=0
    self.cnc2=0
    self.type=type
    self.error_cnc=0
    self.error_start=0
    self.error_end=0
```



```

class RGV(object):
    width = RGV_WIDTH
    height = RGV_HEIGHT

    def __init__(self, location, status):
        # location 包括 1, 2, 3, 4 标识 RGV 在通道上的第几个格子
        self.location = location
        # status 包括 0(等待),1(清洗)
        self.status = status
        # 携带的物料
        self.material_status=None

class CNC(object):
    width = CNC_WIDTH
    height = CNC_HEIGHT

    def __init__(self, location, status, work_type):
        # 1-8 对应 8 个 cnc 的位置
        self.location = location
        # status 包括 0（空闲），1（工作），2（故障）
        self.status = status
        # work_type 表示工种，0（一道工序类型），1（两道工序第一种刀片），2（两道工
序第二种刀片）
        self.work_type = work_type

class Viewer(pyglet.window.Window):
    def __init__(self,rgv_info,cnc_info):
        # 画出 RGV，CNC，轨道

        # 创建窗口的继承
        # vsync 如果是 True，按屏幕频率刷新，反之不按那个频率
        super(Viewer, self).__init__(width=400, height=250, resizable=False, caption='RGV',
vsync=False)

        # 窗口背景颜色
        pyglet.gl.glClearColor(1, 1, 1, 1)

        # 将作图信息放入 batch
        self.batch = pyglet.graphics.Batch() # display whole batch as once

        # 生成 RGV 参数表
        cnc = CNC(1, 0, 0)
        rgv = RGV(RGV_INITIAL.LoC, 0)

```

```

# 添加 RGV
self.rgv = self.batch.add(
    4, pyglet.gl.GL_QUADS, None, # 4 corners
    ('v2f', [(rgv.location - 1) * rgv.width, cnc.height, # x1, y1
              (rgv.location - 1) * rgv.width, cnc.height + rgv.height, # x2, y2
              rgv.location * rgv.width, cnc.height + rgv.height, # x3, y3
              rgv.location * rgv.width, cnc.height, # x4, y4
              ]),
    ('c3B', (86, 109, 249) * 4) # color
)

# 添加 CNC
self.cncs = []
for i, j, k in zip(CNC_LoC, CNC_STATUS, CNC_WoRK_TYPE):
    cnc = CNC(i, j, k)
    if cnc.location == 1 or cnc.location == 2:
        x = 0
    elif cnc.location == 3 or cnc.location == 4:
        x = 1
    elif cnc.location == 5 or cnc.location == 6:
        x = 2
    else:
        x = 3
    y = cnc.location % 2
    self.cncs.append(
        self.batch.add(
            4, pyglet.gl.GL_QUADS, None, # 4 corners
            ('v2f', [x * cnc.width, y * (rgv.height + cnc.height),
                     x * cnc.width, y * (rgv.height + cnc.height) + cnc.height,
                     x * cnc.width + cnc.width, y * (rgv.height + cnc.height) +
cnc.height,
                     x * cnc.width + cnc.width, y * (rgv.height + cnc.height)]),
            ('c3B', [249, 86 + 2 * cnc.work_type, 86 + 2 * cnc.work_type,
                    249, 86 + 2 * cnc.work_type, 86 + 2 * cnc.work_type,
                    150, 86 + 2 * cnc.work_type, 86 + 2 * cnc.work_type,
                    150, 86 + 2 * cnc.work_type, 86 + 2 * cnc.work_type]) #
color
        )
    )

# 添加 RGV 信息
self.rgv_info=rgv_info
self.cnc_info=cnc_info

```

```

def render(self,o):
    # 刷新并呈现在屏幕上
    self._update_RGV(o[1]) # 更新 RGV 内容
    self._update_CNC(o[2]) # 更新 CNC 内容 （暂时没有变化）
    self.switch_to()
    self.dispatch_events()
    self.dispatch_event('on_draw')
    self.flip()

def on_draw(self):
    # 刷新 RGV 的位置，状态，CNC 的状态
    self.clear() # 清屏
    self.batch.draw() # 画上 batch 里面的内容

def _update_RGV(self,rgv_info):
    # 更新 RGV 的位置，状态信息
    loc=rgv_info['loc']

    #RGV 当前坐标信息
    x1,y1=(loc-1)*RGV_WIDTH,CNC_HEIGHT
    x2,y2=(loc-1)*RGV_WIDTH,CNC_HEIGHT+RGV_HEIGHT
    x3,y3=loc*RGV_WIDTH,CNC_HEIGHT+RGV_HEIGHT
    x4,y4=loc*RGV_WIDTH,CNC_HEIGHT

