摘要

关键字：决策模型

目录

一、 问题重述

1.1 问题背景

一个智能加工系统由 8 台计算机数控机床 (Computer Number Controller, CNC)、1 辆轨道式自动导引车 (Rail Guide Vehicle, RGV)、1 条 RGV 直线轨道、1 条上料传送带、1 条下料传送带等附属设备组成。RGV 是一种无人驾驶、能在固定轨道上自由运行的智能车。它根据指令能自动控制移动方向和距离，并自带一个机械手臂、两只机械手爪和物料清洗槽，能够完成上下料及清洗物料等作业任务。[**?**] 通常来说，一个工件成品的完成需要若干步，鉴于 RGV 小车在同一时间只能处理同一种任务，而同时最多可以有 8 台数控机床在运行，因此对 RGV 小车进行正确的调度将显著地提高该智能加工系统的加工效率。在本文中，我们将运用不同模型讨论在不同情形下，如何调度小车能够使该系统达到最高效的工作状态。

1.2 问题提出

根据以上的背景，我们在本文中需要针对下面的三种具体情况：

- 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；
- 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；
- CNC 在加工过程中可能发生故障（据统计：故障的发生概率约为 1%）的情况，每次故障排除（人工处理，未完成的物料报废）时间介于 10 20 分钟之间，故障排除后即刻加入作业序列。要求分别考虑一道工序和两道工序的物料加工作业情况。

完成两项任务：

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；

任务 2：利用已给出的系统作业参数的 3 组数据分别检验模型的实用性和算法的有效性，给出 RGV 的调度策略和系统的作业效率。

二、 模型的假设

基于赛题中给出的一些条件和生活常识，我们在本文中提出如下假设。此后在用到这些假设时，我们将不再另作声明。

- RGV 小车需要对某台数控机床上下料作业时，数控机床对应的传送带上一定有一个预备好的原料工件；
- RGV 小车不能同时进行两项或以上的工作 (包括移动)；
-

三、 符号说明

符号	意义
alg	调度算法/调度方案
n_t	在 t 时间内加工完的工件数
T_0	情况一 CNC 由原料加工为成品的时间
T_1	情况二 CNC 由原料加工为半成品的时间
T_2	情况二 CNC 由半成品加工为成品的时间
T_{movek}	RGV 移动 k 步所花的时间 ($k \in 1, 2, 3$)
T_{wash}	RGV 清洗所花的时间
$T_{loadodd}$	RGV 为奇数号 CNC 上下料所花的时间
$T_{loadeven}$	RGV 为偶数号 CNC 上下料所花的时间
20	640.2

表 1　文中用到的符号和含义

四、 问题分析

我们将针对本问题给出的三种情况分别予以分析，并提出可能的优化算法。

4.1 情况一、二分析

在情况一中，一个工件只需经过一步加工就可以成为成品，然后经过 RGV 小车清洗后放入下料传送带送出该系统。此时，一个工件从原料到成品需要 RGV 小车的如下操作：

1. 移动步骤：RGV 小车从原来的位置运行到 CNC 的位置；
2. 第一次上下料步骤：小车将上料传送带上的原料取下，将 CNC 中的成品置换为原料，若 CNC 原来处于空置状态，则只将原料放入 CNC 中；
3. 等待：等待 CNC 加工完成，此时小车可以进行其他任务；
4. 第二次上下料步骤：小车再次运行到 CNC 的位置，将 CNC 中的成品置换为原料，此时小车不能为已装载成品的状态；
5. 清洗步骤：小车清洗成品，并将成品置入下料传送带上。

而在情况二中，一个工件需要两步加工才能制成成品，因此工件由原料到成品需要多一次上下料步骤，即在第一轮加工完成后，将加工完成的工件放入第二步工序的 CNC 中加工。其他的步骤均是相同的。

