基于多目标遗传算法的 RGV 动态调度优化模型

摘要

本文针对智能 RGV 的动态调度策略设计问题，通过分析工厂生产成本以及生产效率的影响因素，建立最优动态调度的多目标优化模型，利用多次遗传算法对最优解进行搜索，得到最优调度策略和作业效率。最后与 FCFS 原则下的调度模型进行比较，验证了模型的实用性和算法的有效性。

首先，先建立关于调度系统的三个评价指标：生产效率、RGV 能耗以及 CNC 工作效率，然后基于“先到先服务”的原则对四种具体情况 (一道工序无故障、两道工序无故障、一道工序有故障和两道工序有故障的情况)，利用 MATLAB 设计模拟算法并带入题中三组数据进行模拟。计算出各情况下三个评价指标的值，发现 RGV 移动距离、物料完成量与时间成正比，且出现故障对一道工序影响较小，对两道工序的生产影响较大。

对于一道工序无故障的情况，由于直接对整段作业情况进行分析计算量过于庞大，所以采用将整段情况分成多个小段情况进行最优调度策略的求解。通过前面对动态调度模型影响因素的分析，得出三个优化目标：物料加工完成时间最小、RGV 移动距离最小、CNC 工作效率最高，结合工作时间以及 CNC 上熟料判断的约束，以当前时刻规划的下 n 步线路为决策变量，建立智能加工系统的动态调度多目标优化模型，将多个优化目标通过标准化和归一并线性加权变成单优化目标。然后利用遗传算法题目对所给 3 组数据中 RGV 每段时间的活动路径进行搜索，最后得到整段的最优调度策略和作业效率。相比于 FCFS 方案，物料总量平均提升了 30 个左右，但也导致了 RGV 移动距离平均增加了 20 个单位左右，CNC 的工作效率也平均提高了大约 6 个百分点。此情况下的平均系统的作业效率为 47.91 个/小时，相比于 FCFS 提升了 3.7 个/小时。调度策略见附件三。

对于两道工序无故障的情况，相比于一道工序的情况，需要额外对 RGV 的移动路径进行约束。所以在一道工序的目标优化模型的基础上，将 CNC 工作效率的目标函数以及工作时间的约束通过对两道工序分析进行了修改，并增加了关于不同物料对下一个目标 CNC 的约束。同样的将多目标化为单目标后利用遗传算法进行求解。相比于 FCFS 来说，前两组物料的总量平均提升了 40 个左右，但第三组的总量却减少了，可能是因为遗传算法陷入了局部最优解，RGV 的移动距离前两组平均增加了 50 个单位，增幅较小，CNC 的工作效率提高了大约 10 个百分点。此情况下的平均系统的作业效率为 25.75 个/小时，相比于 FCFS 提升了 2.37 个/小时。调度策略见附件三。

对于一道工序有故障的情况，我们在无故障模型的基础上进行修改，设置故障系数，新增了对于 CNC 损坏的约束条件，并对优化目标进行了修改，得到新的多目标优化调度模型。同样利用遗传算法求解得到结果。相比于 FCFS 来说，物料总量平均增加了 12 个左右，RGV 移动距离平均增加大约 15 个单位，增幅较小，CNC 工作效率平均提高了大约 2 个百分点。此情况下平均系统的作业效率为 45.16 个/小时，相比于 FCFS 提升了 1.66 个/小时。调度策略见附件三。

对于两道工序有故障的情况，同样在两道工序无故障的模型上进行修改，设置故障系数，增加 CNC 损坏的约束条件，修改优化目标，得到新的目标优化模型，利用遗传算法求解得到结果。相比于 FCFS，物料总量平均提高了大约 30 个，第一组 RGV 移动距离减少了 60 个单位距离，效果很好，三组平均增加了大约 20 个单位距离，CNC 工作效率平均提高了大约 15 个百分点。此情况下平均系统的作业效率为 24.42 个/小时，相比于 FCFS 提升了 3.55 个/小时。调度策略见附件三。

通过对四种情况三组数据的结果，有力的证明了优化模型的实用性和算法的有效性。

关键字： 多目标优化　遗传算法　动态调度

一、 问题重述

轨道自动引导车（Rail Guided Vehicle，RGV）是一种无人驾驶、能在固定轨道上自由运行的智能车，用于智能加工系统。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。

针对下面的三种具体情况：

(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（发生概率约为 1%）的情况，每次故障排除（人工处理，未完成的物料报废）时间介于 10 ∽20 分钟之间，故障排除后即刻加入作业序列。分别考虑一道工序和两道工序的物料加工作业情况。

完成下列两项任务：

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. 利用所给系统作业参数的 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，并将具体的结果以 EXCEL 表形式展示。

二、 问题分析

轨道自动引导车在车间的调度问题就是要解决如何利用有限的资源在满足工厂各种生产约束的前提下，确定工件和设备的加工顺序和时间，使工厂的生产效率提高，提高工厂的竞争力。然而由于在实际的生产调度过程中，不会单纯的只考虑单个目标，多数情况都需要同时考虑多个目标，及多目标优化问题。所以建立一个优秀的 RGV 动态调度模型是非常重要的。

为了研究建立优秀的 RGV 动态调度模型，首先需要对工厂智能加工系统传统的调度方式进行探究。在早期的车间调度算法中，大多是基于优先指派的近似算法，如以“先到先服务”为基础的 RGV 指派调度算法 [2]。所以可以考虑先对四种具体情况：一道工序无故障、两道工序无故障、一道工序有故障和两道工序有故障的情况利用“先到先服务”的原则，利用题目给的 3 组数据表进行模拟。然后统计整个模拟中，各个情况的物料生产总量、RGV 移动距离等数据，综合分析当前“先到先服务”调度算法产生的问题，总结出设计 RGV 动态调度模型时需要考虑的目标，也便于与之后设计出的调度模型进行比较。

对于一道工序的物料加工作业的最优调度方案，首先需要根据之前的“先到先服务”模型分析出优秀调度方案需要达到的目标，如果有多个目标则将其归一化。然后便可以利用启发式算法对单个连续作业的 8 小时进行寻找，但是这样产生的问题非常明显，由于时间跨度过大，启发式算法的计算量非常大，无法在有限的时间内寻找出 RGV 的最优路径，并且容易陷入局部最优解。于是，我们可以考虑将一次连续作业分为多段时间段，按时间顺序对每段时间段分别使用启发式算法中的遗传算法进行求解并记录各物料加工的 CNC 的编号和上下料时刻，这样能够保证各段时间求出的最优路径能够衔接。当然，为了让遗传算法更加有效，还应该考虑 RGV 和 CNC 运行过程中的约束，最

后即可完成对最优调度模型的建立和求解，并与 FCFS 模型的相同情况进行比较，检验模型的实用性和算法的有效性。

对于两道工序的情况，需要额外对 RGV 的移动路径进行约束。在 RGV 持有不同的物料时，它下一步能够前往的 CNC 就不同。所以在一道工序模型的基础上，需要加上对 RGV 持有物料的判断，并增加对应的约束。同样的，将整个加工过程分为多个小段，对每个小段使用遗传算法求解，在每个时间段最优 RGV 路径寻找完成后，输出物料编号、物料加工的 CNC 的编号，以及各自上下料的时刻。最后将模型得到的结果与 FCFS 模型的结果进行比较，检验模型的实用性和算法的有效性。

对于可能发生故障的情况，同样只需要在无故障模型的基础进行改进。将加工系统整段运行时间分为小段后，分别利用遗传算法寻找 RGV 最优路径。CNC 在加工的过程中有 1% 的机率发生故障，为了模拟这样的情况，可以在每段遗传算法中的每个 CNC 上料之前进行故障的判断，若产生故障，则记录故障开始时间，并生成一个故障修复时间。在遗传算法的搜索中，将其考虑进去即可。最后将模型得到的结果与 FCFS 模型的结果进行比较，检验模型的实用性和算法的有效性。

三、 模型假设

1. 假设工序是有序的，每个 CNC 在任何时刻都只加工一个物料；
2. 所有 CNC 准备时间为零，即在空闲情况下物料能立即进入加工；
3. 各物料之间开始进行第一道加工时没有有先权；
4. 假设上下料传送带在运行中速度均匀；
5. CNC 加工物料时才会故障，并且故障时刻服从平均分布，在完成工序时间内每一刻发生故障概率相等；
6. 修理 CNC 的时间服从平均分布，在 10 分钟和 20 分钟之间任一时刻修理完成的概率相等。

四、 名词解释与变量说明

变量名称	含义
q	模拟运行过程中物料队列
p	物料清完完的时刻
$p_{j\#}$	上次去 j 号上料结束的时刻
x_{ij}	熟料系数
x'_{ij}	第二道工序熟料系数
g_{ij}	故障系数
Pos_i	Pi 时刻 RGV 的位置
d_{ij}	RGV 在 i,j 号 CNC 之间的移动距离
t_d	RGV 的移动时间
t_c	上下料时间
t_w	清洗时间

五、 建模前准备

带 RGV 的智能加工系统是一个运行机理清晰的系统，为了建立优秀的 RGV 动态调度模型，设计出符合工厂利益的 RGV 动态调度方案，首先需要对智能 RGV 调度系统进行研究，进而寻找出现行 RGV 调度系统的缺点，然后才能有针对性的对现行方案进行修正。

为了对生产过程中，不同的调度方式进行评价，需要先建立关于调度系统的评价指标，只有在生产调度的过程中综合考虑各个目标，才能建立优秀的调度模型，然后产生合适的调度决策：

5.1 评价指标建立

通过查阅相关文献 [3]，可以将生产车间的调度问题分为以下几类：

(1) 基于系统生产效率的指标

工厂生产零件的首要目标便是较高的生产效率。单位时间内生产的物料越多，说明工厂下效率越高，反之，工厂的加工速度缓慢，有可能不能完全满足客户的需求，不能及时交货，从而影响工厂的利益。所以生产效率是生产调度是否良好的重要标志。

(2) 基于 RGV 能耗的指标

对于工厂来说，每日的生产消耗是特别巨大的，特别是机器加工使用的能源。如果能够有效的减少能源的消耗，能够给工厂节约一大笔成本费用。对于 CNC 来说，耗能与加工物料的数量成正比，无法对其进行节能的设计。而对于 RGV，若能够尽量减少 RGV 的移动，便能够有效的节约能源，减少工厂的支出。所以 RGV 的能耗也是生产调度是否良好的重要标志。

(3) 基于 CNC 设备的工作效率

在工厂整体处于运营的过程中，CNC 是处于一直耗能的状态，即使 CNC 在某段时间内不进行生产。为了节约能源，使能源能够尽可能的被使用而不是被浪费，需要使 CNC 尽量一直处于有效生产的状态，即提高 CNC 的工作效率。这样能够有效的对能源进行使用，节约了工厂的成本。CNC 设备的工作效率也是生产调度是否良好的重要标志。

一个动态调度系统最核心的部分便是生产作业排序。生产作业排序是企业制定生产作业计划的重要问题，生产作业排序既包括确定工件的加工顺序，又包括确定每台机器设备加工每个工件的开始和完成时间。在早期的工厂车间调度算法中，大多数都是基于优先指派规则的近似算法，它是根据事先安排好的规则从所有的调度集合的子集中找到一个准备调度的操作，最典型的便是“先到先服务（FCFS）”规则。它是指根据任务到达的先后次序安排加工顺序，先到先加工。所以可以根据题目所给的一道工序、两道工序的无故障和有故障四种情况，分别进行模拟。

5.2 一道工序无故障情况

在一道工序无故障的情况下，每个待加工物料只需要通过一次 CNC 车床切割。首先将 RGV 置于 CNC1# 和 CNC2# 中间的位置，所有的 CNC 都处于空闲的状态。在 CNC 处于空闲或者作业完成的状态时，它会向 RGV 发出上料需求信号。RGV 在收到某 CNC 的需求信号后，它会自行确定该 CNC 的上下料作业次序，由于当前使用的是“先到先服务”规则，RGV 会对所有收到信号对应的 CNC 按照需求信号接收顺序排序，然后向

最早收到的需求信号所对应的 CNC 移动，并完成其的下料及上料，然后将已加工的熟料移动到清洗槽上方，进行清洗作业。在完成这项作业任务后，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。而 CNC 在加工完一个物料或者处于空闲状态时，会等待 RGV 而不进行任何操作。

具体的作业流程如下：

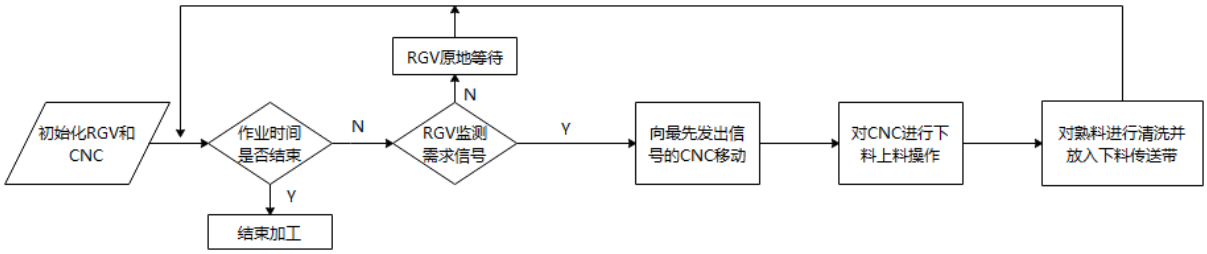


图 **1**　使用 **FCFS** 规则的一道工序无故障情况流程图

通过上述的分析，清晰的展示出了对应情况下加工系统的运行原理。为了进一步进行分析，可以对次流程进行算法的设计，利用 MATLAB 对使用 FCFS 规则的一道工序无故障情况进行模拟。

