智能RGV的动态调度策略

【摘要】

针对情况一:首先为了能达到*RGV*智能的目标制定了三个原则：第一个是**提前响应**原则，如果所有的*CNC*都在工作状态并且*RGV*即刻进入空闲期那么就让*RGV*提前移动到最快完成的*CNC*的位置；第二个是距离优先原则，主要针对完成工作的*CNC*和*RGV*在同一位置，直接给该*CNC*工作是最优的；第三个是预判原则：当不是*RGV*旁边的*CNC*发出信号(针对*RGV*要移动2个单位及以上的情况)，**预判**移动后会不会有更近的*CNC*发出信号。比较处理完这两个*CNC*的总时间的不同优先处理两种的计划，取最优计划。然后设计算法，通过枚举不同初始上料顺序，得出每种上料顺序在8小时内依据三原则产生的成料个数，可取得最优的答案。其中第一组生产成料数量为382个，第二组生产成料数量为359个，第三组生产成料数量为392个。

针对情况二：根据每台*CNC*只能完成其中的一道工序的要求。可以利用枚举算法枚举 合理的做工序一*CNC*和做工序二*CNC*的比例和在当前比例做第一道工序*CNC*的编号。因为有两道工序，在合理比例下两道工序*CNC*产出成料是接近的，利用这一点设置RGV为先给做第一道工序的*CNC*工作再给工序二*CNC*工作，往后交替进行。每次结合情况一规定的三原则可模拟得出8小时总产量。第一组：第一道工序*CNC*为编号1、3、5、7，第二道工序*CNC*编号为2、4、6、8，生产的成料数量为253个；第二组：第一道工序*CNC*为编号2、4、6、8，第二道工序*CNC*编号为1、3、5、7，生产的成料数量为211个；第三组：第一道工序*CNC*为编号1、2、4、6、7，第二道工序*CNC*编号为3、5、8，生产的成料数量为240个；

针对情况三：利用随机生成算法，在每次下料后随机出此次加工是否会故障，其中利用均匀分布将故障率设置在了百分之一。如果会出现故障再次利用均匀分布随机故障开始时间，其中概率均匀分布在此次加工开始到加工结束。最好利用均匀分布随机维修时间，其中时间均匀分布在600到1200秒。如果*CNC*出现故障，其加工件直接报废，并且重置加工编号和加工完成时间，添加维修完成时间，算法中添加判断维修完成视为正常无加工*CNC*一般。针对一道工序，多次实验取平均解，三组参数生产平均成料数量分别为368.04、338.98、377.19个，平均故障次数分别为4.16、3.46、3.88次；针对两道工序，多次实验取平均解，三组参数生产平均成料数量分别为232.55、200.55、233.72个，平均故障次数分别为4.94、4.16、5次；

关键词：智能预判 提前响应 枚举排序 均匀分布

一、问题重述

1.1 引言

随着工业的发展，出现了越来越多的工厂，以前的工厂使用人力来完成加工，现如今智能化普及逐渐广泛，许多工厂都进入了半智能化或者全智能化，智能化加工可减少人力雇佣和管理费用。

1.2 问题提出

在本文中需根据系统构成和作业流程，结合附件所给系统作业参数的 3 组数据，需针对以下三种不同情况：

1、针对一道工序的物料加工情况，每台 *CNC* 安装相同的刀具，物料可在任意一台 *CNC* 上加工完成。

2、针对两道工序的物料加工情况，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、针对 *CNC* 在加工过程中可能发生故障（故障发生的概率约为 1%）的情况，每次故障排除（人工进行处理，未完成的物料报废）时间介于 10 到 20 分钟之间，故障排除后即刻加入作业序列。

任务一：分别考虑一道工序和两道工序的物料加工情况，建立动态调度模型和相应的求解算法。

任务二：利用系统参数的 3 组数据检验模型的实用性和算法的有效性，并给出相对应 *RGV* 的调度策略和系统的作业效率，并将具体的结果分别填入附件 2 的 *EXCEL* 表中。

二、问题分析

本题主要对智能 *RGV* 制定动态调度策略，根据附件所给系统构成和作业流程，针对三种不同的具体情况，给出 *RGV* 动态调度模型和相应的算法，并利用附件中所给智能加工系统作业参数的 3 组数据对模型的实用性和算法的有效性进行检验，给出相对应的调度策略和系统的作业效率。

2.1 任务一分析

问题一中需针对一道工序、两道工序和故障可能发生的物料加工情况，建立 *RGV* 动态调度模型和设计相应的算法，解决该问题需从以下几个方面进行分析：

① 为了满足一道工序的物料加工情况，需保证每个物料经历从生料到熟料到成料的过程，且从生料到熟料这个过程中 *RGV* 仅需在同一个 *CNC* 中进行上料和下料流程，即可成为熟料，而熟料到成料，需 *RGV* 在原地将熟料

进行清洗工作，清洗完后*RGV*将熟料放入下料传送带中，在整个过程中需保证*RGV*和*CNC*的工作时间都不能超过一个班次的作业时间。

② 为了满足两道工序的物料加工情况，需保证每个物料经历从生料到熟料到成料的过程，且从生料到熟料这个过程中*RGV*需在不同*CNC*中分别进行第一道工序和第二道工序的上料和下料流程，即可成为熟料，而熟料到成料，需*RGV*在原地将熟料进行清洗工作，清洗完后*RGV*将熟料放入下料传送带中，在整个过程中需保证*RGV*和*CNC*的工作时间都不能超过一个班次的作业时间。

③ 考虑进行提前响应，若存在没有*CNC*发出需求信号，则*RGV*需提前去往最快完成加工的*CNC*位置处，等到*CNC*加工完成，*RGV*直接进行上下料工作。

④ 考虑进行预判，当*CNC*发出信号时，若发出信号最近的*CNC*超过两个及两个以上单位的距离，则需进行预判，判断*RGV*附近是否存在*CNC*能在*RGV*前往发出信号最近的*CNC*过程中发出需求信号，若存在，则需对它们的顺序进行制定，选择*RGV*耗费时间最短的上下料顺序，确保*RGV*的工作效率。

⑤ 考虑*CNC*在加工过程中可能发生故障的情况，在*CNC*加工过程中可能会出现故障，每次*CNC*出现故障时，该*CNC*就无法进行加工作业，且正在加工的物料也会报废，据此，设置随机发生故障且发生的概率为1%，若*CNC*出现了故障，故障时间随机在10到20分钟之间，该*CNC*在故障期间无法向*RGV*发出需求信号，也无法进行物料加工工作，只有在故障时间过了之后才能发出需求响应和加工物料。

2.2 任务二分析

任务二中需根据任务一建立的一道工序、两道工序分别考虑*CNC*不发生故障和发生故障的情况建立*RGV*动态调度模型和设计相应的算法，利用附件所给系统作业参数的三*RGV*组数据分别对模型的实用性和算法的有效性进行检验，并给出*RGV*的调度策略和系统的作业效率。

为了检验模型的实用性和算法的有效性，需使用附件所给的智能加工系统作业参数的3组数据，将三组数据分别利用编程进行计算，计算三组分别在一道工序和两道工序考虑不提前响应、提前响应、不发生故障、可能发生故障四种情况的*RGV*调度策略和系统的作业效率，将不同情况下的工作效率进行对比。

三、符号说明

符号	描述说明
S_{ijk}	第i个物料第j道工序在第k个CNC上开始上料时间
T_{ijk}	第i个物料第j道工序在第k个CNC上物料加工完成时间
C_{ijk}	第i个物料第j道工序在第k个CNC上成料时间
f_{ijk}	第i个物料第j道工序是否选择在第k个CNC上进行加工
g_{ijk}	第i个物料第j道工序在第k个CNC上是否进行等待
q_{jk}	第k个CNC是否对第j道工序进行加工
tup_k	第k个CNC上下料时间
tdo_j	第j道工序的加工时间
$move_{k'k}$	第k'个CNC到第k个CNC的移动时间
tw_{ik}	第i个物料在k号CNC上开始故障时间
tb_{ik}	第i个物料在k号CNC上修理时间

四、模型假设

1、假设加工过程中不出现断电情况

2、假设RGV可在一个班次内进行连续工作

3、假设RGV移动过程中未受到阻碍

五、模型建立与求解

本题主要对智能RGV制定动态调度策略。根据附件所给智能加工系统的组成与作业流程，建立RGV动态调度模型和设计相对应的求解算法。并结合附件所给系统参数的3组数据分别检验模型的实用性和算法的有效性，并给出RGV的调度策略和系统的作业参数。

5.1 任务1：RGV动态调度模型、算法

在本问中，需根据附件所给智能加工系统的组成与作业流程，建立RGV动态调度模型和设计相对应的求解算法。

5.1.1 不考虑 CNC 发生故障

不对 *CNC* 在加工过程中可能发生故障进行考虑，即仅需考虑物料经过上料、加工、下料、清洗后成为成料这个过程，针对这个过程建立 *RGV* 动态调度模型和设计相对应的求解算法。

5.1.1.1 RGV 智能预判

当存在 *CNC* 发出需求响应时，需预判移动到该 *CNC* 处时，会不会存在更近的 *CNC* 发出需求响应，据此，需对 *RGV* 移动到发出需求响应的 *CNC* 位置上过程中发出响应需求更近 *CNC* 确定其最优的先后顺序。

据此，需对该情况进行分析，存在 *RGV* 移动到发出需求响应的 *CNC*6 位置过程中其余更近的 *CNC*3 会发出响应，即 $t_3 \le t_2$ 时：

① 当 $t_3 < t_1$ 时，若从 *RGV* 先移动到 *CNC*3 再去往 *CNC*6，除去工作时间，*RGV* 的移动时间为 $2t_1$，反之 *RGV* 的移动时间为 $t_2 + t_1$ 具体如下：

$$t_3 < t_1 \begin{cases} CNC3 \to CNC6 : 2t_1 \\ CNC6 \to CNC3 : t_2 + t_1 \end{cases}$$

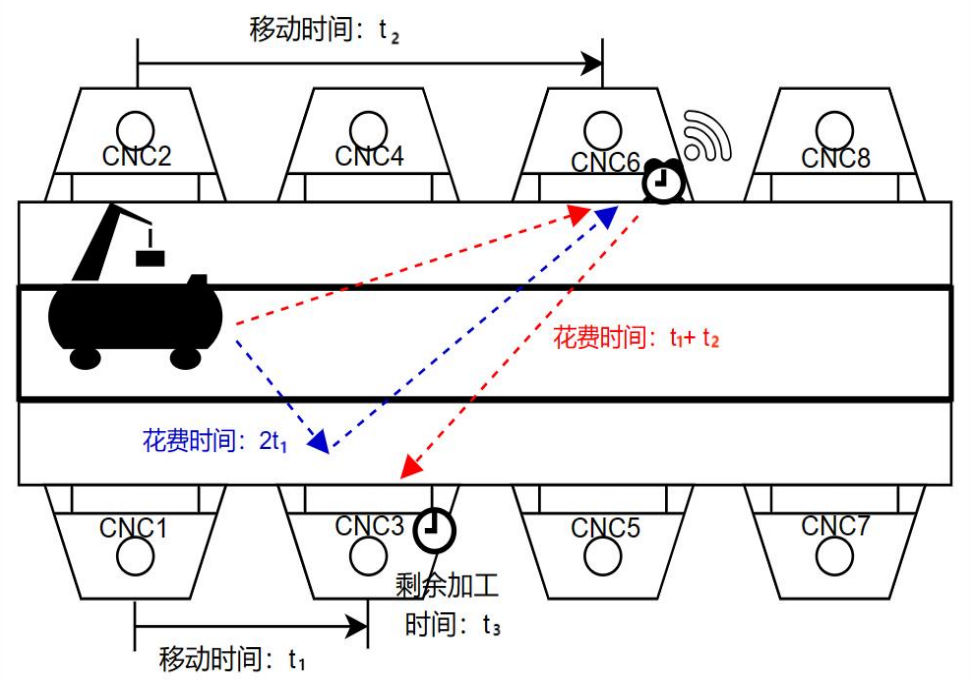

图 1：剩余加工时间小于移动时间时分析图

若 *RGV* 先移动到 *CNC*3 再去往 *CNC*6 的花费时间比先移动到 *CNC*6 再去往 *CNC*3 花费的时间短，则先去 *CNC*3，反之则去 *CNC*6，具体如下：

$$2t_1 < t_2 + t_1 : CNC3 \to CNC6$$
$$2t_1 > t_2 + t_1 : CNC6 \to CNC3$$

② 当 $t_3 > t_1$ 时，若从 *RGV* 先移动到 *CNC*3 再去往 *CNC*6，除去工作时间，

RGV 的移动时间为 t_3+t_1，反之 RGV 的移动时间为 t_2+t_1 具体如下：

$$t_3>t_1\begin{cases}CNC3\rightarrow CNC6:t_3+t_1\\CNC6\rightarrow CNC3:t_2+t_1\end{cases}$$

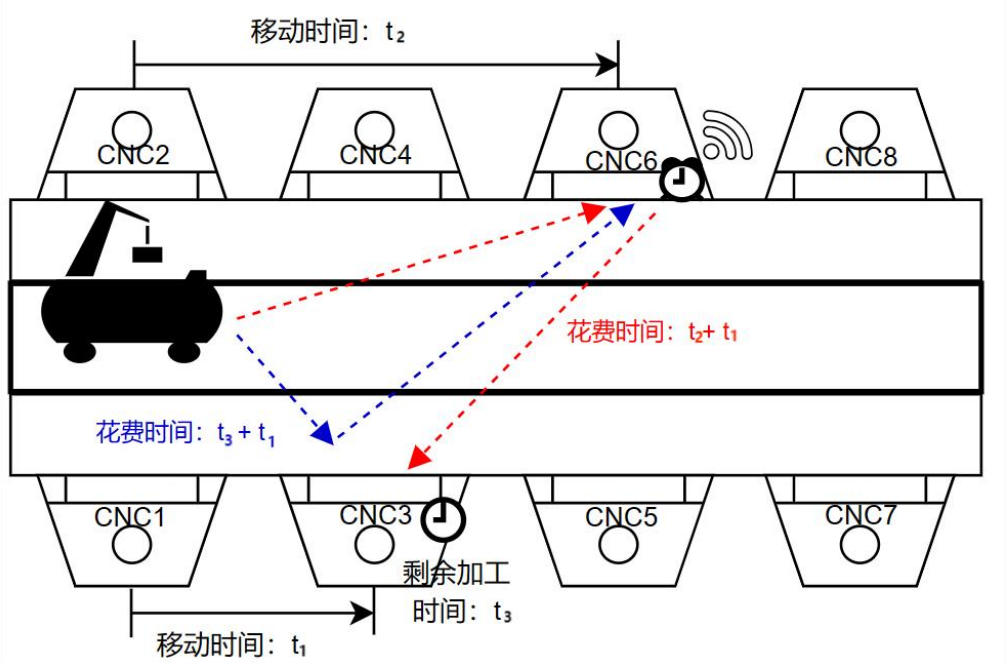

图 2：剩余加工时间大于移动时间时分析图

考虑 RGV 进行等待，RGV 先移动到 $CNC3$ 再去往 $CNC6$ 的花费时间比先移动到 $CNC6$ 再去往 $CNC3$ 花费的时间短，则先去 $CNC3$，反之则去 $CNC6$，具体如下：

$$t_3+t_1<t_2+t_1:CNC3\rightarrow CNC6$$
$$t_3+t_1>t_2+t_1:CNC6\rightarrow CNC3$$

综上，为 RGV 预判过程，当存在不在 RGV 旁边的 CNC 发出需求信号时，即 RGV 需移动两个及两个以上单位的情况下，需对当前 RGV 位置旁边的 CNC 进行预判，比较出更加优化的先后顺序。

5.1.1.2 一道工序的物料加工情况

针对一道工序的物料加工作业情况，每台 CNC 安装相同的刀具，物料仅需经过任意一台 CNC 加工后，即可变成成料。

（1）一道工序 RGV 工作流程

根据智能加工系统的组成与作业流程，方便题目的求解，可绘制一道工序的物料加工情况下的 RGV 工作流程图，具体如下图所示：

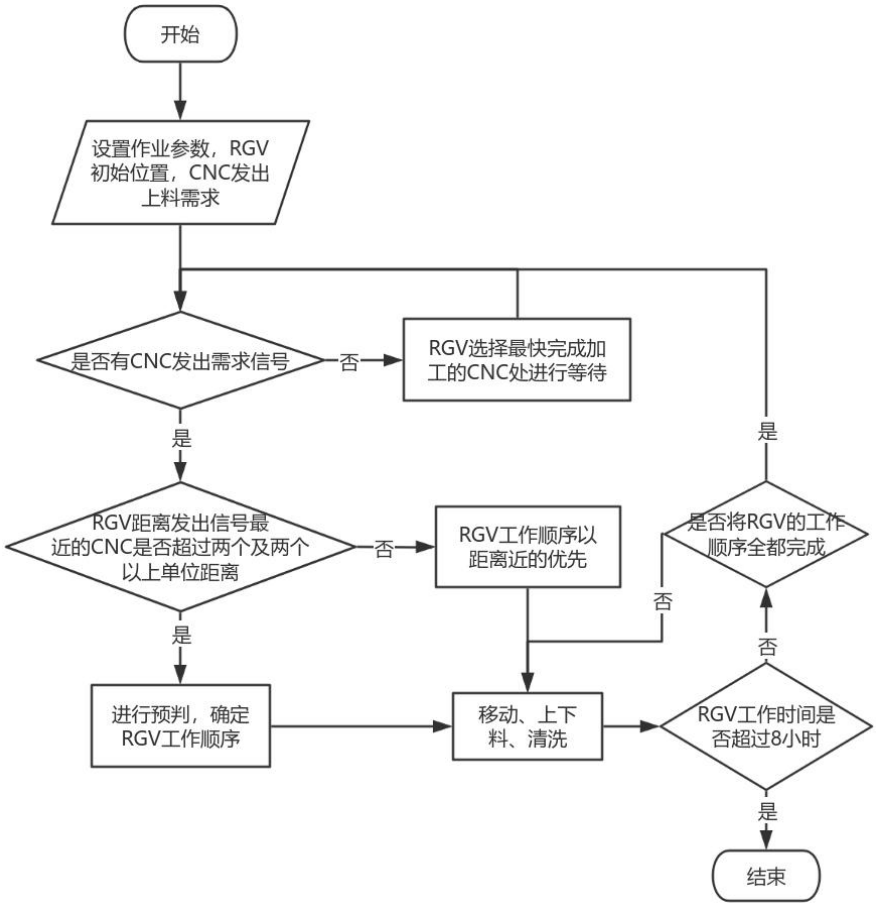

图 3：一道工序的 RGV 工作流程图

智能加工系统起点后，*RGV* 处于 *CNC*1 和 *CNC*2 正中间，此时所有 *CNC* 都向 *RGV* 发出上料需求，第一次上料 *RGV* 先按距离远近进行逐一完成上料需求，完成所有上料需求后，若此时无 *CNC* 发出需求信号，*RGV* 考虑提前响应，提前去到最先完成加工的 *CNC* 位置前，空闲等待 *CNC* 完成加工。