显然，因为不存在任何随机因素，因此所有小车调度方案的数量是有限的，故情况一、二的全局最优解是一定存在并且确定的，意即：

$$\exists \text{alg s.t. } n_{t,\text{alg}} = \text{Max}[n_t]$$

所以我们需要找到该最优的调度方案或接近最优的调度方案。

要得到最优的调度方案，可能可以使用的算法有：穷举法、模拟退火算法、决策树算法、遗传算法、神经网络等。其中，穷举法和模拟退火算法的时间复杂度过高，超过了我们所拥有的算力极限，故我们不考虑用此两种方法。同样是基于统计的算法，遗传算法在该题中表现出比模拟算法更优的时间性能，因此遗传算法将成为我们探寻最优解的一种尝试。

4.2 情况三分析

对于情况三，由于加入了随机因素（CNC 发生故障），因此不存在固定的全局最优解。在此情况下，调度过程转化成了小车的决策过程。空闲的小车需要灵活地根据当前加工系统的状态来决定下一步将要进行的步骤。在该情况下，遗传算法仍然可以使用，但是由原来的提供静态方案转变为即时演算出动态决策。

五、 模拟环境

为了能够实际测试调度算法产生的指令序列在 RGV 智能加工系统中的表现，我们基于题目中给出的文档与数据，设计了一个简易的智能加工系统模拟器。每当 RGV 小

车处于空闲状态时，模拟器会将当前智能加工系统各部分的状态以及当前的时刻作为参数传递给调度算法，调度算法会以此作为输入进行计算，并且输出当前小车需要执行的指令。

模拟器使用 python 设计，主要由 4 个模块组成。它们分别是 world.py，rgv.py，cnc.py 以及 cargo.py。

world.py 是模拟器的核心部分。在 world.py 中包含模拟器的主对象 World，构造该对象时需要传入工作时间或者加工工件总数，以及调度器作为参数。对象构造完成后，调用对象的 simulate() 方法就可以开始模拟。模拟结束后，可以调用 result() 和 final() 方法获得加工的工件个数/加工的总时间长，以及每一个工件上料和下料的时间日志记录。

rgv.py 包含了 RGV 对象，其中定义了 RGV 可以接受的指令和它们的编码，它们在模拟器中的对应关系如下表所示。

编码	指令
0	空闲
1	向左移动一格
2	向右移动一格
3	向左移动两格
4	向右移动两格
5	向左移动三格
6	向右移动三格
7	向 #1，#3，#5，#7 号 CNC 上下料
8	向 #2，#4，#6，#8 号 CNC 上下料
9	清洗工件

表 2　RGV 的指令和编码

RGV 对象可以通过 inst() 方法来接受指令。调用这个方法时需要传入一个有效指令的编码，同时根据指令的种类传入一个可选参数，类型为代表工件的 Cargo 类。关于 Cargo 类的信息会在下文描述。调用该方法后会设置 RGV 内部的一些属性，包括对象内部的计时器。

同时，RGV 对象也拥有 update() 方法，调用这个方法时，如果 RGV 正在执行空闲之外的指令，那么程序会将 RGV 内部的计时器减少 1 秒。接着，程序将会检查 RGV

内部的计时器是否已经达到零。如果计时器已经达到零，那么表明当前指令已经执行完成，那么 RGV 对象会根据指令的不同执行一些动作，比如修改 CNC 机床内部的工件对象或者修改自身内部的工件状态。

cnc.py 包含了 CNC 对象。CNC 对象的可能状态如下表所示：

编码	指令
0	空闲
1	加工中
2	加工完成
3	故障

表 3　CNC 对象的可能状态

与 RGV 对象类似，CNC 对象也拥有 inst() 方法与 update() 方法。它们的功能也与 RGV 对象中的对应方法类似。

由于智能加工系统的要求，CNC 对象拥有不同模式。可能的模式如下表所示：

编码	状态
0	一阶段加工
1	二阶段加工，第一阶段
2	二阶段加工，第二阶段

表 4　CNC 对象的可能模式

cargo.py 内部包含 Cargo 对象。Cargo 对象的状态如下表所示：Cargo（工件）对象的状态可以用 RGV 对象和 CNC 对象中的相应指令来修改。同时，保存工件的状态也使得模拟器能够检测出输入的指令序列中的错误，防止不能出现的情况发生。

