基于遗传算法的 RGV 动态调度建模与优化

摘要

RGV 轨道引导车是智能生产中重要的生产和运输工具，合理制定 RGV 的动态调度策略具有重要意义。本文以生产效率作为性能指标，以 RGV 的指令执行次序作为优化对象，以加工物料的工序、时序以及加工班次的限制时间作为约束条件，以短距离优先原则改进的遗传算法作为优化方法，建立 RGV 最优动态调度模型，并在三组不同作业参数下分析调度模型的实用性和算法有效性。生产效率定义为加工系统有效生产时间与总加工时间的比值。

物料加工为单工序时，以短距离优先原则过滤初始种群，以 CNC 的位置和 RGV 当前状态作为基因编码染色体，以加工数量、CNC 等待时间、RGV 运动时间的加权作为适应度函数，利用遗传算法求解 RGV 的动态调度模型。在该调度模型下，第一、二、三组作业参数的生产效率分别为 92.62%、91.45%、92.76%。

物料加工为双工序时，以 CNC 的位置、RGV 当前状态、物料所处工序作为基因编码染色体，保留单工序的适应度函数，求解 RGV 的动态调度模型。在该调度模型下，第一、二、三组作业参数的生产效率分别为 85.73%、73.35%、72.39%。

CNC 在加工过程中有 1% 的概率发生故障，以随机数法为物料确定故障随机数和 CNC 故障发生时刻。基于无故障时的 RGV 调度模型，以 CNC 的位置、RGV 当前状态、物料所处工序、故障随机数作为基因编码染色体，在无故障时的适应度函数的基础上，以 CNC 原始等待时间与维修时间的和作为新的 CNC 等待时间，利用遗传算法求解 RGV 调度模型。在该调度模型下，单工序时的第一、二、三组作业参数的生产效率分别为 89.81%、88.26%、92.01%。双工序时，第一、二、三组作业参数的生产效率分别为 82.46%、71.15% 、70.19%。

基于遗传算法确定 RGV 调度模型，利用 Java 面向对象语言编程，设计 RGV-CNC 的生产调度模拟器，求解无故障发生和有故障发生两种情况下单双工序的物料加工信息表。本文基于遗传算法建立的 RGV 生产调度模型具有较强的实用性，在解决 RGV 生产调度时具有较好的可推广性。

关键词：RGV 轨道引导车；最优动态调度；遗传算法；生产调度；Java 程序设计

一、 问题重述

1.1 问题背景

某智能加工系统由 8 台计算机数控机床（Computer Number Controller，CNC）、1 台带机械手和清洗槽的轨道式自动引导车（Rail Guide Vehicle，RGV）、1 条 RGV 直线轨道、1 条上料传送带和 1 条下料传送带等附属设备构成。RGV 是一种无人驾驶、能在固定轨道上自由运行的智能车。它自带一个机械手臂、两只机械手爪和物料清洗槽，可以根据指令自动控制移动方向和距离，完成上下料及清洗物料等作业任务。智能加工系统的示意图如图 1 所示。

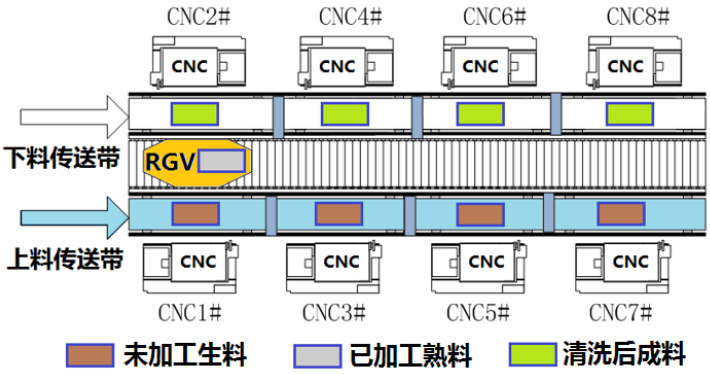

图 1　智能加工系统示意图

1.2 问题提出

考虑物料加工的以下三种情况：（1）加工一个物料需要进行一道工序且 CNC 不发生故障，每台 CNC 安装相同的刀具；（2）加工一个物料需要两道工序且 CNC 不发生故障，每台 CNC 安装两种刀具中的一种且加工过程中不可替换，安装不同种刀具的 CNC 只能加工对应的工序；（3）加工过程中 CNC 发生故障，发生故障的概率为 1%，修复 CNC 的时间为 10~20 分钟，CNC 修复后立即进入作业状态。

对于每一种情况，建立 RGV 的动态调度模型并求解相应的调度算法，同时，基于表 1 中所给出的数据分析调度模型的实用性和算法的有效性。

表 1　智能加工系统作业参数的 3 组数据表

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

二、 模型的假设

- 假设轨道引导车 RGV 在运行过程中不发生故障和意外事故；
- 不考虑 CNC 的刀具在加工过程中的磨损对生产效率造成的影响；
- 不考虑 RGV 在对称位置中移动的时间损耗；
- 忽略 RGV 接收和判别指令的时间以及 CNC 上下料后发送请求的时间；
- 假设 CNC 发生故障后可立即得到维修；
- CNC 前的传送带上始终有物料供 RGV 上下料。

三、 符号说明

符号	意义
T_z	一个班次内加工系统的总作业时间（s）
T_i	一个班次内第 i 台 CNC 的作业时间（s）
N_z	一个班次内加工系统的总物料加工数量（个）
N_i	一个班次内第 i 台 CNC 的物料加工数量（个）
η	一个班次内系统的生产效率
t_i	第 i 个 CNC 的加工时钟
t_R	RGV 的加工时钟

四、 问题分析

4.1 单工序时 RGV 调度策略分析

针对第一种情况，即物料的加工过程为单工序过程时，需要确定 RGV 的动态调度策略，使得在一个班次内整个智能加工系统的生产效率最高。鉴于整个生产流水线为单个 RGV 为多个 CNC 提供服务，为提高加工效率，在对 RGV 进行调度时，需要使得 8 个 CNC 的加工总量最大。当 CNC 空闲或者加工完当前物料时，向 RGV 发送请求服务指令，RGV 收到指令后将请求服务指令存储至请求服务指令列表。当 RGV 处于空闲或者执行完上一条服务指令后，对其请求服务指令列表内的指令作出判断，选择当前最优服务指令进行执行。当前最优指令的选择标准为使得整个生产流水线加工效率最高的调度指令。对于处于对称位置上的 CNC，若两者均存在于请求服务列表中，由于 RGV 为偶数编号 CNC 一次上下料所需时间要大于为奇数编号 CNC 一次上下料所需时间，从优化服务时间的角度考虑，RGV 优先为奇数编号的 CNC 进行上下料。在任意时刻，RGV 接收指令后直接将指令存储至指令列表。当 RGV 处于空闲状态或者执行完上一条指令时，基于最优指令的选择标准和对位 CNC 选择优先级，即可从指令列表中确定当前最优执行指令。在执行指令时，RGV 处于接收指令但不处理指令的状态，当 RGV 执行完当前指令时立即判别下一条执行指令。当班次结束时，加工完成的物料数即为该调度模型的性能指标。加工系统在单一班次内加工物料数量最大时的 RGV 调度算法即为 RGV 最优调度策略。计算加工系统的物料加工数量时，考虑全部 CNC 在单一班次内的物料加工数量的总和，由于总生产周期较长，且只存在一辆 RGV 轨道引导车，故只需要计算当前最优解，即计算当前状态下 RGV 内已经接收的指令的最优执

行策略。每一阶段的最优调度策略使加工系统所生产的物料的总和即为整个生产周期内的最大物料加工量。每次判别执行指令时，以指令列表内已经存在的指令所能组成的指令执行序列作为优化对象，以加工系统的生产效率作为优化目标，以 CNC 的运行时间序列、单个物料的加工时间和系统整体加工时间作为约束条件，利用遗传算法寻找全局最优解，即寻找当前最优指令执行序列，使得加工系统的生产效率最高。