若 *CNC* 发出需求信号，*RGV* 会自行确定上下料顺序，该顺序判别过程在 5.1.1.1 中详细描述，按照顺序逐一进行上下料工作，直至 *RGV* 将顺序内的 *CNC* 工作全部做完，*RGV* 再次判断是否有 *CNC* 发出需求响应，直到 *RGV* 工作时间超过一个班次工作时间时，结束工作。

（2）确定目标

为了更好地制定 *RGV* 动态调度，在一个班次内需确保生产出的成料越多越好。设 *m* 为最后的成料，以最后的成料最大为目标，具体目标如下：

$$Max = m$$

（3）约束条件

根据附件所给智能加工系统的组成与作业流程，为了更好地制定 *RGV* 动态调度策略，需对各个条件进行如下约束：

① “0-1”变量约束

为了制定*RGV*的动态调度策略，可设f_{ik}进行“0-1”约束，当f_{ik}为1时表示第i个物料选择在第k个*CNC*上进行加工，f_{ik}为0时表示第i个物料不选择在第k个*CNC*上进行加工，具体约束如下：

$$f_{ik} \in (0,1) \qquad i=1,2,...,m, k=1,2,...,8$$

② 第一次上料时间约束

*RGV*电启后，所有*CNC*都发出上料需求响应，由于*RGV*启电时位于*CNC*1和*CNC*2的中间位置，则*RGV*从*CNC*1开始进行上料，设S_{ik}为第i个物料在第k个*CNC*上开始上料时间，具体约束如下：

$$S_{(1,1)} = 0$$

其余7个*CNC*的上料时间，应为上一个物料上下料时间加上移动时间，具体约束如下：

$$S_{ik} = S_{(i-1,k')} + move_{k'k} \qquad i=1,2,...,7, k=1,2,...,8$$

③ 物料加工完成约束

物料加工完成时间从该物料开始上料时间进行计算，加上物料上料时间和物料加工时间，设T_{ik}为第i个物料在第k个*CNC*上物料加工完成时间，S_{ik}为第i个物料在第k个*CNC*上开始上料时间，tup_k为第k个*CNC*上下料时间，*tdo*为一道工序的物料加工时间，物料加工完成时间为该物料上料开始时间加上上下料时间和物料加工时间，具体约束如下：

$$T_{ik} = S_{ik} + tup_k + tdo \qquad i=1,2,...,m, k=1,2,...,8$$

④ 上料时间约束

由于*RGV*是同时进行上下料操作的，故同一台*CNC*物料开始时间应为该*CNC*上一个物料开始下料时间，设S_{ik}为第i个物料在第k个*CNC*上开始上料时间，$C_{i'k}$为第k个*CNC*加工的上一个物料的成料时间，*tclean*为熟料清洗时间，物料在*CNC*开始上料时间为该*CNC*上一个物料的成料时间减去相应的清洗时间，具体约束如下：

$$S_{ik} = C_{i'k} - tclean \qquad i=9,10,...,m, k=1,2,...,8$$

⑤ 物料成料完成时间

一道工序的每个物料只有经过上料、加工、下料、清洗工作才能成料，

设 C_{ik} 为第 i 个物料在第 k 个 CNC 加工的成料时间，T_{ik} 为第 i 个物料在第 k 个 CNC 的物料加工完成时间，$tdown_k$ 为第 k 个 CNC 物料加工时间，$tclean$ 为物料清洗时间，$C_{(i',k')}$ 为 RGV 当前在第 k' 个 CNC 加工物料的成料时间，$move_{k'k}$ 为第 k' 个 CNC 到第 k 个 CNC 的移动时间，g_{ik} 为第 i 个物料在第 k 个 CNC 的物料加工是否等待，物料完成时间为该物料的加工完成时间加上下料时间、清洗时间，再加上该 CNC 的等待时间，具体约束如下：

$$C_{ik} = T_{ik} + tdown_k + tclean + \left(C_{(i',k')} + move_{k'k} - T_{ik}\right) \cdot g_{ik} \qquad i = 1,2,...,m, k = 1,2,...,8$$

⑥ CNC 是否等待约束

若 CNC 完成加工工作后，RGV 未及时赶过来对该 CNC 进行上下料工作，则该 CNC 就会进入空闲等待状态，设 g_{ik} 为第 i 个物料在第 k 个 CNC 的物料加工是否等待。

若当前物料完成时间加上 RGV 当前所在位置与第 k 个 CNC 位置移动时间小于第 k 个 CNC 物料加工完成时间，则说明 RGV 可提前前往 CNC 进行等待，等 CNC 加工完成后直接进行上下料工作；若当前物料完成时间加上 RGV 当前所在位置与第 k 个 CNC 位置移动时间不小于第 k 个 CNC 物料加工完成时间，则说明 RGV 来不及在 CNC 加工完成之前赶到，故 CNC 需进行等待；具体约束如下：

$$g_{ik} = \begin{cases} 0, C_{(i',k')} + move_{k'k} - T_{ik} < 0 \\ 1, C_{(i',k')} + move_{k'k} - T_{ik} \geq 0 \end{cases} \qquad i = 1,2,...,m, k = 1,2,...,8$$

⑦ 一个班次工作能力约束

RGV 在一个班次内的工作时间是有限制的，需保证 RGV 在一个班次的工作时间不能超过 8 小时，设 C_{mk} 为最后一块物料 m 在第 k 个 CNC 的成料时间，f_{mk} 为最后一块物料 m 是否选择在第 k 个 CNC 上加工，具体约束如下：

$$\sum_{k=1}^{8} C_{mk} \cdot f_{mk} \leq 28800$$

⑧ 加工物料单一 CNC 约束

RGV 为每个物料选择加工的 CNC 需受到限制，保证每个物料仅有一个 CNC 进行加工，设 f_{ik} 为第 i 个物料是否选择在第 k 个 CNC 上加工，具体约束如下：

$$\sum_{k=1}^{8} f_{ik} = 1 \qquad i = 1,2,...,m$$

⑨ RGV 动态选择物料加工

考虑提前响应和智能预判两种情况，设 $chose_h$ 为第 h 个指令 *RGV* 所在位置，T_{ik} 为第 i 个物料在第 k 个 *CNC* 的物料加工完成时间，*nowtime* 为当前时间，cnc_k 为第 k 个 *CNC* 所在位置，TT_k 为 5.1.1.1 智能预判中 *RGV* 选择的最优开始工作位置。

若所有 *CNC* 中最早的加工完成时间比当前时间大，考虑提前响应，*RGV* 当前指令选择最早加工完成时间的 *CNC*，提前去该 *CNC* 位置上进行等待；若当前时间大于等于 *RGV* 当前所在位置上的物料完成时间，则 *RGV* 当前指令选择对所在位置上的完成物料的 *CNC* 进行工作；

考虑进行预判，*RGV* 需根据 5.1.1.1 智能预判中 *RGV* 选择的最优开始工作位置，具体分析过程详见 5.1.1.1，综上，具体约束如下：

$$chose_h=\begin{cases}k_0,\min(T_{ik})\geq nowtime\wedge T_{ik_0}=\min(T_{ik})\\ k_1,nowtime\geq T_{ik_1}\wedge chose_{h-1}=cnc_{k_1}\\ k',TT_{k'}=\min(TT_{k'})\end{cases}$$

（4）一道工序不发生故障 RGV 动态调度模型的建立与求解

综合以上的目标模型和约束条件，以加工物料的数量最多为目标，建立一道工序不发生故障的 *RGV* 动态调度模型，具体模型如下：

目标：$Max=m$

$$\begin{cases}S_{(1,1)}=0\\ S_{ik}=S_{(i-1,k')}+move_{k'k} & i=1,2,...,7,k=1,2,...,8\\ T_{ik}=S_{ik}+tup_k+tdo & i=1,2,...,m,k=1,2,...,8\\ S_{ik}=C_{i'k}-tclean & i=9,10,...,m,k=1,2,...,8\\ C_{ik}=T_{ik}+tdown_k+tclean+\left(C_{(i',k')}+move_{k'k}-T_{ik}\right)\cdot g_{ik} & i=1,2,...,m,k=1,2,...,8\\ g_{ik}=\begin{cases}0,C_{(i',k')}+move_{k'k}-T_{ik}<0\\ 1,C_{(i',k')}+move_{k'k}-T_{ik}\geq 0\end{cases} & i=1,2,...,m,k=1,2,...,8\\ chose_h=\begin{cases}k_0,\min(T_{ik})\geq nowtime\wedge T_{ik_0}=\min(T_{ik})\\ k_1,nowtime\geq T_{ik_1}\wedge chose_{h-1}=cnc_{k_1}\\ k',TT_{k'}=\min(TT_{k'})\end{cases}\\ \sum_{k=1}^{8}C_{mk}\cdot f_{mk}\leq 28800\\ \sum_{k=1}^{8}f_{ik}=1 & i=1,2,...,m\\ f_{ik}\in(0,1) & i=1,2,...,m,k=1,2,...,8\\ g_{ik}\in(0,1) & i=1,2,...,m,k=1,2,...,8\end{cases}$$

（5）一道工序不故障情况下 RGV 动态调度算法的具体描述

根据以上的一道工序不故障情况下 *RGV* 动态调度模型，设计相对应的动态调度算法，具体伪代码如下：

算法 1: 一道工序RGV工作流程(无故障)

Input: $t, updown[2], T, clean, location, Clocation$

矩阵t表示RGV移动所需时间

$updown_1$表示RGV为奇数编号的CNC上下料所需时间

$updown_0$表示RGV为偶数编号的CNC上下料所需时间

T表示CNC加工完成一个一道工序的物料所需时间

$clean$是RGV清洗时间

$location$表示RGV当前的位置

矩阵$Clocation$表示CNC的位置

Output: ans

ans表示最大加工数量

```
1  begin
       // 枚举初始上料排列,找到最优解
2      arr = [1,2,3,4,5,6,7,8]
3      while next_permutation(arr, arr+size) do
4          solve(arr)
5      end
6      return ans
7  end
8  void solve( vector v)
9  {
10     ending , time = 0
       // 初始上料
11     根据v的顺序依次给对应CNC上料
       // 根据智能调度算法进行模拟
12     while time ≤ 28800 do
13         调用RGV智能调度算法得出下一个要处理的CNC编号k
           time = time + max(t_{Clocation_k,location}, T_k)
14         location = Clocation_k
15         time = time + updown_{k&1}
16         time = time + clean
17         ending = ending + 1
18     end
19     ans = max(ans,ending)
20 }
```

1

算法 1: RGV智能调度算法

Input: $t, updown[2], T, Signal, location, time, Clocation$

矩阵t表示RGV移动所需时间

$updown_1$表示RGV为奇数编号的CNC上下料所需时间

$updown_0$表示RGV为偶数编号的CNC上下料所需时间

T表示CNC加工完成一个一道工序的物料所需时间

矩阵$Signal$是每个CNC关于时间的信号矩阵

$location$表示RGV当前的位置

矩阵$Clocation$表示CNC的位置

$time$表示当前时间

Output: ans

ans表示RGV下一个要去的CNC的编号

begin

// 提前响应原则

if $|Signal_{1:8,time}| = 0$ **then**

index = 1

for $i \leftarrow 2$ **to** 8 **do**

if $Singal_{i,time} < Singal_{index,time}$ **then**

$index = i$

end

end

return $ans \leftarrow index$

end

// 距离优先原则

for $i \leftarrow 1$ **to** 8 **do**

if $Singal_{i,time}$ *is true and* $location = Clocation_i$ **then**

return $ans \leftarrow i$

end

end

// 智能体现原则(预判)

将目前离RGV最近且完成工作的CNC定为k_0

$timet = time + t_{location,Cloccation_{k_0}} + updown_{k_0\&1}$

找出timet时间内加工完成且离RGV更近的CNCk_1

$time2 = time + max(T_{k_1}, t_{location,Cloccation_{k_1}}) + updown_{k_1\&1}$

$time2 = time2 + t_{Cloccation_{k_1},Cloccation_{k_0}} + updown_{k_0\&1}$

$timet = timet + t_{Cloccation_{k_1},Cloccation_{k_0}} + updown_{k_1\&1}$

if $timet \leq time2$ **then**

return $ans \leftarrow k_0$

else

return $ans \leftarrow k_1$

end

end

5.1.1.3 两道工序的物料加工情况

针对两道工序的物料加工作业情况，物料加工过程中需有两道工序，则需有不同的*CNC*安装不同的刀具分别加工完成，且加工过程中不能更换刀具，物料需经过两道工序才可变成成料。

（1）两道工序 RGV 工作流程

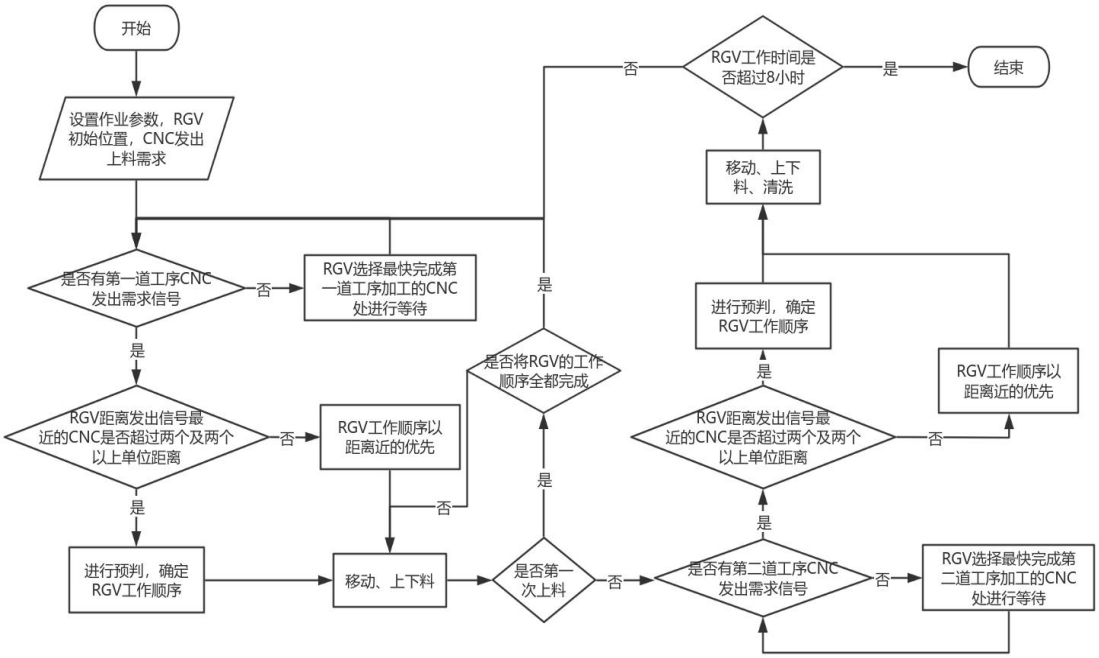

图 4：两道工序的 RGV 工作流程图

智能加工系统起点后，*RGV* 处于*CNC*1和*CNC*2正中间，此时所有*CNC*都向*RGV*发出上料需求，第一次上料*RGV*先按距离远近进行逐一完成第一道工序的上料需求，完成所有上料需求后，若此时无第一道工序*CNC*发出需求信号，*RGV*考虑提前响应，提前去到最先完成加工的第一道工序*CNC*位置前，空闲等待该*CNC*完成加工。

若第一道工序*CNC*发出需求信号，*RGV*会自行确定上下料顺序，该顺序判别过程在 5.1.1.1 中详细描述，按照顺序逐一进行上下料工作，将第一道工序*CNC*加工完成的物料送到第二道工序*CNC*中进行加工，该顺序由*RGV*自行确定，直至*RGV*将顺序内的*CNC*工作全部做完，*RGV*再次判断是否有*CNC*发出需求响应，直到*RGV*工作时间超过一个班次工作时间时，结束工作。

（2）确定目标

为了更好地制定*RGV*动态调度，在一个班次内需确保生产出的成料越多越好。设*m*为最后的成料，以最后的成料最大为目标，具体目标如下：

$$Max = m$$

（3）约束条件

根据附件所给智能加工系统的组成与作业流程，为了更好地制定 *RGV* 动态调度策略，需对部分条件进行约束，部分 5.1.1.3 已有的约束条件不再赘述，其他具体约束条件如下：

