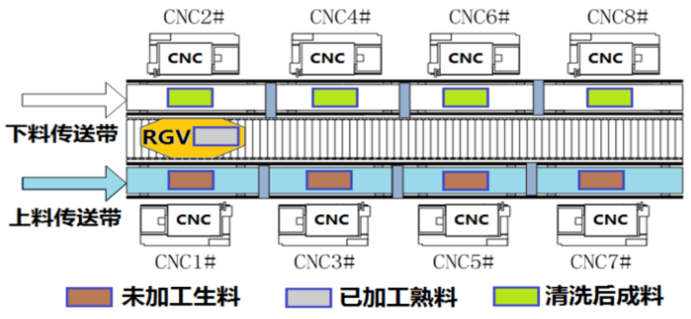
基于遗传及贪婪算法的智能 RGV 动态调度规划研究
——如何最大限度提高系统产量

2018 年 9 月 16 日

基于遗传及贪婪算法的智能 RGV 动态调度规划研究
——如何最大限度提高系统作业效率

摘要

RGV（轨道式导引小车）是一种在车间或者自动化立体仓库中沿着轨道运行的物料运送工具。作为输送系统的重要一环，RGV 的调度是否合理、是否能做出最优的上下料决策，很大程度上决定了整个生产系统的作业效率，因此 RGV 的动态调度是一个研究的热点。



图表 1 智能加工系统示意图

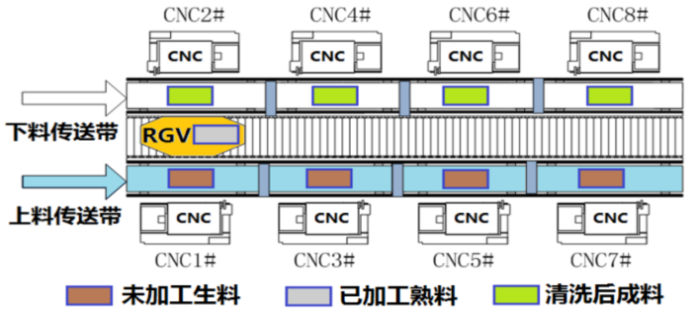
在如图表 1 所示的生产线情景下，本文先对一道工序的物料加工及两道工序物料加工两种情况分别给出相应的算法模型对 RGV 的决策进行动态规划：针对一道工序的情况，利用贪婪算法给出了 RGV 的一个动态调度模型，每一步决策都做局部最优解最终得到全局最优解，较好地解决了题目给出的问题；针对两道工序的情况，基于遗传算法并以贪婪算法为辅，对 CNC 的类型及位置分布利用多次迭代得出符合题意的较优解，给出了 RGV 的另一个动态调度模型，较好地解决了题目给出的问题；而针对 CNC 在加工过程中可能发生故障的情况，则在算法中增加了随机的“故障状态”来体现机器故障这一事件，并在原有的贪心策略中增设针对故障处理的贪心策略，较好地模拟出了总体的故障情况。

根据本文提出的模型，能通过调整 RGV 的工作决策策略，最大限度地提高系统的作业效率，更快完成工厂的生产指标。

关键词：RGV 动态调度、遗传算法、贪婪算法、系统作业效率

一、　问题重述

图表 2 是一个智能加工系统的示意图，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。RGV 是一种无人驾驶、能在固定轨道上自由运行的智能车。它根据指令能自动控制移动方向和距离，并自带一个机械手臂、两只机械手爪和物料清洗槽，能够完成上下料及清洗物料等作业任务。



图表 2 智能加工系统示意图

每班次工作有下列三种情况：

（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）两道工序的物料加工作业情况，每个物料的第一和第二道工序分别由两台不同的 CNC 依次加工完成；

（3）CNC 在加工过程中可能发生故障，此时未完成的物料报废，并开始人工修复。故障排除后即刻加入作业序列。

1.1 问题的提出

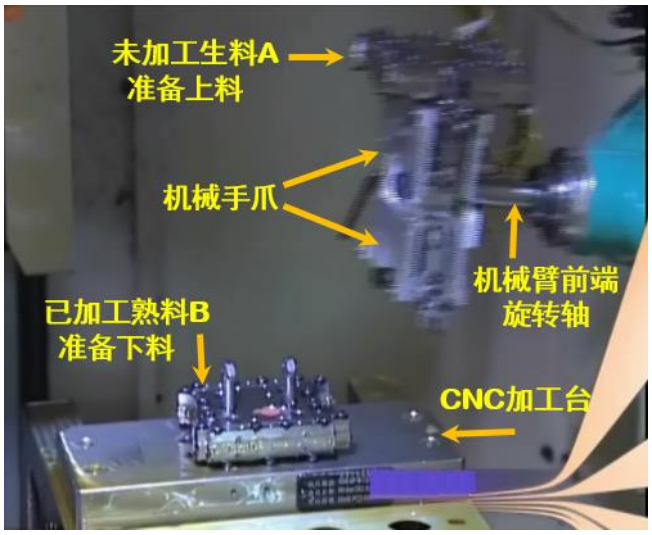
根据题干，在以上三种情况下分别提出以下问题：

（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）利用给出的系统作业参数的 3 组数据分别检验模型的实用性和算法的有效性，给出 RGV 的调度策略和系统的作业效率，并导出具体的结果。

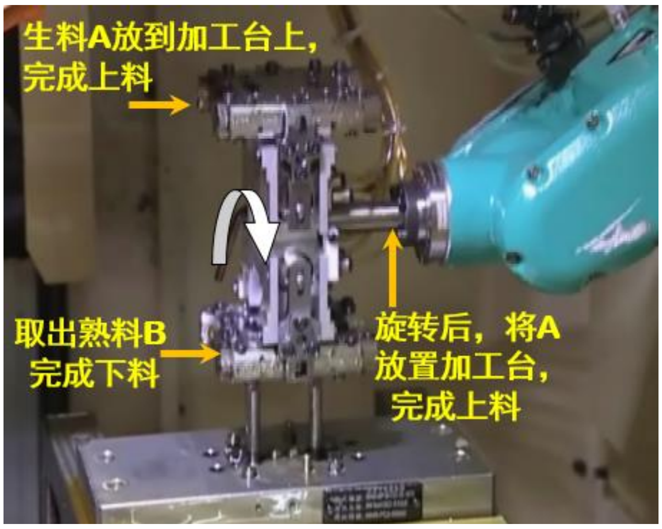
二、　问题分析

2.1 RGV 工作原理



图表 3 RGV 工作图 1

在图表 3 中，机械臂前端上方手爪抓有 1 个生料 A，CNC 加工台上有 1 个熟料 B。RGV 机械臂移动到 CNC 加工台上方，机械臂下方空置的手爪准备抓取熟料 B，在抓取了熟料 B 后即完成下料作业。



图表 4 RGV 工作图 2

在图表 4 中，RGV 机械臂下方手爪已抓取了 CNC 加工台上的熟料 B 抬高手臂，并旋转手爪，将生料 A 对准加工位置，安放到 CNC 加工台上，即完成上料作业。

2.2 系统作业流程

（1）智能加工系统通电启动后，所有 CNC 都处于空闲状态。

（2）在工作正常情况下，如果 CNC 处于空闲状态，则向 RGV 发出上料需求信号；否则 CNC 处于加工作业状态，加工作业完成即刻向 RGV 发出需求信号。

（3）RGV 收到 CNC 的需求信号后，自行确定该 CNC 的上下料作业次序，并依次按顺序为其上下料作业。根据需求指令，RGV 运行至需要作业的某 CNC 处，同时上料传送带将生料送到该 CNC 正前方，供 RGV 上料作业。

（4）在 RGV 为某 CNC 完成一次上下料作业后，就会转动机械臂，将一只机械手上的熟料移动到清洗槽上方，进行清洗作业。（只清洗加工完成的熟料）

（5）RGV 在完成一项作业任务后，立即判别执行下一个作业指令。此时如果没有接到其他的作业指令，则 RGV 就在原地等待直到下一个作业指令。

某 CNC 完成一个物料的加工作业任务后，即刻向 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。

（6）系统周而复始地重复（3）至（5），直到系统停止作业，RGV 回到初始位置。

三、 模型假设

（1）智能加工系统通电启动后,RGV 在 CNC1#和 CNC2#正中间的初始位置,所有 CNC 都处于空闲状态。

（2）每台 CNC 完全相同；同一时间只能加工一个物料；每次只能加工一道工序, 在加工过程中不能变更所加工的工序。

（3）RGV 同一时间只能执行移动、停止等待、上下料和清洗作业中的一项。

（4）第三种情况中，每台 CNC 在一次加工过程中发生故障的概率约为 1%，修复时间为 10min～20min。

（5）系统作业效率=每班次计算产量／每班次理想化产量。其中每班次理想化产量是在成品均用最短时间产出、无故障发生的情况下计算所得；特别地，在两道工序的情况下，两种 CNC 之比为 1：1，且按照两道工序中产量较多的一道计算成品量。

