基于简单优先规则的动态调度模型及其仿真模拟

摘要

随着自动化生产的普及，智能加工系统被广泛地应用于车间生产。一个智能加工系统的生产是否高效，关键就在于自动引导车 RGV 的动态调度模式。本文基于题目给定的智能加工系统模型与生产状况，力求寻找最优的 RGV 动态调度模型及相应算法，并通过给定参数表进行仿真模拟。

针对问题一，我们要在给定车间模型下，建立能适用于包含一道、两道工序的 RGV 调度模型，并考虑 CNC 加工机器发生故障时模型如何调整。为了建立调度模型，我们首先需要确定调度规则。经过比较，我们选择了能简洁、高效地处理车间调度问题的动态规则、简单优先规则作为我们的调度规则。鉴于题述各生产状况下，RGV 的操作流程有较多共通性，我们先以只包含一步工序的情形为基础，建立判断调度优先级别的总完成时间指标T。指标T的建立考虑到了 RGV 与某 CNC 之间移动时长t_{move}、移动后对于上一作业的等待时间t_{wait}、RGV 上下料时间t_{load}及熟料清洗时间t_{wash}。这一指标量概括了一套完整的 RGV 操作循环，完成上述步骤之后 RGV 将进入下一个新的循环。RGV 有移动、上下料、清洗、停止等待四种状态，且各状态之间无交集。只有在 RGV 处于停止等待状态时，操作系统需要为其下达调度指令。根据我们的调度规则，进入停止等待状态的 RGV 将调度至最优级 CNC（即拥有最小T值的 CNC）。对于包含两道工序的生产情况，我们依照经过第一次加工得到的半熟料是放入上料传送带还是直接放在第二工序的 CNC 台上创建了两套 T 值公式。两种操作方法各有优劣，可按照仿真结果比较效率。最后，考虑 CNC 故障率时，我们用 RGV 运行到 CNC 前剩余的故障排除时间替换T值基本算式中的上一作业等待时间，创建了适用于包容机器故障的 RGV 调度规则。

针对问题二，为了检验调度模型，我们建立了基于时间的仿真模型。我们将整个过程分为若干个时间步长Δt（$\Delta t = 1s$），在每个时间步Δt开始时更新 CNC 和 RGV 的状态，根据建立的调度模型进行决策。我们可以得到整个过程的调度策略和所有物料的信息，根据 CNC 的实际加工时间计算作业效率η。为了模拟第三种情况中 CNC 的故障，我们经过计算，在每一个Δt中用蒙特卡洛方法模拟故障，使发生故障的概率约等于 1%。经 100 亿秒的模拟可知这种方法使可行的。第二种情况中，在题目未给如何传送半成品的情况下，我们对用 RGV 的机械臂传送和用传送带传送两种方式进行比较，发现前者效率更高。我们对 CNC 刀具的安装方式进行了枚举，找到了最优安装方式。最后，我们求出了三组过程的调度策略及物料信息，计算出作业效率η，并用 Matplotlib 画出了它们的甘特图。

我们的模型具有作业效率高、环境适应能力强、耗费时间短等优点，可以在各种环境下使用。

关键词：动态调度、简单优先规则、车间调度模型、仿真模型、蒙特卡罗方法

§1 问题重述

1.1 情况说明

车间模型中，有一个主要由 8 台计算机数控机床（CNC）、1 辆轨道式自动引导车、1 条 RGV 直线轨道、1 条上料传送带、1 条下料传送带组成的智能加工系统。这种系统可以控制物料储运、设备加工过程从而提高生产效益。其中，RGV 是一种无人驾驶、能在固定轨道上自由运行的智能车，可根据指令能自动控制移动、完成上下料及清洗物料等作业任务。具体操作流程可简述为：RGV 移动至一台空闲 CNC 台前，利用机械臂从上料传送带上抓取生料，用生料替换台上的熟料，熟料在 RGV 台上完成清洗成成料，由机械臂下料至下料传送带运出。

1.2 问题的相关信息

根据题目提供的相关信息，可知如下条件：

附件 1 结合实物图，具体说明了智能加工系统的组成与作业流程；

附件 2 提供了 4 张未完成的 Excel 表，要求利用表 1 中系统作业参数的 3 组数据分别检验所建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。

1.3 需解决的问题

问题一：对智能加工系统一道工序，两道工序以及机器发生故障的物料加工作业情况进行研究，建立 RGV 动态调度模型并给出相应的求解算法；

问题二：利用表 1 中系统作业参数的 3 组数据，分别检验题一所建 RGV 动态调度模型的实用性和算法的有效性，给出 RGV 的具体调度策略和系统的作业效率，并将结果呈现于附件 2 的 EXCEL 表中。

1.4 引言

车间作业调度问题（JSP）自提出以来，就一直是众多企业关注的核心。在生产过程中，考虑到各种不确定因素的发生（如机器故障，加急单等），企业需要通过一个科学的模型，合理分配机器，确定其处理的工件以及加工开始时间，实现资源的最优化配置[1]。作为典型的 NP 困难问题，已经有诸多学者提出相关模型[2]，主要分为两类：最优化方法和近似/启发式方法[3,4]。本题其实就是一个简化的人工智能车间调度问题，从最基础的一道工序入手，扩展到两道工序，需要合理分配 CNC 的数量，确定其最佳位置，实现目

标的最优化，也即在确定的时间内生产尽量多的成料。为此，我们参考了大量文献，研究了常见的模型，如遗传算法、模拟退火、贪婪算法等。在我们能够掌握的范围内，提出了相应的算法。

§2 问题分析

我们的问题主要是确定 RGV 动态调度模型，并用 3 组数据检验模型的实用性和算法的有效性。

为了建立调度模型，我们需要确定调度规则。由题中所给信息可知，移动所需时长随 RGV 的位置变化，且调度时需要利用 CNC 的状态信息（如机器故障），因此我们需要使用动态规则。这里，我们可以应用简单优先规则为调度规则，基于 RGV 完成一个完整的作业流程所需的时间给出优先级别，设计调度规则，从而建立动态调度模型。为了充分利用 CNC 机器，我们可以考虑到即将到达的作业[5]，对于到达时间较短的作业，RGV 可以优先完成。

为了检验调度模型，我们需要用仿真方法进行测试。我们将整个过程分为若干个时间步长，根据建立的调度模型，模拟每个步长中发生的事件，从而得到整个过程的调度策略，根据 CNC 的实际加工时间计算出工作效率。对于两道工序的情况中 CNC 刀具的安装，我们可以针对每一组枚举出 256 种刀具安装策略，找到最优安装策略。前两种情况为稳定系统，而最后一种情况存在瞬变现象，针对这种情况，我们在每一个时间步长中用蒙特卡洛方法[6]模拟机器的损坏，使发生故障的概率约等于 1%，从而检验我们的模型是否能应对环境的变化。

§3 模型假设

1、 各工件（生料、半熟料、熟料）具备相同的优先级；
2、 所有工件在零时刻都可以被加工；
3、 各工件所有工序间的先后顺序已知；
4、 熟料在 RGV 清洗槽中清洗的时间远小于机械臂将成料下放至下料传送带的时间；；
5、 一台 CNC 机器只负责加工一道工序，且各工序一旦开始加工就不能中断；
6、 各工序具有多台可选加工机器，且在各台可选机器上的理论加工时间已知；
7、 除第三种情况 CNC 可能以 1%概率发生故障，系统内一切设备都按照给定参数无差错运行；
8、 CNC 机器的故障维修时间服从 10 到 20 上的均匀分布，故障排除后 CNC 可立即加入

加工序列；

9、 传送带速度很快；

10、 RGV 在工作过程中始终正常工作；

11、 RGV 在不同位点移动单位距离的时间、每个 CNC 加工时长、RGV 为各 CNC 上下料时间均固定非随机的；

12、 发生故障的 CNC 在故障排除之后能够立即使用。

§4 符号说明

4.1 符号说明

符号	说明
t_{move}	RGV 在某位点向某一方向移动单位距离的时间
$t_{process}$	每台 CNC 进行某一工序加工的时长
t_{load}	RGV 为 CNC 完成上下料的时间
t_{wash}	清洗时间（含取成料、洗熟料、下成料的过程）
t_{left}	RGV 到达 CVC 之后仍需等待 CVC 对前料的加工时间
t_{fix}	CNC 出现故障后的维修时长
T	RGV 关于某个 CNC 的用于调度决策的总时长指标
Δt	时间步长
n_s	物料成品的数量
t_{total}	总模拟时长
n_{CNC}	CNC 机器的数量
η	作业效率
λ	发生故障的概率
$t_{fix_{total}}$	CNC 进行故障排除的总时间
r	Python 生成的随机数
p	定值，即每个时间步内发生故障的概率
n_{fix}	故障次数