① “0-1”变量约束

为了制定 *RGV* 的动态调度策略，可设 f_{ijk} 进行“0-1”约束，当 f_{ijk} 为 1 时表示第 i 个物料第 j 道工序选择在第 k 个 *CNC* 上进行加工，f_{ijk} 为 0 时表示第 i 个物料第 j 道工序不选择在第 k 个 *CNC* 上进行加工，具体约束如下：

$$f_{ijk} \in (0,1) \qquad i=1,2,...,m, j=1,2, k=1,2,...,8$$

为了制定 *RGV* 的动态调度策略，可设 g_{ijk} 进行“0-1”约束，当 g_{ijk} 为 1 时表示第 i 个物料第 j 道工序在第 k 个 *CNC* 上进行加工时进行等待，g_{ijk} 为 0 时表示第 i 个物料第 j 道工序在第 k 个 *CNC* 上进行加工时不等待，具体约束如下：

$$g_{ijk} \in (0,1) \qquad i=1,2,...,m, j=1,2, k=1,2,...,8$$

② 第一次上料时间约束

RGV 电启后，所有 *CNC* 都发出上料需求响应，由于 *RGV* 启电时位于 *CNC* 1 和 *CNC* 2 的中间位置，则 *RGV* 从 *CNC* 1 开始进行上料，设 S_{ijk} 为第 i 个物料第 j 道工序在第 k 个 *CNC* 上开始上料时间，$move_{k'k}$ 为第 k' 个 *CNC* 到第 k 个 *CNC* 的移动时间，具体约束如下：

$$S_{(1,1,1)} = 0$$

其余 7 个 *CNC* 的上料时间，应为上一个物料上下料时间加上移动时间，具体约束如下：

$$S_{(i,1,k)} = S_{(i-1,1,k')} + move_{k'k} \qquad i=2,3,...,8, k=1,2,...,8$$

③ 物料加工完成约束

物料加工完成时间从该物料开始上料时间进行计算，加上物料上料时间和物料加工时间，设 T_{ijk} 为第 i 个物料第 j 道工序在第 k 个 *CNC* 上物料加工完成时间，S_{ijk} 为第 i 个物料第 j 道工序在第 k 个 *CNC* 上开始上料时间，tup_k 为第 k 个 *CNC* 上下料时间，tdo_j 为第 j 道工序的加工时间，物料加工完成时间为该物料上料开始时间加上上下料时间和物料加工时间，具体约束如下：

$$T_{ijk} = S_{ijk} + tup_k + tdo_j \qquad i=1,2,...,m, j=1,2, k=1,2,...,8$$

④ 上料时间约束

由于 *RGV* 是同时进行上下料操作的，故同一台 *CNC* 物料开始时间应为该 *CNC* 上一个物料开始下料时间，设 S_{ijk} 为第 i 个物料第 j 道工序在第 k 个 *CNC* 上开始上料时间，$C_{i'jk}$ 为第 k 个 *CNC* 加工的上一个物料的成料时间，*tclean* 为熟料清洗时间。

第一道工序不需要进行清洗工作，物料加工完成后上下料就可完成第一道工序的加工工作，即物料在 *CNC* 开始上料时间为该 *CNC* 上一个物料的成料时间，具体约束如下：

$$S_{(i,1,k)} = C_{(i',1,k)} \qquad i = 9,10,...,m, k = 1,2,...,8$$

第二道工序，加工结束后还需进行清洗工作，故第二道工序的上料开始时间为该 *CNC* 上一个物料的成料时间减去相应的清洗时间，具体约束如下：

$$S_{(i,2,k)} = C_{(i',2,k)} - tclean \qquad i = 9,10,...,m, k = 1,2,...,8$$

⑤ 物料成料完成时间

一道工序的每个物料经过上料、加工、下料工作即表示完成第一道工序的物料加工工作，设 C_{ijk} 为第 i 个物料第 j 道工序在第 k 个 *CNC* 加工的成料时间，T_{ijk} 为第 i 个物料第 j 道工序在第 k 个 *CNC* 的物料加工完成时间，$tdown_k$ 为第 k 个 *CNC* 的上下料时间，*tclean* 为物料清洗时间，$C_{(i',j,k')}$ 为 *RGV* 当前第 j 道工序在第 k' 个 *CNC* 加工物料的成料时间，$move_{k'k}$ 为第 k' 个 *CNC* 到第 k 个 *CNC* 的移动时间，g_{ijk} 为第 i 个物料第 j 道工序在第 k 个 *CNC* 的物料加工是否等待，第一道工序的物料完成时间为该物料的加工完成时间加上下料时间，再加上其 *CNC* 等待时间，具体约束如下：

$$C_{(i,1,k)} = T_{(i,1,k)} + tdown_k + \left(C_{(i',1,k')} + move_{k'k} - T_{(i,1,k)}\right) \cdot g_{(i,1k)} \qquad i = 1,2,...,m, k = 1,2,...,8$$

第二道工序的每个物料只有经过上料、加工、下料、清洗工作才能成料，故第二道工序的物料完成时间为该物料的加工完成时间加上下料时间、清洗时间，再加上其 *CNC* 等待时间，*CNC* 等待时间又与 *RGV* 上一个成料完成时间和两个 *CNC* 之间的单位时间有关，具体约束如下：

$$C_{(i,2,k)} = T_{(i,2,k)} + tdown_k + tclean + \left(C_{(i',2,k')} + move_{k'k} - T_{(i,2,k)}\right) \cdot g_{(i,2,k)} \quad i = 1,2,...,m, k = 1,2,...,8$$

⑥ 一个班次工作能力约束

RGV 在一个班次内的工作时间是有限制的，需保证 *RGV* 在一个班次的工作时间不能超过 8 小时，设 $C_{(m,2,k)}$ 为最后一块物料 m 第二道工序在第 k 个

CNC 的成料时间，$f_{(m,2,k)}$为最后一块物料 m 第二道工序是否选择在第 k 个 *CNC* 上加工，具体约束如下：

$$\sum_{k=1}^{8} C_{(m,2,k)} \cdot f_{(m,2,k)} \le 28800$$

⑦ 加工物料单一 CNC 约束

RGV 为每个物料选择加工的 *CNC* 需受到限制，保证每个物料每道工序都仅有一个 *CNC* 进行加工，设 f_{ijk} 为第 i 个物料第 j 道工序是否选择在第 k 个 *CNC* 上加工，具体约束如下：

$$\sum_{k=1}^{8} f_{ijk} = 1 \qquad i = 1,2,...,m, j = 1,2$$

⑧ CNC 两道工序刀具限制

由于两道工序需要有不同的 *CNC* 安装不同的刀具分别加工完成，即 *CNC* 有两种不同刀具，各 *CNC* 分别需要安装这两种刀具的其中之一，设 α 为第一道工序的 *CNC* 数量，β 为第二道工序的 *CNC* 数量，q_{jk} 为第 k 个 *CNC* 是否对第 j 道工序进行加工。则一道工序的 *CNC* 数量限制如下所示：

$$\sum_{k=1}^{8} q_{(1,k)} = \alpha$$

两道工序的 *CNC* 数量限制如下所示：

$$\sum_{k=1}^{8} q_{(2,k)} = \beta$$

各 *CNC* 仅能安装加工一道工序或两道工序的刀具，同一个 *CNC* 不能安装两个刀具，且每个 *CNC* 都要安装刀具，具体约束如下；

$$\sum_{j=1}^{2} q_{jk} = 1 \qquad k = 1,2,...,8$$

一道工序和两道工序的 *CNC* 数量限制，需满足两道工序的 *CNC* 数量之和为 8，具体如下所示：

$$\alpha + \beta = 8$$

RGV 为物料两道不同工序选择 *CNC* 进行物料加工，受到该 *CNC* 是否安装该工序的刀具的影响，若 *CNC* 安装第一道工序的刀具，则 *RGV* 可选择该 *CNC* 加工第一道工序，若 *CNC* 不安装第一道工序的刀具，即安装第二道工序，

则 *RGV* 无法选择该 *CNC* 进行第一道工序，具体约束如下：

$$f_{ijk} \le q_{jk} \qquad i=1,2,...,m,\ j=1,2,\ k=1,2,...,8$$

（4）两道工序不发生故障 RGV 动态调度模型的建立与求解

综合以上的目标模型和约束条件，以成料数量最多为目标，建立两道工序不发生故障的 *RGV* 动态调度模型，具体模型如下：

目标：$Max = m$

$$\begin{cases}
S_{(1,1,1)} = 0 & \\
S_{(i,1,k)} = S_{(i-1,1,k')} + move_{k'k} & i=1,2,...,7,\ k=1,2,...,8 \\
T_{ijk} = S_{ijk} + tup_k + tdo_j & i=1,2,...,m,\ j=1,2,\ k=1,2,...,8 \\
S_{(i,1,k)} = C_{(i',1,k)} & i=9,10,...,m,\ k=1,2,...,8 \\
S_{(i,2,k)} = C_{(i',2,k)} - tclean & i=9,10,...,m,\ k=1,2,...,8 \\
C_{(i,1,k)} = T_{(i,1,k)} + tdown_k + \left(C_{(i',1,k')} + move_{k'k} - T_{(i,1,k)}\right)\cdot g_{(i,1k)} & i=1,2,...,m,\ k=1,2,...,8 \\
C_{(i,2,k)} = T_{(i,2,k)} + tdown_k + tclean + \left(C_{(i',2,k')} + move_{k'k} - T_{(i,2,k)}\right)\cdot g_{(i,2,k)} & i=1,2,...,m,\ k=1,2,...,8 \\
g_{ijk} = \begin{cases} 0, C_{(i',j,k')} + move_{k'k} - T_{ijk} < 0 \\ 1, C_{(i',j,k')} + move_{k'k} - T_{ijk} \ge 0 \end{cases} & i=1,2,...,m,\ j=1,2,\ k=1,2,...,8 \\
chose_h = \begin{cases} k_0, \min(T_{ik}) \ge nowtime \wedge T_{ik_0} = \min(T_{ik}) \\ k_1, nowtime \ge T_{ik_1} \wedge chose_{h-1} = cnc_{k_1} \\ k', TT_{k'} = \min(TT_{k'}) \end{cases} & h=1,2,...,n \\
\sum_{k=1}^{8} C_{(m,2,k)} \cdot f_{(m,2,k)} \le 28800 & \\
\sum_{k=1}^{8} q_{(1,k)} = \alpha & \\
\sum_{k=1}^{8} q_{(2,k)} = \beta & \\
\alpha + \beta = 8 & \\
\sum_{j=1}^{2} q_{jk} = 1 & k=1,2,...,8 \\
\sum_{k=1}^{8} f_{ijk} = 1 & i=1,2,...,m,\ j=1,2 \\
f_{ijk} \le q_{jk} & i=1,2,...,m,\ j=1,2,\ k=1,2,...,8 \\
f_{ijk} \in (0,1) & i=1,2,...,m,\ j=1,2,\ k=1,2,...,8 \\
g_{ijk} \in (0,1) & i=1,2,...,m,\ j=1,2,\ k=1,2,...,8 \\
q_{jk} \in (0,1) & j=1,2,\ k=1,2,...,8
\end{cases}$$

（5）两道工序不故障情况下 RGV 动态调度算法的具体描述

根据以上的两道工序不故障情况下 *RGV* 动态调度模型，设计相对应的动态调度算法，具体伪代码如下：

算法 1: 两道工序RGV工作流程(无故障)

Input: $t, updown[2], T, clean, location, Clocation, proportion, cncnow$

矩阵t表示RGV移动所需时间

$updown_1$表示RGV为奇数编号的CNC上下料所需时间

$updown_0$表示RGV为偶数编号的CNC上下料所需时间

矩阵T表示CNC加工完成一个两道工序物料的指定的工序所需时间

$clean$是RGV清洗时间

$location$表示RGV当前的位置

矩阵$Clocation$表示CNC的位置

$proportion$表示加工第一道工序的CNC的个数(人为输入)

矩阵$cncnow$表示当前CNC上面的工件编号

Output: ans

ans表示最大加工数量

```
1  begin
       // 枚举工作第一道工序的CNC编号排列,找到最优解
2      arr = [1,2,3,4,5,6,7,8]
3      while next_permutation(arr, arr+size) do
4          solve(arr of porportion numsize)
5      end
6      return ans
7  end
8  void solve( int v[proportion])
9  {
10     ending , time , flag , RGVhand = 0
       // 初始上料
11     根据v的顺序依次给对应CNC上料
       // 根据智能调度算法进行模拟
12     while time ≤ 28800 do
13         调用RGV智能调度算法(2)得出下一个要处理的CNC编号k
           time = time + max(t_{Clocation_k,location}, T_k)
14         location = Clocation_k
15         RGVhand = cncnow_k
16         cncnow_k = flag?RGVhand : k_new
17         time = time + updown_{k&1}
18         if flag is true then
19             time = time + clean ; ending = ending + 1
20         end
21         flag = flag ⊕ 1
22     end
23     ans = max(ans,ending)
24 }
```

算法 1: RGV智能调度算法(2)

Input: $t, updown[2], T, Signal, location, time, Clocation, flag, cncflag$

矩阵t表示RGV移动所需时间

矩阵$updown$表示RGV为奇偶数编号的CNC上下料所需时间

T表示CNC加工完成一个一道工序的物料所需时间

矩阵$Signal$是每个CNC关于时间的信号矩阵

$location$表示RGV当前的位置

矩阵$Clocation$表示CNC的位置

$time$表示当前时间

$flag$表示传入的标志,代表下一个找的是哪一个工序的cnc

矩阵$cncflag$表示cnc工作的工序

Output: ans

ans表示RGV下一个要去的CNC的编号

begin
  if $|Signal_{1:8,time}| = 0$ **then**
    index = The first one that meets the requirements with flag
    for $i \leftarrow 2$ **to** 8 **do**
      if $Singal_{i,time} < Singal_{index,time}$ *and* $cncflag_i \& flag$ **then**
        $index = i$
      end
    end
    return $ans \leftarrow index$
  end
  // 距离优先原则
  for $i \leftarrow 1$ **to** 8 *and* $cncflag_i \& flag$ **do**
    if $Singal_{i,time}$ *is true and* $location = Clocation_i$ **then**
      return $ans \leftarrow i$
    end
  end
  // 智能体现原则(预判)
  将目前离RGV最近且完成工作且满足flag要求的CNC定为k_0
  $timet = time + t_{location,Cloccation_{k_0}} + updown_{k_0\&1}$
  找出timet时间内加工完成且离RGV更近且满足要求的CNCk_1
  $time2 = time + max(T_{k_1}, t_{location,Cloccation_{k_1}}) + updown_{k_1\&1}$
  $time2 = time2 + t_{Cloccation_{k_1},Cloccation_{k_0}} + updown_{k_0\&1}$
  $timet = timet + t_{Cloccation_{k_1},Cloccation_{k_0}} + updown_{k_1\&1}$
  if $timet \leq time2$ **then**
    return $ans \leftarrow k_0$
  else
    return $ans \leftarrow k_1$
  end
end

5.1.2 考虑 CNC 发生故障

对 *CNC* 在加工过程中可能发生故障进行考虑，针对此情况建立 *RGV* 动态调度模型和设计相对应的求解算法。

5.1.2.1 CNC 维修时长与发生故障时刻概率模型

故障条件下的 *RGV* 动态调度模型除了具有情况一二中 *RGV* 动态调度模型的特点和约束之外，生产系统中的机器可能发生故障。发生故障的机器停止处理任务，进行机器的修复工作。发生故障时机器上的物料被丢弃。机器完成修复后可继续工作。本题假设 *CNC* 的故障时间和维修时间服从均匀分布。

维修时间的均匀分布的概率密度函数，维修时长在 10 分钟到 20 分钟之间。

$$tBf_{ijk}(t)=\begin{cases}0 & ，其他\\ \dfrac{1}{1200-600} & ，600\le t\le 1200\end{cases}$$

维修时长的均匀分布的分布函数

$$tBF_{ijk}(t)=\begin{cases}0 & t<600\\ \dfrac{x-600}{1200-600} & 600\le t\le 1200\\ 1 & t>1200\end{cases}$$

tb：维修时长服从以上均匀分布

发生故障时刻的均匀分布的概率密度函数

$$tWf_{ijk}(t)=\begin{cases}0 & ，其他\\ \dfrac{1}{C_{ijk}-S_{ijk}} & ，C_{ijk}\le t\le S_{ijk}\end{cases}$$

发生故障时刻的均匀分布的分布函数

$$tWF_{ijk}(t)=\begin{cases}0 & t<S_{ijk}\\ \dfrac{t-S_{ijk}}{C_{ijk}-S_{ijk}} & S_{ijk}\le t\le C_{ijk}\\ 1 & t>C_{ijk}\end{cases}$$

tw:发生故障开始时间

p_{ijk}：表示加工 i 号物料 j 个步骤在 k 号 *CNC* 是否发生故障

5.1.2.2 一道工序的物料加工情况

针对一道工序的物料加工作业情况，对 *CNC* 在加工过程中可能发生故障进行考虑，每台 *CNC* 安装相同的刀具，物料仅需经过任意一台 *CNC* 加工后，即可变成成料。

（1）确定目标

为了更好地制定 *RGV* 动态调度，在一个班次内需确保生产出的成料越多越好。设 m 为最后的成料，以生产的成料数量最多为目标，即最后的成料最大为目标，具体目标如下：

$$Max = m$$

（2）约束条件

根据附件所给智能加工系统的组成与作业流程，为了更好地制定 *RGV* 动态调度策略，需对部分条件进行约束，部分 5.1.1.2 已有的约束条件不再赘述，其他具体约束条件如下：

① “0-1”变量约束

为了制定 *RGV* 的动态调度策略，可设 f_{ik} 进行“0-1”约束，当 f_{ik} 为 1 时表示第 i 个物料选择在第 k 个 *CNC* 上进行加工，f_{ik} 为 0 时表示第 i 个物料不选择在第 k 个 *CNC* 上进行加工，具体约束如下：

$$f_{ik} \in (0,1) \qquad i = 1,2,...,m, k = 1,2,...,8$$

为了制定 *RGV* 的动态调度策略，可设 g_{ik} 进行“0-1”约束，当 g_{ik} 为 1 时表示第 i 个物料在第 k 个 *CNC* 上进行加工时进行等待，g_{ik} 为 0 时表示第 i 个物料在第 k 个 *CNC* 上进行加工时不等待，具体约束如下：

$$g_{ik} \in (0,1) \qquad i = 1,2,...,m, k = 1,2,...,8$$

② 损坏 CNC 重新开始上料时间

当 *CNC* 发生损坏时，*CNC* 当前物料受到损坏，且该 *CNC* 需停止当前工作，进行人工维修，设 $S_{i''k}$ 为出现故障 k 号 *CNC* 加工下一个物料的上料时间，tw_{ik} 为第 i 个物料在 k 号 *CNC* 上开始故障时间，tb_{ik} 为第 i 个物料在 k 号 *CNC* 上修理时间，可得到下一个物料开始时间为故障开始时间加上物料修理时间，具体约束如下：

$$S_{i''k} = tw_{ik} + tb_{ik}$$

③ 故障 CNC 上料时间约束

当 *CNC* 发生故障时，*CNC* 的物料完成时间应与不发生故障的物料完成时间不同，设 T_{ik} 为第 i 个物料在第 k 个 *CNC* 加工完成时间，p_{ik} 为第 i 个物料在第 k 个 *CNC* 是否发生故障，$tdown_k$ 为第 k 个 *CNC* 的上下料时间，*tclean* 为物料清洗时间。

