元胞自动机在解决RGV动态调度策略选择问题上的应用

摘要 针对**RGV的动态调度**策略问题，利用仿真软件Plant Simulation测得基准实验数据，获得对智能加工系统的直观理解。借助**排队论**的思想建立起了一套排队系统，将原问题等价转化为排队系统中**优先队列**的构造问题。受**贪心算法**思想的启发，在此基础上加入了**预测机制**用于增强RGV的调度能力，并基于Java语言和排队模型面向对象地设计和实现了第一套**元胞自动机**，用来模拟排队过程中系统的运行情况，采集到一道工序、一道工序带随机故障、两道工序和两道工序带随机故障四组实验的模拟数据。

然而基于排队论的元胞自动机无法寻找到两道工序中不同工序**CNC比例和空间分布的最优组合**，从而较难获取最优解。借助**图论**的思想方法和数学工具，将问题进一步抽象为RGV在**图模型**中的定向游走过程，对于一道工序，图模型选择**8阶完全图**，对于两道工序，图模型选择**二部图**。定义衡量系统生产效率的指标：**系统利用率**，在图论的框架下借助矩阵空间和对拓扑结构的图编码统一了一道两道工序问题，同时基于python语言面向过程地设计了第二套元胞自动机。基于图模型的元胞自动机的计算输出被设定为系统利用率。于是问题进一步转化为在**解空间**里寻找代表**最优拓扑结构**的矩阵使得带入元胞自动机运算时得到最高的利用率。穷举所有的不同工序CNC比例和空间分布，确定了三组参数下对应的最优比例和最优空间分布。将这一信息返回给前述基于排队论的元胞自动机，可以确保最终写进excel表格的实验数据是最优的实验数据。

总体而言，排队模型、基于排队模型设计的元胞自动机和图模型、基于图模型设计的元胞自动机是对同一问题的两种不同模型刻画形式，但却是对同一问题的等价表述，两者聚焦解决的核心问题不一样，组合使用可以收到较好的实验效果。

关键词 RGV动态调度 排队系统 元胞自动机 图模型与图编码 Plant Simulation仿真

一、问题的重述

1.1 问题背景

问题中给定的智能加工系统配备有8台计算机数控机床（CNC），配备有1辆轨道式自动引导车（RGV），配备有上料传送带和下料传送带等一些附属设备。自动引导车（RGV）是一种行进在一维轨道上的无人驾驶智能车，通过接收系统指令决定移动方向和移动距离，其上装备有一只可旋转手臂、两只机械手爪，装备有清洗熟料的物料清洗槽。RGV能够完成给CNC上下料、清洗熟料等作业任务。

CNC工作台固定在系统中，负责生料的加工，加工完成以后会给RGV发出上下料请求信号，并在RGV到来之前持续等待，RGV接到请求信号之后前往智能选择的CNC工作台，双方准备就绪即开始上下料操作，上下料完成后CNC开始一轮新的生料加工，RGV开始清洗从CNC上取下来的熟料，待清洗完毕，RGV选择移动到下一个CNC或者等待。

考虑整个系统的生产效率，若要使RGV动态调度相对较优，那么就要充分考虑RGV的移动时间、CNC的工作时长等各方面因素。问题的提出具有实际的对工业生产、绿色经济和可持续发展的指导意义。选择一种有效RGV的动态调度策略，可以减少系统的空跑空耗，提高复合作用率且能降低能耗，实现高效经济的大规模工业生产。

1.2 问题重述

智能加工系统的作业任务分为一道工序、两道工序和CNC故障三种具体情况。一道工序的物料加工可由每台CNC独立完成；两道工序的物料加工需要两台安装不同刀片的CNC依次完成；考虑随机故障的情况时，故障发生概率为1%，排除故障（未完成的物料报废）时间为10-20分钟，故障CNC排除问题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情况，完成下列两项任务：

1.2.1 对一般问题进行研究，给出RGV动态调度模型和相应的求解算法；

1.2.2 利用表1中系统作业参数的3组数据分别检验模型的实用性和算法的有效性，给出RGV的调

度策略和系统的作业效率，并将具体的结果分别填入附件 2 的 EXCEL 表中。

附加说明：表 1 给出了 RGV 移动单位所需时间，CNC 完成各个工序所需时间，RGV 为奇数、偶数编号一次上下料所需时间；附件 2 要求填入一道工序、两道工序以及考虑故障时，CNC 对应不同参数组的上料开始时间和下料开始时间。

二、问题分析

2.1 问题一的分析

与生活中随处可见的排队行为类似，在 RGV 的动态调度过程中也随处可见排队现象。当 RGV 正忙无法及时响应所有上下料请求时 CNC 工作台等待 RGV 将自动形成排队队列，RGV 从队列中选择 CNC 进行服务。选择几组参数可以建立起关于这个排队系统模型的描述，与一般的排队系统不同的是，排队队列无须遵循先入先出原则。确定队列中接受服务的先后顺序的规则或者构造优先队列的规则即是 RGV 的动态调度策略。要分析 RGV 的动态调度，借助排队论思想，我们将这一问题转化为排队系统优先队列的构造问题。顾客流输入排队系统不是泊松过程，服务时间也不服从指数分布，除了考虑 1%的故障事件是随机性现象之外，调度策略一旦确定系统每一时刻的状态也将随之确定。给定初始条件，给定动力过程（即是指调度策略），可以通过实现基于排队论的元胞自动机来模拟仿真排队系统每一时刻的状态。在最初的实现中，只考虑一道工序和不故障的情况，之后逐步加入随机故障、两道工序不带随机故障和两道工序带入随机故障。排队论模型转化寻找 RGV 的动态调度策略问题，基于排队论设计的元胞自动机模型完成具体条件下系统的仿真。

设计优先队列是优化动态调度的关键所在，受贪心算法思想的启发，在此基础上同时建立预测机制，所设计的基于贪心算法的预测算法既要能较好地弥补贪心算法无法保证全局最优的缺点，又要能很好地考虑实际的生产情况。一道工序时使用单条排队队列，两道工序时使用双排队队列，基于 Java 语言面向对象地完成元胞自动机的设计和实现，在一道工序的基础上逐步加入故障和两道工序的因素，导出实验数据，作为模拟仿真的实验结果。

然而，基于排队论设计的元胞自动机在面对两道工序时如何安排不同工序 CNC 工作台比例和空间布局问题上将显示出不足，原因在于优先队列的设计不同工序时是不同考虑的，即是说一道工序和两道工序的设计必须严格区分开来，使得排队论元胞自动机考虑两道工序时对工作台配置和布局应当如何最优鞭长莫及，较难在预测算法的基础上得到一组最优的解。

将问题进一步抽象，借助图论的思想和数学工具，可以将 RGV 的动态调度过程等价成 RGV 在一个图模型中按照一定策略游走的过程。前述图模型可以考虑使用 8 阶完全图，对应一道工序时的描述，即将 CNC 抽象成结点，RGV 的移动抽象成边，RGV 游经各个结点之后可以抵达任意的其他结点，RGV 的动态调度策略即是图模型中的 RGV 游走策略。进入两道工序的领域时，基于工序 1 与工序 2 之间存在依赖关系的考虑，原来的完全图应当相应变更为二部图，即是分别处理工序 1 和工序 2 的 CNC 构成的集合之间产生无向联边。要统一一道工序和两道工序的两种情况，就必须找到两种情况的相同之处，容易发现，在图论的框架内，这两种情况可以有同一种表述形式，即是矩阵。矩阵和图之间存在一一对应关系，无论是完全图还是二部图，都可以被矩阵编码成矩阵空间中的统一元素，从而一道工序和两道工序无须严格区分，但是需要有另一个元胞自动机来实现这一过程。在前述建立起的图模型基础上设计元胞自动机用于模拟 RGV 在模型中的游走，建立一个指标用来衡量系统的生产效率，通过矩阵编码方法穷举所有的 CNC 比例和空间布局的组合，寻找到最优的一种配置和布局。基于图模型的元胞自动机能够返回基于排队论的元胞自动机需要的空间布局信息，辅助基于排队论的元胞自动机采集到最优的解。基于图模型的元胞自动机拟基于 python 语言面向过程地实现。

综合上述分析，对于问题一，既考虑使用排队论建立起排队模型用来描述 RGV 的动态调度过程，又考虑使用元胞自动机模拟排队系统的运行；既考虑使用图论建立起图模型用来统一一道工序和两道工序的不同情况，又考虑使用基于图模型设计的元胞自动机模拟 RGV 在图上的游走。问题一为 RGV 动态调度的一般问题，排队模型和图模型可用于对同一问题的不同表述，基于排队论设计的元胞自动机和基于图模型设计的元胞自动机可用于对同一问题的不同模拟，只是在针对问题的具体细节上，这两套理论各自具有各自的

优缺点。组合使用上述分析中提到的拟采用的数学模型和元胞自动机，或将采集到较好的实验结果。

2.2 问题二的分析

按照前述分析，问题 1 中拟建立排队模型和基于排队模型拟设计第一套元胞自动机，拟建立图模型和基于图模型拟设计第 2 套元胞自动机。对于问题 2，即是检验上述模型的实用性和检验借鉴贪心思想并加入预测机制的算法的有效性。实用性和有效性检验需要有一个基准对比的仿真测试结果，仿真软件 Plant Simulation 可以提供我们所需的基准测试数据。定义系统利用率作为对系统生产效率的衡量指标，带入三组参数，图模型元胞自动机将返回三组最优利用率和最优空间布局，排队模型元胞自动机使用前述预测算法计算得到一道工序、一道工序带入随机故障两组实验的数据，数据填入 excel 表格中；使用预测算法，结合图模型元胞自动机返回的信息计算得到两道工序、两道工序带入随机故障两组实验的数据，数据写入相应 excel 表格。

RGV 的动态调度策略即是前述预测算法，系统的作业效率即是前述定义的系统利用率指标。这两项都将由建立的模型和算法给出。

三、基本假设

3.1 排队论模型中的假设

3.1.1 顾客流的输入在单位班次内是有限的，顾客是逐个逐个到达的

3.1.2 顾客到达时如果 RGV 服务台被占用，顾客将进入排队队列

3.1.3 排队队列优先规则为 RGV 选择预期能够最先完成上下料的 CNC

3.1.4 一道工序的排队方式为单个排队队列，两道工序的排队方式为两个排队队列

3.2 图论模型中的假设

3.2.1 RGV 在图的结点上时，只能执行上下料或者清洗熟料操作。RGV 在图的边上时，只能执行移动或者等待操作。CNC 能且只能处以图的结点上。当 CNC 在图的结点上时，只能执行上下料、等待或者加工生料操作。

3.2.2 结点的权值为上下料和清洗熟料的时间开销，边的权值通过具体的计算得出

3.2.3 RGV 在游走过程中从边进入结点之后将计算下一结点

3.2.4 结点之间要么不产生联边，要么必定是无向边

3.2.5 图模型用于描述一道工序时，图的形态为 8 阶完全图；用于描述两道工序时，图的形态为二部图

四、符号说明

4.1 用于描述排队论模型的符号

4.1.1 使用一组参数来描述建立起来的排队系统 Q：

Q	排队系统
X	CNC 顾客相继到达的间隔时间的统计分布
Y	顾客接受服务的时间的统计分布
D	系统中服务台个数
M	系统 Q 的容量限制
C	顾客源数目
R	服务规则

4.1.2 使用另一组参数描述排队系统 Q 的服务性能指标：

L_s	服务队列长度，即是指接受服务台服务的顾客数

L_q	队列长度，即是指队列中的顾客数
$L=L_s+L_q$	队列总长，即是指系统中的顾客总数
W_s	逗留时间，即是指顾客在服务中的等待时间
W_q	等待时间，即是指顾客在队列中的等待时间
$W=W_s+W_q$	总时间，即是指顾客在系统中的总停留时间
TB	忙期，即是指服务机构两次空闲的时间间隔
S	稳态，即是指系统运行充分长时间后，初始状态的影响基本消失，系统出现相对稳定的状态

4.2 描述图模型过程中使用到的的符号和函数

G	8 阶无向加权完全图
N	图的结点集
E	图的边集，为结点集的笛卡尔积
W_N	结点上的权值构成的集合
W_E	边上的权值构成的集合
$t\_load(N_i)$	结点 N_i 上下料的时间开销
t_clean	熟料清洗的时间开销
$d(N_i,N_j)$	结点 N_i 和结点 N_j 之间的距离，由给定的参数和空间距离确定
$r(N_j)$	距离完成本轮生料加工结点 N_j 还需要的时间开销
T	单位班次时长
path	单位班次内 RGV 游走形成的通路
$P_i, i = 1,2,...,8$	通路中经过的结点
n	一个班次完成的成料
S_1	决定下一轮游走的目标结点的策略
C	系统空耗函数
F	完成第一道工序的 CNC 工作台构成的结点集
S	完成第二道工序的 CNC 工作台构成的结点集
E	边集，为 F 和 S 笛卡尔积
W_F	点集 F 每个元素上的权值构造的集合
W_S	点集 S 每个元素上的权值构造的集合

五、模型的建立与求解

5.1 任务一模型的建立与求解

与乘客们排队进入地铁和车站，顾客排队购买商品的行为相类似，在 RGV 的动态调度过程中同样存在排队现象。或者是 CNC 工作台完成本轮次生料加工等待 RGV 上下料，或者是 RGV 完成熟料清洗等待 CNC 工作台发出上下料请求信号，这些等待过程都可以使用排队论语言进行描述。智能制造系统是一个精确的控制系统。而当我们将整个智能制造系统抽象成排队系统时，服务相当于 RGV 的上下料操作；RGV 相当于是单服务台，响应 CNC 工作台的上下料请求信号；CNC 工作台相当于是顾客，进入排队系统接受或者等待接受服务台的服务，在服务台无法及时响应所有工作台的服务请求时 CNC 工作台自动排成队列，但是此处的队列未必遵循先入先出原则。我们考虑的排队系统与一般的排队系统不同，顾客进入排队系统不再是泊松过程，顾客接受服务的时间也不再服从指数分布，除了低概率的 CNC 工作台故障事件具有随机性之外，整个排队系统各个组件都以确定无疑的节奏完成各自的工作，这就意味着任意时刻的状态都可以被计算得知，唯一可变之处只有工作台排成队列后 RGV 的选择策略，而选择策略一经确定随后各个时刻的状态也都随之

确定。RGV 的动态调度过程即等价于制定排队系统中队列的出队列规则过程。

5.1.1 排队论用于分析 RGV 动态调度一般问题

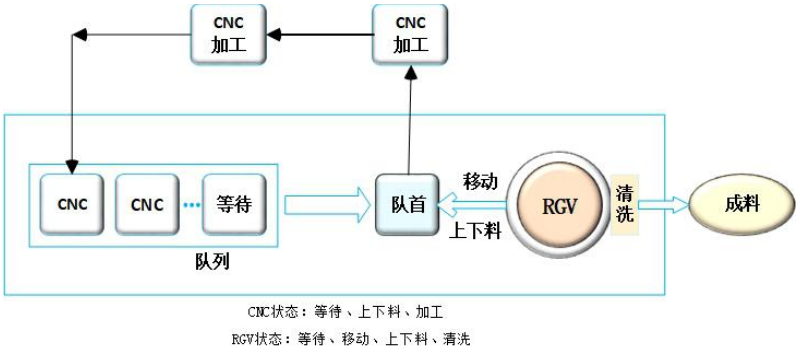


图 1 - 基于排队论建立的模型

如图 1 所示，利用 4.1 中各项参数建立起一个用于描述 RGV 动态调度问题的排队模型：确定型排队系统 Q，配备有单服务台，每次进入服务队列的顾客数目为 1，每次进入排队队列的顾客数目至多为 8（在我们考虑的问题中，只有 8 台 CNC 工作台），Q 的容量有限，顾客源有限，顾客相继到达的时间间隔的分布和 RGV 服务顾客的时间的分布都是可以通过仿真软件利用计算机模拟进行统计。

