基于图论的智能 RGV 动态调度研究

摘要

本文以工件加工车间的轨道式自动引导车（RGV）为研究对象，基于动态规划理论、以二叉树为基础的深度搜索方案，建立单一工艺生产流程中静态调度策略模型；并以图论中的二部图抽象出两道工艺工件的流程路线基本结构，基于贪婪算法，采用返程动态规划最优求解方案建立双重工艺生产流程中静态调度策略模型；最后通过对两种工艺生产流程以及随机故障情形的模拟与求解，检验模型动态调度策略的有效性，同时以单位连续时间内生产工件数衡量的作业效率，得出模型具备提升生产流程整体性能的结论。

针对情形一，我们采用局部动态规划和二叉树算法对 RGV 的策略点集进行优化，将整个策略点集分解为若干阶段的策略点集，并对每一个策略点集进行局部最优化。以进一步消除 RGV 策略的后效性，确保每一步决策w_i达到局部时间最小化效果，从而达到全局策略的最优化。事实上，通过编程时不同参数的设置，我们还可实现三叉树、四叉树的应用，使全局策略进一步优化。而最终通过系统作业参数进行检验，验证了此种模型的有效性与高效性。

针对情形二，我们在情形一的基础上加以改进，首先采用对偶规划，先设置分配比例、距离矩阵等为参数集，设置完成成品数为变量，借助图论的二部图理论建立了动态调度模型。然后改设分配比例、距离矩阵等为变量集，设置完成成品数为参数，通过调节参数来获得合理的分配比例、距离矩阵。最后借助启发式算法，获得基于已构建动态调度模型的最优调度方案。最终 3 组检验结果与 3 组系统作业参数情况非常契合。

针对情形三，我们利用蒙特卡洛算法引入 3 个随机变量，以此来模拟故障发生的情况，在情形一、二的基础上分别进行模拟，探究故障对系统性能的影响。

关键词：返程动态规划 二叉树 图论 蒙特卡洛算法

一、问题重述

1.1 问题背景

轨道式自动导引车（Rail Guide Vehicle，RGV）在现代物流存储和机械加工中起到非常突出的作用，RGV 是一种无人驾驶、能在固定轨道上自由运行的智能车，在输送货物时运行速度快且准确无误。因此，RGV 既弥补了货物与其他设备的缺口，同时也降低了管理人员的工作量，显著提高了生产效率。

但在轨道式自动导引车（RGV）的运输效率是瓶颈问题。计算机数控机床（Computer Number Controller，CNC）的位置、工序的先后及刀具的放置等都会影响 RGV 运行的路线，从而极大影响其运行效率。因此，为提高自动化立体仓库的整体效率，对 RGV 的动态调度进行研究具有重要意义。

1.2 问题提出

考虑由 8 台计算机数控机床、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构成的智能加工系统：

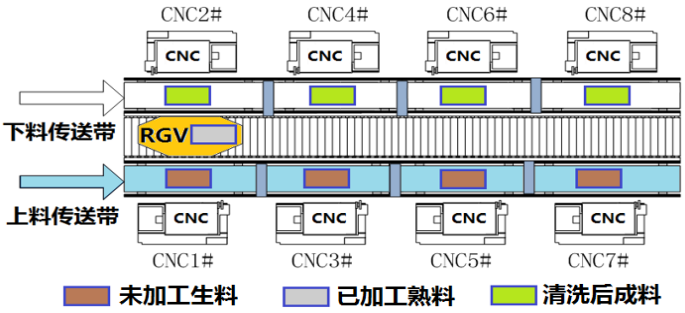

图 1 智能加工系统示意图

我们将针对此系统的三种具体情况建立数学模型，分别给出 RGV 动态调度的模型和相应的求解算法，并且利用数据检验模型的实用性和和算法的有效性，给出 RGV 的调度策略和智能加工系统的作业效率。三种具体情况如下：

1、一道工序的物料加工作业情况，每台 CNC 安装同样的刀具，物料可以在任一台 CNC 上加工完成；

2、两道工序的物料加工作业情况，每个物料的第一和第二道工序分别由两台不同的 CNC 依次加工完成；

3、CNC 在加工过程中可能发生故障（据统计，故障的发生率约为 1%）的清苦那个，每次故障排除后即刻加入作业序列。分别考虑一道工序和两道工序的物料加工作业情况。

二、问题分析

2.1 对情况一的分析

现代车间工艺生产中涉及的工件多具有工艺需求量大、复杂性较高、制作周期较长等特点。此时需要结合零件工序的变动方案，将零件工序的加工任务调度到适当的机床上，以制订有效的车间作业计划，确保当日工件生产最大化并解决关键资源不出现等待的问题。为解决这类问题，多采用遗传算法、模拟退火算法、蚁群算法等现代算法进行求解和智能控制。然而，此类算法在全局收敛性、收敛速度等缺陷也日益显现。针对情形一中单一工艺的特点和上下料与清洗时间等多种因素，我们在模型中采用以动态规划为主、图论思想为辅的深度搜索原理进行生产程序的优化。[1]

2.2 对情况二的分析

情形二较情形一更为复杂，需要考虑的问题包括：8 台数控机床中安装第一、二道工序刀具的比例、安装好刀具后 8 台数控机床的位置分布、RGV 的动态调度模型等。为获得最大生产效率，我们将尽量减少机床空闲时间，并且尽量避免出现半成品冗余情况。

因此，第一步，以分别负责两道工序的机床数量、机床之间的距离矩阵为参数，设计 RGV 的动态调度模型；第二步，我们将以分别负责两道工序的机床数量、机床之间的距离矩阵为变量，以成品数量 n 为参数，建立目标函数 f，从而得到 n 件成品所需要的时间；第三步，给目标函数赋值时间 h，调节参数 n，获得欲在 h 时间内生产 n 件成品各变量满足的关系 s；第四步，当 n 参数尽可能大时，在利用约束条件，获得满足关系 s 的变量的值。若无法进行有效约束，则使用启发式算法确定机床关于工序的分配和位置分布。

2.3 对情况三的分析

机器故障是企业在实际生产加工中经常遇到的。动态柔性作业车间调度更加贴合企业的生产实际。在该调度过程中，机器的故障、紧急工件的插入、原材料的短缺、工期的弹性调整等动态事件是企业在生产加工过程中经常面临的问题，而机器的故障又是所有动态事件中最普遍的。性能较优的设备具备较低的故障概率，但也无法做到完全消除故障概率。当加工过程中发生机器故障时，需要人工的修理调度和 RGV 的重调度策略来调整生产流程。[2]

三、模型假设

1) 假设 RGV 只有在收到 CNC 发出的需求信号后，才能进行路线规划，执行移动、停止等待、上下料、清洗作业中的一项；
2) 在需求信号发出前，RGV 无法得知各台 CNC 的加工情况；

3) 假设 RGV 承载生料时只能去往安装有第一道工序刀具的 CNC 上下料，承载半成品时只能去往安装有第二道工序刀具的 CNC 上下料；
4) 各时刻（除故障时）RGV 和 CNC 的状态保持稳定；
5) 各台 CNC 加工状态保持独立

四、符号说明

符号	含义
t	时间模拟器的时刻
t_w	RGV 清洗工件所需时间
t_m	RGV 上下料所需时间
t_0	RGV 等待指令所需时间
t^*	CNC 工序所需时间
O_{ij}	工件状态点
w_i	RGV 策略点
$K\{O_{ij}\}$	全体工件状态点集
$K\{w_i\}$	全体 RGV 策略点集
v_i	CNC 序号
E	单次策略所需的期望时间
$\bar{E}$	单次二部图路径所需的时间
n	成品数量
f	计算时间总量的目标函数
X	目标函数f的变量集合
h	对目标函数f的赋值时间
d_{ij}	边(v_i, v_j)对应的权重
D	有向边权重矩阵
G	有向边概率矩阵
A	从事第一道工序的 CNC 序号集合
B	从事第二道工序的 CNC 序号集合
ξ_1	故障随机变量
ξ_2	发生故障时间点随机变量
ξ_3	维修时间随机变量
F	ξ_2对应的分布函数
G	ξ_3对应的分布函数
η	模型效率