为了调试方便，我们为模拟器添加了输出函数 info()。输出示例如下：

```
supply cargo 2
Clock: 3931
RGV:       CNC 1:     CNC 2:     CNC 3:     CNC 4:     CNC 5:     CNC 6:     CNC 7:     CNC 8:
idle       processed  processed  idle       idle       processed  processing idle       idle
8, ready   4, ready   5, ready                         7, ready   9, raw
```

编码	状态
0	未加工
1	半加工
2	加工完成，未清洗
3	加工完成，已清洗

表 5　Cargo（工件）对象的可能状态

```
current cargo: 9
```

模拟结束后，模拟器会将数据写入对应的文件中。

六、 标准调度模型

为了能够定量的评价后续模型的优劣，我们按照题意设计了一个标准调度模型，来与之后的模型对比。标准调度模型是一种确定性模型，由人来确定小车调度中应当依次完成工作的优先级，然后小车就依照确定的优先级处理 CNC 发送的请求。

6.1 情况一的标准调度模型

对于情况一，标准调度模型基于以下几条原则：

- 小车会优先执行完当前的指令队列，然后再接收 CNC 的服务请求；
- 在所有 CNC 都工作时，若小车的执行队列为空，则小车处于空闲状态；
- 若只有一台 CNC 处于可服务状态，小车将会为这台 CNC 提供服务；
- 若有多台 CNC 处于可服务状态，小车将会优先服务近的、奇数号的 CNC；
- 在完成一次上下料作业后，若需要清洗，小车会立即进行清洗。

在这个情况下，小车的调度成为了一个确定问题。在每次小车处于空闲状态时，它可能会接收多个 CNC 的服务请求，但是它将要执行的指令其实已经确定了。我们可以认为，小车会将 CNC 的优先级按照距离和奇偶性排序。在它接收了多个 CNC 的服务请求以后，将按照图 **??** 所示的流程依次判断服务对象，然后将整个服务流程所需的步骤编入其指令队列。例如，若小车处于位置 1，而编号为 3，4，5，7 的 CNC 需要服务，那么小车会按照 1，2，……，7，8 的顺序询问 CNC 是否需要服务。最终，小车将对编号为 3 的机器提供服务。在有机器出现故障的情况下，该标准模型仍能正常工作。

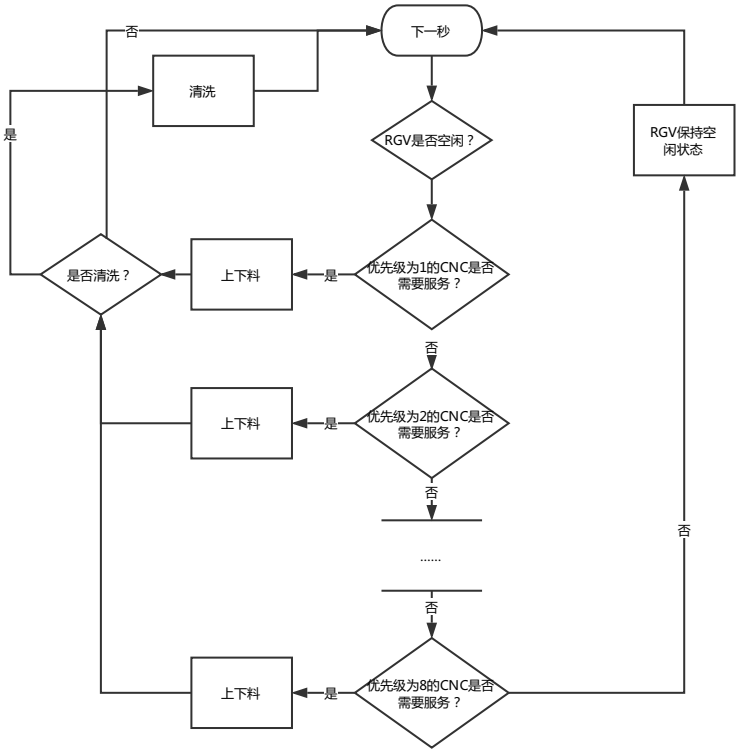

图 1　标准模型（情况一）的运行流程

6.2 情况二的标准调度模型

对于情况二，一个工件从原料到成品需要两个工序，依次经过两台 CNC。因此，我们对情况一的原则稍加改动，引入情况二标准模型的几条原则：

- 小车会优先执行完当前的指令队列，然后再接收 CNC 的服务请求；
- 在所有 CNC 都工作时，若小车的执行队列为空，则小车处于空闲状态；
- 若小车当前只能为一台 CNC 提供服务，则小车将会为这台 CNC 提供服务；
- 若小车当前能为多台 CNC 提供服务，则小车会按照先服务第一工序 CNC，再服务第二工序 CNC；先近后远；先奇后偶的顺序依次对要服务的 CNC 进行排序，然后服务第一顺序的 CNC；
- 在完成一次上下料作业后，若需要清洗，小车会立即进行清洗。