4.2 名词解释

1、机器故障率：机器故障停机时间与机器预期负荷时间的比值；
2、半熟料：生料在进行第一道工序之后，第二道工序之前的状态称为半熟料。

§5 模型的建立与求解

5.1 对问题一的分析与求解

5.1.1 调度规则的选取与单工序作业 RGV 动态调度模型

先考虑只包含一道工序的加工作业过程。当设定 RGV 在不同位点移动单位距离的时间（t_{move}）、每个 CNC 加工时长（$t_{process}$）、RGV 为各 CNC 上下料的时间（t_{load}）及清洗时间（t_{wash}）为一固定值时，我们在调度决策中面临的所有时间均是非随机的。每台 CNC 机器在加工完成后都会处于停止等待状态，需要 RGV 尽早为其完成下熟料、上生料的操作才能够恢复工作状态。在我们的车间模型中，总共有 8 台 CNC，相互独立无差错运行，并仅依靠一台 RGV 为其上下料。RGV 在上下料工作之外，还需要承担熟料的清洗与成料的下料输出工作。RGV 的上下料、清洗、移动、等待均不可同时进行。因此，为了最大化发挥 CNC 机器的利用率、提高加工车间总体效率，我们需要考虑最适宜的车间调度策略[7]。

寻找车间调度策略，首先要确定调度规则。所谓调度规则，亦称优先分派规则，是能够决定如何将工件分配给机器、实现实际生产中车间作业调度的一或多种启发式规则的组合[8]。其具体作用为：每当有机器出现闲置，调度规则便会为等待中的工件分派各自的优先值。这一优先值可基于车间、机器、工件的特征而产生，并被用于为机器安排具有最高优先值的工件进行加工作业的过程。随着研究的深入，我们已经可以选择具有不同优先属性、结构特征的调度规则来配合实际生产要求。比如，根据规则与时间之间的关系，调度规则有动态静态之分；根据规则的结构特征，调度规则又有简单优先规则、组合规则、加权规则和启发式规则的分别[8-10]。

结合本题车间模型的特征，我们认为动态规则、简单优先规则是能够有效而简洁的解决车间调度策略的调度规则。动态规则是与动态的工作环境相对应的，工作优先值会随着时间变化[5]，如常见的 WINQ 规则[11]。本题中，考虑到机器故障等随机事件的发生，整个工作环境是动态的，CNC 的选择需要根据具体情况作出相应调整，从而达到最佳。

简单优先规则，一般情况下只包含一个车间系统参数，比如基于加工工时的 SPT 规则，基于交货期的 EDD 规则，以及其他基于工序数、到达时间、松弛时间等参数的规则。本题中，为了尽可能地提高工件加工效率，我们依据 SPT 规则，提出以下 RGV 动态调度模型。

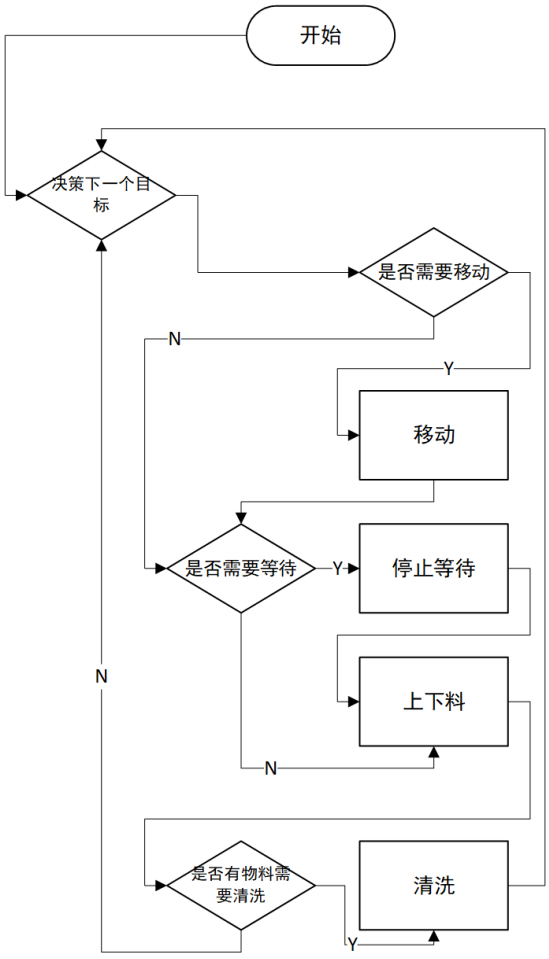

图 1 RGV 工作流程图

已知 RGV 有四个状态：移动状态、上下料状态、清洗状态、停止等待状态。当 RGV 转换至停止等待状态时，我们需要判断处于当下位点的 RGV 关于每个 CNC 的 T 的相对大小。指标 T 的定义式如下：

$$T = t_{move} + t_{left} + t_{load} + t_{wash} \tag{1}$$

其中，对于一个特定 CNC，t_{move}为从 RGV 当前位点移动到该 CNC 的时间，t_{left}为 RGV 到达 CNC 之后仍需等待 CNC 对前料的加工时间，t_{load}和t_{wash}分别为 RGV 上下料时间与熟料清洗时间。四个量均为非负值。

这个 T 值便是我们简单优化原则的优先级判断指标，T 越小代表 RGV 选择此 CNC 工作所带来系统的效率越高，因此该 CNC 所值得的优先级越高。通过比较 8 台 CNC 对应的 T 值，选取拥有最小 T 值的 CNC 作为最高优先级拥有者，并令 RGV 向此机器移动。

当 RGV 完成对新 CNC 的移动、等待上料（可不存在）、上下料、清洗操作后，又会

进入停止等待状态，如此重复上述决策，完成整个动态调度过程。

5.1.2 双工序作业 RGV 动态调度模型

参考题目给出的智能加工系统参数，在单工序作业系统各种，“RGV 为 CNC 上下料的时间”包含了 RGV 从上料传送带上取料的时间。这启示我们，对于两道工序的生产而言，进行完第一道工序的半熟料可以放置在上料传送带上，和生料一样在上料之前被抓取。除此之外，我们还考虑由机械臂抓取半熟料之后直接放入负责第二道工序的 CNC 上加工为熟料。参照这两种思路的 RGV 调度模型将分别被讨论。并且下一部分的仿真模拟将给出模型效率的比较。

5.1.2.1 假设半熟料需放置在上料传送带上

当我们考虑两道工序的车间作业时，若假设经过第一道工序加工完成的半熟料均被放置在上料传送带上，RGV 的动态调度模型与只考虑一道工序的情况是相同的。即 T 值仍遵循：

$$T = t_{move} + t_{left} + t_{load} + t_{wash} \tag{2}$$

当 RGV 进入停止等待状态时，无论它下一步移动至进行第一道工序还是第二道工序加工的 CNC 机器，其操作流程均遵循移动、等待上料（可不存在）、上下料、清洗的过程。CNC 具体进行第几道工序的加工对于模型设置并无影响。故上述 RGV 动态调度模型仍成立。

在对模型进行仿真决策时，由于 CNC 刀具无法更换，我们需要考虑到分别负责两道工序的 CNC 机器排布问题。两种 CNC 的数量与位置将影响 RGV 的调度与生产效率。比如，两道工序加工时间不同，故要合理配置两类机器数量使得两道工序顺畅高效的衔接。再者，考虑到只有两道工序配合衔接紧密才不会出现某一种机器的大量闲置，两类机器的分布差异所导致的 RGV 在其间移动时间的差异也要考虑在内。

5.1.2.2 假设机械臂抓取半熟料之后直接放入负责第二道工序的 CNC

这一情形下，RGV 的工作流程与调度决策将会有所不同。当 RGV 从负责第一道工序的 CNC 上抓取半熟料后，没有将其下放至上料传送带的时间，而直接计算 8 台 CNC 中负责第二道工序的 CNC 的 T 值，确定优先级最高的 CNC 并向其移动将半熟料上至该 CNC。

具体的，对于负责第一道工序是 CNC，RGV 的决策指标 T 为：

$$T = t_{move} + t_{left} + t_{load} \quad (3)$$

对于负责第二道工序是 CNC，RGV 的决策指标 T 为：

$$T = t_{move} + t_{left} + t_{load} + t_{wash} \quad (4)$$

这两种 CNC 的决策指标 T 虽然组成不同，但是鉴于假设的清洗时间（主要为下料时间）固定，相当于加上一个常数。所以对于具体决策结果不会有影响。

5.1.2.3 两种双工序操作模式的对比

第二种模式相对于第一种节省了对于半熟料的清洗时间（主要为下料至上料传送带的时间），但是也增加了调度决策的限制。具体表现为，当 RGV 的机械手上持有半熟料时，他的决策对象只有负责第二道工序的 CNC。因此，两种模式各有优劣，具体哪一种在实际生产中的效率更高，还需要仿真模拟来实现。