五、模型的建立、求解与检验

5.1 情形一

5.1.1 模型的准备

由于考虑到这一情况：只对当前情况进行最优化规划，会由此忽略后效性，对系统最终结果造成较大影响，因此贪婪算法在此处并不适用。从初始情况开始，情况如下：

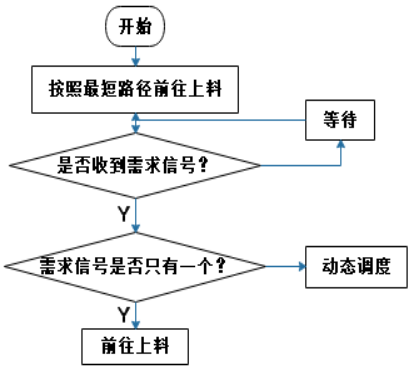

图 2 情况一初始上料情况

5.1.2 模型的建立

为使系统的作业效率达到最高，单位时间加工的物料应达到最多，或一定量的物料所需要的加工时间最短。单次连续工作时间为八小时，目的是获得八小时中最多的下件个数 n，则每次移动响应都会清洗操作一次，而八个机器在八小时内必然会使得一个机器投入生产，则：移动次数+8=n。

考虑当固定 n，则问题变为求总时间最小值 t。此时以尽可能缩短 RGV 应答 CNC 需求信号所需时间为主要目的。建立目标函数如下：

$$\boldsymbol{min}t=\sum_{i=1}^{n} t_i$$

其中，$t_i = \mathrm{mov}(v_{i-1}, v_i) + t_{iw} + t_{im} + t_{io}$

——t_{iw}为清洗一次所需要的时间，t_{im}为上下料一次所需要的时间，t_{io}为等待下一次请求所需要的时间。

vi∈{1,2,3,4,5,6,7,8}；

以下是求解目标函数最小值的约束条件：

1）v_i代表 RGV 的位置状态点。v_i与下一状态点v_{i+1}之间的距离可用示性函数衡量：

$$\mathrm{mov}(v_{i-1}, v_i) = \begin{cases} 0, \left|\left[\frac{v_i}{2}\right] - \left[\frac{v_{i-1}}{2}\right]\right| = 0 \\ d_1, \left|\left[\frac{v_i}{2}\right] - \left[\frac{v_{i-1}}{2}\right]\right| = 1 \\ d_2, \left|\left[\frac{v_i}{2}\right] - \left[\frac{v_{i-1}}{2}\right]\right| = 2 \\ d_3, \left|\left[\frac{v_i}{2}\right] - \left[\frac{v_{i-1}}{2}\right]\right| = 3 \end{cases}$$

其中，[.]为向下取整函数。

2）令每一工件的状态点为O_{ij}，其中 i 表示工件的序号，j 表示 CNC 机器的序号，于是一个工件的基本信息可由O_{ij}表达出来。针对 i，j 有如下约束条件：

①$\forall k_1,k_2$，$\left|\mathrm{T1}\left(O_{k_1 j}\right)-\mathrm{T1}\left(O_{k_2 j}\right)\right|<t^*$

其中：T1（.）是一个状态点集$K\{O_{ij}\}\rightarrow\{v_i\}$的单射，返回$O_{ij}$状态点所代表的上料时刻；$t^*$表示完成工序所需要的时间。

特别地，$\left|\mathrm{T1}\left(O_{k_1 j}\right)-\mathrm{T1}\left(O_{k_2 j}\right)\right|<t^*+\mathrm{mov}\left(V\left(O_{k_1 j}\right),V\left(O_{k_2 j}\right)\right)+t_{iw}$

其中:V（.）是一个状态点集$K\{O_{ij}\}\rightarrow\mathbb{N}^+$的单射，返回$O_{ij}$状态点所在的机器序号。

②若$\exists k_1\ s.t.\ O_{k_1 j}\in K$，则$\forall k_2\ O_{k_2 j}\in K.$

3）RGV 的行为受到 CNC 的约束。若w_i表示 RGV 将下料第 i 号工件并上料第 i+1 号工件的整个行为或决策，则

$$\mathrm{T2}(w_i)-\mathrm{T1}(O_{ik})\geq 0$$

其中：T2（.）是一个策略点集$K\{w_i\}\rightarrow\mathbb{N}^+$的单射，返回RGV决定前往下料第 i 号工件的时刻；

4)设上下料时间值域为$\{t_{m1},t_{m2}\}$,则

$$T3\left(v_k\right)=\begin{cases}t_{m1}, & \text{若 } \mathrm{k}=1(\mathrm{mod}2)\\ t_{m2}, & \text{若 } \mathrm{k}=0(\mathrm{mod}2)\end{cases}$$

5.1.3 模型的求解

本模型采用局部动态规划和二叉树算法对 RGV 的策略点集进行优化，将整个策略点集分解为若干阶段的策略点集，并对每一个策略点集进行局部最优化。以进一步消除 RGV 策略的后效性，确保每一步决策w_i达到局部时间最小化效果，从而达到全局策略的最优化。

首先定义测度E来衡量某策略的时间效率。令w_i为从k_1号 CMC 机器前往k_2号 CMC 机器的策略，此时k_1号 CMC 机器的工件号为 i，即对应状态点集 K 的元素O_{ik_1}。

根据多阶段决策最优性定理，对阶段数为 n 的多阶段决策过程，设其阶段编号为 k=0,1,…,n-1,则允许策略$p_{0,n-1}^*=(u_0^*,u_1^*,\ldots,u_{n-1}^*)$是最优策略的充分必要条件是对任一个 k,0<k<n-1 和$s_0\in S_0$，有

$$V_{0,n-1}\left(s_0, p^*_{0,n-1}\right) = \operatorname*{Opt}_{p_{0,k-1}\in P_{0,k-1}(s_0)} V_{0,k-1}\left(s_0, p_{0,k-1}\right) + \operatorname*{Opt}_{p_{k,n-1}\in P_{k,n-1}(\widetilde{s_k})} V_{k,n-1}\left(\widetilde{s_k}, p_{k,n-1}\right)$$

其中：$p_{0,n-1} = (p_{0,k-1}, p_{k,n-1})$，$\widetilde{s_k}$=$T_{k-1}(s_{k-1}, u_{k-1})$.[3]

因此，根据深度优先搜索原理，对策略w_i所处时刻向后作 m 步模拟，获得策略向量（$w_{i+1}, w_{i+2}, w_{i+3}, \ldots, w_{i+m}$），各策略$w_k, k = i+1, \ldots, i+m$所指向的 CNC 机器序号需满足约束条件 3）的条件，形成备选点集$\{v_{jk}\}$。

