# 智能 RGV 的动态调度系统方案

## 1 摘要

为求解 RGV 最优动态调度模型和系统最大作业效率，本文建立了关于智能 RGV 系统的数学模型，并通过系统作业参数的三组数据检验了模型的实用性和算法的有效性。

本文分别给出**一道工序物料**和**两道工序物料**的解法。具体地：

1. 一道工序：

- 将智能 RGV 系统问题抽象为 **TSP 模型**，列出目标函数与约束方程；
- 考虑有故障周期与无故障周期两种情况，并分别求得最优动态调度路线；
- 带入系统作业参数，检验模型的实用性，求解得到各自最大作业效率。

2. 两道工序：

(1) 分别从产品的产出效益和时间的利用率两个角度定义两道工序的动态系统效益；

- 基于产出效益的目的，我们使用**深度优先搜索算法**，将 RGV 的调度系统简化为 “背包问题”，在 8 小时内尽可能多地历遍所有的决策可能，即为**深度搜索**；通过每一个决策枝的可能的最大产出效益，及时对深度搜索过程进行合理剪枝，探索出产出效益驱动下的最优解，并求得该调度系统的作业效率；

- 基于时间利用率的目的，我们使用**最短寻道时间优先算法**，优先考虑 RGV 在每一步决策时的局部最优，在具有多个可选择的决策时，它移动到需要时间最短的一个 CNC，从而通过控制整个系统的最终时间利用效率最高，来制定出 RGV 的移动路线，训练总产出效益削弱系数后得出最优解，从而求得该情况下调度系统的作业效率。

(2) 基于模型的建立和求解，我们提出了若干可能对模型有优化的措施，并且将其他可能会影响企业效益决策的因素，例如能源消耗等，纳入最优决策的考量，便于本文提出的模型落实到具体的工厂时，都可以制定出个性化的最优 RGV 调度策略。

# 2 问题重述

有轨制导车辆-RGV(Rail Guided Vehicle)，又叫有轨穿梭小车。在工业化不断提升的当下，RGV 小车可用于各类高密度储存方式的仓库，可以避免传统生产流水线较长或是基于电机带动轮、链轮传输而导致的效率低下、容易损坏的问题。题中的 RGV 物流搬运车系统，通过一台 RGV，将物料运送到 8 台 CNC 工作台加工。并且根据加工物料的不同，可以分为一道工序和两道工序情况。

首先，我们对题干中可能争议的概念，作如下解读：

1. 一次上下料时间：RGV 可以同时完成上下料作业，也可单独完成上料或下料作业，作业时间均等于一次上下料时间。

2. 故障发生率 1%：当 CNC 正在加工某个物料时，它有 1% 的机率发生故障。

3. RGV“口袋假说”：完成第一道工序的半成品，可以被储存在 RGV 的 “口袋” 中，随 RGV 一起移动，不占用机械手爪。

为求解 RGV 最优动态调度模型和系统最大作业效率，本文我们将设定一些合理目标函数，将一道工序和二道工序加工情况抽象为数学问题，通过 TSP 模型、最短寻道时间优先算法、深度优先搜索算法得到最优调度模型并计算出系统最大的作业效率。

# 3 一道工序的模型建立与求解——旅行售货商（TSP）模型

问题一强调一道工序的 CNC 在 8 小时内的作业效率。而在时间一定的情况下，作业效率即正比于一班次内完成的物料量。若将整个班次中每台 CNC 均完成一次物料定义为一个循环，即 8 台 CNC 均各自完成一个物料的加工，则在每个循环内完成物料数量一定的情况下，最短的周期循环时间，则意味着最多的循环次数和最多的物料最终完成数量。

由此，问题就转化为在一周期内 RGV 到访的 CNC 数量、位置一定时，如何安排路线使花费的总时间最少。由此，我们可以把该问题转化为 TSP 模型，利用混合整数规划的思想求解最优。

## 3.1 符号说明

| 符号 | 含义说明 |
|---|---|
| $T$ | 完成一周期所需的时间 |
| $x_{ij}$ | $x_{ij}$ 表示从路线 i 到 j,$x_{ij}=0$ 则表示不走 i-j 路线 |

| 符号 | 含义说明 |
|---|---|
| $dist$ | 表示从 1 台 CNC 到下一台 CNC 时间的矩阵 |
| $p_t$ | 故障周期出现的概率 |
| $n$ | 共有 n 台所需要到达的 CNC |
| $u_i$ | 表示到达 CNC 次数。起点处 $u_i$=0，每到达一台 CNC，$u_i$ 加 1 |

以完成一个循环的时间：

$$T = \sum_{i=1}^{n}\sum_{j=1}^{n} dist_{ij} * x_{ij}$$

最小为目标建立一个规划模型。

## 3.2 概念界定

**混合周期：** 8 台 CNC 系统从无故障周期到故障周期直至恢复无故障周期这整个周期循环被称作混合周期。

## 3.3 目标函数

$$Min(T) = \sum_{i=1}^{n}\sum_{j=1}^{n} dist_{ij} * x_{ij}$$

## 3.4 约束条件

1. RGV 每次停止都到达一台 CNC，约束方程：

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad j = 1, 2, ..., n, j \neq i$$

2. RGV 最终要到达目标点，约束方程：

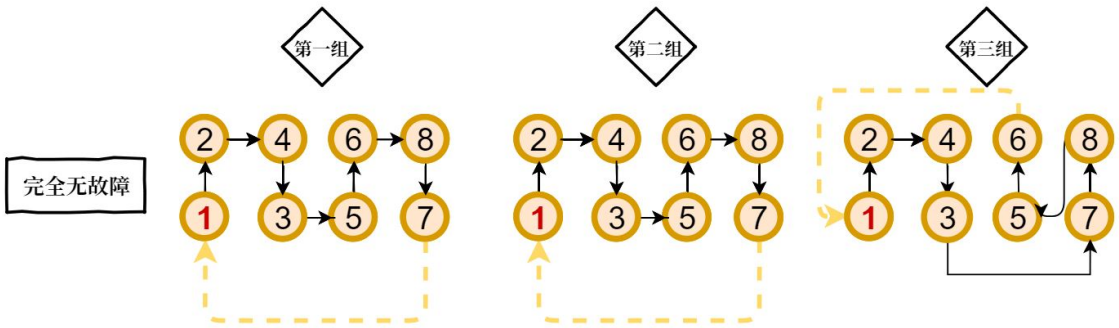
$$\sum_{i=1}^{n} x_{ij} = 1 \qquad i = 1, 2, ..., n, i \neq j$$

3. 定义 x 为 0-1 变量，约束方程：

$$x_{ij} = 0, 1 \qquad i, j = 1, 2, ..., n$$

4. RGV 的路线不含子巡回，约束方程：

$$u_i - u_j + n * x(I, J) \leq n - 1 \qquad u_(i), u_(j) \geq 0, i = 1, 2, .., n, j = 2, 3, ..n, i \neq j$$




注：1. 红色字体的编号表示一个循环的起点CNC。

图 1: 模型无故障对应的完整一周期路线

应用上述模型并且构造 dist 矩阵：

$$dist = \begin{bmatrix} x_{11} & \cdots & x_{1j} \\ \vdots & \ddots & \vdots \\ x_{i1} & \cdots & x_{ij} \end{bmatrix}$$

由此 TSP 模型建立完成。

## 3.5 基于数据对模型的求解

### 3.5.1 基于模型无故障完整一周期的求解

根据上述建模，分别带入第一、二、三组数据，通过 Lingo 求解得到各自最优路径和单周期最短时间：$T_1 = 342s$，$T_2 = 388s$，$T_3 = 336s$。各自对应的最优路线见图 1。

### 3.5.2 基于模型有故障完整一周期的求解

探讨有故障完整一周期的最优时间。

#### 3.5.2.1 假设一

已知每台 CNC 发生故障的概率为 1%，故一个周期为故障周期的概率为：$p_t = 1 - 0.99^8 = 0.077$。

只要出现 1 台 CNC 的损坏，即为故障周期。但每台 CNC 发生故障的概率为 1%, 两台机器在一周期内均发生故障的概率为 1/10000，量级过小，在最终求解故障周期平均发生故障的时间期望时，权重过小，不会对结果产生过大影响，可以忽略此种情况。至于三台乃