四、　符号说明

系统作业参数	符号
RGV 移动 1 个单位所需时间	t_1
RGV 移动 2 个单位所需时间	t_2
RGV 移动 3 个单位所需时间	t_3
CNC 加工完成一个一道工序的物料所需时间	T_{11}
CNC 加工完成一个两道工序物料的第一道工序所需时间	T_{21}
CNC 加工完成一个两道工序物料的第二道工序所需时间	T_{22}
RGV 为 CNC1#，3#，5#，7#一次上下料所需时间	T_1
RGV 为 CNC2#，4#，6#，8#一次上下料所需时间	T_2
RGV 完成一个物料的清洗作业所需时间	T_c

表格 1 符号说明表

五、　模型建构

5.1 一道工序情况下的模型

5.1.1 贪心算法简介

贪心算法（又称贪婪算法）的基本思路是，从问题的某一个初始解出发一步一步地进行，根据某个优化测度，每一步都要确保能获得局部最优解。每一步只考虑一个数据，它的选取应该满足局部优化的条件。在对问题求解时，总是做出在当前看来是最好的选择。也就是说不从整体最优上加以考虑，所做出的是在某种意义上的局部最优解。

贪心算法的基本要素有:

(1)贪心选择。贪心选择是指所求问题的整体最优解可以通过一系列局部最

优的选择，即贪心选择来达到。贪心选择是采用从顶向下、以迭代的方法做出相继选择，每做一次贪心选择就将所求问题简化为一个规模更小的子问题。

(2)最优子结构。当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。问题的最优子结构性质是该问题可用贪心算法或动态规划算法求解的关键特征。

为了解决问题，需要寻找一个构成解的候选对象集合，它可以优化目标函数，使贪婪算法一步一步的进行。起初，算法选出的候选对象的集合为空。接下来的每一步中，根据选择函数，算法从剩余候选对象中选出最有希望构成解的对象。如果集合中加上该对象后不可行，那么该对象就被丢弃并不再考虑；否则就加到集合里。每一次都扩充集合，并检查该集合是否构成解。如果贪婪算法正确工作，那么找到的第一个解通常是最优的。

5.1.2 贪心算法应用

在对产品仅需加工一道工序的情况下，由于每台 CNC 的加工时间相同，RGV 只需按运输加工时间由短到长来排列空闲 CNC 的上下料次序，即体现出策略选择的无后效性，因此这里选择使用贪心算法.

在本场景中，贪心策略主要体现在 RGV 接到一个新的 CNC 上料信号后，将其与已有的上料信号所在的 CNC 进行位置比对，按运输加工所需时间由短到长进行优先级排列；每次都如此排序选择，最后合成一个整体最优解。

设 RGV 接到了 n 台空闲 CNC 的上料请求，机号分别为 L_1，L_2，...，L_n，它们的运输加工时间分别为 $Time_1$，$Time_2$，...，$Time_n$，其中 $Time_i=T_k+t_j+T_c(i=1,2,...,n;\ k=1,2;\ j=1,2,3)$。将 Time 按小到大排列后得出一个排列 $Time_{i1}$，$Time_{i2}$，...，$Time_{in}$，其所对应的排列 L_{i1}，L_{i2}，...，L_{in} 即为 RGV 所要处理 CNC 的优先级排序。

依据以上调度模型编写程序（具体程序见支撑材料 Case_1_result），并分别代入表格 2 中智能加工系统作业参数的三组数据，最终得到三组数据分别对应的结果如表格 3（详细 RGV 运行策略见支撑材料 Case_1_result）

系统作业参数	第 1 组	第 2 组	第 3 组
RGV 移动 1 个单位所需时间	20	23	18
RGV 移动 2 个单位所需时间	33	41	32
RGV 移动 3 个单位所需时间	46	59	46
CNC 加工完成一个一道工序的物料所需时间	560	580	545
CNC 加工完成一个两道工序物料的第一道工序所需时间	400	280	455
CNC 加工完成一个两道工序物料的第二道工序所需时间	378	500	182
RGV 为 CNC1#，3#，5#，7#一次上下料所需时间	28	30	27
RGV 为 CNC2#，4#，6#，8#一次上下料所需时间	31	35	32
RGV 完成一个物料的清洗作业所需时间	25	30	25

表格 2 智能加工系统作业参数的 3 组数据

	运算结果（每班次产量）	系统作业效率
第 1 组	344	91.5%
第 2 组	323	89.7%
第 3 组	352	91.7%

表格 3 Case1 Result

5.2 两道工序情况下的模型

5.2.1 遗传算法介绍

遗传算法（Genetic Algorithm）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，是一种通过模拟自然进化过程搜索最优解的随机化搜索方法。其主要特点是直接对结构对象进行操作，具有内在的隐并行性和更好的全局寻优能力；采用概率化的寻优方法，能自动获取和指导优化的搜索空间，自适应地调整搜索方向，不需要确定的规则。

遗传算法的基本运算过程如下：

（1）初始化：设法把握最优解所在范围，在此范围内设定初始群体。

（2）个体评价：根据对问题解的接近程度对个体各指定一个适应度的值。

（3）选择运算：将选择算子作用于群体，目的是将优良个体的基因直接遗传至下一代，或通过配对交叉产生新个体再遗传到下一代。

（4）交叉运算：将交叉算子作用于群体，交叉运算是遗传算法中产生新个体的主要操作过程。

（5）变异运算：将变异算子作用于群体，即变动某些个体串的基因值。

（6）终止：迭代遗传的代数达到预设代数后，且当前子代的适应度得到较好收敛，遗传算法终止，以进化过程中所得到的具有最大适应度的个体作为最优解输出。

5.2.2 遗传算法应用

对二道工序情况下，每个位置的 CNC 加工第一道工序或加工第二道工序的选择问题使用遗传算法。以群体中每代有四个个体为例，具体操作如下：

首先，对每个个体编码。在本场景中，个体是一条 CNC 生产线，其基因组用八位二进制字符串表示。其中 0 表示该 CNC 加工第一道工序，1 表示该 CNC 加工第二道工序；每一个个体都是八位长度的二进制数，从左往右的位次数对应 CNC 生产线的位置。如 10011011 表示如图表 5 所示的 CNC 设置：

位置编号	#2	#4	#6	#8
CNC 类型	0	1	0	1
RGV				
位置编号	#1	#3	#5	#7
CNC 类型	1	0	1	1

图表 5 10011101 所代表的 CNC 设置

编码后进行如下几步：

（1）初始群体的产生。本场景中取群体规模的大小为 4，即群体由 4 个个体组成，每个个体可通过随机方法产生。

（2）适应度计算。对于每一个个体而言，以其在贪心算法下得出的每班次产量作为个体适应度。

（3）选择运算，采用与个体适应度成正比的概率来选定个体基因遗传到下一

代群体中。具体操作如下：

设个体 i 的适应度为 f_i，选择的次数为 N，则个体 i 所携基因遗传到下一代的概率为 $f_i/\Sigma f_i$；每个概率值组成一个区域，全部概率值之和为 1；随机产生一个 0 到 1 之间的数，依据该随机数出现在上述哪一个概率区域内来确定哪一个个体被选中；循环 N 次，得出最终选中对象。

最终结果如表格 4 所示：

个体编号	初始种群	适应值	占总适值的百分比	选择次数	选择结果
1	10011011	168	24.8%	1	11001011
2	01100010	180	26.6%	1	10011011
3	00110111	163	24.0%	1	01100010
4	11001011	166	24.6%	1	00110111

表格 4 个体选择运算示例

（4）交叉运算。本题采用单点交叉的方法，先对群体进行随机的两两配对，对每一组分别随机设置交叉点位置后，再相互交换配对染色体之间的交叉点后的基因。具体操作及结果如表格 5 所示：

个体编号	选择结果	配对情况	交叉点位置	交叉结果
1	10|011011	1-2	1-2:2	10100010
2	01|100010			10011011
3	0011|0111	3-4	3-4:4	00111011
4	1100|1011			11001011

表格 5 交叉运算示例

（5）变异运算。本场景中采用基本位变异的方法来进行变异运算，首先确定各个体的基因变异位置，然后依照某一概率将变异点的原有基因值取反。

具体操作结果如表格 6 所示，其中的变异点表示随机产生的变异点位置，即变异点设置在该基因座处。

个体编号	交叉结果	变异点	变异结果	子代群体
1	10100010	4	10110010	10110010
2	10011011	5	10010011	10010011
3	00111011	2	01111011	01111011
4	11001011	6	11001111	11001111