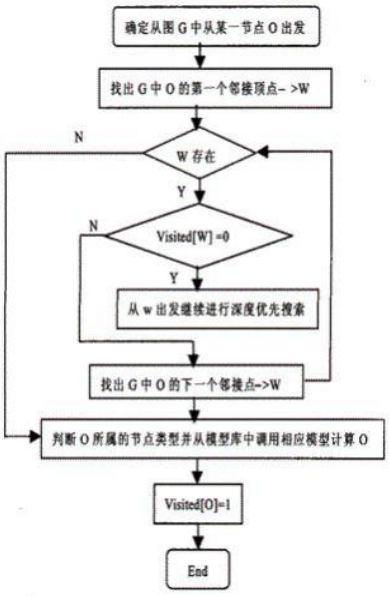

图 3 深度优先算法流程图

下面对测度 E 进行定义。E 可看做是 RGV 在策略点w_k处选择v_{ik}所需要消耗的期望时间，即

$$E_{jk} = \mathrm{mov}\left(v_k, v_{jk}\right) + \mathrm{T3}（v_{jk}）$$

通过遍历算法获得第 j 层中最小的两个（仅有一个备选点时只能取一个）E_{jk}并提取出对应的v_{jk}作为策略点w_k可能指向的 CNC 机器。依此对下一层分别计算 E 值并提取备选点，从而得到一个深度为 m 的带权二叉决策树，其权重即最小子代计算所得的最小期望时间。选取最小期望时间所指代的通路（$v_{i+1}, v_{i+2}, v_{i+3}, \ldots, v_{i+m}$）就是策略向量（$w_{i+1}, w_{i+2}, w_{i+3}, \ldots, w_{i+m}$）所对应的目标位置向量。

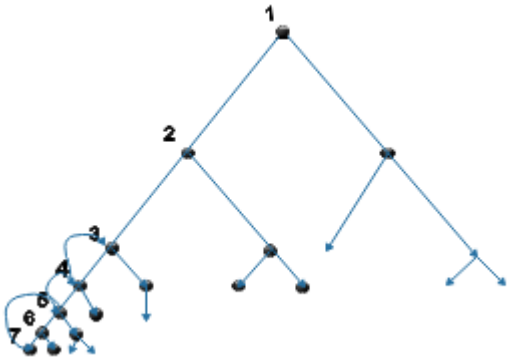

图 4 二叉树算法图解

现讨论在模型动态搜索模拟过程中，RGV 于每个策略点w_k所采取的实际行为：

1）时刻 t= $T2(w_k)$时备选点集$\{v_{jk}\}_{j=1}=\emptyset$，取$\Delta t=\max_j\{T1(O_{kj})+t^*\}-t>0$，则在时间模拟器中跳过这段时间间隔$\Delta t$，故在时刻 t+$\Delta t$时$\{v_{jk}\}_{j=1}\neq\emptyset$。

2）若$|\{v_{jk}\}_{j=1}|=1$，（|.|表示集合元素个数），则由约束条件 3），该集合中的唯一元素成为策略点w_k指向的位置点。

3）若$|\{v_{jk}\}_{j=1}|\geq 2$，则通过基于测度 E 的深度优先搜索算法确定w_k指向的位置点。

5.1.4 模型的检验

利用系统作业参数的 3 组数据进行检验，由此得到了 8h（28800s）内完成的成品件数以及 CNC 具体的调度情况：

表 1 完成成品数比较

系统作业参数及完成成品数	第 1 组	第 2 组	第 3 组
RGV 移动 1 个单位所需时间	20	23	18
RGV 移动 2 个单位所需时间	33	41	32
RGV 移动 3 个单位所需时间	46	59	46
CNC 加工完成一个一道工序的物料所需时间	560	580	545
RGV 为 CNC1#，3#，5#，7#一次上下料所需时间	28	30	27
RGV 为 CNC2#，4#，6#，8#一次上下料所需时间	31	35	32
RGV 完成一个物料的清洗作业所需时间	25	30	25
完成成品数	**375**	**354**	**381**

分析可知，若完全不考虑机床等待时间，在 8h（28800s）时间内最多能完成 411 件，由此，通过建立模型获得的完成成品数是属于合理范围内的。并且，从系统作业参数上发现，第 2 组系统作业参数的加工时间、RGV 单位移动时间、上下料时间、清洗时间在 3 组数据之中都是最长的，而由第 2 组数据所得的完成成品数也是最少的；同时，第 3 组系统作业参数的加工时间、RGV 单位移动时间、清洗时间在 3 组数据之中都是最短的，且各项时间基本略少于第 1 组系统作业参

数，而由第 3 组数据所得的完成作品数也是最多的，且略多于第 1 组数据所得完成作品数。由此，模型和算法的有效性得到检验。

$$效率（\eta）=\frac{完成成品数（n）*加工时间（t）}{8*28800}\times 100\%$$

通过计算得出，在 3 组系统参数下，系统的作业效率（η）分别为 91.14%、89.11%、90.12%，作业效率较高，反映出了模型的高效性。

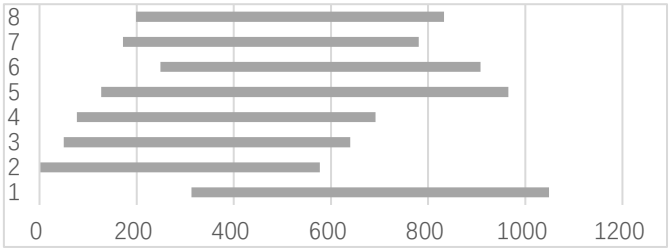

图 5 第 3 组数据加工第 1-8 个工件 CNC 调度情况[5]

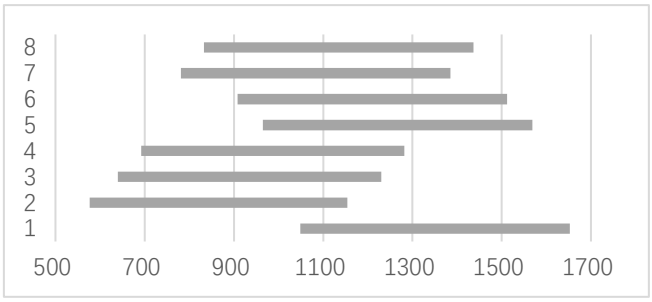

图 6 第 3 组数据加工第 9-16 个工件 CNC 调度情况

通过具体调度情况（详情见“Case_1_results.xls”）发现，在只有一道工序的加工情况下，调度具有周期性，以 8 个工件为一组，除第一个周期（第 1-8 个）外都符合这一特点。由此说明，在只有一道工序的情况下，系统的加工是稳定的。

5.2 情形二

5.2.1 模型的准备

在题设条件之下，我们的思路主要分为如下四步：第一步，以分别负责两道工序的机床数量、机床之间的距离矩阵为参数，设计 RGV 的动态调度模型；第二步，我们将以分别负责两道工序的机床数量、机床之间的距离矩阵为变量，以成品数量 n 为参数，建立目标函数 f，从而得到 n 件成品所需要的时间；第三步，给目标函数赋值时间 h，调节参数 n，获得欲在 h 时间内生产 n 件成品各变量满足的关系 s；第四步，当 n 参数尽可能大时，在利用约束条件，获得满足关系 s 的变量的值。若无法进行有效约束，则使用启发式算法确定机床关于工序的分配和位置分布。

5.2.2 模型的建立

现以分别负责两道工序的机床数量、机床之间的距离矩阵为参数α，以成品

数量 n 为变量建立动态调度模型。

根据题设条件，首先承认以下事实：

① 若策略$w_i(1)\in \mathrm{A}$,则$w_i(2)\in \mathrm{B}$；而$w_i(1)\in \mathrm{B}$,则$w_i(2)\in \mathrm{A}$ 或者$w_i(2)\in \mathrm{B}$。这是由于完成 A 中 CNC 机器的上下料操作后 RGV 的机械臂中握有半成品，此时无法继续对 A 中其他的 CNC 机器进行上下料操作；

② 完成 B 中 CNC 机器的上下料操作后 RGV 的机械臂中无物料，此时仍可以对 B 中其他的 CNC 机器进行下料操作并清洗成料，因此$w_i(1)\in \mathrm{B}$时$\mathrm{P}(w_i(2)\in \mathrm{B})>0$。此时若 B 发出需求信号而 A 未发出，则 RGV 确定前往 B 执行上下料操作；