    # RGV 移动
    self.rgv.vertices=([x1,y1,
                        x2,y2,
                        x3,y3,
                        x4,y4])

def _update_CNC(self,cnc_info):
    # 更新 CNC 的状态信息
    status=cnc_info['status']
    work_type=cnc_info['work_type']
    color_map={
        0:0,
        1:100,
        2:150,
        3:200,
        4:255
    }
    # CNC 状态改变
    for i in range(8):
        self.cncs[i].colors[0]=color_map[int(status[i])]

```

4. Q-learning 算法

import pandas as pd

import numpy as np

class QLearning(object):

def __init__(self,as,learning_rate=0.01,reward_decay=0.9,e_greedy=0.9,path=None):

self.as = as # a list

self.lr = learning_rate # 学习率

self.gamma = reward_decay # 奖励衰减

self.epsilon = e_greedy # 贪婪度

if path==None:

self.q_table = pd.DataFrame(columns=self.as, dtype=np.float64) # 初始 q_table

else:

self.q_table=pd.read_csv(path,index_col=0)

self.q_table.columns=self.as

def a_limit(self,s,as):

根据环境设置动作限制条件

as_=[_ for _ in as]

time=s[0]

rgv_info=s[1]

cnc_info=s[2]

cnc_num=list(range(2*int(rgv_info['loc'])-2,2*int(rgv_info['loc']))) # rgv 附近两台 cnc

的序号

cnc_1_a=[_ for _ in range(6+(cnc_num[0]+1)*3-2,6+(cnc_num[0]+1)*3+1)] #cnc 序号

对应的上下料动作

cnc_2_a = [_ for _ in range(6 + (cnc_num[1]+1) * 3 - 2, 6 + (cnc_num[1]+1) * 3 + 1)]

cnc_a=[cnc_1_a,cnc_2_a]

for i in range(7,31):#将 rgv 触碰不到的 cnc 动作都排除

if i not in cnc_1_a+cnc_2_a:

as_.remove(i)

as_.remove(31) #先将清洗动作去除

as_.remove(32) #先将放料动作去除

if rgv_info['loc']==1: #移动限制规则

as_.remove(2)

as_.remove(4)

as_.remove(6)

elif rgv_info['loc']==2:

as_.remove(5)

as_.remove(4)

```

        as_.remove(6)
    elif rgv_info['loc']==3:
        as_.remove(5)
        as_.remove(3)
        as_.remove(6)
    elif rgv_info['loc']==4:
        as_.remove(5)
        as_.remove(3)
        as_.remove(1)
    if rgv_info['material_type']==1 or rgv_info['material_type']==5 or
    rgv_info['material_type']==3: #身上有携带工料即刻清洗
        as_=[31]
    if rgv_info['material_type']==2 or rgv_info['material_type']==6: # 身上有可放置工料
    即刻放置
        as_=[32]

    if rgv_info['material_type']!=0: # rgv 携带工料不为空时
        for num in cnc_num:
            if cnc_info['work_type'][num]==0 or cnc_info['work_type'][num]==1: # 此时
            cnc 若为 0 或者 1 工种则不允许上料，下料，上下料
                try:
                    as_.remove(cnc_a[num % 2][0])
                    as_.remove(cnc_a[num % 2][1])
                    as_.remove(cnc_a[num % 2][2])
                except ValueError:
                    pass
    if rgv_info['material_type']!=4:
        for num in cnc_num:
            if cnc_info['work_type'][num] == 2 : # 此时 cnc 若为 2 工种则不允许上下
            料
                for a_num in cnc_a[num%2]:
                    try:
                        as_.remove(a_num)
                    except ValueError:
                        pass
    for num in cnc_num:
        # 先将下料动作去除
        try:
            as_.remove(cnc_a[num%2][1])
        except ValueError:
            pass
    if cnc_info['status'][num]!=0:#如果 cnc 不空闲，不允许上下料
        for a_num in cnc_a[num%2]:
            try:

```

料

```
        as_.remove(a_num)
    except ValueError:
        pass
    if cnc_info['material_type'][num]==0: #如果 cnc 上没有工料，不允许下料和上下
        try:
            as_.remove(cnc_a[num%2][2])
        except ValueError:
            pass
    elif cnc_info['material_type'][num]!=0: #如果 cnc 上有工料，不允许上料
        try:
            as_.remove(cnc_a[num%2][0])
        except ValueError:
            pass
    print(as_)

    return as_

def choose_a_QL(self,s):
    o=self.state_extract(s) # 从环境中提取观察状态
    self.check_state_exist(o) # 检测本 state 是否在 q_table 中存在
    as=self.a_limit(s,self.as)
    # 选择 a
    if np.random.uniform()<self.epsilon: #选择 Q value 最高的 a
        state_a=self.q_table.loc[str(o),as]