如前所述，RGV 的动态调度问题等价于服务规则的选择问题，服务规则决定了队列中顾客被服务的先后顺序，这一模型不同于一般的排队系统，不遵循先入先出规则。设定服务规则为随机，使用仿真软件模拟排队系统的运行，采集一批模拟数据，能够帮助直观上理解随机规则排队系统的服务性能或服务质量。要使得排队模型的服务性能得到提升，很关键的一步是设计一组合理的规则，既要考虑实际生产环境下系统的运行，也要考虑理论上参数的优化改进。而针对具体系统、具体规则，则需要有一套具体的、定向设计的仿真工具用来采集模拟数据和评估模型性能，5.1.2 中介绍的基于排队论的元胞自动机正是出于这一目的而设计实现的。

排队系统Q的各项属性：

A. 输入过程:CNC顾客到来时间的规律性，包含如下情况

1. 顾客的组成是有限且固定的（单位班次）
2. 顾客到达的方式是逐个逐个的
3. 顾客到达是相互独立的，即以前的到达情况对以后的到达没有影响
4. 输入过程按照优先规则是确定的

B. 排队规则

1．等待制：当顾客到达时，如果RGV服务台被占用，顾客就排队等待，直到接受完服务才离去

2．一道工序的排队方式为单个排队队列，两道工序的排队方式为两个排队队列

C. 服务过程

1．服务机构：RGV单服务台

2．服务规则：优先服务，优先给RGV能够最先完成上下料的CNC服务

5.1.2 基于排队论设计的元胞自动机

元胞自动机(Cellular Automaton，简称CA)是一种时空离散的局部网格动力学模型，是复杂系统研究的一个典型方法，特别适合用于空间复杂系统的时空动态模拟研究。与数学物理方法不同，它不使用具体的解析函数，而是采用规则来描述系统的状态，用规则取代数值计算，能有效研究并描述智能RGV加工系统的运作过程及调度效果。

智能RGV加工系统中相关名词的含义：

1. 元胞：RGV以及每台CNC均作为一个元胞

2. 元胞状态：RGV小车分为繁忙（包括上下料以及清洗）、移动、空闲三种状态，CNC分为加工、空闲、上下料三种状态。根据问题，CNC在两道工序中有两种不同类型，同时CNC在部分具体情况中还会考虑故障这一状态。
3. 状态更新规则：预测目标最优化原则、提前到达原则、贪心服务原则、坚定目标原则、以概率P随机故障原则、工序连贯原则

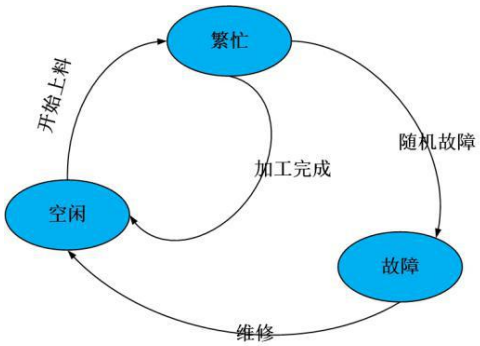


图 2 - CNC的有限状态机

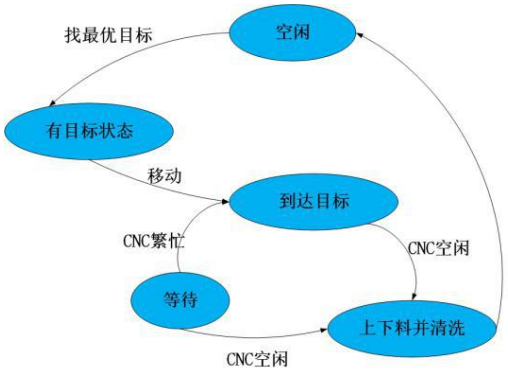


图 3 - RGV的有限状态机

企业是以营利为目的的组织。在智能RGV加工系统性能好坏的诸多描述参数（成品数量、RGV平均等待时间、RGV移动时间、CNC平均等待时间）中，成品数量是直接反映企业盈亏从而体现加工系统优劣的一个参数。因此，我们考虑用成品数量作为智能RGV加工系统的主要描述参数。

调度算法对一个智能加工系统来说是至关重要的。在考虑排队队列中CNC接受服务先后顺序时，可以通过预测CNC状态为RGV选择优先服务的对象，同时可以考虑让RGV提前抵达需要优先服务的CNC所在的位置。通过元胞自动机仿真这一复杂的过程，再借助问题给出的表1，即可以得出相应模型和算法下该加工系统的作业效率，从而检验模型的实用性以及算法的有效性。

RGV与目标CNC之间只有两种状态：RGV等待CNC或者CNC等待RGV。设RGV运行至CNC#i的时间为t(i)，RGV为CNC#i上下料耗时t_load，当前时间current，CNC#i工作结束时间t_stop。两种情况下从当前时刻到为目标CNC上下料完成需要时间如下：

如果RGV等待CNC，则从当前时刻到为目标CNC上料完成需要时间(t_stop - current + t_load)

如果CNC等待RGV，则从当前时刻到为目标CNC上料完成需要时间 (t(i) + t_load)

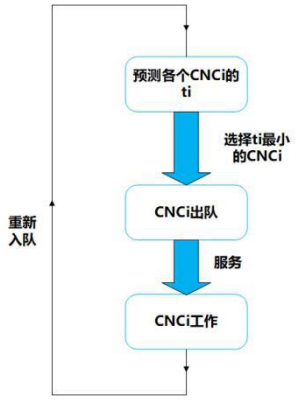


图 4 - CNC的排队规则

因此，RGV从当前时刻到为目标CNC上料完成需要时间:

t_load_stop = max(t_stop - C + t_load, t(i) + t_load)

由于同种CNC的工作效率都一样，所以运用贪心的思想，为了让成料数量最大化，必须尽快让CNC工作起来，也就是说，优先队列的优先服务规则是：t_load_stop越小，CNC#i优先级越高。同时，上述选择最优服务目标运用了预测思想，RGV寻找目标，并不只是从空闲状态的CNC中找，而是预先考虑每一台CNC的完成时间，从而选择t_load_stop最小的。

5.1.3 RGV动态调度模型的构建

5.1.3.0 参数表

RGV移动1个单位所需时间	M1
RGV移动2个单位所需时间	M2
RGV移动3个单位所需时间	M3
CNC加工完成一个一道工序的物料所需时间	A
RGV为CNC1#，3#，5#，7#一次上下料所需时间	Lo
RGV为CNC2#，4#，6#，8#一次上下料所需时间	Le
RGV完成一个物料的清洗作业所需时间	W
CNC 加工完成一个两道工序物料的第一道工序所需时间	B1
CNC 加工完成一个两道工序物料的第二道工序所需时间	B2

5.1.3.1 考虑一道工序的情况

在这种情况下，CNC工作台只有一种类型，RGV需要服务的对象只有一种，故排队方式为单个排队队列排列。每次优先服务时，根据前述排队系统Q的服务规则，只考虑从该队列中选择最优的服务对象。

我们通过面向对象的思想，将复杂的问题抽象化，设计相应的求解过程：

I	将所有CNC入队，构建优先队列
II	# RGV根据下式选择最优目标 # 表示RGV会优先选择t_load_stop小的CNC choice（RGV） = k
III	被选中的CNC出队接受服务后重新入队排队
IV	重复II、III过程直到时间结束（一个班次8小时）

基于Java语言的面向对象编程，给出了一道工序无随机故障情况下RGV的动态调度模型代码（源代码参见附录），代码流程图如下所示：

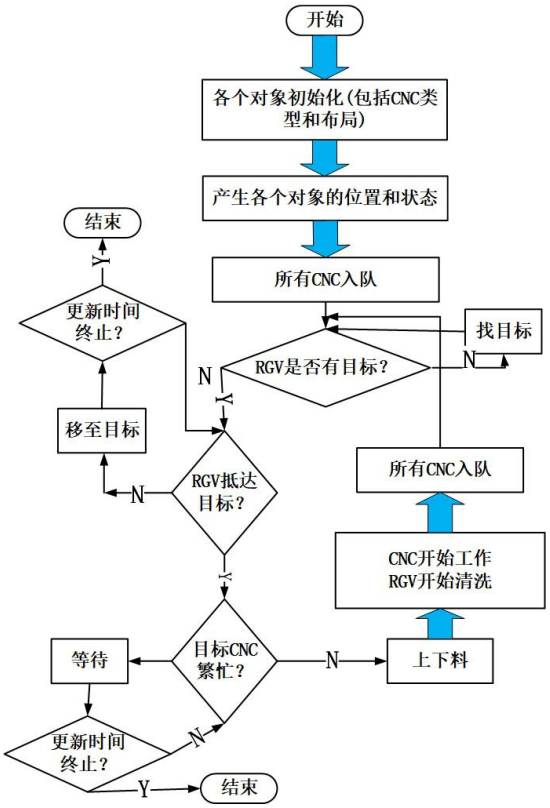


图 5- 一道工序时的算法流程图

在无故障单工序情况下，输入加工系统的相关参数，我们便可以得到该调度算法下物料的加工情况，从而分析检验模型的实用性和算法的有效性。

5.1.3.2 考虑两道工序的情况

这种情况下，CNC有两种类型，RGV需要根据自己装载的物料选取合适的CNC服务对象，故排队方式为1两个排队队列排列。每次最优服务时根据RGV装载的物料选择一条队列，根据排队系统Q的服务规则，从中选择最优的服务对象CNC。

与一道工序时情况类似，相应的求解过程如下：

I 我们通过将所有CNC根据类型分别入队，构建优先队列

II 设Q1为只能执行工序一的CNC队列，Q2为只能执行工序二的CNC队列

```
if RGV装载的物料需要在工序一加工，then
    # 表明当RGV装载的物料需要在工序1加工时
    # RGV会从队列Q1里选择t_load_stop最小的CNC
    choice（RGV） = k
if RGV装载的物料需要在工序二加工，then
    # 表明当RGV装载的物料需要在工序2加工时
    # RGV会从队列Q2里选择t_load_stop最小的CNC
    choice（RGV） = k
    # 因此RGV会根据装载的物料选择合适的队列服务
```

III 被选中的CNC出队接受服务后重新入队排队

IV 重复II、III过程直到时间结束（一个班次8小时）

基于Java语言面向对象编程，给出两道工序无随机故障情况下RGV的动态调度模型代码（源代码参见附

录），代码流程图如下所示：

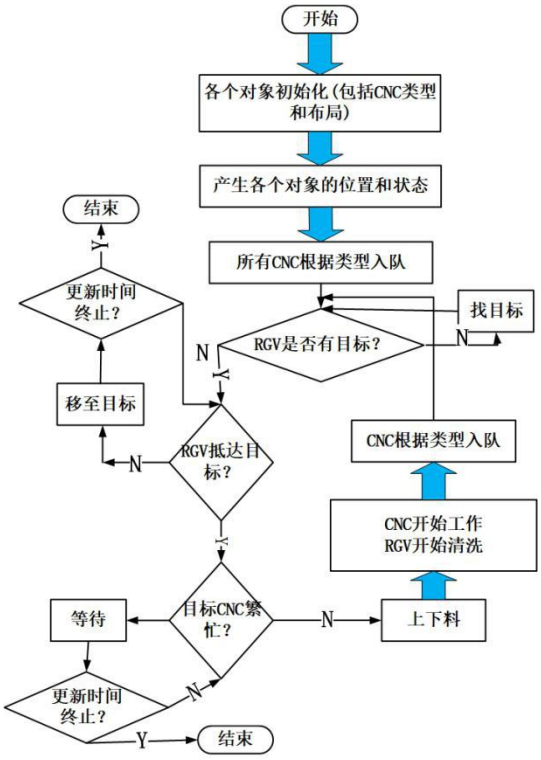


图 6 - 两道工序时的算法流程图

在两道工序无随机故障的情况下，输入加工系统的相关参数，我们便可以得到该调度算法下物料的加工情况，从而分析检验模型的实用性和算法的有效性。

5.1.4 考虑可能发生故障的情况

5.1.4.1 考虑一道工序的情况

在排队的过程中，CNC会以一定的概率P随机地发生故障，也就是说，相比一道工序无随机故障的情况又多了一种排队规则：随机后退规则。同时，由于故障发生后，未完成的物料将报废，因此，在CNC顾客后退的同时，上一次的服务也会失效。

```
# 算法过程与无故障第一道工序唯一不同的是CNC在工作的过程中会随机出现故障并更新队列
# 相应的求解过程：
    I      将所有CNC入队，构建优先队列
    II     # RGV根据下式选择最优目标
           # 表示RGV会优先选择t_load_stop小的CNC。
           choice（RGV） = k
    III    被选中的CNC出队接受服务后重新入队排队，每次入队有1%的概率出现故障，
           故障后进入维修状态，t_load_stop 增大，该CNC后退。
    IV     重复II、III过程直到时间结束（一个班次8小时）
```

基于Java语言面向对象编程，给出了一道工序带入随机故障情况下RGV的动态调度模型代码（源代码参见附录）。

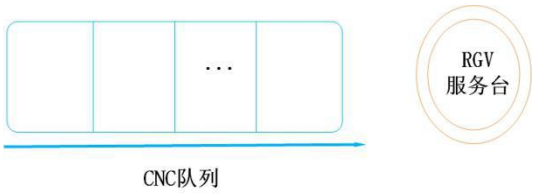


图 7 - 单队列排队规则

一道工序带入随机故障的情况下，输入加工系统的相关参数，我们便可以得到该调度算法下物料的加工情况，从而分析检验模型的实用性和算法的有效性。

5.1.4.2 考虑两道工序的情况

与上面分析类似，该情况相比无故障两道工序的情况多了一种排队规则：随机后退规则。同时，由于故障发生后，未完成的物料将报废，因此，在CNC顾客后退的同时，上一次的服务也会失效。

算法过程与无故障第二道工序唯一不同的是CNC在工作的过程中会随机出现故障并更新队列相应的求解过程：

I 将所有CNC根据类型分别入队，构建优先队列

II 设Q1为只能执行工序一的CNC队列，Q2为只能执行工序二的CNC队列

if RGV装载的物料需要在工序一加工，则

choice（RGV） = k| 5

if RGV装载的物料需要在工序二加工，则

choice（RGV） = k| 6

III 被选中的CNC出队接受服务后重新入队排队，CNC每次入队有1%的概率出现故障，故障后进入维修状态，ti增大，该CNC后退

IV 重复ii、iii过程直到时间结束（一个班次8小时）

5式表明当RGV装载的物料需要在工序一加工时，RGV会从队列Q1里选择ti最小的CNC

6式表明当RGV装载的物料需要在工序二加工时，RGV会从队列Q2里选择ti最小的CNC

因此RGV会根据装载的物料选择合适的队列服务。

基于Java语言面向对象编程，给出了一道工序带入随机故障情况下RGV的动态调度模型代码（源代码参见附录）。

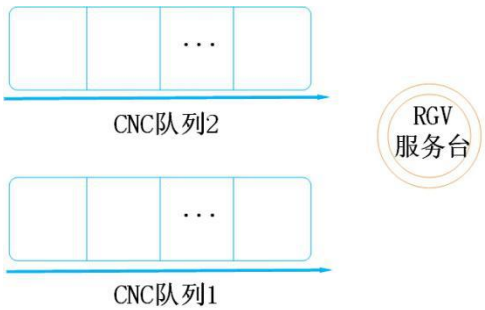


图 8 - 两条队列时的排队规则

两道工序带入随机故障的情况下，输入加工系统的相关参数，我们便可以得到该调度算法下物料的加工情况，从而分析检验模型的实用性和算法的有效性。

5.1.6 图论和图模型用于优化两道工序时的工作台配置和布局

图论是组合数学的一个分支，主要研究图对象和图模型。最早的关于图论的叙述来自 18 世纪的大数学家欧拉的一篇论文，这篇论文重述了当时颇为著名的格尼斯堡七桥问题，并创造性地提出格尼斯堡问题可以抽象成图来解决，巧妙地证明了格尼斯堡七桥问题的无解性，欧拉因此成为图论的创始人。图论的思想和模型、图算法广泛应用于不同的科学和工程领域，诸如利用图论分析传染性疾病的流行模式，分析计算机病毒在网络中的散播，建立车辆道路交通流的简化模型，优化天然气和石油的管道运输等等。而在我们考虑的问题中，图论和图模型主要是用来确定两道工序时不同工序工作台的数目比例和空间分布，以下统一将确定比例的过程称配置，将确定空间分布的过程称布局。