4.2 双工序时 RGV 调度策略分析

针对第二种情况，即物料的加工过程为双工序时，确定 RGV 的动态调度策略以及 CNC 加工类型的组合方式，使得一个班次内整个加工系统的生产效率最高。在确定 RGV 的动态调度策略之前，需要先确定 CNC 加工类型的最优组合。在确定最优组合时，考虑到每组数据的特殊性，需要分别对每组数据给定 CNC 加工类型的组合方式。确定 CNC 的加工类型的组合方式后，需要确定 RGV 的动态调度策略。与第一种情况不同的是，RGV 在为加工第一道工序的某个非空的 CNC 上下料后，已完成第一道工序的半成品不能被直接送出系统而会停留在 RGV 的机械手上，使得 RGV 无法为加工第一道工序的 CNC 进行上料操作，因此下一条执行的指令必须是为加工第二道工序的 CNC 上下料。当 RGV 空闲时或为加工第二道工序的 CNC 上下料以及清洗成品工作完成后，RGV 就既可以为加工第一道工序的 CNC 上下料，也可以为加工第二道工序的 CNC 下料。与第一种加工情况类似，RGV 的在作业时如果收到服务指令，直接将指令存储至指令队列。当 RGV 空闲或者完成当前指令后，按照调度原则为 CNC 进行上下料。该种情况下调度策略的求解方案分为 CNC 加工类型组合的确定和 RGV 最优执行指令序列的确定。确定 CNC 加工类型组合时采用遍历法尝试每一种不同加工工序组合方案，除去全部加工第一道工序和全部加工第二道工序的情况，一共有 2^8-2 种可行方案。确定 RGV 的最优执行指令序列时，RGV 的调度的限制条件为：RGV 完成第一道工序的上下料后，如果有半成品留在 RGV 中，则只能完成第二道工序的上下料；RGV 空闲、完成第二道工序的上下料后或完成第一道工序的上下料但没有半成品留在 RGV 中时，可以完成第一道或者第二道工序的上下料。基于第一种情况下的遗传算法求解 RGV 指令执行序列的方法，加入 RGV 调度的限制条件，依然将加工系统的生产效率作为优化指标，求解第二种情况下的 RGV 动态调度算法。确定初始调度模型后使用遗传算法对调度模型进行优化，使整个加工系统的生产效率最高。

4.3 CNC 出现故障时 RGV 的调度策略分析

针对第三种情况，即物料的加工过程有出错可能时，在原来第一二种情况下，考虑实际生产中 CNC 有发生故障的可能，对上文所建的模型进一步优化。由于 CNC 发生故障的概率在 1% 左右，在整个班次内，对每一次上料操作取 $[0,1)$ 范围内的一个随

机数，若随机数落在 $[0, 0.01)$ 范围内，则将该次上料操作对应 CNC 标记为出错，然后分别在加工时间内和 10~20 分钟范围内随机选择一个时刻作为出错时间和维修耗时。当 CNC 发生故障时，清除 CNC 中正在运行的物料并停止发出任何指令直至其被修复，修复完成后 CNC 发出上料请求并回归正常工作状态。基于上文所建立的模型以及改进的遗传算法，求解发生故障情况下的动态调度模型。

五、 模型的建立与求解

5.1 单工序时 RGV 动态调度模型的建立与求解

5.1.1 单工序时 RGV 的调度模型

物料的加工过程为单工序时，以整个系统的生产效率作为调度算法的性能优化指标，对 RGV 的动态调度算法进行优化。整个智能加工系统的运作流程如下, 智能加工系统作业流程图如图 2 所示。

（1）当 CNC 处于空闲或者已完成物料加工时，向 RGV 发送上下料服务请求。

（2）RGV 接收到 CNC 发送的上下料的服务请求时，直接将服务请求指令存储至接收指令列表。

（3）判断 RGV 是否处于空闲（即不处于上下料、轨道运动、清洗作业状态）。若 RGV 空闲，则从服务指令列表中按照最优指令调度算法选择最优指令执行；若 RGV 不空闲，则继续执行当前指令，直至当前指令执行完毕。

（4）每当当前指令执行完毕时，判断当前班次是否结束。若当前班次结束，则加工系统结束作业；若班次不结束，则返回第（1）步，继续执行作业流程。

由智能加工系统的作业流程图可知，对于单工序的加工过程，要提高系统的加工效率，主要是对 RGV 调度算法进行优化。在每一加工班次内，假设整个加工系统的作业总时间为时间常数 T_z，即在 8 小时内所有 CNC 的整体运行时间，假设第 i 个 CNC 在每一班次内的作业时长为 T_i，则加工系统的作业总时间为：

$$T_z = \sum_{i=1}^{8} T_i \tag{1}$$

由于每一班次的加工时间为 8 小时，以秒作为时间单位计算 T_z 可得:

$$T_z = 230400s$$

假设每一班次内第 i 台 CNC 加工物料的数量为 N_i，整个加工系统的物料加工数量为 N_z，物料加工数量的计算方法为：

$$N_z = \sum_{i=1}^{8} N_i \tag{2}$$

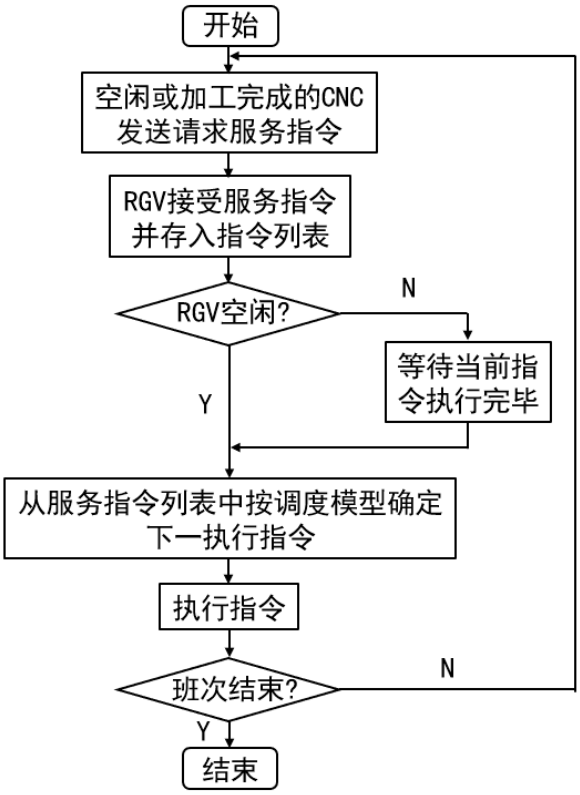

图 2　智能加工系统的作业流程图

在计算生产效率时，定义加工系统的有效作业时间为物料加工的总数量与加工单个物料所需时间的乘积，用符号 T_y 表示。设加工单个物料所需时间为 T_0，则加工系统的有效工作时间的计算方式为：

$$T_y = N_z \cdot T_0 \tag{3}$$

由于不考虑加工成本即其他设备损耗对生产效率造成的干扰，故可直接将每一班次内整个加工系统的有效作业时间 T_y 与每一班次的系统总作业时间 T_z 的比值作为系统的生产效率。假设系统的生产效率为 η，则生产效率 η 的计算方式为：

$$\eta = \frac{T_y}{T_z} \times 100\% \tag{4}$$

调度算法的优化性能指标为使生产效率尽可能大，即：

$$P : \eta \to max \tag{5}$$

以加工系统通电启动的时刻作为系统开始加工的零时刻，假设 $t_i(i = 1, 2, ...8)$ 为第 i 台 CNC 的加工时钟，t_R 为 RGV 轨道引导车的加工时钟。第 i 台 CNC 的作业序列时间轴如图 3 所示，RGV 轨道引导车的作业序列时间轴如图 4 所示。

在第 i 个 CNC 的作业序列时间轴中，t_{is}^j 表示第 i 个 CNC 上加工第 j 个物料时的开始上料时刻。t_{if}^j 表示第 i 个 CNC 上加工第 j 个物料时的加工完成时刻，且 CNC 在完