为了模拟”先来先服务“原则，首先设置一个 CNC 的请求队列 q，当某个 CNC 处于空闲或已加工熟料的状态时会向 RGV 发送需求，这时此 CNC 从队尾入队。由于 CNC 数量的限制，最大会有 8 个 CNC 在队列 q 中。将 RGV 的活动区域从沿着传送带的方向依次设为 1 到 4 号位置。设每一次对 CNC 上料并清洗后的时刻为 $P_1, P_2, ...P_i$；最近一次前往 j 号 CNC 上料结束的时刻为 $P_{j\#}$；X_{ij} 表示 j 号 CNC 在 P_i 时刻的熟料情况：

$$X_{ij} = \begin{cases} 1, & \text{CNC 上存在熟料} \\ 0, & \text{CNC 上不存在熟料} \end{cases}$$

为了计算出 8 小时内工厂物料的生产量，需要计算 RGV 在不同的 CNC 间移动时所消耗的时间。而不同距离的移动消耗的时间各不相同，通过对 CNC 分布位置以及各 CNC 间距离的分析，可以得到从 i 号 CNC 移动到 j 号 CNC 的距离 d_{ij} 与当前位置 Pos_i(由于 RGV 会处于偶数编号和奇数编号的 CNC 之间，为了方便计算移动距离，将临近 RGV 且编号为奇数的 CNC 的编号作为 RGV 的当前位置 Pos_i) 以及目标 j 号 CNC 所在位置的关系如下：

$$d_{ij} = \begin{cases} \dfrac{|j - Pos_i|}{2}, & \text{j 为奇数} \\ \dfrac{\lfloor (2j-1)/2 \rfloor}{2}, & \text{j 为偶数} \end{cases}$$

使用 FCFS 规则的一道工序无故障情况的具体算法如下：

Step1　首先对 RGV 和 8 个 CNC 进行初始化。由于 8 个 CNC 都处于空闲的状态，所以将 8 个 CNC 放入请求队列，置为：$q = \{1, 2, 3, 4, 5, 6, 7, 8\}$。初始位置 $Pos_0 = 1$，处于 1 号 CNC 和 2 号 CNC 中间。

Step2　RGV 通过队列判断下一步移动的目的地，由“先来先服务”的原则，RGV 向队列头部的目标 j 号 CNC 移动，初始队列中为 1 号 CNC，同时计算 RGV 的移动距离 d_{ij}。

Step3　RGV 移动到目标 j 号 CNC 后，需要将未加工的生料上料，若 j 号 CNC 存在已加工的熟料，则上下料完成后对熟料进行清洗并移动到下料传送带上。对于 P_1 来

说，它等于 RGV 移动到队列头部的 1 号 CNC 的距离所需要的时间 t_d^1、上料到消耗的时间 t_c^1 以及熟料的清洗时间 t_w 与判断熟料的系数 X_{ij} 之积、开始时刻 P_0 之和:

$$P_1 = t_d^1 + t_c^1 + X_{11}t_w + P_0$$

对于 P_1 来说，1 号 CNC 上没有熟料，所以 $X_{11} = 0$。同理，可以推出第 i 次上料并清洗后的时刻 P_i 为：

$$P_i = t_d^i + t_c^j + X_{ij}t_w + P_{i-1}$$

其中 P_{i-1} 表示 $i-1$ 次上料并清洗后的时刻。

Step4 计算完成后将队列顶端的 CNC 出队。

Step5 若队列 q 不为空，则回到 Step2；若队列为空，则判断确定新的请求队列。此时监测各 CNC 是否发送需求信号，通过判断下列式子的大小关系实现：

$$P_i - (P_{j\#} + 560), \quad j = 1, 2, ..., 8$$

其中 P_i 代表当前上下料以及对下料清洗完毕后的时刻，它减去加工时间以及加工开始时刻之和，等于当前 CNC 加工完毕到 RGV 对其进行上下料之间的等待时间。当等待时间大于 0 时，代表 j 号 CNC 已经进入队列 q；等待时间小于 0，则代表加工尚未完成。如果所有的 CNC 等待时间小于 0，则 RGV 没有收到需求信号，原地等待，直到收到信号，将对应的 CNC 入队，回到 Step2。如果收到信号，先将收到的信号按照先后顺序入队 q，先收到的先入队，回到 Step2。

5.3 一道工序有故障情况

对于一道工序有故障的情况，CNC 会在生产过程中发生故障，概率为 1%，每次故障需要人工排除，消耗 10 至 20 分钟并将未完成的物料报废。所以需要对故障情况进行考虑。通过设置故障系数 g_{ij} 和修改等待时间的表达式来实现，当 CNC 故障时，修改后的式子恒小于 0，无法进入队列。当完成 CNC 的修复之后，将其熟料系数置 0，并进入排队序列继续参加生产。

利用 MATLAB 模拟此过程的具体算法如下：

Step1 对 RGV 和 8 个 CNC 进行初始化。请求队列置为：$q = \{1, 2, 3, 4, 5, 6, 7, 8\}$。初始位置 $Pos_0 = 1$。

Step2 RGV 通过队列判断下一步移动的目的地，由“先来先服务”的原则，RGV 向队列头部的目标 j 号 CNC 移动，计算 RGV 的移动距离 d_{ij}。

Step3 同一道工序的情况，可以推出第 i 次上料并清洗后的时刻 P_i 等于 RGV 移动到队列头部的 i 号 CNC 的距离所需要的时间 t_d^i、上料到消耗的时间 t_c^i 以及熟料的清洗时间 t_w 与判断熟料的系数 X_{ij} 之积、$i-1$ 次上料并清洗后的时刻开始时刻 P_{i-1} 之和:

$$P_i = t_d^i + t_c^j + X_{ij}t_w + P_{i-1}$$

Step4 计算完成后将队列 q 顶端的 CNC 出队。

Step5 若队列 q 不为空，则回到 Step2；若队列为空，则判断确定新的请求队列。此时监测各 CNC 是否发送需求信号，通过判断下列式子与 0 的大小关系实现，但由于

CNC 有 1% 的机率发生故障，所以需要对故障进行考虑。设 P_i 时刻 j 号 CNC 的故障系数为 g_{ij}：

$$g_{ij}=\begin{cases}1, & \text{j 号 CNC 在 } P_i \text{ 时刻故障}\\ 0, & \text{j 号 CNC 在 } P_i \text{ 时刻正常运行}\end{cases}$$

且故障的概率 $P(g_{ij}=1)=0.01$，不故障的概率 $P(g_{ij}=0)=0.99$。然后对原等待时间的表达式进行处理后得：

$$(1-g_{ij})[P_i-(P_{j\#}+560)]-0.5g_{ij},\quad j=1,2,...,8$$

通过对等待时间进行修改，能够有效避免损坏的 CNC 进入队列。当 CNC 正常运行，上式大于 0，对应的计算结果与无故障情况相同，都等于等待时间；当对应的 CNC 故障，上式结果恒小于 0，此 CNC 无法进入队列。CNC 的恢复时间则利用随机数随机生成 10 到 20 分钟，在 CNC 恢复后，将其入队并置熟料系数 $X_{ij}=0$。如果所有的 CNC 等待时间小于 0，原地等待，直到收到信号，将对应的 CNC 入队，回到 Step2。如果收到信号，则顺序入队 q，回到 Step2。

5.4 两道工序无故障情况

对于两道工序来说，基本原则还是“先到先服务”。通过对题目的分析，可以得知，负责不同道工序的 CNC 数量、分布不同，都会导致不同的结果，为了能够研究各种调度方案的优劣性，让负责不同道工序的 CNC 数量、分布相同，去除了负责不同道 CNC 数量和分布对研究不同调度方案的影响。因此得出符合情况的安排为：1、3、6、8 号 CNC 负责第一道工序，其余 CNC 负责第二道工序。对于第一道工序来说，在 CNC 存在第一道熟料时，RGV 在取出熟料后不需要清洗，并且由于机械手上拿着第一道熟料，必须前往负责第二道工序的 CNC；而当 CNC 不存在第一道熟料时，则不能进行第二道工序的加工，必须前往第一道工序的 CNC；在前往第二道工序的 CNC 后，无论是否得到第二道熟料，都会保持机械臂为空，下一步需要前往第一道工序的 CNC。综上分析，得到二道工序 RGV 的移动规律如下：

1. 对于负责第一道工序的 CNC，若存在第一道熟料，则 RGV 完成上下料后前往负责第二道工序的 CNC；
2. 对于负责第一道工序的 CNC，若不存在第一道熟料，则 RGV 完成上料后前往负责第一道工序的 CNC；
3. 对于负责第二道工序的 CNC，完成上下料后前往负责第一道工序的 CNC。

通过上述分析，可以得到具体算法如下：

Step1 对 RGV 和 8 个 CNC 进行初始化。请求队列置分为 q_a 和 q_b，分别表示第一道工序和第二道工序。通过题目易分析出较好的 CNC 配置方案为：$A=\{1,3,6,8\}$，$B=\{2,4,5,7\}$，则对应的初始队列就为：$q_a=\{1,3,6,8\}$，$q_b=\{2,4,5,7\}$。初始位置 $Pos_0=1$。

Step2 RGV 通过队列判断下一步移动的目的地并计算移动距离。最初时，RGV 向队列 q_a 头部的目标 j 号 CNC 移动，其余时刻的移动同上面的判断。计算 RGV 的移动距离 d_{ij}。

Step3 利用 RGV 移动到队列 q_a 头部的 i 号 CNC 的距离所需要的时间 t_d^i、上料到消耗的时间 t_c^i 以及熟料的清洗时间 t_w 与判断熟料的系数 X_{ij} 之积、$i-1$ 次上料并清

洗后的时刻开始时刻 P_{i-1} 求解第 i 次上料并清洗后的时刻 P_i:

$$P_i = t_d^i + t_c^j + X_{ij}t_w + P_{i-1}$$

Step4 计算完成后将第一道工序队列 q_a 顶端的 CNC 出队。

Step5 这时对 RGV 的下一个去向进行判断。通过 RGV 的移动规律，如果当前负责第一道工序的 CNC 上不存在加工完成的第一道熟料，RGV 向第一道工序的 CNC 移动，将此 CNC 从 q_a 前端出队，跳到 Step6；若存在加工完成的第一道熟料，则 RGV 向第二道工序的 CNC 移动，处理第二道工序的队列 q_b。

Step6 利用同样的 RGV 移动距离确定方法确定当前位置与队列 q_b 前端第一个 CNC 的距离。

Step7 对第二道工序队伍 q_b 前端对应的 CNC 进行结束时刻 P_{i+1} 的计算。结束时刻 P_{i+1} 相比于 P_i，需要加上 RGV 移动时间 t_d^{i+1}、上下料时间 $t_c^{j'}$、及判断第二道熟料的系数 $X'_{i+1j'}$ 与清洗时间的乘积：

$$P_{i+1} = t_d^{i+1} + t_c^{j'} + X'_{i+1j'}t_w + P_i$$

Step8 计算完成后将第二道工序队列 q_b 顶端的 CNC 出队。

Step9 若队列 q_a 或 q_b 不为空，则回到 Step2；若队列为空，则判断确定新的请求队列。此时监测各 CNC 是否发送需求信号，同样通过判断等待时间的正负关系来确定新的队列。对等待时间为正的 CNC 从大到小的排序，然后进行入队处理，回到 Step2。若不存在等待时间为正的 CNC，则 RGV 原地等待。

5.5 两道工序有故障情况

而大豆工序有故障的情况很容易从前面的分析可以得到，以二道工序无故障的情况为基础，结合一道工序有故障时设置的故障系数 g_{ij}，以及 RGV 在二道工序时的移动规律，便可以得出二道工序有故障情况的具体算法：

Step1 对 RGV 和 8 个 CNC 进行初始化。第一道工序和第二道工序请求队列置分为 q_a 和 q_b。初始队列为：$q_a = \{1, 3, 6, 8\}$，$q_b = \{2, 4, 5, 7\}$。初始位置 $Pos_0 = 1$。

Step2 RGV 通过队列判断下一步移动的目的地并计算移动距离。同理计算 RGV 的移动距离 d_{ij}。

Step3 计算第 i 次上料并清洗后的时刻 P_i:

$$P_i = t_d^i + t_c^j + X_{ij}t_w + P_{i-1}$$

Step4 将第一道工序队列 q_a 顶端的 CNC 出队。

Step5 这时对 RGV 的下一个去向进行判断。通过 RGV 的移动规律，如果当前负责第一道工序的 CNC 上不存在加工完成的第一道熟料，同时设置熟料标志 $flag = 0$，将此 CNC 从 q_a 前端出队，跳到 Step6；若存在加工完成的第一道熟料，熟料标志 $flag = 1$，RGV 处理第二道工序的队列 q_b，。