        # 同一个 state，可能会有多个相同的 Q a value，所以我们打乱顺序
        a = np.random.choice(state_a[state_a == np.max(state_a)].index)

    else:
        a=np.random.choice(as)
    return a

def choose_a_static(self,s):
    o = self.state_extract(s) # 从环境中提取观察状态
    self.check_state_exist(o) # 检测本 state 是否在 q_table 中存在
    as = self.a_limit(s, self.as)
    # 选择 a
    # 选择 Q value 最高的 a
    state_a = self.q_table.loc[str(o), as]
    # 同一个 state，可能会有多个相同的 Q a value，所以我们打乱顺序
    a = np.random.choice(state_a[state_a == np.max(state_a)].index)

    return a
```

```

def learn(self,s,a,r,s_):
    o = self.state_extract(s) # 从环境中提取观察状态
    o_ = self.state_extract(s_) # 从环境中提取观察状态
    self.check_state_exist(o_) #检测 q_table 中是否存在 s_
    q_predict = self.q_table.loc[str(o), a]
    if o_ != 'terminal':
        q_target = r + self.gamma * self.q_table.loc[str(o_), :].max() # 下个 state 不是
终止符
    else:
        q_target = r # 下个 state 是终止符
    self.q_table.loc[str(o), a] += self.lr * (q_target - q_predict) # 更新对应的 state-a 值

```

```

def check_state_exist(self, state):
    if str(state) not in self.q_table.index:
        # append new state to q table
        self.q_table = self.q_table.append(
            pd.Series(
                [0] * len(self.as),
                index=self.q_table.columns,
                name=str(state),
            )
        )

```

```

def state_extract(self,s):# 从环境因素中提取观察状态
    if s=='terminal':
        return s
    state=[]
    rgv_info=s[1]
    cnc_info=s[2]
    state.append(int(rgv_info['loc']))
    state.append(int(rgv_info['material_type']))
    for i in range(8):
        state.append(int(cnc_info['status'][i]))
        state.append(int(cnc_info['work_type'][i]))
        state.append(int(cnc_info['material_type'][i]))
    state=np.array(state)
    return state

```

5. 刀片分布搜索范围生成算法 smart_generate.py

```

import numpy as np
import itertools

```

```

def __work_type_generate_all__():
    j = 1

```

```

CNC_WoRK_TYPE = []
while j < 8:
    list1 = [1 for count in range(j)]
    extend_list = [2 for count in range(8-len(list1))]
    list1.extend(extend_list)
    list2 = []
    for i in range (1, len (list1) + 1):
        iter = itertools.permutations (list1, i)
        list2.append (list (iter))
    CNC_WoRK_TYPE.extend(list(set(list2[7])))
    j = j+ 1
return CNC_WoRK_TYPE

def __smart_work_type_genarate__(CNC_2_1 , CNC_2_2,Type = 1):
    ALL = np.array(__work_type_genarate_all__())
    if Type == 1:
        return ALL
    if CNC_2_1 < CNC_2_2:      # 需求更多的刀片 2
        mask = np.where(np.sum(ALL-1,axis=1)<4,False,True)
    else:      # 需求更多的刀片 1
        mask = np.where(np.sum(ALL-1,axis=1)<=4,True,False)
    return ALL[mask]

def smart_work_type_genarate(CNC_2_1 , CNC_2_2,Type = 1):
    ALL = __smart_work_type_genarate__(CNC_2_1 , CNC_2_2,Type)
    for group in ALL:
        yield group

```

6. 刀片分布搜索学习模式 train_for_arrangement.py

```

from enviroment import Env
from smart_generate import smart_work_type_genarate
from rl import QLearning
import enviroment
import pandas as pd
import numpy as np
# 设置全局变量
MAX_EPISoDES = 30

```

```

def update():
    score_record=[]
    for episode in range(MAX_EPISoDES):
        # 初始化 state 的观测值
        o = env.reset()

        while True:
            # 更新可视化环境

```



```

# env.render(o)

# RL 大脑根据 state 的观测值挑选 a
a = RL.choose_a_QL(o)

# 探索者在环境中实施这个 a, 并得到环境返回的下一个 state 观测值,
reward 和 done (是否是掉下地狱或者升上天堂)
o_, reward, done = env.step(a)

# RL 从这个序列 (state, a, reward, state_) 中学习
RL.learn(o, a, reward, o_)

# 将下一个 state 的值传到下一次循环, 并更新环境参数
if o_ != 'terminal':
    o = o_
env.update(o)

print("获得 reward:",str(reward),"采用动作:",a,"RGV 位置",o[1]['loc'], "剩余时
间:",env.time)
print("RGV 携带工料:",o[1]['material_type'])
print("RGV 状态",str(o[1]),"CNC 状态",str(o[2]))

print("加工物料:",str(env.score))