5.1.3 考虑 CNC 故障率的 RGV 动态调度模型

最后我们抛弃各个 CNC 均无差错运转的前提，允许 CNC 发生故障。这时候我们对于 RGV 动态调度模型进行微调。即将除去 RGV 移动事件后仍剩余的 CNC 故障维修时间 t_{fix}（随机变量，服从 10 到 20 上的均匀分布）替代t_{left}[12]。

$$T = t_{move} + t_{fix} + t_{load} + t_{wash} \quad (5)$$

因为根据假设，发生故障的 CNC 在维修结束后可以立即投入生产，因此对于将要为其投入新料的 RGV 来说，CNC 的维修状态与加工作业状态无异。其他模型构建维持不变。

5.2 问题二的分析和求解

5.2.1 基于时间的仿真模型的建立

为了检验模型的实用性和算法的有效性，我们需要用仿真模型对调度模型进行测试。

为了模拟整个过程，我们将其分为若干个时间步长Δt，根据建立的调度模型，模拟每个步长中发生的事件。这里，Δt不能过大或过小。若Δt过大，会导致模拟的精度较低；若Δt过小，会增加不必要的计算量。这里，我们取Δt为题中所给所有时间的最大公约数 1s，因而能在保证精度的同时，使计算量最小。

在每一个时间步Δt开始时，我们需要更新 CNC 和 RGV 的状态信息，判断上一步的状

态（如加工、上下料等）是否结束。若 CNC 的加工状态结束，则将其状态改为空闲状态。若 RGV 上一步状态已经结束，则我们需要根据调度模型，进行决策，判断 RGV 此步的状态（例如，RGV 上下料结束，则此步开始清洗）。

为了跟踪加工物料，我们对加工物料从 1 开始编号，在每个物料上下料时，记录其加工 CNC 编号、上料开始时间、下料开始时间。我们用 Python 程序实现了上述模拟过程（详见附录 2）。

我们模拟一定的时间后，可以得到整个过程的调度策略，及每个物料成品的信息。根据物料成品的数量，我们可以计算出作业效率η：

$$\eta = \frac{n_s t_{process}}{t_{total} n_{CNC}} \tag{6}$$

其中，n_s为物料成品的数量，$t_{process}$为每个物料的加工时间，t_{total}为总模拟时长，n_{CNC}为 CNC 机器的数量。

5.2.2 基于蒙特卡罗方法的机器损坏的模拟

在第三种情况中，CNC 在加工过程中可能发生故障，发生的概率约为 1%。这里，根据几何概型，发生故障的概率为故障时间（即故障排除时间）与总时间的比值，即：

$$\lambda = \frac{t_{fix_{total}}}{t_{total}} \tag{7}$$

式中，λ为发生故障的概率，$t_{fix_{total}}$为 CNC 进行故障排除的总时间，t_{total}为总时长

为了模拟故障情况，我们在每一个时间步长Δt中用蒙特卡洛方法[6]模拟机器的损坏。这里，我们用 Python 内置的随机数生成器[13]生成一个随机数r，若$r \le p$，则机器发生损坏。因而，故障发生次数n_{fix}的期望值为：

$$n_{fix} = \frac{t_{total}}{\Delta t} p \tag{8}$$

进行故障排除的总时间与故障发生次数的比值即为单次故障排除时间：

$$t_{fix} = \frac{t_{fix_{total}}}{n_{fix}} \tag{9}$$

根据上式，我们可以求解得到p的值：

$$p = \frac{\lambda \Delta t}{t_{fix}} \tag{10}$$

题中所给$\lambda = 0.01$，我们已经设Δt为 1s。t_{fix}介于 10~20 分钟之间，我们假设t_{fix}为平均分布，则其平均值为 15min。可得$p = 1.111 \times 10^{-5}$。

为了验证上述p值计算是否正确，我们以$\Delta t = 1s$进行 100 亿秒（约 317 年）的模拟，

在每个时间步Δt开始时，以p为概率，模拟机器故障，设置故障排除时间为 10 分钟至 20 分钟的随机数，并计算每一步故障时间与总时间的比值（详见附录 5）。我们运行程序发现，第 100 亿步时，该比值为 0.0099182538（约等于 1%），整个过程的比值如下图所示：

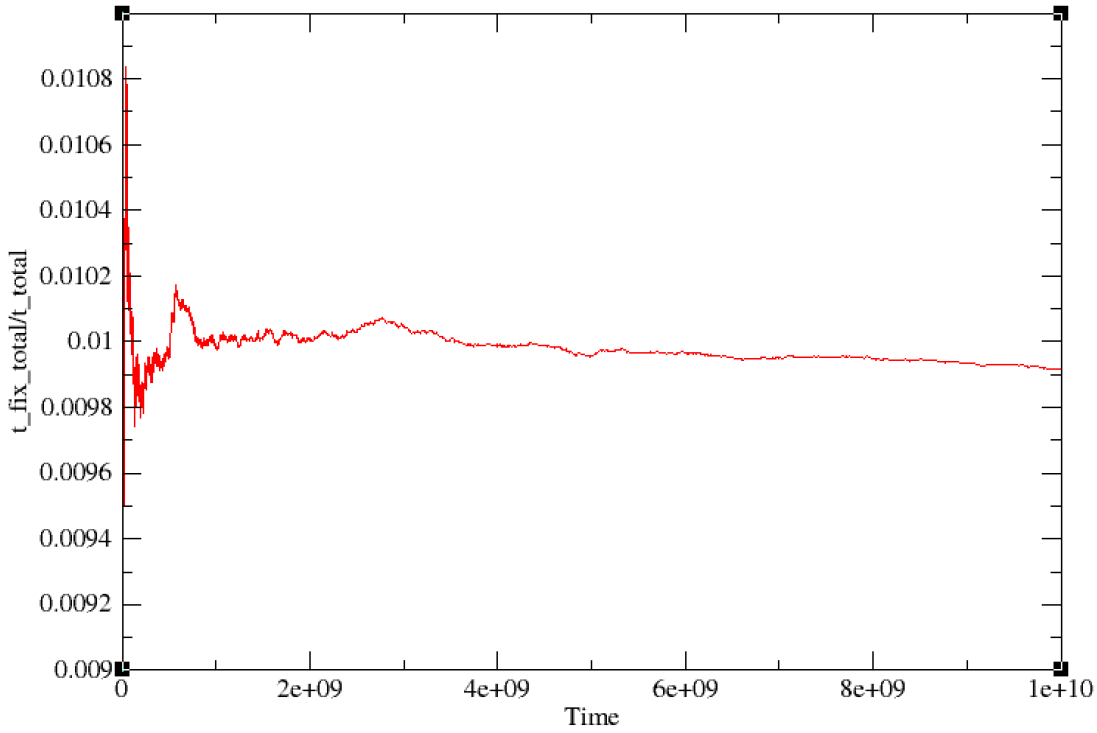

图 2 100 亿秒比值的计算

从图中及最后一步的比值可以看出，随着时间的增加，故障时间与总时间的比值趋近于约 1%。我们对 p 的计算是合理的，可以使得发生故障的概率约等于 1%。

模拟故障后，我们即可以检验我们的模型是否能应对环境的变化。

5.2.3 两道工序之间半成品传送方式的选择

对于两道工序的情况，题目没有给出两道工序间半成品的传送方式。在 RGV 设备允许的情况下，我们猜想可能有用 RGV 的机械臂传送（详见附录 2）和用传送带传送（详见附录 6）等两种物料传送方式。前者需要占用 RGV 的机械臂，使 RGV 只能进行移动操作，而后者将物料从机械臂放至传送带上需要一定的时间。

为了比较两种方式哪一种效率更高，我们在 CNC 刀具的最优安装方式下（详见下一节），对不同组别中两种方式得到的成品数量进行了对比，结果如下：

表 1 两种传送方式得到的成品数量

	成品数(用 RGV 的机械臂传送)	**成品数（用传送带传送）**
第 2 种情况，第 1 组	253	206

第 2 种情况，第 2 组	211	179
第 2 种情况，第 3 组	244	219

由表中数据可知，用 RGV 的机械臂传送的工作效率显著优于用传送带传送的工作效率。因此我们在后文模拟两道工序时，均用 RGV 的机械臂传送半成品。

5.2.4 CNC 刀具最优安装方式的寻找

对于两道工序的情况，每一台 CNC 有两种刀具可供选择安装。不同种刀具的安装数量和位置，影响整个调度模型的效率。由于每个过程的仿真时间较短，我们可以采用枚举法，找到最合适的刀具安装策略（详见附录 4）。