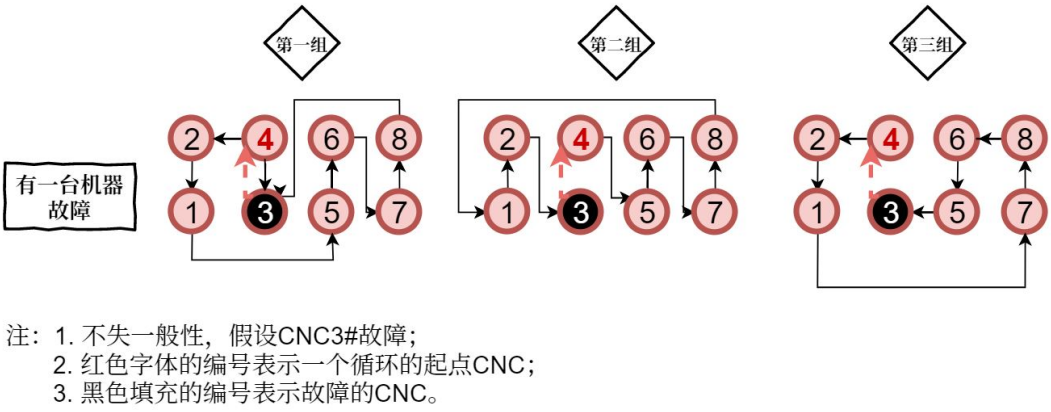



图 2: 模型无故障对应的完整一周期路线

至以上多台机器同时故障的概率更小，将会导致更小的权重，因此在此主要考虑 1 台 CNC 发生故障时对故障周期最优事件的影响。此时故障周期发生的概率为 1%。

#### 3.5.2.2 假设二

在一道工序的情况下，由于 RGV 在单周期内最优路径皆可等价按照 CNC 编号由小到大依次前行进行作业的路径，故无论哪台 CNC 发生故障，其情况等价。在此，假设编号为 3 的 CNC 机器发生故障。

根据上述假设和模型，分别带入第一、二、三组数据，通过 Lingo 求解得到各自最优路径和单周期最短时间: $T_{*1} = 314$, $T_{*2} = 358$, $T_{*3} = 309$。各自对应的最优路线见图 2。

### 3.5.3 混合周期下工作效率的求解

#### 3.5.3.1 以第 1 组数据为例:

第一组数据在没有机器发生故障的情况下, 优化模型下运行一个完整周期最优路径花费的时间为：$25 * 8 + 342 = 542s$。在此时，RGV 机器人完成 8 台 CNC 物料的放置并且根据最优路径的设置返回编号为 1 的 CNC。此时编号为 3 的 CNC 发生故障，RGV 按照故障周期最优路线依次前行：

此时编号为 1 的 CNC 加工情况：$560 - 342 + 28 = 246s$。因此，若要得到加工完毕的物料并且下料需要再等待 10 秒。

同理按编号顺序依次分析编号**第 2 至第 8 号 CNC** 需要等待的时间为:6s,37s,0s,0s,9s,0s

在取完物料后 RGV 机器人依旧从未故障的 1、2 号 CNC 开始放置新的物料。从 3 号机故障到以上整个过程花费时间为：$342 + 25 * 7 - 59 - 28 + 246 + 6 + 9 + 37 = 728s$。从 RGV 离开故障 3 号机到 RGV 机器人再次到 3 号机前时间为：$728 + 20 + 28 + 31 = 807s$。

机器故障的时间为 600 秒至 1200 秒，故此时 RGV 一定不能在 3 号机上上下料，将直接从 4 号机继续新一次的有故障周期循环。

其过程如上文推导，并根据优化模型计算在这一个故障周期里，当 RGV 跟随着新的最优路径从离开 3 号机到再次回到 3 号机时候花费时间为 522 秒，因此从第一次故障到第二次回到 3 号机经过了：

$$314 + 807 = 1121s$$

此时 3 号故障机器上必有物料存在，下一个周期仍可视为有故障周期循环

易分析完成一个周期上下料最优化时间期望为

$$E(T) = 738.5s$$

在 8 小时工作时间内，可完成大约 312 件产品。

#### 3.5.3.2 以第 2 组数据为例：

以第二组数据为例: 在没有机器发生故障的情况下, 优化模型下运行一个完整的 8 周期最优路径花费的时间为：$30 * 8 + 388 = 628s$。在此时，RGV 机器人完成 8 台 CNC 物料的放置并且根据最优路径的设置返回编号为 1 的 CNC，并且按照故障周期下的最优路径前行。

通过分析，此时编号为 1 的 CNC 加工时间：$628 - 30 = 598s$。此时早已经加工好，RGV 不用等待。同理，分析编号 2 到 8 号的正常 CNC 需要等待的时间。

易有，除 2 号需等待 227 秒、4 号 CNC 需等待 40 秒外，其余正常作业的 CNC 均不需要等待。在取完物料后 RGV 机器人依旧从未故障的 1、2 号 CNC 开始放置新的物料。

从 3 号机故障到以上整个过程花费时间为：$648 + 30 * 7 - 65 - 30 + 7 = 770s$。从 RGV 离开故障 3 号机到 RGV 机器人再次到 3 号机前时间为：$770 + 23 + 30 + 35 = 858s$。

机器故障的时间为 600 秒至 1200 秒，故此时 RGV 一定不能在 3 号机上上下料，将直接从 4 号机继续新一次的有故障周期循环。

其过程如上文推导，并根据优化模型计算在这一个故障周期里，当 R 跟随着新的最优路径从离开 3 号机到再次回到 3 号机时候花费时间为 522 秒，因此从第一次故障到第二次回到 3 号机经过了 $358 + 858 = 1216s$。

此时若机器故障时间只要在范围 $(600, 656)$ 之间，下一个周期为无故障周期循环，概率为 0.093. 范围 $(656, 1200)$ 之间，下一个周期为故障周期循环，概率 0.907.

根据上述求解得到：完成一个混合周期上下料最优化时间期望为：

$$E(T) = 1639.9s$$

在 8 小时工作时间内，可完成大约 281 件产品。

#### 3.5.3.3 以第 3 组数据为例：

以第三组数据为例: 在没有机器发生故障的情况下, 优化模型下运行一个完整的 8 周期最优路径花费的时间为：$25 * 8 + 336 = 536s$。

在此时，RGV 机器人完成 8 台 CNC 物料的放置并且根据最优路径的设置返回编号为 1 的 CNC，并且按照故障周期下的最优路径前行。此时编号为 1 的 CNC 加工情况：$545 - 336 + 27 = 236s$。因此，可以得到加工完毕的物料需要等待 236 秒。

同理，依次分析编号第 2 至第 8 号 CNC 物料加工情况：除 1 号机、4 号机、7 号机分别需要等待 263 秒、19 秒、6 秒，其余正常机器不用等待。在取完物料后 RGV 机器人依旧从未故障的 1、2 号 CNC 开始放置新的物料。从 3 号机故障到以上整个过程花费时间为：$336 + 25 * 7 - 59 - 27 + 236 + 19 + 6 = 713s$。

从 RGV 离开故障 3 号机到 RGV 机器人再次到 3 号机前时间为：$713 + 18 + 27 + 32 = 790s$。

机器故障的时间为 600 秒至 1200 秒，故此时 RGV 一定不能在 3 号机上上下料，将直接从 4 号机继续新一次的有故障 8 周期循环。

其过程如上文推导，并根据优化模型计算在这一个故障周期里，当 RGV 跟随着新的最优路径从离开 3 号机到再次回到 3 号机时候花费时间为 490 秒，因此从第一次故障到第二次回到 3 号机经过了 $309 + 790 = 1099s$。

若下周期仍旧为有故障周期循环：

根据上述求解得到：完成一个周期上下料最优化时间期望为

$$E(T) = 738s$$

在 8 小时工作时间内，可完成大约 312 件产品。

# 4 两道工序的模型建立与求解——深度优先搜索

## 4.1 问题描述

我们将加工第一道工序的 CNC 定义为“第一区域” 的 CNC,将加工第二道工序的 CNC 定义为 “第二区域” 的 CNC。某个物料被第一区域的 CNC 加工完成后，即发出 “完成信号”，等待 RGV 来进行下料和运输，目的地为第二区域的某台 CNC。在第二区域的 CNC 加工完成后再次发出完成信号，等待 RGV 来下料并且洗料。如果某台 CNC 是空闲状态，即发出 “需求信号”，等待 RGV 来上料。此问题的目标是在满足目标函数要求下达到尽可能有效的调度。由于在一个班次内规定为 8 小时，RGV 工作的最长时间是一定的，所以我们将两道工序的问题转化为可以量化计算的如下问题。