发生故障时，物料的完成时间为下一个物料的上料时间减去下料时间和

清洗时间，具体约束如下：

$$T_{ik}=(S_{i''k}-tdown_k-tclean)p_{ik}$$

不发生故障时，物料的完成时间为该物料开始上料时间加上下料时间和物料加工时间，具体约束如下：

$$T_{ik}=S_{ik}+tup_k+tdo$$

（3）一道工序发生故障 RGV 动态调度模型的建立与求解

综合以上的目标模型和约束条件，以加工物料的数量最多为目标，建立一道工序发生故障的 *RGV* 动态调度模型，具体模型如下：

目标：$Max=m$

$$\begin{cases}
S_{(1,1)}=0 & \\
S_{ik}=S_{(i-1,k')}+move_{k'k} & i=1,2,...,7,k=1,2,...,8\\
S_{ik}=C_{i'k}-tclean & i=9,10,...,m,k=1,2,...,8\\
C_{ik}=T_{ik}+tdown_k+tclean+\left(C_{(i',k')}+move_{k'k}-T_{ik}\right)\cdot g_{ik} & i=1,2,...,m,k=1,2,...,8\\
g_{ik}=\begin{cases}0,C_{(i',k')}+move_{k'k}-T_{ik}<0\\1,C_{(i',k')}+move_{k'k}-T_{ik}\geq 0\end{cases} & i=1,2,...,m,k=1,2,...,8\\
chose_h=\begin{cases}k_0,\min(T_{ik})\geq nowtime\wedge T_{ik_0}=\min(T_{ik})\\k_1,nowtime\geq T_{ik_1}\wedge chose_{h-1}=cnc_{k_1}\\k',TT_{k'}=\min(TT_{k'})\end{cases} & \\
\sum_{k=1}^{8}C_{mk}\cdot f_{mk}\leq 28800 & \\
\sum_{k=1}^{8}f_{ik}=1 & i=1,2,...,m\\
S_{i''k}=tw_{ik}+tb_{ik} & i=1,2,...,m,k=1,2,...,8\\
T_{ik}=\begin{cases}(S_{i''k}-tdown_k-tclean)p_{ik} & ,p_{ik}=1\\S_{ik}+tup_k+tdo & ,p_{ik}=0\end{cases} & i=1,2,...,m,k=1,2,...,8\\
f_{ik}\in(0,1) & i=1,2,...,m,k=1,2,...,8\\
g_{ik}\in(0,1) & i=1,2,...,m,k=1,2,...,8
\end{cases}$$

（4）一道工序故障情况下 RGV 动态调度算法的具体描述

根据以上的一道工序故障情况下 *RGV* 动态调度模型，设计相对应的动态调度算法，具体伪代码如下：

算法 1: 一道工序RGV工作流程(有故障)

Input: $t, updown[2], T, clean, location, Clocation$

Output: ans

```
begin
    // 枚举初始上料排列,找到最优解
    arr = [1,2,3,4,5,6,7,8]
    while next_permutation(arr, arr+size) do
        solve(arr)
    end
    return ans
end
void solve( vector v)
{
    ending , time = 0
    // 初始上料
    根据v的顺序依次给对应CNC上料
    // 根据智能调度算法进行模拟
    while time ≤ 28800 do
        调用RGV智能调度算法3得出下一个要处理的CNC编号k
        time = time + max(t_{Clocation_k,location}, T_k)
        location = Clocation_k
        time = time + updown_{k&1}
        time = time + clean
        ending = ending + 1
        给当前的CNC随机是否故障和故障时间以及维修时间
        记录故障各个数据，将当前CNC上物料后续工作时间都设
         为-1
    end
    ans = max(ans,ending)
}
```

5.1.2.3 两道工序的物料加工情况

两道工序的物料加工作业情况下， 对 *CNC* 在加工过程中可能发生故障进行考虑，需考虑物料经过一道工序上料、一道工序加工、一道工序下料、二道工序上料、二道工序加工、二道工序下料、清洗后成为成料这个过程，且该过程中可能会出现 *CNC* 发生故障的情况，针对该情况建立 *RGV* 动态调度模型和设计相对应的求解算法。

（1）确定目标

为了更好地制定 *RGV* 动态调度，在一个班次内需确保生产出的成料越多

越好。设 m 为最后的成料，以最后的成料最大为目标，具体目标如下：

$$Max = m$$

（2）约束条件

根据附件所给智能加工系统的组成与作业流程，为了更好地制定 *RGV* 动态调度策略，需对部分条件进行约束，部分 5.1.1.3 已有的约束条件不再赘述，其他具体约束条件如下：

① “0-1”变量约束

为了制定 *RGV* 的动态调度策略，可设 f_{ijk} 进行“0-1”约束，当 f_{ijk} 为 1 时表示第 i 个物料第 j 道工序选择在第 k 个 *CNC* 上进行加工， f_{ijk} 为 0 时表示第 i 个物料第 j 道工序不选择在第 k 个 *CNC* 上进行加工，具体约束如下：

$$f_{ijk} \in (0,1) \qquad i = 1,2,...,m, j = 1,2, k = 1,2,...,8$$

为了制定 *RGV* 的动态调度策略，可设 g_{ijk} 进行“0-1”约束，当 g_{ijk} 为 1 时表示第 i 个物料第 j 道工序在第 k 个 *CNC* 上进行加工时进行等待， g_{ijk} 为 0 时表示第 i 个物料第 j 道工序在第 k 个 *CNC* 上进行加工时不等待，具体约束如下：

$$g_{ijk} \in (0,1) \qquad i = 1,2,...,m, j = 1,2, k = 1,2,...,8$$

② 故障 CNC 上料时间约束

当 *CNC* 发生故障时，*CNC* 的物料完成时间应与不发生故障的物料完成时间不同，设 T_{ijk} 为第 i 个物料第 j 道工序在第 k 个 *CNC* 加工完成时间， p_{ijk} 为第 i 个物料第 j 道工序在第 k 个 *CNC* 是否发生故障，$tdown_k$ 为第 k 个 *CNC* 的上下料时间， *tclean* 为物料清洗时间。

发生故障时，物料的完成时间为下一个物料的上料时间减去下料时间和清洗时间，具体约束如下：

$$T_{ijk} = (S_{i''jk} - tdown_k - tclean)\, p_{ijk}$$

不发生故障时，物料的完成时间为该物料开始上料时间加上下料时间和物料加工时间，具体约束如下：

$$T_{ijk} = S_{ijk} + tup_{1k} + tdo_j$$

③ 损坏 CNC 重新开始上料时间

当 *CNC* 发生损坏时， *CNC* 当前物料受到损坏，且该 *CNC* 需停止当前工作，进行人工维修，设 $S_{i''jk}$ 为第 j 道工序出现故障 k 号 *CNC* 加工下一个物料

的上料时间，tw_{ijk} 为第 i 个物料第 j 道工序在 k 号 *CNC* 上开始故障时间，tb_{ijk} 为第 i 个物料第 j 道工序在 k 号 *CNC* 上修理时间，可得到下一个物料开始时间为故障开始时间加上物料修理时间，具体约束如下：

$$S_{i''jk} = tw_{ijk} + tb_{ijk}$$

（3）两道工序发生故障 RGV 动态调度模型的建立与求解

综合以上的目标模型和约束条件，以加工物料的数量最多为目标，建立两道工序发生故障的 *RGV* 动态调度模型，具体模型如下：

目标：$Max = m$

$$
\begin{cases}
S_{(1,1,1)} = 0 \\
S_{(i,1,k)} = S_{(i-1,1,k')} + move_{k'k} & i = 1,2,...,7, k = 1,2,...,8 \\
T_{ijk} = S_{ijk} + tup_k + tdo_j & i = 1,2,...,m, j = 1,2, k = 1,2,...,8 \\
S_{(i,1,k)} = C_{(i',1,k)} & i = 9,10,...,m, k = 1,2,...,8 \\
S_{(i,2,k)} = C_{(i',2,k)} - tclean & i = 9,10,...,m, k = 1,2,...,8 \\
C_{(i,1,k)} = T_{(i,1,k)} + tdown_k + \left(C_{(i',1,k')} + move_{k'k} - T_{(i,1,k)}\right) \cdot g_{(i,1k)} & i = 1,2,...,m, k = 1,2,...,8 \\
C_{(i,2,k)} = T_{(i,2,k)} + tdown_k + tclean + \left(C_{(i',2,k')} + move_{k'k} - T_{(i,2,k)}\right) \cdot g_{(i,2,k)} & i = 1,2,...,m, k = 1,2,...,8 \\
g_{ijk} = \begin{cases} 0, C_{(i',j,k')} + move_{k'k} - T_{ijk} < 0 \\ 1, C_{(i',j,k')} + move_{k'k} - T_{ijk} \geq 0 \end{cases} & i = 1,2,...,m, j = 1,2, k = 1,2,...,8 \\
chose_h = \begin{cases} k_0, \min(T_{ik}) \geq nowtime \wedge T_{ik_0} = \min(T_{ik}) \\ k_1, nowtime \geq T_{ik_1} \wedge chose_{h-1} = cnc_{k_1} \\ k', TT_{k'} = \min(TT_{k'}) \end{cases} & h = 1,2,...,n \\
\sum_{k=1}^{8} C_{(m,2,k)} \cdot f_{(m,2,k)} \leq 28800 \\
\sum_{k=1}^{8} q_{(1,k)} = \alpha \\
\sum_{k=1}^{8} q_{(2,k)} = \beta \\
\alpha + \beta = 8 \\
\sum_{j=1}^{2} q_{jk} = 1 & k = 1,2,...,8 \\
\sum_{k=1}^{8} f_{ijk} = 1 & i = 1,2,...,m, j = 1,2 \\
f_{ijk} \leq q_{jk} & i = 1,2,...,m, j = 1,2, k = 1,2,...,8 \\
S_{i''jk} = tw_{ijk} + tb_{ijk} & i = 1,2,...,m, j = 1,2, k = 1,2,...,8 \\
T_{ijk} f_{ijk} = \begin{cases} (S_{i''jk} - tdown_k - tclean) p_{ijk} & , p_{ijk} = 0 \\ S_{ijk} + tup_{1k} + tdo_j & , p_{ijk} = 1 \end{cases} & i = 1,2,...,m \\
f_{ijk} \in (0,1) & i = 1,2,...,m, j = 1,2, k = 1,2,...,8 \\
g_{ijk} \in (0,1) & i = 1,2,...,m, j = 1,2, k = 1,2,...,8 \\
q_{jk} \in (0,1) & j = 1,2, k = 1,2,...,8
\end{cases}
$$

（4）两道工序故障情况下 RGV 动态调度算法的具体描述

根据以上的两道工序故障情况下 *RGV* 动态调度模型，设计相对应的动态调度算法，具体伪代码如下：

算法 1: 两道工序RGV工作流程(有故障)

Input: $t, updown[2], T, clean, location, Clocation, proportion, cncnow$

Output: ans

```
1  begin
       // 枚举工作第一道工序的CNC编号排列,找到最优解
2      arr = [1,2,3,4,5,6,7,8]
3      while next_permutation(arr, arr+size) do
4          solve(arr of porportion numsize)
5      end
6      return ans
7  end
8  void solve( int v[proportion])
9  {
10     ending , time , flag , RGVhand = 0
       // 初始上料
11     根据v的顺序依次给对应CNC上料
       // 根据智能调度算法进行模拟
12     while time ≤ 28800 do
13         调用RGV智能调度算法(2_1)得出下一个要处理的CNC编号k
           time = time + max(t_{Clocation_k,location}, T_k)
14         location = Clocation_k
15         RGVhand = cncnow_k
16         cncnow_k = flag?RGVhand : k_new
17         time = time + updown_{k&1}
18         if flag is true then
19             time = time + clean ; ending = ending + 1
20         end
21         flag = flag ⊕ 1
22         给当前的CNC随机是否故障和故障时间以及维修时间
23         记录故障各个数据，将当前CNC上物料后续工作时间都设
           为-1
24     end
25     ans = max(ans,ending)
26 }
```

5.2 任务 2 ：检验模型实用性和算法有效性

在本问中，需根据任务中建立 *RGV* 动态调度模型和相对应的求解算法，结合所给智能加工系统作业参数的 3 组数据进行检验，分别考虑一道工序和两道工序的情况，给出 *RGV* 的调度策略和系统的作业效率。

5.2.1 不提前响应模型

在 5.1 中考虑提前响应和智能预判两种情况，设 $chose_h$ 为第 h 个指令 *RGV* 所在位置，T_{ik} 为第 i 个物料在第 k 个 *CNC* 的物料加工完成时间，*nowtime* 为当前时间，cnc_k 为第 k 个 *CNC* 所在位置，TT_k 为 5.1.1.1 智能预判中 *RGV* 选择的最优开始工作位置，以下为提前响应情况下的模型：

$$chose_h=\begin{cases}k_0,\min(T_{ik})\geq nowtime\wedge T_{ik_0}=\min(T_{ik})\\ k_1,nowtime\geq T_{ik_1}\wedge chose_{h-1}=cnc_{k_1}\\ k',TT_{k'}=\min(TT_{k'})\end{cases}$$

若考虑不提前响应，则需将提前响应模型内的提前响应部分删去即可，具体如下所示：

$$chose_h=\begin{cases}k_1,nowtime\geq T_{ik_1}\wedge chose_{h-1}=cnc_{k_1}\\ k',TT_{k'}=\min(TT_{k'})\end{cases}$$

5.2.1 不考虑 CNC 发生故障

不对 *CNC* 在加工过程中可能发生故障的情况进行考虑，即仅需分别对物料加工一道工序和两道工序的过程进行考虑。

（一）一道工序物料加工情况

针对一道工序的物料加工作业情况，每台 *CNC* 安装相同的刀具，物料仅需经过任意一台 *CNC* 加工后，即可变成成料。考虑提前响应情况下，利用所给系统参数的 3 组数据，制定出 *RGV* 提前响应下的调度策略和系统作业效率。

① 第一组

利用智能加工系统作业参数的第一组数据，编程求解出该参数下的 *RGV* 的调度策略部分如下表所示，具体策略详见支撑材料 Case_1_result.xls 中的第一组数据。

表 1：一道工序不发生故障第一组 RGV 的调度策略（时间单位：秒）

加工物料序号	加工 CNC 编号	上料开始时间	下料开始时间	清洗结束时间
1	1	0	588	641
2	2	28	641	697
3	3	79	717	770
4	5	127	846	899
⋮	⋮	⋮	⋮	⋮
379	3	27903	28494	28547
380	4	27956	28547	28603
381	5	28032	28623	28676
382	6	28085	28676	28732

由上表可看出，该参数下，*RGV* 在一个班次内可生产 382 个物料。

对以上表格的数据进行可视化展示，具体如下图所示：

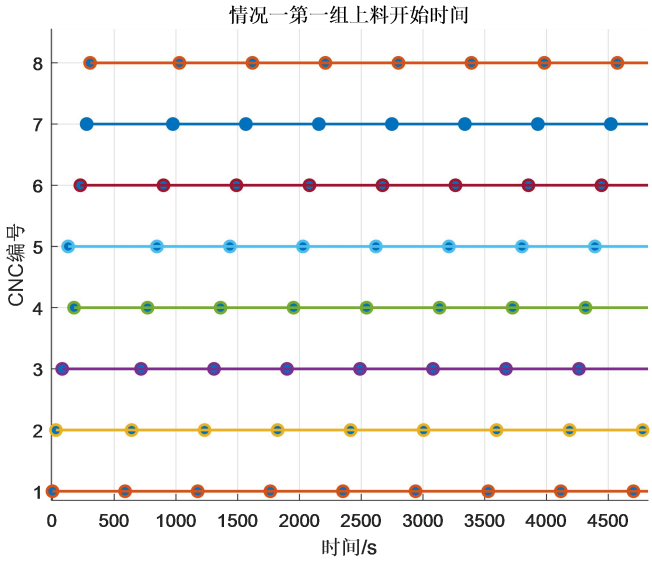

图 5：一道工序不发生故障第一组的上料开始时刻可视化图

上图为第一组参数各 *CNC* 开始上料时刻图，可看出第一轮上料顺序为 *CNC* 1、*CNC* 2、*CNC* 3、*CNC* 4、*CNC* 5、*CNC* 6、*CNC* 7、*CNC* 8，后几轮的上料顺序也与第一轮相同，相应的下料开始时刻如下图所示：

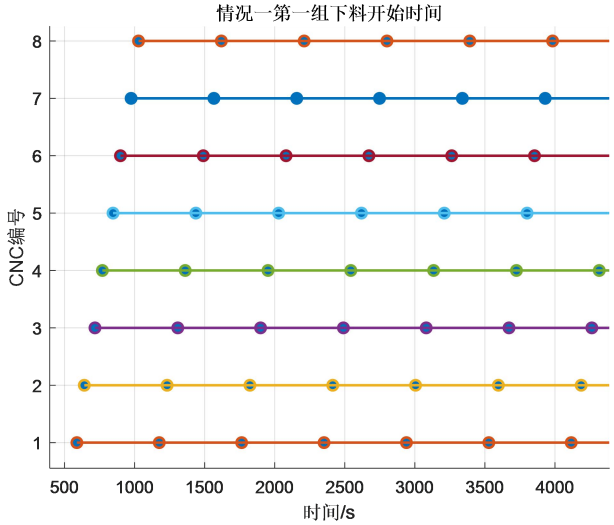

图 6：一道工序不发生故障第一组的下料开始时刻可视化图

从上图可看出 *CNC* 下料开始顺序与上料开始顺序相同。

② 第二组

利用智能加工系统作业参数的第一组数据，编程求解出该参数下的 *RGV* 的调度策略部分如下表所示，具体策略详见支撑材料 Case_1_result.xls 中的第二组数据。