这里，由于每个 CNC 可以选择两种刀具安装，每一组安装策略的数量为

$$2^8 = 256 \tag{11}$$

我们找到 256 种策略的模拟结果中，成品数量的最大值：

$$\mathrm{n}_{\mathrm{s}_{\mathrm{best}}} = \max_{1 \le i \le 256} n_s \tag{12}$$

最大值所对应的刀具安装方式，即为所求最优安装方式。

我们将题设三组数据的第二种情况代入仿真模型，求得最优安装方式如下所示（0 表示第 1 道工序的刀具，1 表示第 2 道工序的刀具）：

表 2 刀具的最优安装方式

数据名	刀具安装方式	成品数
第 2 种情况，第 1 组	0，1，0，1，0，1，0，1	253
第 2 种情况，第 2 组	1，0，1，0，1，0，1，0	211
第 2 种情况，第 3 组	0，1，0，0，1，0，0，1	244

我们在下文模拟两道工序时，均采用上表中的刀具安装方式。

5.2.5 具体结果的求解

我们将题设的三组数据代入仿真模型（详见附录 3），求得三种情况的调度策略如支撑材料所示，作业效率如下表所示：

表 3 各种情况的作业效率

数据名	成品数	作业效率（%）
第 1 种情况，第 1 组	383	93.09
第 1 种情况，第 2 组	360	90.63
第 1 种情况，第 3 组	392	92.73
第 2 种情况，第 1 组	253	85.43

第 2 种情况，第 2 组	211	71.43
第 2 种情况，第 3 组	244	71.27
第 3 种情况，结果 1，第 1 组	380	92.36
第 3 种情况，结果 1，第 2 组	356	89.62
第 3 种情况，结果 1，第 3 组	389	92.02
第 3 种情况，结果 2，第 1 组	238	80.37
第 3 种情况，结果 2，第 2 组	200	67.71
第 3 种情况，结果 2，第 3 组	241	70.40