Step6 利用同样的 RGV 移动距离确定方法确定当前位置与队列 q_b 前端第一个 CNC 的距离。

Step7 对第二道工序队伍 q_b 前端对应的 CNC 进行结束时刻 P_{i+1} 的计算。结束时刻 P_{i+1} 相比于 P_i，需要加上 RGV 移动时间 t_d^{i+1}、上下料时间 $t_c^{j'}$、及判断第二道熟料的系数 $X'_{i+1j'}$ 与清洗时间的乘积：

$$P_{i+1} = t_d^{i+1} + t_c^{j'} + X'_{i+1j'}t_w + P_i$$

Step8 计算完成后将第二道工序队列 q_b 顶端的 CNC 出队。

Step9 若队列 q_a 或 q_b 不为空，则回到 Step2；若队列为空，则判断确定新的请求队列。此时监测各 CNC 是否发送需求信号，由于 CNC 有 1% 的概率发生故障，同样设置故障系数 g_{ij}。故障的概率通过 MATLAB 的随机数函数实现。当 CNC 出现故障，未加工完成的物料报废，X_{ij} 置为 0，故障系数 $g_{ij} = 1$。同样的，可以得到原等待时间的表达式进行处理后的式子：

$$(1 - g_{ij})[P_i - (P_{j\#} + 560)] - 0.5g_{ij}, \quad j = 1, 2, ..., 8$$

当某 CNC 上式大于 0 时，CNC 准备入队；小于 0 时，不入队。当 CNC 恢复后，置其恢复系数 X_{ij} 或 X'_{ij} 为 0，并入队。CNC 会有对于等待时间为正的 CNC 从大到小的排序，然后进行入队处理，回到 Step2。若不存在等待时间为正的 CNC，则 RGV 原地等待。

5.6 FCFS 情况结果分析

通过对四种不同情况的加工情况进行分析，利用 FCFS 规则进行模拟，首先得到了不同情况下物料的生产具体数据，具体数据见附件一，MATLAB 源程序见附录 A。

将四种情况 RGV 移动总距离，物料完成个数随时间的变化绘制成折线图如下：

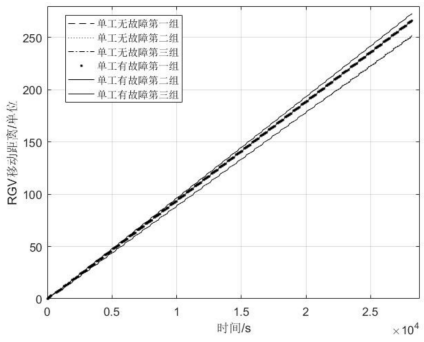


图 **2** 一道工序 **RGV** 移动总距离的变化

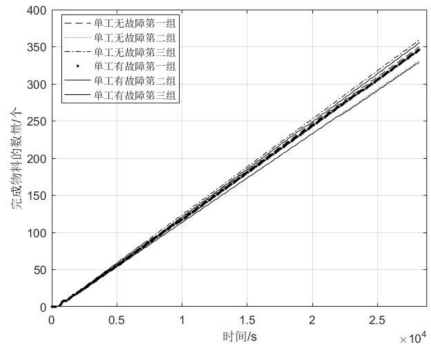


图 **3** 一道工序物料完成个数的变化

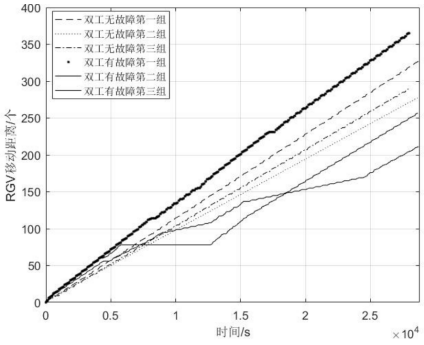


图 **4** 两道工序 **RGV** 移动总距离的变化

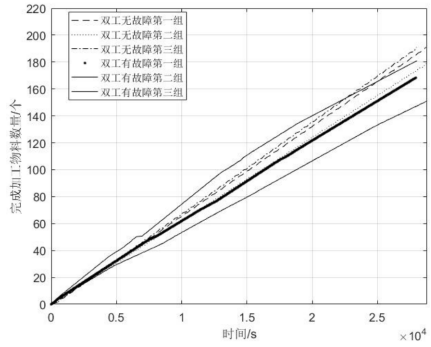


图 **5** 两道工序物料完成个数的变化

通过上面四张图中有故障、无故障对应的一道工序、两道工序情况的比较，可以得到如下信息：

(I) 一道工序中 RGV 的移动距离与时间成正比，并且故障对其移动距离的影响很小；而二道工序中有两组数据在有故障时，RGV 停止移动了很长时间，浪费了生产时间，对工厂生产不利。

(II) 无论是一道工序还是两道工序，无故障情况生产总量都比有故障情况生产总量多。

(III)

接着从 8 个小时总的数据对 FCFS 不同情况下的结果进行分析，作出柱状图进行比较：

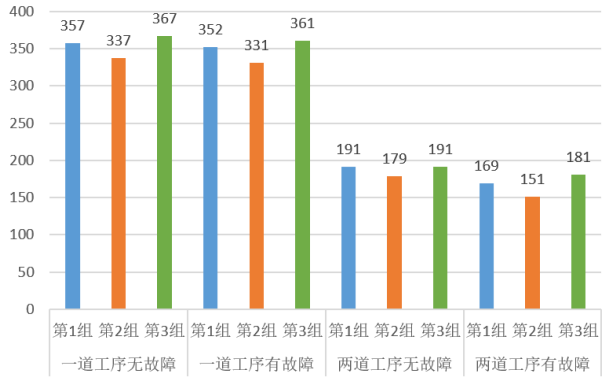


图 **6**　四种情况生产物料数量

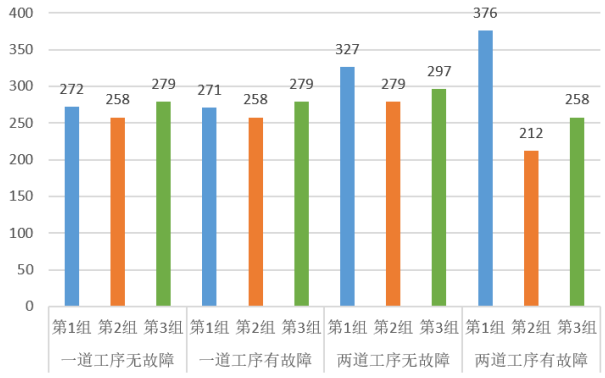


图 **7**　四种情况 **RGV** 移动路程

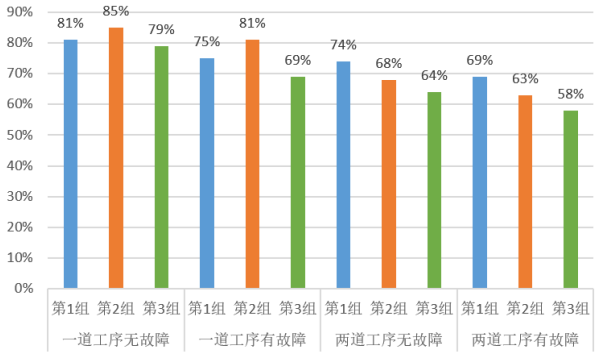


图 **8**　四种情况 **CNC** 工作效率

根据上面的三张图四种情况生产物料数量、RGV 移动路程以及 CNC 工作效率的比较，可以得到如下信息：

(I) 在相同的 8 个小时内，两道工序生产物料的数量均少于一道工序生产物料的数量。

(II) 在一道工序的生产中，故障对生产数量的影响很小；并且故障对两道工序的影响比一道工序的影响大。

(III) 一道工序下的 CNC 生产效率普遍高于两道工序中 CNC 生产效率。

六、 模型的建立与求解

在日常的智能加工系统的生产调度过程之中，工厂会考虑多方面的因素，比如物料的生产总量、生产效率、车间生产的能量消耗和计算机数控机床 CNC 的工作效率等，这些多个调度的影响因素之间相互影响、相互制约，对最优车间生产调度方案的设计有着很大的影响，加大了调度的难度，影响到工厂的生产成本和效率以及客户的满意程度，只有解决多目标的调度问题才能给工厂和客户带来最大的价值。智能加工系统动态调度的关键是找到有效的方法，能够使得多个目标折中解决。[3]

由于对 8 个小时的车间动态调度情况进行模拟情况很复杂，所以可以先将 8 个小时分为多个小段的时间，之后再对每个小段的进行最优化的求解，然后将方案结合起来，便能得到最终的动态调度模型，能有效的简化动态调度方案的设计问题。取当前时刻为 P_m，P_m 表示 RGV 在第 m 次完成上料并清洗后的时刻。在 P_m 时刻，设 RGV 规划的下 n 步线路为 $a_0 \to a_1 \to a_2 \to ... \to a_n$，$a_i$ 表示当前时刻第 i 步到达的 CNC。

6.1 一道工序无故障情况

6.1.1 智能加工系统动态调度模型的建立

针对智能加工系统的动态调度问题，综合考虑了实际生产过程中遇到的问题，从完成时间、RGV 总移动距离和 CNC 的工作效率 3 个方面建立多目标优化模型。完成时间用加工 n 个物料的最短时间 $\sum_{i=1}^{n-1} T_{i,i+1}$ 表示，其中 $T_{i,i+1}$ 表示完成第 i 个物料下料、清洗的时间与完成第 $i+1$ 个物料下料、清洗时间的差值，即为第 $i+1$ 个物料加工时间；RGV 的移动距离目标用 RGV 在加工 n 个物料时间内移动的总距离来表示；各 CNC 的工作效率则用 CNC 在 RGV 在加工 n 个物料的时间与各 CNC 工作时间 $t_{j\#}$ 来表示，j 表示 CNC 的编号，$j = 1, 2, ..., 8$。

通过这三个目标建立最优的动态调度模型是非常复杂的，一方面要考虑到实际情况的实施的困难，另一方面又要把各部分之间的联系协调好，接下来进行详细的介绍：

目标一：物料加工完成时间

在动态调度的过程中，物料加工的完成时间表示工厂生产的效率，也表现出动态调度系统的优劣性。生产物料一定时，完成时间越短说明动态调度的效果越好，效率越高，能够为工厂带来的效益也就越大。

在 RGV 线路中，a_i 步操作完成到 a_{i+1} 步操作完成之间，RGV 首先从 a_i 号 CNC 移动到 a_{i+1} 号 CNC，移动耗时设为 $t_{i,i+1}$。然后 RGV 可能会需要等待 a_{i+1} 号 CNC 生产的完成，等待时间为完成工序需要的时间 t_f^{i+1} 减去已经此工序实际运行时间 t_{work}^{i+1}。其中，在一道工序的情况下，完成工序需要的时间都相同；在两道工序的情况下，负责第一道工序的 CNC 和负责第二道工序的 CNC 完成工序所需要的时间不同：

$$t_f^{i+1} = \begin{cases} t_4, & \text{一道工序的情况} \\ t_5, & \text{两道工序中负责第一道工序的情况} \\ t_6, & \text{两道工序中负责第二道工序的情况} \end{cases}$$

当完成工序需要的时间 t_f^{i+1} 小于此工序实际运行时间 t_{work}^{i+1}，则等待时间为 0。接着 RGV 对此 CNC 进行上下料的操作，耗时为 t_c^{i+1}，其值为：

$$t_c^{i+1} = \begin{cases} t_7, & \text{CNC 编号为奇数} \\ t_8, & \text{CNC 编号为偶数} \end{cases}$$

其中，$t_7 < t_8$。最后，如果有换下来的熟料，则需要加上熟料清洗时间 t_w。是否存在熟料可以利用熟料系数 X_{i+1} 来判断。于是，a_i 步操作完成到 a_{i+1} 步操作完成的时间间隔为：

$$t_{i,i+1} + (t_f^{i+1} - t_{work}^{i+1}) + t_c^{i+1} + X_{i+1}t_w$$

因此物料加工完成时间的目标函数是使物料的总的完成时间最小，也就是：

$$\min F_1 = \sum_{i=1}^{n-1} T_{i,i+1} = \sum_{i=0}^{n-1} [t_{i,i+1} + (t_f^{i+1} - t_{work}^{i+1}) + t_c^{i+1} + X_{i+1}t_w] \tag{1}$$

目标二：RGV 移动距离

在工厂智能加工系统运行过程中，RGV 作为执行动态调度的中心，需要消耗大量的能源进行移动，以及对生料进行上料，对熟料进行下料以及清洗的操作。为了节约工厂生产的成本，需要尽可能的减少能源的消耗。由于上下料以及物料清洗的操作与物料加工数量成正比，无法对其进行优化，所以需要尽量减少 RGV 的移动来节约成本。因此 RGV 移动距离的目标函数为：在完成 n 个物料的最短时间内，使 RGV 总的移动距离最小。目标函数等于每次加工前 RGV 移动的距离 $d_{i,i+1}$ 之和，即：

$$\min F_2 = \sum_{i=0}^{n-1} d_{i,i+1} \tag{2}$$

目标三：CNC 工作效率

CNC 在整个智能加工系统中负责对物料进行加工，且在加工系统运转的过程中一直运行，一直有着对能源的消耗。同样为了节约能源，需要保证在完成 n 个物料的最短时间内，让 8 台 CNC 总工作时间和总时间之比最大，即 CNC 工作效率最大。因此 CNC 工作效率的目标函数为， 8 个 CNC 的工作效率最大。

为了求出工作效率，需要先求出 8 台 CNC 总的工作时间。对于此刻 P_m 来说，CNC 的工作时间为现在未完成的 CNC 加工到加工完成的时间，加上 P_m 之后开始的 CNC 所有的工作时间之和，分别称为前段加工时间和后段加工时间。对于当前时刻仍然在工作的 CNC 来说，它到本次工作结束时所消耗的时间为本次生产开始时刻 $P_{i\#}$ 加上生产需要时间 t_4 再减去已经工作的时间 P_m，然后求和，即：

$$\sum_{k=1}^{l} P_{i\#} + t_4 - P_m$$

其中 l 表示 P_m 时正在加工的 CNC 的数量。

为了求解将 P_m 时刻后的工作时间，首先将加工的步骤设为 a_0 到 a_n，计算 a_0 到 a_n 中各个 CNC 开始加工的频数 f_1 到 f_8。对于每个 CNC 来说，当前时间段内最后一个加工物料可能完成也可能未完成，且每个 CNC 至多有 1 个物料未完成。所以对于每个 CNC，都有后段加工时间为：

$$t_4(f_j - 1) + \begin{cases} t_4, & P_e - P_{i\#} > t_4 \\ P_e - P_{i\#}, & P_e - P_{i\#} \le t_4 \end{cases}$$