表 2：一道工序不发生故障第二组 RGV 的调度策略（时间单位：秒）

加工物料序号	加工 CNC 编号	上料开始时间	下料开始时间	清洗结束时间
1	1	0	610	670
2	2	30	670	735
3	3	88	758	818
4	5	141	906	966
⋮	⋮	⋮	⋮	⋮
356	4	27822	28450	28515
357	5	27910	28538	28598
358	6	27970	28598	28663
359	7	28058	28686	28746

由上表可看出，该参数下，*RGV* 在一个班次内可生产 359 个物料。

③ 第三组

利用智能加工系统作业参数的第一组数据，编程求解出该参数下的 *RGV* 的调度策略部分如下表所示，具体策略详见支撑材料 Case_1_result.xls 中的第三组数据。

表 3：一道工序不发生故障第三组 RGV 的调度策略（时间单位：秒）

加工物料序号	加工 CNC 编号	上料开始时间	下料开始时间	清洗结束时间
1	1	0	572	624
2	2	27	624	681
3	3	77	699	751
4	5	122	826	878
⋮	⋮	⋮	⋮	⋮
389	5	27945	28522	28574
390	6	27997	28574	28631
391	7	28072	28649	28701
392	8	28124	28701	28758

由上表可看出，该参数下，*RGV* 在一个班次内可生产 392 个物料。

④ 提前响应与不提前响应工作效率对比

为了对比 *RGV* 提前响应比不提前响应的效率变化情况，需对不提前响应和提前响应两种情况下，分别生产的物料数量和相应的系统效率，其中系统效率为实际生产的物料数量比上每台 *CNC* 在一个班次内不停歇生产的物料数量，具体两种情况生产的物料数量和系统效率具体如下表所示：

表 4：不提前响应和提前响应情况下相应的物料数量和系统工作效率

	不提前响应		提前响应		增加效率
	物料数量	系统工作效率	物料数量	系统工作效率	
第一组	370	94.67%	382	97.74%	3.07%
第二组	353	93.84%	359	95.44%	1.60%
第三组	380	94.75%	392	97.74%	2.99%

由上表可看出，提前响应的工作效率比不提前响应的工作效率平均提高了 2.55%。

（二）两道工序物料加工情况

针对两道工序的物料加工作业情况，不同的 *CNC* 安装不同的刀具，物料需按顺序完成第一道工序和第二道工序，才能完成物料的加工。

① 第一组

利用智能加工系统作业参数的第一组数据，调整加工第一道工序与第二道工序的 *CNC* 数量比例，从而生成不同的物料数量，以下是调整的部分比例下生产的成料数量，具体如下表所示：

表 5：工序一工序二 CNC 不同比例成料数量

	工序一 CNC 数量：工序二 CNC 数量		
成料数量	3:5	4:4	5:3
	196	253	205

根据上表可看出工序一 *CNC* 与工序二 *CNC* 的数量比例为 4：4 所生产的成料数量最多，其中第一道工序 CNC 为编号 1、3、5、7，第二道工序 CNC 编号为 2、4、6、8，据此编程求解出该参数下的 *RGV* 的调度策略部分如下表所示，具体策略详见支撑材料 Case_2_result.xls 中的第一组数据。

表 6：两道工序不发生故障第一组 RGV 的调度策略（时间单位：秒）

	工序一			工序二			
	加工 CNC	上料开	下料开	加工 CNC	上料开	下料开	清洗结
1	1	0	428	2	456	884	940
2	3	48	507	4	535	988	1044
3	5	96	586	6	614	1092	1148
4	7	144	665	8	693	1196	1252
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
250	3	27480	27922	4	27950	28392	28448
251	5	27584	28026	6	28054	28496	28552
252	7	27688	28130	8	28158	28600	28656
253	1	27818	28260	2	28288	28730	28786

由上表可看出，该参数下， *RGV* 在一个班次内可生产 253 个物料。

② 第二组

利用智能加工系统作业参数的第一组数据，调整加工第一道工序与第二道工序的*CNC*数量比例，从而生成不同的物料数量，以下是调整的部分比例下生产的成料数量，具体如下表所示：

表 7：工序一工序二 CNC 不同比例成料数量

	工序一 CNC 数量:工序二 CNC 数量		
成料数量	2:6	3:5	4:4
	179	198	211

根据上表可看出工序一*CNC*与工序二*CNC*的数量比例为 4：4 所生产的成料数量最多，其中第一道工序 CNC 为编号 2、4、6、8，第二道工序 CNC 编号为 1、3、5、7，据此编程求解出该参数下的*RGV*的调度策略部分如下表所示，具体策略详见支撑材料 Case_2_result.xls 中的第二组数据。

表：两道工序不发生故障第二组 RGV 的调度策略（时间单位：秒）

加工物料序号	工序一			工序二			
	加工 CNC	上料开	下料开	加工 CNC	上料开	下料开	清洗结
1	2	0	315	1	350	880	940
2	4	58	403	3	438	998	1058
3	6	116	491	5	526	1157	1217
4	8	174	579	7	614	1275	1335
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
208	6	27187	27717	7	27775	28305	28365
209	8	27305	27835	1	27929	28459	28519
210	2	27459	27989	3	28047	28577	28637
211	4	27577	28107	5	28187	28717	28777

由上表可看出，该参数下，*RGV*在一个班次内可生产 211 个物料。

③ 第三组

利用智能加工系统作业参数的第一组数据，调整加工第一道工序与第二道工序的*CNC*数量比例，从而生成不同的物料数量，以下是调整的部分比例下生产的成料数量，具体如下表所示：

表 8：工序一工序二 CNC 不同比例成料数量

	工序一 CNC 数量：工序二 CNC 数量		
成料数量	4:4	5:3	6:2
	231	240	233

根据上表可看出工序一*CNC*与工序二*CNC*的数量比例为 5：3 所生产的成料数量最多，其中第一道工序 CNC 为编号 1、2、4、6、7，第二道工序 CNC 编号为 3、5、8，据此编程求解出该参数下的*RGV*的调度策略部分如下表所

示，具体策略详见支撑材料 Case_2_result.xls 中的第三组数据。

表 9：两道工序不发生故障第三组 RGV 的调度策略（时间单位：秒）

加工物料序号	工序一			工序二			
	加工 CNC 编号	上料开始时间	下料开始时间	加工 CNC 编号	上料开始时间	下料开始时间	清洗结束时间
1	1	0	482	3	527	804	856
2	2	27	572	5	636	906	958
3	4	77	856	5	906	1169	1221
4	6	127	663	8	713	1382	1439
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
237	2	27368	27951	3	28001	28321	28373
238	6	27488	28071	5	28103	28450	28502
239	7	27590	28173	8	28200	28783	28840
240	4	27706	28289	3	28321	28584	28636

由上表可看出，该参数下，*RGV* 在一个班次内可生产 240 个物料。

④ 不提前响应和提前响应的系统工作效率

为了对比 *RGV* 提前响应比不提前响应的效率变化情况，需对不提前响应和提前响应两种情况下，分别生产的物料数量和相应的系统效率，其中系统效率为实际生产的物料数量比上每台 *CNC* 在一个班次内不停歇生产的物料数量，具体两种情况生产的物料数量和系统效率具体如下表所示：

表 10：不提前响应和提前响应情况下相应的物料数量和系统工作效率

	不提前响应		提前响应		增加效率
	物料数量	系统工作效率	物料数量	系统工作效率	
第一组	235	87.31%	253	94.00%	6.69%
第二组	189	86.95%	211	97.07%	10.12%
第三组	240	80.83%	240	80.83%	0.00%

由上表可看出，提前响应的工作效率比不提前响应的工作效率平均提高了 5.60%。

5.2.2 考虑 CNC 发生故障

对 *CNC* 在加工过程中可能发生故障的情况进行考虑，即不仅需对物料经过一道工序和两道工序的加工过程进行考虑，还需考虑 *CNC* 在加工过程中存在 1%的概率会发生故障。

（一）一道工序物料加工情况

针对一道工序的物料加工作业情况，每台 *CNC* 安装相同的刀具，物料仅需经过任意一台 *CNC* 加工后，即可变成成料，并且考虑 *CNC* 可能会发生故障的情况，利用所给系统参数的 3 组数据，制定出 *RGV* 提前响应下 *CNC* 发生

故障的调度策略和系统作业效率。

① 第一组

利用智能加工系统作业参数的第一组数据，编程求解出该参数下的*RGV*的调度策略部分如下表所示，具体策略详见支撑材料Case_3_result_1.xls中的第一组数据。

表11：一道工序发生故障第一组RGV的调度策略（时间单位：秒）

加工物料序号	加工CNC编号	上料开始时间	下料开始时间	清洗结束时间
1	1	0	588	641
2	3	48	661	714
3	6	96	734	790
4	8	147	863	919
⋮	⋮	⋮	⋮	⋮
374	2	27736	28327	28383
375	1	27792	28383	28436
376	6	27878	28469	28525
377	3	27954	28545	28598

由上表可看出，该参数下，*RGV*在一个班次内可生产375个物料，其中发生故障2次，具体如下图所示：

表12：一道工序发生故障的CNC情况表（时间单位：秒）

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	故障时间
307	6	22665	23586	921
355	7	26350	27463	1113

上表即为发生故障*CNC*的编号和故障时间。

② 第二组

利用智能加工系统作业参数的第一组数据，编程求解出该参数下的*RGV*的调度策略部分如下表所示，具体策略详见支撑材料Case_3_result_1.xls中的第二组数据。

表13：一道工序发生故障第二组RGV的调度策略（时间单位：秒）

加工物料序号	加工CNC编号	上料开始时间	下料开始时间	清洗结束时间
1	1	0	610	670
2	2	30	670	735
3	4	88	758	823
4	5	146	906	966
⋮	⋮	⋮	⋮	⋮
348	6	27588	28262	28327
349	7	27676	28350	28410
350	8	27736	28410	28475
351	2	27860	28534	28599

由上表可看出，该参数下，*RGV* 在一个班次内可生产 349 个物料，其中发生故障 2 次，具体如下图所示：

表 14：一道工序发生故障的 CNC 情况表

故障时的物料序号	故障 CNC 编号	故障开始时间（单位：秒）	故障结束时间（单位：秒）	故障时间（单位：秒）
154	2	12298	12981	683
298	4	23736	24735	999

上表即为发生故障 *CNC* 的编号和故障时间。

③ 第三组

利用智能加工系统作业参数的第一组数据，编程求解出该参数下的 *RGV* 的调度策略部分如下表所示，具体策略详见支撑材料 Case_3_result_1.xls 中的第三组数据。

表 15：一道工序发生故障第三组 RGV 的调度策略（时间单位：秒）

加工物料序号	加工 CNC 编号	上料开始时间	下料开始时间	清洗结束时间
1	1	0	572	624
2	3	45	642	694
3	6	90	712	769
4	7	140	787	839
⋮	⋮	⋮	⋮	⋮
359	3	27460	28088	28140
360	8	27544	28172	28229
361	4	27633	28261	28318
362	7	27722	28350	28402

由上表可看出，该参数下，*RGV* 在一个班次内可生产 357 个物料，其中发生故障 5 次，具体如下图所示：

表 16：一道工序发生故障的 CNC 情况表

故障时的物料序号	故障 CNC 编号	故障开始时间（单位：秒）	故障结束时间（单位：秒）	故障时间（单位：秒）
59	6	4369	5053	684
125	5	9411	10433	1022
146	7	10968	12007	1039
172	6	12958	13560	602
312	3	23872	24865	993

上表即为发生故障 *CNC* 的编号和故障时间。

④ 系统平均工作效率

为了检验模型的实用性和算法的有效性，需计算使用该模型和算法给出的 *RGV* 调度策略相应的系统作业效率，由于 *CNC* 出现故障的不确定性，故

多次实验得出多组数据，求其平均值，计算其工作效率，其中系统效率为实际生产的物料数量比上每台*CNC*在一个班次内不停歇生产的物料数量，具体生产的物料数量和系统效率具体如下表所示：

表 17：不提前响应和提前响应情况下相应的物料数量和系统工作效率

		不提前响应			提前响应		
		物料数	故障	系统工	物料数	故障	系统工
第一组	数据 1	366	4	93.64%	374	3	95.69%
	数据 2	360	6	92.11%	373	5	95.43%
	数据 3	380	1	97.23%	370	5	94.67%
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	数据 99	366	4	93.64%	369	3	94.41%
	数据 100	345	4	88.27%	364	8	93.13%
	平均值	366.65	3.95	93.81%	368.04	4.16	94.17%
第二组	数据 1	330	2	87.73%	340	6	90.38%
	数据 2	347	1	92.25%	357	5	94.90%
	数据 3	318	2	84.54%	357	5	94.90%
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	数据 99	350	2	93.04%	352	2	93.57%
	数据 100	327	7	86.93%	303	4	80.55%
	平均值	338.98	3.51	90.11%	338.98	3.46	90.24%
第三组	数据 1	382	3	95.25%	388	3	96.75%
	数据 2	388	2	96.75%	381	2	95.00%
	数据 3	383	2	95.50%	363	3	90.51%
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	数据 99	373	3	93.01%	366	4	91.26%
	数据 100	371	7	92.51%	385	4	96.00%
	平均值	377.87	3.7	94.22%	377.19	3.88	94.05%

由上表可看出，提前响应的工作效率比不提前响应的工作效率高一些，但因为*CNC*故障的不可控，其系统工作效率也会受到一定影响。

（二）两道工序物料加工情况

针对两道工序的物料加工作业情况，不同的*CNC*安装不同的刀具，物料需按顺序完成第一道工序和第二道工序，才能完成物料的加工，并且考虑*CNC*可能会发生故障的情况，利用所给系统参数的 3 组数据，制定出*RGV*提前响应下*CNC*发生故障的调度策略和系统作业效率。

针对两道工序的物料加工作业情况，不同的*CNC*安装不同的刀具，物料需按顺序完成第一道工序和第二道工序，才能完成物料的加工。

① 第一组

根据 5.2.1 中工序一 *CNC* 与工序二 *CNC* 的数量最优比例为 4：4，利用智能加工系统作业参数的第一组数据，据此编程求解出该参数下的 *RGV* 的调度策略部分如下表所示，具体策略详见支撑材料 Case_3_result_2.xls 中的第一组数据。

表 18：两道工序发生故障第一组 RGV 的调度策略（时间单位：秒）

加工物料序号	工序一			工序二			
	加工 CNC	上料开	下料开	加工 CNC	上料开	下料开	清洗结
1	2	0	431	1	462	893	946
2	4	51	510	3	541	997	1050
3	5	102	589	6	617	1098	1154
4	7	150	668	8	696	1202	1258
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
243	7	27201	27643	8	27671	28113	28169
244	5	27305	27747	6	27775	28217	28273
245	4	27409	27851	3	27882	28324	28377
246	2	27513	27955	1	27986	-1	-1

由上表可看出，该参数下，*RGV* 在一个班次内可生产 243 个物料，发生故障 3 次，其中下料时间和清洗时间为-1 表示该 *CNC* 物料加工过程中出现了故障，此物料损坏，具体的故障如下表所示：

表 19：两道工序第一组发生故障的 CNC 情况表

故障时的物	故障 CNC	故障开始时间	故障结束时间	故障时间
93	2	10520	11617	1097
206	7	23057	24029	972
246	1	28186	29294	1108

上表即为发生故障 *CNC* 的编号和故障时间。

② 第二组

据 5.2.1 中工序一 *CNC* 与工序二 *CNC* 的数量最优比例为 4：4，利用智能加工系统作业参数的第一组数据，据此编程求解出该参数下的 *RGV* 的调度策略部分如下表所示，具体策略详见支撑材料 Case_3_result_2.xls 中的第二组数据。

表 20：两道工序发生故障第二组 RGV 的调度策略（时间单位：秒）

加工物料序号	工序一			工序二			
	加工 CNC 编号	上料开始时间	下料开始时间	加工 CNC 编号	上料开始时间	下料开始时间	清洗结束时间
1	1	0	310	2	340	875	940
2	4	53	398	3	433	998	1058
3	5	111	486	6	516	1152	1217
4	8	164	574	7	609	1270	1330
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
202	4	26858	27393	6	27451	27986	28051
203	5	26981	27516	7	27569	28104	28164
204	8	27094	27629	2	27750	28285	28350
205	1	27280	27815	3	27868	28403	28463

由上表可看出，该参数下，*RGV* 在一个班次内可生产 202 个物料，其中发生故障 3 次，具体的故障如下表所示：

表 21：两道工序第二组发生故障的 CNC 情况表

故障时的物料序号	故障 CNC 编号	故障开始时间（单位：秒）	故障结束时间（单位：秒）	故障时间（单位：秒）
93	2	12791	13652	861
149	1	19981	21119	1138

上表即为发生故障 *CNC* 的编号和故障时间。

③ 第三组

据 5.2.1 中工序一 *CNC* 与工序二 *CNC* 的数量最优比例为 5：3，利用智能加工系统作业参数的第一组数据，据此编程求解出该参数下的 *RGV* 的调度策略部分如下表所示，具体策略详见支撑材料 Case_3_result_2.xls 中的第三组数据。

表 22：两道工序发生故障第三组 RGV 的调度策略（时间单位：秒）

加工物料序号	工序一			工序二			
	加工 CNC 编号	上料开始时间	下料开始时间	加工 CNC 编号	上料开始时间	下料开始时间	清洗结束时间
1	1	0	482	2	509	791	848
2	4	45	559	3	591	925	977
3	6	95	636	7	686	1161	1213
4	8	145	713	2	791	1022	1079
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
232	8	26969	27556	3	27620	27976	28028
233	1	27103	27690	2	27717	28304	28361
234	4	27205	27792	7	27856	28091	28143
235	6	27339	27926	3	27976	28207	28259