5.1.6.1 建立图模型

在智能制造系统中，RGV 存在移动、上下料、等待和清洗四种状态，CNC 工作台存在加工、等待、上下料三种状态，考虑 RGV 和 CNC 不同的状态组合和变化是一个相对复杂的问题，将之抽象成动态图或者动态网络，基于这一网络实现元胞自动机，模拟动态演化才是比较好的选择。不妨用图的结点来表示 CNC 工作台，用 RGV 在图中的游走来表示 RGV 在一维轨道上的移动和在 CNC 工作台前的等待。借助于图论的抽象方法、图论的数学工具和其形式化语言的重述分析，RGV 的动态调度过程将被抽象和理解为 RGV 在一个边权值动态变化的无向加权图中的游走问题，从而可以搭建起针对 RGV 动态调度的图模型。在后来的分析中，这种抽象方法逐渐显现出一些优点，比如很高的可扩展性，即是说无论考虑一道工序还是考虑两道工序，图模型都无须作大的改变就能获得较好的描述效果。

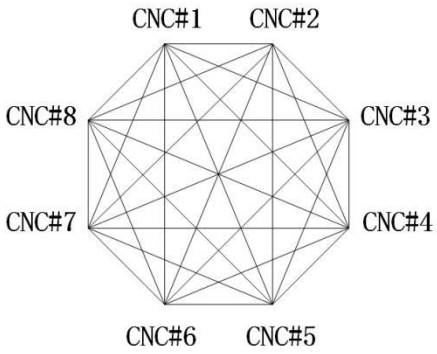
定义智能制造系统的利用率：

$$rate = \frac{\sum_{i=1}^{8} t_{work}(i)}{t_{total}}$$

其中分子项表示 8 台 CNC 工作台处以加工生料状态的时间总和，分母项表示系统从开启到关闭的时间间隔，直观上容易理解，一个班次内 CNC 工作台工作时间越长，所得到的熟料越多，清洗之后得到的成料也越多，这个比值衡量了系统的制造能力，即单位班次系统的生产效率。

在假设的前提下，可以认为智能制造系统的实际生产周期内系统运行的任何一个状态和过程，都能够找到与模型中的结点、边或 RGV 之间的映射关系。实际生产环境中，RGV 在一维轨道上移动，响应来自 CNC 工作台的请求信号，为结束本轮加工的 CNC 工作台提供下一轮加工需要的生料，取走 CNC 工作台上的熟料并清洗，最后将清洗好的成料放置在下料传送带上，经历一次完整的动作周期。而加工完生料的 CNC 工作台只需要等待 RGV 取走熟料并为自己上料便可以进行下一轮的生料加工。使用图模型的描述语言，上述过程即是 RGV 在结点之间游走，与结点进行交互，根据结点的权值信息决定停留时间，同时在特定条件下计算下一轮游走的目标结点。系统单位班次内，RGV 的游走将产生图上的一条通路，这条通路可能重复经过某些结点或者某些边，记录了游走的结点顺序，一般地，这条通路越长，所获得的成料也越多，系统利用率也越高。

1. 8 阶无向加权完全图的定义



$$G=(N,E,W_N,W_E)$$

$$N=\{cnc\#1,cnc\#2,cnc\#3,cnc\#4,cnc\#5,cnc\#6,cnc\#7,cnc\#8\}$$

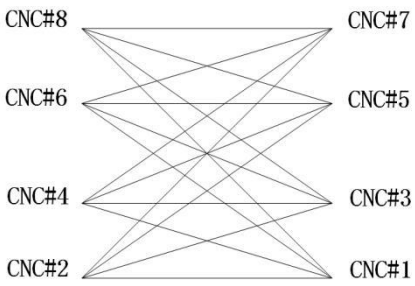
$$E=N\times N$$

$$W_N=\{w_i|w_i=t_{load}(N_i)+t_{clean},i=1,2,...,8\}$$

$$W_E=\{w_{ij}|w_{ij}=|d(N_i,N_j)-r(N_j)|,i,j=1,2,...,8\}$$

边上权值的计算方法借鉴了贪心算法的思想，描述了确定下一轮游走的目标结点的策略，其中，$d(N_i,N_j)$ 度量了两个结点 N_i 和 N_j 之间互相抵达需要花去的时间开销，$r(N_j)$ 度量了结点 N_j 代表的的 CNC 工作台距离完成当前轮次的生料加工还需要花去的时间开销，$|d(N_i,N_j)-r(N_j)|$ 度量了 RGV 和 CNC 工作台的相对等待时间。如果 $d(N_i,N_j)-r(N_j)$ 是正数，那么表示在结点之间转移的时间开销大于下一结点完成当前轮次的加工过程所需要的时间开销，从而 RGV 抵达下一结点时 CNC 工作台处以等待状态，造成系统空耗；如果 $d(N_i,N_j)-r(N_j)$ 是负数，那么表示转移开销小于下一结点完成当前轮次所需的开销，RGV 将等待 CNC 工作台完成加工过程，RGV 空耗但是系统不空耗。考虑 RGV 和 CNC 之间的相对等待时间，尽可能避免出现 CNC 长期空等 RGV 或者 RGV 长期空等 CNC 的情况，有助于提高系统的利用率。

2. 二部图的定义



$$G=(F，\ S,E,W_F,W_S,W_E)$$

$$F,S\in N,|F||S|\geq 1,|F||S|=|E|$$

$$E = F \times S$$

$$W_F = \{w_i | w_i = t_{load}(F_i) + t_{clean}, i = 1,2,...,8\}$$

$$W_S = \{w_i | w_i = t_{load}(S_i) + t_{clean}, i = 1,2,...,8\}$$

$$W_E = \{w_{ij} \left| w_{ij} = \left|d(F_i, S_j) - r(S_j)\right| \ or\ w_{ij} = \left|d(S_i, F_j) - r(F_j)\right|, S_i, S_j \in S, F_i, F_j \in F\right.\}$$

在最优化配置和布局这一问题上，需要使用元胞自动机仿真二部图。仿照 8 阶无向加权完全图，可以给出二部图的定义。与考虑一道工序实现元胞自动机用于仿真测试系统在不同参数下的系统利用率不同，基于二部图实现的元胞自动机仿真测试了所有的比例和空间分布组合下系统的利用率，目的是找到在不同参数下最优的配置和布局。

3. RGV 游走经过的结点和边构成的路径

$$Path = p_1 p_2 \cdots p_i p_j \cdots p_n, where\ p_i \in N$$

$$P_1 \to P_2 \to P_3 \to \cdots \to P_i \to P_j \to \cdots \to P_n\ where\ P_i \in N$$

存在一种最优的 RGV 动态调度策略使得系统的利用率最高，等价于图 G 中存在一条通路 Path，使得位于通路上的边的权值总和尽可能小，路径长度尽可能大。

4. 单位轮次生产的成料数量的上限

$$T \geq n(t_{load} + t_{clean}) + \sum_{i=1}^{n-1} \max(d(P_i P_{i+1}), r(P_{i+1}))$$

$$n \leq \frac{T - \sum_{i=1}^{n-1} \max(d(P_i P_{i+1}), r(P_{i+1}))}{t_{clean} + t_{load}}$$

5. 选择相对等待时间最短的策略和这种策略下的系统空耗函数

$$S_1 = Node_i \to Node_j, if\ w_{ij} = \min(W_E)$$

$$C = \sum_{i=1}^{n-1} w_{P_i P_{i+1}} = \sum_{i=1}^{n-1} \left|d(P_i P_{i+1}) - r(P_{i+1})\right|$$

6. 单纯选择抵达时间最短的策略和这种策略下的系统空耗函数

$$S_2 = Node_i \to Node_j, if\ d(i,j) = \min\{d(i,k), k = 1,2,...,8\}\ and\ r(P_{i+1}) = 0$$

$$C = \sum_{i=1}^{n-1} w_{P_i P_{i+1}} = \sum_{i=1}^{n-1} d(P_i P_{i+1})$$

5.1.6.2 基于图模型实现的元胞自动机

在计算机科学领域，元胞自动机因其良好的可并行特性较常被视为并行计算机而参与并行计算的研究。除此之外，元胞自动机作为框架式的动力学模型，还经常被用作电磁场模拟、热力学过程模拟、洋流模式

等。参照 5.1.5.1 建立图模型过程中所作的假设，设计元胞的运行规则，依据这一系列规则和如图所示的有限状态机基于 python 语言实现基于图模型的元胞自动机。

规则 1：CNC 结点元胞只有 3 种状态，RGV 元胞相当于网络中传递的令牌，只有接到令牌才能进行上下料操作，否则要么在等待，要么在加工生料

规则 2：RGV 元胞在任意时刻只能位于边上或者点上，位于边上时处以移动或者等待状态，位于点上时处以上下料或者清洗熟料状态

规则 3：系统每秒刷新一次，各项状态转移条件均参考计时器

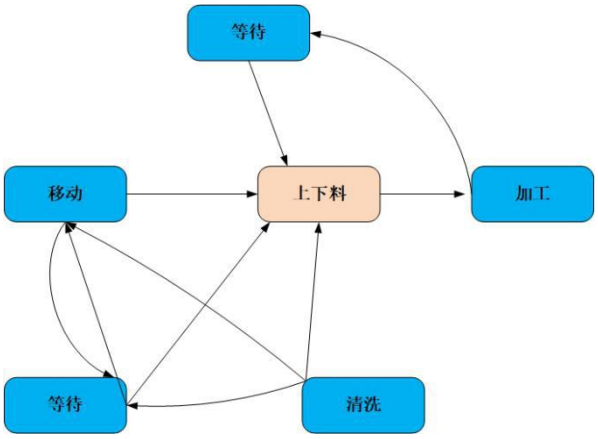


图 9 - 元胞自动机设计依据的有限状态机

基于图模型设计的元胞自动机不直接模拟 RGV 和 CNC 在制造系统中的动作，而是模拟 RGV 在 CNC 构成的动态网络中的最优游走，与基于排队论设计的元胞自动机不同，显然图模型元胞自动机更加抽象，但本质上两者不过都是同一问题的不同刻画形式，无非是一个问题的两种等价表述。

5.1.6.3 图模型在两道工序上的具体分析

假设处理工序 1 的 CNC 工作台构成集合 A，处理工序 2 的 CNC 工作台构成集合 B，要求

$$A \bigcap B = \varnothing \ and \ A \bigcup B = N$$

穷举集合 A 与 B 的所有可能情况即是穷举集合 A 与 B 对结点集 N 的所有可能划分。利用集合 A 与 B 即可以解决两道工序问题时的对最优配置和布局的全空间搜索。A 与 B 各自集合中包含的元素代表了两两不产生联边的结点，联边只在两集合之间产生。集合 A 与 B 一旦确定，网络的拓扑结构和结点之间的距离也就相应确定，实际编程发现，构造集合 A 与 B 的所有可能并计算相应的距离矩阵即可对 254 种情况进行有效编码。

事实上，二部图与完全图存在于同一个拓扑空间中，而这个空间中的元素都是可以通过编码空间（即放置距离矩阵的空间）进行编码（即映射）的，编码过程实际上统一了一道工序和两道工序的问题。改变一次拓扑结构，距离矩阵就需要计算一次，距离矩阵与拓扑结构存在一一对应关系，从而编码空间可以完备地编码一个解空间，进行搜索时只需要考虑距离矩阵就能相应地得到一个函数值而无须再进入拓扑空间单独考虑一道工序和两道工序的问题。完全图与二部图所在的拓扑空间等价于距离矩阵所在的编码空间，这两类图完全可以被距离矩阵统一起来，它们之间的关系图如下所示：

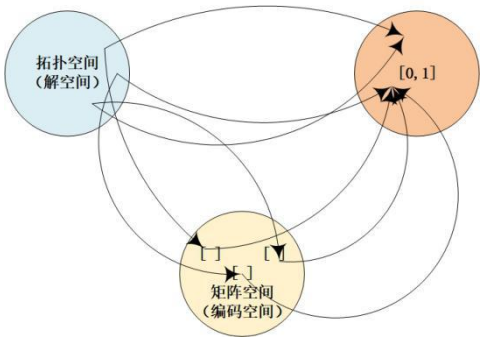


图 10 - 不同空间之间的映射关系

例如图 11 所示的二部图和图 12 所示的矩阵是等价的。我们寻找的解在拓扑空间内，而在元胞自动机的实现过程中拓扑空间中的元素都将被编码成矩阵。

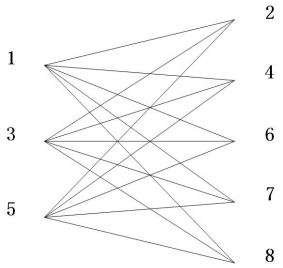


图 11 - 二部图

$$\begin{pmatrix} -1 & 0 & -1 & 1 & -1 & 2 & 3 & 3 \\ 0 & -1 & 1 & -1 & 2 & -1 & -1 & -1 \\ -1 & 1 & -1 & 0 & -1 & 1 & 2 & 2 \\ 1 & -1 & 0 & -1 & 1 & -1 & -1 & -1 \\ -1 & 2 & -1 & 1 & -1 & 0 & 1 & 1 \\ 2 & -1 & 1 & -1 & 0 & -1 & -1 & -1 \\ 3 & -1 & 2 & -1 & 1 & -1 & -1 & -1 \\ 3 & -1 & 2 & -1 & 1 & -1 & -1 & -1 \end{pmatrix}$$

图 12 - 二部图对应的矩阵

基于图模型的元胞自动机的高可扩展性即来源于编码空间对拓扑结构的强大描述能力。图模型在两道工序上的具体分析即等价于距离矩阵对二部图的编码分析。

5.1.6.4 基于图模型的元胞自动机在两道工序上的模拟仿真

与基于排队论的元胞自动机不同，出于快速找到两道工序时的工作台的最优配置和布局的目的，基于图模型的元胞自动机没有将 1%的故障率纳入考虑范围，尽管处理的是简化的问题，但仿真得到的模拟数据却并不失一般性。在两道工序问题中，处理不同工序的 CNC 工作台以不同的比例和不同的空间分布直接影响系统的利用率，下表给出了不同比例下不同空间分布的组合数目，基于图模型实现的元胞自动机搜索并计算了所有的可能情况，寻找使得系统利用率最高的不同工序 CNC 工作台的比例和空间分布。