## 4.2 符号说明（适用于两道工序的两种方法）

| 符号 | 含义说明 |
|---|---|
| $k$ | 在加工台 CNC 上的第 k 个物料编号 |
| $product$ | 完成品的个数 |
| $product_i$ | 第 i 种调度方式下的完成品个数 |
| $half-product$ | 半成品的个数 |
| $half-product_i$ | 第 i 种调度方式下的半成品个数 |
| $A$ | 归一化处理后的完成品个数 |
| $B$ | 归一化处理后的半成品个数 |
| $load_i$ | 在第 i 台 CNC 上，RGV 进行上下料的时间 = CNC 进行上下料的时间 |
| $t_{wash}$ | RGV 进行洗料的时间 |
| $t_{work}$ | 物料被加工的时间 |
| $t_{w1}$ | 第一道工序的加工时间 |
| $t_{w2}$ | 第二道工序的加工时间 |
| $\alpha$ | 总产出效益削弱系数。由于半成品的数量在一定程度上影响了全部产品的产出效益，因此加入一个系数 $\alpha$，体现产出成果的总效益，$\alpha$ 的值由训练得出。 |
| $d_{ij}$ | 距离矩阵 |
| $M_{ij}$ | 距离矩阵 |
| $m$ | 第一区域的 CNC 数量 |
| $n$ | 第二区域的 CNC 数量 |
| $C_{1i}$ | 第一区域的编号为 i 的 CNC，i = 1,2,…,m |
| $C_{2j}$ | 第二区域的编号为 j 的 CNC，j = 1,2,…,n |
| $S_{1i}$ | 第 i 台 CNC 的状态，取值为 0,1,2，0 表示空闲，1 表示正在加工，2 表示加工完成在等待 |
| $S_{2j}$ | 同上 |
| $O_i$ | 描述 CNC 是否发出信号，$O_i$ 为 0 表示发出信号，1 表示不发出 (i=1,…,8) |
| $nt_i$ | 第 i 台 CNC 下次发信号的时间 |
| $des_k$ | 第 k 次移动到的 CNC 编号 |
| $dt_k$ | 第 k 次到 k+1 次移动中路上花费的时间 |
| $t_k$ | 第 k 到 k+1 次移动之前等待的时间 |

| 符号 | 含义说明 |
|---|---|
| $tp_i$ | CNC 的类型描述，取值为 1 表示第一区域的 CNC，取值为 2 表示第二区域的 CNC |

## 4.3 对故障的特别说明（适用于两道工序的两种方法）

两个问题中对发生故障做不同处理。在两道工序的问题中，以物料每一次被放上加工台 CNC 为一个单位，即**一次上料**。考虑所有第一、第二区域的 CNC，如果在 8 小时中所有 CNC 共有 `m` 次上料，则其中平均发生 `0.01m` 次故障，出现的机器可能是当前没有故障的 CNC 中正在加工物料的任意一台。

## 4.4 算法介绍

**示例问题：背包问题（示意图见图 3）**

| i | v | w |
|---|---|---|
| 1 | 45 | 5 |
| 2 | 48 | 8 |
| 3 | 35 | 3 |

假设在一个选择背包的问题中，i 列表示 id，v 列表示背包中物品的价值 value，w 表示背包的重量重量 weight，一个人选择背包的最大重量是 10w，如何在限制背包重量的情况下选择最大价值的背包。根据 value 和 weight 可以计算处每个背包单位重量的价值，记为 $v_i/w_i$。在此示例表格下，三个单位价值分别是：$v_1/w_1 = 9$, $v_2/w_2 = 6$, $v_3/w_3 = 11.7$。

符号说明如下：

| 符号 | 含义说明 |
|---|---|
| estimate | 可能的最大价值 |
| value | 已经得到的价值 |
| room | 剩余的空间 |

在一个限制总重量取得最大总价值的问题中，起点处可能的最大价值 $estimate = 1 * 9 + 1 * 11.7 + 0.25 * 6 = 92$，即在 $value = 0$ 时的状态为：

$$value = 0, room = 10, estimate = 92$$

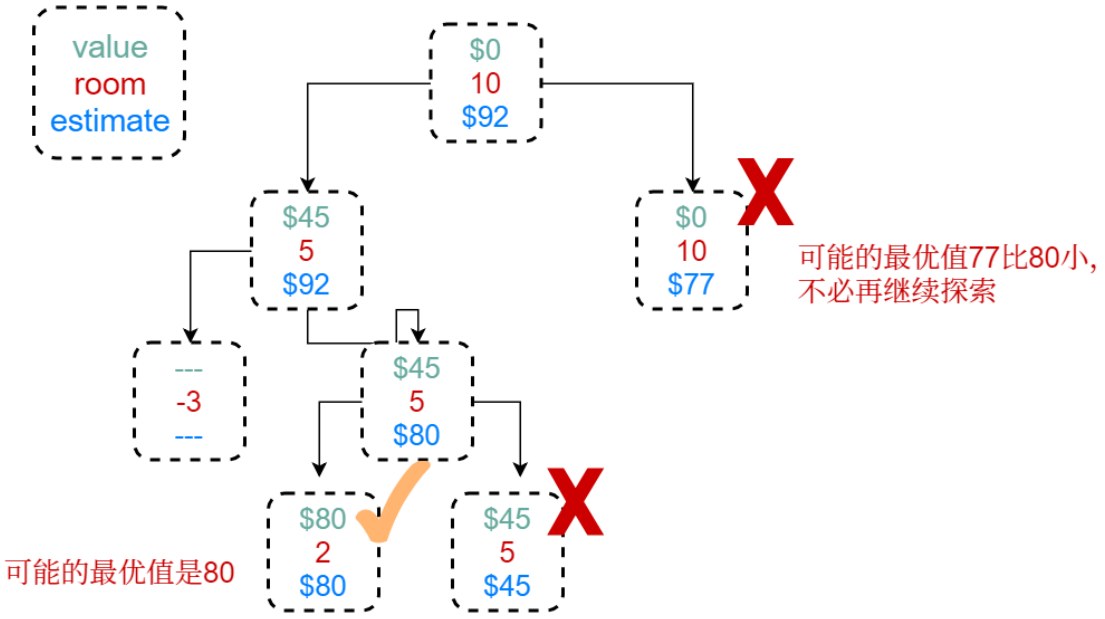



图 3: 深度优先搜索背包问题示意图

在面临多个选择时的决策图如下所示：

**算法应用**

由于 Depth-First 算法的应用背景是面临多个决策方案，在 RGV 调度系统中，只有当调度系统高速运作的状态下，RGV 才面临在多个 CNC 目标中做决策的问题。**调度系统高速运作**表示为：

$$S_{1i} \neq 0 (i = 1, 2, ..., m)$$

当调度系统进入高速运作状态之后，满足此算法应用的条件。

**搜索环节说明**

如果当前决策枝的下一个节点可能的最大价值 (*estimate*) 大于或等于当前决策之前的任何一次已经实现的最大价值，则继续沿着这个节点向更深搜索下一个节点，进行同样的判断。

**剪枝环节说明**

在“背包问题”中，被剪枝的决策可能的最优解是小于在这一决策之前已经实现的最优解的，因此关键在于**得出某一决策可能的最优解的方法**，即如何估计这一决策下，从当前时刻开始到限制时刻截至所能产生的最优解。

在智能 RGV 调度策略规划中，选择可能的最大完成品数量 (*product*) 的方法是：

在某一时刻的截面状态，假设 8 台 CNC 都在高速运转，并且有 4 台在加工第一道工序，另外 4 台加工第二道工序，并且一二两道工序的 CNC 可以紧密对接，即一个完成品

所需的时间为：

$$max(t_{w1_i}, t_{w2_i}) + t_{wash} + load_j$$

其中，i 为第 i 个物料，j 为第 j 台 CNC。

## 4.5 目标函数

**产品总效益**

$$\begin{cases} Max(product) & Max(product_i) \neq Max(product_j)(\forall i, j) \\ Max(half - product) & Max(product_i) = Max(product_j)(\exists i, j)) \end{cases}$$

## 4.6 情景假设

此算法的目标函数只和产品总效益相关，因此为了达到目标函数的要求，并结合科学性、合理性、有效性和工厂生产的实际情况的考虑，我们对工作流程做以下假设：

1. 在流程开始时，所有的 CNC 都处于空闲状态，此时 RGV 先填满第一区域的空闲 CNC；在第一区域 CNC 被填满之后，所有该区域的 CNC 都不再发出 “需求信号”，除非该区域的某台 CNC 发生故障，在故障修复后，重新发出 “需求信号”。

2. 在填满第一区域的 CNC 之后，如果 RGV 的位置元素位于第一区域，只能进行两种动作：

- 对一台 CNC 进行上下料（换料），得到一个半成品，同时该 CNC 获得一个准备加工第一道工序的生料；
- 对一台刚从故障中恢复的 CNC 进行上料，得不到半成品；