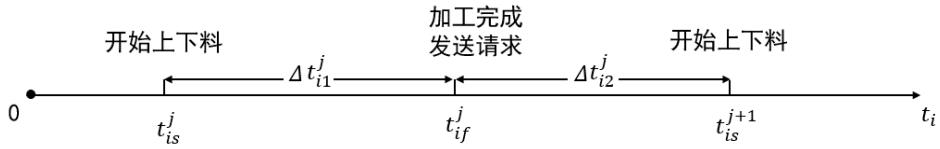

图 3　第 i 台 CNC 的作业序列时间轴

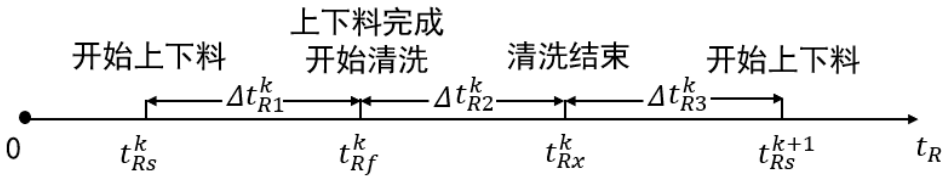

图 4　RGV 轨道引导车的作业序列时间轴

成时刻直接向 RGV 发送上下料指令。由基本假设可知，忽略 CNC 完成加工作业后发送上下料请求的时间，即物料的加工完成时刻即为 CNC 向 RGV 发送上下料请求的时刻。t_{is}^{j+1} 表示第 i 个 CNC 上加工第 j+1 个物料的开始上料时刻，同时，也是第 j 个物料的下料时刻。$\triangle t_{i1}^{j}$ 表示第 i 个 CNC 在加工第 j 个物料时从开始上料时刻到物料加工完成时刻之间的时间差，考虑实践加工过程，即为 RGV 为 CNC 上下料的时间与 CNC 加工完成一个一道工序的物料所需时间的和，即为 T_0。$\triangle t_{i2}^{j}$ 表示第 i 个 CNC 在加工第 j 个物料时从发送上下料请求时刻到为加工第 j+1 个物料进行上下料的开始时刻之间的时间差。该时间差即为 RGV 为其他 CNC 上下料、清洗、轨道运行的时间的求和。假设 RGV 为第 i 个 CNC 进行上下料之前还需要对 m 个 CNC 进行上下料，则 $\triangle t_{i2}^{j}$ 的计算表达式为：

$$\triangle t_{i2}^{j} = m \cdot (t_{Rs}^{k+1} - t_{Rs}^{k}) \tag{6}$$

其中 m 的取值由 RGV 的调度算法决定，m 为非负正整数。在 RGV 的作业序列时间轴中，t_{Rs}^{k} 表示 RGV 为整个加工系统生产的第 k 个物料进行上料的开始时刻。t_{Rf}^{k} 表示 RGV 为整体加工系统生产的第 k 个物料进行上料的完成时刻，同时也是第 k+1 个物料的上料开始时刻和第 k 个物料的清洗作业开始时刻。t_{Rx}^{k} 表示第 k 个物料的清洗完成时刻。t_{Rs}^{k+1} 是第 k+1 个物料的开始上料时刻，也是第 k 个物料的下料时刻。$\triangle t_{R1}^{k}$ 表示第 k 个物料的开始上料时刻与上料完成时刻的时间差，即 RGV 为 CNC 的上下料时间。$\triangle t_{R2}^{k}$ 表示第 k 个物料的下料时刻与清洗作业完成时刻的时间差，即 RGV 完成清洗物料作业的时间。$\triangle t_{R3}^{k}$ 表示第 k 个物料的清洗结束时刻与第 k+1 个物料的上料时刻之间的时间差，即 RGV 选择下一执行指令后运动至下一 CNC 处的时间。由基本假设可知，不考虑 RGV 在对称位置中移动的时间损耗，设 Move 表示 RGV 的移动单位的数量。

则 $\triangle t_{R3}^{k}$ 的表达式为：

$$\triangle t_{R3}^{k}=\begin{cases}0 & , \quad Move=0\\ t_{m1} & , \quad Move=1\\ t_{m2} & , \quad Move=2\\ t_{m3} & , \quad Move=3\end{cases} \tag{7}$$

设第 i 个 CNC 在一个班次内最多加工 n_i 个物料，一个班次内整个系统最多加工 N_z 个物料。即 i=1,2,$\cdots$,8；j=1,2,$\cdots$,n_i；k=1,2,$\cdots$,N_z。由 CNC 的作业序列时间轴和 RGV 的作业序列时间轴可以建立单工序时 RGV 的调度模型：

$$max:\eta=\frac{N_z\cdot T_0}{T_z}\times 100\%$$

$$s.t.\quad\begin{cases}t_{is}^{j+1}>t_{is}^{j}+\triangle t_{i1}^{j}\\ t_{Rs}^{k+1}>t_{Rs}^{k}+\triangle t_{R1}^{k}+\triangle t_{R2}^{k}\\ t_{Rs}^{k+1}=min\{t_{is}^{j+1}\}\\ T_i>t_{Rs}^{N_z+1}+\triangle t_{R2}^{k}\\ T_i>N_z\cdot T_0+\triangle t_{R2}^{k}\\ \triangle t_{i2}^{j}\geq\triangle t_{R3}^{k}\end{cases}$$

其中 $\triangle t_{i1}^{j}=T_0$，即 RGV 的上下料时间与 RGV 加工单个工序的物料的时间的和；$\triangle t_{R1}^{k}$ 是 RGV 上下料时间；$\triangle t_{R2}^{k}$ 是 RGV 完成清洗作业的时间。T_0 是一个物料的加工加上下料时间。T_i 是每个 CNC 整个班次的加工时间。t_{is}^{j} 是第 i 个 CNC 上加工的第 j 个物料的开始上料时刻，也是第 j-1 个物料的开始下料时刻；t_{Rs}^{k} 是 RGV 为整个加工系统加工的第 k 个物料的开始上料时刻，也是第 k-1 个物料的开始下料时刻。$\triangle t_{i2}^{j}$ 是第 i 个 CNC 的等待上下料时间。$\triangle t_{R3}^{k}$ 是 RGV 的运动时间。

5.1.2 基于改进遗传算法的单工序 RGV 调度算法

（1) 初始化种群并优化

经典的遗传算法 [1] 的流程图如图 5 所示，其在求解动态调度问题时基本使用随机生成若干染色体方案，以此作为初始种群，染色体生成的随机性使得初始种群中包含多种合理性极低的方案，增加算法的复杂度，会使得算法收敛过快，增加陷入局部最优解的风险。故本文在迭代开始前先对初始种群过滤，以最短距离原则过滤掉合理性过低的方案，以此获取高质量的种群。

（2）染色体编码方式

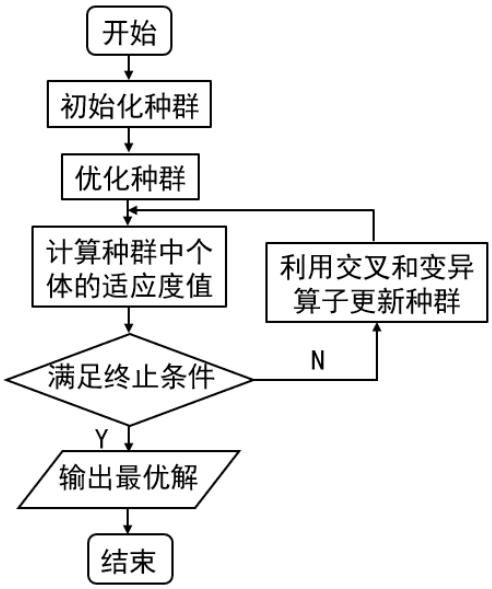

图 5　经典遗传算法流程图

染色体编码应包含指令列表中 CNC 的位置以及 RGV 当前的状态，本文采用二维向量的方式编码，其结构如下：

$$\begin{bmatrix} r_1 & r_2 & \cdots & r_n \\ c_1 & c_2 & \cdots & c_n \end{bmatrix}$$

其中 r_k 用以标记接受到任务信号时 RGV 是否空闲，若 RGV 处于作业状态则不能立即响应指令，将 r_k 的值取 0，否则取 1；c_k 表示发送服务请求信号的 CNC 序号。

（3）适应度函数

首先确定适应度子函数。以 RGV 移动时间和最小作为第一个适应度子函数，即：

$$f_1 = \sum_{k=1}^{N_z} r_k \times t_g^k \tag{8}$$

其中 t_g^k 表示加工第 k 个物料时 RGV 移动到发送请求服务信号的 CNC 的时间。以 CNC 的整体等待时间和最小作为第二个适应度子函数，即：

$$f_2 = \sum_{i=1}^{8} \sum_{j=1}^{n} \triangle t_{i2}^j \tag{9}$$

其中 $\triangle t_{i2}^j$ 表示 CNC 从发出上下料指令到被服务的等待时间。以完成任务数 N_z 最大作为第三个适应度子函数，即：

$$f_3 = N_z \tag{10}$$

然后对适应度子函数进行加权 [2] 。设适应度函数为 f_z，则适应度函数的的表达式为：

$$f_z = a \cdot (f_{1max} - f_1) + b \cdot (f_{2max} - f_2) + c \cdot (f_3 - f_{3min}) \tag{11}$$

其中，权值 a、b、c 的计算公式如下：

$$\begin{cases} a = \frac{1}{f_{1max} - f_{1min}} \\ b = \frac{1}{f_{2max} - f_{2min}} \\ c = \frac{1}{f_{3max} - f_{3min}} \end{cases} \tag{12}$$