表格 6 变异运算示例

（6）终止。设遗传代数上限为 T，当遗传至 T 代时终止遗传，取出现过的所有个体中适应度最高的作为最优解。

遗传算法还有以下几个关键：

（1） 种群规模的设定。种群规模指的是种群中个体的数量。如果种群规模太小，则种群的多样性不够，不但不能找到多种区域的解，而且在交叉过程中容易造成“近亲结婚”的现象，这样会使种群过快的收敛，从而只能找到局部的最优解，无法找到全局的最优解；而如果种群太大，种群的多样性增加，固然可以找到更多的优化解，但是会大大增加遗传算法的搜索速度，影响效率，在实际应用中受到限制。一般取 20~100 比较合适，具体的数量可以根据实际问题的复杂程度确定。

（2） 交叉概率的选择对遗传算法的性能影响比较大，一般来说要通过多次试验后，根据实施的效果来确定。如果交叉概率的取值过大，则种群很快就会收敛，往往会收敛到一个局部最优解，无法找到全局最优解；而如果交叉概率太小，则会导致收敛速度过慢，甚至无法收敛。一般来说交叉概率取值在 0.7~1.0 之间比较合适。

（3） 变异操作可以产生新的基因，在种群中引入全新的个体，从而实现局部的优化。变异操作发生的概率相对比较小，但是当个体的多样性陷入停滞状态时，变异能够起到很好的推动作用。个体变异也有一定的概率，变异概率的取值也是要通过多次实验来获得，一般建议取值为 0.001~0.1。

综上，本题取定种群规模 N=16，交叉概率 P1=0.9，编译概率 P2=0.1,迭代数为 T=30。

本题依据以上模型编写程序（具体程序见支撑材料 Case_2_result），并分别代入表格 2 中智能加工系统作业参数的三组数据。最终得到三组数据分别对应的结果如表格 7（详细 RGV 运行策略见支撑材料 Case_2_result）

	运算结果（每班次产量）	系统作业效率
第 1 组	221	82.5%
第 2 组	193	56.8%
第 3 组	238	48.4%

表格 7 Case2.Result

5.3 CNC 在加工过程中随机出现故障情况下的模型

5.3.1 一道工序+故障情况下的模型

如 5.1 模型，仍使用贪心算法，但在贪婪算法中加入故障机制，具体操作如下：每台 CNC 开始加工后，有 1%的概率进入“故障模式”。若选择进入“故障模式”，则在该 CNC 接下来的加工过程中随机抽取一个时刻 t_0,从 t_0 时刻开始直到 t_0+t（t 是从 10～20 中随机抽取的实数，单位为分钟）时刻，该台 CNC 不参与加工且未完成的物料报废。t_0+t 时刻开始，该台 CNC 进入空闲状态（无成品）。

依据以上模型编写程序（具体程序见支撑材料 Case_3_result_1），并分别代入表格 2 中智能加工系统作业参数的三组数据。最终得到三组数据分别对应的结果如表格 8（详细 RGV 运行策略见支撑材料 Case_3_result_1）

	运算结果（每班次产量）	系统作业效率
第 1 组	338	89.9%
第 2 组	316	87.8%
第 3 组	345	89.8%

表格 8 Case3 Result1

5.3.2 两道工序+故障情况下的模型

如 5.2 模型，仍使用遗传算法+贪心算法，但在贪婪算法中加入故障机制，具体操作如下：每台 CNC 开始加工后，有 1%的概率进入“故障模式”。若选择进入“故障模式”，则在该 CNC 接下来的加工过程中随机抽取一个时刻 t_0,从 t_0时刻开始直到 t_0+t（t 是从 10～20 中随机抽取的实数，单位为分钟）时刻，该台 CNC 不参与加工且未完成的物料报废。t_0+t 时刻开始，该台 CNC 进入空闲状态（无成品）。

依据以上模型编写程序（具体程序见支撑材料 Case_3_result_2），并分别代入表格 2 中智能加工系统作业参数的三组数据。最终得到三组数据，分别对应的结果如表格 9（详细 RGV 运行策略见支撑材料 Case_3_result_2）

	运算结果（每班次产量）	系统作业效率
第 1 组	217	81.0%
第 2 组	187	55.0%
第 3 组	235	47.8%

表格 9 Case3 Result2

六、　误差分析

（1）二道工序的情况下所使用的遗传算法有其局限性：如容易过早收敛，导致可能只得到满意解而不是最优解；也可能收敛过晚导致无法得到最终解。

（2）对系统作业效率公式中每班次理想化产量的计算，未能使用某一算法较精确地得出。

（3）在故障的处理上，本文仅做了简单的随机处理，而在实际情况中故障发生情况应服从泊松分布，因此模型对故障情况的模拟拟合度仍未达到理想高度。

七、　模型评价

7.1 模型评价

学者指出，遗传算法的整体搜索策略和优化搜索方法在计算时不依赖于梯度信息或其它辅助知识，而只需要影响搜索方向的目标函数和相应的适应度函数，因此其提供了一种不依赖于问题具体领域而能求解复杂系统问题的通用框架，使得它广泛应用于许多科学领域。

本文在遗传算法的基础上，结合贪婪算法对几种情况下 RGV 的动态调度规划各给出了较合适的算法模型，并较好地解决了题目中给出的问题。但由于遗传算法本身一定的局限性，具体操作过程中仍可能出现对最优解的计算偏差。

7.2 模型推广

本文提出的以遗传算法为主，贪婪算法为辅的 RGV 动态调度规划模型，除了能应用在题目加工场景中外，还可以应用于对自动化立体仓库（AS/RS）系统的运输效率的研究。

参考文献

[1]刘永强.基于遗传算法的 RGV 动态调度研究.工业工程与管理.2012

[2]陈华.基于 TS 算法的直线往复 2_RGV 系统调度研究.2015

[3]姜启源、谢金星等.数学模型（第四版）.2010

[4]江唯、何非等.基于混合算法的环形轨道 RGV 系统调度优化研究.计算机工程与应用.2016

附录（由于对于相同的情况算法相同故以第一组数据为例展示代码，其余代码都在支撑数据里）

1. 以第一组生产一道工序产品的代码：

```
from CNC import CNC
from RGV import RGV


time = 0
goodsnum = 0
relativeloca = [10000, 0, 0, 0, 0, 0, 0, 0, 0]
rgv = RGV([0, 20, 33, 46], 28, 31, 25, 1)
workingtime = 560
cnclist = []
sequence = [0]
uptsequence = [0]
updowntsequence = [0]


def addtime(secs):
    global time
    global rgv
    global cnclist
    time += secs
    for k in range(1, 8 + 1):
        cnclist[k].addmytime(secs)


def rgvmovement(steps):
    global time
    global rgv
    global cnclist
    rgv.move(steps)
    addtime(rgv.movetime[abs(steps)])

def oddup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    cnclist[choice].changestatu()
    sequence.append(choice)
    uptsequence.append(time)
    addtime(rgv.oddchangetime)

def odddownup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    sequence.append(choice)
    uptsequence.append(time)
    updowntsequence.append(time)
    addtime(rgv.oddchangetime)
    addtime(rgv.washtime)
    goodsnum += 1
    rgvmovement(0)
    cnclist[choice].goodtrans()
    cnclist[choice].changestatu()
```

```
def evenup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    cnclist[choice].changestatu()
    sequence.append(choice)
    uptsequence.append(time)
    addtime(rgv.evenchangetime)

def evendownup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    sequence.append(choice)
    uptsequence.append(time)
    updowntsequence.append(time)
    addtime(rgv.evenchangetime)
    addtime(rgv.washtime)
    goodsnum += 1
    rgvmovement(0)
    cnclist[choice].goodtrans()
    cnclist[choice].changestatu()

def opration():
```

```
global time
global goodsnum
global rgv
global cnclist
global useamount
for j in range(1, 8 + 1):
    relativeloca[j] = abs(cnclist[j].num - rgv.location)
    if cnclist[j].having == True:
        relativeloca[j] += 1000
    if cnclist[j].working == True:
        relativeloca[j] += 1000
if min(relativeloca) >= 2000:
    addtime(1)
elif min(relativeloca) >= 1000:
    choice = relativeloca.index(min(relativeloca))
    if relativeloca[1] == 1006 and relativeloca[2] == 1005:
        choice = 1
    truediff = (choice - rgv.location)
    if choice % 2 == 0:
        if truediff == 1:
            rgvmovement(0)
            evendownup(choice)
        elif truediff == 3:
            rgvmovement(1)
            evendownup(choice)
        elif truediff == 5:
            rgvmovement(2)
            evendownup(choice)
        elif truediff == 7:
            rgvmovement(3)
            evendownup(choice)
        elif truediff == -1:
            rgvmovement(-1)
            evendownup(choice)
        elif truediff == -3:
            rgvmovement(-2)
            evendownup(choice)
        elif truediff == -5:
            rgvmovement(-3)
            evendownup(choice)
    else:
        if truediff == 0:
            rgvmovement(0)
            odddownup(choice)
        elif truediff == 2:
            rgvmovement(1)
            odddownup(choice)
        elif truediff == 4:
            rgvmovement(2)
            odddownup(choice)
        elif truediff == 6:
            rgvmovement(3)
            odddownup(choice)
        elif truediff == -2:
            rgvmovement(-1)
            odddownup(choice)
        elif truediff == -4:
            rgvmovement(-2)
            odddownup(choice)
        elif truediff == -6:
            rgvmovement(-3)
            odddownup(choice)
else:
    choice = relativeloca.index(min(relativeloca))
    truediff = (choice - rgv.location)
    if choice % 2 == 0:
        if truediff == 1:
            rgvmovement(0)
            evenup(choice)
        elif truediff == 3:
            rgvmovement(1)
            evenup(choice)
        elif truediff == 5:
            rgvmovement(2)
            evenup(choice)
        elif truediff == 7:
            rgvmovement(3)
            evenup(choice)
        elif truediff == -1:
            rgvmovement(-1)
            evenup(choice)
        elif truediff == -3:
            rgvmovement(-2)
            evenup(choice)
        elif truediff == -5:
            rgvmovement(-3)
            evenup(choice)
        else:
            if truediff == 0:
                rgvmovement(0)
                oddup(choice)
            elif truediff == 2:
                rgvmovement(1)
                oddup(choice)
            elif truediff == 4:
                rgvmovement(2)
                oddup(choice)
            elif truediff == 6:
                rgvmovement(3)
                oddup(choice)
            elif truediff == -2:
                rgvmovement(-1)
                oddup(choice)
            elif truediff == -4:
                rgvmovement(-2)
                oddup(choice)
            elif truediff == -6:
                rgvmovement(-3)
                oddup(choice)


for i in range(0, 8 + 1):
    cnc = CNC(i, False, workingtime, 0)
    cnclist.append(cnc)
while time <= (8*3600):
    opration()
    print("%-8s" % "CNC", "%-8s" % "Uptime", "%-8s" % "Downtime")
    for i in range(1, len(updowntsequence)):
        print("%-8d" % sequence[i], "%-8d" % uptsequence[i], "%-
8d" % updowntsequence[i])
    print("Total goods number: ", goodsnum)
```