同样地，在情况二下小车调度也是一个确定问题，流程如图所示。

七、 情况一的解

7.1 标准调度模型

使用标准调度模型可以快速地得到一个基本令人满意的解。鉴于标准调度模型的时间复杂度为 $O(n)$，标准调度模型测试程序运行需要的时间非常短，约在秒以内。而此模型的结果为：在 8 小时内共有个成品被送上了下料传送带，如图所示。

八、 遗传算法

8.1 简介

遗传算法在 RGV 调度中有着广泛的应用。Runwei Cheng 等人认为 [**?**] 遗传算法之所以有效，是因为遗传算法能交替在解空间和编码空间之间切换。通过染色体编码将问题转换至编码空间，通过选择、交叉重组和突变等模拟自然遗传的过程寻找最优解，再通过解码将获得的编码转换成解。

8.2 编码方式

染色体的编码决定了计算难度。在车间调度领域，遗传算法的编码方式主要有如下几种：

1. 编码操作
2. 编码任务
3. 偏好队列编码
4. 任务对关系编码
5. 完成时间编码
6. 机器编码
7. 随机密钥编码

在本题中，我们使用的是基于数控机床序号的编码。使用序号编码的好处在于任何编码的顺序都是合法的，编码顺序代表了 RGV 遍历机器的顺序。表 **??** 给出了一个染色体编码的例子。表中的染色体编码说明小车将按 12345678 的路线为 CNC 上下料。

表 6　染色体编码示例

1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8

8.3 解码

在 ga.py 中，decode() 函数负责将染色体编码解释为 RGV 的运动路线和命令。染色体解码遵循以下原则：

1. RGV 会严格地执行染色体编码上的 CNC 遍历顺序。例如，如果 RGV 在编码中指定的 CNC 前就位，但是 CNC 仍在加工工件，RGV 会选择在 CNC 前等待；
2. RGV 会提前在下一台 CNC 前就位。

decode() 函数的返回值是一个函数。该函数能够被用于初始化 World 对象，控制 RGV 小车移动。

8.4 适应度

fitness() 函数计算种群中所有染色体的适应度。每一个染色体的适应度就是染色体对应的 RGV 运动路线在八小时内能够生产的工件的数量。select_new_population() 根据计算得到的适应度选择新一代。所有染色体的适应度使用 Softmax 函数归一，作为该个体在这一代选择中生存下来的概率。使用 Softmax 函数而不是直接用某一染色体的适应度处以所有适应度之和，是因为我们发现在初代中尽管染色体之间相差很大，但由于初代染色体是随机产生的，适应度非常接近。为了将被选择的概率拉开，使用了含有指数的 Softmax。

8.5 重组与突变

模拟重组和突变的函数分别是 crossover() 和 mutation()。crossover() 实现了单染色体交叉点重组。突变会改变染色体上的单个编码。重组与突变的概率均可通过其参数来调整。

8.6 遗传流程

函数 ga() 通过调用前文提及的函数对初始种群进行反复迭代，有以下几步。

1. initial_population() 产生初代种群；
2. 在循环中先计算每一代的适应度，根据适应度新一代被选择出来；
3. 选择得到的新一代进行重组和突变后，输出其适应度，再进入下一代选择。