③ 完成 B 中 CNC 机器的上下料操作后 RGV 的机械臂中无物料，此时若 A 发出需求信号而 B 未发出，则 RGV 确定前往 A 执行上下料操作；

表 2 运行默认规则

	出发	到达
①	A	B
②	B	B
③	B	A

基于以上规则，讨论在整个过程中，RGV 于每个策略点w_k所采取的实际行为，着重考虑 A、B 同时有（无）机床发出需求信号的情况：

1）时刻 t= $\mathrm{T2}(w_k)$时备选点集$\{v_{jk}\}^A{}_{j=1}=\{v_{jk}\}^B{}_{j=1}=\emptyset$或$w_k(1)\in A$且$\{v_{jk}\}^B{}_{j=1}=\emptyset$，取$\Delta t=\max_j\{\mathrm{T1}(O_{kj})+t^*\}-\mathrm{t}>0$,则在时间模拟器中跳过这段时间间隔$\Delta t$，故在时刻 t+$\Delta t$时$\{v_{jk}\}_{j=1}\neq\emptyset$。判断是否属于 1)、2)、3)、4）中任意一种，并按照判断执行；

2)若$\left|\{v_{jk}\}^A{}_{j=1}\right|=1$，$\left|\{v_{jk}\}^B{}_{j=1}\right|=0$（|.|表示集合元素个数)，则由约束条件 3)，该集合中的唯一元素成为策略点w_k指向的位置点当且仅当$w_k(1)\in B$。同理，$\left|\{v_{jk}\}^B{}_{j=1}\right|=1$且$w_k(1)\in A$，该集合中的唯一元素成为策略点$w_k$指向的位置点；同理，$\left|\{v_{jk}\}^B{}_{j=1}\right|=1$，$w_k(1)\in B$且$\{v_{jk}\}^A{}_{j=1}=\emptyset$，该集合中的唯一元素成为策略点$w_k$指向的位置点。判断是否属于 1)、2)、3)、4）中任意一种，并按照判断执行；

3）若$\left|\{v_{jk}\}^B{}_{j=1}\right|>1$，$w_k(1)\in A$，则选择到距离$w_k(1)$最短的 B 中点作为下一个策略点；同理若$\left|\{v_{jk}\}^A{}_{j=1}\right|>1$，$\{v_{jk}\}^B{}_{j=1}=0$，$w_k(1)\in B$，则选择到距离$w_k(1)$权最小的 A 中点作为下一个策略点；若$\left|\{v_{jk}\}^B{}_{j=1}\right|>1$，$w_k(1)\in B$，

$\{v_{jk}\}^{A}{}_{j=1} = \emptyset$，则选择到距离$w_k(1)$权最小的 B 中点作为下一个策略点；判断是否属于 1）、2）、3）、4）中任意一种，并按照判断执行；

4）若$\left|\{v_{jk}\}^{A}{}_{j=1}\right| = m \geq 2$，$\left|\{v_{jk}\}^{B}{}_{j=1}\right| = n \geq 2$，此时为避免对同一台负责第二道工序的机床上料和下料分开进行（也即，先将成品下料而无半成品上料，前往负责第一道工序的机床进行半成品下料后，再前往刚才未上料的负责第二道工序的半成品进行半成品的上料），因而此时按照先进行半成品下料、再进行成品下料的顺序进行。

设$\{v_{jk}\}^{A}{}_{j=1} \times \{v_{jk}\}^{B}{}_{j=1}$为两个集合之间满足约束条件的所有通路的集合，则$\cup(\{v_{jk}\}^{A}{}_{j=1} \times \{v_{jk}\}^{B}{}_{j=1}) = k_{m,n}$，即两部分顶点数分别为 m,n 的完全二部图。利用启发式算法，在默认规则以及约束条件下，结合权重矩阵计算得到，连通$\mathrm{r} = 2 * \min\{m,n\} + 1$个策略点且权值最小的路$\mathrm{R}_k$，并使该通路$\mathrm{R}_k$的第一个顶点$\mathrm{R}_k[1]$作为策略点$w_k$指向的位置点。

4.1）当在路R_k的执行过程当中有机床发出需求信号时：

若此时 RGV 处于策略点处且已经历R_k中r_0个策略点，则立刻进行下一次规划调度，判断情况是否属于 2）、3）中任意一种，结合权重矩阵计算得到，连通$\mathrm{r} = \mathrm{r} - r_0$个策略点且权值最小的路$R_{k+r_0}$，当在路$R_{k+r_0}$的执行过程当中有机床发出需求信号时，重复 4.1）直至在运行 R 的过程中机床无需求信号发出。此过程完成后，判断是否属于 1）、2）、3）、4）中任意一种，并按照判断执行；

若此时 RGV 处于策略点之间且已经历R_k中r_0个策略点，则 RGV 运行至R_k路上最相邻的下一策略点后进行下一次规划调度，判断此时情况是否属于 2）、3）中任意一种，结合权重矩阵计算得到，连通$\mathrm{r} = \mathrm{r} - r_0 - 1$个策略点且权值最小的路$R_{k+r_0+1}$，当在路$R_{k+r_0+1}$的执行过程当中有机床发出需求信号时，重复 4.1）直至在运行 R 的过程中机床无需求信号发出；此过程完成后，判断是否属于 1）、2）、3）、4）中任意一种，并按照判断执行；

注，各有向边$v(i,j)$的权重定义如下：

$$d[v(i,j)] = \mathrm{mov}(v_i, v_j) + T3(v(j)) + t_{iw} * \chi_B(i)$$

其中：χ_B为集合 B 的特征函数：$\chi_B(i) = \begin{cases} 1, & 若 i \epsilon B \\ 0, & 若 i \notin B \end{cases}$，

得到权重矩阵 D=$\{d_{ij}\}$=$\{d[v(i,j)]\}$[4]

5.2.3 模型的求解

参照动态调度模型过程，现以成品数量 n 为参数，以分别负责两道工序的机床数量、机床之间的距离矩阵为变量集 X，建立目标函数 f（X）。

$$f(X)=t_1n_1+t_2n_2+t^*n_3+t_{mov}$$

其中：t_1为完成第一道工序所需时间，t_2为完成第二道工序所需时间，t^*为完成一件成品所需上下料、清洗和等待的时间之和，t_{mov}为 RGV 在整个移动过程中的时间。n_1,n_2,n_3分别对应参与上述三种活动的工件数。

根据“使做第二道工序的 CNC 机器等待时间最小化”和“半成品冗余情况最小化”的原则，可知$\lim_{n\to\infty}\frac{n_2-n_1}{n_2}=0$，其中$\begin{cases}n_2\le n_1\\ n=\min\{n_1,n_2\}\end{cases}$,因此$n_1\sim n_2=n$；又由于$n_2\le n_3\le n_1$，故$n_3=$n。

由上述分析可知，f(X)由线性部分（t_1+t_2）n和随机部分$t^*n_3+t_{mov}$组成，其中$t^*n_3+t_{mov}$为关于 X 的函数，记为 g(X)。

下面分析随机部分的约束条件：

1)令 A 为从事第一道工序的 CNC 机器位置集合，B 为从事第二道工序的 CNC 机器位置集合。A=$\{a_i\}$,B=$\{b_i\}$,则$|A|+|B|=8$