由上表可看出，该参数下，*RGV* 在一个班次内可生产 232 个物料，发生故障 3 次，具体的故障如下表所示：

表 23：两道工序第一组发生故障的 CNC 情况表（时间单位：秒）

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	故障时间
49	6	5742	6519	777
133	7	16155	16980	825
164	1	19333	19958	625

上表即为发生故障 *CNC* 的编号和故障时间。

④ 系统平均工作效率

为了检验模型的实用性和算法的有效性，需计算使用该模型和算法给出的 *RGV* 调度策略相应的系统作业效率，由于 *CNC* 出现故障的不确定性，故多次实验得出多组数据，求其平均值，计算其工作效率，其中系统效率为实际生产的物料数量比上每台 *CNC* 在一个班次内不停歇生产的物料数量，具体生产的物料数量和系统效率具体如下表所示：

表 24：不提前响应和提前响应情况下相应的物料数量和系统工作效率

		不提前响应			提前响应		
		物料数量	故障次数	系统工作效率	物料数量	故障次数	系统工作效率
第一组	数据 1	227	3	84.34%	239	5	88.80%
	数据 2	222	7	82.48%	234	2	86.94%
	数据 3	219	7	81.36%	245	2	91.02%
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	数据 99	234	2	86.94%	220	8	81.74%
	数据 100	242	2	89.91%	250	1	92.88%
	平均值	226.87	4.73	84.29%	232.55	4.94	86.40%
第二组	数据 1	187	2	86.03%	201	4	92.47%
	数据 2	181	8	83.27%	200	4	92.01%
	数据 3	189	0	86.95%	201	3	92.47%
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	数据 99	185	6	85.11%	198	5	91.09%
	数据 100	186	5	85.57%	200	3	92.01%
	平均值	185.25	4.08	85.23%	200.55	4.16	92.27%
第三组	数据 1	234	2	78.81%	231	7	77.80%
	数据 2	226	4	76.12%	231	7	77.80%
	数据 3	236	5	79.48%	235	4	79.15%
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	数据 99	230	7	77.46%	232	6	78.14%
	数据 100	232	5	78.14%	237	3	79.82%
	平均值	232.09	4.69	78.17%	233.72	5	78.72%

由上表可看出，提前响应的工作效率比不提前响应的工作效率高一些。

六、模型评价与推广

6.1 模型优点

1）本模型给出了 RGV 动态调度模型和相应的求解算法

2）本模型检验模型的实用性和算法的有效性

6.2 模型推广

本文基本给出了 RGV 动态调度模型和相应的求解算法，通过　算法对空闲信号进行排序，给出相应的动态调度模型，并且检验模型的实用性和算法的有效性，最终给出了不同情况下 RGV 的调度策略和系统的作业效率。根据数据测试结果，本文提出的算法工作效率均值在 85%以上，具有较高效率。基本解决了 RGV 的动态调度问题，因此可以初步运用在小规模的流水线生产上。运用本模型可最大程度地提高小工厂或流水线的生产效率。由此可见，本模型具有很大的推广价值。

七、参考文献

[1]王佳,巫江,方异锋,等.铸造车间 RGV 柔性动态调度系统的设计与应用[J].机电工程技术,2023,52(09):62-65+104.

[2]李艳春.智能 RGV 的动态调度策略模型的建立与检验[J].通化师范学院学报,2023,44(06):49-55.DOI:10.13877/j.cnki.cn22-1284.2023.06.008.

[3]李艳春.智能 RGV 的动态调度策略模型的建立与检验[J].通化师范学院学报,2023,44(06):49-55.DOI:10.13877/j.cnki.cn22-1284.2023.06.008.