8.7 遗传算法成果

以下是遗传算法运行样例（只有十代）：

```
0 [140. 141. 141. 141. 141. 141. 141. 142. 144. 144. 145. 145. 145. 146.
 146. 146. 146. 146. 146. 146. 146. 146. 146. 146. 147. 148. 149. 150.
 150. 152.]
1 [144. 145. 146. 146. 146. 146. 146. 146. 146. 148. 148. 148. 149. 150.
 150. 150. 150. 150. 151. 151. 151. 151. 152. 152. 152. 152. 152. 152.
 152. 154.]
2 [146. 148. 149. 149. 149. 150. 150. 150. 151. 152. 152. 152. 152. 152.
 152. 152. 152. 154. 154. 154. 154. 154. 154. 154. 154. 154. 154. 154.
 158. 158.]
3 [152. 152. 153. 154. 154. 154. 154. 154. 154. 154. 154. 154. 154. 154.
 154. 154. 155. 156. 157. 158. 158. 158. 158. 158. 158. 158. 158. 158.
 158. 159.]
4 [152. 153. 154. 154. 154. 154. 154. 154. 155. 155. 156. 156. 157. 158.
 158. 158. 158. 158. 158. 158. 158. 158. 158. 158. 158. 159. 159.
 161. 161.]
5 [154. 154. 154. 155. 156. 157. 157. 158. 158. 158. 158. 158. 158. 158.
 158. 159. 159. 159. 159. 159. 160. 160. 160. 161. 161. 161. 161. 161.
 161. 161.]
6 [154. 156. 158. 158. 158. 158. 158. 158. 158. 158. 158. 158. 158. 158.
 159. 159. 160. 160. 161. 161. 161. 161. 161. 161. 161. 161. 161. 163.
 163. 163.]
7 [156. 157. 159. 160. 161. 161. 161. 161. 161. 161. 161. 161. 162. 163.
 163. 163. 163. 163. 163. 163. 163. 163. 163. 163. 163. 163. 165.
 166. 166.]
8 [158. 161. 161. 161. 161. 162. 163. 163. 163. 163. 163. 163. 163. 163.
 163. 163. 163. 163. 165. 165. 166. 166. 166. 166. 168. 168. 168. 168.
 168. 172.]
9 [163. 163. 163. 165. 167. 168. 168. 168. 168. 168. 168. 168. 168. 168.
 168. 168. 169. 169. 169. 171. 171. 171. 171. 171. 172. 172. 172. 172.
 172. 172.]
```

遗传算法得到的结果见表 **??**。

表 7　遗传算法结果

迭代代数	八小时内加工工件个数
10	172
100	204
500	238

8.8 最优解

经过观察和验证，我们认为我们得到了情况一下一种特殊情况的最优解。当该加工系统的各项时间满足

$$T_0 \geq 3T_{move1} + T_{move3} + 8T_{wash} + 3T_{loadodd} + 4T_{loadeven}$$

时，在标准模型的基础上增加一条原则：

- 当小车没有需要服务的对象而自身空闲时，小车将前往最快要完成的 CNC 旁等候。

该情况下，8 小时内加工完成的工件数达到所有情况下能达到的最大值。接下来我们对其进行证明：

引理 1 *在一定时间 t 内（t 足够大），若所有 CNC 的总等待时间最小，则加工完成的工件数达到最大。*

证明 1 *设在 t 内，所有 CNC 的最小总等待时间为 T，此时加工完成的工件数为 n_t，则有：*

$$8t - n_tT_0 > T \geq Max[0, 8t - n_tT_0 - 8T_0]$$

因此若 $T' > 8t - n_tT_0$，则显然此时 $n'_t < n_t$。
若 $8t - n_tT_0 > T' > T$，而 $n'_t > n_t$，则有 ■

定理 1 *小车空闲时，若有多台 CNC 请求服务，小车应当先应对耗时最短的服务请求。*

证明 2 *设有 k 台 CNC 向小车发送了服务请求，其服务需要的时间分别为 $t_1 < t_2 < ... < t_k$，则小车先为第 i 台 CNC 服务所产生的总等待时间为 $k \cdot t_i$，取 t_1 可使总等待时间最小化，之后由引理 1 即可得证。* ■

由该定理我们验证了小车对不同 CNC 进行排序服务原则的合理性。

定理 2 *小车空闲时，若没有 CNC 请求服务，小车应当前往最快要完成的 CNC 旁等候。*

证明 3 *对任意的 CNC 来说，若小车完成了一次上下料任务，则小车为其他 CNC 完成上下料，再返回当前 CNC 的位置所需的时间最大值为*

$$T_{needMax} = 3T_{move1} + T_{move3} + 8T_{wash} + 3T_{loadodd} + 4T_{loadeven}$$