2. 基于遗传算法以第一组生产两道道工序产品的代码：

```
    from CNC import CNC
    from RGV import RGV
    from POPULATION import  Population


    def express():
        global cnclist
        global fcnc
        global scnc
        for n in range(1, 8 + 1):
            if genome[n] == 1:
                cnclist[n].whethersec = True
                scnc.append(cnclist[n].num)
        else:
            fcnc.append(cnclist[n].num)



def addtime(secs):
    global time
    global rgv
    global cnclist
    time += secs
    for k in range(1, 8 + 1):
        cnclist[k].addmytime(secs)



def rgvmovement(steps):
    global time
    global rgv
    global cnclist
    rgv.move(steps)
    addtime(rgv.movetime[abs(steps)])



def oddup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    cnclist[choice].changestatu()
    if cnclist[choice].whethersec == False:
            sequence1.append(choice)
            uptsequence1.append(time)
            addtime(rgv.oddchangetime)
        elif cnclist[choice].whethersec == True and rgv.haveone ==
True:
            sequence2.append(choice)
            uptsequence2.append(time)
            rgv.putone()
            addtime(rgv.oddchangetime)
        else:
            addtime(1)



    def odddownup(choice):
        global time
        global rgv
        global cnclist
        global goodsnum
        cnclist[choice].goodtrans()
        if cnclist[choice].whethersec == False:
            sequence1.append(choice)
            uptsequence1.append(time)
            updowntsequence1.append(time)
            rgv.catchone()
            addtime(rgv.oddchangetime)
            cnclist[choice].goodtrans()
            cnclist[choice].changestatu()
        elif cnclist[choice].whethersec == True and rgv.haveone ==
False:
```

```
                sequence2.append(choice)
                updowntsequence2.append(time)
                addtime(rgv.oddchangetime)
                addtime(rgv.washtime)
                goodsnum += 1
                print("2 end when rgv do not has 1")
            elif cnclist[choice].whethersec == True and rgv.haveone == 
True:
                sequence2.append(choice)
                uptsequence2.append(time)
                updowntsequence2.append(time)
                addtime(rgv.oddchangetime)
                addtime(rgv.washtime)
                goodsnum += 1
                rgv.putone()
                cnclist[choice].goodtrans()
                cnclist[choice].changestatu()
            rgvmovement(0)


        def evenup(choice):
            global time
            global rgv
            global cnclist
            global goodsnum
            cnclist[choice].goodtrans()
            cnclist[choice].changestatu()
            if cnclist[choice].whethersec == False:
                sequence1.append(choice)
                uptsequence1.append(time)
                addtime(rgv.evenchangetime)
            elif cnclist[choice].whethersec == True and rgv.haveone == True:
                sequence2.append(choice)
                uptsequence2.append(time)
                rgv.putone()
                addtime(rgv.evenchangetime)
            else:
                addtime(1)



        def evendownup(choice):
            global time
            global rgv
            global cnclist
            global goodsnum
            cnclist[choice].goodtrans()
            if cnclist[choice].whethersec == False:
                sequence1.append(choice)
                uptsequence1.append(time)
                updowntsequence1.append(time)
                rgv.catchone()
                addtime(rgv.evenchangetime)
                cnclist[choice].goodtrans()
                cnclist[choice].changestatu()
            elif cnclist[choice].whethersec == True and rgv.haveone == False:
                sequence2.append(choice)
```

```
                updowntsequence2.append(time)
                addtime(rgv.evenchangetime)
                addtime(rgv.washtime)
                goodsnum += 1
                print("2 end when rgv do not has 1")
            elif cnclist[choice].whethersec == True and rgv.haveone ==
True:
                sequence2.append(choice)
                uptsequence2.append(time)
                updowntsequence2.append(time)
                addtime(rgv.evenchangetime)
                addtime(rgv.washtime)
                goodsnum += 1
                rgv.putone()
                cnclist[choice].goodtrans()
                cnclist[choice].changestatu()
            rgvmovement(0)


        def initialize():
            for j in range(0, len(fcnc)):
                choice = cnclist[fcnc[j]].num
                truediff = (choice - rgv.location)
                if choice % 2 == 0:
                    if truediff == 1:
                        rgvmovement(0)
                        evenup(choice)
                    elif truediff == 3:
                        rgvmovement(1)
        evenup(choice)
    elif truediff == 5:
        rgvmovement(2)
        evenup(choice)
    elif truediff == 7:
        rgvmovement(3)
        evenup(choice)
    elif truediff == -1:
        rgvmovement(-1)
        evenup(choice)
    elif truediff == -3:
        rgvmovement(-2)
        evenup(choice)
    elif truediff == -5:
        rgvmovement(-3)
        evenup(choice)
else:
    if truediff == 0:
        rgvmovement(0)
        oddup(choice)
    elif truediff == 2:
        rgvmovement(1)
        oddup(choice)
    elif truediff == 4:
        rgvmovement(2)
        oddup(choice)
    elif truediff == 6:
        rgvmovement(3)
        oddup(choice)
```

```
                elif truediff == -2:
                    rgvmovement(-1)
                    oddup(choice)
                elif truediff == -4:
                    rgvmovement(-2)
                    oddup(choice)
                elif truediff == -6:
                    rgvmovement(-3)
                    oddup(choice)


    def twoprooperation():
        global time
        global goodsnum
        global rgv
        global cnclist
        if not rgv.haveone:
            for j in range(1, len(fcnc)):
                relativeloca[fcnc[j]] = abs(cnclist[fcnc[j]].num -
rgv.location)
                if cnclist[fcnc[j]].having == True:
                    relativeloca[fcnc[j]] += 1000
                if cnclist[fcnc[j]].working == True:
                    relativeloca[fcnc[j]] += 1000
            if min(relativeloca) >= 2000:
                for m in range(1, len(scnc)):
                    relativeloca2[scnc[m]]                    =
abs(cnclist[scnc[m]].num - rgv.location)
                    if cnclist[scnc[m]].having == True:
                        relativeloca2[scnc[m]] += 1000
                    if cnclist[scnc[m]].working == True:
                        relativeloca2[scnc[m]] += 1000
                    if cnclist[scnc[m]].having == False and
cnclist[scnc[m]].working == False:
                        relativeloca2[scnc[m]] += 2000
                if min(relativeloca2) >= 2000:
                    addtime(1)
                elif min(relativeloca2) >= 1000:
                    choice =
relativeloca2.index(min(relativeloca2))
                    if relativeloca2[1] == 1006 and relativeloca2[2]
== 1005:
                        choice = 1
                    truediff = (choice - rgv.location)
                    if choice % 2 == 0:
                        if truediff == 1:
                            rgvmovement(0)
                            evendownup(choice)
                        elif truediff == 3:
                            rgvmovement(1)
                            evendownup(choice)
                        elif truediff == 5:
                            rgvmovement(2)
                            evendownup(choice)
                        elif truediff == 7:
                            rgvmovement(3)
                            evendownup(choice)
                        elif truediff == -1:
```

```
        rgvmovement(-1)
        evendownup(choice)
    elif truediff == -3:
        rgvmovement(-2)
        evendownup(choice)
    elif truediff == -5:
        rgvmovement(-3)
        evendownup(choice)
else:
    if truediff == 0:
        rgvmovement(0)
        odddownup(choice)
    elif truediff == 2:
        rgvmovement(1)
        odddownup(choice)
    elif truediff == 4:
        rgvmovement(2)
        odddownup(choice)
    elif truediff == 6:
        rgvmovement(3)
        odddownup(choice)
    elif truediff == -2:
        rgvmovement(-1)
        odddownup(choice)
    elif truediff == -4:
        rgvmovement(-2)
        odddownup(choice)
    elif truediff == -6:
        rgvmovement(-3)
```

```
                            odddownup(choice)
            elif min(relativeloca) >= 1000:
                choice = relativeloca.index(min(relativeloca))
                if relativeloca[1] == 1006 and relativeloca[2] ==
1005:
                    choice = 1
                truediff = (choice - rgv.location)
                if choice % 2 == 0:
                    if truediff == 1:
                        rgvmovement(0)
                        evendownup(choice)
                    elif truediff == 3:
                        rgvmovement(1)
                        evendownup(choice)
                    elif truediff == 5:
                        rgvmovement(2)
                        evendownup(choice)
                    elif truediff == 7:
                        rgvmovement(3)
                        evendownup(choice)
                    elif truediff == -1:
                        rgvmovement(-1)
                        evendownup(choice)
                    elif truediff == -3:
                        rgvmovement(-2)
                        evendownup(choice)
                    elif truediff == -5:
                        rgvmovement(-3)
                        evendownup(choice)
```

```
    else:
        if truediff == 0:
            rgvmovement(0)
            odddownup(choice)
        elif truediff == 2:
            rgvmovement(1)
            odddownup(choice)
        elif truediff == 4:
            rgvmovement(2)
            odddownup(choice)
        elif truediff == 6:
            rgvmovement(3)
            odddownup(choice)
        elif truediff == -2:
            rgvmovement(-1)
            odddownup(choice)
        elif truediff == -4:
            rgvmovement(-2)
            odddownup(choice)
        elif truediff == -6:
            rgvmovement(-3)
            odddownup(choice)
else:
    choice = relativeloca.index(min(relativeloca))
    truediff = (choice - rgv.location)
    if choice % 2 == 0:
        if truediff == 1:
            rgvmovement(0)
            evenup(choice)
    elif truediff == 3:
        rgvmovement(1)
        evenup(choice)
    elif truediff == 5:
        rgvmovement(2)
        evenup(choice)
    elif truediff == 7:
        rgvmovement(3)
        evenup(choice)
    elif truediff == -1:
        rgvmovement(-1)
        evenup(choice)
    elif truediff == -3:
        rgvmovement(-2)
        evenup(choice)
    elif truediff == -5:
        rgvmovement(-3)
        evenup(choice)
else:
    if truediff == 0:
        rgvmovement(0)
        oddup(choice)
    elif truediff == 2:
        rgvmovement(1)
        oddup(choice)
    elif truediff == 4:
        rgvmovement(2)
        oddup(choice)
    elif truediff == 6:
```

```
                        rgvmovement(3)
                        oddup(choice)
                    elif truediff == -2:
                        rgvmovement(-1)
                        oddup(choice)
                    elif truediff == -4:
                        rgvmovement(-2)
                        oddup(choice)
                    elif truediff == -6:
                        rgvmovement(-3)
                        oddup(choice)
        else:
            for j in range(1, len(scnc)):
                relativeloca[scnc[j]] = abs(cnclist[scnc[j]].num -
rgv.location)
                if cnclist[scnc[j]].having == True:
                    relativeloca[scnc[j]] += 1000
                if cnclist[scnc[j]].working == True:
                    relativeloca[scnc[j]] += 1000
            if min(relativeloca) >= 2000:
                addtime(1)
            elif min(relativeloca) >= 1000:
                choice = relativeloca.index(min(relativeloca))
                if relativeloca[1] == 1006 and relativeloca[2] ==
1005:
                    choice = 1
                truediff = (choice - rgv.location)
                if choice % 2 == 0:
                    if truediff == 1:
```

```
        rgvmovement(0)
        evendownup(choice)
    elif truediff == 3:
        rgvmovement(1)
        evendownup(choice)
    elif truediff == 5:
        rgvmovement(2)
        evendownup(choice)
    elif truediff == 7:
        rgvmovement(3)
        evendownup(choice)
    elif truediff == -1:
        rgvmovement(-1)
        evendownup(choice)
    elif truediff == -3:
        rgvmovement(-2)
        evendownup(choice)
    elif truediff == -5:
        rgvmovement(-3)
        evendownup(choice)
else:
    if truediff == 0:
        rgvmovement(0)
        odddownup(choice)
    elif truediff == 2:
        rgvmovement(1)
        odddownup(choice)
    elif truediff == 4:
        rgvmovement(2)
```

```
            odddownup(choice)
        elif truediff == 6:
            rgvmovement(3)
            odddownup(choice)
        elif truediff == -2:
            rgvmovement(-1)
            odddownup(choice)
        elif truediff == -4:
            rgvmovement(-2)
            odddownup(choice)
        elif truediff == -6:
            rgvmovement(-3)
            odddownup(choice)
else:
    choice = relativeloca.index(min(relativeloca))
    truediff = (choice - rgv.location)
    if choice % 2 == 0:
        if truediff == 1:
            rgvmovement(0)
            evenup(choice)
        elif truediff == 3:
            rgvmovement(1)
            evenup(choice)
        elif truediff == 5:
            rgvmovement(2)
            evenup(choice)
        elif truediff == 7:
            rgvmovement(3)
            evenup(choice)
    elif truediff == -1:
        rgvmovement(-1)
        evenup(choice)
    elif truediff == -3:
        rgvmovement(-2)
        evenup(choice)
    elif truediff == -5:
        rgvmovement(-3)
        evenup(choice)
else:
    if truediff == 0:
        rgvmovement(0)
        oddup(choice)
    elif truediff == 2:
        rgvmovement(1)
        oddup(choice)
    elif truediff == 4:
        rgvmovement(2)
        oddup(choice)
    elif truediff == 6:
        rgvmovement(3)
        oddup(choice)
    elif truediff == -2:
        rgvmovement(-1)
        oddup(choice)
    elif truediff == -4:
        rgvmovement(-2)
        oddup(choice)
    elif truediff == -6:
```

```
                        rgvmovement(-3)
                        oddup(choice)


    popu = Population()

    allthat = [0]

    while popu.generation <= 50:
        for every in (1, 15 + 1):
            time = 0
            goodsnum = 0
            relativeloca = [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000]
            relativeloca2 = [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000]
            rgv = RGV([0, 20, 33, 46], 28, 31, 25, 1)
            workingtime = [0, 400, 378]
            cnclist = []
            fcnc = [0]
            scnc = [0]
            sequenceall = [0]
            sequence1 = [0]
            sequence2 = [0]
            uptsequence1 = [0]
            uptsequence2 = [0]
            updowntsequence1 = [0]
            updowntsequence2 = [0]
            genome = popu.allchromo[every].copy()
            for i in range(0, 8 + 1):
                cnc = CNC(i, False, workingtime[1], workingtime[2])
                cnclist.append(cnc)
            express()
            while time <= (8 * 3600):
                twoprooperation()
                relativeloca = [10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000]
                relativeloca2 = [10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000]
            #print("%-10s" % "CNC1", "%-10s" % "Uptime1", "%-10s" %
"Downtime1",
            #     "%-10s" % "CNC2", "%-10s" % "Uptime2", "%-10s" %
"Downtime2")
            #for i in range(1, len(updowntsequence2)):
                #print("%-10d"    %    sequence1[i],    "%-10d"    %
uptsequence1[i], "%-10d" % updowntsequence1[i],
                 #          "%-10d"   %   sequence2[i],   "%-10d"   %
uptsequence2[i], "%-10d" % updowntsequence2[i])
            print("Total goods number: ", goodsnum)
            print(genome)
            popu.setfitness(every, goodsnum)
            allthat.append(goodsnum)
        popu.setchoice()
        popu.selecteva()
    print(max(allthat))
```