比例	1:7	2:6	3:5	4:4	5:3	6:2	7:1	total
组合数目	8	28	56	70	56	28	8	254

通过模拟测试每组给定参数都采集到 254 组数据，每组数据包括系统利用率和空间分布情况。

5.1.6.5 工作台的最优配置和布局

使用三组给定参数，运行基于图模型的元胞自动机 python 代码，测试采集到一批实验数据，每组参数对应 254 种可能情况下的系统利用率和空间分布，将这些数据绘制成如下图所示的图表。纵向对比发现，比例 4：4 的系统测得的利用率曲线表现出较好的性能，1:7 的系统测得的利用率曲线表现出较差的性能。横向对比发现，在同一比例下测得的数据呈现波动的趋势，说明不同的空间分布会影响系统的利用率但不会是主要影响。分析可知，存在一种最优配置和布局使得系统利用率在这组参数下取得最大值。参考每组参数下的最优配置和布局改进基于排队论的元胞自动机的设计，有助于提升系统的利用率。基于图模型的元胞自动机即辅助基于排队论的元胞自动机寻找最优的配置和布局，这种组合充分利用图模型的高可扩展

性完成解空间搜索和排队模型的高可靠性完成模拟仿真。

实验得到的三组参数下的最优利用率和不同工序 CNC 工作台的最优配置和布局：

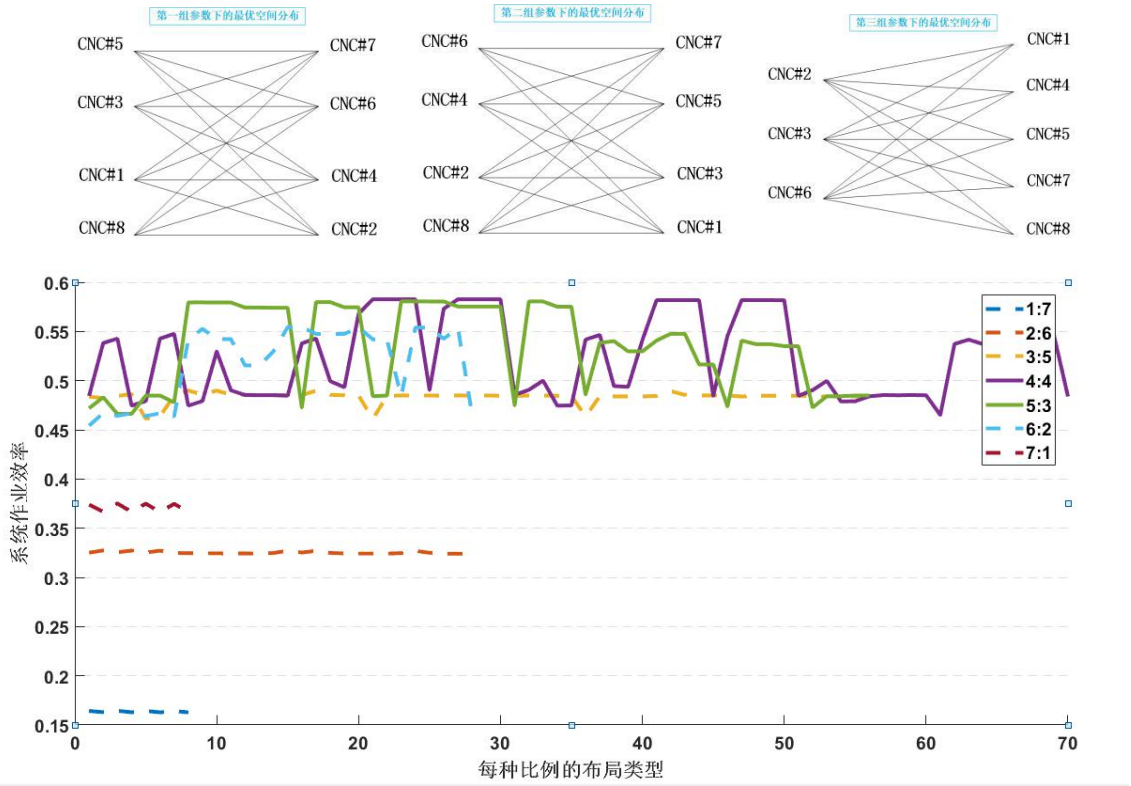


图 13 - 不同工序的 CNC 工作台比例和空间分布对系统利用率的影响

5.2 任务二：模型的分析检验

5.2.1 定性分析

A. **模型的实用性分析：**

四个模型均运用面向对象的思想，将复杂的问题抽象化，依据现实情况利用元胞自动机进行仿真，具有较高的真实性，适用于现实的大多数情况。因此，四个模型都具有较高的实用性。

B. **算法的有效性分析：**

首先通过预测算法得出各个CNC最早能接受服务的时间，然后再通过优先队列，利用贪心的思想选出最优服务对象，具有高度的时间利用率，减少了许多等待时间。因此，算法不仅具有有效性，还具有高效性。

5.2.2定量分析

由以上分析可知，产量是衡量RGV调度策略和系统优劣的主要描述参数，因此可以利用实际产量与理想产量的比值 η 来衡量RGV调度策略和系统的作业效率。即

$$\eta = \frac{\text{实际产量}}{\text{理想产量}}$$

其中，理想产量为CNC连续工作的成品数量：

$$\textit{理想产量} = \frac{\text{总时长} * 8}{\text{完成一个物料加工用时}}$$

其中8表示八台CNC连续工作

将三组参数输入到四个模型中可得各个模型对应数据的产量如下：

组数\模型	1	2	3	4

1	382	253	356	221
2	359	177	296	180
3	392	219	360	213

5.2.2.1 一道工序情况分析

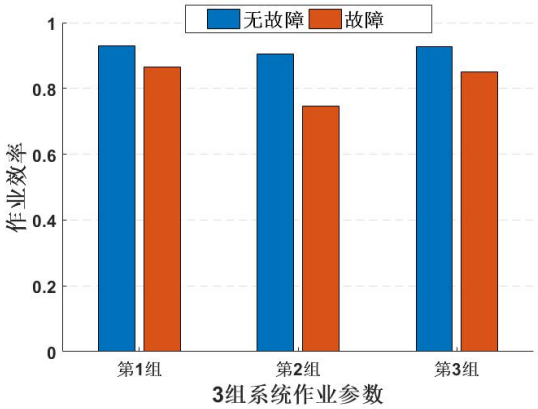


图 14 - 一道工序系统的作业效率

通过如图14所示的三组实验数据处理的结果可以发现，作业效率 η 最高可达92.9%，相较仿真软件得到的结果可知：在排队论框架下使用基于贪心算法的预测机制构造的优先队列具有较高的实用性及有效性。

5.2.2.2 两道工序情况分析

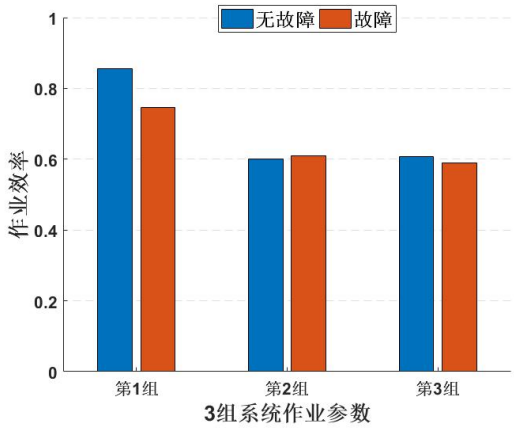


图 15 - 两道工序系统的作业效率

通过三组实验数据处理的结果可以发现，作业效率 η 最高可达85.4%，但是更多地分布在60%左右，可见在前述调度算法下，实验并没有达到较高的作业效率，很大程度上是由于两道工序导致RGV的等待时间变长。

六、模型的分析检验

6.1 结合实例分析的模型检验

针对问题一我们建立了 RGV 动态调度模型和相应的求解算法，然后利用原题表 1 提供的 3 组系统作业参数来检验我们的模型和算法是否正确。通过 3 组系统作业参数结合我们设计的模型和算法，系统达到了较高的作业完成量，并且这个结果符合实际情况。因此，我们认为，在一定误差范围内，我们的 RGV 动态调度模型具有很强的可靠性。

6.2 基于生产系统仿真模拟软件 Plant Simulation 的模型检验

如图 16 所示，我们通过 Plant Simulation 软件来模拟智能 RGV 加工系统，包括事件控制器，控制仿真运行的开始、停止、运行时间、仿真速度；工作区，完成物料的加工；发生源和传送带，负责为 CNC 输送生料；接收源，接收已完成清洗的熟料；搬运机器人，通过旋转自身机器臂，完成上下料操作；直线轨道以及编写好的程序和记录数据的表单。

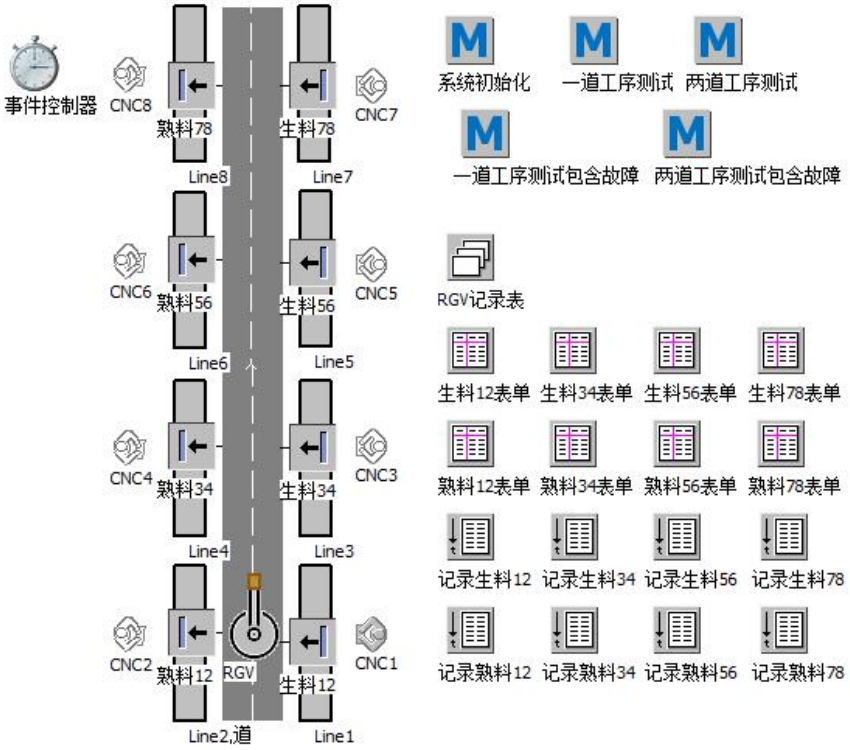


图 16 - RGV 动态调度仿真示意图

仿真过程包括一道工序测试、两道工序测试和故障对比组。为降低仿真过程中随机性所造成的误差，我们对每一组测试实验都进行 10 次重复试验并取其作业数量的平均值。得到一道工序和两道工序的实验结果如图 17, 18 所示

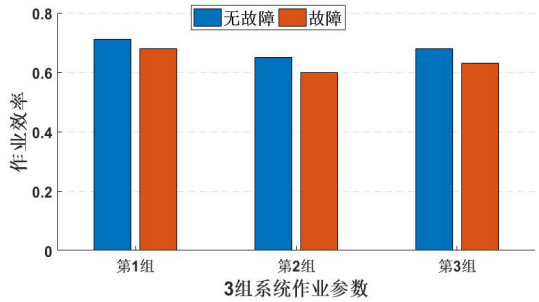


图 17 - 一道工序仿真实验结果

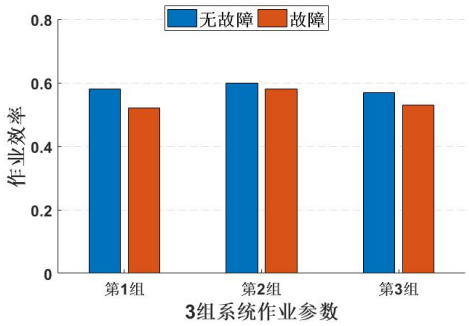


图 18 - 两道工序仿真实验结果

通过对比图 14,15 可以看出，基于排队论的元胞自动机很好地实现了我们设计的 RGV 动态调度策略，并得到了较好的实验结果，能够预见的是，使用我们设计的动态调度策略可以很大提升智能加工系统的作业效率，可为大规模工业生产带来可观的经济效益。

七、模型的评估与优化

7.1 模型评估

7.1.1 排队模型与基于排队模型的元胞自动机

排队模型可以很好地模仿 RGV 的动态调度过程，是对这一问题的近似和抽象，基于排队论的元胞自动机在制定规则时虽然考虑到了主要的实际情况，但是仍然有可能遗漏了一些生产过程中的细节。优先队列的设计实质等价于 RGV 动态调度策略的选择，在排队论的基础上设计元胞自动机，其可信度更高，真实性更强。

7.1.2 图模型与基于图模型的元胞自动机

图模型的主要优点在于很好地把一道工序和两道工序时的问题统一了起来，通过图编码的方式在解空间中寻找了全局最优的解，从而辅助基于排队论的元胞自动机提高了算法性能。基于图模型的元胞自动机是严格按照制定的基本运行规则实现出来的，从而可以确信这套模型具有较高的可靠性以及可信度。这套模型在模拟实际的 RGV 动态调度过程中表现良好，同时还具备较高的可扩展性。

7.2 模型优化

在建立图模型的过程中发现，RGV 在网络中定向游走时其选路策略仍存在进一步优化的空间，主要是因为边的权值计算方式只考虑了一种相对的情况，如前所述，实际上还可以通过判断权值取绝对值之前的正负性选择更优的策略，即是在某些情况下按照贪心选路的规则选路，某些情况下按照相对等待时间最短的规则选路。

八、参考文献

[1] 王树禾. 图论(第二版)[M]. 北京:科学出版社, 2009.

[2] 段晓东,王存睿,刘向东. 元胞自动机理论研究及其仿真应用[M]. 北京:科学出版社, 2012.

[3] 张兴强,汪滢,胡庆华. 交叉口混合交通流元胞自动机模型及仿真研究[J]. 物理学报, 2014(1):82-89.

[4] 孙健,丁日佳,陈艳艳. 基于排队论的单车道出租车上客系统建模与仿真[J]. 系统仿真学报, 2017(5):996-1004.

[5] 顾红,邹平,徐伟华. 环行穿梭车优化调度问题的自学习算法[J]. 系统工程理论与实践, 2013(12):3223-3230.

[6] 聂峰,程珩. 多功能穿梭车优化调度研究[J]. 物流技术, 2008(10):251-253.

[7] 叶其孝,江启源. 数学建模(原书第五版)[M]. 北京:机械工业出版社, 2014.

[8] 周金平. 生成系统仿真: Plant Simulation 应用教程[M]. 北京:电子工业出版社, 2011.