对于每组数据，我们用 Matplotlib[14]做出甘特图[15]（详见附录 7），如下所示：

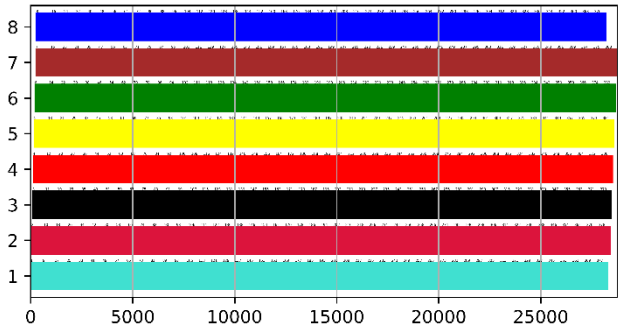

图 3 第 1 种情况，第 1 组的甘特图

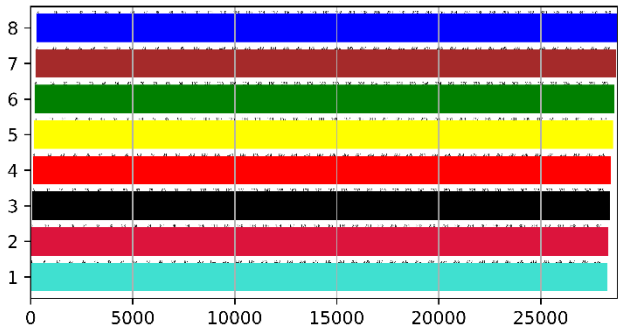

图 4 第 1 种情况，第 2 组的甘特图

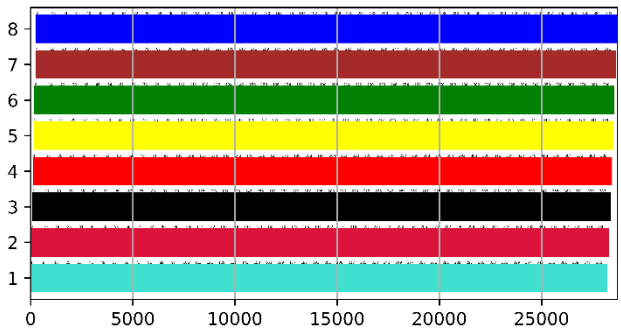

图 5 第 1 种情况，第 3 组的甘特图

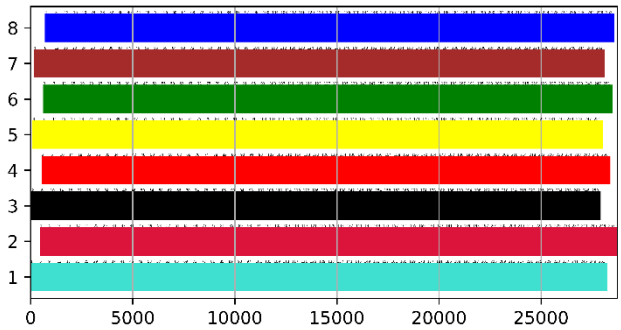

图 6 第 2 种情况，第 1 组的甘特图

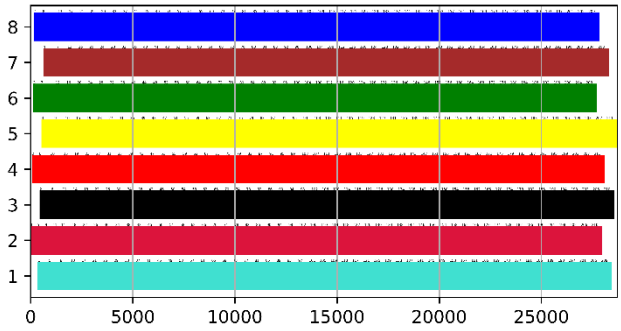

图 7 第 2 种情况，第 2 组的甘特图

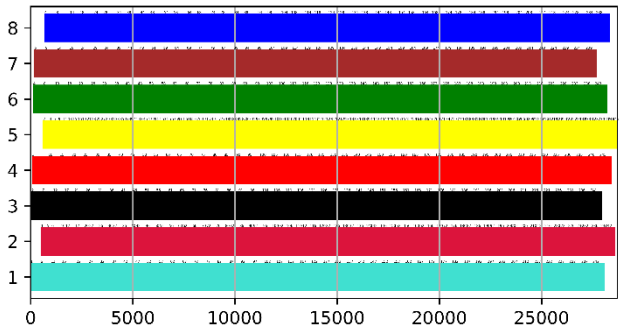

图 8 第 2 种情况，第 3 组的甘特图

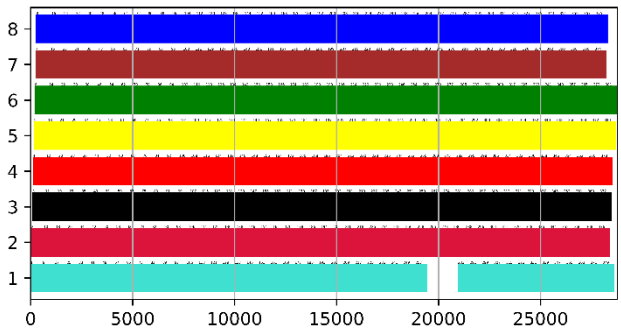

图 9 第 3 种情况，结果 1，第 1 组的甘特图

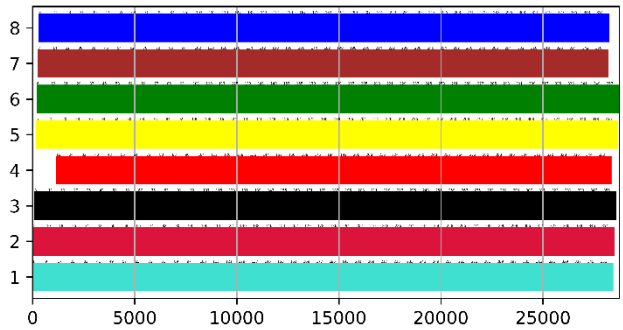

图 10 第 3 种情况，结果 1，第 2 组的甘特图

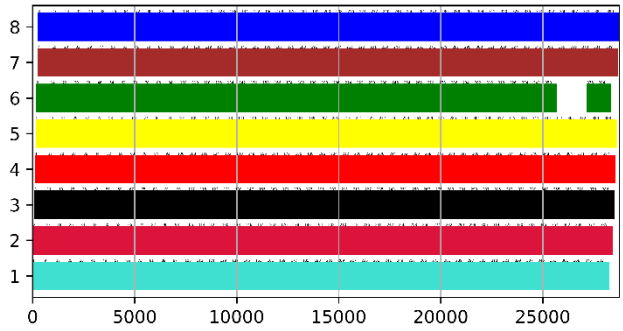

图 11 第 3 种情况，结果 1，第 3 组的甘特图

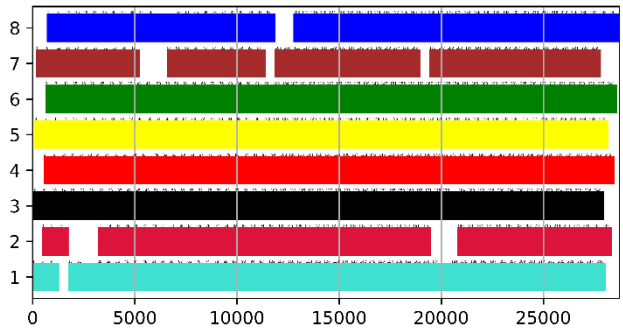

图 12 第 3 种情况，结果 2，第 1 组的甘特图

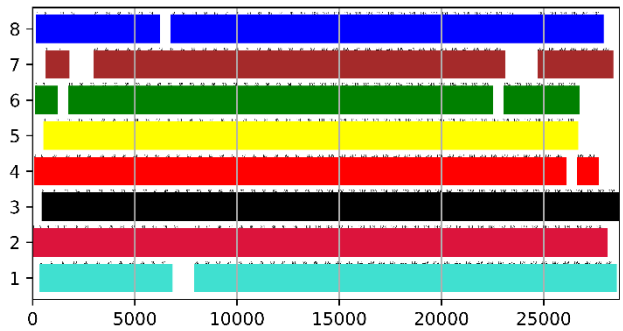

图 13 第 3 种情况，结果 2，第 2 组的甘特图

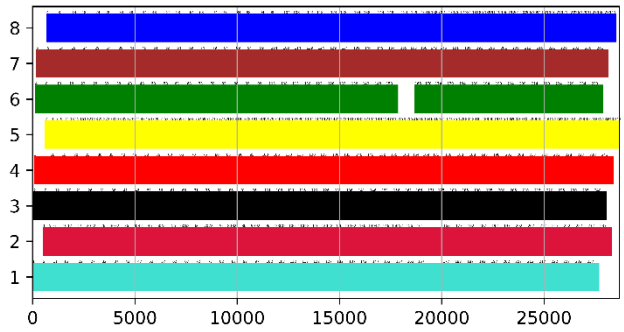

图 14 第 3 种情况，结果 2，第 3 组的甘特图

§6 模型的评价

6.1 模型的优点

1、 利用了调度规则中的动态规则和简单优先规则，将机器故障等不确定因素考虑在内，既符合了现实车间的生产情况，又实现了资源的最优化配置，提高了生产效率；

2、 建立仿真模型，将整个过程分为若干时间步长，模拟每个步长中发生的事件，从而检验我们的模型是否能应对环境的变化；

3、 用蒙特卡洛方法模拟机器的损坏，使发生故障的概率约等于 1%；

4、 对于两道工序的情况中 CNC 刀具的安装，我们利用枚举算法，得到 256 种刀具安装策略，从而找到最优安装策略；

5、 对于两道工序的情况，我们提出了两种方式，一种是将进行完第一道工序的半熟料放置在上料传送带上，由传送带传送至相应 CNC 处；另一种由机械臂直接抓取半熟料放入负责第二道工序的 CNC。这两种模型效率的比较由仿真模拟给出。

6、 计算时间较短。

6.2 模型的缺点

模型从局部入手，将每个 CNC 的T值（$T = t_{move} + t_{left} + t_{load} + t_{wash}$）作为我们简单优化原则的优先级判断指标，这样得到的结果与全局最优解相比，可能会损失一定的精度。

§7 参考文献

[1] 王新, 贾志强, 尚宏美. 基于遗传算法和贪婪算法的作业车间调度[J]. 机械工程师, 2015, (1): 118-120.

[2] Carlier J, Pinson É. An algorithm for solving the job-shop problem[J]. Management science, 1989, 35(2): 164-176.

[3] 叶建芳, 王正肖, 潘晓弘. 免疫粒子群优化算法在车间作业调度中的应用[J]. 浙江大学学报 (工学版), 2008, 5: 029.

[4] 张超勇. 基于自然启发式算法的作业车间调度问题理论与应用研究[D]. 武汉: 华中科技大学, 2006.

[5] El-Bouri A. A cooperative dispatching approach for minimizing mean tardiness in a dynamic flowshop[J]. Computers & Operations Research, 2012, 39(7): 1305-1314.

[6] Metropolis N, Ulam S. The monte carlo method[J]. Journal of the American statistical association, 1949, 44(247): 335-341.

[7] Kiran A S, Smith M L. Simulation studies in job shop sheduling—I a survey[J]. Computers & Industrial Engineering, 1984, 8(2): 87-93.

[8] 范华丽, 熊禾根, 蒋国璋, et al. 动态车间作业调度问题中调度规则算法研究综述[J]. 计算机应用研究, 2016, 33(3): 648-653.

[9] Mouelhi-Chibani W, Pierreval H. Training a neural network to select dispatching rules in real time[J]. Computers & Industrial Engineering, 2010, 58(2): 249-256.

[10] Phanden R K, Jain A, Verma R. A genetic algorithm-based approach for job shop scheduling[J]. Journal of Manufacturing Technology Management, 2012, 23(7): 937-946.

[11] Branke J, Pickardt C W. Evolutionary search for difficult problem instances to support the design of job shop dispatching rules[J]. European Journal of Operational Research, 2011, 212(1): 22-32.

[12] 吴正佳, 何海洋, 黄灿超, et al. 带机器故障的柔性作业车间动态调度[J]. 机械设计与研究, 2015, 31(3): 94-98.

[13] Oliphant T E. Python for scientific computing[J]. Computing in Science & Engineering, 2007, 9(3).

[14] Hunter J D. Matplotlib: A 2D graphics environment[J]. Computing in science & engineering, 2007, 9(3): 90-95.

[15] Maylor H. Beyond the Gantt chart:: Project management moving on[J]. European Management Journal, 2001, 19(1): 92-100.

附录

附录 1 实际使用的软件名称、命令

软件名称：

Python 3.6.6

Grace 5.1.25

Visio 2016

命令：

```
$ python tool.py
$ python broken.py
$ python solve.py
$ python gantt.py
```