3. 生产一道工序的产品加入故障扰动(第一组)：

```
from CNC import CNC
from RGV import RGV
import random

time = 0
goodsnum = 0
relativeloca = [10000, 0, 0, 0, 0, 0, 0, 0, 0]
rgv = RGV([0, 20, 33, 46], 28, 31, 25, 1)
workingtime = 560
cnclist = []
sequence = [0]
uptsequence = [0]
updowntsequence = [0]


def addtime(secs):
    global time
    global rgv
    global cnclist
    for s in range(1, secs + 1):
        time += 1
        for k in range(1, 8 + 1):
            cnclist[k].addmytime(1)
            if (cnclist[k].working == True
                and cnclist[k].broken == False
                and cnclist[k].shouldbro == True
                and            cnclist[k].mytime            ==
cnclist[k].brokenmoment):
                    cnclist[k].brokenfix()
                    print("broken now", k, time)

    def rgvmovement(steps):
        global time
        global rgv
        global cnclist
        rgv.move(steps)
        addtime(rgv.movetime[abs(steps)])

    def whetherbroken(cncnum):
        global cnclist
        random.seed()
        ran = random.randint(1, 100)
        if ran == 1:
            cnclist[cncnum].shouldbro = True
            cnclist[cncnum].brokentime = random.randint(600, 1200)
            cnclist[cncnum].brokenmoment = random.randint(1,560)

    def oddup(choice):
        global time
        global rgv
        global cnclist
        global goodsnum
        cnclist[choice].goodtrans()
        cnclist[choice].changestatu()
        sequence.append(choice)
        uptsequence.append(time)
    addtime(rgv.oddchangetime)
    whetherbroken(choice)



def odddownup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    sequence.append(choice)
    uptsequence.append(time)
    updowntsequence.append(time)
    addtime(rgv.oddchangetime)
    whetherbroken(choice)
    addtime(rgv.washtime)
    goodsnum += 1
    rgvmovement(0)
    cnclist[choice].goodtrans()
    cnclist[choice].changestatu()

def evenup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    cnclist[choice].changestatu()
    sequence.append(choice)
    uptsequence.append(time)
    addtime(rgv.evenchangetime)
    whetherbroken(choice)

def evendownup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    sequence.append(choice)
    uptsequence.append(time)
    updowntsequence.append(time)
    addtime(rgv.evenchangetime)
    whetherbroken(choice)
    addtime(rgv.washtime)
    goodsnum += 1
    rgvmovement(0)
    cnclist[choice].goodtrans()
    cnclist[choice].changestatu()

def opration():
    global time
    global goodsnum
    global rgv
    global cnclist
    global useamount
    for j in range(1, 8 + 1):
```

```
    relativeloca[j] = abs(cnclist[j].num - rgv.location)
    if cnclist[j].broken == True:
        relativeloca[j] += 2000
    if cnclist[j].having == True:
        relativeloca[j] += 1000
    if cnclist[j].working == True:
        relativeloca[j] += 1000
if min(relativeloca) >= 2000:
    addtime(1)
elif min(relativeloca) >= 1000:
    choice = relativeloca.index(min(relativeloca))
    if relativeloca[1] == 1006 and relativeloca[2] == 1005:
        choice = 1
    truediff = (choice - rgv.location)
    if choice % 2 == 0:
        if truediff == 1:
            rgvmovement(0)
            evendownup(choice)
        elif truediff == 3:
            rgvmovement(1)
            evendownup(choice)
        elif truediff == 5:
            rgvmovement(2)
            evendownup(choice)
        elif truediff == 7:
            rgvmovement(3)
            evendownup(choice)
        elif truediff == -1:
            rgvmovement(-1)
        evendownup(choice)
    elif truediff == -3:
        rgvmovement(-2)
        evendownup(choice)
    elif truediff == -5:
        rgvmovement(-3)
        evendownup(choice)
else:
    if truediff == 0:
        rgvmovement(0)
        odddownup(choice)
    elif truediff == 2:
        rgvmovement(1)
        odddownup(choice)
    elif truediff == 4:
        rgvmovement(2)
        odddownup(choice)
    elif truediff == 6:
        rgvmovement(3)
        odddownup(choice)
    elif truediff == -2:
        rgvmovement(-1)
        odddownup(choice)
    elif truediff == -4:
        rgvmovement(-2)
        odddownup(choice)
    elif truediff == -6:
        rgvmovement(-3)
        odddownup(choice)
```

```
else:
    choice = relativeloca.index(min(relativeloca))
    truediff = (choice - rgv.location)
    if choice % 2 == 0:
        if truediff == 1:
            rgvmovement(0)
            evenup(choice)
        elif truediff == 3:
            rgvmovement(1)
            evenup(choice)
        elif truediff == 5:
            rgvmovement(2)
            evenup(choice)
        elif truediff == 7:
            rgvmovement(3)
            evenup(choice)
        elif truediff == -1:
            rgvmovement(-1)
            evenup(choice)
        elif truediff == -3:
            rgvmovement(-2)
            evenup(choice)
        elif truediff == -5:
            rgvmovement(-3)
            evenup(choice)
    else:
        if truediff == 0:
            rgvmovement(0)
            oddup(choice)
```

```
                elif truediff == 2:
                    rgvmovement(1)
                    oddup(choice)
                elif truediff == 4:
                    rgvmovement(2)
                    oddup(choice)
                elif truediff == 6:
                    rgvmovement(3)
                    oddup(choice)
                elif truediff == -2:
                    rgvmovement(-1)
                    oddup(choice)
                elif truediff == -4:
                    rgvmovement(-2)
                    oddup(choice)
                elif truediff == -6:
                    rgvmovement(-3)
                    oddup(choice)


    for i in range(0, 8 + 1):
        cnc = CNC(i, False, workingtime, 0)
        cnclist.append(cnc)
    while time <= (8*3600):
        opration()
    print(″%-8s″ % ″CNC″, ″%-8s″ % ″Uptime″, ″%-8s″ % ″Downtime″)
    for i in range(1, len(updowntsequence)):
        print(″%-8d″ % sequence[i], ″%-8d″ % uptsequence[i], ″%-
8d″ % updowntsequence[i])
```

```
        print("Total goods number: ", goodsnum)
```