考虑工厂生产的实际情况，RGV 拿到半成品后可以将其储存在临时的半成品储存区中，其可以从第一区域移动到一或二区域。

3. 在填满第一区域的 CNC 之后，如果 RGV 的位置元素位于第二区域，只能进行两种动作：

- 对一台刚才没有发生故障的 CNC 进行上下料（换料），得到一个完成品，同时该 CNC 获得一个准备加工第二道工序的半成品；
- 对一台 CNC 进行下料，下料之后该 CNC 转换成空闲等待的状态。

4. RGV 只能在听到“需求信号”或“完成信号”时才可以移动，如果没有任何信号，RGV 可以选择在上一步的终点等待信号，不可以自由移动（深度搜索每一种可能）。如何选择下一步将要去的目的地 CNC，我们在这一时刻生成截面参数 $tmp_i(i = 1, ..., 8)$，表示此刻开始计算，RGV 在第 i 台 CNC 处开始进行加工的时刻之间间隔的时间（包含在 CNC 处上料的时间）。选择 $max(tmp_i)$ 的第 i 台 CNC，作为此刻 RGV 下一刻将要去的位置坐标。

## 4.7 约束条件

1. 从移动的条件来看：

$$O_{des_k} = 1, O_{des_{k+1}} = 0$$

2. 从整个系统开始工作起，累积时刻的最大值不超过 8 小时。对时间的约束方程：

$$\sum_{j=1}^{k}[t_k + t_{wash} * O_{des_k} + load_{tp_{des_k}} + dt_k] < 8 * 3600$$

3. 此外，根据深度搜索，决定 $des_{k+1} = i$，同时，需更新 $O_i, nt_j(j = 1, ..., 8; j \neq i)$，更新方法为：

$$O_i = 1$$

$$nt_j = max(nt_j - [t_k + t_{wash} * O_{des_k} + load_{tp_{des_k}} + dt_k], 0)$$

$$O_j = min(nt_j * O_j, 1)$$

且 $i \neq j$

## 4.8 模型结果

### 4.8.1 第 1 组数据的结果

最优状态：229 个完成品；5 个半成品。

其中，8 小时结束时还在 CNC 生产的物料分别为：

| CNC 编号 | 工序 | 开始加工时间/s |
|---|---|---|
| 1 | 1 | 28429 |
| 2 | 2 | 28485 |
| 3 | 1 | 28533 |
| 5 | 1 | 28581 |
| 6 | 2 | 28637 |
| 7 | 1 | 28772 |

#### 4.8.2 第 2 组数据的结果

s 最优状态：160 个完成品；83 个半成品。

其中，8 小时结束时还在 CNC 生产的物料分别为：

| CNC 编号 | 工序 | 开始加工时间/s |
|---|---|---|
| 3 | 1 | 28209 |
| 1 | 1 | 28272 |
| 4 | 2 | 28370 |
| 5 | 1 | 28433 |
| 2 | 2 | 28539 |
| 6 | 2 | 28645 |
| 7 | 1 | 28708 |
| 8 | 2 | 28773 |

#### 4.8.3 第 3 组数据的结果

最优状态：190 个完成品；1 个半成品。

其中，8 小时结束时还在 CNC 生产的物料分别为：

| CNC 编号 | 工序 | 开始加工时间/s |
|---|---|---|
| 1 | 1 | 28010 |
| 5 | 1 | 28126 |
| 7 | 1 | 28292 |
| 3 | 1 | 28469 |

### 4.9 模型简评

利用深度优先搜索算法，我们的首要目标是最大化产品效益。在同时具有多个 CNC 选择时，才可以执行深度探索，尝试所有可能的决策，以寻求到产品效益最大的调度策略。因此，该算法的应用场景是智能 RVG 调度系统高速运作的情况。

- 该模型的**优点**是：以工厂生产的常规思路为出发点，最大化产品效益，因此完成品的数量达到最高；
- 该模型的**缺点**是：由于深度优先搜索工作的原理是在限定条件下（8 小时），尽可能历遍所有可能的决策，将大量的时间都使用在移动的过程中，从而得到最优解。然而，

对于一个生产工厂而言，除了产品效益，时间利用、能源利用也同样重要，深度优先搜索算法并没有考虑到这些方面。基于此，我们进行下述模型——最短寻道时间优先法的探索。

# 5 两道工序的模型建立与求解——最短寻道时间优先法

## 5.1 情景假设

此模型的目标函数同时考虑了产品效益和时间利用率，使得**综合效益**更高，我们在模型二的基础上，对情景假设做了以下补充：

1. 一旦有信号（无论信号种类）发出，RGV 就向发出该信号的 CNC 移动。
2. 当有多台 CNC 同时发出**相同种类**信号时，RGV 优先选择距离自身位置最近的一台发出信号的 CNC 移动，如果有距离相同的 CNC，则随机选择其中一台移动。
3. 当有多台 CNC 同时发出**不同种类**信号时，RGV 没有自己的判断，会通过效益最大化（尽可能满足目标函数要求）来进行决策。

## 5.2 目标函数

**对时间的利用率**

$$Max\{[(\sum t_{r_{load}} + \sum t_{r_{off}} + \sum t_{wash}) + \sum t_{work}]/(8*8*3600)]\}$$

**产品总效益**

$$Max(AB^{\alpha})$$

## 5.3 约束条件

考虑 RGV 从 $des_k$ 到 $des_{k+1}$ 过程中的约束：

1. 从移动的条件来看：
$$O_{des_k} = 1, O_{des_{k+1}} = 0$$
2. 如果面临多台正在同时发信号的 CNC，决策条件为：
$$t_k = min(nt_i * O_i), i = 1, ..., 8$$

3. 同约束条件 3，决策条件为：

$$dt_k = min(d_{des_k i} * (1 - O_i))$$

且 $dt_k > 0$

4. 从整个系统开始工作起，累积时刻的最大值不超过 8 小时。对时间的约束方程：

$$\sum_{j=1}^{k}[t_k + t_{wash} * O_{des_k} + load_{tp_{des_k}} + dt_k] < 8 * 3600$$

5. 此外，根据满足 2 和 3 约束条件的 i, 决定 $des_{k+1} = i$，同时，需更新 $O_i, nt_j (j = 1, ..., 8; j \neq i)$，更新方法为：

$$O_i = 1$$

$$nt_j = max(nt_j - [t_k + t_{wash} * O_{des_k} + load_{tp_{des_k}} + dt_k], 0)$$

$$O_j = min(nt_j * O_j, 1)$$

且 $i \neq j$

## 5.4 模型结果

| 数据组 | 完成品数 | 半成品数 | 时间利用率 |
|---|---|---|---|
| 1 | 64 | 322 | 82.7% |
| 2 | 85 | 257 | 93.5% |
| 3 | 48 | 192 | 71.9% |

**模型比较**

根据该模型的计算指标时间利用率，对深度优先算法的模型结果增加相应指标计算，用于对比：

| 数据组 | 完成品数 | 半成品数 | 时间利用率 |
|---|---|---|---|
| 1 | 229 | 5 | 89.1% |
| 2 | 160 | 83 | 80.6% |
| 3 | 190 | 1 | 59.0% |

### 5.5 模型简评

凭借最短寻道时间优先法，首要目标是最大化时间利用效率。根据该算法的原理，在同时有多个 CNC 可以选择时，优先选择移动时长最短的 CNC，如果没有 CNC 发出信号时，RGV 只能在原地等待，不能四处移动进行探索，在多种选择中寻求时间利用效率最大的最优解。

- 该模型的**优点**是：不同于常规对于优化 RGV 调度系统的角度，最大化时间利用效率，因此在同时产生较多完成品和较多半成品的情况下，保证对于物料来说，上下料、洗料、加工之外的时间占比最小；
- 该模型的**缺点**是，由于最大时间利用效率的要求，对于第一区域的 CNC 而言，如果有加工挖成的半成品，RGV 一定会选择优先把半成品下料，以避免加工完成的物料在加工台上等待较长时间。然而，这个流程导致真正生成的成品数量较少，而最优情况下的半成品数量较多，从完成品效益来看，并不是十分完美，堆积半成品对于工厂的生产线转化率有负向影响。基于此，我们提出研究展望。

## 6 模型不足与展望

- 定义：每连续工作 8 小时，称完成一次单位作业；
- 定义：单位作业产成品期望值，是指工厂依次完成单位作业，并且上一单位半成品可计入下一单位作业的条件下，每单位作业产生的产成品的期望；
- 定义：负责第二道工序的 CNC: $CNC_2$。