（4）进化算子

种群数目为 N 时，遗传概率为 P_s，将种群中的个体按照适应度大小从大到小排序，从前面保留 $N * P_s$ 个个体，放入繁殖池中作为新种群。当 $N * P_s$ 不是整数时，对其向上取整。

（5）交叉算子

交叉是对种群更新的一种重要方法，合适的交叉概率和交叉方法的选择才能避免非法子染色体的产生，本文使用的交叉方法 [3] 如下：

- 生成 N 个（0，1）间的随机数，分别对应 N 个个体；
- 生成父代集合：取随机数小于交叉概率 P_c 的个体作为交叉的父代，形成父代集合；
- 交叉：任选两个父代 A 和 B，对于父代染色体 A，生成两个小于 A 染色体长度的随机数，以这两个数对应的基因中间的基因串作为待交换的基因串 S，删除其对应位置的 B 染色体中的基因串，再将 S 插入，即形成一个新的子染色体 C。
- 将 C 染色体投入繁殖池，重复第 2、3 步，直到父代集合中的个体个数小于等于 1。

（6）变异算子

变异能提高种群的多样性，扩大可行解的搜索区间。本文使用的变异方法如下：

- 生成 N 个（0，1）间的随机数，分别对应 N 个个体；
- 生成父代集合：取随机数小于变异概率的个体作为变异的父代，形成父代集合；
- 变异：从父代集合中任取一个个体 A, 生成小于该染色体长度的两个随机数，交换两个随机数对应位置的基因，即形成新的子染色体 B；
- 将 B 染色体投入繁殖池，重复第 2、3 步，直至父代集合为空。

基于改进遗传算法的单工序的调度模型为：以最短距离优先原则过滤初始遗传种群，以 RGV 移动时间和、CNC 整体等待时间和、加工系统加工的物料数量这三者的加权和作为适应度函数，利用改进的遗传算法确定 RGV 内已经接收的指令的最佳执行次序。

5.1.3 单工序调度模型的实用性和算法有效性分析

分别对基于单工序调度模型的短距离优先原则的调度模型和基于改进遗传算法的调度模型进行实用性分析。以短距离优先作为调度原则时，三组不同数据下整个加工系统的加工物料总数量、有效工作时间、生产效率的数值如下表 2 所示。

表 2　短距离优先调度的加工信息

组号	加工物料总量 (个)	有效加工时间 (s)	生产效率
1	356	209862	91.09%
2	337	206410	89.59%
3	366	210267	91.26%

使用改进遗传算法所确定的调度次序作为调度模型时，三组不同数据下整个加工系统的加工物料总数量、有效工作时间、生产效率的数值如下表 3 所示。

表 3　改进遗传算法确定的调度模型的加工信息

组号	加工物料总量 (个)	有效加工时间 (s)	生产效率
1	362	213399	92.62%
2	344	206410	91.45%
3	372	213714	92.76%

5.2 双工序时 RGV 动态调度模型的建立与求解

5.2.1 双工序时 RGV 的调度模型

双工序调度的数学模型与单工序一致。双工序时的 RGV 的调度算法与单工序调度算法的不同之处在于：双工序的调度算法中，首先需要确定各台 CNC 加工类型的组合方式，然后确定 RGV 选择执行指令序列的调度算法。在确定 CNC 的加工工序的组合方式时，为消除各台 CNC 加工类型的组合方式对 RGV 调度算法选择的影响，直接全局遍历 CNC 所有可能的加工工序组合方式，即穷举法遍历 $2^8 - 2$ 种组合方式。对于每一种组合方式，都以加工系统的生产效率作为优化指标，对调度算法进行优化，选择当前组合方式中最优调度算法并记录生产效率，每一种组合方式内的最优调度算法所确定的生产效率称为该种组合方式下的最大生产效率。比较所有组合的最大生产效率，将最大生产效率数值最大的组合方式确定为最优加工类型的组合方式，该组合方式内的最优调度算法即为 RGV 的最优调度算法。在 RGV 的调度过程中，需要注意以下两个约束：当有半成品在 RGV 的机械手中时，RGV 只能为第二道工序进行上下料；当 RGV 机械手中没有半成品或者为第二道工序进行上下料后，可以为第一道工序上下料或者第二道工序进行下料。即 RGV 机械手中有半成品时，只能对指令列表内的加工第

二道工序的 CNC 发送的指令做出响应；RGV 在完成为第二道工序进行下料后或者处于空闲状态且机械手中为空半成品时，能对指令列表中的所有 CNC 发出的指令做出响应。在单工序 RGV 的调度模型基础上增加协调双工序运行的约束条件，就可以得到双工序时的 RGV 调度模型：

$$max:\eta=\frac{N_z\cdot T_0}{T_z}\times 100\%$$

$$s.t.\quad\begin{cases}t_{is}^{j+1}>t_{is}^{j}+\triangle t_{i1}^{j}\\ t_{Rs}^{k+1}>t_{Rs}^{k}+\triangle t_{R1}^{k}+\triangle t_{R2}^{k}\\ t_{Rs}^{k+1}=min\{t_{is}^{j+1}\}\\ T_i>t_{Rs}^{N_z+1}+\triangle t_{R2}^{k}\\ T_i>N_z\cdot T_0+\triangle t_{R2}^{k}\\ \triangle t_{i2}^{j}\geq\triangle t_{R3}^{k}\\ \sum_{i=1}^{Card(CNC1)}E_{ijr}=1\\ \sum_{j=1}^{Card(CNC2)}E_{ijr}=1\\ T_r^j\geq T_r^i+\triangle T_{CNC_i}+\triangle T_{move(i,j)}\end{cases}$$

在三个双工序协调约束条件中，$CNC1$ 和 $CNC2$ 分别表示处理第一道工序和第二道工序的 CNC 的集合，E_{ijr} 是一个指示变量，如果第 r 号物料由第 i 号工序一 CNC 加工第一道工序且由第 j 号工序二 CNC 加工第二道工序，则 E_{ijr} 为 1，否则为 0；T_r^i 表示第 r 号物料进入第 i 号 CNC 的时刻，$\triangle T_{CNC_i}$ 表示 i 号 CNC 的加工时间和上下料时间之和，$T_{move(i,j)}$ 表示 RGV 从第 i 号 CNC 处移动到第 j 号 CNC 处所用时间。

5.2.2 基于遗传算法的双工序 RGV 调度算法

双工序时由遗传算法确定的 RGV 调度模型由单工序时的调度模型衍生，其双工序时的编码方式、种群优化方法不一致，但优化目标均为使得整体加工系统的生产效率最大。双工序时的遗传算法的流程图如图 6 所示。以下是双工序时的遗传算法确定 RGV 调度模型的过程：

（1）新的编码方式

两道工序下的动态调度问题，染色体编码还应包含发送信号的 CNC 所做的工序序号信息，故我们采用三维向量的方式对染色体编码，即在前文编码方式的基础上增加第三行，表示发送信号的 CNC 所做的工序序号，其中 p_k 表示第 k 个物料的加工状态，物料处于第一道工序时取值为 1，处于第二道工序时取值为 2。则新的染色体结构为：

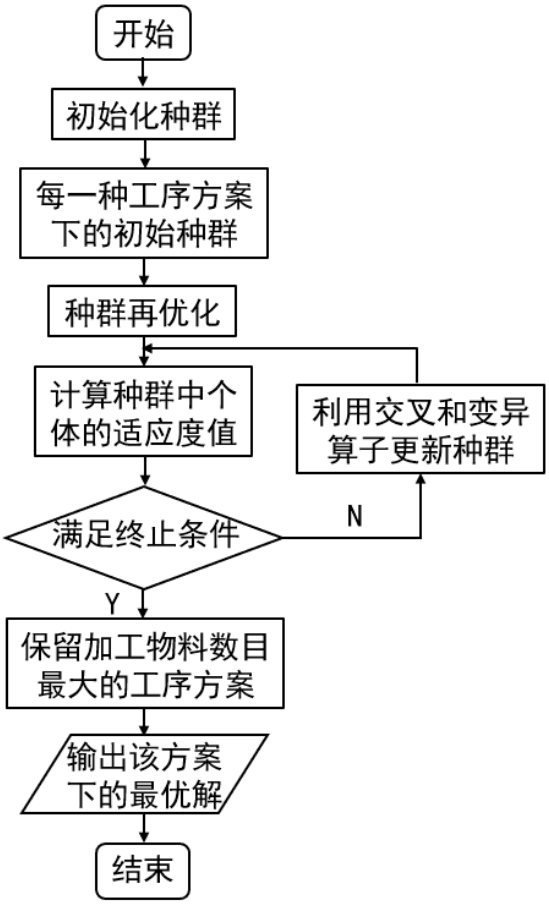

图 6　双工序时遗传算法的流程图

$$\begin{bmatrix} r_1 & r_2 & \cdots & r_n \\ c_1 & c_2 & \cdots & c_n \\ p_1 & p_2 & \cdots & p_n \end{bmatrix}$$