2）$t^*n_3+t_{mov}=2\mathrm{n}*\|D.*P\|_1$

3）基于 1）2），令$\frac{i}{8-i},\frac{j}{8-j}$为$\frac{|A|}{|B|}$的可能值，其中$|i-j|=1$,则可估计i,j的取值如下：

$$\frac{i}{8-i}\le\frac{t_1+\|D.*P\|_1}{t_2+\|D.*P\|_1}\le\frac{j}{8-j}$$

其中$t_1+\|D.*P\|_1$，$t_2+\|D.*P\|_1$分别为 A、B 中 CNC 占据时间的期望估计值。

为使该式具有运用价值，设$\|D.*P\|_1=t_w+2(\bar{d}+\varepsilon_1)+2(\overline{t_m}+\varepsilon_2)$,则$\Delta\varepsilon=$

2（$\varepsilon_1+\varepsilon_2$）为微小扰动，$\bar{d}=d_2$，$\overline{t_m}=(t_{m1}+t_{m2})/2$,则

$$\frac{t_1+\|D.*P\|_1}{t_2+\|D.*P\|_1}\sim\frac{t_1+t_w+2d_2+(t_{m1}+t_{m2})}{t_2+t_w+2d_2+(t_{m1}+t_{m2})}$$

由此可大致确定i,j的取值。

概率矩阵 P 定义如下：

为计算t_{mov}的理论值，引入未知的概率矩阵 P=$\{p_{ij}\}$，其中p_{ij}为 RGV 到达有向边$v(i,j)$的概率。由辛钦大数定律，当 n 趋于无穷时该概率可由频率进行逼近，即

$$\lim_{n\to\infty}P\left\{\left|\frac{\mu_n}{n}-p\right|<\varepsilon\right\}=1$$

以上约束条件均为针对t_{mov}所给定的。

然后可给目标函数 f(X)赋值时间 h，并联系约束条件将不等式化为等式，由此建立以 X 为变量的方程组；此时若固定参数 n，则可获得欲在 h 时间内完成 n 件成品情况下分别负责两道工序的机床数量、机床之间的距离矩阵需满足的关系 s(X)；再调节参数 n，使 n 尽可能大的同时令关系 s(X)在题设情景中可达到，在临界处可求得变量 X 的值。

但由于t^*n_3部分与实际操作有密切联系，因此求解存在较大的困难。最终转而使用启发式算法—返程动态规划最优求解法建立模型并求解。

本模型求解与情形一相似，即采用局部动态规划和二叉树算法对 RGV 的策略点集进行优化。鉴于情形二中 A 与 B 集合之间的特殊约束条件，重新定义相关测度E，记为$\bar{E}$。

各策略点w_k的备选点集为$\{v_{jk}\}^A{}_{j=1}$和$\{v_{jk}\}^B{}_{j=1}$，遍历所有二部图路径所对应的测度$\bar{E}$，则

$$\bar{E}_{ik} = \sum_i d(v_{pk}, v_{p+1,k}),$$

其中$v_{pk} \epsilon \{v_{jk}\}^A{}_{j=1}, v_{p+1,k} \in \{v_{jk}\}^B{}_{j=1}$，或$v_{pk} \epsilon \{v_{jk}\}^B{}_{j=1}, v_{p+1,k} \in \{v_{jk}\}^A{}_{j=1}$。

根据上述约束条件，当$\{v_{jk}\}^A{}_{j=1} = \emptyset$ 或$\{v_{jk}\}^B{}_{j=1} = \emptyset$时累和终止，得到$\bar{E}_{ik}$值。遍历完毕后，得到$\min_i \bar{E}_{ik}$和该测度所对应的$\mathrm{R}_k$路。此时令$w_k(2) = \mathrm{R}_k[1]$，而后对策略点$w_{k+1}$进行操作，以此类推，直到 t>28800s 时终止循环。最终得到的策略点列$\{w_k\}$即最优解。

5.2.4 模型的检验

利用系统作业参数的 3 组数据进行检验，由此得到连续工作时间区间（28800s）内完成的成品件数以及 CNC 具体的调度情况（见材料 Case_2_result.xls）。

表 3 3 组数据下完成成品数比较

系统作业参数及完成成品数	第 1 组	第 2 组	第 3 组
RGV 移动 1 个单位所需时间	20	23	18
RGV 移动 2 个单位所需时间	33	41	32
RGV 移动 3 个单位所需时间	46	59	46
CNC 加工完成一个两道工序物料的第一道工序所需时间	400	280	455
CNC 加工完成一个两道工序物料的第二道工序所需时间	378	500	182
RGV 为 CNC1#，3#，5#，7#一次上下料所需时间	28	30	27
RGV 为 CNC2#，4#，6#，8#一次上下料所需时间	31	35	32
RGV 完成一个物料的清洗作业所需时间	25	30	25
完成成品数	**252**	**207**	**265**

由实验数据可知，第三组数据的完成成品数最多，第二组数据的完成成品数最少。从第一、二道工序的时间之和可见，成品数与工序时间呈正相关关系，并且第二组的上下料、清洗与移动时间均高于另外两组，因此所得实验结果可靠性较强。

下面考虑实验细节进行进一步分析。从 CNC 具体的调度情况可见，选择$|A| = |B|$=4 情况，CNC 的调度在时间模拟器远离零点的长时间内呈现某种周期性，周期值$T_1 = T_2 = 4$；而在初始时间内，任一周期值内的策略点集出现微调、波动的情况，而后趋于稳定状态。若$|A| \neq |B|$，此时 CNC 的调度仍可能存在周期性，但发生波动的可能性相比于$|A| = |B|$的情况有明显增加，此时对 RGV 的调度显得更加重要。以上分析说明，动态调度模型的稳定性与初始 CNC 机器安排具有密切关

系。

$$效率（\eta）=\frac{完成成品数（n）*两道工序加工时间（t）}{8*28800}\times 100\%$$

通过计算得出，在 3 组系统参数下，系统的作业效率（η）分别为 85.09%、70.08%、73.27%，作业效率较高，反映出了模型的高效性。但与只有一道工序的加工情况相比，效率偏低。同时发现，第 1 组系统作业参数中两道工序加工时间最相近，而第 1 组系统作业参数下系统作业效率亦是最高。这可能是因为两道工序必须按照顺序执行，使得对系统的限制增多，两道工序加工时间相差越大，越易出现 CNC 闲置的情况，使得系统的作业效率降低。

5.3 情形三

5.3.1 模型的建立

本模型在情况一、二模型的基础上，加入随机变量ξ_1，ξ_2和ξ_3，其中ξ_1服从伯努利分布，ξ_2和ξ_3服从均匀分布，以此模拟 CNC 加工过程中出现故障的情况，借助模型一、二及其算法实现，以此获得在故障存在时一道工序和两道工序的物料加工作业情况。

5.3.2 模型的求解

1．蒙特卡洛算法基本原理

本模型通过蒙特卡洛模拟方法获得伯努利分布和均匀分布中随机变量的取值。蒙特卡洛模拟方法又称作随机抽样或统计试验方法。由概率定义知，某事件的概率可以用大量试验中该事件发生的频率来估算，当样本容量足够大时，可以认为该事件的发生频率即为其概率。因此，可以先对影响其可靠度的随机变量进行大量的随机抽样，然后把这些抽样值一组一组地代入功能函数式，确定结构是否失效，最后从中求得结构的失效概率。蒙特卡罗法正是基于此思路进行分析的。

2. 随机变量的功能实现

随机变量在模拟过程中的具体作用如下：

若$\xi_1=1$，则对工件O_{ij}的完成工序所需要的时间进行修正，$\widetilde{T_1}(O_{ij})=\xi_2+\xi_3$，

以此作为约束条件$|T1(O_{k_1j})-T1（O_{k_2j})|<t^*$中$t^*$的值。此后继续沿用前两种情形的算法，以修正后的约束条件为基础进行策略点集$M\{w_i\}$的搜索选取。