附录

一、支撑材料的文件列表

1、Case_1_result.xls

2、Case_2_result.xls

3、Case_3_result_1.xls

4、Case_3_result_2.xls

二、程序代码

```
#include <iostream>
    #include <map>
    #include <vector>
    #include <queue>
    #include <set>
    #include <tuple>
    #include <cstring>
    #include <cmath>
    #include <limits>
    #include <ostream>
    #include <fstream>
    #include <algorithm>
    #include <random>
    #include <cstdlib>
    #include <ctime>

    #define GROUP 3 //  组数
    #define early 0 //  提前响应
    #define STRONG_JUDE 0 //强判
    #define OPEN_HAND_PROPORTION
    #undef OPEN_HAND_PROPORTION
    #define PROPORTION 3 //  工序一个数
    using namespace    std;

    const int N = 500;
```

```
//constant
int move_t1;//RGV 移动 1 个单位所需时间
int move_t2;//RGV 移动 2 个单位所需时间
int move_t3;//RGV 移动 3 个单位所需时间
int move_time[4] ;//
int complete;//CNC 加工完成一个一道工序的物料所需时间
int complete2_1;//CNC 加工完成一个两道工序物料的第一道工序所需时间
int complete2_2;//CNC 加工完成一个两道工序物料的第二道工序所需时间
int updown_even;//RGV 为 CNC1#，3#，5#，7#一次上下料所需时间
int updown_odd;//RGV 为 CNC2#，4#，6#，8#一次上下料所需时间
int updown[2];
int clean;//RGV 完成一个物料的清洗作业所需时间
//var
int rgv_pos = 1;
int cnc_pos[9] = {0,1,1,2,2,3,3,4,4};
int cnc_now[9];
int cnc_endtime[9];
int cnt;
int cnt_end;
queue<int>instruct;
vector<int>start;
struct Result{
    int i;
    int cnc;
    int start;
    int end;
    int last_end;
};
Result result[N];
int ans;
int a[9];
int ansa[9];
bool vis[9];
```

```
int main();
void init();
void solve();
void outfile();
void dfs(int ); //  跑出最优的上料组合
void reset();

struct Result2{
	int i;
	int first_cnc;
	int second_cnc;
	int first_start;
	int second_start;
	int first_end;
	int second_end;
	int last_end;
};
Result2 result2[N];
int a2[9];//第一道工序的 CNC
int a2_size; //第一工序的 CNC 数量
bool rgv_hand;//判断 rgv 手上有没有第一道工序的成品
bool cnc_pp[9];//first false , second true
int rgv_cnt;//rgv  手上的工件编号是什么
void solve2();
void outfile2();
void dfs2(int);
void hand_init();
void outfile3_3();

struct Fault{
	int i;
	int cnc;
	int cnc_now;
```

```
    int start_time;
    int end_time;
};
Fault fault[N];
int fault_size;
int cnc_fault_endtime[9];
void dfs3_1(int x);
void dfs3_2(int x);
void solve3_1();
void solve3_2();
void outfile3_1();
inline pair<bool,int> lucky_Roulette();

void menu(){
    cout << "situation one    GROUP one: 11";
}
void init(int );

int main() {
    std::ofstream file("jun.xls"); //  创建 ofstream 对象并打开文件
    if (!file.is_open()) {
        std::cerr << "无法打开文件" << std::endl;
    }
    file    << "物料数量\t 系统工作效率\t 故障数量\n";
    int _ = 0;
    init(1);
    dfs3_2(1);
    reset();
    for(int i=1;i<=a2_size;i++){
        a2[i] = ansa[i];
        cnc_pp[a2[i]] = false;
    }
    while(++_ <= 100){
        reset();
```

```
            solve3_2();
            file    << cnt_end << '\t' <<    ((double)cnt_end/269.158878504673) <<
'\t' << fault_size<<'\n';
        }
        file.close(); //  关闭文件
        return 0;



        cout << "group:";
        int group;
        cin >> group;
        init(group);
        switch(group){
            case 1:break;
            case 2:break;
            case 3:break;
            default:return 0;
        }

        cout << "situation: (1,2,31,32)";
        int situation;
        cin >> situation;
        switch (situation) {
            case 1:
                dfs(1);
                reset();
                for(int i=1;i<=8;i++){
                    a[i] = ansa[i];
                }
                solve();
                outfile();
                cout << "\ncomplete mun:" << cnt_end;
                break;
```

```
case 2:
        cout << "PROPORTION NUM:";
        cin >> a2_size;
        dfs2(1);
        reset();
        for(int i=1;i<=a2_size;i++){
            a2[i] = ansa[i];
            cnc_pp[a2[i]] = false;
        }
        solve2();
        outfile2();
        cout << "\ncomplete mun:" << cnt_end;
        break;
case 31:
        dfs3_1(1);
        reset();
        for(int i=1;i<=8;i++){
            a[i] = ansa[i];
        }
        solve3_1();
        outfile();
        outfile3_1();
        cout << "\ncomplete mun:" << cnt_end << '\n';
        cout << "fault_num:" << fault_size;

        break;
case 32:
        cout << "PROPORTION NUM:";
        cin >> a2_size;
        dfs3_2(1);
        reset();
        for(int i=1;i<=a2_size;i++){
            a2[i] = ansa[i];
            cnc_pp[a2[i]] = false;
```

```
            }
            solve3_2();
            outfile2();
            outfile3_1();
            cout << "\ncomplete mun:" << cnt_end << '\n';
            cout << "fault_num:" << fault_size;
            break;
        default:return 0;
}
//getchar();getchar();getchar();
return 0;

init();
//hand_init();
dfs3_2(1);
reset();
for(int i=1;i<=a2_size;i++){
        a2[i] = ansa[i];
        cnc_pp[a2[i]] = false;
}
//hand_init();
solve3_2();
outfile2();
outfile3_1();
cout << cnt_end << '\n';
cout << fault_size << '\n';
return 0;
init();
dfs3_1(1);
reset();
for(int i=1;i<=8;i++){
        a[i] = ansa[i];
}
solve3_1();
```

```
	outfile();
	outfile3_1();
	cout << fault_size;

	return 0;
	init();
	//hand_init();
	dfs2(1);
	reset();
	for(int i=1;i<=a2_size;i++){
		a2[i] = ansa[i];
		cnc_pp[a2[i]] = false;
	}
	//hand_init();
	solve2();
	outfile2();
	cout << cnt_end;
	return 0;
	init();
	dfs(1);
	reset();
	for(int i=1;i<=8;i++){
		a[i] = ansa[i];
	}
	solve();
	outfile();
	cout << ans;
	return 0;
}
void hand_init(){
	cnc_pp[2] = cnc_pp[4] = cnc_pp[6] = cnc_pp[3] = cnc_pp[8] = true;
	a2_size = 3;
	a2[1] = 1;
	a2[2] = 7;
```

```
        a2[3] = 5;

}
void reset(){
        rgv_pos = 1;//rvg 归位
        cnt = cnt_end = 0;
        for(int i=1;i<=8;i++){
                cnc_endtime[i] = 0;
                cnc_now[i] = 0;
                cnc_fault_endtime[i] = 0;
                //cnc_pp[i] = true;
        }
        rgv_hand = false;
        rgv_cnt = 0;
        fault_size = 0;
}
void dfs(int x){
        if(x == 9){
                reset();
                solve();
                if(cnt_end > ans){
                        ans = cnt_end;
                        for(int i=1;i<=8;i++)
                                ansa[i] = a[i];
                }
                return ;
        }
        if(x == 1){
                for(int i=1;i<=2;i++){
                        if(!vis[i]){
                                vis[i] = true;
                                a[x] = i ;
                                dfs(x+1);
                                vis[i] = false;
```

```
                }
            }
        }
        else if(x == 2){
            for(int i=1;i<=4;i++){
                if(!vis[i]){
                    vis[i] = true;
                    a[x] = i ;
                    dfs(x+1);
                    vis[i] = false;
                }
            }
        }
        else{
            for(int i=1;i<=8;i++){
                if(!vis[i] ){
                    if(x<=4 && cnc_pos[a[x-1]] >= cnc_pos[i])
                        continue;
                    if(abs(cnc_pos[a[x-1]] - cnc_pos[i]) >= 2) //  认为规定
                        continue;
                    vis[i] = true;
                    a[x] = i ;
                    dfs(x+1);
                    vis[i] = false;
                }
            }
        }
}
void dfs2(int x){
    if(x == a2_size+1){
        reset();
        solve2();
        if(cnt_end > ans){
            ans = cnt_end;
```

```
                for(int i=1;i<=a2_size;i++){
                        ansa[i] = a2[i];
                }
        }
        return ;
}
if(x == 1){
        for(int i=1;i<=2;i++){
                if(!vis[i]){
                        vis[i] = true;
                        a2[x] = i ;
                        cnc_pp[a2[x]] = false;
                        dfs2(x+1);
                        vis[i] = false;
                        cnc_pp[a2[x]] = true;
                }
        }
}
else if(x == 2){
        for(int i=1;i<=4;i++){
                if(!vis[i]){
                        vis[i] = true;
                        a2[x] = i ;
                        cnc_pp[a2[x]] = false;
                        dfs2(x+1);
                        vis[i] = false;
                        cnc_pp[a2[x]] = true;
                }
        }
}
else{
        for(int i=1;i<=8;i++){
                if(!vis[i]){
                        if(x<=4 && cnc_pos[a2[x-1]] >= cnc_pos[i])
```

```
                continue;
            if(abs(cnc_pos[a2[x-1]] - cnc_pos[i]) >= 2) //  人为规定
                continue;
            vis[i] = true;
            a2[x] = i ;
            cnc_pp[a2[x]] = false;
            dfs2(x+1);
            vis[i] = false;
            cnc_pp[a2[x]] = true;
        }
    }
}
void dfs3_1(int x){
    if(x == 9){
        reset();
        solve3_1();
        if(cnt_end > ans){
            ans = cnt_end;
            for(int i=1;i<=8;i++)
                ansa[i] = a[i];
        }
        return ;
    }
    if(x == 1){
        for(int i=1;i<=2;i++){
            if(!vis[i]){
                vis[i] = true;
                a[x] = i ;
                dfs3_1(x+1);
                vis[i] = false;
            }
        }
    }
```

```
    else if(x == 2){
        for(int i=1;i<=4;i++){
            if(!vis[i]){
                vis[i] = true;
                a[x] = i ;
                dfs3_1(x+1);
                vis[i] = false;
            }
        }
    }
    else{
        for(int i=1;i<=8;i++){
            if(!vis[i] ){
                if(x<=4 && cnc_pos[a[x-1]] >= cnc_pos[i])
                    continue;
                if(abs(cnc_pos[a[x-1]] - cnc_pos[i]) >= 2) //  认为规定
                    continue;
                vis[i] = true;
                a[x] = i ;
                dfs3_1(x+1);
                vis[i] = false;
            }
        }
    }
}
void dfs3_2(int x){
    if(x == a2_size+1){
        reset();
        solve3_2();
        if(cnt_end > ans){
            ans = cnt_end;
            for(int i=1;i<=a2_size;i++){
                ansa[i] = a2[i];
            }
```

```
        }
        return ;
    }
    if(x == 1){
        for(int i=1;i<=2;i++){
            if(!vis[i]){
                vis[i] = true;
                a2[x] = i ;
                cnc_pp[a2[x]] = false;
                dfs3_2(x+1);
                vis[i] = false;
                cnc_pp[a2[x]] = true;
            }
        }
    }
    else if(x == 2){
        for(int i=1;i<=4;i++){
            if(!vis[i]){
                vis[i] = true;
                a2[x] = i ;
                cnc_pp[a2[x]] = false;
                dfs3_2(x+1);
                vis[i] = false;
                cnc_pp[a2[x]] = true;
            }
        }
    }
    else{
        for(int i=1;i<=8;i++){
            if(!vis[i]){
                if(x<=4 && cnc_pos[a2[x-1]] >= cnc_pos[i])
                    continue;
                if(abs(cnc_pos[a2[x-1]] - cnc_pos[i]) >= 2) //  人为规定
                    continue;
```

```
                    vis[i] = true;
                    a2[x] = i ;
                    cnc_pp[a2[x]] = false;
                    dfs3_2(x+1);
                    vis[i] = false;
                    cnc_pp[a2[x]] = true;
                }
            }
        }
}
int jude(vector<int>&v,int time){
    //距离优先
    for (auto vi : v){
        if(cnc_pos[vi] == rgv_pos)
            return vi;
    }
    cout << '/';
    //预判
    bool pos[4] = {false};
    for (auto vi : v){
        pos[abs(cnc_pos[vi] - rgv_pos)]|=1;
    }
    for(int i=1;i<=3;i++)cout << pos[i] << ' ';

    if(pos[1]){
        for (auto vi : v)
            if(abs(cnc_pos[vi] - rgv_pos) == 1)
                return vi;
    }
    else if(pos[2]){

        int index2 ;
        for(auto vi:v)
            if(2==abs(cnc_pos[vi] - rgv_pos))
```

```
                index2 = vi;


        int time1 = time + move_t2;
        time1 += updown_even; // if all is even

        int index = 0;
        int timet=move_time[2]+time;
        while(cnc_fault_endtime[index] > timet)index++;
        for(int i=1;i<=8;i++){
            if(cnc_endtime[i] <= timet    && cnc_fault_endtime[i] <= time){
                index = i*(1==abs(cnc_pos[i] - rgv_pos));
            }
        }

        if(index){
            int time2 =    cnc_endtime[index] - time;
            time2 += move_t1;
            time2 += updown_even;

            time2 += move_t1;
            time2 += updown_even;

            time1 += move_time[abs(cnc_pos[index2]-cnc_pos[index])];
            time1 += updown_even;

            if(time1 < time2)//move2
                return index2;
            else
                return index;
        }
        else
            return index2;
    }
```

```
    else{ // pos[3] no

        int index3;
        for(auto vi:v)
            if(3==abs(cnc_pos[vi] - rgv_pos))
                index3 = vi;

        int time1 = time + move_t3;
        time1 += updown_even; // if all is even

        int index = 0;
        int timet=move_time[3];
        while(cnc_fault_endtime[index] > timet)index++;
        for(int i=1;i<=8;i++){
            if(cnc_endtime[i] <= timet  && cnc_fault_endtime[i] <= timet
&& (!index|| abs(cnc_pos[i] - rgv_pos) < abs(cnc_pos[index] - rgv_pos))){
                index = i;
            }
        }

        if(index){
            int time2 =  cnc_endtime[index] - time;
            time2 += move_time[abs(cnc_pos[index] - rgv_pos)];
            time2 += updown_even;



            time2 += move_time[abs(cnc_pos[index]-cnc_pos[index3])];
            time2 += updown_even;

            time1 += move_time[abs(cnc_pos[index]-cnc_pos[index3])];
            time1 += updown_even;

            if(time1 < time2){//move3
```

```
                return index3;
            }
            else
                return index;
        }
        else
            return index3;

    }

}
int jude2_find_second(vector<int>&v,int time){
    //距离优先
    for (auto vi : v){
        if(cnc_pos[vi] == rgv_pos)
            return vi;
    }
    cout << '/';
    //预判
    bool pos[4] = {false};
    for (auto vi : v){
        pos[abs(cnc_pos[vi] - rgv_pos)]|=1;
    }
    for(int i=1;i<=3;i++)cout << pos[i] << ' ';

    if(pos[1]){
        for (auto vi : v)
            if(abs(cnc_pos[vi] - rgv_pos) == 1)
                return vi;
    }
    else if(pos[2]){

        int index2 ;
        for(auto vi:v)
```

```
            if(2==abs(cnc_pos[vi] - rgv_pos))
                index2 = vi;


        int time1 = time + move_t2;
        time1 += updown[index2&1];

        int index = 0;
        int timet=move_time[2]+time;
        vector<int>text = {2,4,6,8,1,3,5,7};
        for(auto i : text){
            if(cnc_pp[i] && cnc_endtime[i] <= timet &&
cnc_fault_endtime[i] <= timet){
                index = i*(1==abs(cnc_pos[i] - rgv_pos));
            }
        }
        text.clear();

        if(index){
            int time2 =    cnc_endtime[index];
            time2 += move_t1;
            time2 += updown[index&1];

            time2 += move_time[abs(cnc_pos[index]-cnc_pos[index2])];
            time2 += updown[index2&1];

            time1 += move_time[abs(cnc_pos[index2]-cnc_pos[index])];
            time1 += updown[index&1];

            if(time1 < time2)//move2
                return index2;
            else
                return index;
        }
```

```
        else
            return index2;
    }
    else{ // pos[3] no

        int index3;
        for(auto vi:v)
            if(3==abs(cnc_pos[vi] - rgv_pos))
                index3 = vi;

        int time1 = time + move_t3;
        time1 += updown[index3&1];

        int index = 0;
        int timet=move_time[3]+time;
        vector<int>text = {2,4,6,8,1,3,5,7};
        for(auto i: text){
            if(cnc_pp[i] && cnc_endtime[i] <= timet && (!index|| abs(cnc_pos[i] - rgv_pos) < abs(cnc_pos[index] - rgv_pos) && cnc_fault_endtime[i] <= timet)){
                index = i;
            }
        }
        text.clear();

        if(index){
            int time2 =    cnc_endtime[index];
            time2 += move_time[abs(cnc_pos[index] - rgv_pos)];
            time2 += updown[index&1];

            time2 += move_time[abs(cnc_pos[index]-cnc_pos[index3])];
            time2 += updown[index3&1];

            time1 += move_time[abs(cnc_pos[index]-cnc_pos[index3])];
```

```
                time1 += updown[index&1];

                if(time1 < time2){//move3
                    return index3;
                }
                else
                    return index;
            }
            else
                return index3;

        }

}
int jude2_find_first(vector<int>&v,int time){
        //距离优先
        for (auto vi : v){
            if(cnc_pos[vi] == rgv_pos)
                return vi;
        }
        cout << '/';
        //预判
        bool pos[4] = {false};
        for (auto vi : v){
            pos[abs(cnc_pos[vi] - rgv_pos)]|=1;
        }
        for(int i=1;i<=3;i++)cout << pos[i] << ' ';

        if(pos[1]){
            for (auto vi : v)
                if(abs(cnc_pos[vi] - rgv_pos) == 1)
                    return vi;
        }
        else if(pos[2]){
```

```
            int index2 ;
            for(auto vi:v)
                if(2==abs(cnc_pos[vi] - rgv_pos))
                    index2 = vi;


            int time1 = time + move_t2;
            time1 += updown[index2&1];

            int index = 0;
            int timet=move_time[2]+time;
            vector<int>text = {2,4,6,8,1,3,5,7};
            for(auto i : text){
                if(!cnc_pp[i] && cnc_endtime[i] <= timet &&
cnc_fault_endtime[i] <= timet){
                    index = i*(1==abs(cnc_pos[i] - rgv_pos));
                }
            }
            text.clear();

            if(index){
                int time2 =    cnc_endtime[index] - time;
                time2 += move_t1;
                time2 += updown[index&1];

                time2 += move_time[abs(cnc_pos[index]-cnc_pos[index2])];
                time2 += updown[index2&1];

                time1 += move_time[abs(cnc_pos[index2]-cnc_pos[index])];
                time1 += updown[index&1];

                if(time1 < time2)//move2
                    return index2;
```

```
                else
                    return index;
            }
            else
                return index2;
        }
        else{ // pos[3] no

            int index3;
            for(auto vi:v)
                if(3==abs(cnc_pos[vi] - rgv_pos))
                    index3 = vi;

            int time1 = time + move_t3;
            time1 += updown[index3&1];

            int index = 0;
            int timet=move_time[3]+time;
            vector<int>text = {2,4,6,8,1,3,5,7};
            for(auto i:text){
                if(!cnc_pp[i] && cnc_endtime[i] <= timet && (!index|| abs(cnc_pos[i] - rgv_pos) < abs(cnc_pos[index] - rgv_pos)) && cnc_fault_endtime[i] <= timet){
                    index = i;
                }
            }
            text.clear();

            if(index){
#if STRONG_JUDE
                if(abs(cnc_pos[index]-rgv_pos)==2){
                    int timett = max(timet + move_time[abs(cnc_pos[index]-cnc_pos[index3])] ,cnc_endtime[index] + move_time[abs(cnc_pos[index]-rgv_pos)] );
```

```
                int index1 = 0;
                for(int i=1;i<=8;i++){
                    if(!cnc_pp[i] && cnc_endtime[i] <= timet && (!index1|| abs(cnc_pos[i] - rgv_pos) < abs(cnc_pos[index1] - rgv_pos)) && cnc_fault_endtime[i] <= timet){
                        index1 = i;
                    }
                }
                if(abs(cnc_pos[index1]-rgv_pos)==1){//找到一个 1 单位的
                    int time4 = cnc_endtime[index1];

                }
            }
#endif
            int time2 =    cnc_endtime[index] - time + time;
            time2 += move_time[abs(cnc_pos[index] - rgv_pos)];
            time2 += updown[index&1];



            time2 += move_time[abs(cnc_pos[index]-cnc_pos[index3])];
            time2 += updown[index3&1];

            time1 += move_time[abs(cnc_pos[index]-cnc_pos[index3])];
            time1 += updown[index&1];

            if(time1 < time2){//move3
                return index3;
            }
            else
                return index;
        }
        else
            return index3;
```

```
    }

}
void solve2(){
    int time = 0;
    //上 1 料
    for(int i=1;i<=a2_size;i++){
        time += move_time[abs(rgv_pos - cnc_pos[a2[i]])];//移动
        rgv_pos = cnc_pos[a2[i]];

        cnc_now[a2[i]] = ++cnt;
        result2[cnc_now[a2[i]]].i = cnt;
        result2[cnc_now[a2[i]]].first_cnc = a2[i];
        result2[cnc_now[a2[i]]].first_start = time;
        time += updown[a2[i]&1];
        cnc_endtime[a2[i]] = time + complete2_1;
    }
    while(time < 28800){
        if(rgv_hand){//有第一道工序的成品，需要去第二道工序

            bool flag = false;
            for(int i=1;i<=8;i++){
                if(cnc_pp[i] && cnc_endtime[i] <= time){
                    flag |= 1;
                    break;
                }
            }
            if(!flag){
                //提前移动
                int index = 1;
                while(!cnc_pp[index])index++; //找到第一个是第二工序的
cnc
                for(int i=2;i<=8;i++){
```

```
                    if(cnc_pp[i] && cnc_endtime[index] > cnc_endtime[i]){
                        index = i;
                    }
                    else if(cnc_pp[i] && cnc_endtime[index] == cnc_endtime[i]){
                        if(abs(cnc_pos[index] - rgv_pos) > abs(cnc_pos[i] - rgv_pos))
                            index = i;
                    }
                }


#if early
                time += max(move_time[abs(cnc_pos[index] - rgv_pos)] , cnc_endtime[index] - time);//pass time
                rgv_pos = cnc_pos[index];
#else
                time +=  cnc_endtime[index] - time;//pass time
#endif
                continue;
            }

            vector<int>v;
            for(int i=1;i<=8;i++){
                if(cnc_pp[i] && cnc_endtime[i] <= time){
                    v.push_back(i);
                    cout << i << ' ';
                }
            }
            //jude
            cout << '*';
            int next = jude2_find_second(v,time);//rgv 下一个要到的地点为 next 个 cnc
```

```
                cout << next << '\n';

                time+=move_time[abs(cnc_pos[next] - rgv_pos)]; //增加移动的
时间
                rgv_pos = cnc_pos[next];
                //不能默认有熟料


                cnc_endtime[next] = time+complete2_2+updown[next&1]; // 更
新结束时间
                if(cnc_now[next]) {//如果 cnc 有工件
                    result2[cnc_now[next]].last_end = time + clean +
updown[next & 1];//记录成品
                    if(result2[cnc_now[next]].last_end < 28800)
                        cnt_end ++ ; //加工完
                    result2[cnc_now[next]].second_end = time; // 记录开始下
料时间
                    cnc_now[next] = rgv_cnt; // 更新现在 cnc 上的工件

                    result2[cnc_now[next]].i = cnc_now[next];
                    result2[cnc_now[next]].second_start = time; // 记录开始上
料时间

                    time+=updown[next&1];//上下料
                    time+=clean;//   清洗

                    result2[cnc_now[next]].second_cnc = next;

                    v.clear();
                    cout << time << '\n';

                    rgv_hand = false; // rgv 手上没有了
                    continue;
                }
```

```
                else{
                    cnc_now[next] = rgv_cnt; //  更新现在 cnc 上的工件

                    result2[cnc_now[next]].i =cnc_now[next];
                    result2[cnc_now[next]].second_start = time; //  记录开始上
料时间
                    time+=updown[next&1];//上下料
                    result2[cnc_now[next]].second_cnc = next;
                    v.clear();
                    cout << time << '\n';

                    rgv_hand = false; // rgv 手上没有了
                    continue;
                }
            }
            //rgv 手上没有加工件

            bool flag = false;
            for(int i=1;i<=8;i++){
                if(!cnc_pp[i] && cnc_endtime[i] <= time){
                    flag |= 1;
                    break;
                }
            }
            if(!flag){
                //提前移动
                int index = 1;
                while(cnc_pp[index])index++; //找到第一个是第一工序的 cnc