（2）新的初始种群优化方法

对于两道工序下的动态调度问题，首先要规划做第二道工序的 CNC 数量及位置，以求得最优的方案。考虑到第一二道工序的完成时间有差异，但做两道工序的 CNC 数量不能相差太大，否则易造成大量 CNC 处于等待状态，故为了简化算法，仅考虑做两道工序的 CNC 数量分布为 4:4、3:5、2:6 三种情况，建立新的初始种群优化方法如下：

- 按数量分布 4:4、3:5 和 2:6 随机生成若干 CNC 加工工序组合，如 1、3、5、7 号 CNC 做第一道工序，2、4、6、8 号 CNC 做第二道工序。
- 随机生成若干个体，形成初始化种群，再基于上一步的工序方案，淘汰掉不符合该方案的个体，故对于每一种工序方案，都生成一个与之对应的初始种群。

（3）种群再优化

当某 CNC 完成第一道工序并得到 RGV 的响应时，产生的半熟料应立即被另一个

加工第二道工序的 CNC 加工，故在编码的第三行，凡出现 1 的基因，其后的基因都应为 2，以此为原则淘汰掉不符合条件的个体，对种群再优化。

（4）更新与求解

由于两道工序下和一道工序下动态调度问题的优化目标一致，依然是减少 CNC 等待的时间和和 RGV 移动的时间和，并使完成的任务数目尽量多，故保留前文的进化算子、交叉算子和变异算子，对每一个再优化后的种群，基于交叉算子和变异算子更新种群，扩大可行解搜索范围，基于进化算子优胜劣汰，逐步迭代求得全局最优解。

（5）工序方案的选择

优化的最终目标是整个加工系统的生产效率最高，故以最大任务数目原则选择工序方案，将该方案下的全局最优解为两道工序下的最优解，将该调度算法即为 RGV 最优调度策略。

5.2.3 双工序调度模型的实用性和算法有效性分析

以改进遗传算法确定的 RGV 调度模型作为 RGV 调度策略时，三组不同数据下整个加工系统的 CNC 加工工序组合、加工物料总数量、有效工作时间、生产效率的数值如下表 4 所示。

表 4　遗传算法的双工序调度算法的加工信息

组号	工序一的 CNC 编号	工序二的 CNC 编号	加工总数量	生产效率
1	1、3、5、7	2、4、6、8	235	85.73%
2	1、2、5、7	3、4、6、8	200	73.35%
3	1、2、4、6、7	3、5、8	240	72.39%

5.3 CNC 出现故障时 RGV 动态调度模型的建立与求解

5.3.1 CNC 出现故障时的调度模型

CNC 出现故障时的调度模型与情况一中的调度模型一致。在实际生产中，考虑 CNC 在加工过程中会发生故障，在题设条件给出故障的发生概率为 1%，以（0,1）作为随机数的产生范围，当随机数落在 [0,0.01）区间时，认为 CNC 发生故障。若 CNC 在加工过程中会发生故障，则在 CNC 的加工时间段内选择一个随机时间点，当 CNC 加工至该时间点时，CNC 发生故障。由基本假设可知，CNC 发生故障后可立即得到维修，故 CNC 的故障发生时刻即为开始维修时刻。确定维修时间时，选择（10,20）内的某一随机数作为 CNC 的所需维修时间。当 CNC 处于被维修的状态时，无法向 RGV

发送任何指令请求，RGV 的指令执行序列与该 CNC 无关。RGV 只在其他正常工作的 CNC 中选择指令执行，RGV 的调度模型与情况一和情况二中的调度模型一致。

5.3.2 CNC 出现故障时基于遗传算法的调度算法

CNC 出现故障时的调度算法与前两种情况基本相同。基于 CNC 出现故障时的实际调度模型，在前文的遗传算法模型上作出以下两点改进：

（1）编码方式

由于发生故障的概率约为 1%，且故障排除时间介于十到二十分钟，故确立新的编码方式如下：

- 对于单工序和双工序下的编码方式，均增加新的一行，随机产生（0,1）的随机数，填进新的一行，将 [0,0.01）的随机数更新为 1，表示其对应的 CNC 发生故障，将 [0.01,1）的随机数更新为 0，表示该 CNC 正常工作.
- 随机产生在加工时间段内的随机时间点，表示发生故障的时间节点，逐次求出其与当前新行的乘积，更新当前行。
- 随机产生（10，20）间的随机数，表示维修的时间，逐次求出其与当前新行的乘积，并对新行作最后一次更新，记为 z_k。

则新的编码方式如下所示：

$$\begin{bmatrix} r_1 & r_2 & \cdots & r_n \\ c_1 & c_2 & \cdots & c_n \\ p_1 & p_2 & \cdots & p_n \\ z_1 & z_2 & \cdots & z_n \end{bmatrix}$$

（2）适应度函数

CNC 正常工作时，上下料后开始加工，一段加工时间后完成加工并发送请求服务指令，等待一段时间后开始下一次上下料，当 CNC 在加工过程中发生故障，其加工时间被延长。为了简化模型，将此维修时间加在等待时间上，对第二个适应度子函数作如下更新：

$$f_2 = \sum_{i=1}^{8} \sum_{j=1}^{n_i} \triangle t_{i2}^{j} + \sum_{k=1}^{N_z} z_k$$

5.3.3 CNC 出现故障时模型的实用性和算法的有效性分析

（1）单工序时出现故障的实用性分析

第一、二、三组数据下发生故障的物料序号、CNC 编号、故障发生时间、故障结束时间分别如表 5、6、7 所示。

表 5　单工序时第一组数据下故障信息表

物料序号	CNC 编号	故障开始时刻 (s)	故障结束时刻 (s)
7	7	769	1646
149	6	12270	13398
360	3	28751	29694

表 6　单工序时第二组数据下故障信息表

物料序号	CNC 编号	故障开始时刻 (s)	故障结束时刻 (s)
59	3	5385	6537
177	2	14992	16166
227	7	19144	20151

表 7　单工序时第三组数据下故障信息表

物料序号	CNC 编号	故障开始时刻 (s)	故障结束时刻 (s)
14	6	1355	2363
216	1	16405	17219
331	5	25236	25924

单工序时，在可能发生故障的情况下，三组数据下遗传算法确定的调度模型的实用性分析如表 8 所示。

表 8　单工序且有故障时基于遗传算法的调度模型的加工信息

组号	加工数量 (个)	有效作业时间 (s)	生产效率
1	351	206913	89.81%
2	332	203355	88.26%
3	369	211993	92.01%

（2）双工序时出现故障的实用性分析

第一、二、三组数据下发生故障的物料序号、CNC 编号、故障发生时间、故障结束时间分别如表 9、10、11 所示。

表 9　双工序时第一组数据下故障信息表

物料序号	CNC 编号	故障开始时刻 (s)	故障结束时刻 (s)
12	8	2056	2689
21	1	2621	3611
57	4	8048	9089
131	4	16299	17264
190	3	22729	23702
218	4	26601	27716

表 10　双工序时第二组数据下故障信息表

物料序号	CNC 编号	故障开始时刻 (s)	故障结束时刻 (s)
68	8	10220	11241
122	8	18077	18849
141	7	20032	21099
152	4	22176	23306

表 11　双工序时第三组数据下故障信息表

物料序号	CNC 编号	故障开始时刻 (s)	故障结束时刻 (s)
2	5	718	1745
30	7	3419	4458
48	6	5934	7010
80	4	9816	10740
186	1	22297	23441

双工序时，在可能发生故障的情况下，三组数据下遗传算法确定的调度模型的实用性分析和算法的有效性分析如表 12 所示。

表 12　双工序且有故障时基于遗传算法的调度模型的加工信息

组号	工序一的 CNC 编号	工序二的 CNC 编号	加工熟料数量 (个)	生产效率
1	1、3、5、7	2、4、6、8	226	82.46%
2	1、2、5、7	3、4、6、8	190	71.15%
3	1、2、4、6、7	3、5、8	231	70.19%