附录 2 system.py

```
import numpy as np
import random
import math

# state
STATE_REST=0
STATE_PROCESS=1
STATE_LOAD=2
STATE_WASH=3
STATE_MOVE=4
STATE_BROKEN=5

class process_system(object):
    def
__init__(self,t_move,t_process,tools,n_process,t_load,t_wash,may_broken=Fal
se):
        self.t_move=t_move
        self.t_process=t_process
```

```
        self.t_load=t_load
        self.t_wash=t_wash
        self.tools=tools
        self.n_process=n_process
        self.may_broken=may_broken

        self.timestep=1
        self.t_broken_range=[10*60,20*60]
        self.p_final=0.01

self.p_broken=self.p_final/np.average(self.t_broken_range)*self.timestep

        self.n_CNC=8
        self.t_move_CNC=np.array([[
            ([0]+self.t_move)[abs(i//2-j//2)]
        for j in range(self.n_CNC)] for i in range(self.n_CNC)])


self.CNC=[machine_CNC(id=i,t_process=self.t_process[i],tool=self.tools[i],s
ystem=self,may_broken=self.may_broken) for i in range(self.n_CNC)]

self.RGV=machine_RGV(t_move_CNC=self.t_move_CNC,t_load=self.t_load,t_wash=t
_wash,system=self)
        self.n_sample=0
        self.samples=[]
        if self.may_broken:self.brokens=[]

    def run(self,t_run):
        self.state=STATE_PROCESS
        for self.time in range(0,t_run,self.timestep):
            # Last
            for CNC in self.CNC:
                CNC.before(self.timestep)
            self.RGV.before(self.timestep)
            for CNC in self.CNC:
                CNC.last(self.timestep)
```

```
            self.RGV.last(self.timestep)
        if self.RGV.sample is not None:
            if self.RGV.sample.processstep+1>=self.n_process:
                self.samples.append(self.RGV.sample)

    def determine(self,requirements):
        # Here is what we need to discuss
        move_time=[(0        if        self.CNC[requirement].t_statetotal-
self.CNC[requirement].t_state<=self.t_move_CNC[self.RGV.position][requireme
nt]  else  self.CNC[requirement].t_statetotal-self.CNC[requirement].t_state-
self.t_move_CNC[self.RGV.position][requirement])+
self.t_move_CNC[self.RGV.position][requirement]+self.t_load[requirement]+se
lf.t_wash for requirement in requirements]
        best=requirements[np.where(move_time==np.min(move_time))[0][0]]
        return best

class machine(object):
    def last(self,timestep):
        if not self.state==STATE_REST:
            self.t_state+=timestep

class machine_CNC(machine):
    def __init__(self,id,t_process,tool,system,may_broken=False):
        self.t_process=t_process
        self.system=system
        self.tool=tool
        self.may_broken=may_broken
        self.id=id

        self.state=STATE_REST
        self.t_state=0
        self.t_statetotal=0
        self.sample=None
        self.determine=None

    def before(self,timestep):
```

```
        #before
        self.checkbroken()
        if not self.state==STATE_REST:
            if self.t_state>=self.t_statetotal:
                self.state=STATE_REST
                self.t_state=0
                self.t_statetotal=0

    def checkbroken(self):
        if self.may_broken and self.state==STATE_PROCESS:
            if random.random()<self.system.p_broken:
                # broken
                broken=Broken(sampleid=self.sample.id,CNCid=self.id)
                self.sample=None
                self.state=STATE_BROKEN
                self.t_state=0

self.t_statetotal=random.randint(self.system.t_broken_range[0],self.system.
t_broken_range[1])
                broken.starttime=self.system.time
                broken.endtime=self.system.time+self.t_statetotal
                self.system.brokens.append(broken)
                if self.system.RGV.determine==self.id:
                    self.system.RGV.determine=None

class machine_RGV(machine):
    def __init__(self,t_move_CNC,t_load,t_wash,system):
        self.t_move_CNC=t_move_CNC
        self.t_load=t_load
        self.t_wash=t_wash
        self.system=system

        self.state=STATE_REST
        self.t_state=0
        self.t_statetotal=0
        self.sample=None
```

```
        self.unload_sample=None
        self.load_sample=None
        self.position=1
        self.determine=None

    def before(self,timestep):
        if not self.state==STATE_REST:
            if self.t_state>=self.t_statetotal:
                self.t_statetotal=0
                self.t_state=0
                self.todo()
        else:
            self.todo()

    def todetermine(self):
        if self.unload_sample is not None:
            processstep=self.unload_sample.processstep
        else:
            processstep=0

        if self.determine is None:
        # Determine what to do
            requirements=[i for i in range(len(self.system.CNC))
                            if self.system.CNC[i].tool==processstep]


            if requirements:
                self.determine=self.system.determine(requirements)

    def todo(self):
        #Before a timestep starts
        self.todetermine()
        if self.determine is not None:
            if self.state==STATE_REST:
                # move
                if self.t_move_CNC[self.position][self.determine]>0:
```

```
                self.move(self.determine)
            else:
                self.position=self.determine
                if self.system.CNC[self.determine].state==STATE_REST:
                    self.load(self.determine)
                else:
                    self.state=STATE_REST
                    self.determine=None
        elif self.state==STATE_MOVE:
            self.position=self.determine
            if self.system.CNC[self.determine].state==STATE_REST:
                self.load(self.determine)
            else:
                self.done()
        elif self.state==STATE_LOAD:
            self.load_complete(self.determine)
        elif self.state==STATE_WASH:
            self.done()

def done(self):
    self.state=STATE_REST
    self.determine=None
    self.todetermine()
    self.todo() # do the next thing

def move(self,move_to):
    self.state=STATE_MOVE
    self.t_statetotal=self.t_move_CNC[self.position][move_to]

def load(self,id):
    self.state=STATE_LOAD
    self.t_statetotal=self.t_load[id]
    if self.system.state==STATE_PROCESS:
        if self.unload_sample is not None:
            # uncompleted_samples
            self.load_sample=self.unload_sample
```

```
            self.unload_sample=None
        if self.load_sample is None:
            # new sample
            self.load_sample=Sample(id=self.system.n_sample)
            self.system.n_sample+=1
        self.load_sample.starttime.append(self.system.time)

    if self.system.CNC[id].sample is not None:
        # unload
        self.unload_sample=self.system.CNC[id].sample
        self.unload_sample.endtime.append(self.system.time)
        self.system.CNC[id].sample=None

def load_complete(self,id):
    # load sample
    if self.load_sample is not None:
        self.system.CNC[id].sample=self.load_sample
        self.load_sample=None
        self.system.CNC[id].state=STATE_PROCESS
        self.system.CNC[id].t_statetotal=self.system.CNC[id].t_process
        self.system.CNC[id].sample.CNCid.append(id)
    if self.unload_sample is not None:
        self.unload_sample.processstep+=1
        if self.unload_sample.processstep>=self.system.n_process:
            # wash sample
            if self.sample is not None:
                # complete
                self.sample.processstep+=1
                self.system.samples.append(self.sample)
                self.sample=None
            self.sample=self.unload_sample
            self.unload_sample=None
            self.state=STATE_WASH
            self.t_statetotal=self.t_wash
        else:
            self.done()
```

```
        else:
            self.done()

class Sample(object):
    def __init__(self,id):
        self.id=id
        self.processstep=0
        self.CNCid=[]
        self.starttime=[]
        self.endtime=[]

class Broken(object):
    def __init__(self,sampleid,CNCid):
        self.sampleid=sampleid
        self.CNCid=CNCid
```