1. 两道工序问题的模型结合

基于上述对两道工序的**深度优先搜索算法**和**最短寻道时间优先算法**优缺点的阐述：深度优先适合于高速运作的系统，尽可能历遍所有决策，可以最大化产品效益（$max(product)$）；时间优先适合于整个系统还未进入高速运作时，尽可能减少在**非物料生产**阶段花费的时间，最大化整个系统的时间利用效率。在这种考量下，将两个模型分别加入一个系统运作的相应阶段，可能很大程度上同时提高完成品的产出效益和时间利用效率，从而使得全局的效率更优。

2. 考虑上一次单位作业的半成品，可直接转入此次作业，进行后续加工。即就是说，每一次单位作业开始时，半成品的数量不是 0，而是上一次单位作业结束时，半成品的量。在这种考量之下：

（1）模型一，将每次单位作业的完成品数量最多（$max(product)$）作为唯一目标函数，可能无法得到最大的单位作业产成品期望值。因为，成品最大化情况下，单位作业结束时半成品的数量较少，在下一单位作业开始时，多台 $CNC_2$ 不能直接加工上一轮的半成品，必须等待新的半成品加工完成，$CNC_2$ 的闲置等待较长时间。

（2）模型二，按照原假设得到的最优解，经常会出现加工第二道工序的 CNC 有空闲的状况，故而最优解一般存在着，半成品远多于成品数量。新假设下，半成品数量足够，只要负责第二道工序的 CNC 发出信号，RGV 就可以将半成品放置其上，不会再出现空闲的 $CNC_2$。在保证时间利用率最高的情况下，增加了 RGV 单位作业的平均产成品数量。

新假设下可能的最优解：

考虑 p 次依次发生的单位作业，每次的产成品数是 $product_i(i = 1, ..., p)$，以模型二为基础，添加新的目标函数，$max(\sum_i product_i))$

3、考虑其他可能的影响效率评价的因素：

（1）能耗（功率）：

| - | CNC | RGV |
|---|---|---|
| 待机功率 | 328.5W | <1000W |
| 全功率 | 4035.4W | 6720W |

- CNC 待机功率 ($W_{cw}$)：加工台上无工件、工件等待被取走
- CNC 全功率 ($W_{ct}$)：正在加工工件
- RGV 待机功率 ($W_{rw}$)：没有接收到任何 CNC 发出的信号，原地等待
- RGV 全功率 ($W_{rt}$)：移动、上下料、清洗

**能耗的计算**

- CNC 的总待机时间 ($T_{cw}$) = 8*8 - 所有的用于加工的时间
- CNC 的全功率时间 ($T_{ct}$) = 所有的用于加工的时间 6
- RGV 的总待机时间 ($T_{rw}$) = $\sum_k t_k$
- RGV 的全功率时间 ($T_{rt}$) = $8 - T_{rw}$
- 总能耗 = $W_{cw}\ T_{cw} + W_{ct}\ T_{ct} + W_{rw}\ T_{rw} + W_{rt}\ T_{rt}$

电费的收取是阶梯式的，生产相同数量的成品，耗能越多，成本就越高。并且，多数工厂使用烧煤来供能，能耗越高，需要燃烧的煤炭量就越多，产生的水、空气等污染也就越严重。一方面，国家对于企业的工业污染排放量有严格的限制，超额排放需缴纳大量费用，成本很高；另一方面，从维护环境的角度，也应尽量将能耗控制住合理范围内，以减少污染。

# 7 参考文献

[1] 陈华. 基于分区法的 2-RGV 调度问题的模型和算法 [J]. 工业工程与管理,2014,19(06):70-77.

[2] 刘永强. 基于遗传算法的 RGV 动态调度研究 [D]. 合肥工业大学,2012.

[3] 龚建华. 深度优先搜索算法及其改进 [J]. 现代电子技术,2007(22):90-92.

# 8 附录

## 8.1 Lingo 代码

```
# 无故障周期代码
MODEL:

SETS:
  CNC/1..8/:u;
  LINK(CNC,CNC):dist,x;
 ENDSETS
 DATA:
  dist=?;
 ENDDATA
 n=@size(CNC);
 min=@sum(LINK:dist*x);
 @FOR(LINK: @BIN(x));
 @FOR(CNC(K):@SUM(CNC(I)|I#NE# K:x(I,K))=1;
             @SUM(CNC(J)|J#NE# K:x(K,J))=1;);
 @FOR(CNC(I):@FOR(CNC(J)|J#GT#1 #AND# I#NE#J:
             u(I)-u(J)+n*x(I,J)<=n-1););
 @FOR(CNC(I): u(I)<=n-1);

END

# 有故障周期代码
MODEL:

 SETS:
  CNC/1..7/:u;
  LINK(CNC,CNC):dist,x;
 ENDSETS
 DATA:
  dist=?;
 ENDDATA
 n=@size(CNC);
```

```
 min=@sum(LINK:dist*x);
 @FOR(LINK: @BIN(x));
 @FOR(CNC(K):@SUM(CNC(I)|I#NE# K:x(I,K))=1;
             @SUM(CNC(J)|J#NE# K:x(K,J))=1;);
 @FOR(CNC(I):@FOR(CNC(J)|J#GT#1 #AND# I#NE#J:
             u(I)-u(J)+n*x(I,J)<=n-1););
 @FOR(CNC(I): u(I)<=n-1);

END

# 无故障周期矩阵数据输入
DATA

GROUP1
dist =  28 59 76 79 89 92 102 105
        59 31 79 82 92 95 105 108
        76 79 28 59 76 79  89  92
        79 82 59 31 79 82 92 95
        89 92 76 79 28 59 76 79
        92 95 79 82 59 31 79 89
       102 105 89 92 76 79 28 59
       105 108 92 95 79 82 59 31;




GROUP2
dist =  30 65 83 88 101 106 119 124
        65 35 88 93 106 111 124 129
        83 88 30 65  83   88  101 106
        88 93 65 35  88   93  106 111
       101 106 83 88 30   65    83   88
       106 111 88 93 65 35 88 93
       119 124 101 106 83 88 30 65
       124 129 106 111 88 93 65 35;
```

```
GROUP3
dist =  29 59 72 77 86 91 100 105
        59 32 77 82 91 96 105 110
        72 77 27 59 72 77 86 91
        77 82 59 32 77 82 91 96
        86 92 72 77 27 59 72 77
        91 96 77 82 59 32 77 82
       100 105 86 91 72 77 27 59
       105 110 91 96 77 82 59 32;

# 有故障周期矩阵数据输入

DATA:
3号机故障

Group1
dist = 31 79 82 92 95 79 82
       79 28 59 76 79 89 92
       82 59 31 79 82 92 95
       92 76 79 28 59 102 105
       95 79 82 59 31 105 108
       79 89 92 102 105 28 59
       82 92 102 105 108 59 31;


Group2
dist =  35 88 93 106 111 88 93
        88 30 65 83 88 101 106
        93 65 35 88 93 106 111
       106 83 88 30 65 83 88
       111 88 93 65 35 88 93
        88 101 106 83 88 30 65
```

```
        93  106 111 88 93 65 35;



Group3
dist =  32 77 82 91 96 77 82
        77 27 59 72 77 86 91
        82 59 32 77 82 91 96
        91 72 77 27 59 72 77
        96 77 82 59 32 77 82
        77 86 91 72 77 27 59
        82 91 96 77 82 59 32;
```

## 8.2 Python 代码

### 8.2.1 针对两道工序的最短寻道时间优先算法

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Sep 15 12:49:39 2018
算法：针对两道工序的最短寻道时间优先算法

"""

import sys
reload(sys)
sys.setdefaultencoding('utf8')
import numpy as np
from numpy import array

clean=25
updown=array([27,32,27,32,27,32,27,32])
t1=545
t2=455
```

```
num=0

t=[0,0,0,0,0,0,0,0]#初始化信号提示时间
d=array([[0,0,18,18,32,32,46,46],
        [0,0,18,18,32,32,46,46],
        [18,18,0,0,18,18,32,32],
        [18,18,0,0,18,18,32,32],
        [32,32,18,18,0,0,32,32],
        [32,32,18,18,0,0,32,32],
        [46,46,32,32,18,18,0,0],
        [46,46,32,32,18,18,0,0]])#移动距离--时间匹配矩阵