而下一次该 CNC 需要上下料服务的时间为 $T_0 > T_{needMax}$ 因此在 RGV 完成其他上下料任务后，其应当返回该 CNC 处准备对该 CNC 进行上下料任务。此时，该 CNC 即为最快要完成的 CNC。 ■

对于其他原则，我们认为其十分显然，就不再给出额外的证明。由此，我们可以获知该调度模型即为最优的调度模型。特殊地，对于情况一，该调度模型可以简化为一个周期性调度方案：

RGV 为 CNC1 上下料（含清洗）⇒RGV 为 CNC2 上下料（含清洗）⇒...
⇒RGV 为 CNC8 上下料（含清洗）⇒RGV 回到位置 1
⇒ 等待 CNC1 完成 ⇒RGV 为 CNC1 上下料（含清洗）⇒...

该最优算法可以使加工系统在 8 小时内完成 383(数据组 1) 和 392(数据组 3) 个成品，远远超过了标准模型和遗传算法模型的效率。而数据组 2 虽然在此情况下并非最优解，但是也在 8 小时内完成了 360 个成品，其效率仍然超过了标准模型和遗传算法模型。

九、 情况二的解

对于情况二，CNC 的分配在模型中也是十分重要的一个因素。我们认为 CNC 的分配应当遵从平均原则，即通过给加工时间长的 CNC 赋予更短的平均服务时间和更多的数量来使 CNC 与 RGV 的工作更为协调。因此，对于给出的三组数据：对第一组数据，只需将处理时间长的第一道工序 CNC 放在 1，3，5，7 位置，将第二道工序 CNC 放在 2，4，6，8 位置即可。而对第二组数据，为满足平均原则，有：

$$\frac{x}{280} \approx \frac{8-x}{500}$$

则可得到 $x \approx 2.9$，因此安排 3 台第一道工序 CNC。经过试验，得到将第二道工序 CNC 放在 1，2，3，5，7 位置效率最高。
对第三组数据，同样有：

$$\frac{x}{455} \approx \frac{8-x}{182}$$

可得 $x \approx 5.7$，因此安排 6 台第一道工序 CNC。同样地，经过试验，得到将第一道工序 CNC 放在位置效率最高。

9.1 标准调度模型

同样地，应用在情况二中，标准调度模型依然可以得到一个可以接受的解。用标准调度模型运行情况二的系统分别可以在 8 小时内完成个成品。

9.2 遗传算法

9.3 遗传算法的并行化

遗传算法需要计算每一代个体的适应度。如果将 v_{ch} 作为染色体向量，$v_{ch}^{(i)}$ 作为第 i 个个体的染色体，那么适应度的计算可以抽象成一个函数 $h(v_{ch}^{(i)}) = v_{ch}^{(i)} \to \mathbf{R}$，对于整代

个体来说就是计算 $h(v_{ch}) = [h(v_{vh}^{(1)}), h(v_{vh}^{(2)}), ..., h(v_{vh}^{(k)})]^T$。这一部分计算需要使用模拟世界来得出生产的工件数量，需要消耗较长的时间，因此将这部分内容并行化，加速算法的计算。

实现上使用了 python 的 mutiprocessing 包，使用了其中的进程池 Pool 的 map() 方法来完成遗传算法的并行化。这样，python 解释器会创建多个进程，并行地计算个体的适应度。代码如下：

```
import multiprocessing

cores = multiprocessing.cpu_count()
pool = multiprocessing.Pool(processes=cores)

fitness_values = pool.map(
   Functor,
   chromosome_pairs
)

pool.close()
pool.join()
```

9.4 不同的编码方式

值得注意的是，RGV 在智能加工系统中收到的指令序列常常具有周期性。然而简单的将目标 CNC 的编号作为个体的染色体编码不能体现这种周期性的特征。因此，我们设计了另一种编码，以体现这种周期性的特征。