附录 3 solve.py

```
# result:
         # g1      g2      g3
# c1      383     360     392
# c2      253     211     244
# c3r1    380     356     389
# c3r2    238     200     241

from system import process_system
if __name__=='__main__':
    # group1
    t_move1=[20,33,46]
    t_load1=[28,31]*4
    t_wash1=25

    t_process_single1=560
    tools1=[0, 1, 0, 1, 0, 1, 0, 1]
    t_process1=[400 if tool==0 else 378 for tool in tools1]
```

```
# group2
t_move2=[23,41,59]
t_load2=[30,35]*4
t_wash2=30

t_process_single2=580
tools2=[1, 0, 1, 0, 1, 0, 1, 0]
t_process2=[280 if tool==0 else 500 for tool in tools2]

# group3
t_move3=[18,32,46]
t_load3=[27,32]*4
t_wash3=25

t_process_single3=545
tools3=[0, 1, 0, 0, 1, 0, 0, 1]
t_process3=[455 if tool==0 else 182 for tool in tools3]

runtime=8*60*60

print("\t","g1","\t","g2","\t","g3")
print("c1","\t",end='')
# case 1 , group 1
system=process_system(
    t_move=t_move1,
    t_process=[t_process_single1]*8,
    tools=[0]*8,
    n_process=1,
    t_load=t_load1,
    t_wash=t_wash1
)
system.run(runtime)
with open("case1group1.txt",'w') as f:
    for sample in sorted(system.samples,key=lambda x:x.id):
```

```
print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],f
ile=f)
    print(len(system.samples),"\t",end='')

    # case 1 , group 2
    system=process_system(
        t_move=t_move2,
        t_process=[t_process_single2]*8,
        tools=[0]*8,
        n_process=1,
        t_load=t_load2,
        t_wash=t_wash2
    )
    system.run(runtime)
    with open("case1group2.txt",'w') as f:
        for sample in sorted(system.samples,key=lambda x:x.id):

print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],f
ile=f)
    print(len(system.samples),"\t",end='')

    # case 1 , group 3
    system=process_system(
        t_move=t_move3,
        t_process=[t_process_single3]*8,
        tools=[0]*8,
        n_process=1,
        t_load=t_load3,
        t_wash=t_wash3
    )
    system.run(runtime)
    with open("case1group3.txt",'w') as f:
        for sample in sorted(system.samples,key=lambda x:x.id):

print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],f
ile=f)
```

```
    print(len(system.samples),"\t")

    print("c2","\t",end='')
    # case 2, group 1
    system=process_system(
        t_move=t_move1,
        t_process=t_process1,
        tools=tools1,
        n_process=2,
        t_load=t_load1,
        t_wash=t_wash1
    )
    system.run(runtime)
    with open("case2group1.txt",'w') as f:
        for sample in sorted(system.samples,key=lambda x:x.id):

print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],s
ample.CNCid[1]+1,sample.starttime[1],sample.endtime[1],file=f)
    print(len(system.samples),"\t",end='')

    # case 2, group 2
    system=process_system(
        t_move=t_move2,
        t_process=t_process2,
        tools=tools2,
        n_process=2,
        t_load=t_load2,
        t_wash=t_wash2
    )
    system.run(runtime)
    with open("case2group2.txt",'w') as f:
        for sample in sorted(system.samples,key=lambda x:x.id):

print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],s
ample.CNCid[1]+1,sample.starttime[1],sample.endtime[1],file=f)
    print(len(system.samples),"\t",end='')
```

```
    # case 2, group 3
    system=process_system(
        t_move=t_move3,
        t_process=t_process3,
        tools=tools3,
        n_process=2,
        t_load=t_load3,
        t_wash=t_wash3
    )
    system.run(runtime)
    with open("case2group3.txt",'w') as f:
        for sample in sorted(system.samples,key=lambda x:x.id):

print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],s
ample.CNCid[1]+1,sample.starttime[1],sample.endtime[1],file=f)
    print(len(system.samples),"\t")

    print("c3r1","\t",end='')
    # case 3 , result 1, group 1
    system=process_system(
        t_move=t_move1,
        t_process=[t_process_single1]*8,
        tools=[0]*8,
        n_process=1,
        t_load=t_load1,
        t_wash=t_wash1,
        may_broken=True
    )
    system.run(runtime)
    with open("case3result1group1.txt",'w') as f:
        for sample in sorted(system.samples,key=lambda x:x.id):

print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],f
ile=f)
    with open("case3result1group1_broken.txt",'w') as f:
```

```
        for broken in sorted(system.brokens,key=lambda x:x.sampleid):

print(broken.sampleid+1,broken.CNCid+1,broken.starttime,broken.endtime,file
=f)
    print(len(system.samples),"\t",end='')

    # case 3 , result 1 , group 2
    system=process_system(
        t_move=t_move2,
        t_process=[t_process_single2]*8,
        tools=[0]*8,
        n_process=1,
        t_load=t_load2,
        t_wash=t_wash2,
        may_broken=True
    )
    system.run(runtime)
    with open("case3result1group2.txt",'w') as f:
        for sample in sorted(system.samples,key=lambda x:x.id):

print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],f
ile=f)
    with open("case3result1group2_broken.txt",'w') as f:
        for broken in sorted(system.brokens,key=lambda x:x.sampleid):

print(broken.sampleid+1,broken.CNCid+1,broken.starttime,broken.endtime,file
=f)
    print(len(system.samples),"\t",end='')

    # case 3 , result 1 , group 3
    system=process_system(
        t_move=t_move3,
        t_process=[t_process_single3]*8,
        tools=[0]*8,
        n_process=1,
        t_load=t_load3,
```

```
        t_wash=t_wash3,
        may_broken=True
    )
    system.run(runtime)
    with open("case3result1group3.txt",'w') as f:
        for sample in sorted(system.samples,key=lambda x:x.id):

print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],f
ile=f)
    with open("case3result1group3_broken.txt",'w') as f:
        for broken in sorted(system.brokens,key=lambda x:x.sampleid):

print(broken.sampleid+1,broken.CNCid+1,broken.starttime,broken.endtime,file
=f)
    print(len(system.samples),"\t")

    print("c3r2","\t",end='')
    # case 3 , result 2 , group 1
    system=process_system(
        t_move=t_move1,
        t_process=t_process1,
        tools=tools1,
        n_process=2,
        t_load=t_load1,
        t_wash=t_wash1,
        may_broken=True
    )
    system.run(runtime)
    with open("case3result2group1.txt",'w') as f:
        for sample in sorted(system.samples,key=lambda x:x.id):

print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],f
ile=f)
    with open("case3result2group1_broken.txt",'w') as f:
        for broken in sorted(system.brokens,key=lambda x:x.sampleid):
```

```
print(broken.sampleid+1,broken.CNCid+1,broken.starttime,broken.endtime,file
=f)
    print(len(system.samples),"\t",end='')

    # case 2, result 2 , group 2
    system=process_system(
        t_move=t_move2,
        t_process=t_process2,
        tools=tools2,
        n_process=2,
        t_load=t_load2,
        t_wash=t_wash2,
        may_broken=True
    )
    system.run(runtime)
    with open("case3result2group2.txt",'w') as f:
        for sample in sorted(system.samples,key=lambda x:x.id):

print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],f
ile=f)
    with open("case3result2group2_broken.txt",'w') as f:
        for broken in sorted(system.brokens,key=lambda x:x.sampleid):

print(broken.sampleid+1,broken.CNCid+1,broken.starttime,broken.endtime,file
=f)
    print(len(system.samples),"\t",end='')

    # case 2, result 2 , group 3
    system=process_system(
        t_move=t_move3,
        t_process=t_process3,
        tools=tools3,
        n_process=2,
        t_load=t_load3,
        t_wash=t_wash3,
        may_broken=True
```

```
    )
    system.run(runtime)
    with open("case3result2group3.txt",'w') as f:
        for sample in sorted(system.samples,key=lambda x:x.id):

print(sample.id+1,sample.CNCid[0]+1,sample.starttime[0],sample.endtime[0],f
ile=f)
    with open("case3result2group3_broken.txt",'w') as f:
        for broken in sorted(system.brokens,key=lambda x:x.sampleid):

print(broken.sampleid+1,broken.CNCid+1,broken.starttime,broken.endtime,file
=f)
print(len(system.samples),"\t",end='')
```

附录 4 tool.py

```
# choose tool for case 2
# Result:
# group1 [0, 1, 0, 1, 0, 1, 0, 1] 253
# group2 [1, 0, 1, 0, 1, 0, 1, 0] 211
# group3 [0, 1, 0, 0, 1, 0, 0, 1] 244

from system import process_system
if __name__=='__main__':
    # group1
    t_move1=[20,33,46]
    t_load1=[28,31]*4
    t_wash1=25

    # group2
    t_move2=[23,41,59]
    t_load2=[30,35]*4
    t_wash2=30

    # group3
```

```
t_move3=[18,32,46]
t_load3=[27,32]*4
t_wash3=25

runtime=8*60*60

# case 2, group 1
max1=0
for i in range(2**8):
    tools=[int(x) for x in bin(i)[2:].zfill(8)]
    t_process1=[400 if tool==0 else 378 for tool in tools]
    system=process_system(
        t_move=t_move1,
        t_process=t_process1,
        tools=tools,
        n_process=2,
        t_load=t_load1,
        t_wash=t_wash1
    )
    system.run(runtime)
    if len(system.samples)>max1:
        max1=len(system.samples)
        tools1=tools

print("group1",tools1,max1)

# case 2, group 2
max2=0
for i in range(2**8):
    tools=[int(x) for x in bin(i)[2:].zfill(8)]
    t_process2=[280 if tool==0 else 500 for tool in tools]
    system=process_system(
        t_move=t_move2,
        t_process=t_process2,
        tools=tools,
        n_process=2,
```

```
            t_load=t_load2,
            t_wash=t_wash2
        )
        system.run(runtime)
        if len(system.samples)>max2:
            max2=len(system.samples)
            tools2=tools

    print("group2",tools2,max2)

    # case 2, group 3
    max3=0
    for i in range(2**8):
        tools=[int(x) for x in bin(i)[2:].zfill(8)]
        t_process3=[455 if tool==0 else 182 for tool in tools]
        system=process_system(
            t_move=t_move3,
            t_process=t_process3,
            tools=tools,
            n_process=2,
            t_load=t_load3,
            t_wash=t_wash3
        )
        system.run(runtime)
        if len(system.samples)>max3:
            max3=len(system.samples)
            tools3=tools

print("group3",tools3,max3)
```

附录 5 broken.py

```
# Last line of the result:
# 10000000000 0.0099099874
```

```
import random,math
p_final=0.01
timestep=1
t_fix_ava=15*60
time=0
p=p_final/t_fix_ava*timestep

state=0
t_fixing=0
t_fixtotal=0

while time<10**10:
    if state==1:
        if t_fixing<t_fix:
            t_fixing+=timestep
            t_fixtotal+=timestep
        else:
            state=0
    else:
        if random.random()<p:
            state=1
            t_fixing=0
            t_fix=random.randint(10*60,20*60)

    time+=timestep
    if time%1000==0:
        print(time,t_fixtotal/time)
```