六、 模型的评价与推广

6.1 模型的评价

6.1.1 模型的优点

- 优化算法方面，使用了遗传算法对 RGV 调度算法进行优化，在保证答案准确性的同时避免了遍历带来的巨大计算量；
- 模型建立方面，使用 Java 语言编写建模程序、python 语言进行数据处理，使建模程序与数据处理程序分离，在结合两种语言的优势的同时降低了调试的难度，减少建模和调试的工作量；
- 程序结构方面，使用注入式 IoC 程序结构，充分发挥了 Java 语言在解决复杂结构化问题方面的天然优势，使模型程序结构清晰，具有很强的泛化和扩展能力，并能大大减少建模所需的代码量。

6.1.2 模型的缺点

- 优化算法方面，对双工序时各 CNC 的加工类型和排列方式进行优化使用的遍历方法在 CNC 数目很多时不适用；
- 模型建立方面，仅考虑了 RGV 控制器计算能力较低时使用的调度方法；实际上某些计算能力较强的智能 RGV 可以做到对 CNC 的运行状态进行读取和预测，在这种情况下，生产线的效率有进一步提高的可能；
- 程序结构方面，建模和分析时使用的编程语言没有便捷的可视化工具，输出不够直观，不利于模拟结果的可视化。

6.2 模型的推广

- 依据题目中所给的数据和流程说明，建立了 RGV-CNC 生产线的生产流程模型，使用遗传算法分别确定了单工序和双工序情况下的最优 RGV 调度过程，使用遍历的方法确定了 8 台 CNC、双工序情况下使用三组不同配置时的最优 CNC 加工类型的组合，对 RGV-CNC 式生产线的效率优化方面有一定的参考价值；
- 模型中分析解决问题的一些独到方法，遗传优化的一般思想，对于其他数学问题的解决有一定的参考价值；
- 模型实现时使用的针对复杂流程结构问题的现代编程语言和先进的程序构造方法方法在解决其他复杂流程问题时也能广泛适用。

参考文献

[1] 杨书森, 郭顺生. 基于混合遗传算法的生产调度研究 [J]. 机械制造,2017,55(10):108-111+116.

[2] 江唯, 何非, 童一飞, 李东波. 基于混合算法的环形轨道 RGV 系统调度优化研究 [J]. 计算机工程与应用,2016,52(22):242-247.

[3] 朱虹. 基于遗传算法的集成电路生产制造调度 [J]. 集成电路应用,2018,35(04):55-58.

附录 A RGV-CNC 模拟器 -Java 核心代码

1.1 RGV 核心框架

```
package RGV;

import java.util.LinkedList;
import java.util.List;

public abstract class RGV
{//状态相关
   int CNC_id;//在哪个前面RGVCNC
   Material hand;//机械手抓着的工件
   List<Integer> q;//消息队列
   //当前系统时间
   private int time;//时间
   //常量
   private final int[] move_time;//移动时间
   private final int[] load_time;//上下料时间
   private final int wash_time;//清洗作业时间RGV
   private final CNC[] cncs;//所有的CNC

   /**
   * 的构造函数复制进行进行基本的初始化RGV
   *
   * @param rc 设置RGV
   */
   RGV(RGVConfig rc){
      //变量初始化
      CNC_id = 0;
      hand = null;
      q = new LinkedList<Integer>();
      time = 0;
      //常量初始化
      this.move_time = rc.move_time;
      this.load_time = rc.load_time;
      this.wash_time = rc.wash_time;
```

```
    this.cncs=rc.cncs;
    Material.idReset();//材料重新计数id
}

/**
* 调度方法，根据系统状态返回下一步要去哪个CNC
*
* @return 要去的的CNCid
*/
abstract int dispatch();

/**
* 系统运行一步
* 有请求就判断移动拿起装载清洗----
* 没有请求就等待一步
* 然后更新和消息队列状态CNC
*/
public void run(){
    if (!q.isEmpty()){//有请求才会响应
        moveto(dispatch());//移动
        pick();//拿起一个物料
        load();//装载
        wash();//清洗
    }
    else waiting();//否则等待
    update();
}

/**
* 更新所有的状态CNC
* 并更新消息队列
*/
void update(){
    for (CNC cnc : cncs)
        cnc.run(time, q);//更新状态CNC
}
```

```
/**
* 移动RGV
* 并更新时间
*
* @param CNC_id 移动到哪个前面CNC
*/
private void moveto(int CNC_id){
   //位置CNC=编号CNC/2
   int bias = this.CNC_id / 2 - CNC_id / 2;
   this.CNC_id = CNC_id;
   time += move_time[Math.abs(bias)];
}

/**
* 拿起一个物料
* 时间忽略
*/
protected void pick(){
   if (hand != null)
      throw new IllegalArgumentException("机械手中有物料不可抓取,");
   hand = new Material();
   //调试输出**********************
   System.out.printf("工序一\t%d\t%d\t%d\上料开
       始t\n", hand.id + 1, time,CNC_id+1);
   //调试输出**********************
}

/**
* 拿出面前的里面的物料，然后向装载物料RGVCNCCNC
* 并更新时间
*/
private void load(){
   //调试输出**********************
   if (cncs[CNC_id].m != null)
      System.out.printf("%s\t%d\t%d\t%d\下料开始t\n", cncs[CNC_id].type?"工序
          二":"工序一",cncs[CNC_id].m.id + 1, time,CNC_id+1);
   if (hand != null&&hand.type)
```

```
            System.out.printf("工序二\t%d\t%d\t%d\上料开
                始t\n",hand.id + 1, time,CNC_id+1);
        //调试输出*********************
        time += load_time[CNC_id];
        hand = cncs[CNC_id].reload(time, hand);***
    }

    /**
    * 清洗手中的物料并送出系统
    * 并更新时间
    */
    protected void wash(){
        if (hand != null){
            //手里有东西就去清洗
            time += wash_time;
            hand = null;
        }
    }

    /**
    * 等待1s
    * 并更新时间
    */
    protected void waiting(){
        time++;
    }

    /**
    * 获取当前时间RGV
    *
    * @return 时间RGV
    */
    public int getTime(){
        return time;
    }
}
```

1.2 单工序 RGV 模拟类

```
package RGV;
public class SRGV extends RGV{
   public SRGV(RGVConfig rc){
      super(rc);
   }

   int dispatch(){//这是最近距离调度demo
      int lmin = CNC_id - q.get(0);
      int inearst = 0;
      for (int i = 1; i < q.size(); i++){
         if (Math.abs(CNC_id - q.get(i)) < lmin){
            lmin = Math.abs(CNC_id - q.get(i));
            inearst = i;
         }
      }
      int nearst = q.get(inearst);
      q.remove(inearst);
      return nearst;
   }
}
```

1.3 双工序 RGV 模拟类

```
package RGV;
public class MRGV extends RGV{
   private boolean[] CNC_type;

   public MRGV(RGVConfig rc){
      super(rc);
      this.CNC_type = new boolean[rc.CNC_num];
      for (int i = 0; i < rc.CNC_num; i++)
         this.CNC_type[i] = rc.cncs[i].type;
   }

   int dispatch(){//前面的算法可以保证到这里的时候队列里有能响应的
```

```
    boolean type_tofind = hand != null && hand.type;
    //手没有东西的时候去找原料或者是拿着东西的时候去找CNChand.型对应的typeCNC
    int lmin = 65535;
    int r = -1;
    for (int i = 0; i < q.size(); i++){
        if (CNC_type[q.get(i)] == type_tofind){//有要找的cnc
            if (Math.abs(CNC_id - q.get(i)) < lmin)//而且路径比较短
                r = i;//就选这个
            lmin = Math.abs(CNC_id - q.get(i));//更新最小值
        }
    }
    if (r == -1)
        throw new IllegalArgumentException("找不到可以响应的CNC");
    int res = q.get(r);
    q.remove(r);
    return res;
}

@Override
public void run(){
    if (hand != null && !hand.type)//如果手里拿着东西但不是半成品,
        throw new IllegalArgumentException("机械手不能一直抓着成品或原料");
    else{//剩下两种情况手里没拿东西手里拿着半成品:|
        if (!q.isEmpty()){
            for (int id : q)
                if (CNC_type[id] == (hand != null)){
                //手里拿着半成品即hand != null找半成品,即CNCtrue
                //手里拿着东西即hand==null原料半成品,即CNCfalse
                    super.run();//就运行
                    return;//然后返回
                }
        }
        waiting();
        update();//没有半成品加工请求就等待CNC
    }
}
```

```
    @Override
    protected void pick(){
    //如果去半成品就不任何物料CNCpick手里是什么就给什么否则就要,,一个新物料pick
    if (!CNC_type[CNC_id])
        super.pick();
    }

    @Override
    protected void wash(){
        if (hand != null && !hand.type){
        //手里拿着完成品才去清洗
            super.wash();
        }
    }
}
```