九、附录

9.1 无故障一道工序的 java 代码

```
import java.util.Queue;
import java.util.Scanner;

public class Simulation {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int finishKind = 2;
        int travelTime[] = new int [4];
        int workTime;
        int clearTime = 0, target = -1, cur = 1;
        int exchangeTime[] = new int [2];

        //boolean hasTarget = false;
        Scanner in = new Scanner(System.in);
        travelTime[0] = 0;
        for (int i = 1; i <= 3; i++) travelTime[i] = in.nextInt();
        workTime = in.nextInt();
        exchangeTime[1] = in.nextInt();
        exchangeTime[0] = in.nextInt();
        clearTime = in.nextInt();
        RGV rgv = new RGV(0, clearTime, target);
    /*  int CNC_1 = (8 * workTime[0]) / (workTime[1] + workTime[0]);
        int CNC_2 = 8 - CNC_1;
        int t1 = CNC_1 - 1;
        int t2 = CNC_2;*/
        int kind = 1;
        CNC cnc[] = new CNC[8];
        for (int i = 0; i < 8; i++) {
            cnc[i] = new CNC(0, -1, i + 1, workTime);
        }

        RGV.travelTime = travelTime;
        RGV.exchangeTime = exchangeTime;
        Time.cnc = cnc;

        Queue<CNC> CNC_Queue = new java.util.PriorityQueue(8);
        Queue<CNC> tmp = new java.util.PriorityQueue(8);
        for (int i = 0; i < 8; i++) {
            CNC_Queue.add(cnc[i]);
```

```
        }

        rgv.loadNew();
        while (true) {
            if (!rgv.hasTarget()) {
                rgv.setTarget(CNC_Queue.poll().getNum());
            }
            if (rgv.isReach() ) {
                if (!cnc[rgv.getTarget() - 1].isBusy()) {
                    rgv.exchange();
                    cnc[rgv.getTarget() - 1].startWork();
                    int preTarget = rgv.getTarget();
                    rgv.setTarget(-1);
                    if (rgv.getLoad().getKind() == finishKind) {
                        rgv.clean();
                        rgv.loadNew();

                    }
                    CNC_Queue.add(cnc[preTarget - 1]);
                     //?
                }
                else Time.go();

            }
            else rgv.run();


        }
    }

}



public class RGV {
    private int isBusy;
//  private int loadKind;
    private int clearTime;
    private int target;
    private Product load;
    static int cur;
    //private boolean hasTarget;
    static int exchangeTime[];
    static int travelTime[];
```

```
    public RGV(int isBusy, int clearTime, int target) {
        super();
        this.load = null;
        this.isBusy = isBusy;
    //  this.loadKind = loadKind;
        this.clearTime = clearTime;
        this.target = target;
        this.cur = 1;
    }
    void loadNew() {
        if (this.load != null) {
            Record.finished.add(this.load);
        }
        this.load = new Product();

    }
    boolean hasTarget() {
        if (target == -1) return false;
        else return true;
    }
//  int getLoadKind() {return this.loadKind;}
    void setTarget(int target) {this.target = target;}
    int getTarget() {return target;}
    boolean isReach() {return (target + 1) / 2 == cur;}
    void exchange() {
        if (Time.cnc[this.target - 1].load != null) {
            Time.cnc[this.target - 1].load.setEnd(Time.curTime);
            Time.cnc[this.target - 1].load.afterMachining();
        }
        this.load.setStart(Time.curTime);
        this.load.setCNC(this.target);

        Product tmp = Time.cnc[this.target - 1].load;
        Time.cnc[this.target - 1].load = this.load;
        this.load = tmp;

        for (int i = 0; i < exchangeTime[target % 2]; i++) {
            Time.go();
        }
        if (this.load == null) {
            this.loadNew();
```

```
            }

        }
        void clean() {
            for (int i = 0; i < this.clearTime; i++) {
                Time.go();
            }
        }
        Product getLoad() {return this.load;}
        void run() {
            int loc = (this.target + 1) / 2;
            //int direction = this.target - this.cur;
            int t = travelTime[Math.abs(loc - this.cur)];
        /*  int runDriection = 0 ;
            if (direction > 0) {
                runDriection = 1;
            }
            else if (direction < 0) {
                runDriection = -1;
            }*/
            for (int i = 0; i < t; i++) {

                Time.go();
            }
            this.cur = loc;


        }
}


import java.util.ArrayList;

public class CNC implements Comparable<CNC>{
    private int busyTime;
    //private int kind;
    Product load;
    private int num;
    private int workTime;


    public CNC(int busyTime, int preProduct, int num, int workTime) {
        super();
        this.load = null;
```

```
        this.busyTime = busyTime;
    //  this.kind = kind;
    //  this.preProduct = preProduct;
        this.num = num;
        this.workTime = workTime;
    }
    void renew() {
        if (this.busyTime < 0)
            this.busyTime++;
    }
    //int getKind() {return this.kind;}
    void startWork() {
        this.busyTime -= workTime;
    //  finishedRecord.add(new Record(this.productNum, this.num, Time.curTime -
RGV.exchangeTime[this.num % 2], 0));
    /*  if (preProduct != -1) {
            finishedRecord.get(preProduct).setEnd(Time.curTime -
RGV.exchangeTime[this.num % 2]);
        }*/
        //preProduct = this.productNum;

    //  this.productNum++;
    }
    int getNum() {return this.num;}
    boolean isBusy() {
        if (this.busyTime < 0) return true;
        else return false;
    }
    @Override
    public int compareTo(CNC arg0) {
        // TODO Auto-generated method stub
        /*int endTime1 = Time.curTime - this.busyTime;
        int endTime2 = Time.curTime - arg0.busyTime;*/
        int loc1 = (this.num + 1) / 2;
        int loc2 = (arg0.num + 1) / 2;
    //  System.out.println(Math.abs(RGV.cur - loc1) + "cur" +RGV.cur + ", loc1" + loc1 + ",
loc2" + loc2);

        int canReachTime1 = Math.max( RGV.travelTime[Math.abs(RGV.cur - loc1)] +
RGV.exchangeTime[this.num % 2], - this.busyTime + RGV.exchangeTime[this.num % 2]);
        int canReachTime2 = Math.max( RGV.travelTime[Math.abs(RGV.cur - loc2)] +
RGV.exchangeTime[arg0.num % 2], - arg0.busyTime + RGV.exchangeTime[arg0.num % 2]);
        if (canReachTime1 > canReachTime2) return 1;
        else return 0;
```

```
    }
}


public class Time {
    static int curTime;
    static int end = 8 * 60 *60;
    static CNC cnc[];

    public Time(int curTime, CNC[] cnc) {
        super();
        this.curTime = curTime;
        this.cnc = cnc;
    }

    static void go() {
        for (int i = 0; i < 8; i++) {
            cnc[i].renew();
        }
        curTime++;

        if (curTime == end) {
            for (int i = 0; i < Record.finished.size(); i++) {

                Record.finished.get(i).show();
            }
            System.exit(end);

        }
    }

}



public class Product {
    private int num;
    private int cnc_num;
    private int start;
    private int end;
    private int kind;
    static int productNum = 0;

    public Product() {
```

```
        super();
        num = productNum++;
        kind = 1;
    }
    void setEnd(int time) {
        if (kind == 1)
            this.end = time;

    }
    void setStart(int time) {
        if (kind == 1) {
            this.start = time;
        }

    }
    void afterMachining() {this.kind++;}
    void setCNC(int target) {
        if (this.kind == 1) cnc_num =target;

    }
    int getKind() {return this.kind;}
    void show() {
        System.out.println( (this.num + 1) + " " + this.cnc_num + " " + this.start+ " " +
this.end );
    }
}

import java.util.ArrayList;

public class Record {

    static ArrayList<Product> finished = new ArrayList<Product>();

}
////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
```

9.1 无故障双工序的 java 代码

```
import java.util.Queue;
import java.util.Scanner;

public class Simulation {

    public static void main(String[] args) {
```

```
// TODO Auto-generated method stub
int travelTime[] = new int [4];
int workTime[] = new int [2];
int clearTime = 0, target = -1, cur = 1;
int exchangeTime[] = new int [2];
//boolean hasTarget = false;
Scanner in = new Scanner(System.in);
travelTime[0] = 0;
for (int i = 1; i <= 3; i++) travelTime[i] = in.nextInt();
workTime[0] = in.nextInt();
workTime[1] = in.nextInt();
exchangeTime[1] = in.nextInt();
exchangeTime[0] = in.nextInt();
clearTime = in.nextInt();

RGV rgv = new RGV(0, clearTime, target);
int CNC_1 = in.nextInt();
int CNC_2 = in.nextInt();
int kind1[] = new int[CNC_1];
int kind2[] = new int[CNC_2];
CNC cnc[] = new CNC[8];
for (int i = 0; i < CNC_1; i++) {
    kind1[i] = in.nextInt();
    cnc[kind1[i] - 1] = new CNC(0, 1, -1, kind1[i], workTime[0]);
}
for (int i = 0; i < CNC_2; i++) {
    kind2[i] = in.nextInt();
    cnc[kind2[i] - 1] = new CNC(0, 2, -1, kind2[i], workTime[1]);
}


RGV.travelTime = travelTime;
RGV.exchangeTime = exchangeTime;
Time.cnc = cnc;

Queue<CNC> CNC_Queue[] = new java.util.PriorityQueue[2];
CNC_Queue[0] = new java.util.PriorityQueue(CNC_1);
CNC_Queue[1] = new java.util.PriorityQueue(CNC_2);
for (int i = 0; i < 8; i++) {
    CNC_Queue[cnc[i].getKind() - 1].add(cnc[i]);
}
rgv.loadNew();
while (true) {
    if (!rgv.hasTarget()) {
```

```
                    rgv.setTarget(CNC_Queue[rgv.getLoad().getKind() - 1].poll().getNum());
               }
               if (rgv.isReach() ) {
                    if (!cnc[rgv.getTarget() - 1].isBusy()) {
                         rgv.exchange();
                         cnc[rgv.getTarget() - 1].startWork();
                         int preTarget = rgv.getTarget();
                         rgv.setTarget(-1);
                         if (rgv.getLoad().getKind() == 3) {
                              rgv.clean();
                              rgv.loadNew();
                         }
                         CNC_Queue[cnc[preTarget - 1].getKind() - 1].add(cnc[preTarget - 1]);
                    }
                    else Time.go();

               }
               else rgv.run();


          }
     }

}


public class RGV {
     private int isBusy;
//   private int loadKind;
     private int clearTime;
     private int target;
     private Product load;
     static int cur;
     //private boolean hasTarget;
     static int exchangeTime[];
     static int travelTime[];



     public RGV(int isBusy, int clearTime, int target) {
          super();
          this.load = null;
          this.isBusy = isBusy;
     //   this.loadKind = loadKind;
```

```
        this.clearTime = clearTime;
        this.target = target;
        this.cur = 1;
    }
    void loadNew() {
        if (this.load != null) {
            Record.finished.add(this.load);
        }
        this.load = new Product();

    }
    boolean hasTarget() {
        if (target == -1) return false;
        else return true;
    }
//  int getLoadKind() {return this.loadKind;}
    void setTarget(int target) {this.target = target;}
    int getTarget() {return target;}
    boolean isReach() {return (target + 1) / 2 == cur;}
    void exchange() {
        if (Time.cnc[this.target - 1].load != null) {
            Time.cnc[this.target - 1].load.setEnd(Time.curTime);
            Time.cnc[this.target - 1].load.afterMachining();
        }
        this.load.setStart(Time.curTime);

        this.load.setCNC(this.target);
        Product tmp = Time.cnc[this.target - 1].load;
        Time.cnc[this.target - 1].load = this.load;
        this.load = tmp;
        for (int i = 0; i < exchangeTime[target % 2]; i++) {
            Time.go();
        }
        if (this.load == null) {
            this.loadNew();
        }

    }
    void clean() {
        for (int i = 0; i < this.clearTime; i++) {
            Time.go();
        }
    }
    Product getLoad() {return this.load;}
```

```
    void run() {
        int loc = (this.target + 1) / 2;
        //int direction = this.target - this.cur;
        int t = travelTime[Math.abs(loc - this.cur)];
    /*  int runDriection = 0 ;
        if (direction > 0) {
            runDriection = 1;
        }
        else if (direction < 0) {
            runDriection = -1;
        }*/
        for (int i = 0; i < t; i++) {

            Time.go();
        }
        this.cur = loc;


    }
}

import java.util.ArrayList;

public class CNC implements Comparable<CNC>{
    private int busyTime;
    private int kind;
    //private int preProduct;
    Product load;
    private int num;
    //static int productNum = 0;
    private int workTime;


    public CNC(int busyTime, int kind, int preProduct, int num, int workTime) {
        super();
        this.load = null;
        this.busyTime = busyTime;
        this.kind = kind;
    //  this.preProduct = preProduct;
        this.num = num;
        this.workTime = workTime;
    }
    void renew() {
        if (this.busyTime < 0)
```

```
            this.busyTime++;
    }
    int getKind() {return this.kind;}
    void startWork() {
        this.busyTime -= workTime;
    //  finishedRecord.add(new    Record(this.productNum,    this.num,    Time.curTime    -
RGV.exchangeTime[this.num % 2], 0));
    /*  if (preProduct != -1) {
            finishedRecord.get(preProduct).setEnd(Time.curTime                          -
RGV.exchangeTime[this.num % 2]);
        }*/
        //preProduct = this.productNum;

    //  this.productNum++;
    }
    int getNum() {return this.num;}
    boolean isBusy() {
        if (this.busyTime < 0) return true;
        else return false;
    }
    @Override
    public int compareTo(CNC arg0) {
        // TODO Auto-generated method stub
        int endTime1 = Time.curTime - this.busyTime;
        int endTime2 = Time.curTime - arg0.busyTime;
        int loc1 = (this.num + 1) / 2;
        int loc2 = (arg0.num + 1) / 2;
    //  System.out.println(Math.abs(RGV.cur - loc1) + "cur" +RGV.cur + ", loc1" + loc1 + ",
loc2" + loc2);

        int canReachTime1 = Math.max(Time.curTime + RGV.travelTime[Math.abs(RGV.cur - loc1)]
+ RGV.exchangeTime[this.num%2], Time.curTime - this.busyTime + RGV.exchangeTime[this.num%2]);
        int canReachTime2 = Math.max(Time.curTime + RGV.travelTime[Math.abs(RGV.cur - loc2)]
+ RGV.exchangeTime[arg0.num%2], Time.curTime - arg0.busyTime + RGV.exchangeTime[arg0.num%2]);
        if (canReachTime1 > canReachTime2) return 1;
        else return 0;
    }
}



public class Time {
    static int curTime;
    static int end = 8 * 60 *60;
```

```
    static CNC cnc[];

    public Time(int curTime, CNC[] cnc) {
        super();
        this.curTime = curTime;
        this.cnc = cnc;
    }

    static void go() {
        for (int i = 0; i < 8; i++) {
            cnc[i].renew();
        }
        curTime++;
        //System.out.println(curTime);
        if (curTime == end) {
            for (int i = 0; i < Record.finished.size(); i++) {

                Record.finished.get(i).show();
            }
            System.exit(end);

        }
    }

}


public class Product {
    private int num;
    private int cnc_num[];
    private int start[];
    private int end[];
    private int kind;
    static int productNum = 0;

    public Product() {
        super();
        num = productNum++;
        cnc_num = new int [2];
        start = new int [2];
        end = new int [2];
        kind = 1;
    }
    void setEnd(int time) {
```

```
        if (kind == 1)
            this.end[0] = time;
        else if (this.kind == 2)
            this.end[1] = time;
    }
    void setStart(int time) {
        if (kind == 1) {
            this.start[0] = time;
        }
        else if (this.kind == 2) this.start[1] = time;
    }
    void afterMachining() {this.kind++;}
    void setCNC(int target) {
        if (this.kind == 1) cnc_num[0] =target;
        else if (this.kind == 2)cnc_num[1] =target;
    }
    int getKind() {return this.kind;}
    void show() {
        System.out.println(" " + (this.num + 1) + " " + this.cnc_num[0] + " " + this.start[0]
+ " " + this.end[0] +" " + cnc_num[1] + "  " + this.start[1] + " " + this.end[1] );
    }
}


import java.util.ArrayList;

public class Record {

    static ArrayList<Product> finished = new ArrayList<Product>();


}
////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
```