# 回合结束
if done:
    print("本回合加工物料:",str(env.score))
    score_record.append(env.score)

pd.DataFrame({'score':score_record}).to_csv('result/record/param3case4_record.csv',index=
False)

    mean_score = np.mean(score_record[-3:])
    if np.abs(mean_score-env.score) < 5 and len(score_record) >9:
        return score_record
    break
return score_record
# 结束并关闭窗口
print('over')
env.destroy(RL,path='result/q_result_p1c1.csv')

if __name__ == "__main__":
    # 定义环境 env 和 RL 方式
    Best_Score = []
    Type = []

```

```

CNC_2_1 = enviroment.CNC_2_1
CNC_2_2 = enviroment.CNC_2_2
smart_genarate = smart_work_type_genarate(CNC_2_1,CNC_2_2,2)
for group in smart_genarate:
    print('刀具策略为',group)
    Type.append(group)
    enviroment.CNC_WoRK_TYPE = group
    env = Env()
    RL = QLearning(as=list(range(env.n_as)))
    # 开始可视化环境 env
    score_record = update()
    Best_Score.append(np.max(score_record))
result = pd.DataFrame({'Best_Score':Best_Score,'Type':Type})
result.to_csv('result_param3_case4.csv',index = False)

```

7.第一种情况的调度模型算法 case1.py

```

import numpy as np
import pandas as pd

def cal_best_m(n = 8,process = 560,move_1 = 20,move_2 = 33, move_3 = 46,exchange_1 =
28,exchange_2 = 31,wash = 25):
    move = [0,0,move_1,move_1,move_2,move_2,move_3,move_3]
    Init = np.ceil(n/2)*exchange_1+np.floor(n/2)*exchange_2+(np.ceil(n/2)-1)*move_1
+move[n-1]
    wait1 = max(process - Init,0)
    Loop = Init + n*wash
    wait2 = max(process-Loop,0)
    time = 0
    max_loop = (8*3600- Init-wait1)*n/(Loop+wait2)
    print(max_loop)
    step = [i for i in range(1,n+1)]
    exchange = [exchange_1,exchange_2]
    num = [] # 记录 CNC 编号
    start = [] # 记录 CNC 上料开始时间
    end = [] # 记录 CNC 下料开始时间
    for i in step:
        num.append(i)
        start.append(time)
        if i % 2 == 1:
            time +=exchange[0]
        else:
            time += exchange[1]
        if i == n:
            time += move[n-1]

```

```

        break
    if i in [2,4,6]:
        time += move_1
time += wait1
while time < 8*3600:
    for i in step:
        num.append(i)
        start.append(time)
        end.append(time)
        time+=wash
        if i % 2 == 1:
            time += exchange[0]
        else:
            time += exchange[1]
        if i == n:
            time += move[n - 1]
            break
        if i in [2, 4, 6]:
            time += move_1
    time += wait2
for i in range(len(start)-len(end)):
    end.append(np.nan)
df = pd.DataFrame({'加工 CNC 编号':num,'上料开始时间':start,'下料开始时间':end},columns=['加工 CNC 编号','上料开始时间','下料开始时间'])
print(df)
df.to_csv('1.3.csv')# 输出结果
if __name__ == '__main__':
    cal_best_m(n = 8,process = 545,move_1 = 18,move_2 = 32, move_3 = 46,exchange_1 = 27,exchange_2 = 32,wash = 25)

```

8. 绘图脚本 draw.py

```

import pandas as pd
import matplotlib.pyplot as plt

def draw_plc1_r():
    record=pd.read_csv('result/record/param1case1_record.csv')
    plt.xlabel('Episode')
    plt.ylabel('score')
    plt.plot(record)
    plt.show()

def draw_plc1_pr_r():
    record = pd.read_csv('result/record/param1case1_predict_record.csv')
    plt.xlabel('Episode')

```

```

plt.ylabel('score')
plt.plot(record)
plt.show()

def draw_plc2_r():
    record = pd.read_csv('result/record/param1case2_record.csv')
    plt.xlabel('Episode')
    plt.ylabel('score')
    plt.plot(record)
    plt.show()

def draw_plc3_1_r():
    record = pd.read_csv('result/record/param1case3_1_record.csv')
    plt.xlabel('Episode')
    plt.ylabel('score')
    plt.plot(record)
    plt.show()

def draw_plc3_2_r():
    record = pd.read_csv('result/record/param1case3_2_record.csv')
    plt.xlabel('Episode')
    plt.ylabel('score')
    plt.plot(record)
    plt.show()