4. 生产两道工序的产品加入故障扰动（第一组）：

```
from CNC import CNC
from RGV import RGV
import random

time = 0
goodsnum = 0
relativeloca = [10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000]
relativeloca2 = [10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000]
rgv = RGV([0, 20, 33, 46], 28, 31, 25, 1)
workingtime = [0, 400, 378]
cnclist = []
fcnc = [0]
scnc = [0]
sequenceall = [0]
sequence1 = [0]
sequence2 = [0]
uptsequence1 = [0]
uptsequence2 = [0]
updowntsequence1 = [0]
updowntsequence2 = [0]
genome = [-1, 0, 1, 0, 1, 0, 1, 0, 1]
```

```
def express():
    global cnclist
    global fcnc
    global scnc
    for n in range(1, 8 + 1):
        if genome[n] == 1:
            cnclist[n].whethersec = True
            scnc.append(cnclist[n].num)
        else:
            fcnc.append(cnclist[n].num)


def addtime(secs):
    global time
    global rgv
    global cnclist
    for s in range(1, secs + 1):
        time += 1
        for k in range(1, 8 + 1):
            cnclist[k].addmytime(1)
            if (cnclist[k].working == True
                and cnclist[k].broken == False
                and cnclist[k].shouldbro == True
                and cnclist[k].mytime == cnclist[k].brokenmoment):
                cnclist[k].brokenfix()
                print("broken now", k, time)
```

```
def rgvmovement(steps):
    global time
    global rgv
    global cnclist
    rgv.move(steps)
    addtime(rgv.movetime[abs(steps)])


def whetherbroken(cncnum):
    global cnclist
    random.seed()
    ran = random.randint(1, 100)
    if ran == 1:
        cnclist[cncnum].shouldbro = True
        cnclist[cncnum].brokentime = random.randint(600, 1200)
        cnclist[cncnum].brokenmoment = random.randint(1,560)


def oddup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    cnclist[choice].changestatu()
    if cnclist[choice].whethersec == False:
        sequence1.append(choice)
        uptsequence1.append(time)
        addtime(rgv.oddchangetime)
        whetherbroken(choice)
    elif cnclist[choice].whethersec == True and rgv.haveone == True:
        sequence2.append(choice)
        uptsequence2.append(time)
        rgv.putone()
        addtime(rgv.oddchangetime)
        whetherbroken(choice)
    else:
        addtime(1)


def odddownup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    if cnclist[choice].whethersec == False:
        sequence1.append(choice)
        uptsequence1.append(time)
        updowntsequence1.append(time)
        rgv.catchone()
        addtime(rgv.oddchangetime)
        whetherbroken(choice)
        cnclist[choice].goodtrans()
        cnclist[choice].changestatu()
    elif cnclist[choice].whethersec == True and rgv.haveone == False:
        sequence2.append(choice)
        updowntsequence2.append(time)
        addtime(rgv.oddchangetime)
        addtime(rgv.washtime)
        goodsnum += 1
        print("2 end when rgv do not has 1")
    elif cnclist[choice].whethersec == True and rgv.haveone == True:
        sequence2.append(choice)
        uptsequence2.append(time)
        updowntsequence2.append(time)
        addtime(rgv.oddchangetime)
        whetherbroken(choice)
        addtime(rgv.washtime)
        goodsnum += 1
        rgv.putone()
        cnclist[choice].goodtrans()
        cnclist[choice].changestatu()
    rgvmovement(0)


def evenup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    cnclist[choice].changestatu()
    if cnclist[choice].whethersec == False:
        sequence1.append(choice)
        uptsequence1.append(time)
        addtime(rgv.evenchangetime)
        whetherbroken(choice)
    elif cnclist[choice].whethersec == True and rgv.haveone == True:
        sequence2.append(choice)
        uptsequence2.append(time)
        rgv.putone()
        addtime(rgv.evenchangetime)
        whetherbroken(choice)
    else:
        addtime(1)



def evendownup(choice):
    global time
    global rgv
    global cnclist
    global goodsnum
    cnclist[choice].goodtrans()
    if cnclist[choice].whethersec == False:
        sequence1.append(choice)
        uptsequence1.append(time)
        updowntsequence1.append(time)
        rgv.catchone()
        addtime(rgv.evenchangetime)
        whetherbroken(choice)
        cnclist[choice].goodtrans()
        cnclist[choice].changestatu()
    if cnclist[choice].whethersec == True and rgv.haveone == False:
        sequence2.append(choice)
        updowntsequence2.append(time)
```

```
            addtime(rgv.evenchangetime)
            addtime(rgv.washtime)
            goodsnum += 1
            print("2 end when rgv do not has 1")
        if cnclist[choice].whethersec == True and rgv.haveone == True:
            sequence2.append(choice)
            uptsequence2.append(time)
            updowntsequence2.append(time)
            addtime(rgv.evenchangetime)
            whetherbroken(choice)
            addtime(rgv.washtime)
            goodsnum += 1
            rgv.putone()
            cnclist[choice].goodtrans()
            cnclist[choice].changestatu()
        rgvmovement(0)


    def initialize():
        for j in range(0, len(fcnc)):
            choice = cnclist[fcnc[j]].num
            truediff = (choice - rgv.location)
            if choice % 2 == 0:
                if truediff == 1:
                    rgvmovement(0)
                    evenup(choice)
                elif truediff == 3:
                    rgvmovement(1)
                    evenup(choice)
                elif truediff == 5:
                    rgvmovement(2)
                    evenup(choice)
                elif truediff == 7:
                    rgvmovement(3)
                    evenup(choice)
                elif truediff == -1:
                    rgvmovement(-1)
                    evenup(choice)
                elif truediff == -3:
                    rgvmovement(-2)
                    evenup(choice)
                elif truediff == -5:
                    rgvmovement(-3)
                    evenup(choice)
            else:
                if truediff == 0:
                    rgvmovement(0)
                    oddup(choice)
                elif truediff == 2:
                    rgvmovement(1)
                    oddup(choice)
                elif truediff == 4:
                    rgvmovement(2)
                    oddup(choice)
                elif truediff == 6:
                    rgvmovement(3)
                    oddup(choice)
                elif truediff == -2:
                    rgvmovement(-1)
                    oddup(choice)
                elif truediff == -4:
                    rgvmovement(-2)
                    oddup(choice)
                elif truediff == -6:
                    rgvmovement(-3)
                    oddup(choice)


    def twoprooperation():
        global time
        global goodsnum
        global rgv
        global cnclist
        if not rgv.haveone:
            for j in range(1, len(fcnc)):
                relativeloca[fcnc[j]]  =  abs(cnclist[fcnc[j]].num  -
    rgv.location)
                if cnclist[fcnc[j]].having == True:
                    relativeloca[fcnc[j]] += 1000
                if cnclist[fcnc[j]].working == True:
                    relativeloca[fcnc[j]] += 1000
                if cnclist[fcnc[j]].broken == True:
                    relativeloca[fcnc[j]] += 2000
            if min(relativeloca) >= 2000:
                for m in range(1, len(scnc)):
                    relativeloca2[scnc[m]] = abs(cnclist[scnc[m]].num -
    rgv.location)
```

```
                if cnclist[scnc[m]].having == True:
                    relativeloca2[scnc[m]] += 1000
                if cnclist[scnc[m]].working == True:
                    relativeloca2[scnc[m]] += 1000
                if cnclist[scnc[m]].broken == True:
                    relativeloca[fcnc[m]] += 2000
                if cnclist[scnc[m]].having == False and cnclist[scnc[m]].working == False:
                    relativeloca2[scnc[m]] += 2000
            if min(relativeloca2) >= 2000:
                addtime(1)
            elif min(relativeloca2) >= 1000:
                choice = relativeloca2.index(min(relativeloca2))
                if relativeloca2[1] == 1006 and relativeloca2[2] == 1005:
                    choice = 1
                truediff = (choice - rgv.location)
                if choice % 2 == 0:
                    if truediff == 1:
                        rgvmovement(0)
                        evendownup(choice)
                    elif truediff == 3:
                        rgvmovement(1)
                        evendownup(choice)
                    elif truediff == 5:
                        rgvmovement(2)
                        evendownup(choice)
                    elif truediff == 7:
                        rgvmovement(3)
```

```
        evendownup(choice)
    elif truediff == -1:
        rgvmovement(-1)
        evendownup(choice)
    elif truediff == -3:
        rgvmovement(-2)
        evendownup(choice)
    elif truediff == -5:
        rgvmovement(-3)
        evendownup(choice)
else:
    if truediff == 0:
        rgvmovement(0)
        odddownup(choice)
    elif truediff == 2:
        rgvmovement(1)
        odddownup(choice)
    elif truediff == 4:
        rgvmovement(2)
        odddownup(choice)
    elif truediff == 6:
        rgvmovement(3)
        odddownup(choice)
    elif truediff == -2:
        rgvmovement(-1)
        odddownup(choice)
    elif truediff == -4:
        rgvmovement(-2)
        odddownup(choice)
```

```
            elif truediff == -6:
                rgvmovement(-3)
                odddownup(choice)
elif min(relativeloca) >= 1000:
    choice = relativeloca.index(min(relativeloca))
    if relativeloca[1] == 1006 and relativeloca[2] == 1005:
        choice = 1
    truediff = (choice - rgv.location)
    if choice % 2 == 0:
        if truediff == 1:
            rgvmovement(0)
            evendownup(choice)
        elif truediff == 3:
            rgvmovement(1)
            evendownup(choice)
        elif truediff == 5:
            rgvmovement(2)
            evendownup(choice)
        elif truediff == 7:
            rgvmovement(3)
            evendownup(choice)
        elif truediff == -1:
            rgvmovement(-1)
            evendownup(choice)
        elif truediff == -3:
            rgvmovement(-2)
            evendownup(choice)
        elif truediff == -5:
            rgvmovement(-3)
            evendownup(choice)
    else:
        if truediff == 0:
            rgvmovement(0)
            odddownup(choice)
        elif truediff == 2:
            rgvmovement(1)
            odddownup(choice)
        elif truediff == 4:
            rgvmovement(2)
            odddownup(choice)
        elif truediff == 6:
            rgvmovement(3)
            odddownup(choice)
        elif truediff == -2:
            rgvmovement(-1)
            odddownup(choice)
        elif truediff == -4:
            rgvmovement(-2)
            odddownup(choice)
        elif truediff == -6:
            rgvmovement(-3)
            odddownup(choice)
else:
    choice = relativeloca.index(min(relativeloca))
    truediff = (choice - rgv.location)
    if choice % 2 == 0:
        if truediff == 1:
            rgvmovement(0)
            evenup(choice)
        elif truediff == 3:
            rgvmovement(1)
            evenup(choice)
        elif truediff == 5:
            rgvmovement(2)
            evenup(choice)
        elif truediff == 7:
            rgvmovement(3)
            evenup(choice)
        elif truediff == -1:
            rgvmovement(-1)
            evenup(choice)
        elif truediff == -3:
            rgvmovement(-2)
            evenup(choice)
        elif truediff == -5:
            rgvmovement(-3)
            evenup(choice)
    else:
        if truediff == 0:
            rgvmovement(0)
            oddup(choice)
        elif truediff == 2:
            rgvmovement(1)
            oddup(choice)
        elif truediff == 4:
            rgvmovement(2)
            oddup(choice)
```

```
                    elif truediff == 6:
                        rgvmovement(3)
                        oddup(choice)
                    elif truediff == -2:
                        rgvmovement(-1)
                        oddup(choice)
                    elif truediff == -4:
                        rgvmovement(-2)
                        oddup(choice)
                    elif truediff == -6:
                        rgvmovement(-3)
                        oddup(choice)
    else:
        for j in range(1, len(scnc)):
            relativeloca[scnc[j]]   =   abs(cnclist[scnc[j]].num   -
rgv.location)
            if cnclist[scnc[j]].broken == True:
                relativeloca[scnc[j]] += 2000
            if cnclist[scnc[j]].having == True:
                relativeloca[scnc[j]] += 1000
            if cnclist[scnc[j]].working == True:
                relativeloca[scnc[j]] += 1000
        if min(relativeloca) >= 2000:
            addtime(1)
        elif min(relativeloca) >= 1000:
            choice = relativeloca.index(min(relativeloca))
            if relativeloca[1] == 1006 and relativeloca[2] == 1005:
                choice = 1
            truediff = (choice - rgv.location)
```

```
if choice % 2 == 0:
    if truediff == 1:
        rgvmovement(0)
        evendownup(choice)
    elif truediff == 3:
        rgvmovement(1)
        evendownup(choice)
    elif truediff == 5:
        rgvmovement(2)
        evendownup(choice)
    elif truediff == 7:
        rgvmovement(3)
        evendownup(choice)
    elif truediff == -1:
        rgvmovement(-1)
        evendownup(choice)
    elif truediff == -3:
        rgvmovement(-2)
        evendownup(choice)
    elif truediff == -5:
        rgvmovement(-3)
        evendownup(choice)
else:
    if truediff == 0:
        rgvmovement(0)
        odddownup(choice)
    elif truediff == 2:
        rgvmovement(1)
        odddownup(choice)
        elif truediff == 4:
            rgvmovement(2)
            odddownup(choice)
        elif truediff == 6:
            rgvmovement(3)
            odddownup(choice)
        elif truediff == -2:
            rgvmovement(-1)
            odddownup(choice)
        elif truediff == -4:
            rgvmovement(-2)
            odddownup(choice)
        elif truediff == -6:
            rgvmovement(-3)
            odddownup(choice)
else:
    choice = relativeloca.index(min(relativeloca))
    truediff = (choice - rgv.location)
    if choice % 2 == 0:
        if truediff == 1:
            rgvmovement(0)
            evenup(choice)
        elif truediff == 3:
            rgvmovement(1)
            evenup(choice)
        elif truediff == 5:
            rgvmovement(2)
            evenup(choice)
        elif truediff == 7:
            rgvmovement(3)
            evenup(choice)
        elif truediff == -1:
            rgvmovement(-1)
            evenup(choice)
        elif truediff == -3:
            rgvmovement(-2)
            evenup(choice)
        elif truediff == -5:
            rgvmovement(-3)
            evenup(choice)
    else:
        if truediff == 0:
            rgvmovement(0)
            oddup(choice)
        elif truediff == 2:
            rgvmovement(1)
            oddup(choice)
        elif truediff == 4:
            rgvmovement(2)
            oddup(choice)
        elif truediff == 6:
            rgvmovement(3)
            oddup(choice)
        elif truediff == -2:
            rgvmovement(-1)
            oddup(choice)
        elif truediff == -4:
            rgvmovement(-2)
```

```
                oddup(choice)
            elif truediff == -6:
                rgvmovement(-3)
                oddup(choice)




for i in range(0, 8 + 1):
    cnc = CNC(i, False, workingtime[1], workingtime[2])
    cnclist.append(cnc)
express()
while time <= (8 * 3600):
    twoprooperation()
    relativeloca = [10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000]
    relativeloca2 = [10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000]
print("%-10s" % "CNC1", "%-10s" % "Uptime1", "%-10s" % "Downtime1",
      "%-10s" % "CNC2", "%-10s" % "Uptime2", "%-10s" % "Downtime2")
for i in range(1, len(updowntsequence2)):
    print("%-10d" % sequence1[i], "%-10d" % uptsequence1[i], "%-10d" %
updowntsequence1[i],
          "%-10d" % sequence2[i], "%-10d" % uptsequence2[i], "%-10d" %
updowntsequence2[i])
print("Total goods number: ", goodsnum)
print(genome)
```