9.3 随机故障单工序的 java 代码

```
import java.util.Queue;
import java.util.Scanner;

public class Simulation {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int finishKind = 2;
        int travelTime[] = new int [4];
```

```
int workTime;
int clearTime = 0, target = -1, cur = 1;
int exchangeTime[] = new int [2];
//boolean hasTarget = false;
Scanner in = new Scanner(System.in);
travelTime[0] = 0;
for (int i = 1; i <= 3; i++) travelTime[i] = in.nextInt();
workTime = in.nextInt();
exchangeTime[1] = in.nextInt();
exchangeTime[0] = in.nextInt();
clearTime = in.nextInt();

RGV rgv = new RGV(0, clearTime, target);

int kind = 1;
CNC cnc[] = new CNC[8];
for (int i = 0; i < 8; i++) {
    cnc[i] = new CNC(0, kind, -1, i + 1, workTime);
}

RGV.travelTime = travelTime;
RGV.exchangeTime = exchangeTime;
Timer.cnc = cnc;

Queue<CNC> CNC_Queue = new java.util.PriorityQueue(8);

for (int i = 0; i < 8; i++) {
    CNC_Queue.add(cnc[i]);
}
rgv.loadNew();
while (true) {
    if (!rgv.hasTarget()) {
        rgv.setTarget(CNC_Queue.poll().getNum());
    }
    if (rgv.isReach() ) {
        if (!cnc[rgv.getTarget() - 1].isBusy()) {
            rgv.exchange();
            cnc[rgv.getTarget() - 1].startWork();
            int preTarget = rgv.getTarget();
            rgv.setTarget(-1);
            if (rgv.getLoad().getKind() == finishKind) {
                rgv.clean();
                rgv.loadNew();
            }
```

```
                    CNC_Queue.add(cnc[preTarget - 1]);
                }
                else Timer.go();

            }
            else rgv.run();


        }
    }

}



public class RGV {
    private int isBusy;
//  private int loadKind;
    private int clearTime;
    private int target;
    private Product load;
    static int cur;
    //private boolean hasTarget;
    static int exchangeTime[];
    static int travelTime[];



    public RGV(int isBusy, int clearTime, int target) {
        super();
        this.load = null;
        this.isBusy = isBusy;
    //  this.loadKind = loadKind;
        this.clearTime = clearTime;
        this.target = target;
        this.cur = 1;
    }
    void loadNew() {
        if (this.load != null) {
            Record.finished.add(this.load);
        }
        this.load = new Product();

    }
```

```
boolean hasTarget() {
     if (target == -1) return false;
     else return true;
}
void setTarget(int target) {this.target = target;}
int getTarget() {return target;}
boolean isReach() {return (target + 1) / 2 == cur;}
void exchange() {
     if (Timer.cnc[this.target - 1].load != null) {
          Timer.cnc[this.target - 1].load.setEnd(Timer.curTime);
          Timer.cnc[this.target - 1].load.afterMachining();
     }
     this.load.setStart(Timer.curTime);

     this.load.setCNC(this.target);
     Product tmp = Timer.cnc[this.target - 1].load;
     Timer.cnc[this.target - 1].load = this.load;
     this.load = tmp;
     for (int i = 0; i < exchangeTime[target % 2]; i++) {
          Timer.go();
     }
     if (this.load == null) {
          this.loadNew();
     }

}
void clean() {
     for (int i = 0; i < this.clearTime; i++) {
          Timer.go();
     }
}
Product getLoad() {return this.load;}
void run() {
     int loc = (this.target + 1) / 2;
     //int direction = this.target - this.cur;
     int t = travelTime[Math.abs(loc - this.cur)];
/*   int runDriection = 0 ;
     if (direction > 0) {
          runDriection = 1;
     }
     else if (direction < 0) {
          runDriection = -1;
     }*/
     for (int i = 0; i < t; i++) {
```

```
                Timer.go();
            }
            this.cur = loc;


        }
}



import java.util.ArrayList;

public class CNC implements Comparable<CNC>{
        private int busyTime;

        Product load;
        private int num;
        private int brokenTime;
        //static int productNum = 0;
        private int workTime;
        //static Timer time;

        public CNC(int busyTime, int kind, int preProduct, int num, int workTime) {
            super();
            this.brokenTime = 1;
        /*  int randNum = (int)(Math.random() * 100);
            if (randNum == 0) {
                this.brokenTime = (int)(Math.random() * (80 * 6 * 6));
            }*/
            //System.out.println(this.brokenTime);
            this.load = null;
            this.busyTime = busyTime;

            this.num = num;
            this.workTime = workTime;
        }
        void renew() {

            if (this.busyTime < 0)
            {
                if (this.busyTime == this.brokenTime) {
                    int randTime = (int) (Math.random() * 10 * 60) + 10 * 60;
                    int brokenNum = -1;
```

```
                    if (this.load != null) {
                         brokenNum = this.load.getNum() + 1;
                    }
                    brokenRecord.br.add(new brokenRecord(brokenNum, this.num, Timer.curTime,
Timer.curTime + randTime));
                    this.busyTime = -randTime;
                    this.load = null;
                    this.brokenTime = 1;
               }
               else this.busyTime++;
          }

     }

     void startWork() {

          this.busyTime -= workTime;
          int randNum = (int)(Math.random() * 100);
          if (randNum == 0) {
               this.brokenTime = -(int)(workTime * Math.random());
          }

     }
     int getNum() {return this.num;}
     boolean isBusy() {
          if (this.busyTime < 0) return true;
          else return false;
     }
     @Override
     public int compareTo(CNC arg0) {
          // TODO Auto-generated method stub
          int endTime1 = Timer.curTime - this.busyTime;
          int endTime2 = Timer.curTime - arg0.busyTime;
          int loc1 = (this.num + 1) / 2;
          int loc2 = (arg0.num + 1) / 2;
     //   System.out.println(Math.abs(RGV.cur - loc1) + "cur" +RGV.cur + ", loc1" + loc1 + ",
loc2" + loc2);

          int canReachTime1 = Math.max(Timer.curTime + RGV.travelTime[Math.abs(RGV.cur - loc1)]
+ RGV.exchangeTime[this.num % 2], Timer.curTime - this.busyTime + RGV.exchangeTime[this.num %
2]);
          int canReachTime2 = Math.max(Timer.curTime + RGV.travelTime[Math.abs(RGV.cur - loc2)]
+ RGV.exchangeTime[arg0.num % 2], Timer.curTime - arg0.busyTime + RGV.exchangeTime[arg0.num %
2]);
```

```
            if (canReachTime1 > canReachTime2) return 1;
            else return 0;
        }
}



public class Product {
    private int num;
    private int cnc_num;
    private int start;
    private int end;
    private int kind;
    static int productNum = 0;

    public Product() {
        super();
        num = productNum++;

        kind = 1;
    }
    int getNum() {return this.num;}
    void setEnd(int time) {
        if (kind == 1)
            this.end = time;

    }
    void setStart(int time) {
        if (kind == 1) {
            this.start = time;
        }

    }
    void afterMachining() {this.kind++;}
    void setCNC(int target) {
        if (this.kind == 1) cnc_num =target;

    }
    int getKind() {return this.kind;}
    void show() {
        System.out.println((this.num + 1) + " " + this.cnc_num + " " + this.start + " " +
this.end);
    }
}
```

```
public class Timer {

	static int end = 8 * 60 *60;
	static CNC cnc[];
	static int curTime;


	static void go() {
		for (int i = 0; i < 8; i++) {
			cnc[i].renew();
		}
		curTime++;
		//System.out.println(curTime);
		if (curTime == end) {
			for (int i = 0; i < Record.finished.size(); i++) {
				Record.finished.get(i).show();
			}
			for (int i = 0; i < brokenRecord.br.size(); i++) {
				brokenRecord.br.get(i).show();
			}
			System.exit(end);

		}
	}
	int getTime() {return curTime;}

}


import java.util.ArrayList;

public class brokenRecord {
	static ArrayList<brokenRecord> br = new ArrayList<brokenRecord>();
	private int productNum;
	private int cnc_num;
	private int start;
	private int end;
	public brokenRecord(int productNum, int cnc_num, int start, int end) {
		super();
		this.productNum = productNum;
		this.cnc_num = cnc_num;
```

```
        this.start = start;
        this.end = end;
    }
    void show() {
        System.out.println((this.productNum) + " " + this.cnc_num + " " + this.start + " " +
this.end);
    }
}


import java.util.ArrayList;

public class Record {

    static ArrayList<Product> finished = new ArrayList<Product>();


}

////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
```

9.4 随机故障双工序的 java 代码

```
import java.util.Queue;
import java.util.Scanner;

public class Simulation {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
    //  Timer time = new Timer();
        int travelTime[] = new int [4];
        int workTime[] = new int [2];
        int clearTime = 0, target = -1, cur = 1;
        int exchangeTime[] = new int [2];
        //boolean hasTarget = false;
        Scanner in = new Scanner(System.in);
        travelTime[0] = 0;
        for (int i = 1; i <= 3; i++) travelTime[i] = in.nextInt();
        workTime[0] = in.nextInt();
        workTime[1] = in.nextInt();
        exchangeTime[1] = in.nextInt();
        exchangeTime[0] = in.nextInt();
        clearTime = in.nextInt();
    //  RGV rgv = new RGV(0, clearTime, target, cur);
```

```
RGV rgv = new RGV(0, clearTime, target);
int CNC_1 = in.nextInt();
int CNC_2 = in.nextInt();
int kind1[] = new int[CNC_1];
int kind2[] = new int[CNC_2];

CNC cnc[] = new CNC[8];
for (int i = 0; i < CNC_1; i++) {
     kind1[i] = in.nextInt();
     cnc[kind1[i] - 1] = new CNC(0, 1, -1, kind1[i], workTime[0]);
}
for (int i = 0; i < CNC_2; i++) {
     kind2[i] = in.nextInt();
     cnc[kind2[i] - 1] = new CNC(0, 2, -1, kind2[i], workTime[1]);
}
```

```
RGV.travelTime = travelTime;
RGV.exchangeTime = exchangeTime;
Timer.cnc = cnc;
//cnc[0].time = time;

Queue<CNC> CNC_Queue[] = new java.util.PriorityQueue[2];
CNC_Queue[0] = new java.util.PriorityQueue(CNC_1);
CNC_Queue[1] = new java.util.PriorityQueue(CNC_2);
for (int i = 0; i < 8; i++) {
     CNC_Queue[cnc[i].getKind() - 1].add(cnc[i]);
}
rgv.loadNew();
while (true) {
     if (!rgv.hasTarget()) {
          rgv.setTarget(CNC_Queue[rgv.getLoad().getKind() - 1].poll().getNum());
     }
     if (rgv.isReach() ) {
          if (!cnc[rgv.getTarget() - 1].isBusy()) {
               rgv.exchange();
               cnc[rgv.getTarget() - 1].startWork();
               int preTarget = rgv.getTarget();
               rgv.setTarget(-1);
               if (rgv.getLoad().getKind() == 3) {
                    rgv.clean();
                    rgv.loadNew();
```

```
                    }
                    CNC_Queue[cnc[preTarget - 1].getKind() - 1].add(cnc[preTarget - 1]);
                }
                else Timer.go();

            }
            else rgv.run();


        }
    }

}



public class RGV {
    private int isBusy;
//  private int loadKind;
    private int clearTime;
    private int target;
    private Product load;
    static int cur;
    //private boolean hasTarget;
    static int exchangeTime[];
    static int travelTime[];



    public RGV(int isBusy, int clearTime, int target) {
        super();
        this.load = null;
        this.isBusy = isBusy;
    //  this.loadKind = loadKind;
        this.clearTime = clearTime;
        this.target = target;
        this.cur = 1;
    }
    void loadNew() {
        if (this.load != null) {
            Record.finished.add(this.load);
        }
        this.load = new Product();
```

```
    }
    boolean hasTarget() {
        if (target == -1) return false;
        else return true;
    }
//  int getLoadKind() {return this.loadKind;}
    void setTarget(int target) {this.target = target;}
    int getTarget() {return target;}
    boolean isReach() {return (target + 1) / 2 == cur;}
    void exchange() {
        if (Timer.cnc[this.target - 1].load != null) {
            Timer.cnc[this.target - 1].load.setEnd(Timer.curTime);
            Timer.cnc[this.target - 1].load.afterMachining();
        }
        this.load.setStart(Timer.curTime);

        this.load.setCNC(this.target);
        Product tmp = Timer.cnc[this.target - 1].load;
        Timer.cnc[this.target - 1].load = this.load;
        this.load = tmp;
        for (int i = 0; i < exchangeTime[target % 2]; i++) {
            Timer.go();
        }
        if (this.load == null) {
            this.loadNew();
        }

    }
    void clean() {
        for (int i = 0; i < this.clearTime; i++) {
            Timer.go();
        }
    }
    Product getLoad() {return this.load;}
    void run() {
        int loc = (this.target + 1) / 2;
        //int direction = this.target - this.cur;
        int t = travelTime[Math.abs(loc - this.cur)];
    /*  int runDriection = 0 ;
        if (direction > 0) {
            runDriection = 1;
        }
        else if (direction < 0) {
            runDriection = -1;
```

```
        }*/
        for (int i = 0; i < t; i++) {

            Timer.go();
        }
        this.cur = loc;


    }
}



import java.util.ArrayList;

public class CNC implements Comparable<CNC>{
    private int busyTime;
    private int kind;
    //private int preProduct;
    Product load;
    private int num;
    private int brokenTime;
    //static int productNum = 0;
    private int workTime;
    //static Timer time;

    public CNC(int busyTime, int kind, int preProduct, int num, int workTime) {
        super();
        this.brokenTime = 1;
    /*  int randNum = (int)(Math.random() * 100);
        if (randNum == 0) {
            this.brokenTime = (int)(Math.random() * (80 * 6 * 6));
        }*/
        //System.out.println(this.brokenTime);
        this.load = null;
        this.busyTime = busyTime;
        this.kind = kind;
    //  this.preProduct = preProduct;
        this.num = num;
        this.workTime = workTime;
    }
    void renew() {
        /*int randNum = (int)(Math.random() * 100);
        if (randNum == 0) {
```

```
                int randTime = (int) (Math.random() * 10 * 60) + 10 * 60;
                int brokenNum = -1;
                if (this.load != null) {
                     brokenNum = this.load.getNum();
                }
                brokenRecord.br.add(new    brokenRecord(brokenNum,    this.num,    Timer.curTime,
Timer.curTime + randTime));
                this.busyTime = -randTime;
                this.load = null;
          }*/
          /*if (Timer.curTime == this.brokenTime) {
                int randTime = (int) (Math.random() * 10 * 60) + 10 * 60;
                int brokenNum = -1;
                if (this.load != null) {
                     brokenNum = this.load.getNum();
                }
                brokenRecord.br.add(new    brokenRecord(brokenNum,    this.num,    Timer.curTime,
Timer.curTime + randTime));
                this.busyTime = -randTime;
                this.load = null;
          }
*/
          if (this.busyTime < 0)
          {
                if (this.busyTime == this.brokenTime) {
                     int randTime = (int) (Math.random() * 10 * 60) + 10 * 60;
                     int brokenNum = -1;
                     if (this.load != null) {
                          brokenNum = this.load.getNum() + 1;
                     }
                     brokenRecord.br.add(new  brokenRecord(brokenNum,  this.num,  Timer.curTime,
Timer.curTime + randTime));
                     this.busyTime = -randTime;
                     this.load = null;
                     this.brokenTime = 1;
                }
                else this.busyTime++;
          }

     }
     int getKind() {return this.kind;}
     void startWork() {

          this.busyTime -= workTime;
```

```
            int randNum = (int)(Math.random() * 100);
            if (randNum == 0) {
                this.brokenTime = -(int)(workTime * Math.random());
            }

        }
        int getNum() {return this.num;}
        boolean isBusy() {
            if (this.busyTime < 0) return true;
            else return false;
        }
        @Override
        public int compareTo(CNC arg0) {
            // TODO Auto-generated method stub
            int endTime1 = Timer.curTime - this.busyTime;
            int endTime2 = Timer.curTime - arg0.busyTime;
            int loc1 = (this.num + 1) / 2;
            int loc2 = (arg0.num + 1) / 2;
        //  System.out.println(Math.abs(RGV.cur - loc1) + "cur" +RGV.cur + ", loc1" + loc1 + ",
loc2" + loc2);

            int canReachTime1 = Math.max(Timer.curTime + RGV.travelTime[Math.abs(RGV.cur - loc1)]
+ RGV.exchangeTime[this.num % 2], Timer.curTime - this.busyTime + RGV.exchangeTime[this.num %
2]);
            int canReachTime2 = Math.max(Timer.curTime + RGV.travelTime[Math.abs(RGV.cur - loc2)]
+ RGV.exchangeTime[arg0.num % 2], Timer.curTime - arg0.busyTime + RGV.exchangeTime[arg0.num %
2]);
            if (canReachTime1 > canReachTime2) return 1;
            else return 0;
        }
}



public class Product {
    private int num;
    private int cnc_num[];
    private int start[];
    private int end[];
    private int kind;
    static int productNum = 0;

    public Product() {
        super();
```

```
        num = productNum++;
        cnc_num = new int [2];
        start = new int [2];
        end = new int [2];
        kind = 1;
    }
    int getNum() {return this.num;}
    void setEnd(int time) {
        if (kind == 1)
            this.end[0] = time;
        else if (this.kind == 2)
            this.end[1] = time;
    }
    void setStart(int time) {
        if (kind == 1) {
            this.start[0] = time;
        }
        else if (this.kind == 2) this.start[1] = time;
    }
    void afterMachining() {this.kind++;}
    void setCNC(int target) {
        if (this.kind == 1) cnc_num[0] =target;
        else if (this.kind == 2)cnc_num[1] =target;
    }
    int getKind() {return this.kind;}
    void show() {
        System.out.println(" " + (this.num + 1) + " " + this.cnc_num[0] + " " + this.start[0]
+ " " + this.end[0] +" " + cnc_num[1] + " " + this.start[1] + " " + this.end[1] );
    }
}



public class Timer {

    static int end = 8 * 60 *60;
    static CNC cnc[];
    static int curTime;


    static void go() {
        for (int i = 0; i < 8; i++) {
            cnc[i].renew();
        }
```

```
            curTime++;
            //System.out.println(curTime);
            if (curTime == end) {
                for (int i = 0; i < Record.finished.size(); i++) {
                    Record.finished.get(i).show();
                }
                for (int i = 0; i < brokenRecord.br.size(); i++) {
                    brokenRecord.br.get(i).show();
                }
                System.exit(end);

            }
        }
        int getTime() {return curTime;}

}


import java.util.ArrayList;

public class brokenRecord {
        static ArrayList<brokenRecord> br = new ArrayList<brokenRecord>();
        private int productNum;
        private int cnc_num;
        private int start;
        private int end;
        public brokenRecord(int productNum, int cnc_num, int start, int end) {
            super();
            this.productNum = productNum;
            this.cnc_num = cnc_num;
            this.start = start;
            this.end = end;
        }
        void show() {
            System.out.println(" " + (this.productNum) + " " + this.cnc_num + " " + this.start +
" " + this.end);
        }
}


import java.util.ArrayList;

public class Record {
```

```
    static ArrayList<Product> finished = new ArrayList<Product>();


}
////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////
```