if __name__=="__main__":
    draw_plc3_2_r()

```

附录 x: python 安装教程

环境配置:

1. Python3.6.0 安装:

系统版本: WindowsXP 及以上

(1) 用浏览器进入 Python 官网 <https://www.python.org/downloads/>, 下拉页面找到版本 3.6.0 的下载链接

Want to help test development versions of Python? [Pre-releases](#)
Looking for Python 2.7? See below for specific releases

Looking for a specific release?
Python releases by version number:

| Release version | Release date | Click for more | |
|-------------------------------|--------------|--------------------------|-------------------------------|
| Python 3.6.2 | 2017-07-17 | Download | Release Notes |
| Python 3.6.1 | 2017-03-21 | Download | Release Notes |
| Python 3.4.6 | 2017-01-17 | Download | Release Notes |
| Python 3.5.3 | 2017-01-17 | Download | Release Notes |
| Python 3.6.0 | 2016-12-23 | Download | Release Notes |
| Python 2.7.13 | 2016-12-17 | Download | Release Notes |
| Python 3.4.5 | 2016-06-27 | Download | Release Notes |

[View older releases](#)

(2) 点击链接进入版本选择页面，下拉页面至末尾的 File 面板选择 Windows x86 executable installer，下载 python-3.6.0.exe，此为 python3.6.0 32 位系统版本，若系统是 64 位版本的也可以下载 Windows x86-64 executable installer，但不论是 64 位还是 32 位的系统，都可以运行 32 位版本的 python3.6.0

Python PSF Docs PyPI Jobs Community

python™

Search GO Socialize