附录6 system_belt.py

```
# 用传送带传送
import numpy as np
import random
import math
```

```
# state
STATE_REST=0
STATE_PROCESS=1
STATE_LOAD=2
STATE_WASH=3
STATE_MOVE=4
STATE_BROKEN=5

class process_system(object):
    def
__init__(self,t_move,t_process,tools,n_process,t_load,t_wash,may_broken=Fal
se):
        self.t_move=t_move
        self.t_process=t_process
        self.t_load=t_load
        self.t_wash=t_wash
        self.tools=tools
        self.n_process=n_process
        self.may_broken=may_broken

        self.timestep=1
        self.t_broken_range=[10*60,20*60]
        self.p_final=0.01

self.p_broken=self.p_final/np.average(self.t_broken_range)*self.timestep

        self.n_CNC=8
        self.t_move_CNC=np.array([[
            ([0]+self.t_move)[abs(i//2-j//2)]
        for j in range(self.n_CNC)] for i in range(self.n_CNC)])


self.CNC=[machine_CNC(id=i,t_process=self.t_process[i],tool=self.tools[i],s
ystem=self,may_broken=self.may_broken) for i in range(self.n_CNC)]

self.RGV=machine_RGV(t_move_CNC=self.t_move_CNC,t_load=self.t_load,t_wash=t
```

```
_wash,system=self)
        self.n_sample=0
        self.samples=[]
        self.uncompleted_samples=[]
        if self.may_broken:self.brokens=[]

    def run(self,t_run):
        self.state=STATE_PROCESS
        for self.time in range(0,t_run,self.timestep):
            # Last
            for CNC in self.CNC:
                CNC.before(self.timestep)
            self.RGV.before(self.timestep)
            for CNC in self.CNC:
                CNC.last(self.timestep)
            self.RGV.last(self.timestep)
        if self.RGV.sample is not None:
            if self.RGV.sample.processstep+1>=self.n_process:
                self.samples.append(self.RGV.sample)

    def determine(self,requirements):
        # Here is what we need to discuss
        move_time=[(0         if         self.CNC[requirement].t_statetotal-
self.CNC[requirement].t_state<=self.t_move_CNC[self.RGV.position][requireme
nt]  else  self.CNC[requirement].t_statetotal-self.CNC[requirement].t_state-
self.t_move_CNC[self.RGV.position][requirement])+
self.t_move_CNC[self.RGV.position][requirement]+self.t_load[requirement]+se
lf.t_wash for requirement in requirements]
        best=requirements[np.where(move_time==np.min(move_time))[0][0]]
        return best

class machine(object):
    def last(self,timestep):
        if not self.state==STATE_REST:
            self.t_state+=timestep
```

```
class machine_CNC(machine):
    def __init__(self,id,t_process,tool,system,may_broken=False):
        self.t_process=t_process
        self.system=system
        self.tool=tool
        self.may_broken=may_broken
        self.id=id

        self.state=STATE_REST
        self.t_state=0
        self.t_statetotal=0
        self.sample=None
        self.determine=None

    def before(self,timestep):
        #before
        self.checkbroken()
        if not self.state==STATE_REST:
            if self.t_state>=self.t_statetotal:
                self.state=STATE_REST
                self.t_state=0
                self.t_statetotal=0

    def checkbroken(self):
        if self.may_broken and self.state==STATE_PROCESS:
            if random.random()<self.system.p_broken:
                # broken
                broken=Broken(sampleid=self.sample.id,CNCid=self.id)
                self.sample=None
                self.state=STATE_BROKEN
                self.t_state=0

self.t_statetotal=random.randint(self.system.t_broken_range[0],self.system.
t_broken_range[1])
                broken.starttime=self.system.time
                broken.endtime=self.system.time+self.t_statetotal
```

```
                self.system.brokens.append(broken)

class machine_RGV(machine):
    def __init__(self,t_move_CNC,t_load,t_wash,system):
        self.t_move_CNC=t_move_CNC
        self.t_load=t_load
        self.t_wash=t_wash
        self.system=system

        self.state=STATE_REST
        self.t_state=0
        self.t_statetotal=0
        self.sample=None
        self.unload_sample=None
        self.load_sample=None
        self.position=1
        self.determine=None

    def before(self,timestep):
        if not self.state==STATE_REST:
            if self.t_state>=self.t_statetotal:
                self.t_statetotal=0
                self.t_state=0
                self.todo()
        else:
            self.todo()

    def todetermine(self):
        if self.determine is None:
        # Determine what to do
            requirements=[i for i in range(len(self.system.CNC))
                          if        (self.system.CNC[i].tool==0        or
len(self.system.uncompleted_samples)>0) ]

            if requirements:
                self.determine=self.system.determine(requirements)
```

```
def todo(self):
    #Before a timestep starts
    self.todetermine()
    if self.determine is not None:
        if self.state==STATE_REST:
            # move
            if self.t_move_CNC[self.position][self.determine]>0:
                self.move(self.determine)
            else:
                self.position=self.determine
                if self.system.CNC[self.determine].state==STATE_REST:
                    self.load(self.determine)
                else:
                    self.state=STATE_REST
                    self.determine=None
        elif self.state==STATE_MOVE:
            self.position=self.determine
            if self.system.CNC[self.determine].state==STATE_REST:
                self.load(self.determine)
            else:
                self.done()
        elif self.state==STATE_LOAD:
            self.load_complete(self.determine)
        elif self.state==STATE_WASH:
            self.done()

def done(self):
    self.state=STATE_REST
    self.determine=None
    self.todo() # do the next thing

def move(self,move_to):
    self.state=STATE_MOVE
    self.t_statetotal=self.t_move_CNC[self.position][move_to]
```

```
def load(self,id):
    self.state=STATE_LOAD
    self.t_statetotal=self.t_load[id]
    if self.system.CNC[id].sample is not None:
        # unload
        self.unload_sample=self.system.CNC[id].sample
        self.unload_sample.endtime.append(self.system.time)
        self.system.CNC[id].sample=None
    if self.system.state==STATE_PROCESS:
        if self.system.CNC[id].tool==0:
            # new sample
            self.load_sample=Sample(id=self.system.n_sample)
            self.system.n_sample+=1
        else:
            # load from uncompleted samples
            self.load_sample=self.system.uncompleted_samples[0]
            del self.system.uncompleted_samples[0]
        self.load_sample.starttime.append(self.system.time)

def load_complete(self,id):
    # load sample
    if self.load_sample is not None:
        self.system.CNC[id].sample=self.load_sample
        self.load_sample=None
        self.system.CNC[id].state=STATE_PROCESS
        self.system.CNC[id].t_statetotal=self.system.CNC[id].t_process
        self.system.CNC[id].sample.CNCid.append(id)
    # wash sample
    if self.unload_sample is not None:
        if self.sample is not None:
            # complete
            self.sample.processstep+=1
            if self.sample.processstep>=self.system.n_process:
                # completed sample
                self.system.samples.append(self.sample)
            else:
```

```
                    # uncompleted sample
                    self.system.uncompleted_samples.append(self.sample)
                self.sample=None

            self.sample=self.unload_sample
            self.unload_sample=None
            self.state=STATE_WASH
            self.t_statetotal=self.t_wash
        else:
            self.done()

class Sample(object):
    def __init__(self,id):
        self.id=id
        self.processstep=0
        self.CNCid=[]
        self.starttime=[]
        self.endtime=[]

class Broken(object):
    def __init__(self,sampleid,CNCid):
        self.sampleid=sampleid
        self.CNCid=CNCid
```

附录 7 gantt.py

```
import matplotlib.pyplot as plt
import numpy as np

height=16
interval=4
colors =
("turquoise","crimson","black","red","yellow","green","brown","blue")

for table,n_process in
zip(["case1group1","case1group2","case1group3","case2group1","case2group2",
"case2group3","case3result1group1","case3result1group2","case3result1group3
```

```
",",case3result2group1","case3result2group2","case3result2group3"],([1]*3+[2
]*3)*2):
    df=np.loadtxt(table+".txt")
    df=df[np.where(df[:,3]<3600*8)]

    fig,ax=plt.subplots(figsize=(6,3))
    labels=[]
    count=0;
    for i,machine in enumerate(range(8)):
        labels.append(str(i+1))
        data=str(i+1)
        rows=df[np.where(df[:,1]==i+1)]
        for row in rows:
            ax.broken_barh([(row[2],row[3]-row[2])],
((height+interval)*i+interval,height), facecolors=colors[i])
            plt.text(row[2],
(height+interval)*(i+1),int(row[0]),fontsize=2)
            if(row[3]>count):
                count=row[3]
    if n_process==2:
        for i,machine in enumerate(range(8)):
            data=str(i+1)
            rows=df[np.where(df[:,4]==i+1)]
            for row in rows:
                ax.broken_barh([(row[5],row[6]-row[5])],
((height+interval)*i+interval,height), facecolors=colors[i])
                plt.text(row[5],
(height+interval)*(i+1),int(row[0]),fontsize=2)
                if(row[6]>count):
                    count=row[6]
    ax.set_ylim(0, (height+interval)*len(labels)+interval)
    ax.set_xlim(0, count+2)

ax.set_yticks(range(interval+height//2,(height+interval)*len(labels),(heigh
t+interval)))
    ax.set_yticklabels(labels)
    ax.xaxis.grid(True)
    plt.savefig(table+'.png',dpi=800)
    plt.clf()
```