其中，P_e 表示当前加工步骤 a_0 到 a_n 所对应的结束时间。所以当前时间段内，j 号 CNC

的加工时间 T_{CNC}^{j} 为：

$$T_{CNC}^{j}=\begin{cases}\sum_{k=1}^{l}P_{i\#}+t_4-P_m+t_4(f_j-1)+t_4, & P_e-P_{i\#}>t_4\\ \sum_{k=1}^{l}P_{i\#}+t_4-P_m+t_4(f_j-1)+P_e-P_{i\#}, & P_e-P_{i\#}\le t_4\end{cases}$$

其中，t_4 表示加工完成一个一道工序的物料所需时间。利用 CNC 总的加工时间除以总时间，便可以得到 CNC 工作效率：

$$\max F_3=\frac{\sum_{j=1}^{8}T_{CNC}^{j}}{8\sum_{i=1}^{n-1}T_{i,i+1}} \tag{3}$$

决策变量与约束条件

在本文的智能加工系统动态调度问题中，希望能够让物料加工完成时间、RGV 移动距离最小，CNC 工作效率最大，根据上面的具体的分析，可以得到此目标优化模型的决策变量为当前时刻 P_m 规划的下 n 步线路 $a_0\to a_1\to a_2\to ...\to a_n$。

在加工物料过程中有如下的约束条件：

(1) 工作时间的约束

单个 CNC 的实际工作时间因 RGV 的安排而不同。对于时间 P_m 来说，在上一次到达 $P_{i+1\#}$ 对应的 CNC 时，开始了第 $i+1$ 次生产。如果 $i+1$ 步后再一次到达此 CNC 时，生产已经完成，即 i 步后的时刻 P_{m+i} 加上移动到此 CNC 的时间 $t_{i,i+1}$ 减去上次到达的时刻 $P_{i+1\#}$ 小于大于加工时间 t_f^{i+1}，则到 P_{m+i} 时，实际工作时间等于加工所需时间；否则等于再一次到达此 CNC 的时刻 $t_{i,i+1}+P_{m+i}$ 与前一次离开此 CNC 时刻 $P_{i+1\#}$ 之差，即：

$$t_w^{i+1}=\begin{cases}P_m-P_{i+1\#}+\sum_{j=0}^{i-1}T_{j,j+1}+t_{i,i+1}, & P_{m+i}+t_{i,i+1}-P_{i+1\#}<t_f^{i+1}\\ t_f^{i+1}, & P_{m+i}+t_{i,i+1}-P_{i+1\#}\ge t_f^{i+1}\end{cases}$$

(2) 熟料系数的约束

由于在对 CNC 进行上下料的时候，需要判断 CNC 中原来是否存在熟料，存在熟料，则需要对熟料进行清洗，计算加工时间时便需要考虑；而不存在熟料时，则不需要计算清洗时间。熟料系数是为了控制此情况而产生的：

$$X_i=\begin{cases}1, & \text{i 步对应的 CNC 上存在熟料}\\ 0, & \text{i 步对应的 CNC 上不存在熟料}\end{cases}$$

6.1.2 模型汇总

通过上文对动态调度模型的分析，对于下 n 步线路的选择问题，建立多目标优化模型如下：

$$\min \quad F_1 = \sum_{i=0}^{n-1}[t_{i,i+1} + (t_f^{i+1} - t_{work}^{i+1}) + t_c^{i+1} + X_{i+1}t_w]$$

$$\min \quad F_2 = \sum_{i=0}^{n-1} d_{i,i+1}$$

$$\max \quad F_3 = \frac{\sum_{j=1}^{8} T_{CNC}^{j}}{8\sum_{i=1}^{n-1} T_{i,i+1}}$$

$$s.t. \begin{cases} t_w^{i+1} = \begin{cases} P_m - P_{i+1\#} + \sum_{j=0}^{i-1} T_{j,j+1} + t_{i,i+1}, & P_{m+i} + t_{i,i+1} - P_{i+1\#} < t_f^{i+1} \\ t_f^{i+1}, & P_{m+i} + t_{i,i+1} - P_{i+1\#} \geq t_f^{i+1} \end{cases} \\ X_i = \begin{cases} 1, & \text{i 步对应的 CNC 上存在熟料} \\ 0, & \text{i 步对应的 CNC 上不存在熟料} \end{cases} \end{cases}$$

优化模型为多目标，需要先将三个目标函数进行无量纲化处理，再采用线性加权法对三个目标函数进行归一化处理，将多目标函数化归为单目标函数[1]：

$$\max \quad \lambda_1 \frac{F_{1max} - F_1}{F_{1max} - F_{1min}} + \lambda_2 \frac{F_{2max} - F_2}{F_{2max} - F_{2min}} + \lambda_3 \frac{F_3 - F_{3min}}{F_{3max} - F_{3min}} \tag{4}$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

其中 λ_1，λ_2，λ_3 代表了每个目标函数的重要程度，三个权重之和等于 1。F_{1max}、F_{2max}、F_{3max} 分别表示 F_1、F_2、F_3 的最大值；F_{1min}、F_{2min}、F_{3min} 分别表示 F_1、F_2、F_3 的最最小值。

6.1.3 模型算法

(1) 算法思想

遗传算法是一种仿照自然界环境对生物种群的优胜劣汰的进化规则的一种进化算法。它具有非常好的全局搜索的能力，而且能够进行并行搜索。在运行算法时，首先需要确定种群的大小，然后根据实际的题目需求设定适应度函数，接着使用和自然界相同的选择、交叉和变异操作对“种群”进行进化。将得到的结果与算法的终止条件进行比较，如果符合终止条件，就退出进化操作，否则继续执行循环，直到符合终止条件，得到最优解。遗传算法对于许多并行搜索的大型复杂问题都能够很好的求解，很适用于本题的智能加工系统动态调度问题。利用遗传算法，可以对整个调度中某段时间的路线进行最优化的求解。

对每一段生产都使用遗传算法求解最优调度方案，然后将各个部分的最优调度方案按照时间进行衔接，最后综合模拟得到最优的整个班次的动态调度最优方案。

(2) 算法步骤

Step1 设每段模拟的步骤长度 $N = 9$，首先取第一段生产的路径 a_0 到 a_9，且为了保持每一段的连贯性，需要保持路径中第一个 CNC 不变，对 8 个小时内所有的生产

路径分组后利用遗传算法确定最终的 RGV 调度方案。

Step2 初步编码。遗传算法的最佳结果的搜索过程是在解空间中进行的。为了仿照自然界中遗传的规律，将决定好的参数进行编码。在本问题中，参数为当前生产的路径，将生产路径编码为 a_n 到 a_{n+9}，其中 $a_i \in [1,8]$，它用来表示 RGV 的移动路径上经过的 CNC，编码结束后的数据结构可以类比为生物上的染色体。

Step3 种群初始化。将编码形成的染色体集合在一起就形成了种群，遗传算法采用随机方法生成 M 个体的集合，该集合称为初始种群，设 $M = 50$，每个种群都代表了当前时间段内不同的 RGV 移动路径。

Step4 设置遗传迭代次数 $C = 2000$，以及迭代循环变量 $c = 0$，用来记录遗传的迭代次数。

Step5 选择操作。根据遗传算法的原理，需要在每一轮的循环中对种群进行筛选，筛选的依据是适应度函数，本问中的适应度函数定为上文求出的目标函数，即式 (9)。选择操作能够将优良的个体直接遗传到下一代，而对于处于劣势的较差的个体，则会大概率被淘汰，这样的操作能够有效的加快进化的进度，提高算法的效率。针对当前问题，我们采用了轮盘赌的选择算法。轮盘赌选择策略，是最基本的选择策略之一，种群中的个体被选中的概率 $P_{select}(b_i)$ 与个体相应的适应度函数的值 $f(b_i)$ 成正比。我们需要将种群中所有个体的适应度值进行累加然后归一化，最终通过随机数对随机数落在的区域对应的个体进行选取，对于单个种群（路径）b_i 来说，它被选中的概率 $P_{select}(b_i)$ 为：

$$P_{select}(b_i) = \frac{f(b_i)}{\sum_{i=1}^{50} f(b_i)}$$

其中 $f(b_i)$ 表示路径 b_i 对应的适应度函数的值。

Step6 交叉操作。所谓交叉运算，是指对两个相互配对的染色体依据交叉概率按某种方式相互交换其部分基因，从而形成两个新的个体。交叉运算在遗传算法中起关键作用，是产生新个体的主要方法。这里的染色体指路径 b。同样先对所有的种群计算适应度函数的值 $f(b)$ 进行计算。对于本问，染色体的交叉即为路径 b 之间进行路径经过的 CNCa 的交换。为了在交叉操作中保留优良基因，规定优秀的染色体只与优秀的染色体进行交叉操作；处于劣势的染色体只能和同类别染色体进行交叉，即适应函数值相近的路径才能进行交叉操作。

Step7 变异操作。为了使遗传操作能够具有局部的随机搜索的能力，需要对种群中个体进行变异操作，通过变异能够有效的扩大种群的种类，防止在寻找最优调度方案的过程中陷入局部最优解。对于本问，同样为了保留优秀的基因，规定优秀的基因，即适应度函数值 $f(b)$ 大的种群 b 尽量不进行变异；而对于较差的基因，则希望有较大的概率进行变异。

Step8 结束一轮遗传迭代，迭代次数 c 自增 1。如果遗传的迭代次数的 c 大于等于 2000，则停止算法循环，记录最优的种群，即最优的 RGV 路径 b_{best}，以及消耗时间 ΔP。计算当前总生产时间 P_m:

$$P_m = \Delta P + P_{m-1}$$

如果当前总生产时间 $P_m \geq 28800s$，即已经超过安排的连续生产时间 8 小时，则退出循环；若总生产时间 $P_m < 28800s$，输出本时间段内加工物料的序号、加工 CNC 的编号，以及上料和下料开始时间。进入下一段生产路径，将此次生产路径

的终点设为下一段生产路径的起点，设置生产路径为 a_{n+9} 到 a_{n+18}，返回 Step3，开始新一轮的遗传算法；否则回到 Step5，继续进行本段路径遗传算法的循环。

6.1.4 模型求解及结果分析

通过对一道工序无故障生产情况的模拟 (MATLAB 源程序见附录 B)，得到了每个物料的上下料开始时间，以第一组为例，调度策略如下表 (完整表格以及其它两组的调度策略见附件三)：

表 1　调度策略部分表

加工物料序号	加工 CNC 编号	上料开始时间	下料开始时间
1	1	0	616
		……	
388	4	28506	停止工作

然后，分别计算了优化模型的三个目标的值，分别与 FCFS 方案进行比较：

表 2　一道工序无故障情况优化方案与 **FCFS** 方案比较

	物料总量（个）		RGV 移动距离（单位）		CNC 工作效率（工作时间/总时间)	
	优化方案	FCFS 方案	优化方案	FCFS 方案	优化方案	FCFS 方案
第一组	388	357	289	272	0.87	0.81
第二组	369	337	276	258	0.89	0.85
第三组	393	367	294	279	0.81	0.79

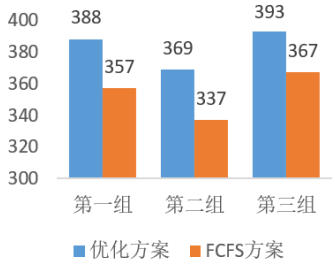


图 9　物料总量比较

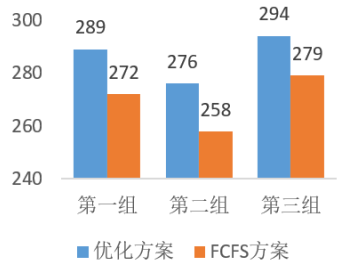


图 10　**RGV** 移动距离比较

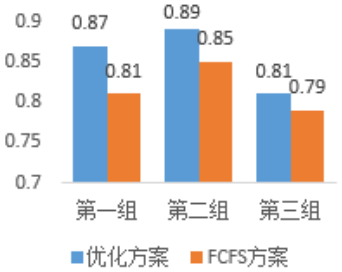


图 11　**CNC** 工作效率比较

通过对三个优化目标的比较，可以看出所设计的调度模型较 FCFS 模型都有很大的提升。对于物料总量，每组数据都提高了 30 个物料左右；对于 RGV 的移动距离，由于物料总量增加，移动距离肯定会增加，但每组只增加了 20 单位左右的移动距离，增幅很小；对于工作效率，每组相比于 FCFS 平均提高了 6 个百分点。整体来说，优化模型相比于 FCFS 模型有较大的提高，很好的提高了工厂的效率，节约了工厂的成本。有力的说明了优化模型的实用性和算法的有效性。

6.2 两道工序无故障情况

对于两道工序无故障的情况，可以在一道工序的情况下，建立修改后的 RGV 动态调度模型。在两道工序中，物料的加工需要依次经历两台不同的 CNC 进行加工。相比于一道工序的情况，需要首先考虑两道工序对应的的 CNC 的安排，然后还需要考虑工序对 RGV 移动路径的约束和影响。