5. RGV.py：

```
class RGV:
 haveone = False

 def __init__(self, movetime, oddchangetime, evenchangetime, washtime, location):
     self.movetime = movetime.copy()
     self.oddchangetime = oddchangetime
     self.evenchangetime = evenchangetime
     self.washtime = washtime
     self.location = location

 def move(self, step):
     self.location += 2*step

 def catchone(self):
     if not self.haveone:
         self.haveone = True
     else:
         print("Error in catching")

 def putone(self):
     if self.haveone:
         self.haveone = False
     else:
         print("Error in putting")

 def rgvprint(self):
     print(self.location, self.haveone)
```

6. CNC.py：

```
class CNC:
working = False
having = False
broken = False
shouldbro = False
whethersec = False
mytime = 0
brokentime = 0
brokenmoment = 0

def __init__(self, num, doubleprocess, oneprotime, twoprotime):
    self.num = num
    self.doubleprocess = doubleprocess
    if doubleprocess:
        self.oneprotime = oneprotime
        self.twoprotime = 0
    else:
        self.oneprotime = oneprotime
        self.twoprotime = twoprotime

def changestatu(self):
    if not self.working:
        self.working = True
        self.mytime = 0
    else:
        self.working = False
```

```
def goodtrans(self):
    if not self.having:
        self.having = True
    else:
        self.having = False

def brokenfix(self):
    if not self.broken:
        self.broken = True
        self.mytime = 0
        self.changestatu()
        self.having = False
        self.shouldbro = False
    else:
        self.broken = False
        self.working = False
        self.having = False

def addmytime(self, secs):
    if self.working:
        self.mytime += secs
    if self.broken:
        self.mytime += secs
    if not self.whethersec:
        if self.mytime >= self.oneprotime and self.working:
            self.changestatu()
    else:
        if self.mytime >= self.twoprotime and self.working:
            self.changestatu()
        if self.mytime >= self.brokentime and self.broken:
            self.brokenfix()
            print("broken fixed", self.mytime)

    def becomesec(self):
        self.whethersec = True

    def rebuild(self):
        self.working = False
        self.having = False
        self.broken = False
        self.shouldbro = False
        self.whethersec = False
        self.mytime = 0
        self.brokentime = 0
        self.brokenmoment = 0

    def cncprint(self):
        print(self.num,  self.working,  self.having,  self.oneprotime,
self.twoprotime)
```

7. POPULATION.py：

```
import random
import operator


class Population:
    generation = 0
```

```
        allchromo = [[-1],
                     [-1], [-1], [-1], [-1], [-1], [-1], [-1], [-
1],
                     [-1], [-1], [-1], [-1], [-1], [-1], [-1], [-
1]]

        fitness = [-1,
                   -1, -1, -1, -1, -1, -1, -1, -1,
                   -1, -1, -1, -1, -1, -1, -1, -1]

        chosenposibilty = [ 0,
                           -1, -1, -1, -1, -1, -1, -1, -1,
                           -1, -1, -1, -1, -1, -1, -1, -1]

        def __init__(self):
            for i in range(1, 16 + 1):
                for j in range(1, 8 + 1):
                    self.allchromo[i].append(random.randint(0, 1))

        def setfitness(self, num, fitamount):
            self.fitness[num] = fitamount

        def setchoice(self):
            totalfits = 0
            for i in range(1, 16 + 1):
                totalfits += self.fitness[i]
            for j in range(1, 16 + 1):
                self.chosenposibilty[j]        =        10        *
round(self.fitness[j] / totalfits, 1) + self.chosenposibilty[j - 1]
```

```
        def selecteva(self):
            copyallchro = self.allchromo.copy()
            for i in range(1, 16 + 1):
                ran = random.randint(1, max(self.chosenposibilty))
                for j in range(1, 16 + 1):
                    if self.chosenposibilty[j] >= ran:
                        self.allchromo[i] = copyallchro[j]
                        break
            for k in range(1, 15 + 1):
                if operator.eq(self.allchromo[k],
self.allchromo[k+1]):
                    k += 1
                    continue
                elif random.randint(1, 10) <= 9:
                    ran2 = random.randint(1, 8)
                    trans = self.allchromo[k][:ran2]
                    self.allchromo[k][:ran2] = self.allchromo[k +
1][:ran2]
                    self.allchromo[k + 1][:ran2] = trans
                    k += 1
                else:
                    k += 1
            for a in range(1, 16 + 1):
                if random.randint(1, 10) == 10:
                    ran3 = random.randint(1, 8)
                    if self.allchromo[a][ran3] == 1:
                        self.allchromo[a][ran3] = 0
                    else:
```

```
                self.allchromo[a][ran3] = 1
        self.generation += 1
```