    #储存第二类加工工具上有无物体的数组，0是没东西
o=[0,0,0,0,0,0,0,0]
c=np.random.randint(1,3,size=8)
#temp=0
total=0
countdict={}
m={}
temp=0
k1=0#半成品
count=0

countobnum=0
countsemima=0
countfinish=0
counttakenum=0
put=0

objectputtime={}
objectputnum={}
semiobject={}
semiobma={}
```

```
finishobject={}
finishobma={}
objecttaketime={}
upfinishob={}



for n in range(500):

          m[n]=0
          total=total+m[n]
          if total<28800:

                for i in range(8):
                    if c[i]==2:
                        t[i]==t1+updown[i]
                for i in range(8):
                    if o[i]==0:
                        if c[i]==1:
                           tp1=i
                           objectputnum[countobnum]=tp1
                           countobnum=countobnum+1
                           m[n]= t[tp1]+d[temp,tp1]+updown[tp1]
                           t[tp1]=t1
                           for i in range(8):
                               if i<>tp1:
                                   t[i]=t[i]-min(t[i],m[n])
                           o[i]=1
                           total=total+m[n]
                           objectputtime[countobnum]=total
                           temp=tp1

                        else:#在第二类为空
                           tp1=i
```

```
                t[tp1]=t[tp1]+d[temp,tp1]+updown[tp1]+t1
                tt=t.index(min(t))
                if c[tt]==2:
                    tp1=tp1

                    t[tp1]=0
                    m[n]=0
                    total=total+m[n]
                    temp=tp1
                else:#下一目的地是1区
                    m[n]=t[tp1]+d[temp,tp1]+updown[tp1]
                    t[tp1]=t1
                    for i in range(8):
                                if i<>tp1:
                                    t[i]=t[i]-min(t[i],m[n])
                total=total+m[n]
                temp=tp1
            else:#o[i]=1
                if c[i]==1:
                    tp1=t.index(min(t))
                    if c[tp1]==1:
                        objectputnum[countobnum]=tp1
                        countobnum=countobnum+1
                        counttakenum=counttakenum+1

                        if o[tp1]==1:
                            tp1=i
                            t[tp1]=0
                            m[n]=0
                            total=total+m[n]
                            temp=tp1
                        else:
                            tp1=tp1
                            m[n]=t[tp1]+d[temp,tp1]+updown[tp1]
```

```
                                t[tp1]=t1

                                for i in range(8):
                                        if i<>tp1:
                                            t[i]=t[i]-min(t[i],m[n])
                                total=total+m[n]
                                temp=tp1
                            objectputtime[countobnum]=total


          #    key_list=[ ]
          #    value_list=[ ]

          #    for key,value in objectputnum.items( ):
          #        key_list.append(key)
          #        value_list.append(value)
          #    if tp1 in value_list:
          #       objecttakenum=value_list.index(tp1)
# objecttakenum=list(objectputnum.keys())
#                        [list(objectputnum.values()).index(tp1)]
          #    objecttaketime[key]=total
                            else:#去2区
                                countsemima=countsemima+1

                                upfinishob[put]=total+t[tp1]+d[temp,tp1]
                                put=put+1
                                if o[tp1]==0:
# objecttakenum=list(objectputnum.keys())
#                      [list(objectputnum.values()).index("tp1")]

 #  objecttaketime[objecttakenum]=total
                                o[tp1]=1
                                m[n]=t[tp1]+d[temp,tp1]+updown[tp1]
                                t[tp1]=t2
```

```
                                for i in range(8):
                                        if i<>tp1:
                                            t[i]=t[i]-min(t[i],m[n])
                                total=total+m[n]
                                temp=tp1

                                semiobma[countsemima]=total
                            else:
                                m[n]=t[tp1]+d[temp,tp1]+updown[tp1]+clean
                                t[tp1]=t2
                                for i in range(8):
                                        if i<>tp1:
                                            t[i]=t[i]-min(t[i],m[n])
                                total=total+m[n]
                                temp=tp1
                                count=count+1
                                countfinish=countfinish+1

                                semiobma[countsemima]=total
                                finishobma[countfinish]=tp1
                                finishobject[countfinish]=total
                                countdict[num]=countobnum
                                num=num+1
                                print count
                                print countdict
                    #print tp1

            else:
                break

                    #countdict[p]=count
                #   continue
```

### 8.2.2 深度优先搜索 + 剪枝算法

```
# encoding: utf8

"""
Created on Sat Sep 16 2018
算法：深度优先搜索+剪枝算法

"""

from heapq import heappush, heappop, heapify
from pprint import pprint
from copy import deepcopy
from collections import deque
from random import random, uniform
import pandas as pd
import os


class OperationTime(object):
    def __init__(self, cnc: int=0, time: int=0, desc: list=None):
        """
        :param cnc: CNC机器编号
        :param time: 消耗时候
        :param desc: 描述
        """
        self.cnc = cnc
        self.time = time
        if not desc:
            self.desc = []
        else:
            self.desc = desc
        self.detail = []

    def add(self, time=0, desc=''):
```

```
        self.time += time
        self.desc.append('%s(%s)' % (desc, time))

    def process(self, product_order, time, is_up_liao):
        """
        :param product_order: 工序
        :param time: 开始时间
        :param is_up_liao: 是否上料, True 则上料, False 则表示下料
        :return:
        """
        self.detail.append((self.cnc, product_order, time, is_up_liao))

    def __str__(self):
        return '[%s]号CNC操作总耗时: %4s; 分步耗时: %s'
                %(self.cnc, self.time, self.desc)


def mini_operation(cnc: int=0, time_cost: int=0, desc:str=''):
    return OperationTime(cnc, time_cost, ['%s(%s)' % (desc, time_cost)])


class CNSState(object):
    def __init__(self, time=0, has_product=0):
        """
        机器状态
        :param time: 是否空闲。0: 空闲;
                     其他: 表示开始加工的时间, 与当前流逝时间对比
        :param has_product: 是否有成品

        (1) 等待上料:                          has_product = 0 & time = 0
        (2) 正在加工: 等待加工完成, 做下料操作. has_product = 0 & time > 0
        (3) 加工完成: 直接下料                  has_product = 1 & time = 0
        """
        self.has_product = has_product
```

```
        self.time = time
        self.breakdown = False
        self.breakdown_time = None
        self.fix_time = None

    def reset(self):
        self.has_product = 0
        self.time = 0

    def __str__(self):
        if self.has_product == 0:
            if self.time > 0:
                return '第 [%s]s 开始加工，目前正在加工...' % self.time
            return '空闲状态'
        return '加工完成，等待下料'

    def random_breakdown(self, break_time:int, prob=0.01):
        # 0.01 的概率随机故障
        if random() > prob:
            return False
        self.breakdown = True
        self.breakdown_time = break_time
        self.fix_time = break_time + (600 * random() + 600)
        # 修复时间：10-20分钟
        return True

    def _fix(self):
        # 修复时，一切清空
        self.breakdown = False
        self.fix_time = None
        self.breakdown_time = None
        self.reset()

    def is_available(self, state, cnc_order:int, product_order:int):
```

```
        # 当故障且暂未修复时，不可用
        if self.breakdown:
            if state.time < self.fix_time:
                return False
            self._fix()
            op = OperationTime(cnc_order)
            op.process(product_order, None, False) # 下料时间为 None，报废
            state.operations.append(op)
        return True


class RgvState(object):
    def __init__(self, maxValue=0, time=0, cnc_state_list=None,
                 product=0, half_product=0, operations=None,
                 breakdowns=None):
        """
        每个时间点：RGV的状态 和 cnc_state_list(各台机器的状态)
        :param maxValue: 剩余时间下能创造的最大价值
        :param time: 已流逝的时间
        :param cnc_state_list: 每台机器的状态，消耗时长
        """
        self.maxValue = 0
        self.time = 0
        if not cnc_state_list:
            self.cns_state_list = [CNSState() for i in range(8)]
        self.product = 0
        self.half_product = 0
        self.move_list = [1]    # 移动路线
        if not operations:
            self.operations = []
        if not breakdowns:
            self.breakdowns = []

    def __str__(self):
```

```
        return '成品数量: %s; 半成品数量: %s; 耗时: %s; 移动路径: %s'
                % (self.product, self.half_product, self.time, self.move_list)

    def __cmp__(self, other):
        # 用于 MinQueue 排序，数值小的排前边

        # 先比较成品数量，成品少的为1
        if self.product < other.product:
            return 1
        if self.product > other.product:
            return -1

        # 再比较成品数量，半成品少的为1
        if self.half_product < other.half_product:
            return 1
        if self.half_product > other.half_product:
            return -1