6.2.1 动态调度模型的修改

针对两道工序的的动态调度问题，同样从完成时间、RGV 总移动距离和 CNC 的工作效率 3 个方面建立多目标优化模型。对于每一个时间段最优路径的选取，都有约束和目标。目标优化模型的决策变量为当前时刻 P_m 规划的下 n 步线路 $a_0 \to a_1 \to a_2 \to ... \to a_n$。目标函数只有 CNC 工作效率发生了变化：

目标修改：CNC 工作效率

为了节约能源，保证在完成 n 个物料的最短时间内，每个 CNC 的工作时间与总时间之比最大，即各 CNC 工作效率最大， 8 个 CNC 的工作效率最大。其中，两种 CNC 的单次工作时长不同，需要分别讨论。当前时间段内，对于负责第一道工序的 p 号 CNC 的加工时间 T^i_{CNC1} 为：

$$T^p_{CNC1} = \begin{cases} \sum_{k=1}^{l} P_{i\#} + t_5 - P_m + t_5(f_p - 1) + t_5, & P_e - P_{i\#} > t_5 \\ \sum_{k=1}^{l} P_{i\#} + t_5 - P_m + t_5(f_p - 1) + P_e - P_{i\#}, & P_e - P_{i\#} \le t_5 \end{cases}$$

其中，t_5 表示加工完成一个两道工序物料的第一道工序所需时间。

对于负责第二道工序的 j 号 CNC 的加工时间 T^j_{CNC2} 为：

$$T^j_{CNC2} = \begin{cases} \sum_{k=1}^{l} P_{i\#} + t_6 - P_m + t_6(f_j - 1) + t_6, & P_e - P_{i\#} > t_6 \\ \sum_{k=1}^{l} P_{i\#} + t_6 - P_m + t_6(f_j - 1) + P_e - P_{i\#}, & P_e - P_{i\#} \le t_6 \end{cases}$$

其中，t_6 表示加工完成一个两道工序物料的第二道工序所需时间。利用负责两道不同工序 CNC 总的加工时间除以总时间，便可以得到 CNC 工作效率：

$$\max F_3 = \frac{\sum_{j,p} T^p_{CNC1} T^j_{CNC2}}{8 \sum_{i=1}^{n-1} T_{i,i+1}}, \quad j + p = 8 \tag{5}$$

约束修改：工作时间的约束

工作时间的约束同一道工序的原理，不同的在于每道工序加工时间不同。约束如下：

$$t^{i+1}_w = \begin{cases} P_m - P_{i+1\#} + \sum_{j=0}^{i-1} T_{j,j+1} + t_{i,i+1}, & P_{m+i} + t_{i,i+1} - P_{i+1\#} < t^{i+1}_f \\ t^{i+1}_f, & P_{m+i} + t_{i,i+1} - P_{i+1\#} \ge t^{i+1}_f \end{cases}$$

其中，当第 $i+1$ 步到达的 CNC 为 CNC1，则 $t^{i+1}_f = t_5$；当第 $i+1$ 步到达的 CNC 为 CNC2，则 $t^{i+1}_f = t_6$，即

$$a_{i+1} \in CNC1 \Longrightarrow t^{i+1}_f = t_5$$

$$a_{i+1} \in CNC2 \Longrightarrow t^{i+1}_f = t_6$$

t_5 表示 CNC 加工完成一个两道工序物料的第一道工序所需时间，t_6 表示 CNC 加工完成一个两道工序物料的第二道工序所需时间。

约束增加：不同物料的约束

与一道工序情况不同的是，RGV 每次的移动受到当前 CNC 的影响。设 CNC1 为负责第一道工序的 CNC，CNC2 为负责第二道工序的 CNC。在 RGV 前往有第一道熟料的 CNC1 后，即 $a_i \in CNC1$ 且 $X_{i-1}=1$，RGV 将第一道熟料取下并放上生料。此时，RGV 带着第一道熟料，只能前往负责第二道工序的 CNC2，即下一个目标 CNC：$a_{i+1} \in CNC2$；在 RGV 前往没有熟料的 CNC1 后，即 $a_i \in CNC1$ 且 $X_{i-1}=0$，RGV 将生料放上 CNC1，RGV 为空，不拥有第一道熟料，无法前网 CNC2 进行加工，只能前往 CNC1，即下一个目标 CNC：$a_{i+1} \in CNC1$；在 RGV 前往 CNC2 后，即 $a_i \in CNC2$，若存在第二道熟料，则将其取下并清洗，RGV 为空，不存在熟料时 RGV 也为空。此时 RGV 只能前往 CNC1 进行生料的上料，即下一个目标 CNC：$a_{i+1} \in CNC1$。综上所述，可以得到：

$$\begin{cases} a_i \in CNC1 \cup X_{i-1}=1 \Longrightarrow a_{i+1} \in CNC2 \\ a_i \in CNC1 \cup X_{i-1}=0 \Longrightarrow a_{i+1} \in CNC1 \\ a_i \in CNC2 \Longrightarrow a_{i+1} \in CNC1 \end{cases} \tag{6}$$

6.2.2 模型汇总

通过上文对动态调度模型的分析，对于下 n 步线路的选择问题，建立多目标优化模型如下：

$$\min \quad F_1 = \sum_{i=0}^{n-1}[t_{i,i+1} + (t_f^{i+1} - t_{work}^{i+1}) + t_c^{i+1} + X_{i+1}t_w]$$

$$\min \quad F_2 = \sum_{i=0}^{n-1} d_{i,i+1}$$

$$\max \quad F_3 = \frac{\sum_{j,p} T_{CNC1}^{p} T_{CNC2}^{j}}{8\sum_{i=1}^{n-1} T_{i,i+1}}, \quad j+p=8$$

$$s.t. \begin{cases} t_w^{i+1} = \begin{cases} P_m - P_{i+1\#} + \sum_{j=0}^{i-1} T_{j,j+1} + t_{i,i+1}, & P_{m+i} + t_{i,i+1} - P_{i+1\#} < t_f^{i+1} \\ t_f^{i+1}, & P_{m+i} + t_{i,i+1} - P_{i+1\#} \geq t_f^{i+1} \end{cases} \\ X_i = \begin{cases} 1, & \text{i 步对应的 CNC 上存在熟料} \\ 0, & \text{i 步对应的 CNC 上不存在熟料} \end{cases} \\ a_i \in CNC1 \cup X_{i-1}=1 \Longrightarrow a_{i+1} \in CNC2 \\ a_i \in CNC1 \cup X_{i-1}=0 \Longrightarrow a_{i+1} \in CNC1 \\ a_i \in CNC2 \Longrightarrow a_{i+1} \in CNC1 \\ a_{i+1} \in CNC1 \Longrightarrow t_f^{i+1} = t_5 \\ a_{i+1} \in CNC2 \Longrightarrow t_f^{i+1} = t_6 \end{cases}$$

将三个目标函数进行无量纲化后线性加权，对三个目标函数进行归一化处理，多目

标函数化归为单目标函数：

$$\max \quad \lambda_1 \frac{F_{1max} - F_1}{F_{1max} - F_{1min}} + \lambda_2 \frac{F_{2max} - F_2}{F_{2max} - F_{2min}} + \lambda_3 \frac{F_3 - F_{3min}}{F_{3max} - F_{3min}} \tag{7}$$
$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

6.2.3 模型算法

(1) 算法思想

对于两道工序的情况，同样的利用遗传算法，根据之前求出的多个约束条件，依次对每一段生产时间进行 RGV 最优路径的选取。与一道工序不同的是，两道工序在加工时，需要根据当前 CNC 上的物料种类来判断下一步的目的 CNC，我们则利用熟料系数 X_i 来辅助遗传算法过程中对目的 CNC 的判断。接着对 RGV 每段的路径利用遗传算法求解即可。

(2) 算法步骤

Step1 设每段模拟的步骤长度 $N = 9$，首先取第一段生产的路径 a_0 到 a_9，需要保持路径中第一个 CNC 不变，对 8 个小时内所有的生产路径分组后利用遗传算法确定最终的 RGV 调度方案。

Step2 初步编码。将本段生产路径编码为 a_n 到 a_{n+9}，其中 $a_i \in [1, 8]$。

Step3 种群初始化。设 $M = 50$，在保持起点 a_n 与前一段找出最优路径的终点一致的情况下，随机生成 50 个初始路径。通过式 (6) 判断本路径中 a_i 与 a_{i+1} 之间的关系是否符合移动规则，对不符合的路径进行剔除，并重新生成，直到 50 个路径均满足为止。

Step4 设置遗传迭代次数 $C = 2000$，以及迭代循环变量 $c = 0$，记录遗传的迭代次数。

Step5 选择操作。适应度函数为优化模型的目标函数，即式 (11)。同样采用了轮盘赌的选择算法对 50 个路径进行选择。

Step6 交叉操作。对所有的种群计算适应度函数的值 $f(b)$ 进行计算，将适应函数值相近的路径之间进行路径间的 CNC 的交换。然后需要利用式 (6) 对交叉产生的新的路径进行验证，对不满足条件的交叉操作进行复原，重新交叉。

Step7 变异操作。同样为了保留优秀的基因，规定优秀的基因，即适应度函数值 $f(b)$ 大的种群 b 尽量不进行变异；而对于较差的基因，则希望有较大的概率进行变异。之后利用式 (6) 对变异产生的新的路径进行验证，对不满足条件的变异操作进行复原。

Step8 结束一轮遗传迭代，迭代次数 c 自增 1。如果遗传的迭代次数的 c 大于等于 2000，则停止算法循环，记录最优的种群，即最优的 RGV 路径 b_{best}，以及消耗时间 ΔP。计算当前总生产时间 P_m，如果当前总生产时间 $P_m \geq 28800s$，退出循环；若总生产时间 $P_m < 28800s$，输出输出本时间段内加工物料的序号、加工两道工序的各 CNC 的编号，以及各自上料和下料开始时间。进入下一段生产路径，将此次生产路径的终点设为下一段生产路径的起点，设置生产路径为 a_{n+9} 到 a_{n+18}，返回 Step3，开始新一轮的遗传算法；否则回到 Step5，继续进行本段路径遗传算法的循环。

6.2.4 模型求解及结果分析

通过对两道工序无故障生产情况的模拟，得到了每个物料的两道工序分别的上下料开始时间，以及具体的情况。以第一组为例，调度策略如下表 (完整表格以及其它两组的调度策略见附件三)：

表 **3**　调度策略部分表

加工物料序号	工序 1 的 CNC 编号	上料开始时间	下料开始时间	工序 2 的 CNC 编号	上料开始时间	下料开始时间
1	1	0	426	2	452	1282
			……			
218	8	28549	停止工作	4	停止工作	停止工作

然后，分别计算了优化模型的三个目标的值，分别与 FCFS 方案进行比较：

表 **4**　两道工序无故障情况优化方案与 **FCFS** 方案比较

	物料总量（个）		RGV 移动距离（单位）		CNC 工作效率（工作时间/总时间)	
	优化方案	FCFS 方案	优化方案	FCFS 方案	优化方案	FCFS 方案
第一组	213	191	357	327	0.77	0.74
第二组	238	179	395	279	0.83	0.68
第三组	167	191	158	297	0.78	0.64

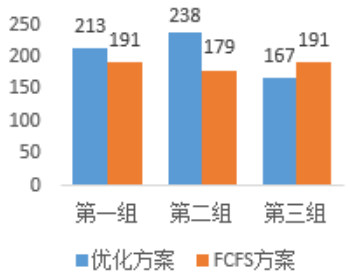


图 **12**　物料总量比较

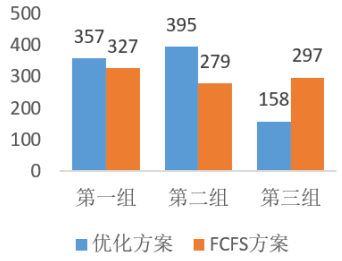


图 **13**　**RGV** 移动距离比较

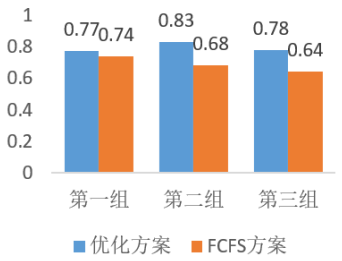


图 **14**　**CNC** 工作效率比较

通过对三个优化目标的比较，可以看出所设计的调度模型较 FCFS 模型都有很大的提升。对于物料总量，前两组数据都提高了 40 个物料左右，提升明显，但第三组优化后的物料数量较 FCFS 模型少，可能是因为遗传算法陷入局部最优解，导致没求解出最优调度方案；对于 RGV 的移动距离，前两组只增加了 50 单位左右的移动距离，增幅较小，第三组由于可能陷入局部最优解，自然移动距离较小；对于工作效率，每组相比于 FCFS 平均提高了 10 个百分点。整体来说，优化模型相比于 FCFS 模型有较大的提高，很好的提高了工厂的效率，节约了工厂的成本。有力的说明了优化模型的实用性和算法的有效性。

6.3 一道工序有故障情况

智能加工系统在加工过程中，CNC 有可能发生故障，且每台 CNC 发生故障的概率为 1%。每一次排除故障都需要一定的时间，为 10 到 20 分钟之间。并且当前 CNC 上的物料会当作生产失败被废弃，故障排除后 CNC 立即可以加入生成序列。同样的，可

以通过建立遗传算法模型来寻找最优路径，不同的是，每次决定 RGV 的移动目标之前，都判断目标 CNC 是否故障。建立的调度模型如下：

6.3.1 动态调度模型的修改

针对一道工序有故障的动态调度模型建立求解的问题，同样从完成时间、RGV 总移动距离和 CNC 的工作效率 3 个方面建立多目标优化模型。由于优化目标和一道工序无故障情况相同，不进行赘述。此目标优化模型的决策变量同样为当前时刻 P_m 规划的下 n 步线路 $a_0 \to a_1 \to a_2 \to ... \to a_n$。相比于一道工序无故障模型来说，增加了 CNC 损坏情况的考虑，即 CNC 损坏的约束：