每个个体的染色体为长度为 p 的向量，向量 p 的元素是等长 (q) 的序列，序列中代表某一个周期内 RGV 小车的目标 CNC 的序列。例如

$$v_{ch}^{(m)} = (\{1, 2, 3, 4, 5, 6, 7, 8\}, \{8, 7, 6, 5, 4, 3, 2, 1\})$$

就是一个长度为 2 的编码。染色体的重组是将序列作为整体进行重组，但是每一个序列内部的顺序没有变化。而染色体的突变就是随机选择编码中的某个序列进行重排 shuffle()，使序列内部的顺序发生变化。

基于这种编码方式，我们重新实现了遗传算法，并且尝试对单阶段加工与二阶段加工进行优化。通过调整编码的长度和序列的长度，我们可以得到相对原有遗传算法较好的结果。

十、 情况三的解

10.1 情况 3-1

情况 3-1 用情况 1 的最优解仍然能够实现。因为机器故障可以视为强制增加的等待时间，因此最优解的证明对情况 3-1 仍然有效。由此可以得到 3-1 的解，在此不再赘述了。

10.2 情况 3-2

对有故障的双工序作业，要通过遗传算法来解决调度问题是十分困难的。我们的遗传算法使用的是确定性序列，而由于故障是随机因素，故障的存在会使得遗传算法的训练受到很大程度的影响，导致优秀的基因被筛去或不好的基因因为故障而被保留。因此使用遗传算法来解决有故障的双工序作业并不能得到理想的解。因此，我们仍然尝试用标准算法来解决 3-2 的情况，

参考文献

[1] 2018 年全国大学生数学建模竞赛 B 题，cumcm.cnki.net，2018.9.13

[2] Cheng, Runwei, Mitsuo Gen, and Yasuhiro Tsujimura. ”A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation.” Computers & industrial engineering 30.4 (1996): 983-997.

附录 A 排队算法–matlab 源程序

```
kk=2;[mdd,ndd]=size(dd);
while ~isempty(V)
[tmpd,j]=min(W(i,V));tmpj=V(j);
for k=2:ndd
[tmp1,jj]=min(dd(1,k)+W(dd(2,k),V));
tmp2=V(jj);tt(k-1,:)=[tmp1,tmp2,jj];
end
tmp=[tmpd,tmpj,j;tt];[tmp3,tmp4]=min(tmp(:,1));
if tmp3==tmpd, ss(1:2,kk)=[i;tmp(tmp4,2)];
else,tmp5=find(ss(:,tmp4)~=0);tmp6=length(tmp5);
if dd(2,tmp4)==ss(tmp6,tmp4)
ss(1:tmp6+1,kk)=[ss(tmp5,tmp4);tmp(tmp4,2)];
else, ss(1:3,kk)=[i;dd(2,tmp4);tmp(tmp4,2)];
end;end
dd=[dd,[tmp3;tmp(tmp4,2)]];V(tmp(tmp4,3))=[];
[mdd,ndd]=size(dd);kk=kk+1;
end; S=ss; D=dd(1,:);
```

```
kk=2;
[mdd,ndd]=size(dd);
while ~isempty(V)
   [tmpd,j]=min(W(i,V));tmpj=V(j);
for k=2:ndd
   [tmp1,jj]=min(dd(1,k)+W(dd(2,k),V));
   tmp2=V(jj);tt(k-1,:)=[tmp1,tmp2,jj];
end
   tmp=[tmpd,tmpj,j;tt];[tmp3,tmp4]=min(tmp(:,1));
if tmp3==tmpd, ss(1:2,kk)=[i;tmp(tmp4,2)];
else,tmp5=find(ss(:,tmp4)~=0);tmp6=length(tmp5);
if dd(2,tmp4)==ss(tmp6,tmp4)
   ss(1:tmp6+1,kk)=[ss(tmp5,tmp4);tmp(tmp4,2)];
else, ss(1:3,kk)=[i;dd(2,tmp4);tmp(tmp4,2)];
end;
end
   dd=[dd,[tmp3;tmp(tmp4,2)]];V(tmp(tmp4,3))=[];
   [mdd,ndd]=size(dd);
   kk=kk+1;
end;
S=ss;
D=dd(1,:);
```