1.4 CNC 核心框架

```
package RGV;

import java.util.List;

public class CNC{
    final int id;//的编号CNC
    final int process_time;//这台的加工时间
    final boolean type;//可以加工哪个种类型的物料在,中仅用于判断物料类型不匹配CNC

    int last_input_time;//上一次上料的时间
    boolean waiting;//是否处于等待状态
    Material m;//当前物料

    /**
    * 初始化CNC
    *
    * @param process_time 这台的加工时间CNC
    * @param id 这台的编号       CNC
```

```
    */
    public CNC(int id, int process_time, boolean type){
        this.id = id;
        this.process_time = process_time;
        this.type = type;
        last_input_time = 0;
        waiting = false;
        m = null;
    }

    /**
    * 运行加工过程到某个时刻
    *
    * @param t 某个时刻
    * @param q 如果一个加工流程完就向此队列输入请求
    */
    void run(int t, List<Integer> q){
    //没有物料或者如果一个加工过程完且未发出上料请求就要发出请求m
        if (!waiting && (t > last_input_time + process_time || m == null)){
            q.add(id);//向此队列输入请求请求内容为这台,的编号CNC
            waiting = true;//置等待状态
    }
}


    /**
    * 装载物料
    * 并记录下物料的装载时间
    * 并让进入工作状态CNC
    *
    * @param t 装载物料的时刻
    * @param m 要装载的物料
    * @return 返回被卸下的物料
    */
    Material reload(int t, Material m){
        if (this.m != null && t < last_input_time + process_time)
            throw new IllegalArgumentException("加工未完成, 不能装载物料\n");
```

```
        if (m != null && m.type != this.type)//物料类型不匹配
            throw new IllegalArgumentException("物料类型不匹配，不能装载物料\n");
        Material tm = this.m;
        this.m = m;
        last_input_time = t;
        waiting = false;
        if (tm != null)//如果有物料出来
            tm.type = !tm.type;//先修改物料类型
        return tm;
    }
}
```

1.5 CNC 概率出错模拟类

```
package RGV;

import java.util.List;

public class ECNC extends CNC
{
    private boolean toError;//本次加工会出故障吗
    private int timetoError;//从加工开始到出故障的时间
    private int timeSolveError;//解决本次故障的时间

    private final double p_error = 0.01;//故障概率
    private final int timeSolveError_max = 20 * 60;//最大检修时间
    private final int timeSolveError_min = 10 * 60;//最小检修时间

    public ECNC(int id, int process_time, boolean type){
        super(id, process_time, type);
        toError = false;
        timetoError = 0;
        timeSolveError = 0;
    }
```

```
void run(int t, List<Integer> q){
   if (t < last_input_time)
      throw new IllegalArgumentException("不可获取上一次上料前的状态");
   if (!toError || t >= last_input_time + timetoError + timeSolveError || t
       < last_input_time + timetoError){
      super.run(t, q);//本次加工不会出错出错了但已经过了排除时间没到故障时间就正
          常运行||||-
      return;//然后返回
   }
   //以上条件都不满足说明处在出错时间
   if (m != null){
      //调试输出**********************
      System.out.printf("%s\t%d\t%d\t%d\物料报废t\n", this.type ? "工序
          二" : "工序一", m.id + 1, last_input_time + timetoError, id + 1);
      System.out.printf("%s\t%d\t%d\t%d\故障排除t\n", this.type ? "工序
          二" : "工序一", m.id + 1, last_input_time + timetoError +
          timeSolveError, id + 1);
      //调试输出**********************
      m = null;
      waiting = false;
   }
}

Material reload(int t, Material m){
   //如果有错误而且在之前发生了而且时没有排除tt
   if (toError && t > last_input_time + timetoError && t < last_input_time +
       timetoError + timeSolveError)
      throw new IllegalArgumentException("该正在进行错误排除CNC不可装卸物
          料,");
   if (m != null){//如果要上料
      toError = Math.random() <= p_error;//随机生成是否出错
      if (toError){//如果有出错
         //在运行时间内随机生成出错时间process_time
         timetoError = (int) (Math.random() * process_time);
         //在和之间随机生成错误排查时间timeSolveError_mintimeSolveError_min
         timeSolveError = (int) (timeSolveError_min + (timeSolveError_max -
             timeSolveError_min) * Math.random());
      }
   }
```

```
        return super.reload(t, m);
    }
}
```

1.6 材料标记类

```
package RGV;

class Material
{
    boolean type=false;
    private static int ids=0;
    int id=ids++;

    static void idReset(){
        ids=0;
    }
}
```

1.7 注入式设置核心

```
package RGV;

public class RGVConfig
{
    int CNC_num;//数量CNC
    int[] move_time;//移动时间
    int[] load_time;//上下料时间
    int wash_time;//清洗作业时间RGV
    CNC[] cncs;//所有CNC

    public RGVConfig(int CNC_num, int[] move_time, int[] load_time, int
        wash_time, CNC[] cncs){
        this.CNC_num = CNC_num;
        this.move_time = move_time;
        this.load_time = load_time;
```

```
        this.wash_time = wash_time;
        this.cncs = cncs;
    }
}
```

1.8 注入式单元测试封装类

```
import RGV.CNC;
import RGV.ECNC;
import RGV.RGV;
import RGV.SRGV;
import RGV.MRGV;
import RGV.RGVConfig;

class DataAnalysis
{
    static RGVConfig singl_noerr_rc(int[] r){
        int CNC_num = 8;
        int[] move_time = {0, r[0], r[1], r[2]};
        int[] load_time = new int[CNC_num];
        int[] process_time = new int[CNC_num];
        for (int i = 0; i < CNC_num; i++)
            process_time[i] = r[3];
        for (int i = 0; i < CNC_num; i++)
            load_time[i] = i % 2 == 0 ? r[6] : r[7];
            int wash_time = r[8];
            CNC[] cncs = new CNC[CNC_num];
        for (int i = 0; i < CNC_num; i++)
            cncs[i] = new CNC(i, process_time[i], false);
        return new RGVConfig(CNC_num, move_time, load_time, wash_time, cncs);
    }

    static RGVConfig multi_noerr_rc(int[] r, int bitnum){
        //数据
        int CNC_num = 8;
        int[] move_time = {0, r[0], r[1], r[2]};
```

```
    boolean[] CNC_type = bitmap(bitnum, CNC_num);
    int[] load_time = new int[CNC_num];
    int[] process_time = new int[CNC_num];
    for (int i = 0; i < CNC_num; i++)
      process_time[i] = !CNC_type[i] ? r[4] : r[5];
    for (int i = 0; i < CNC_num; i++)
      load_time[i] = i % 2 == 0 ? r[6] : r[7];
    int wash_time = r[8];
    CNC[] cncs = new CNC[CNC_num];
    for (int i = 0; i < CNC_num; i++)
      cncs[i] = new CNC(i, process_time[i], CNC_type[i]);
    return new RGVConfig(CNC_num, move_time, load_time, wash_time, cncs);
}

static RGVConfig singl_error_rc(int[] r){
    int CNC_num = 8;
    int[] move_time = {0, r[0], r[1], r[2]};
    int[] load_time = new int[CNC_num];
    int[] process_time = new int[CNC_num];
    for (int i = 0; i < CNC_num; i++)
      process_time[i] = r[3];
    for (int i = 0; i < CNC_num; i++)
      load_time[i] = i % 2 == 0 ? r[6] : r[7];
    int wash_time = r[8];
    CNC[] cncs = new CNC[CNC_num];
    for (int i = 0; i < CNC_num; i++)
      cncs[i] = new ECNC(i, process_time[i], false);
    return new RGVConfig(CNC_num, move_time, load_time, wash_time, cncs);
}

static RGVConfig multi_error_rc(int[] r, int bitnum){
    int CNC_num = 8;
    int[] move_time = {0, r[0], r[1], r[2]};
    boolean[] CNC_type = bitmap(bitnum, CNC_num);
    int[] load_time = new int[CNC_num];
    int[] process_time = new int[CNC_num];
    for (int i = 0; i < CNC_num; i++)
```

```
        process_time[i] = !CNC_type[i] ? r[4] : r[5];
    for (int i = 0; i < CNC_num; i++)
        load_time[i] = i % 2 == 0 ? r[6] : r[7];
    int wash_time = r[8];
    CNC[] cncs = new CNC[CNC_num];
    for (int i = 0; i < CNC_num; i++)
        cncs[i] = new ECNC(i, process_time[i], CNC_type[i]);
    return new RGVConfig(CNC_num, move_time, load_time, wash_time, cncs);
}

static void testSRGV(int[] stage){
    //运行
    RGV rgv = new SRGV(singl_noerr_rc(stage));
    while (rgv.getTime() < 8 * 3600)
    rgv.run();
}

static void testMRGV(int[] stage, int bitnum){
    //运行
    RGV rgv = new MRGV(multi_noerr_rc(stage, bitnum));
    while (rgv.getTime() < 8 * 3600)
        rgv.run();
}

static void testESRGV(int[] stage){
    //运行
    RGV rgv = new SRGV(singl_error_rc(stage));
    while (rgv.getTime() < 8 * 3600)
        rgv.run();
}

static void testEMRGV(int[] stage, int bitnum){
    //运行
    RGV rgv = new MRGV(multi_error_rc(stage, bitnum));
    while (rgv.getTime() < 8 * 3600)
        rgv.run();
}
```

```
    /**
    * 读取出一个正整数的比特图
    *
    * @param n 一个正整数
    * @param w 读取多少位
    * @return 比特图
    */
    private static boolean[] bitmap(int n, int w){
        boolean[] r = new boolean[w];
        for (int i = 1; i <= w; i++)
            r[w - i] = (n >> (i - 1)) % 2 == 1;
        return r;
    }

    static void subtestsMRGV(int[] stage, int bitnum){
        //运行
        RGV rgv = new MRGV(multi_noerr_rc(stage, bitnum));
        while (rgv.getTime() < 8 * 3600)
            rgv.run();
    }

    static void testsMRGV(int[] stage){
        for (int i = 1; i < 255; i++){
            System.out.println("0");
            subtestsMRGV(stage, i);
        }
    }
}
```