3、随机变量的求解

本模型所需要模拟的随机故障情形中主要涉及三个随机变量：ξ_1，ξ_2，ξ_3。随机变量的概率分布如下：

1）$\mathrm{P}(\xi_1=1)=\frac{1}{100}$；$\mathrm{P}(\xi_1=0)=\frac{99}{100}$；$\xi_1$服从伯努利分布。

2）$\mathrm{F(t)}=\mathrm{P}(\xi_2\le \mathrm{t})=\begin{cases}0, & t<0\\ \frac{t}{T_1(O_{ij})}, & 0\le t\le T_1(O_{ij})\\ 1, & t\ge T_1(O_{ij})\end{cases}$；$\xi_2$服从均匀分布。

3）$\mathrm{G(t)}=\mathrm{P}(\xi_3\le t)=\begin{cases}0, & t<10\\ \frac{t-10}{10}, & 10\le t\le 20\\ 1, & t\ge 20\end{cases}$；$\xi_3$服从均匀分布。

其中：

1）ξ_1为二值随机变量，反映工件O_{ij}在 CNC 的生产过程中是否会发生故障。ξ_1的判断在 RGV 完成对该 CNC 上料工作的即刻进行。

2）若$\xi_1=1$，则ξ_2反映工件O_{ij}在加工过程中发生故障的时间点，ξ_3反映工件ξ_3需要的维修时间长度。假设这两个随机变量的分布都满足该区间上的均匀分布。

为求解ξ_1的值，利用随机数函数功能，定义 100 阶向量$\mathrm{p}=(1,0,0,\ldots,0)$,定义函数$Ch(p)$随机提取向量p中的元素，则$\xi_1=1\Leftrightarrow Ch(p)$=1。

ξ_2,ξ_3的求解遵循以下步骤：

1）根据各个随机变量的密度函数，在计算机上产生随机数，实现一次模拟过程所需的足够数量的随机数。

2）根据概率模型的特点和随机变量的分布特性，并对每个随机变量进行直接抽样。

3）按照所建立的模型进行仿真试验、计算，求出问题的随机解。

5.3.3 模型的检验

通过蒙特卡洛算法对故障情况进行模拟，得到如下结果：

表 4 单一工序 3 组数据有无故障情形对比

	无故障的情形	有故障的情形	故障工件数
第 1 组	375	359	5
第 2 组	354	345	2
第 3 组	381	381	1

实验表明，3 组实验中有故障情形相比无故障情形，成品工件数都发生了减少或不变，但是减少量与故障工件数并不成正相关的关系。例如第 1 组数据有无故障的情形前后差值为 9，而故障工件数仅仅为 2，但第 2 组数据前后差值与故障工件数恰好吻合。这说明 CNC 机器发生故障会对 RGV 的动态调度决策产生影响。深入观察动态调度过程细节可发现，当机器发生故障后，原本的调度周期很快被打破，在较多次决策后 RGV 的调度才趋于平稳。这说明 CNC 的故障对整体系统的影响是显著的。

表 5 两项工序 3 组数据有无故障情形对比

	无故障的情形	有故障的情形	故障工件数
第 1 组	252	246	1
第 2 组	207	206	3
第 3 组	265	261	1

实验表明，3 组实验中有故障情形相比无故障情形，成品工件数都发生了减少，然而在第二组数据中出现了故障工件数大于前后差值的情况，从某种意义上可以解释为 CNC 的故障帮助优化了某一些策略方案。进一步观察动态调度过程，二种工艺的动态调度过程显得比单一工艺流程更为复杂，发生故障后系统的恢复平衡能力也相比于上一种情形有所削弱。

六、模型的评价

6.1 模型的优点

1）本模型的静态调度部分针对单一工艺生产流程和两道工序生产流程给出了原理相似的基于动态规划模型，但也根据不同情形的特点进行了优化。例如，在单一工艺生产流程的模型中利用流程后效性较低的特点采用贪婪算法，将全局最优解的求解问题分解为各局部最优解的求解问题，简化了操作，提升了算法的效率；在两道工序生产流程中将 A、B 间的物料运输过程抽象为二部图$K_{m,n}$最小权问题，同样简化了操作，加快了最优解的收敛速度。

2）本模型的动态调度部分以机器模拟的方法，通过遍历随机数获取发生故障的$\{v_i\}$位置点集，并标记状态点O_{ij}和延长理论工作时间的方法，模拟机器修理、工件报废和返回作业序列的过程，使整个模拟过程具备较佳的可靠性。

3）在模型的深度优先搜索部分，可设置弹性参量q_1, q_2，其中q_1表示树每一顶点分叉数的上界，q_2表示深度搜索的层数。通过对这两个参量的调试，可根据实际需求获取不同的调度方案。

4）在两道工序生产流程中,为确定比值$|A|/|B|$，事先以加工时间、有向边权重进行范围估计，从而无需遍历所有在约束条件$|A| + |B| = 8$下的$|A|/|B|$全部情形，只需进行局部遍历，简化了遍历过程。

6.2 模型的缺点

1）在模型深度优先搜索部分中，若弹性参量q_1, q_2选取不当，可能导致计算量过大从而出现计算冗余的情况，降低系统效率。

2）在实际情况中,随机变量ξ_2, ξ_3的分布可能接近与其他的概率分布，例如正态分布、伽马分布等。模型中采用均匀分布可能会造成一定的系统误差，无法反映实际的机器故障情况。

3)在两道工序生产流程中未给出有关 A、B 集合中机器序号安排的可靠算法，

仅以遍历的方法，以目标函数f的最小值确定最优安排方法。

七、参考文献

[1] 刘燚，刘长安. 基于混合遗传算法的车间动态调度研究[J]. 组合机床与自动化加工技术，2017(3):158-160.
[2] 吴正佳，何海洋，黄灿超,等. 带机器故障的柔性作业车间动态调度[J]. 机械设计与研究，2015(3):94-98.
[3] 钟守楠，高成修. 运筹学理论基础[M]. 武汉大学出版社，2005.
[4] 姜存学，蔡力钢，胡于进. 基于约束矩阵与蚁群算法的 CAPP 加工工序优化排序[J]. 中国机械工程，2009(18):2203-2206.
[5] 廖强，周凯，张伯鹏. 基于现场总线的多 Agent 作业车间动态调度问题的研究[J]. 中国机械工程，2000，11(7):757-759.