        # 再比较消耗的时间
        if self.time > other.time:
            return 1
        if self.time < other.time:
            return -1
        return 0

    def compareTo(self, other):
        return self.__cmp__(other)

    def __lt__(self, other):
        return self.__cmp__(other) == -1

    def __gt__(self, other):
        return self.__cmp__(other) == 1
```

```
    def __eq__(self, other):
        return self.__cmp__(other) == 0


class MinQueue(object):
    def __init__(self, items:list=None):
        if not items:
            self.items = []
        else:
            self.items = heapify(items)
        self.N = len(self.items)

    def size(self):
        return self.N

    def push(self, item):
        self.N += 1
        heappush(self.items, item)

    def popMin(self):
        self.N -= 1
        return heappop(self.items)


class MaxTryException(Exception):
    pass


class TwoStepRgvSolver(object):
    def __init__(self, move_speed=(20, 33, 46), process_time={1: 400, 2:378},
                 wash_time=25, liao_time=[28, 31] * 4, max_time=8 * 3600,
                 max_try=10000,
                 order_dict=None):
        """
```

```
        :param move_time: RGV 移动1, 2, 3个单位所需的时间
        :param process: CNC加工完成一个两道工序物料的第一道工序所需时间
                        和 第二道工序所需时间
        :param wash_time: RGV完成一个物料的清洗作业所需时间
        :param liao_time: RGV为CNC1# - CNC8# 一次上下料所需时间
        :param max_time: 总时长, 8小时
        :param max_try: 深度优先最大的搜索数量
        :param order_dict: 每台机器负责第几道工序
        """
        self.move_speed = dict(zip(range(1, 4), move_speed))
        self.move_speed[0] = 0
        self.process_time = process_time
        self.wash_time = wash_time
        self.liao_time = liao_time
        self.max_time = max_time
        self.max_try = max_try
        self.try_time = 0
        if not order_dict:
            self.order_dict = {1: 1, 2: 2, 3: 1, 4: 2,
                               5: 1, 6: 2, 7: 1, 8: 2}
        self.possible_cns = list(range(1, 9))
        self.__set_dist()

        self.RgvStateQueue = MinQueue()
        self.optimizeState = RgvState()

        self.min_product_time = max(self.process_time.values()) + self.wash_time
        # search queue
        self.state_queue = deque()

    def generate_singal(self, state: RgvState):
        # 在某一时间点状态下, 检测每台机器的状况, 生成信号
        current_cnc = state.move_list[-1]
```

```
        signals = MinQueue()
        for cnc_index, cnc in enumerate(state.cns_state_list):
            cnc_index += 1
            if not cnc.is_available(state, cnc_index, self.order_dict[cnc_index]):
                continue
            move_time = self._move_cost_time(current_cnc, cnc_index)
            if cnc.time == 0:
                # 空闲，加入信号 &(等待时长，cnc 机器编号)
                signals.push((move_time, 0, cnc_index))
            else:
                # 加工完成，加入信号
                product_order = self.order_dict[cnc_index]
                # TODO: 加工过程中，某台机器产生异常？
                wait_time = cnc.time + self.process_time[product_order] - state.time
                if wait_time <= 0:  # 加工完成
                    cnc.time = 0
                    cnc.has_product = 1
                    signals.push((move_time, 0, cnc_index))
                else:
                    signals.push((move_time + wait_time, wait_time, cnc_index))
        return signals

    def __set_dist(self):
        # 每台机器之间的距离
        dist = {(1, 1): 0, (2, 2): 0, (3, 3): 0, (4, 4): 0,
                (5, 5): 0, (6, 6): 0, (7, 7): 0, (8, 8): 0,
                (1, 2): 0, (1, 3): 1, (1, 4): 1, (1, 5): 2, (1, 6): 2, (1, 7): 3, (1, 8)
                (2, 3): 1, (2, 4): 1, (2, 5): 2, (2, 6): 2, (2, 7): 3, (2, 8): 3,
                (3, 4): 0, (3, 5): 1, (3, 6): 1, (3, 7): 2, (3, 8): 2,
                (4, 5): 1, (4, 6): 1, (4, 7): 2, (4, 8): 2,
                (5, 6): 0, (5, 7): 1, (5, 8): 1,
                (6, 7): 1, (6, 8): 1,
                (7, 8): 0
                }
```

```
        more_dist = {(s2, s1): self.move_speed[d] for (s1, s2), d in dist.items() if s1
        for (s1, s2), d in dist.items():
            more_dist[(s1, s2)] = self.move_speed[d]
        self.move_time = more_dist

    def _move_cost_time(self, state_from: int, state_to: int):
        # 从一台机器移动到另一台机器的时间
        return self.move_time[(state_from, state_to)]

    def is_timeout(self, state: RgvState):
        # 检查是否超时 & 如果超时, 则检测最终的状态是否是最优的
        if state.time > self.max_time:
            # print('timeout state: ', state.time, state)
            self.update_optimize_state(state)
            return True
        return False

    def update_optimize_state(self, state: RgvState):
        self.try_time += 1
        if self.try_time >= self.max_time:
            raise MaxTryException('max try exceed.')
        # 与最优状态对比
        # print('compare time: %s' % self.try_time)
        if state.compareTo(self.optimizeState) == -1:
            self.optimizeState = state

    def move(self, state: RgvState, cnc_to: int):
        # 从当前状态移动到第 cnc_to 台机器进行操作
        # 移过去后肯定是可以直接加工的!

        operation = OperationTime(cnc_to)
        cnc_from = state.move_list[-1]
        time_cost = self._move_cost_time(cnc_from, cnc_to)
        state.time += time_cost
```

```
        # 移动超时
        if self.is_timeout(state):
            return
        operation.add(time_cost, '移动时长')
        res = self.update_state_cnc(state, cnc_to, operation)
        # 状态更新超时
        if not res:
            return
        state, operation = res
        state.operations.append(operation)
        # 更新移动路径 & 修改最佳剩余时间
        state.move_list.append(cnc_to)
        # print('更新状态  %s -> %s; 移动耗时: %s; 总耗时: %s;
        # move: %s' %
        # (cnc_from, cnc_to, time_cost, state.time, state.move_list))
        # state.maxValue = self.get_max_value(state)

        # 当 RGV 机器到达CNC 时, CNC 一定不会故障,
        # 否则不会发生信号让 RGV 过来
        # 只有在上料后, 加工过程中, 才会产生故障。
        # 所以在加工过程中随机的从 [开始加工的时间点,
        # 到加工结束点] 找到一个时间点, 按1%的概率故障
        # 同时随机生成需要修复的时间(10分钟-20分钟):
        # [10 * 600, 20 * 600]
        # 机器故障时, 下一步的信号列表中不会有此机器,
        # 直到它被修复
        cnc_state = state.cns_state_list[cnc_to - 1]
        product_order = self.order_dict[cnc_to]
        return state

    def update_state_cnc(self, state: RgvState, cnc_order: int, operation:OperationTime)
        """
        到达某个 CNC 机器后, 对 CNC 机器做操作。机器可能的状态:
        因为要发出信号才能移动, 所以过去后一定是空闲的
```

```
        (1) 等待上料:                          has_product = 0 & time = 0
        (2)
        (3) 加工完成: 直接下料                  has_product = 1 & time = 0
        更新总体的状态
        :return:
        """
        cnc_state = state.cns_state_list[cnc_order - 1]

        liao_time = self.liao_time[cnc_order - 1]
        # 上料 or 下料时间
        product_order = self.order_dict[cnc_order]
        # 1: 加工第一道工序, 2: 加工第二道工序
        process_time = self.process_time[product_order]
        # 等待加工
        if cnc_state.time > 0:
            wait_time = cnc_state.time + process_time - state.time
            if wait_time > 0:  # 加工未完成, 继续等待一会儿
                state.time += wait_time
                if self.is_timeout(state):
                    return
                operation.add(wait_time, '等待加工')
            # 更新 has_product 状态
            cnc_state.has_product = 1
            cnc_state.time = 0


        # 加工完成, 等待下料 or 洗料
        if cnc_state.has_product:
            if product_order == 1:
                # 增加下料消耗的时间 & 检测超时
                state.time += liao_time
                if self.is_timeout(state):
                    return
                operation.add(liao_time, '一工序下料->上料')
```

```
            operation.process(product_order, state.time, False)
            operation.process(product_order, state.time, True)
            # 完成第一道工序，半成品 +1
            state.half_product += 1
            cnc_state.time = state.time  # 上料