1.9 主函数测试代码

```
public class Main
{
    static int[] stage1 = new int[]{20, 33, 46, 560, 400, 378, 28, 31, 25};
    static int[] stage2 = new int[]{23, 41, 59, 580, 280, 500, 30, 35, 30};
```

```
    static int[] stage3 = new int[]{18, 32, 46, 545, 455, 182, 27, 32, 25};

    public static void main(String[] args){
        DataAnalysis.testSRGV(stage1);
        DataAnalysis.testSRGV(stage2);
        DataAnalysis.testSRGV(stage3);
        DataAnalysis.testMRGV(stage1,85);
        DataAnalysis.testMRGV(stage2,53);
        DataAnalysis.testMRGV(stage3,41);

        DataAnalysis.testESRGV(stage1);
        DataAnalysis.testESRGV(stage2);
        DataAnalysis.testESRGV(stage3);
        DataAnalysis.testEMRGV(stage1,85);
        DataAnalysis.testEMRGV(stage2,53);
        DataAnalysis.testEMRGV(stage3,41);


        DataAnalysis.testsMRGV(stage1);
        DataAnalysis.testsMRGV(stage2);
        DataAnalysis.testsMRGV(stage3);
    }
}
```

附录 B 模拟结果整理及分析函数 -python 核心代码

```
def original(path):
    f = open(path, 'r')
    original_data = []
    for line in f.readlines():
        data = line.split('\t')
        original_data.append({
                        "工序": data[0],
                        "工件编号": data[1],
                        "时间": str(eval(data[2]) - 1),
                        "编号CNC": data[3],
```

```
                    "信息": data[4]})
    return original_data



def order(original_data):
    #所有子字典的都是工件编号key
    order_data = {
                "工序一": {
                        "编号CNC": {},
                        "上料时间": {},
                        "下料时间": {},
                        "故障编号CNC": {},
                        "故障开始时间": {},
                        "故障排除时间": {}},
                "工序二": {
                        "编号CNC": {},
                        "上料时间": {},
                        "下料时间": {},
                        "故障编号CNC": {},
                        "故障开始时间": {},
                        "故障排除时间": {}}
            }
    for data in original_data:
        if data["工序"].find("工序一") != -1:
            if data["信息"].find("上料开始") != -1:
                order_data["工序一"]["上料时间"][data["工件编号"]] = data["时间"]
                order_data["工序一"]["编号CNC"][data["工件编号"]] = data["编号CNC"]
            if data["信息"].find("下料开始") != -1:
                order_data["工序一"]["下料时间"][data["工件编号"]] = data["时间"]
            if data["信息"].find("物料报废") != -1:
                order_data["工序一"]["故障编号CNC"][data["工件编号"]] = data["编
                    号CNC"]
                order_data["工序一"]["故障开始时间"][data["工件编号"]] = data["时间"]
            if data["信息"].find("故障排除") != -1:
                order_data["工序一"]["故障排除时间"][data["工件编号"]] = data["时间"]
        if data["工序"].find("工序二") != -1:
            if data["信息"].find("上料开始") != -1:
```

```
            order_data["工序二"]["上料时间"][data["工件编号"]] = data["时间"]
            order_data["工序二"]["编号CNC"][data["工件编号"]] = data["编号CNC"]
        if data["信息"].find("下料开始") != -1:
            order_data["工序二"]["下料时间"][data["工件编号"]] = data["时间"]
        if data["信息"].find("物料报废") != -1:
            order_data["工序二"]["故障编号CNC"][data["工件编号"]] = data["编
                号CNC"]
            order_data["工序二"]["故障开始时间"][data["工件编号"]] = data["时间"]
        if data["信息"].find("故障排除") != -1:
            order_data["工序二"]["故障排除时间"][data["工件编号"]] = data["时间"]
    return order_data



def process_output_process1(order_data):
    #遍历工件id
    print('加工物料序号\加工编号tCNC\上料开始时间t\下料开始时间t')
    for mid in order_data["工序一"]["编号CNC"]:
        process = mid + '\t' + order_data["工序一"]["编号CNC"][mid] + '\t'
        if mid in order_data["工序一"]["上料时间"]:
            process += order_data["工序一"]["上料时间"][mid] + '\t'
            if mid in order_data["工序一"]["下料时间"]:
                process += order_data["工序一"]["下料时间"][mid] + '\t'
                print(process)



def process_output_process2(order_data):
    #遍历工件id
    print('加工物料序号\工序的编号t1CNC\上料开始时间t\下料开始时间t\工序的编
        号t2CNC\上料开始时间t\下料开始时间t')
    for mid in order_data["工序一"]["编号CNC"]:
        process = mid + '\t' + order_data["工序一"]["编号CNC"][mid] + '\t'
        if mid in order_data["工序一"]["上料时间"]:
            process += order_data["工序一"]["上料时间"][mid] + '\t'
            if mid in order_data["工序一"]["下料时间"]:
                process += order_data["工序一"]["下料时间"][mid] + '\t'
                if mid in order_data["工序二"]["编号CNC"]:
```

```
            process += order_data["工序二"]["编号CNC"][mid] + '\t'
            if mid in order_data["工序二"]["上料时间"]:
               process += order_data["工序二"]["上料时间"][mid] + '\t'
               if mid in order_data["工序二"]["下料时间"]:
                  process += order_data["工序二"]["下料时间"][mid] + '\t'
                  print(process)



def process_output_error(order_data):
   print('故障时的物料序号\故障编号tCNC\故障开始时间t\故障结束时间t')
   for mid in order_data["工序一"]["故障编号CNC"]:
      error = mid + '\t' + order_data["工序一"]["故障编号CNC"][mid] + '\t'
      if mid in order_data["工序一"]["故障开始时间"]:
         error += order_data["工序一"]["故障开始时间"][mid] + '\t'
         if mid in order_data["工序一"]["故障排除时间"]:
            error += order_data["工序一"]["故障排除时间"][mid] + '\t'
            print(error)
   for mid in order_data["工序二"]["故障编号CNC"]:
      error = mid + '\t' + order_data["工序二"]["故障编号CNC"][mid] + '\t'
      if mid in order_data["工序二"]["故障开始时间"]:
         error += order_data["工序二"]["故障开始时间"][mid] + '\t'
         if mid in order_data["工序二"]["故障排除时间"]:
            error += order_data["工序二"]["故障排除时间"][mid] + '\t'
            print(error)

o = order(original("testSRGV(stage3).txt"))
process_output_process1(o)
o = order(original("testMRGV(stage3,41).txt"))
process_output_process2(o)

o = order(original("testESRGV(stage3).txt"))
#process_output_process1(o)
process_output_error(o)

#o = order(original("testEMRGV(stage1,85).txt"))
#o = order(original("testEMRGV(stage2,53).txt"))
```

```
o = order(original("testEMRGV(stage3,41).txt"))
#process_output_process2(o)
process_output_error(o)
```