About Downloads Documentation Community Success Stories News Events

Python 3.6.0

Release Date: 2016-12-23

Note

Python 3.6.0 is now the latest maintenance release of Python 3.6 and supersedes 3.6.0. [Get 3.6.6 here](#). [Python 3.7](#), a newer feature release, is now also available.

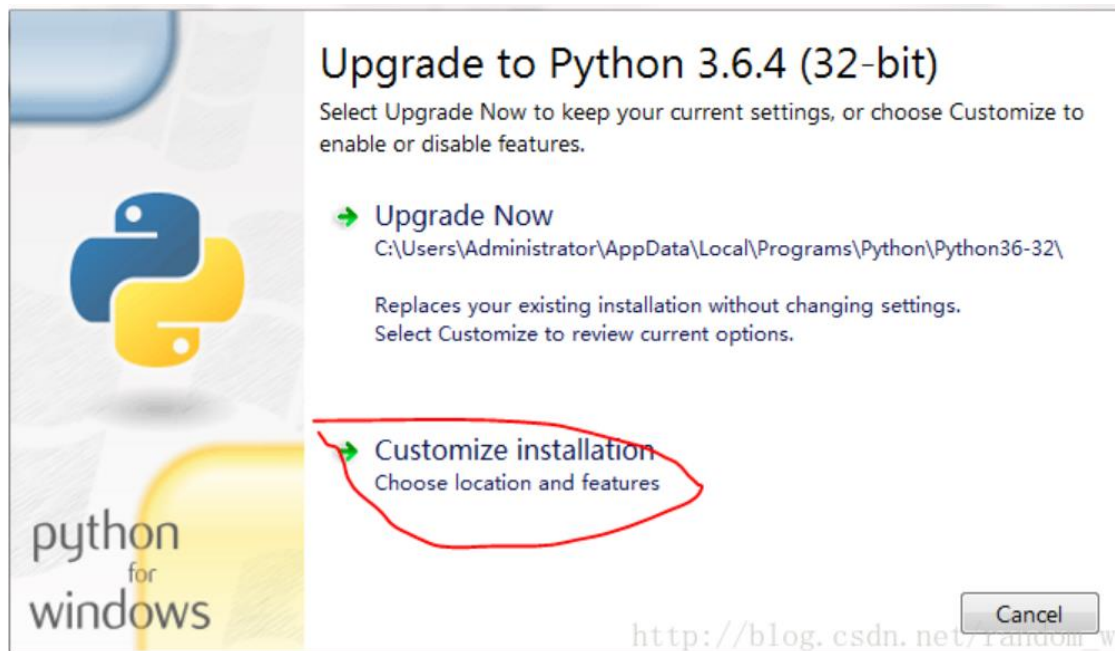
Python 3.6.0 is the newest major release of the Python language, and it contains many new features and optimizations. See the [What's New In Python 3.6](#) document for more information.

[Full Changelog](#)

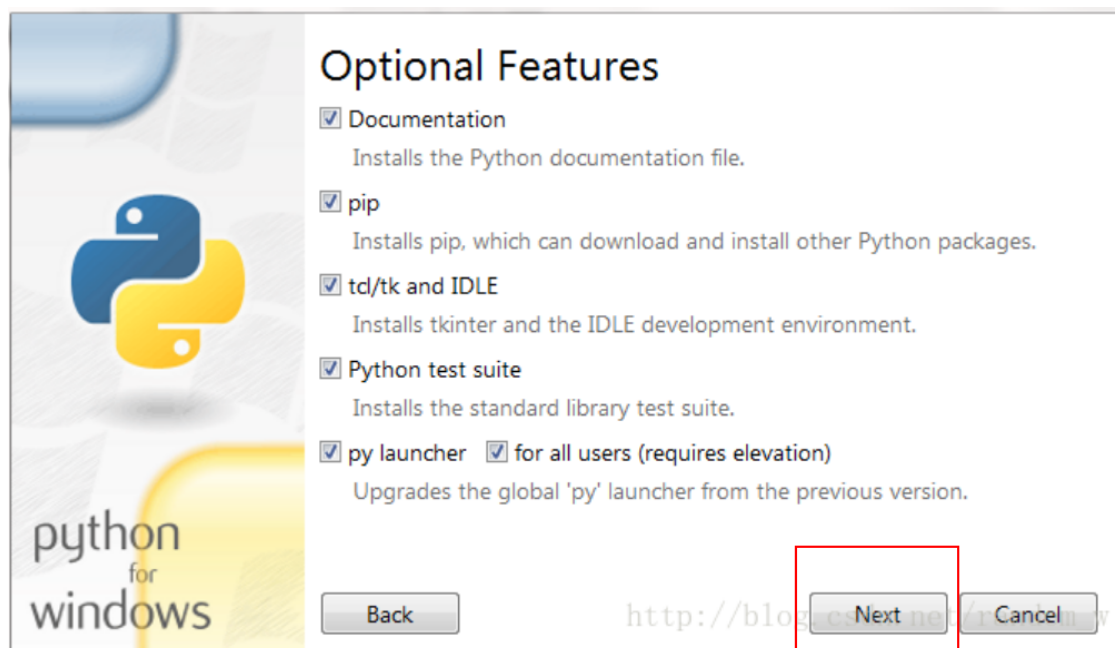
Files

| Version | Operating System | Description | MD5 Sum | File Size | GPG |
|-----------------------------------------------------|------------------|-----------------------------|----------------------------------|-----------|---------------------|
| Gzipped source tarball | Source release | | 3f7062ccf8be76491884d0e47ac8b251 | 22256403 | SIG |
| XZ compressed source tarball | Source release | | 82b143ebbf4514d7e05876bed7a6b1f5 | 16805836 | SIG |
| Mac OS X 64-bit/32-bit installer | Mac OS X | for Mac OS X 10.6 and later | 72acb0175e7622dec7e1b160a43b8c42 | 27442222 | SIG |
| Windows help file | Windows | | 6a842a15ab3b4aa316c91a9779db82ec | 7940890 | SIG |
| Windows x86-64 embeddable zip file | Windows | for AMD64/EM64T/x64 | 0ec0caeea75bae5d2771cf619917c71f | 6925798 | SIG |