            # 当 RGV 机器到达CNC 时，CNC 一定不会故障，
            # 否则不会发生信号让 RGV 过来
            # 只有在上料后，加工过程中，才会产生故障。
            # 所以在加工过程中随机的从 [开始加工的时间
            # 点，到加工结束点] 找到一个时间点，按1%的概率故障
            # 同时随机生成需要修复的时间(10分钟-20分钟):
            # [10 * 600, 20 * 600]
            # 机器故障时，下一步的信号列表中不会有此机器，
            # 直到它被修复
            cnc_state.random_breakdown(uniform(cnc_state.time, cnc_state.time + proc
            if cnc_state.breakdown:
                state.breakdowns.append((cnc_order, cnc_state.breakdown_time, cnc_st


        else:
            # 完成第二道工序  -> 洗料 -> 下料
            state.time += self.wash_time
            if self.is_timeout(state):
                return
            operation.add(self.wash_time, '二工序洗料')
            state.time += liao_time
            if self.is_timeout(state):
                return
            if state.half_product > 0:
                operation.add(liao_time, '二工序下料->上料')
                operation.process(product_order, state.time, False)
                operation.process(product_order, state.time, True)
                cnc_state.time = state.time
```

```
                cnc_state.random_breakdown(uniform(cnc_state.time, cnc_state.time +
                if cnc_state.breakdown:
                    state.breakdowns.append((cnc_order, cnc_state.breakdown_time, cn

                state.half_product -= 1
            else:
                operation.add(liao_time, '二工序下料')
                operation.process(product_order, state.time, False)
                cnc_state.time = 0
            state.product += 1
        cnc_state.has_product = 0
        state.cns_state_list[cnc_order - 1] = cnc_state
        return state, operation


    # 等待上料
    # 给第一道工序加料
    if product_order == 1:
        # 增加上料消耗的时间 & 检测超时
        state.time += self.liao_time[cnc_order - 1]
        if self.is_timeout(state):
            return
        operation.add(liao_time, '一工序上料')
        operation.process(product_order, state.time, True)
        cnc_state.random_breakdown(uniform(cnc_state.time, cnc_state.time + process_
        if cnc_state.breakdown:
            state.breakdowns.append((cnc_order, cnc_state.breakdown_time, cnc_state.

        cnc_state.time = state.time
        state.cns_state_list[cnc_order - 1] = cnc_state
    # 将半成品拿给第二道工序加料
    elif state.half_product > 0:
        # 增加上料消耗的时间 & 检测超时
```

```
            state.time += liao_time
            if self.is_timeout(state):
                return
            operation.add(liao_time, '二工序上料')
            operation.process(product_order, state.time, True)
            state.half_product -= 1
            cnc_state.time = state.time
            cnc_state.random_breakdown(uniform(cnc_state.time, cnc_state.time + process_
            if cnc_state.breakdown:
                state.breakdowns.append((cnc_order, cnc_state.breakdown_time, cnc_state.

            state.cns_state_list[cnc_order - 1] = cnc_state
        # 无半成品，只好什么都不做，这样移动时间就白白消耗了，不返回该状态
        else:
            return
        return state, operation

    def get_max_value(self, state: RgvState):
        """
        给定状态，计算可能的最优目标函数
        (1) 已成功完成两道工序 而且已经清洗下料的成品数量
        (2) 未来得及下料数量
        (3) 除去下料时间 + 剩余时间可能的成品数量
        :param state:
        :return:
        """
        # 随意吧，把半成品都当做成品好了，不可放过
        full_pro_num = state.product
        gone_time = state.time
        for cnc in state.cns_state_list:
            if cnc.has_product:
                full_pro_num += 1
                gone_time += min(self.liao_time)
        full_pro_num += (self.max_time - gone_time) / self.min_product_time * 4
```

```
        return full_pro_num

    def depthSearch(self, init_state):
        try:
            self.realDepthFirstSearch(init_state)
        except MaxTryException as e:
            print(e)
        return self.optimizeState

    def realDepthFirstSearch(self, init_state):
        if self.try_time > self.max_try:
            raise MaxTryException
        state = deepcopy(init_state)
        self.state_queue.append(state)
        # print('add %3s state; time: %5s; timeout: %s;' % (len(state.move_list), stat
        # print(state.move_list)

        while len(self.state_queue) > 0:
            state = self.state_queue.pop()
            signals = self.generate_singal(state)
            while signals.size() > 0:
                s = deepcopy(state)
                total_time, wait_time, cnc = signals.popMin()
                # print('signal from cnc [%s]; time: %s' % (cnc, total_time))
                if wait_time:
                    s.time += wait_time
                    if self.is_timeout(s):
                        break
                    s.operations.append(mini_operation(cnc, wait_time, '等待加工'))
                s = self.move(s, cnc)
                if s:
                    # self.state_queue.append(s)
                    self.realDepthFirstSearch(s)
            self.print_bst(less=True)
```

```
        # self.print_bst()
        # print('state to check: ', len(self.state_queue))

    def to_excel_time(self):
        records = {1: [], 2: []}
        cache_dict = dict()
        for o in self.optimizeState.operations:
            for (cnc_order, product_order, time, is_up_liao) in o.detail:
                key = (cnc_order, product_order)
                if is_up_liao:  # 缓存 上料时间
                    cache_dict[key] = time
                else:           # time 是下料时间
                    # 拿出上料时间，组成 (cnc 机器，上料时间，下料时间) 这样的三元组
                    up_liao_time = cache_dict[key]
                    del cache_dict[key]
                    single_record = (cnc_order, up_liao_time, time)
                    records[product_order].append(single_record)
        return records, cache_dict

    def print_bst(self, less=False, base_path='~/'):
        print('\n' + '-' * 80)
        # print(self.optimizeState)
        print('最优状态: [%3s] 成品; [%2s] 半成品; move: %s' % (self.optimizeState.produ
                                                                self.optimizeState.half_pro
                                                                self.optimizeState.move_lis
                                                            ))
        if less:
            return


        records, cache_dict = self.to_excel_time()
        print('=' * 80 + '\n' * 3)
        print('工序1 加工序列(CNC编号，上料时间，下料时间):')
        pprint(records[1])
```

```
        df1 = pd.DataFrame(records[1], columns=['工序1-CNC编号', '上料时间', '下料时间']
        df1.to_csv(os.path.join(base_path, 'step1.csv'), index=None)

        print('=' * 80 + '\n' * 3)
        print('工序2 加工序列(CNC编号, 上料时间, 下料时间):')
        pprint(records[2])
        df2 = pd.DataFrame(records[2], columns=['工序2-CNC编号', '上料时间', '下料时间']
        df2.to_csv(os.path.join(base_path, 'step2.csv'), index=None)

        print('=' * 80 + '\n' * 3)
        print('故障列表:')
        pprint(self.optimizeState.breakdowns)
        df2 = pd.DataFrame(self.optimizeState.breakdowns, columns=['故障-CNC编号', '故障
        df2.to_csv(os.path.join(base_path, 'breakdown.csv'), index=None)

        print('=' * 80 + '\n' * 3)
        print('结束时还在CNC 机器上生产的小东西: ')
        print(cache_dict)
        print('最优状态: [%3s] 成品; [%2s] 半成品; move: %s' % (self.optimizeState.produ
                                                        self.optimizeState.half_product,
                                                        self.optimizeState.move_list
                                                        ))
        result = """最优状态: [%3s]成品; [%2s]半成品;\n结束时还在CNC 机器上生产的小东西
        """%(self.optimizeState.product, self.optimizeState.half_product, cache_dict)
        with open(os.path.join(base_path, 'result.txt'), 'w') as f:
            f.write(result)


if __name__ == '__main__':
    import time as timer
    t1 = timer.time()
    run_para_order = 3 # 设置运行第几组参数
    base_path = '/Users/yalei/2018-mathmodel'
    if run_para_order == 1:
```

```
        solver = TwoStepRgvSolver(max_time=8 * 3600, max_try=1000)
        solver.depthSearch(RgvState())
        file_path = os.path.join(base_path, 'parameter1')
        if not os.path.exists(file_path):
            os.mkdir(file_path)
        solver.print_bst(base_path=file_path)

    elif run_para_order == 2:
        solver2 = TwoStepRgvSolver(move_speed = (33, 41, 59),
                                   process_time = {1: 280, 2: 500},
                                   wash_time = 30,
                                   liao_time = [30, 35] * 4,
                                   max_try=1000
                                   )
        solver2.depthSearch(RgvState())
        file_path = os.path.join(base_path, 'parameter2')
        if not os.path.exists(file_path):
            os.mkdir(file_path)
        solver2.print_bst(base_path=file_path)

    elif run_para_order == 3:
        solver3 = TwoStepRgvSolver(move_speed=(18, 32, 46),
                                   process_time={1: 455, 2: 182},
                                   wash_time=25,
                                   liao_time=[27, 32] * 4,
                                   max_try=1000
                                   )
        solver3.depthSearch(RgvState())
        file_path = os.path.join(base_path, 'parameter3')
        if not os.path.exists(file_path):
            os.mkdir(file_path)
        solver3.print_bst(base_path=file_path)
```