新增约束条件：CNC 损坏的约束

对于 CNC 来说，在加工过程中有 1% 的机率故障，设故障系数为 g，$g=1$ 表示此 CNC 故障，$g=0$ 表示 CNC 正常运行。可以得到如下约束：

$$\begin{aligned} &P(g_{i+1}=0)=0.01 \\ &P(g_{i+1}=1)=0.99 \end{aligned} \tag{8}$$

6.3.2 模型汇总

通过上文的分析，对于下 n 步线路的选择问题，建立多目标优化模型如下：

$$\min \quad F_1=\sum_{i=0}^{n-1}[t_{i,i+1}+(1-g_{i+1})(t_f^{i+1}-t_{work}^{i+1})+g_{i+1}{t'_w}^{i+1}+t_c^{i+1}+X_{i+1}t_w]$$

$$\min \quad F_2=\sum_{i=0}^{n-1}d_{i,i+1}$$

$$\max \quad F_3=\frac{\sum_{j=1}^{8}T_{CNC}^{j}}{8\sum_{i=1}^{n-1}T_{i,i+1}}$$

$$s.t.\begin{cases} t_w^{i+1}=\begin{cases} P_m-P_{i+1\#}+\sum_{j=0}^{i-1}T_{j,j+1}+t_{i,i+1}, & P_{m+i}+t_{i,i+1}-P_{i+1\#}<t_f^{i+1} \\ t_f^{i+1}, & P_{m+i}+t_{i,i+1}-P_{i+1\#}\geq t_f^{i+1} \end{cases} \\ {t'_w}^{i+1}=\begin{cases} P_{broken}+t_{repair}-(P_{m+i}+t_{i,i+1}), & P_{broken}+t_{repair}>P_{m+i}+t_{i,i+1} \\ 0, & P_{broken}+t_{repair}<P_{m+i}+t_{i,i+1} \end{cases} \\ X_i=\begin{cases} 1, & \text{i 步对应的 CNC 上存在熟料} \\ 0, & \text{i 步对应的 CNC 上不存在熟料} \end{cases} \\ P(g_{i+1}=0)=0.01 \\ P(g_{i+1}=1)=0.99 \end{cases}$$

其中，$t_{i,i+1}$ 表示 RGV 从 a_i 号 CNC 移动到 a_{i+1} 号 CNC 的时间，t_f^{i+1} 表示 a_{i+1} 号 CNC 完成生产的耗时，t_{work}^{i+1} 表示 a_{i+1} 号 CNC 实际已生产耗时，t_c^{i+1} 表示 a_{i+1} 号 CNC 上下料的时长，t_w 表示熟料清洗时长，$d_{i,i+1}$ 表示 RGV 从 a_i 号 CNC 移动到 a_{i+1} 号 CNC 的距离，$T_{i,i+1}$ 表示第 $i+1$ 个物料对应的行走时间、生产、下料及清洗的时长，$P_{i+1\#}$ 表示最近一次前往 $i+1$ 号 CNC 并上料完成的时刻，P_m 表示当前时刻。

将多目标函数化归为单目标函数，可得：

$$\max \quad \lambda_1 \frac{F_{1max} - F_1}{F_{1max} - F_{1min}} + \lambda_2 \frac{F_{2max} - F_2}{F_{2max} - F_{2min}} + \lambda_3 \frac{F_3 - F_{3min}}{F_{3max} - F_{3min}} \tag{9}$$
$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

6.3.3 模型算法

(1) 算法思想

对每一段生产都使用遗传算法求解最优调度方案，然后将各个部分的最优调度方案按照时间进行衔接，最后综合模拟得到最优的整个班次的动态调度最优方案。需要注意的是在遗传算法中的生成每一个路径中的 CNC 之前，都需要对当前 CNC 进行是否故障的判断，对于故障的 CNC，则记录故障所需时长，在此期间，将此 CNC 排除在遗传算法之外，不作考虑；经过故障时长，即修复后，再将其纳入遗传算法考虑范围。

(2) 算法步骤

Step1 设每段模拟的步骤长度 $N = 9$，首先取第一段生产的路径 a_0 到 a_9，保持路径中第一个 CNC 不变，对 8 个小时内所有的生产路径分组后利用遗传算法确定最终的 RGV 调度方案。

Step2 初步编码。将生产路径编码为 a_n 到 a_{n+9}，其中 $a_i \in [1, 8]$。

Step3 种群初始化。采用随机方法生成 $M = 50$ 个体的初始路径。

Step4 故障判断。初始化每个路径的时候，需要对路径中的每个 CNC 进行故障的判断。如果 CNC 之前没有生产物料，即 CNC 刚被修复或未开始加工，不进行故障判断；如果 CNC 之前在生产物料，则利用式 (10) 对其进行判断，同时利用随机函数计算其修复时长，在这个时长内，遗传算法寻找路径不对其进行考虑。

Step5 设置遗传迭代次数 $C = 2000$，以及迭代循环变量 $c = 0$。

Step6 选择操作。采用轮盘赌的选择算法对路径 b 进行选择。

Step7 交叉操作。对适应函数值 $f(b)$ 相近的路径间的 CNC 进行交叉操作。然后利用式 (6) 对新的路径进行验证，对不满足条件的交叉操作进行复原，重新交叉。

Step8 变异操作。适应度函数值 $f(b)$ 大的路径 b 尽量不进行变异；$f(b)$ 较小的路径以较大的概率进行变异。变异后，同样需要利用式 (6) 对新的路径进行验证。

Step9 结束一轮遗传迭代，迭代次数 c 自增 1。如果遗传的迭代次数的 c 大于等于 2000，则停止算法循环，记录最优的种群，即最优的 RGV 路径 b_{best}，以及消耗时间 ΔP。计算当前总生产时间 P_m，如果当前总生产时间 $P_m \geq 28800s$，退出循环；若总生产时间 $P_m < 28800s$，输出本时间段内加工物料的序号、加工 CNC 的编号，以及上料和下料开始时间。进入下一段生产路径，设置生产路径为 a_{n+9} 到 a_{n+18}，返回 Step3，开始下一段路径的遗传算法；否则回到 Step5，继续进行本段路径遗传算法的循环。

6.3.4 模型求解及结果分析

通过对一道工序有故障生产情况的模拟，得到了每个物料的两道工序分别的上下料开始时间，以及具体的情况。以第一组为例，调度策略以及故障情况如下表 (完整表格以及其它两组的调度策略见附件三)：

表 5　调度策略部分表

加工物料序号	加工 CNC 编号	上料开始时间	下料开始时间
1	1	0	616
		……	
376	4	28758	停止作业

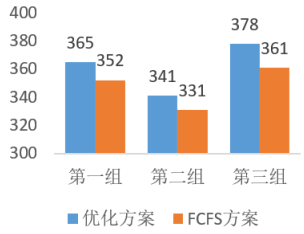
表 6　故障情况

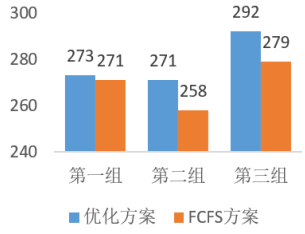
故障时的物料序号	故障 CNC 编号	故障开始时间	故障结束时间
176	8	13274	14457
359	5	27123	27946

然后，分别计算了优化模型的三个目标的值，分别与 FCFS 方案进行比较：

表 7　一道工序有故障情况优化方案与 **FCFS** 方案比较

	物料总量（个）		RGV 移动距离（单位）		CNC 工作效率（工作时间/总时间）	
	优化方案	FCFS 方案	优化方案	FCFS 方案	优化方案	FCFS 方案
第一组	365	352	273	271	0.75	0.75
第二组	341	331	271	258	0.82	0.81
第三组	378	361	292	279	0.75	0.69





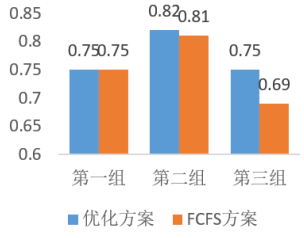


图 **15**　物料总量比较　　图 **16**　**RGV** 移动距离比较　图 **17**　**CNC** 工作效率比较

通过对三个优化目标的比较，可以看出所设计的调度模型较 FCFS 模型都有很大的提升。对于物料总量，前两组数据都提高了 12 个物料左右，提升明显；对于 RGV 的移动距离，前两组只增加了 15 单位左右的移动距离，增幅较小；对于工作效率，每组相比于 FCFS 平均提高了 2 个百分点。整体来说，优化模型相比于 FCFS 模型有较大的提高，很好的提高了工厂的效率，节约了工厂的成本。有力的说明了优化模型的实用性和算法的有效性。

6.4 两道工序有故障情况

对于两道工序有故障的情况，可以在两道工序的情况下，建立修改后的 RGV 动态调度模型。在有故障存在的两道工序中，同样的设置故障系数 g，通过判断每一个 CNC 的故障系数来判断其是否故障，若产生故障，则将其暂时排除在遗传算法搜索的范围之外，待修复时长后，再次加入搜索范围即可。

6.4.1 动态调度模型的修改

针对两道工序的的动态调度问题，以两道工序无故障的情况为基础，从完成时间、RGV 总移动距离和 CNC 的工作效率 3 个方面建立多目标优化模型。从工厂开始运行时开始模拟，将 8 个小时完成的生产时间同样按照一道工序的情况进行划分，以便于对每个时间段使用遗传算法求解出最优 RGV 移动路径。

对于当前情况来说，调度模型的优化目标与之前的两道工序无故障情况保持一致，不再详细说明，决策变量同为当前时刻 P_m 规划的下 n 步线路 $a_0 \to a_1 \to a_2 \to ... \to a_n$。相比于两道工序约束条件有所修改。

约束条件的修改：CNC 损坏的约束

和单个工序中的 CNC 损坏情况相同，负责不同生产工序的 CNC 都会产生故障。同理可以得到如下约束：

$$\begin{aligned} P(g_{i+1}=0) &= 0.01 \\ P(g_{i+1}=1) &= 0.99 \end{aligned} \tag{10}$$

其中，故障系数为 g，$g=1$ 表示此 CNC 故障，$g=0$ 表示 CNC 正常运行。

6.4.2 模型汇总

通过上文对动态调度模型的分析，对于下 n 步线路的选择问题，建立多目标优化模型如下：

$$\min \quad F_1 = \sum_{i=0}^{n-1}[t_{i,i+1} + (1-g_{i+1})(t_f^{i+1} - t_{work}^{i+1}) + g_{i+1}t_w'^{\,i+1} + t_c^{i+1} + X_{i+1}t_w]$$

$$\min \quad F_2 = \sum_{i=0}^{n-1} d_{i,i+1}$$

$$\max \quad F_3 = \frac{\sum_{j,p} T_{CNC1}^p T_{CNC2}^j}{8\sum_{i=1}^{n-1} T_{i,i+1}}, \quad j+p=8$$

$$s.t. \begin{cases} t_w^{i+1} = \begin{cases} P_m - P_{i+1\#} + \sum_{j=0}^{i-1} T_{j,j+1} + t_{i,i+1}, & P_{m+i} + t_{i,i+1} - P_{i+1\#} < t_f^{i+1} \\ t_f^{i+1}, & P_{m+i} + t_{i,i+1} - P_{i+1\#} \geq t_f^{i+1} \end{cases} \\ t_w'^{\,i+1} = \begin{cases} P_{broken} + t_{repair} - (P_{m+i} + t_{i,i+1}), & P_{broken} + t_{repair} > P_{m+i} + t_{i,i+1} \\ 0, & P_{broken} + t_{repair} < P_{m+i} + t_{i,i+1} \end{cases} \\ X_i = \begin{cases} 1, & \text{i 步对应的 CNC 上存在熟料} \\ 0, & \text{i 步对应的 CNC 上不存在熟料} \end{cases} \\ a_i \in CNC1 \cup X_{i-1} = 1 \Longrightarrow a_{i+1} \in CNC2 \\ a_i \in CNC1 \cup X_{i-1} = 0 \Longrightarrow a_{i+1} \in CNC1 \\ a_i \in CNC2 \Longrightarrow a_{i+1} \in CNC1 \\ a_{i+1} \in CNC1 \Longrightarrow t_f^{i+1} = t_5 \\ a_{i+1} \in CNC2 \Longrightarrow t_f^{i+1} = t_6 \\ P(g_{i+1}=0) = 0.01 \\ P(g_{i+1}=1) = 0.99 \end{cases}$$

其中，P_{broken} 表示故障时刻，t_{repair} 表示修理时间，均为随机生成。同样的将三个目标函数进行无量纲化后线性加权、归一化的处理，得到单目标函数：

$$\max \quad \lambda_1 \frac{F_{1max} - F_1}{F_{1max} - F_{1min}} + \lambda_2 \frac{F_{2max} - F_2}{F_{2max} - F_{2min}} + \lambda_3 \frac{F_3 - F_{3min}}{F_{3max} - F_{3min}} \tag{11}$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

6.4.3 模型算法

(1) 算法思想

对于两道工序有故障的情况，同样的以前面的两道工序无故障模型为基础，对其增加故障情况以建立模型。利用故障系数 g，在遗传算法对路径的寻找过程中进行判断，到达每个 CNC 时都利用故障系数 g 来判断具体的情况。在 CNC 出故障时，将其置于遗传算法搜索范围之外即可。在完成修理后重新加入模型。