附录

附录一：单一工艺情形算法实现（Python）

```
from enum import Enum
import random


class CarriedType(Enum):
    normal = 0
    first = 1
    second = 2


total_time = 8*60*60
total_cost = 0
now_min_cost = 1000000
now_complete = 0

input1 = [20,33,46,560,400,378,28,31,25]
input2 = [23,41,59,580,280,500,30,35,30]
input3 = [18,32,46,545,455,182,27,32,25]

machines = [[0, None] for i in range(8)]

machine_process_costs = []
put_fetch_costs = []
move_costs = []
global_put_down_cost = 0
global_abs = []
```

```
now_abs=[]
global_step_list = []
step_list = [-1 for i in range(400)]
min_step_list = []
min_begin_of_process = []
global_begin_of_process=[]
begin_of_process = [-1 for i in range(400)]

now_pos = 1
now_carried = None


def input_with(raw):
    global move_costs
    global machine_process_costs
    global put_fetch_costs
    global global_put_down_cost
    move_costs.append(0)
    move_costs.extend(raw[0:3])
    machine_process_costs = [raw[3] for i in range(8)]
    put_fetch_costs = [raw[6] if i % 2 == 0 else raw[7] for i in range(8)]
    global_put_down_cost = raw[8]


def is_abnormal():
    """
    :param x: 工件的发生时间
    :return: (是否故障,故障时间,故障时长)
    """
    ab = False
    ab_val = random.uniform(0,1)
    if ab_val>=0.99:
        ab = True
    return ab


def get_ab_info(x):
    bg_time = int(random.uniform(0, get_process_cost(x)))
    len_time = int(random.uniform(600, 1200))
    return bg_time, len_time


def make_decision_norm(n, pos, machines):
    possibles = [x[0] for x in list(zip(range(8),machines)) if x[1][0] == 0]
```

```
    s = sorted(possibles, key=lambda y: get_move_cost(y,pos)+get_put_fetch_cost(y)+get_process_cost(y))
    # print(s)
    return s[0:min(n,len(s))]


def get_move_cost(cur, next):
    return move_costs[abs(cur//2-next//2)]


def get_put_fetch_cost(next):
    return put_fetch_costs[next]


def get_process_cost(next):
    return machine_process_costs[next]


def predict_norm_steps(branch_num, steps, cur_step,
                       cur_cost, cur_pos, cur_machine,cur_abs,
                       cur_wash_box, make_decision=make_decision_norm):
    """
    :param branch_num: 准备探索的分支个数
    :param steps: 准备探索的步数
    :param cur_step: 当前位置,开始可以填 0
    :param cur_cost: 当前的时间花销,开始可以填 0
    :param cur_pos: RGV 开始的位置
    :param  cur_machine:  开始的机器状态[[加工剩余时间,现在正在加工的类型]....]. 开始可以填全局变量 machines
    :param cur_wash_box: 当前的冲洗槽内工件类型,开始可以填 None
    :param make_decision: 选取策略,默认选择离得最近的 branch_num 个进行探索
    :return:
    """
    if steps == cur_step:
        global now_min_cost
        global min_step_list
        global min_begin_of_process
        global machines
        global now_pos
        global now_carried
        global now_abs
        if cur_cost <= now_min_cost:
            now_min_cost = cur_cost
            min_step_list = step_list[0: steps]
```

```
            min_begin_of_process = begin_of_process[0: steps]
            machines = cur_machine
            now_pos = cur_pos
            now_carried = cur_wash_box
            now_abs = cur_abs
        return

    min_time = min([x[0] for x in cur_machine])
    if min_time > 0:
        cur_machine = [[x[0]-min_time, x[1]] for x in cur_machine]
        cur_cost += min_time
    next_poses = make_decision(branch_num, cur_pos, cur_machine)

    ab = is_abnormal()

    for next_pos in next_poses:
        # next_machines = cur_machine.copy()

        process_cost = get_process_cost(next_pos)
        move_cost = get_move_cost(cur_pos, next_pos)
        put_fetch_cost = get_put_fetch_cost(next_pos)
        put_down_cost = 0
        if cur_wash_box == CarriedType.normal and cur_machine[next_pos][1] is
not None:
            put_down_cost = global_put_down_cost
        cst = move_cost + put_fetch_cost + put_down_cost
        next_machines = [[max(solution[0] - cst, 0), solution[1]] for solution
in cur_machine]
        next_machines[next_pos][0] = process_cost - put_down_cost
        next_machines[next_pos][1] = CarriedType.normal

        next_abs = cur_abs.copy()
        # if ab:
        #     bg_time, len_time = get_ab_info(next_pos)
        #     next_machines[next_pos][0] = bg_time + len_time - put_down_cost
        #     next_machines[next_pos][1] = None
        #     next_abs.append([next_pos+1, bg_time+cur_cost+cst-put_down_cost,
        #                      bg_time+cur_cost+cst-put_down_cost+len_time])

        wash_box = next_machines[next_pos][1]
        if cur_wash_box is not None and cur_machine[next_pos][1] is None:
            wash_box = CarriedType.normal

        step_list[cur_step] = next_pos
```

```
        begin_of_process[cur_step] = cst-put_down_cost+cur_cost
        predict_norm_steps(branch_num, steps, cur_step+1,
                        cst+cur_cost, next_pos, next_machines,next_abs,
                        wash_box)
    return


if __name__ == '__main__':
    input_with(input2)

    for i in range(20):
        predict_norm_steps(2, 25, 0,
                           total_cost, now_pos, machines,[],
                           now_carried)
        print("===============")
        print(now_abs)
        total_cost = now_min_cost
        now_min_cost = 1000000000
        global_step_list.extend(min_step_list)
        global_begin_of_process.extend(min_begin_of_process)
        global_abs.extend(now_abs)

    min_step_list = global_step_list
    min_begin_of_process =global_begin_of_process

    rst = []
    bgs = [x-get_put_fetch_cost(min_step_list[i])
           for    x,    i    in    list(zip(min_begin_of_process,    range(0,
len(min_begin_of_process))))]

    for x,i in zip(min_step_list, range(0,len(min_step_list))):
        rv = []
        if i != 0:
            rv = min_step_list[0:i][::-1]
        # print(rv)
        n = -1
        if x in rv:
            n = rv.index(x)

        if n != -1:
            rst[i-n-1][1] = bgs[i]
        rst.append([bgs[i],-1])
    with open('norm2','w+') as f:
        for i in range(len(rst)):
```

```
            f.write(str(min_step_list[i]+1)+"\t")
            f.write(str(rst[i][0]))
            f.write('\t')
            f.write(str(rst[i][1]))
            f.write('\n')
        f.flush()
    # with open('norm3ab', 'w+') as f:
    #     for i in range(len(global_abs)):
    #         f.write(str(global_abs[i][0]) + "\t")
    #         f.write(str(global_abs[i][1]) + "\t")
    #         f.write(str(global_abs[i][2]) + "\t")
    #         f.write('\n')
    #     f.flush()
```