                for(int i=2;i<=8;i++){
                    if(!cnc_pp[i] && cnc_endtime[index] > cnc_endtime[i]){
                        index = i;
                    }
                    else if(!cnc_pp[i] && cnc_endtime[index] ==
cnc_endtime[i]){
```

```
                        if(abs(cnc_pos[index] - rgv_pos) > abs(cnc_pos[i] -
rgv_pos))
                            index = i;
                    }
                }

#if early
                time += max(move_time[abs(cnc_pos[index] - rgv_pos)] ,
cnc_endtime[index] - time);//pass time
                rgv_pos = cnc_pos[index];
#else
                time +=    cnc_endtime[index] - time;//pass time
#endif
                continue;
            }

            vector<int>v;
            for(int i=1;i<=8;i++){
                if(!cnc_pp[i] && cnc_endtime[i] <= time){
                    v.push_back(i);
                    cout << i << ' ';
                }
            }
            //jude
            cout << '*';
            int next = jude2_find_first(v,time);//rgv 下一个要到的地点为 next 个
cnc
            cout << next << '\n';

            time+=move_time[abs(cnc_pos[next] - rgv_pos)]; //增加移动的时间
            rgv_pos = cnc_pos[next];


            cnc_endtime[next] = time+complete2_1+updown[next&1]; //  更新结
```

```
束时间
            result2[cnc_now[next]].first_end = time; //  记录开始下料时间
            rgv_cnt = cnc_now[next];
            cnc_now[next] = ++cnt; //  更新现在 cnc 上的工件

            result2[cnc_now[next]].i = cnt;
            result2[cnc_now[next]].first_start = time; //  记录开始上料时间

            time+=updown[next&1];//上下料

            result2[cnc_now[next]].first_cnc = next;
            v.clear();
            cout << time << '\n';

            rgv_hand = true; // rgv 手上有了
        }
}
void solve(){
        int time = 0;
        //纯上料
        //int a[9] = {0,1,3,5,7,8,6,4,2};
        for(int i=1;i<=8;i++){

            time += move_time[abs(rgv_pos - cnc_pos[a[i]])];//移动
            rgv_pos = cnc_pos[a[i]];

            cnc_now[a[i]] = ++cnt;
            result[cnt].start = time;
            time += updown[a[i]&1];
            cnc_endtime[a[i]] = time + complete;
            result[cnt].i = cnt;
            result[cnt].cnc = a[i];

        }
```

```
    while(time < 28800){
        bool flag = false;
        for(int i=1;i<=8;i++){
            if(cnc_endtime[i] <= time){
                flag |= 1;
                break;
            }
        }
        if(!flag){
            //提前移动
            int index = 1;
            for(int i=2;i<=8;i++){
                if(cnc_endtime[index] > cnc_endtime[i]){
                    index = i;
                }
                else if(cnc_endtime[index] == cnc_endtime[i]){
                    if(abs(cnc_pos[index] - rgv_pos) > abs(cnc_pos[i] - rgv_pos))
                        index = i;
                }
            }
#if early
            time += max(move_time[abs(cnc_pos[index] - rgv_pos)] , cnc_endtime[index] - time);//pass time
            rgv_pos = cnc_pos[index];
#else
            time +=  cnc_endtime[index] - time;//pass time
#endif
            continue;
        }
        vector<int>v;
        for(int i=1;i<=8;i++){
            if(cnc_endtime[i] <= time){
                v.push_back(i);
```

```
                cout << i << ' ';
            }
        }
        //jude
        cout << '*';
        int next = jude(v,time);//rgv 下一个要到的地点为 next 个 cnc
        cout << next << '\n';

        time+=move_time[abs(cnc_pos[next] - rgv_pos)]; //增加移动的时间
        rgv_pos = cnc_pos[next];
        //默认有熟料
        result[cnc_now[next]].end = time; //  记录开始下料时间
        cnc_endtime[next] = time+complete+updown[next&1];

        result[cnc_now[next]].last_end = time+clean+updown[next&1];//add
        if(result[cnc_now[next]].last_end < 28800)
            cnt_end ++ ; //加工完
        cnc_now[next] = ++cnt;
        result[cnc_now[next]].i = cnc_now[next];
        result[cnc_now[next]].start = time; //  记录开始上料时间

        time+=updown[next&1];//上下料
        time+=clean;//    清洗

        result[cnc_now[next]].cnc = next;

        v.clear();
        cout << time << '\n';
    }
}
void solve3_1(){
    int time = 0;
    //纯上料
    //int a[9] = {0,1,3,5,7,8,6,4,2};
```

```
for(int i=1;i<=8;i++){

    time += move_time[abs(rgv_pos - cnc_pos[a[i]])];//移动
    rgv_pos = cnc_pos[a[i]];

    cnc_now[a[i]] = ++cnt;
    result[cnt].start = time;
    time += updown[a[i]&1];
    cnc_endtime[a[i]] = time + complete;
    result[cnt].i = cnt;
    result[cnt].cnc = a[i];

    //生产故障
    pair<bool,int>p = lucky_Roulette();
    if(p.first){
        ++fault_size;
        fault[fault_size].cnc = a[i];
        int fst = (rand()%complete) + time;
        fault[fault_size].start_time = fst;
        fault[fault_size].end_time = fst+p.second;
        fault[fault_size].cnc_now = cnc_now[fault[fault_size].cnc];
        fault[fault_size].i = fault_size;

        cnc_now[a[i]] = 0;
        cnc_endtime[a[i]] = 0;
        cnc_fault_endtime[a[i]] = fst+p.second;

        result[cnt].end = -1;
        result[cnt].last_end = -1;
    }

}
while(time < 28800){
    bool flag = false;
```

```
        for(int i=1;i<=8;i++){
            if(cnc_endtime[i] <= time && cnc_fault_endtime[i] <= time){
                flag |= 1;
                break;
            }
        }
        if(!flag){
            //提前移动
            int index = 1;
            while(cnc_fault_endtime[index] > time)index++;//排除故障
            for(int i=2;i<=8;i++){
                if(cnc_endtime[index] > cnc_endtime[i] &&
cnc_fault_endtime[i] <= time){
                    index = i;
                }
                else if(cnc_endtime[index] == cnc_endtime[i] &&
cnc_fault_endtime[i] <= time){
                    if(abs(cnc_pos[index] - rgv_pos) > abs(cnc_pos[i] -
rgv_pos))
                        index = i;
                }
            }
            time += max(move_time[abs(cnc_pos[index] - rgv_pos)] ,
cnc_endtime[index] - time);//pass time
            rgv_pos = cnc_pos[index];
            continue;
        }
        vector<int>v;
        for(int i=1;i<=8;i++){
            if(cnc_endtime[i] <= time  && cnc_fault_endtime[i] <= time){
                v.push_back(i);
                cout << i << ' ';
            }
        }
```

```
            //jude
            cout << '*';
            int next = jude(v,time);//rgv 下一个要到的地点为 next 个 cnc
            cout << next << '\n';

            time+=move_time[abs(cnc_pos[next] - rgv_pos)]; //增加移动的时间
            rgv_pos = cnc_pos[next];
            if(cnc_now[next])
                result[cnc_now[next]].end = time; //  记录开始下料时间
            cnc_endtime[next] = time+complete+updown[next&1];
            if(cnc_now[next]){
                result[cnc_now[next]].last_end                                    =
time+clean+updown[next&1];//add
                if(result[cnc_now[next]].last_end < 28800)
                    cnt_end ++ ; //加工完
            }


            cnc_now[next] = ++cnt;
            result[cnc[cnt].i = cnt;
            result[cnt].start = time; //  记录开始上料时间

            time+=updown[next&1];//上下料
            time+=clean;//    清洗

            result[cnt].cnc = next;

            v.clear();


            pair<bool,int>p = lucky_Roulette();
            if(p.first){
                ++fault_size;
                fault[fault_size].cnc = next;
```

```
                int fst = (rand()%complete) + time;
                fault[fault_size].start_time = fst;
                fault[fault_size].end_time = fst+p.second;
                fault[fault_size].cnc_now = cnc_now[fault[fault_size].cnc];
                fault[fault_size].i = fault_size;

                cnc_now[next] = 0;
                cnc_endtime[next] = 0;
                cnc_fault_endtime[next] = fst+p.second;

                result[cnt].end = -1;
                result[cnt].last_end = -1;
            }

            cout << time << '\n';
        }
}
void solve3_2(){
    int time = 0;
    //上 1 料
    for(int i=1;i<=a2_size;i++){
        time += move_time[abs(rgv_pos - cnc_pos[a2[i]])];//移动
        rgv_pos = cnc_pos[a2[i]];

        cnc_now[a2[i]] = ++cnt;
        result2[cnc_now[a2[i]]].i = cnt;
        result2[cnc_now[a2[i]]].first_cnc = a2[i];
        result2[cnc_now[a2[i]]].first_start = time;
        time += updown[a2[i]&1];
        cnc_endtime[a2[i]] = time + complete2_1;

        pair<bool,int>p = lucky_Roulette();
        if(p.first){
            ++fault_size;
```

```
                fault[fault_size].cnc = a2[i];
                int fst = (rand()%complete2_1) + time;
                fault[fault_size].start_time = fst;
                fault[fault_size].end_time = fst+p.second;
                fault[fault_size].cnc_now = cnc_now[fault[fault_size].cnc];
                fault[fault_size].i = fault_size;

                result2[cnc_now[a2[i]]].first_end = -1;
                result2[cnc_now[a2[i]]].second_start = -1;
                result2[cnc_now[a2[i]]].second_end = -1;
                result2[cnc_now[a2[i]]].second_cnc = -1;
                result2[cnc_now[a2[i]]].last_end = -1;



                cnc_now[a2[i]] = 0;
                cnc_endtime[a2[i]] = 0;
                cnc_fault_endtime[a2[i]] = fst+p.second;

            }


        }
        while(time < 28800){
            if(rgv_hand){//有第一道工序的成品，需要去第二道工序

                bool flag = false;
                for(int i=1;i<=8;i++){
                    if(cnc_pp[i] && cnc_endtime[i] <= time &&
cnc_fault_endtime[i] <= time){
                        flag |= 1;
                        break;
                    }
                }
                if(!flag){
```

```
                    //提前移动
                    int index = 1;
                    while((!cnc_pp[index]||cnc_fault_endtime[index] > time )&& index<=8)index++; //找到第一个是第二工序的 cnc
                    if(index == 9){//所有的都故障了
                        //找到故障时间最短的
                        index = 1;
                        while(!cnc_pp[index])index++;
                        for(int i=2;i<=8;i++){
                            if(cnc_pp[i] && cnc_fault_endtime[index] > cnc_fault_endtime[i]){
                                index = i;
                            }
                            else if(cnc_pp[i] && cnc_endtime[index] == cnc_endtime[i]){
                                if(abs(cnc_pos[index] - rgv_pos) > abs(cnc_pos[i] - rgv_pos))
                                    index = i;
                            }
                        }
#if early
                        time += max(move_time[abs(cnc_pos[index] - rgv_pos)] , cnc_fault_endtime[index] - time);//pass time
                        rgv_pos = cnc_pos[index];
#else
                        time += cnc_fault_endtime[index] - time;//pass time
#endif
                        continue;
                    }//都故障
                    for(int i=2;i<=8;i++){
                        if(cnc_pp[i] && cnc_endtime[index] > cnc_endtime[i] && cnc_fault_endtime[i] <= time){
                            index = i;
                        }
```

```
                else if(cnc_pp[i] && cnc_endtime[index] == cnc_endtime[i] && cnc_fault_endtime[i] <= time){
                    if(abs(cnc_pos[index] - rgv_pos) > abs(cnc_pos[i] - rgv_pos))
                        index = i;
                }
            }


#if early
            time += max(move_time[abs(cnc_pos[index] - rgv_pos)] , cnc_endtime[index] - time);//pass time
            rgv_pos = cnc_pos[index];
#else
            time +=    cnc_endtime[index] - time;//pass time
#endif
            continue;
        }

        vector<int>v;
        for(int i=1;i<=8;i++){
            if(cnc_pp[i] && cnc_endtime[i] <= time && cnc_fault_endtime[i] <= time){
                v.push_back(i);
                cout << i << ' ';
            }
        }
        //jude
        cout << '*';
        int next = jude2_find_second(v,time);//rgv 下一个要到的地点为 next 个 cnc
        cout << next << '\n';

        time+=move_time[abs(cnc_pos[next] - rgv_pos)]; //增加移动的
```

```
时间
                rgv_pos = cnc_pos[next];
                //不能默认有熟料


                cnc_endtime[next] = time+complete2_2+updown[next&1]; //  更新结束时间
                if(cnc_now[next]) {//如果 cnc 有工件
                    result2[cnc_now[next]].last_end = time + clean + updown[next & 1];//记录成品
                    if(result2[cnc_now[next]].last_end < 28800)
                        cnt_end ++ ; //加工完
                    result2[cnc_now[next]].second_end = time; //  记录开始下料时间
                    cnc_now[next] = rgv_cnt; //  更新现在 cnc 上的工件

                    result2[cnc_now[next]].i = cnc_now[next];
                    result2[cnc_now[next]].second_start = time; //  记录开始上料时间

                    time+=updown[next&1];//上下料
                    time+=clean;//    清洗

                    result2[cnc_now[next]].second_cnc = next;

                    v.clear();
                    cout << time << '\n';

                    pair<bool,int>p = lucky_Roulette();
                    if(p.first){
                        ++fault_size;
                        fault[fault_size].cnc = next;
                        int fst = (rand()%complete2_2) + time - clean;//减去多加的清洗时间
```

```
                    fault[fault_size].start_time = fst;
                    fault[fault_size].end_time = fst+p.second;
                    fault[fault_size].cnc_now                                    =
cnc_now[fault[fault_size].cnc];
                    fault[fault_size].i = fault_size;

                    result2[cnc_now[next]].second_end = -1;
                    result2[cnc_now[next]].last_end = -1;



                    cnc_now[next] = 0;
                    cnc_endtime[next] = 0;
                    cnc_fault_endtime[next] = fst+p.second;

                }

                rgv_hand = false; // rgv 手上没有了
                continue;
            }
            else{
                cnc_now[next] = rgv_cnt; //  更新现在 cnc 上的工件

                result2[cnc_now[next]].i =cnc_now[next];
                result2[cnc_now[next]].second_start = time; //  记录开始上
料时间
                time+=updown[next&1];//上下料
                result2[cnc_now[next]].second_cnc = next;
                v.clear();
                cout << time << '\n';

                pair<bool,int>p = lucky_Roulette();
                if(p.first){
                    ++fault_size;
                    fault[fault_size].cnc = next;
```

```
                    int fst = (rand()%complete2_2) + time;
                    fault[fault_size].start_time = fst;
                    fault[fault_size].end_time = fst+p.second;
                    fault[fault_size].cnc_now                    =
cnc_now[fault[fault_size].cnc];
                    fault[fault_size].i = fault_size;

                    result2[cnc_now[next]].second_end = -1;
                    result2[cnc_now[next]].last_end = -1;



                    cnc_now[next] = 0;
                    cnc_endtime[next] = 0;
                    cnc_fault_endtime[next] = fst+p.second;


                }

                rgv_hand = false; // rgv 手上没有了
                continue;
            }
        }
        //rgv 手上没有加工件

        bool flag = false;
        for(int i=1;i<=8;i++){
            if(!cnc_pp[i]  &&  cnc_endtime[i]  <=  time  &&
cnc_fault_endtime[i] <= time){
                flag |= 1;
                break;
            }
        }
        if(!flag){
            //提前移动
```

```
                int index = 1;
                while(index<=8 &&(cnc_pp[index] || cnc_fault_endtime[index] >
time ))index++; //找到第一个是第一工序的 cnc
                if(index == 9){//所有的都故障了
                    //找到故障时间最短的
                    index = 1;
                    while(cnc_pp[index])index++;
                    for(int i=2;i<=8;i++){
                        if(!cnc_pp[i]  &&  cnc_fault_endtime[index]  >
cnc_fault_endtime[i]){
                            index = i;
                        }
                        else  if(!cnc_pp[i]  &&  cnc_endtime[index]  ==
cnc_endtime[i]){
                            if(abs(cnc_pos[index] - rgv_pos) > abs(cnc_pos[i]
- rgv_pos))
                                index = i;
                        }
                    }
#if early
                    time  +=  max(move_time[abs(cnc_pos[index]  -  rgv_pos)] ,
cnc_fault_endtime[index] - time);//pass time
                    rgv_pos = cnc_pos[index];
#else
                    time +=    cnc_fault_endtime[index] - time;//pass time
#endif
                    continue;
                }//都故障
                for(int i=2;i<=8;i++){
                    if(!cnc_pp[i] && cnc_endtime[index] > cnc_endtime[i] &&
cnc_fault_endtime[i] <= time){
                        index = i;
                    }
                    else if(!cnc_pp[i] && cnc_endtime[index] == cnc_endtime[i]
```

```
&& cnc_fault_endtime[i] <= time){
                        if(abs(cnc_pos[index] - rgv_pos) > abs(cnc_pos[i] - rgv_pos))
                            index = i;
                    }
                }

#if early
                time += max(move_time[abs(cnc_pos[index] - rgv_pos)] , cnc_endtime[index] - time);//pass time
                rgv_pos = cnc_pos[index];
#else
                time +=  cnc_endtime[index] - time;//pass time
#endif
                continue;
            }

            vector<int>v;
            for(int i=1;i<=8;i++){
                if(!cnc_pp[i] && cnc_endtime[i] <= time && cnc_fault_endtime[i] <= time){
                    v.push_back(i);
                    cout << i << ' ';
                }
            }
            //jude
            cout << '*';
            int next = jude2_find_first(v,time);//rgv 下一个要到的地点为 next 个 cnc
            cout << next << '\n';

            time+=move_time[abs(cnc_pos[next] - rgv_pos)]; //增加移动的时间
            rgv_pos = cnc_pos[next];
```

```
cnc_endtime[next] = time+complete2_1+updown[next&1]; // 更新结束时间
result2[cnc_now[next]].first_end = time; // 记录开始下料时间
rgv_cnt = cnc_now[next];
cnc_now[next] = ++cnt; // 更新现在 cnc 上的工件

result2[cnc_now[next]].i = cnt;
result2[cnc_now[next]].first_start = time; // 记录开始上料时间

time+=updown[next&1];//上下料

result2[cnc_now[next]].first_cnc = next;
v.clear();
cout << time << '\n';

pair<bool,int>p = lucky_Roulette();
if(p.first){
    ++fault_size;
    fault[fault_size].cnc = next;
    int fst = (rand()%complete2_1) + time;
    fault[fault_size].start_time = fst;
    fault[fault_size].end_time = fst+p.second;
    fault[fault_size].cnc_now = cnc_now[fault[fault_size].cnc];
    fault[fault_size].i = fault_size;


    result2[cnc_now[next]].first_end = -1;
    result2[cnc_now[next]].second_start = -1;
    result2[cnc_now[next]].second_end = -1;
    result2[cnc_now[next]].second_cnc = -1;
    result2[cnc_now[next]].last_end = -1;

    cnc_now[next] = 0;
```

```
                cnc_endtime[next] = 0;
                cnc_fault_endtime[next] = fst+p.second;

            }

            rgv_hand = true; // rgv 手上有了
        }
    }
    void outfile(){
        std::ofstream file("ans.xls"); //  创建 ofstream 对象并打开文件
        if (!file.is_open()) {
            std::cerr << "无法打开文件" << std::endl;
        }
        file << "加工物料序号\t 加工 CNC 编号\t 上料开始时间\t 下料开始时间\t 清晰结束时间\n";
        for(int i=1;i<=cnt_end;i++){
            file << result[i].i << '\t';
            file << result[i].cnc << '\t' << result[i].start << '\t' << result[i].end << '\t' <<   result[i].last_end << '\n';
        }
        file.close(); //  关闭文件
    }
    void outfile2(){
        std::ofstream file("ans.xls"); //  创建 ofstream 对象并打开文件
        if (!file.is_open()) {
            std::cerr << "无法打开文件" << std::endl;
        }
        file << "加工物料序号\t 工序 1 的 CNC 编号\t 上料开始时间\t 下料开始时间\t 工序 2 的 CNC 编号\t 上料开始时间\t 下料开始时间\t 清洗结束时间\n";
        for(int i=1;i<=cnt_end;i++){
            file << result2[i].i << '\t' << result2[i].first_cnc << '\t' << result2[i].first_start << '\t'<< result2[i].first_end << '\t';
            file << result2[i].second_cnc << '\t' << result2[i].second_start << '\t' << result2[i].second_end << '\t' << result2[i].last_end << '\n';
```

```
    }
    file.close(); //  关闭文件
}
void outfile3_3(){
    std::ofstream file("jun.xls"); //  创建 ofstream 对象并打开文件
    if (!file.is_open()) {
        std::cerr << "无法打开文件" << std::endl;
    }
    file << "故障时的物料序号\t 故障 CNC 编号\t 故障开始时间\t 故障结束时间\n";
    for(int i=1;i<=fault_size;i++){
        file << fault[i].cnc_now << '\t' << fault[i].cnc << '\t' << fault[i].start_time << '\t' << fault[i].end_time << '\n';
    }
    file.close(); //  关闭文件
}
void outfile3_1(){
    std::ofstream file("ans_fault.xls"); //  创建 ofstream 对象并打开文件
    if (!file.is_open()) {
        std::cerr << "无法打开文件" << std::endl;
    }
    file  << "物料数量\t 系统工作效率\n 故障数量\n";
    file    << cnt_end << '\t' <<    ((double)cnt_end/390.84) << '\t' << fault_size<<'\n';
    file.close(); //  关闭文件
}
void init(){
    //init constant  初始化常量
#if GROUP==1
    move_t1 = 20;
    move_t2 = 33;
    move_t3 = 46;
    complete = 560;
    complete2_1 = 400;
```

```
        complete2_2 = 378;
        updown_odd = 31;
        updown_even = 28;
        clean = 25;
        a2_size = 4;
#elif GROUP == 2
        move_t1 = 23;
        move_t2 = 41;
        move_t3 = 59;
        complete = 580;
        complete2_1 = 280;
        complete2_2 = 500;
        updown_odd = 35;
        updown_even = 30;
        clean = 30;
        a2_size = 4;
#elif GROUP == 3
        move_t1 = 18;
        move_t2 = 32;
        move_t3 = 46;
        complete = 545;
        complete2_1 = 455;
        complete2_2 = 182;
        updown_odd = 32;
        updown_even = 27;
        clean = 25;
        a2_size = 5;
#endif
        move_time[0]=0;
        move_time[1]=move_t1;
        move_time[2]=move_t2;
        move_time[3]=move_t3;
        updown[0] = updown_odd;
        updown[1] =updown_even;
```

```
        //init  初始化调度情况
        //得到调度指令
#ifdef OPEN_HAND_PROPORTION
        a2_size = PROPORTION;
#endif

        //init cnc_pp
        for(int i=1;i<=8;i++)cnc_pp[i] = true;
}
void init(int x){
        switch (x) {
                case 1:move_t1 = 20;
                        move_t2 = 33;
                        move_t3 = 46;
                        complete = 560;
                        complete2_1 = 400;
                        complete2_2 = 378;
                        updown_odd = 31;
                        updown_even = 28;
                        clean = 25;
                        a2_size = 4;
                        break;
                case 2: move_t1 = 23;
                        move_t2 = 41;
                        move_t3 = 59;
                        complete = 580;
                        complete2_1 = 280;
                        complete2_2 = 500;
                        updown_odd = 35;
                        updown_even = 30;
                        clean = 30;
                        a2_size = 4;
                        break;
                case 3: move_t1 = 18;
```

```
                move_t2 = 32;
                move_t3 = 46;
                complete = 545;
                complete2_1 = 455;
                complete2_2 = 182;
                updown_odd = 32;
                updown_even = 27;
                clean = 25;
                a2_size = 5;
                break;
            default:return ;
        }
        move_time[0]=0;
        move_time[1]=move_t1;
        move_time[2]=move_t2;
        move_time[3]=move_t3;
        updown[0] = updown_odd;
        updown[1] =updown_even;
        //init cnc_pp
        for(int i=1;i<=8;i++)cnc_pp[i] = true;
}


inline pair<bool,int> lucky_Roulette(){
        //srand((rand()%100)+10);
        //srand(time(nullptr));
        //mt19937 Rnd(random_device{}());

        std::random_device rd;  // 将用于为随机数引擎获得种子
        std::mt19937 gen(rd()); //  以播种标准  mersenne_twister_engine
        std::uniform_int_distribution<> dis(1, 100),dis2(600,1200);
        int x = dis(gen);
        if(x!=30)return make_pair(false,0);
        //return make_pair(false,0);
```

```
        //srand(time(nullptr));
        //int lucky_time = (rand()%600)+600;
        int lucky_time = dis2(gen);
        return make_pair(true,lucky_time);
}
```