(2) 算法步骤

Step1 设每段模拟的步骤长度 $N = 9$，首先取第一段生产的路径 a_0 到 a_9，保持路径中第一个 CNC 不变，对 8 个小时内所有的生产路径分组后利用遗传算法确定最终的 RGV 调度方案。

Step2 初步编码。将生产路径编码为 a_n 到 a_{n+9}，其中 $a_i \in [1, 8]$。

Step3 种群初始化。采用随机方法生成 $M = 50$ 个体的初始路径。并且需要利用式 (6) 对路径的合理性进行判断，对不符合条件的路径进行重新选取，直到完成 50 个初始路径的选取。

Step4 故障判断。初始化每个路径的时候，需要对路径中的每个 CNC 进行故障的判断。如果 CNC 之前没有生产物料，即 CNC 刚被修复或未开始加工，不进行故障判断；如果 CNC 之前在生产物料，则利用式 (10) 对其进行判断，同时利用随机函数计算其修复时长，在这个时长内，遗传算法寻找路径不对其进行考虑。

Step5 设置遗传迭代次数 $C = 2000$，以及迭代循环变量 $c = 0$。

Step6 选择操作。采用轮盘赌的选择算法对路径 b 进行选择。

Step7 交叉操作。对适应函数值 $f(b)$ 相近的路径进行交叉操作。

Step8 变异操作。适应度函数值 $f(b)$ 大的路径 b 尽量不进行变异；$f(b)$ 较小的路径以较大的概率进行变异。

Step9 结束一轮遗传迭代，迭代次数 c 自增 1。如果遗传的迭代次数的 c 大于等于 2000，则停止算法循环，记录最优的种群，即最优的 RGV 路径 b_{best}，以及消耗时间 ΔP。计算当前总生产时间 P_m，如果当前总生产时间 $P_m \geq 28800s$，退出循环；若总生产时间 $P_m < 28800s$，输出本时间段内加工物料的序号、加工 CNC 的编号，以及上料和下料开始时间。进入下一段生产路径，将此次生产路径的终点设为下一段生产路径的起点，设置生产路径为 a_{n+9} 到 a_{n+18}，返回 Step3，开始新一轮的遗传算法；否则回到 Step5，继续进行本段路径遗传算法的循环。

6.4.4 模型求解及结果分析

通过对两道工序有故障生产情况的模拟，得到了每个物料的两道工序分别的上下料开始时间，以及具体的情况。以第一组为例，调度策略以及故障情况如下表 (完整表格以及其它两组的调度策略见附件三)：

表 **8**　调度策略部分表

加工物料序号	工序 1 的 CNC 编号	上料开始时间	下料开始时间	工序 2 的 CNC 序号	上料开始时间	下料开始时间
1	1	0	428	2	481	920
			……			
209	6	28256	28687	4	28763	停止工作

表 **9**　故障情况

故障时的物料序号	故障 CNC 编号	故障开始时间	故障结束时间
26	5	3326	4454
121	3	16466	17643
159	1	21427	22404
166	4	22517	23218
206	4	27972	28745

然后，分别计算了优化模型的三个目标的值，分别与 FCFS 方案进行比较：

表 **10**　两道工序有故障情况优化方案与 **FCFS** 方案比较

	物料总量（个）		RGV 移动距离（单位）		CNC 工作效率（工作时间/总时间)	
	优化方案	FCFS 方案	优化方案	FCFS 方案	优化方案	FCFS 方案
第一组	209	169	316	376	0.83	0.69
第二组	171	151	283	212	0.74	0.63
第三组	206	181	326	258	0.79	0.58

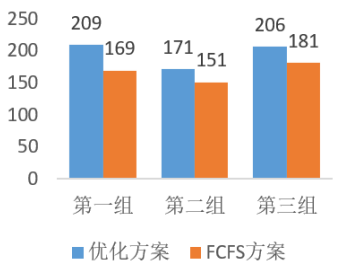


图 **18**　物料总量比较

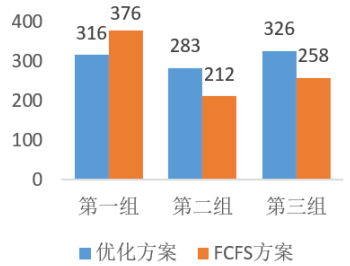


图 **19**　**RGV** 移动距离比较

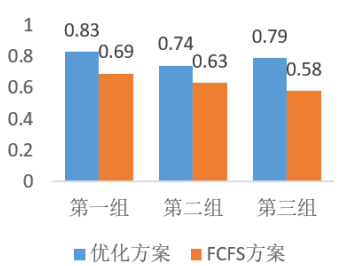


图 **20**　**CNC** 工作效率比较

通过对三个优化目标的比较，可以看出所设计的调度模型较 FCFS 模型都有很大的提升。对于物料总量，前两组数据都提高了 30 个物料左右，提升明显,；对于 RGV 的移动距离，第一组移动距离减少了 60 单位距离，效果很好，后两组只增加了 60 单位左

右的移动距离，增幅较小；对于工作效率，每组相比于 FCFS 平均提高了 15 个百分点。整体来说，优化模型相比于 FCFS 模型有较大的提高，很好的提高了工厂的效率，节约了工厂的成本。有力的说明了优化模型的实用性和算法的有效性。

七、 模型的评价及改进

7.1 模型优点

1. 在模型准备的时候先对“先到先来”调度规则进行分析，并且编写模拟系统且使用“先到先来”调度规则进行单道工序无故障、单道工序有故障、双道工序无故障、双道工序有故障四种情况的模拟调度，并且对模拟结果进行分析，得出我们模型所需要的三个指标；

2. 对于制定调度规则，我们使用了由前面得到的三个指标，并对三个指标进行线性加权得到综合评价指标，衡量不同调度规则的优劣；

3. 由于 8 个小时的时间尺度太长，因此我们将 8 个小时分割为多段时间，对多段时间进行研究；

4. 制定调度规则的时候，我们使用了遗传算法，能快速和较为准确地得到调度策略。

7.2 模型的缺点

1. 未考虑负责不同道工序的 CNC 数量和分布不同的情况下，对调度策略的影响；

2. 使用综合评价指标的时候，未对三个分指标的权值进行不同情况下的分析；

3. 使用遗传算法的时候，有可能会遗漏全局最优解，陷入局部最优解，导致得到的调度策略不是全局的最优调度策略。

7.3 模型的改进

1. 由于在负责不同道工序的 CNC 数量和分布不同的情况下，会对我们的调度策略造成影响。而对于 CNC 数量和分布，不同的情况太多，因此也可以考虑使用遗传算法对 CNC 数量和分布进行编码，在不同 CNC 数量和分布的编码情况下，结合智能加工系统作业参数的 3 组数据，先对“先来先到”调度规则进行分析，然后使用本文模型中建立的遗传算法对不同情况的调度策略进行计算；

2. 在进行调度策略计算的时候，对于综合指标，可以查阅资料，得到三个分指标对于公司利益的具体影响，使用模糊综合评价对三个分指标权值进行确认；

3. 对于使用遗传算法有可能会遗漏全局最优解的问题，可以考虑使用其它算法于遗传算法进行结合，例如结合禁忌搜索的遗传算法。

八、 参考文献

[1] 司守奎. 数学建模算法与应用习题解答 [M]. 国防工业出版社, 2015.

[2] 张桂琴, 张仰森. 直线往复式轨道自动导引车智能调度算法 [J]. 计算机工程, 2009, 35(15): 176-178.

[3] 张志鹏. 基于多目标遗传粒子群混合算法求解混合流水车间调度问题研究 [D]. 大连交通大学,2014.

附录清单

编号	名称
A	模拟单工无故障的 MATLAB 代码
B	优化单工无无故障的 MATLAB 代码

附件清单

编号	名称
附件一	FCFS 具体结果及相关源程序
附件二	遗传算法优化具体结果及相关源程序
附件三	最优 RGV 动态调度方案

附录 A 模拟单工无故障的 MATLAB 代码

```
%模拟：先到先服务
clc,clear all
%已知参数
t=zeros(1,3);
can=[18 32 46 545 455 182 27 32 25];%系统作业参数
t(1)=can(1); % RGV移动一个单位所需时间
t(2)=can(2); % RGV移动二个单位所需时间
t(3)=can(3); % RGV移动三个单位所需时间
t(4)=0;
T=can(4); %CNC加工完成一道工序的物料所需时间
T_down=can(7); %RGV为CNC1#,3#,5#,7#一次上下料所需时间
T_up=can(8); %RGV为CNC2#,4#,6#,8#一次上下料所需时间
T_clean=can(9); %RGV完成一个物料的清洗作业所需时间

%模拟过程
p=[];%每一次上料并清洗后的时刻
pp=zeros(1,8);%最近一次前往i#CNC上料的时刻
shuLiao=zeros(1,8); %有无熟料，当有熟料时为1，否则为0
position=1; %RGV初始位置
%初始状态
queue=[1 2 3 4 5 6 7 8];%初始请求队列
tail=8;%队尾为8
p(1)=0;%初始时刻
k=1;
sum=1;%访问CNC的物料次序
distance=0;%GRV移动距离
result =[];
while true
if mod(queue(1),2)==1
d=abs((queue(1)-position))/2;
else
d=abs((floor((2*queue(1)-1)/2)-position))/2;
end
distance=distance+d;
if queue(1) > position
position=position+2*d;
else
position=position-2*d;
end

if mod(queue(1),2)==1
t_shangLiao=T_down;
else
t_shangLiao=T_up;
end
if d==0;
d=4;
end
pp(queue(1))=t(d)+t_shangLiao+p(k);%最近一次前往i#CNC上料的时刻
result=[result;sum queue(1) pp(queue(1)) distance];
p(k+1)=pp(queue(1))+shuLiao(queue(1))*T_clean;%此时时刻
sum=sum+1;
shuLiao(queue(1))=1;%i#CNC有熟料

%出队
if tail~=0 %判断队列是否为空
for j=2:length(queue)
queue(j-1)=queue(j);
end
tail=tail-1;
end
```

```
if tail==0
for i=1:8
temp(i)=p(k+1)-pp(i)-T;
end
if length(find(temp<0))==8 %此刻还没有熟料
[m,n]=max(temp);%m: 最大值 n: 位置
p(k+1)=p(k+1)-m;%更新时间
queue(1)=n; %入队
tail=tail+1;
else
a=find(temp>0);%位置
aa=temp(a);%值
m=sort(aa,'descend');%m:排序结果 n:位置
Length=length(m);
for i=1:Length
queue(i)=find(temp==m(i)); %入队
tail=tail+1;
end
end
end

if p(k+1)>8*60*60
break;
end
k=k+1;
end
```