| Windows x86-64 executable installer | Windows | for AMD64/EM64T/x64 | 71c9d30c1110abf7f80a428970ab8ec2 | 31505640 | SIG |
| Windows x86-64 web-based installer | Windows | for AMD64/EM64T/x64 | 25b8b6c93a098dfade3b014630f9508e | 1312376 | SIG |
| Windows x86 embeddable zip file | Windows | | 1adf2fb735c5000af32d42c39136727c | 6315855 | SIG |
| Windows x86 executable installer | Windows | | 38d9b036b25725f6acb553d4aece4db4 | 30566536 | SIG |
| Windows x86 web-based installer | Windows | | f71f4590be2cc5cdc43069594d4ea98d | 1286984 | SIG |

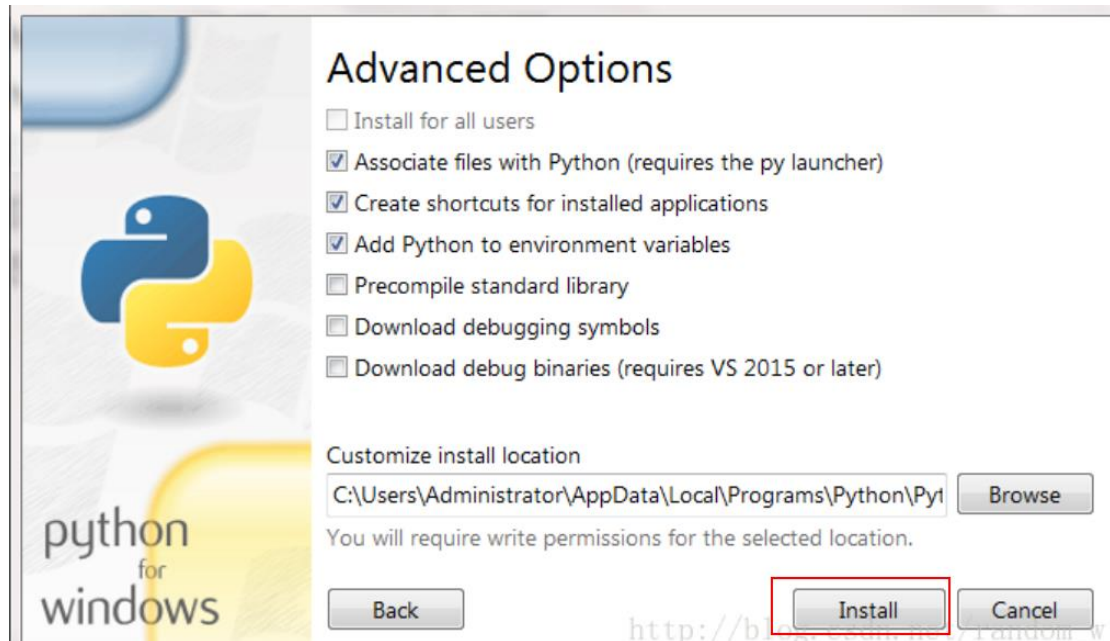
(3) 下载完成后双击安装包，进入如下界面，选择圈起来的项



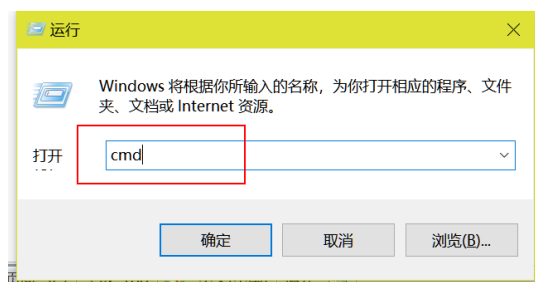
(4) 默认点击 next



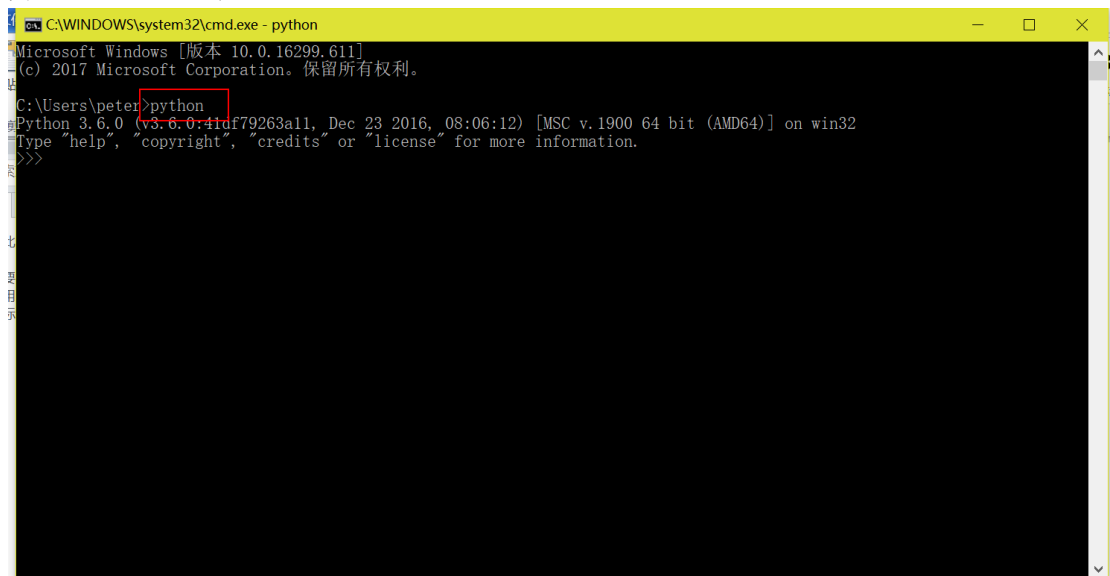
(5) 点击 install, 完成安装



(6) 按下快捷键 win+R 可以打开运行界面，在运行界面输入 cmd 打开命令行



(7) 直接在命令行中输入 python, 弹出以下界面即说明 python3.6.0 成功安装, 并已自动配置好环境变量

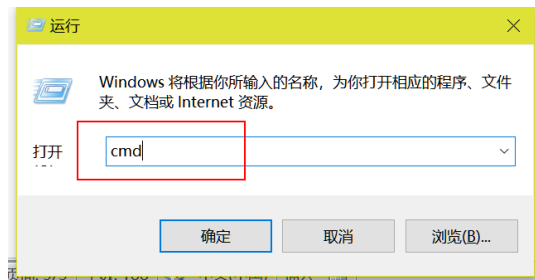


参考文献: windows 上安装 python3 教程以及环境变量配置

https://blog.csdn.net/random_w/article/details/78897365

2. 需要的第三方库的安装

(1) 需要安装的 python 第三方库有 5 个, 分别为 Numpy, Pandas, Matplotlib, Scikit-learn, Seaborn, 安装过程并不繁琐. 首先按下快捷键 win+R 可以打开运行界面, 在运行界面输入 cmd 打开命令行。



(2) 然后在命令行中直接输入: `pip install numpy`, 然后按下回车, 便完成了 numpy 的安装



(3) 同样的, 在命令行中按顺序输入:

```
pip install pandas
pip install matplotlib
pip install piglet
pip install seaborn
```

即可完成剩余所有第三方库的安装, 在进入 python 界面后输入 `import numpy`, 可以看到 numpy 是否安装成功, 其它的库也可以用同样的方法测试。

(4) 如果安装未能成功, 参考

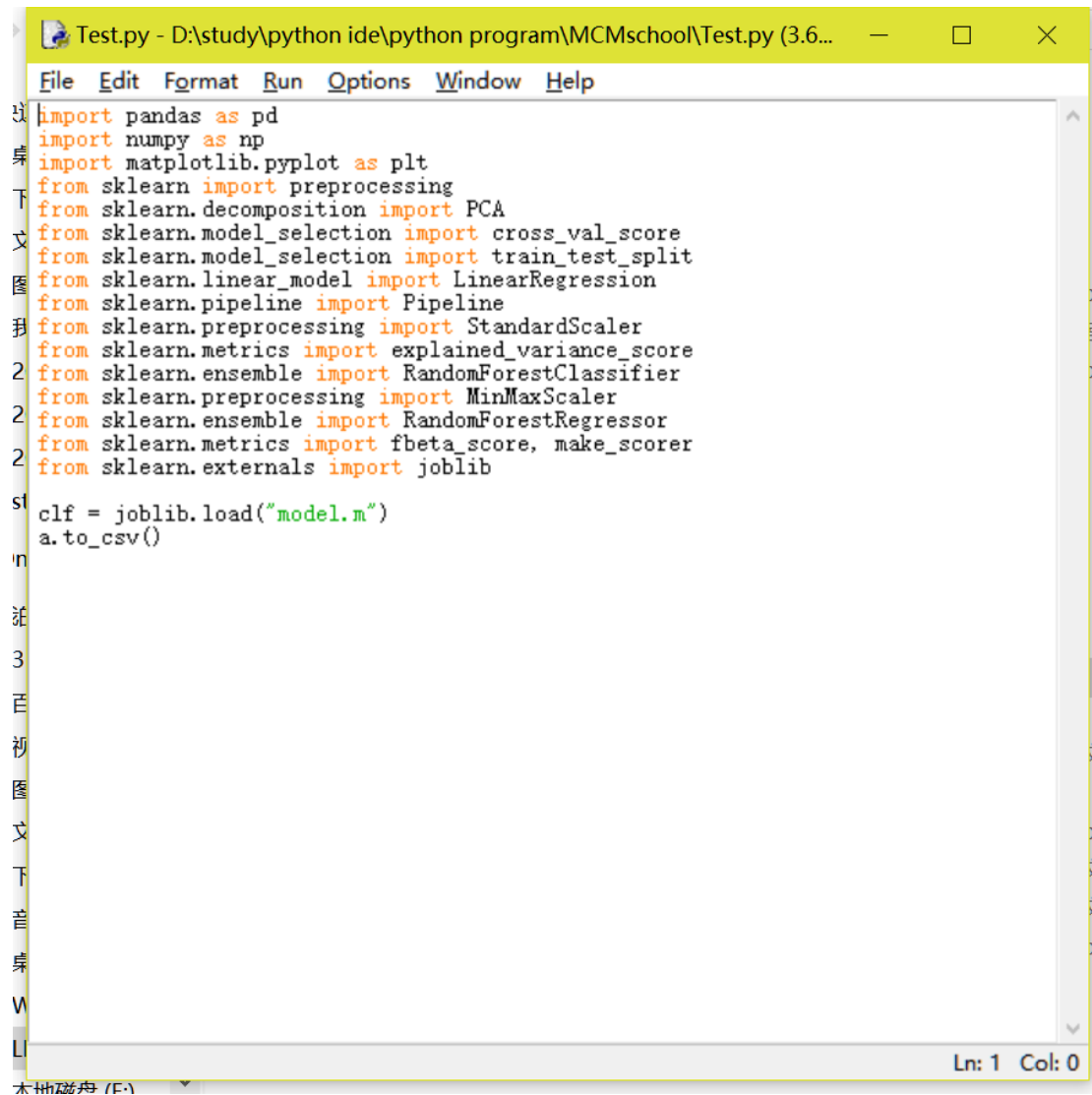
<https://www.cnblogs.com/yehui-mmd/p/7932169.html> 通过下载然后本地 pip install 第三方库的 whl 文件实现安装。

参考文献: <https://www.jb51.net/article/131357.htm>

3. 查看 python 文件

(1) 在对应的 python 文件上右键, 选择 Edit with IDLE, 即出现 python 脚本编写界面, 在这里可以查看代码内容





```
Test.py - D:\study\python ide\python program\MCMschool\Test.py (3.6...
File Edit Format Run Options Window Help
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import explained_variance_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.externals import joblib

clf = joblib.load("model.m")
a.to_csv()
```

Ln: 1 Col: 0