附录二：双工艺情形算法实现（Python）

```
from enum import Enum
from collections import namedtuple
import random

amachines = [0,1,3,4,7]


class MachineType(Enum):
    a = 1
    b = 2


class CarriedType(Enum):
    normal = 0
    first = 1
    second = 2


total_time = 8*60*60
total_cost = 0
now_min_cost = 1000000
now_complete = 0

input1 = [20,33,46,560,400,378,28,31,25]
input2 = [23,41,59,580,280,500,30,35,30]
input3 = [18,32,46,545,455,182,27,32,25]

Machine = namedtuple('Machine', ('type', 'time', 'now'))
```

```
machines = [i for i in range(8)]

machine_process_costs = []
put_fetch_costs = []
move_costs = []
global_put_down_cost = 0

global_abs = []
now_abs = []
global_begin_of_process = []
global_step_list = []
step_list = [-1 for i in range(800)]
min_step_list = []
min_begin_of_process = []
begin_of_process = [-1 for i in range(800)]

now_pos = 0
now_carried = None
now_carried_first= False


def input_with(raw,x):
    global move_costs
    global machine_process_costs
    global put_fetch_costs
    global global_put_down_cost
    global machines
    move_costs.append(0)
    move_costs.extend(raw[0:3])
    machine_process_costs = [raw[4] if i in x else raw[5] for i in range(8)]
    put_fetch_costs = [raw[6] if i % 2 == 0 else raw[7] for i in range(8)]
    global_put_down_cost = raw[8]
    machines = [Machine(MachineType.a, 0, None)
                if i in x else Machine(MachineType.b, 0, None) for i in range(8)]


# def is_abnormal(x):
#     """
#     :param x: 工件的发生时间
#     :return: (是否故障,故障时间,故障时长)
#     """
#     random.uniform(1,100)
#     pass
```

```
def make_decision_norm(n, pos, machines, carried_first):
    possibles = []
    if carried_first:
        possibles = [x[0] for x in list(zip(range(8),machines))
                     if x[1].time == 0 and x[1].type==MachineType.b]
    else:
        # print(list(zip(range(8),machines))[0])
        possibles = [x[0] for x in list(zip(range(8),machines))
                     if x[1].time == 0 and (x[1].type == MachineType.a or
                        (x[1].type == MachineType.b and x[1].now is not None))]
    s = sorted(possibles,
               key=lambda y: cost_esimate(pos,y))
    # print(s)
    return s[0:n]


def cost_esimate(pos, y):
    if y in amachines:
        return get_move_cost(pos, y) + get_put_fetch_cost(y)
    else:
        return 0


def get_move_cost(cur, next):
    return move_costs[abs(cur//2-next//2)]


def get_put_fetch_cost(next):
    return put_fetch_costs[next]


def get_process_cost(next):
    return machine_process_costs[next]


def is_abnormal():
    """
    :param x: 工件的发生时间
    :return: (是否故障,故障时间,故障时长)
    """
    ab = False
    ab_val = random.uniform(0,1)
    if ab_val>=0.99:
```

```
        ab = True
    return ab


def get_ab_info(x):
    bg_time = int(random.uniform(0, get_process_cost(x)))
    len_time = int(random.uniform(600, 1200))
    return bg_time, len_time


def predict_two_steps(branch_num, steps, cur_step,
                      cur_cost, cur_pos, cur_machine, cur_carried_first,
                      cur_wash_box, cur_abs, make_decision=make_decision_norm):
    """
    :param branch_num: 准备探索的分支个数
    :param steps: 准备探索的步数
    :param cur_step: 当前位置,开始可以填 0
    :param cur_cost: 当前的时间花销,开始可以填 0
    :param cur_pos: RGV 开始的位置
    :param  cur_machine:  开始的机器状态[[加工剩余时间,现在正在加工的类型]....].
开始可以填全局变量 machines
    :param cur_wash_box: 当前的冲洗槽内工件类型,开始可以填 None
    :param make_decision: 选取策略,默认选择离得最近的 branch_num 个进行探索
    :return:
    """
    if steps == cur_step:
        global now_min_cost
        global min_step_list
        global min_begin_of_process
        global machines
        global now_carried
        global now_carried_first
        global now_abs
        if cur_cost <= now_min_cost:
            now_min_cost = cur_cost
            min_step_list = step_list[0: steps]
            min_begin_of_process = begin_of_process[0: steps]
            machines = cur_machine
            now_carried = cur_wash_box
            now_carried_first = cur_carried_first
            now_abs = cur_abs
        return

    if cur_carried_first is True:
```

```
        min_time = min([x.time for x in cur_machine if x.type == MachineType.b])
        if min_time > 0:
            cur_machine = [Machine(x.type, max(x.time-min_time,0), x.now) for x
in cur_machine]
            cur_cost += min_time
    else:
        min_time = min([x.time for x in cur_machine
                        if x.type == MachineType.a or
                        (x.type == MachineType.b and x.now is not None)])
        if min_time>0:
            cur_machine = [Machine(y.type, max(y.time-min_time,0), y.now) for y
in cur_machine]
            cur_cost += min_time

    next_poses     =     make_decision(branch_num,     cur_pos,     cur_machine,
cur_carried_first)

    ab = is_abnormal()

    for next_pos in next_poses:
        # next_machines = cur_machine.copy()

        process_cost = get_process_cost(next_pos)
        move_cost = get_move_cost(cur_pos, next_pos)
        put_fetch_cost = get_put_fetch_cost(next_pos)
        put_down_cost = 0
        if cur_wash_box == CarriedType.second \
            and cur_machine[next_pos].type==MachineType.b \
            and cur_machine[next_pos].now is not None:
            put_down_cost = global_put_down_cost
        cst = move_cost + put_fetch_cost + put_down_cost
        next_machines   =   [Machine(solution.type,   max(solution.time-cst,0),
solution.now)
                             for solution in cur_machine]
        # next_machines[next_pos].time = process_cost-put_down_cost
        if next_machines[next_pos] == MachineType.b:
            next_machines[next_pos]   =   Machine(next_machines[next_pos].type,
process_cost - put_down_cost,
                                                  CarriedType.second)
        else:
            next_machines[next_pos]   =   Machine(next_machines[next_pos].type,
process_cost - put_down_cost,
                                                  CarriedType.first)
```

```
        ########################################
        wash_box = cur_wash_box
        carried_first = False
        if      cur_machine[next_pos].type      ==      MachineType.a      and
cur_machine[next_pos].now is not None:
            carried_first = True
        if cur_machine[next_pos].type == MachineType.b:
            wash_box = cur_machine[next_pos].now
        ########################################

        next_abs = cur_abs.copy()
        if ab:
            bg_time, len_time = get_ab_info(next_pos)
            next_machines[next_pos] = Machine(next_machines[next_pos].type,
                                              bg_time + len_time - put_down_cost,
None)
            next_abs.append([next_pos+1, bg_time+cur_cost+cst-put_down_cost,
                             bg_time+cur_cost+cst-put_down_cost+len_time])

        step_list[cur_step] = next_pos
        begin_of_process[cur_step] = cst-put_down_cost+cur_cost
        predict_two_steps(branch_num, steps, cur_step+1,
                          cst+cur_cost, next_pos, next_machines,
                          carried_first,wash_box, next_abs)
    return


def init_machines(x):
    global machines
    machines    =    [Machine(MachineType.a,0,None)    if    i    in    x    else
Machine(MachineType.b,0,None) for i in range(8)]


def find_last_pos(lst, i):
    x = lst[i]
    rv = []
    if i != 0:
        rv = lst[0:i][::-1]
    print(rv)
    n = -1
    if x in rv:
        n = rv.index(x)+2
        n = len(lst)-n
```

```
    return n


def find_next(lst, i):
    if i>=len(lst):
        return -1
    s = lst[i+1:]
    if lst[i] in s:
        return s.index(lst[i])+i+1
    return -1


if __name__ == '__main__':

    input_with(input3, amachines)
    # predict_two_steps(2, 10, 0,
    #                   0, 1, machines,False,
    #                   None)
    # print(min_step_list)
    # print(min_begin_of_process)

    for i in range(32):
        predict_two_steps(2, 30, 0,
                          total_cost, now_pos, machines,now_carried_first,
                          now_carried, [])
        print("===============")
        total_cost = now_min_cost
        now_min_cost = 1000000000
        global_step_list.extend(min_step_list)
        global_begin_of_process.extend(min_begin_of_process)
        global_abs.extend(now_abs)

    min_step_list = global_step_list
    min_begin_of_process = global_begin_of_process

    rst = []
    ids = []
    bgs = [x - get_put_fetch_cost(min_step_list[i])
           for x, i in list(zip(min_begin_of_process, range(0,
len(min_begin_of_process))))]

    for x, i in zip(min_step_list, range(0, len(min_step_list))):
        if x in amachines:
            next_a = find_next(min_step_list, i)
```

```
            next_bb = find_next(min_step_list, next_a+1)
            next_bb_val = -1 if next_bb == -1 else bgs[next_bb]
            next_a_val = -1 if next_a+1>=len(bgs) or next_a==-1 else bgs[next_a+1]
            rst.append([x+1,                bgs[i],                bgs[next_a],
min_step_list[next_bb]+1,next_a_val, next_bb_val])

    with open('two3_ab', 'w+') as f:
        for i in range(len(rst)):
            f.write(str(rst[i][0])+"\t")
            f.write(str(rst[i][1])+"\t")
            f.write(str(rst[i][2]) + "\t")
            f.write(str(rst[i][3]) + "\t")
            f.write(str(rst[i][4]) + "\t")
            f.write(str(rst[i][5]) + "\t")
            f.write('\n')
        f.flush()
    with open('two3ab', 'w+') as f:
        for i in range(len(global_abs)):
            f.write(str(global_abs[i][0]) + "\t")
            f.write(str(global_abs[i][1]) + "\t")
            f.write(str(global_abs[i][2]) + "\t")
            f.write('\n')
        f.flush()
```