附录 B 优化单工无无故障的 MATLAB 代码

```
%遗传算法
% function GA
clc,clear
%输入参数
global N;
N=9; %一定次序的CNC编号
M=100; %种群个数
C=200; %迭代次数
Pc=0.4; %交叉概率
Pm=0.5; %变异概率
k1=0.5; %目标一的权重
k2=0.2; %目标二的权重
k3=1-k1-k2; %目标三的权重
flag=1;
%已知参数
%can=[20 33 46 560 400 378 28 31 25];
%can=[23 41 59 580 280 500 30 35 30];
can=[18 32 46 545 455 182 27 32 25];
t1=can(1); % RGV移动一个单位所需时间
t2=can(2); % RGV移动二个单位所需时间
t3=can(3); % RGV移动三个单位所需时间
T=can(4); %CNC加工完成一道工序的物料所需时间
% T1=can(5); %CNC加工完成一道工序的物料的第一道工序所需时间
% T2=can(6); %CNC加工完成一道工序的物料的第二道工序所需时间
T_down=can(7); %RGV为CNC1#,3#,5#,7#一次上下料所需时间
T_up=can(8); %RGV为CNC2#,4#,6#,8#一次上下料所需时间
T_clean=can(9); %RGV完成一个物料的清洗作业所需时间

RESULT=[];
result=cell(C,M);%结果
count=1;
```

```
%生成RGV在两个CNC之间的移动时间矩阵
t_yiDong=[0 0 t1 t1 t2 t2 t3 t3
0 0 t1 t1 t2 t2 t3 t3
t1 t1 0 0 t1 t1 t2 t2
t1 t1 0 0 t1 t1 t2 t2
t2 t2 t1 t1 0 0 t1 t1
t2 t2 t1 t1 0 0 t1 t1
t3 t3 t2 t2 t1 t1 0 0
t3 t3 t2 t2 t1 t1 0 0];
d_yiDong=[0 0 1 1 2 2 3 3
0 0 1 1 2 2 3 3
1 1 0 0 1 1 2 2
1 1 0 0 1 1 2 2
2 2 1 1 0 0 1 1
2 2 1 1 0 0 1 1
3 3 2 2 1 1 0 0
3 3 2 2 1 1 0 0];
fix=[1 2 3 4 5 6 7 8 1
2 3 4 5 6 7 8 1 2
3 4 5 6 7 8 1 2 3
4 5 6 7 8 1 2 3 4
5 6 7 8 1 2 3 4 5
6 7 8 1 2 3 4 5 6
7 8 1 2 3 4 5 6 7
8 1 2 3 4 5 6 7 8
2 3 4 5 6 7 8 2 3
3 4 5 6 7 8 1 3 4
4 5 6 7 8 1 2 4 5
5 6 7 8 1 2 3 5 6
6 7 8 1 2 3 4 6 7
7 8 1 2 3 4 5 7 8
8 1 2 3 4 5 6 8 1
1 2 3 4 5 6 7 1 2
2 3 4 5 6 7 8 2 1
3 4 5 6 7 8 1 3 2
4 5 6 7 8 1 2 4 3
5 6 7 8 1 2 3 5 4
6 7 8 1 2 3 4 6 5
7 8 1 2 3 4 5 7 6
8 1 2 3 4 5 6 8 7
1 2 3 4 5 6 7 1 8
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1];
length_fix=size(fix,1);
%初始状态
pp=zeros(M,8);%最近一次前往i#上料完成的时刻
p=zeros(1,M);%每个种群的初始时刻
pm=zeros(1,M);%每次规划初始时间
shuLiao=zeros(M,8); %每个种群中每个个体的熟料状态，有熟料为1，否则为0
seq_CNC=1;%初始访问CNC的序号
seq=1;
generte=1;
%生成初始群体
popm=zeros(M,N+1);
popm(1,:)=[1 1 2 3 4 5 6 7 8 1 ];
for i=2:M
popm(i,1)=seq_CNC;
for j=2:N+1
popm(i,j)=round(rand*7)+1;
end
```

```
end

%初始化种群及其适应函数
fitness=zeros(M,1); %每个种群的适应度
goal=zeros(M,3);%三个目标
for i=1:M %分别计算每个种群前两个目标
location=find(pp(i,:)~=0);
for k=1:length(location)
if pm(i)-pp(i,location(k))-T<0
goal(i,3)=goal(i,3)-(pm(i)-pp(i,location(k))-T);
end
end

for j=2:N+1
goal(i,2)=goal(i,2)+d_yiDong(popm(i,j-1),popm(i,j));
p(i)=p(i)+t_yiDong(popm(i,j-1),popm(i,j));%更新当前时间
if  pp(i,popm(i,j))==0%popm(i,j+1)没有在加工，因此不需要计算等待时间
if mod(popm(i,j),2)==1
pp(i,popm(i,j))=p(i)+T_down;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i};seq popm(i,j) pp(i,popm(i,j))];
p(i)=p(i)+T_down;
else

pp(i,popm(i,j))=p(i)+T_up;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i};seq popm(i,j) pp(i,popm(i,j))];
p(i)=p(i)+T_up;
end

else
if p(i)-pp(i,popm(i,j))<T %需要计算等待时间，并且计算完后计算上下料时间
goal(i,1)=goal(i,1)+T-p(i)+pp(i,popm(i,j));

p(i)=T-p(i)+pp(i,popm(i,j))+p(i);%等待，并且更新当前时间到等待完成时刻
end
%%计算完等待时间后计算上下料时间
if mod(popm(i,j),2)==1
pp(i,popm(i,j))=p(i)+T_down;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i};seq popm(i,j) pp(i,popm(i,j))];
p(i)=p(i)+T_down;
else
pp(i,popm(i,j))=p(i)+T_up;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i};seq popm(i,j) pp(i,popm(i,j))];
p(i)=p(i)+T_up;
end
end

%第四个计算清洗时间
if shuLiao(i,popm(i,j))==1 %需要清洗
p(i)=p(i)+T_clean;
end

end
seq=seq+1;
goal(i,1)=p(i)-pm(i);%总时间
end
pingshu=zeros(M,8);
for i=1:M
for j=1:8
pingshu(i,j)=length(find(popm(i,:)==j));
if p(i)-pp(i,j)>=T && pp(i,j)~=0
goal(i,3)=goal(i,3)+pingshu(i,j)*T;
elseif  p(i)-pp(i,j)<=T && pp(i,j)~=0
goal(i,3)=goal(i,3)+(pingshu(i,j)-1)*T+p(i)-pp(i,j);
end
```

```
end
end
goal(:,3)=goal(:,3)./(8*goal(:,1));
pm=p;

maxGoal=[max(goal(:,1)) max(goal(:,2)) max(goal(:,3))];
minGoal=[min(goal(:,1)) min(goal(:,2)) min(goal(:,3))];

%计算适应度
for i=1:M
fitness(i)=k1*(maxGoal(1)-goal(i,1))/(maxGoal(1)-minGoal(1)+0.0001)+...
k2*(maxGoal(2)-goal(i,2))/(maxGoal(2)-minGoal(2)+0.0001)+...
+(1-k1-k2)*(goal(i,3)-minGoal(3))/(maxGoal(3)-minGoal(3)+0.0001);
end
fitness=fitness/sum(fitness);
p_zuiYou=0;
sum=0;

fix_pp=zeros(M,8);%最近一次前往i#上料完成的时刻
fix_p=zeros(1,M);%每个种群的初始时刻
fix_pm=zeros(1,M);%每次规划初始时间
fix_shuLiao=zeros(M,8); %每个种群中每个个体的熟料状态，有熟料为1，否则为0


while fix_p(1)<60*60*8
sum=sum+1;

C=200;
while C>0
pm=fix_pm;%更新pm
pp=fix_pp;
p=fix_p;
shuLiao=fix_shuLiao;

kk=1;
for i=M-29:M
popm(i,:)=[popm(i,1) fix(kk,:)];
kk=kk+1;
end




if flag==0
%求适应度函数
fitness=zeros(M,1); %每个种群的适应度
goal=zeros(M,3);%三个目标
for i=1:M %分别计算每个种群前两个目标
location=find(pp(i,:)~=0);
for k=1:length(location)
if pm(i)-pp(i,location(k))-T<0
goal(i,3)=goal(i,3)-(pm(i)-pp(i,location(k))-T);
end
end

for j=2:N+1
goal(i,2)=goal(i,2)+d_yiDong(popm(i,j-1),popm(i,j));

p(i)=p(i)+t_yiDong(popm(i,j-1),popm(i,j));%更新当前时间
if  pp(i,popm(i,j))==0%popm(i,j+1)没有在加工，因此不需要计算等待时间
if mod(popm(i,j),2)==1
pp(i,popm(i,j))=p(i)+T_down;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i}; seq popm(i,j) pp(i,popm(i,j))];
```

```
p(i)=p(i)+T_down;
else

pp(i,popm(i,j))=p(i)+T_up;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i};seq popm(i,j) pp(i,popm(i,j))];
p(i)=p(i)+T_up;
end

else
if p(i)-pp(i,popm(i,j))<T %需要计算等待时间，并且计算完后计算上下料时间
goal(i,1)=goal(i,1)+T-p(i)+pp(i,popm(i,j));

p(i)=T-p(i)+pp(i,popm(i,j))+p(i);%等待，并且更新当前时间到等待完成时刻
end
%%计算完等待时间后计算上下料时间
if mod(popm(i,j),2)==1
pp(i,popm(i,j))=p(i)+T_down;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i};seq popm(i,j) pp(i,popm(i,j))];
p(i)=p(i)+T_down;
else
pp(i,popm(i,j))=p(i)+T_up;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i};seq popm(i,j) pp(i,popm(i,j))];
p(i)=p(i)+T_up;
end
end

%第四个计算清洗时间
if shuLiao(i,popm(i,j))==1 %需要清洗
p(i)=p(i)+T_clean;
end

end
goal(i,1)=p(i)-pm(i);%总时间
seq=seq+1;
end
for i=1:M
for j=1:8
pingshu(i,j)=length(find(popm(i,:))==j);
if p(i)-pp(i,j)>=T && pp(i,j)~=0
goal(i,3)=goal(i,3)+pingshu(i,j)*T;
elseif  p(i)-pp(i,j)<=T && pp(i,j)~=0
goal(i,3)=goal(i,3)+(pingshu(i,j)-1)*T+p(i)-pp(i,j);
end
end
end
goal(:,3)=goal(:,3)./(8*goal(:,1));
maxGoal=[max(goal(:,1)) max(goal(:,2)) max(goal(:,3))];
minGoal=[min(goal(:,1)) min(goal(:,2)) min(goal(:,3))];

%计算适应度
for i=1:M
fitness(i)=k1*(maxGoal(1)-goal(i,1))/(maxGoal(1)-minGoal(1)+0.0001)+...
k2*(maxGoal(2)-goal(i,2))/(maxGoal(2)-minGoal(2)+0.0001)+...
+(1-k1-k2)*(goal(i,3)-minGoal(3))/(maxGoal(3)-minGoal(3)+0.0001);
end
sum_fit=0;
for i=1:M
sum_fit=sum_fit+fitness(i);
end
fitness=fitness./sum_fit;

end
flag=0;
```

```
if C==1
break;
end




%选择操作
%        totalFit=0;
%        for i=1:M
%            totalFit=totalFit+fitness(i);
%        end
fit=cumsum(fitness);
ms=sort(rand(M,1));
fitin=1;
newin=1;
while newin<=M
if(ms(newin))<fit(fitin)
popm_sel(newin,:)=popm(fitin,:);
newin=newin+1;
else
fitin=fitin+1;
end
end

%    %交叉操作
%    %变异操作
popm_sel_1=popm_sel;
for i=6:M
pick=rand;
while pick==0
pick=rand;
end
if pick<Pm
index1=round(rand*(N-1))+2;
index2=round(rand*(N-1))+2;
temp=0;
temp=popm_sel_1(i,index1);
popm_sel_1(i,index1)=popm_sel_1(i,index2);
popm_sel_1(i,index2)=temp;
end
end
for i=M-10:M
index3=round(rand*(N-1))+2;
popm_sel_1(i,index3)=round(rand*(7))+1;
end
C=C-1;
popm= popm_sel_1;
count=count+1;

end
count=1;



[fit_min,fit_location]=max(fitness);
zuiyou=popm(fit_location,:);
RESULT=[RESULT; result{200,fit_location}];
result=cell(200,M);
```

```
%每个阶段最优解
%更新状态
%每段初始状态
for i=1:M
fix_pp(i,:)=pp(fit_location,:);%最近一次前往i#上料完成的时刻
end
fix_p(1,:)=p(fit_location);%每个种群的初始时刻
fix_pm(1,:)=p(fit_location);%每次规划初始时间
for i=1:M
fix_shuLiao(i,:)=shuLiao(fit_location,:); %每个种群中每个个体的熟料状态，有熟料为1，否则为0
end
pm=fix_pm;%更新pm
pp=fix_pp;
p=fix_p;
shuLiao=fix_shuLiao;
%更新下一段

for i=1:M
popm(i,1)=zuiyou(N+1);
for j=2:N+1
popm(i,j)=round(rand*7)+1;
end
end


%随机选择一个种群

%初始化种群及其适应函数
fitness=zeros(M,1); %每个种群的适应度
goal=zeros(M,3);%三个目标
for i=1:M %分别计算每个种群前两个目标
location=find(pp(i,:)~=0);
for k=1:length(location)
if pm(i)-pp(i,location(k))-T<0
goal(i,3)=goal(i,3)-(pm(i)-pp(i,location(k))-T);
end
end

for j=2:N+1
goal(i,2)=goal(i,2)+d_yiDong(popm(i,j-1),popm(i,j));

p(i)=p(i)+t_yiDong(popm(i,j-1),popm(i,j));%更新当前时间
if  pp(i,popm(i,j))==0%popm(i,j+1)没有在加工，因此不需要计算等待时间
if mod(popm(i,j),2)==1
pp(i,popm(i,j))=p(i)+T_down;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i};popm(i,j) pp(i,popm(i,j))];
p(i)=p(i)+T_down;
else

pp(i,popm(i,j))=p(i)+T_up;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i};popm(i,j) pp(i,popm(i,j))];
p(i)=p(i)+T_up;
end

else
if p(i)-pp(i,popm(i,j))<T %需要计算等待时间，并且计算完后计算上下料时间
goal(i,1)=goal(i,1)+T-p(i)+pp(i,popm(i,j));

p(i)=T-p(i)+pp(i,popm(i,j))+p(i);%等待，并且更新当前时间到等待完成时刻
end
%%计算完等待时间后计算上下料时间
if mod(popm(i,j),2)==1
pp(i,popm(i,j))=p(i)+T_down;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i};popm(i,j) pp(i,popm(i,j))];
```

```
p(i)=p(i)+T_down;
else
pp(i,popm(i,j))=p(i)+T_up;%更新每个CNC最近一次上料完成的时刻
result{count,i}=[result{count,i};popm(i,j) pp(i,popm(i,j))];
p(i)=p(i)+T_up;
end
end

%第四个计算清洗时间
if shuLiao(i,popm(i,j))==1 %需要清洗
p(i)=p(i)+T_clean;
end

end
goal(i,1)=p(i)-pm(i);%总时间
end
pingshu=zeros(M,8);
for i=1:M
for j=1:8
pingshu(i,j)=length(find(popm(i,:)==j));
if p(i)-pp(i,j)>=T && pp(i,j)~=0
goal(i,3)=goal(i,3)+pingshu(i,j)*T;
elseif  p(i)-pp(i,j)<=T && pp(i,j)~=0
goal(i,3)=goal(i,3)+(pingshu(i,j)-1)*T+p(i)-pp(i,j);
end
end
end
goal(:,3)=goal(:,3)./(8*goal(:,1));

maxGoal=[max(goal(:,1)) max(goal(:,2)) max(goal(:,3))];
minGoal=[min(goal(:,1)) min(goal(:,2)) min(goal(:,3))];

%计算适应度
for i=1:M
fitness(i)=k1*(maxGoal(1)-goal(i,1))/(maxGoal(1)-minGoal(1)+0.0001)+...
k2*(maxGoal(2)-goal(i,2))/(maxGoal(2)-minGoal(2)+0.0001)+...
+(1-k1-k2)*(goal(i,3)-minGoal(3))/(maxGoal(3)-minGoal(3)+0.0001);
end
sum_fit=0;
for i=1:M
sum_fit=sum_fit+fitness(i);
end
fitness=fitness./sum_fit;
flag=1;
end
```