9.5 基于图模型的元胞自动机_一道工序的 python 程序

```
# coding : utf-8
# @author : JiaRong Zhong
# @date : 2018-09-15

import pandas as pd
from PIL import Image
import numpy as np

# 基于图论的元胞自动机
# 传入参数
'''
# CNC 加工完成一个单工序物料所需时间:t_process
# RGV 为 CNC1#，3#，5#，7#一次上下料所需时间:odd_load
# RGV 为 CNC2#，4#，6#，8#一次上下料所需时间:even_load
# RGV 完成一个物料的清洗作业所需时间:t_clean
# RGV 移动 1 个单位所需时间:unit1
# RGV 移动 2 个单位所需时间:unit2
# RGV 移动 3 个单位所需时间:unit3
'''

def CellularAutomatonBasedOnGraphTheory(t_process, odd_load, even_load, t_clean, unit1, unit2,
unit3):

    # 生成物料
    material = 1
    # 初始化 8 个结点，每个结点表示一台 CNC
    cnc = []
    cnc.append({'id':1,'t_load':odd_load,'t_clean':t_clean,'t_remind':0,'cooked_list':[]})
    cnc.append({'id':2,       't_load':even_load,       't_clean':t_clean,       't_remind':0,
'cooked_list':[]})
    cnc.append({'id':3,'t_load':odd_load,'t_clean':t_clean,'t_remind':0,'cooked_list':[]})
    cnc.append({'id':4,       't_load':even_load,       't_clean':t_clean,       't_remind':0,
'cooked_list':[]})
    cnc.append({'id':5,'t_load':odd_load,'t_clean':t_clean,'t_remind':0,'cooked_list':[]})
    cnc.append({'id':6,       't_load':even_load,       't_clean':t_clean,       't_remind':0,
'cooked_list':[]})
    cnc.append({'id':7,'t_load':odd_load,'t_clean':t_clean,'t_remind':0,'cooked_list':[]})
```

```
    cnc.append({'id':8,       't_load':even_load,       't_clean':t_clean,       't_remind':0,
'cooked_list':[]})

    # 初始化 1 台 GRV
    rgv = {'t_stay_currentnode':1, 't_stay_currentedge':0, 'current_node':1, 'next_node':0,
'path':[], 'tick_tock':0}

    # 初始化距离矩阵
    distance = [[0,0,1,1,2,2,3,3],
                [0,0,1,1,2,2,3,3],
                [1,1,0,0,1,1,2,2],
                [1,1,0,0,1,1,2,2],
                [2,2,1,1,0,0,1,1],
                [2,2,1,1,0,0,1,1],
                [3,3,2,2,1,1,0,0],
                [3,3,2,2,1,1,0,0]]

    # 利用距离信息构造时间字典
    dis_time = dict()
    for i in range(1, 9):
        dis_time.setdefault(i, dict())
        for j in range(1, 9):
            dis_time[i].setdefault(j, 0)
            if distance[i-1][j-1] == 0:
                dis_time[i][j] = 0
            elif distance[i-1][j-1] == 1:
                dis_time[i][j] = unit1
            elif distance[i-1][j-1] == 2:
                dis_time[i][j] = unit2
            elif distance[i-1][j-1] == 3:
                dis_time[i][j] = unit3

    # 用于存储 CNC 变化状态信息的字典
    cnc_info = dict()
    cnc_info.setdefault('rgv', [])
    for c in cnc:
        cnc_info.setdefault(c['id'], [])
    # 设置全局变量 DISTANCE，用来记录抵达下一个结点需要花掉的时间
    DISTANCE = 0
    # 设置时间条件，一个班次 8 小时
    t_end = 8*60*60
    while rgv['tick_tock'] <= t_end:

        # 每秒刷新一次 grv 的状态
```

```
        current = rgv['current_node']
        next_node = rgv['next_node']
        # 如果现在 grv 位于边上
        if rgv['t_stay_currentnode'] == 0:
            # 如果还在靠近目标 CNC 的途中，那么继续计时
            if rgv['t_stay_currentedge'] <= DISTANCE:
                rgv['t_stay_currentedge'] += 1
            # 抵达目标结点
            else:
                # 进入新的点，改变 current_node 信息
                rgv['current_node'] = rgv['next_node']
                # current = rgv['current_node']
                for c in cnc:
                    if c['id'] == rgv['current_node']:
                        c['cooked_list'].append(material)
                        material += 1
                rgv['t_stay_currentedge'] = 0
                rgv['t_stay_currentnode'] = 1

            # 如果 grv 在边上，那么 8 台 CNC 需要全部刷新
            for c in cnc:
                if c['t_remind'] != 0:
                    c['t_remind'] -= 1
        # 如果现在 grv 位于点上
        elif rgv['t_stay_currentedge'] == 0:
            # 如果 grv 还在上下料，那么继续计时
            if rgv['t_stay_currentnode'] <= cnc[current-1]['t_load']:
                rgv['t_stay_currentnode'] += 1
            # 如果 grv 上下料结束，正在清洗熟料，那么将当前点的 t_remind 相应地进行刷新操作
            elif rgv['t_stay_currentnode'] > cnc[current-1]['t_load'] and
rgv['t_stay_currentnode'] < cnc[current-1]['t_load'] + cnc[current-1]['t_clean']:
                rgv['t_stay_currentnode'] += 1
                cnc[rgv['current_node'] - 1]['t_remind'] = t_process + 1 -
(rgv['t_stay_currentnode'] - cnc[current-1]['t_load'])
            else:
                # 进入新的边，存储刚走过的点的信息
                rgv['path'].append(rgv['current_node'])
                rgv['t_stay_currentnode'] = 0
                rgv['t_stay_currentedge'] += 1
                DISTANCE = max(dis_time[current][next_node], cnc[next_node - 1]['t_remind'])

            # 如果 grv 在结点上，那么 7 台 CNC 需要刷新
            for c in cnc:
                if c['id'] != rgv['current_node'] and c['t_remind'] != 0:
```

```
                    c['t_remind'] -= 1
            # 计算下一结点的位置
            next_node = 0
            min_value = t_process
            for i in range(len(cnc)):
                if cnc[i]['id'] != rgv['current_node'] and
abs(dis_time[rgv['current_node']][cnc[i]['id']] - cnc[i]['t_remind']) < min_value:
                    next_node = cnc[i]['id']
                    min_value = abs(dis_time[rgv['current_node']][cnc[i]['id']] -
cnc[i]['t_remind'])
            rgv['next_node'] = next_node

        # grv 的时钟计时
        rgv['tick_tock'] += 1

        # 将每一秒更新的 CNC 状态信息存储成文件
        cnc_info['rgv'].append(rgv['tick_tock'])
        for c in cnc:
            cnc_info[c['id']].append(c['t_remind'])


    # 检查 cnc_info 字典的输出
    Grv, c1, c2, c3, c4, c5, c6, c7, c8 = [],[],[],[],[],[],[],[],[]
    Grv = cnc_info['rgv']
    c1 = cnc_info[1]
    c2 = cnc_info[2]
    c3 = cnc_info[3]
    c4 = cnc_info[4]
    c5 = cnc_info[5]
    c6 = cnc_info[6]
    c7 = cnc_info[7]
    c8 = cnc_info[8]

    # 字典中的 key 值即为 csv 中列名
    cols = ['rgv', 'cnc1', 'cnc2', 'cnc3', 'cnc4', 'cnc5', 'cnc6', 'cnc7', 'cnc8']
    dataframe = pd.DataFrame({'rgv':Grv,'cnc1':c1,'cnc2':c2,'cnc3':c3,'cnc4':c4,'cnc5':c5,
'cnc6':c6, 'cnc7':c7, 'cnc8':c8})
    # 将 DataFrame 存储为 csv,index 表示是否显示行名，default=True
    dataframe.to_csv("cnc_info.csv", index = False, sep=',', columns = cols)

    # 计算系统的利用率
    count_use = 0
    count_zero = 0
    for c in cnc_info.keys():
```

```
        if c != 'rgv':
            array = cnc_info[c]
            count_use += len(array)
            for j in array:
                if j == 0:
                    count_zero += 1
    # 返回系统的利用率
    return (count_use - count_zero) / count_use

if __name__=='__main__':
    # 参数列表
    parameter = [[560, 28, 31, 25, 20, 33, 46],
                 [580, 30, 35, 30, 23, 41, 59],
                 [545, 27, 32, 25, 18, 32, 46]]

    for i in range(3):
        t_process, odd_load, even_load, t_clean, unit_1, unit_2, unit_3  = parameter[i]
        rate = CellularAutomatonBasedOnGraphTheory(t_process, odd_load, even_load, t_clean-1,
unit_1, unit_2, unit_3)
        print('第 %s 组参数测试得到的系统利用率 rate = %s' % ((i+1), rate))
```

##

9.6 基于图模型的元胞自动机_两道工序的 python 代码

```
# coding : utf-8
# @author : JiaRong Zhong
# @date : 2018-09-15

import pandas as pd
from PIL import Image
import numpy as np

# 按照工序 1 和 2CNC 工作台不同比例生成距离矩阵的函数
def createDistanceMatrix(set_process1, set_process2):
    # 生成一个元素全为-1 的八阶矩阵
    distance = [[-1,-1,-1,-1,-1,-1,-1,-1],
                [-1,-1,-1,-1,-1,-1,-1,-1],
                [-1,-1,-1,-1,-1,-1,-1,-1],
                [-1,-1,-1,-1,-1,-1,-1,-1],
                [-1,-1,-1,-1,-1,-1,-1,-1],
                [-1,-1,-1,-1,-1,-1,-1,-1],
                [-1,-1,-1,-1,-1,-1,-1,-1],
                [-1,-1,-1,-1,-1,-1,-1,-1]]
    # 生成一个八阶完全图的距离矩阵
    k8 = [[0,0,1,1,2,2,3,3],
```

```
            [0,0,1,1,2,2,3,3],
            [1,1,0,0,1,1,2,2],
            [1,1,0,0,1,1,2,2],
            [2,2,1,1,0,0,1,1],
            [2,2,1,1,0,0,1,1],
            [3,3,2,2,1,1,0,0],
            [3,3,2,2,1,1,0,0]]
    # 所有存在的边类似于两个集合的笛卡尔积，因为这里的边不是单向的
    for x in set_process1:
        for y in set_process2:
            # 距离矩阵是一个对称阵
            distance[x-1][y-1] = k8[x-1][y-1]
            distance[y-1][x-1] = k8[x-1][y-1]
    '''
    # 测试输出
    for i in range(len(distance)):
        for j in range(len(distance[i])):
            print('%3d' % distance[i][j], end = ' ')
        print('\n')
    '''
    return distance

# 基于图论的元胞自动机，考虑双工序的情况
# 传入参数
'''
# RGV 移动 1 个单位所需时间:unit1
# RGV 移动 2 个单位所需时间:unit2
# RGV 移动 3 个单位所需时间:unit3
# CNC 加工完成一个两道工序物料的第一道工序所需时间:first_process
# CNC 加工完成一个两道工序物料的第二道工序所需时间:second_process
# RGV 为 CNC1#，3#，5#，7#一次上下料所需时间:odd_load
# RGV 为 CNC2#，4#，6#，8#一次上下料所需时间:even_load
# RGV 完成一个物料的清洗作业所需时间:t_clean
# 工序 1CNC 工作台的集合:cnc_process1
# 工序 1CNC 工作台的集合:cnc_process2
'''
def CellularAutomatonBasedOnGraphTheory(first_process, second_process, odd_load, even_load,
t_clean, cnc_process1, cnc_process2, unit_1, unit_2, unit_3):

    # 生成物料
    material = 1
    # 初始化 8 个结点，每个结点表示一台 CNC
    cnc = []
    cnc.append({'id':1,'t_load':odd_load,'t_clean':t_clean,'t_remind':0,'cooked_list':[]})
```

```
    cnc.append({'id':2,       't_load':even_load,       't_clean':t_clean,       't_remind':0,
'cooked_list':[]})
    cnc.append({'id':3,'t_load':odd_load,'t_clean':t_clean,'t_remind':0,'cooked_list':[]})
    cnc.append({'id':4,       't_load':even_load,       't_clean':t_clean,       't_remind':0,
'cooked_list':[]})
    cnc.append({'id':5,'t_load':odd_load,'t_clean':t_clean,'t_remind':0,'cooked_list':[]})
    cnc.append({'id':6,       't_load':even_load,       't_clean':t_clean,       't_remind':0,
'cooked_list':[]})
    cnc.append({'id':7,'t_load':odd_load,'t_clean':t_clean,'t_remind':0,'cooked_list':[]})
    cnc.append({'id':8,       't_load':even_load,       't_clean':t_clean,       't_remind':0,
'cooked_list':[]})

    # 初始化 1 台 GRV
    rgv = {'t_stay_currentnode':1, 't_stay_currentedge':0, 'current_node':1, 'next_node':0,
'path':[], 'tick_tock':0}

    # 计算在给定集合的情况下无向加权图的距离矩阵
    distance = createDistanceMatrix(cnc_process1, cnc_process2)

    dis_time = dict()
    for i in range(1, 9):
        dis_time.setdefault(i, dict())
        for j in range(1, 9):
            dis_time[i].setdefault(j, 0)
            if distance[i-1][j-1] == 0:
                dis_time[i][j] = 0
            elif distance[i-1][j-1] == 1:
                dis_time[i][j] = unit_1
            elif distance[i-1][j-1] == 2:
                dis_time[i][j] = unit_2
            elif distance[i-1][j-1] == 3:
                dis_time[i][j] = unit_3
            elif distance[i-1][j-1] == -1:
                dis_time[i][j] = 2018
    '''
    for i in dis_time.keys():
        for j in dis_time[i].keys():
            print('i = %s, j = %s, dis_time[i][j] = %s' % (i, j, dis_time[i][j]))
    '''
    # 用于存储 CNC 变化状态信息的字典
    cnc_info = dict()
    cnc_info.setdefault('rgv', [])
    for c in cnc:
        cnc_info.setdefault(c['id'], [])
```

```
    # 设置全局变量 DISTANCE，用来记录抵达下一个结点需要花掉的时间
    DISTANCE = 0
    # 设置时间条件，一个班次 8 小时
    t_end = 8*60*60
    while rgv['tick_tock'] <= t_end:

        # 每秒刷新一次 grv 的状态
        current = rgv['current_node']
        next_node = rgv['next_node']
        # 如果现在 grv 位于边上
        if rgv['t_stay_currentnode'] == 0:
            # 如果还在靠近目标 CNC 的途中，那么继续计时
            if rgv['t_stay_currentedge'] <= DISTANCE:
                rgv['t_stay_currentedge'] += 1
            # 抵达目标结点
            else:
                # 进入新的点，改变 current_node 信息
                rgv['current_node'] = rgv['next_node']
                # current = rgv['current_node']
                for c in cnc:
                    if c['id'] == rgv['current_node']:
                        c['cooked_list'].append(material)
                        material += 1
                rgv['t_stay_currentedge'] = 0
                rgv['t_stay_currentnode'] = 1

            # 如果 grv 在边上，那么 8 台 CNC 需要全部刷新
            for c in cnc:
                if c['t_remind'] != 0:
                    c['t_remind'] -= 1
        # 如果现在 grv 位于点上
        elif rgv['t_stay_currentedge'] == 0:
            # 如果 grv 还在上下料，那么继续计时
            if rgv['t_stay_currentnode'] <= cnc[current-1]['t_load']:
                rgv['t_stay_currentnode'] += 1
            # 如果 grv 上下料结束，正在清洗熟料，那么将当前点的 t_remind 相应地进行刷新操作
            elif rgv['t_stay_currentnode'] > cnc[current-1]['t_load'] and
rgv['t_stay_currentnode'] < cnc[current-1]['t_load'] + cnc[current-1]['t_clean']:
                rgv['t_stay_currentnode'] += 1

                work_time = 0
                if rgv['current_node'] in cnc_process1:
                    work_time = first_process
                else:
```

```
                    work_time = second_process
                cnc[rgv['current_node'] - 1]['t_remind'] = work_time -
(rgv['t_stay_currentnode'] - cnc[current-1]['t_load'])
            else:
                # 进入新的边，存储刚走过的点的信息
                rgv['path'].append(rgv['current_node'])
                rgv['t_stay_currentnode'] = 0
                rgv['t_stay_currentedge'] += 1
                DISTANCE = max(dis_time[current][next_node], cnc[next_node - 1]['t_remind'])

            # 如果 grv 在结点上，那么 7 台 CNC 需要刷新
            for c in cnc:
                if c['id'] != rgv['current_node'] and c['t_remind'] != 0:
                    c['t_remind'] -= 1
             # 计算下一结点的位置
            next_node = 0
            min_value = 560
            for i in range(len(cnc)):
                if cnc[i]['id'] != rgv['current_node'] and
abs(dis_time[rgv['current_node']][cnc[i]['id']] - cnc[i]['t_remind']) < min_value:
                    next_node = cnc[i]['id']
                    min_value = abs(dis_time[rgv['current_node']][cnc[i]['id']] -
cnc[i]['t_remind'])
            rgv['next_node'] = next_node

        # grv 的时钟计时
        rgv['tick_tock'] += 1

        # 将每一秒更新的 CNC 状态信息存储成文件
        cnc_info['rgv'].append(rgv['tick_tock'])
        for c in cnc:
            cnc_info[c['id']].append(c['t_remind'])

    # 计算系统的利用率
    count_use = 0
    count_zero = 0
    for c in cnc_info.keys():
        if c != 'rgv':
            array = cnc_info[c]
            count_use += len(array)
            for j in array:
                if j == 0:
                    count_zero += 1
    # 返回系统利用率参数
```

```
    return (count_use - count_zero) / count_use

if __name__=='__main__':

    set_node = range(1, 9)
    cnc_process1 = []
    cnc_process2 = []

    parameter = [[400, 378, 28, 31, 25-1, 20, 33, 46],
                 [280, 500, 30, 35, 30-1, 23, 41, 59],
                 [455, 182, 27, 32, 25-1, 18, 32, 46]]

    for para in range(3):
        # 测试结果收集器
        test_result = []
        name = str(para+1)
        name += "_test_result.csv"
        print('No. %s process...' % (para+1))
        print('[1] 第 %s 组参数测试... ' % (para+1))
        first_process, second_process, odd_load, even_load, t_clean, unit_1, unit_2, unit_3 =
parameter[para]

        # 记录这组参数下得到的最高的利用率，并记录 CNC 工作台的分布
        best_rate = 0
        best_choice = []

        # [1] 比例 1:7 测试
        sub_list = []
        for i in set_node:
            cnc_process1.append(i)
            for j in set_node:
                if j not in cnc_process1:
                    cnc_process2.append(j)
            rate = CellularAutomatonBasedOnGraphTheory(first_process, second_process, odd_load,
even_load, t_clean, cnc_process1, cnc_process2, unit_1, unit_2, unit_3)
            if rate > best_rate:
                best_rate = rate
                best_choice = []
                best_choice.append([cnc_process1, cnc_process2])
            sub_list.append(rate)
            # print('比例为 1:7 且 cnc%s 处理工序 1, 其余 cnc 处理工序 2 时系统的利用率 rate
= %s' % (i, rate))
            cnc_process1 = []
            cnc_process2 = []
```

```
        test_result.append(sub_list)
        print('test [1] done...')

        # [2] 比例 2:6 测试
        sub_list = []
        for i in range(1,8):
            for j in range(i+1, 9):
                cnc_process1 = [i, j]
                for j in set_node:
                    if j not in cnc_process1:
                        cnc_process2.append(j)
                rate  =  CellularAutomatonBasedOnGraphTheory(first_process,  second_process,
odd_load, even_load, t_clean, cnc_process1, cnc_process2, unit_1, unit_2, unit_3)
                if rate > best_rate:
                    best_rate = rate
                    best_choice = []
                    best_choice.append([cnc_process1, cnc_process2])
                sub_list.append(rate)
                # print('比例为 2:6 且 cnc%s 和 cnc%s 处理工序 1, 其余 cnc 处理工序 2 时系统的
利用率 rate = %s' % (i, j, rate))
                cnc_process2 = []
            cnc_process1 = []
        test_result.append(sub_list)
        print('test [2] done...')

        # [3] 比例 3:5 测试
        sub_list = []
        set_temp = []
        temp_list = []
        for i in range(1, 9):
            for j in range(1, 9):
                if j != i:
                    for k in range(1, 9):
                        if k != j and k != i:
                            temp_list.append([i,j,k])
        for i in temp_list:
            if set(i) not in set_temp:
                set_temp.append(set(i))
        temp_list = []
        for i in set_temp:
            temp_list.append(list(i))
        for i in range(len(temp_list)):
            cnc_process2 = []
            for j in set_node:
```

```
                if j not in temp_list[i]:
                    cnc_process2.append(j)
            rate = CellularAutomatonBasedOnGraphTheory(first_process, second_process, odd_load,
even_load, t_clean, temp_list[i], cnc_process2, unit_1, unit_2, unit_3)
            if rate > best_rate:
                best_rate = rate
                best_choice = []
                best_choice.append([temp_list[i], cnc_process2])
            sub_list.append(rate)
            cnc1, cnc2, cnc3 = temp_list[i]
            # print('No. %s, 比例为 3:5 且 cnc%s cnc%s cnc%s 处理工序 1, 其余 cnc 处理工序 2 时
系统的利用率 rate = %s' % ((i+1), cnc1, cnc2, cnc3, rate))
        test_result.append(sub_list)
        print('test [3] done...')

        # [4] 比例 4:4 测试
        sub_list = []
        four = []
        fo = []
        for i in range(1, 9):
            for j in range(1, 9):
                if j != i:
                    for k in range(1, 9):
                        if k != j and k != i:
                            for l in range(1, 9):
                                if l != i and l != j and l != k:
                                    fo.append([i,j,k,l])
        for i in fo:
            if set(i) not in four:
                four.append(set(i))
        fo = []
        for i in four:
            fo.append(list(i))
        for i in range(len(fo)):
            cnc_process2 = []
            for j in set_node:
                if j not in fo[i]:
                    cnc_process2.append(j)
            rate = CellularAutomatonBasedOnGraphTheory(first_process, second_process, odd_load,
even_load, t_clean, fo[i], cnc_process2, unit_1, unit_2, unit_3)
            if rate > best_rate:
                best_rate = rate
                best_choice = []
                best_choice.append([fo[i], cnc_process2])
```

```
            sub_list.append(rate)
            cnc1, cnc2, cnc3, cnc4 = fo[i]
            # print('No. %s, 比例为 4:4 且 cnc%s cnc%s cnc%s cnc%s 处理工序 1, 其余 cnc 处理工序 2 时系统的利用率 rate = %s' % ((i+1), cnc1, cnc2, cnc3, cnc4, rate))
        test_result.append(sub_list)
        print('test [4] done...')

        # [5] 比例 5:3 测试
        sub_list = []
        set_temp = []
        temp_list = []
        for i in range(1, 9):
            for j in range(1, 9):
                if j != i:
                    for k in range(1, 9):
                        if k != j and k != i:
                            temp_list.append([i,j,k])
        for i in temp_list:
            if set(i) not in set_temp:
                set_temp.append(set(i))
        temp_list = []
        for i in set_temp:
            temp_list.append(list(i))
        for i in range(len(temp_list)):
            cnc_process1 = []
            for j in set_node:
                if j not in temp_list[i]:
                    cnc_process1.append(j)
            rate = CellularAutomatonBasedOnGraphTheory(first_process, second_process, odd_load, even_load, t_clean, cnc_process1, temp_list[i], unit_1, unit_2, unit_3)
            if rate > best_rate:
                best_rate = rate
                best_choice = []
                best_choice.append([cnc_process1, temp_list[i]])
            sub_list.append(rate)
            cnc1, cnc2, cnc3, cnc4, cnc5 = cnc_process1
            # print('No. %s, 比例为 5:3 且 cnc%s cnc%s cnc%s cnc%s cnc%s 处理工序 1, 其余 cnc 处理工序 2 时系统的利用率 rate = %s' % ((i+1), cnc1, cnc2, cnc3, cnc4, cnc5, rate))
        test_result.append(sub_list)
        print('test [5] done...')

        # [6] 比例 6:2 测试
        sub_list = []
        for i in range(1,8):
```

```
            for j in range(i+1, 9):
                cnc_process2 = [i, j]
                for j in set_node:
                    if j not in cnc_process2:
                        cnc_process1.append(j)
                rate = CellularAutomatonBasedOnGraphTheory(first_process, second_process,
odd_load, even_load, t_clean, cnc_process1, cnc_process2, unit_1, unit_2, unit_3)
                if rate > best_rate:
                    best_rate = rate
                    best_choice = []
                    best_choice.append([cnc_process1, cnc_process2])
                sub_list.append(rate)
                # print('比例为 6:2 且 cnc%s 和 cnc%s 处理工序 2, 其余 cnc 处理工序 1 时系统的
利用率 rate = %s' % (i, j, rate))
                cnc_process2 = []
            cnc_process1 = []
        test_result.append(sub_list)
        print('test [6] done...')

        # [7] 比例 7:1 测试
        sub_list = []
        for i in set_node:
            cnc_process2.append(i)
            for j in set_node:
                if j not in cnc_process2:
                    cnc_process1.append(j)
            rate = CellularAutomatonBasedOnGraphTheory(first_process, second_process, odd_load,
even_load, t_clean, cnc_process1, cnc_process2, unit_1, unit_2, unit_3)
            if rate > best_rate:
                best_rate = rate
                best_choice = []
                best_choice.append([cnc_process1, cnc_process2])
            sub_list.append(rate)
            # print('比例为 1:7 且 cnc%s 处理工序 2, 其余 cnc 处理工序 1 时系统的利用率 rate
= %s' % (i, rate))
            cnc_process1 = []
            cnc_process2 = []
        test_result.append(sub_list)
        print('test [7] done...')

        # 按列存储 test_result 字典
        t1, t2, t3, t4, t5, t6, t7 = [],[],[],[],[],[],[]
        t1 = test_result[0]
        t2 = test_result[1]
```

```
        t3 = test_result[2]
        t4 = test_result[3]
        t5 = test_result[4]
        t6 = test_result[5]
        t7 = test_result[6]

        max_length = 0
        for i in test_result:
            if len(i)>max_length:
                max_length = len(i)
        for i in test_result:
            if len(i) != max_length:
                for j in range(max_length - len(i)):
                    i.append(0)

        # 字典中的key值即为csv中列名
        cols = ['1:7', '2:6', '3:5', '4:6', '5:5', '6:2', '7:1']
        dataframe = pd.DataFrame({'1:7':t1, '2:6':t2, '3:5':t3, '4:6':t4, '5:5':t5, '6:2':t6,
'7:1':t7})
        # 将DataFrame存储为csv,index表示是否显示行名，default=True
        dataframe.to_csv(name, index = False, sep=',', columns = cols)

        print('the best rate is %s, the best choice is %s' % (best_rate, best_choice))
        print('No. %s process done...' % (para+1))
```