计算机仿真求解RGV-CNC最优调度问题

摘 要

科技时代的来临不仅意味着人们生活的智能化，也意味着生产生活的智能化，本次解决的问题便是有关车间生产加工过程中，智能轨道式自动引导车的调度优化问题。本文从实际情况入手，通过适应生产条件的部分假设，着重于计算机编程的仿真环节，基于c++语言构筑有效的算法模拟多种调度策略，同时借助别具创新的故障概率建模，最终求得优化的调度策略。

针对情况一（一道工序），我们首先基于高等代数的理论，模式化地描述了RGV在工作过程中的时间代价，并通过对待测数据的前沿分析，确立了调度过程中的两条基本原则。之后我们借助于c++语言良好的模拟特性，以编程的方式再现了车间调度策略的种种方案，并利用计算机帮助求解其中最优的调度方法，最后通过Matlab绘制其甘特图验证结果的合理性和有效性。

针对情况二（两道工序），我们在情况一的基础上，通过再次观察待测数据分析刀具选择与排布的重要性，重新审视情况一的路径方案，将相关情况反映在c++代码中进行情况再现，最终借助于计算机编程在仿真的同时，得到了适应于情况二的结果，并绘制相应的甘特图。

针对情况三（考虑故障），我们首先通过实际情况简化问题，将范围缩小到单台机器一次加工的过程中，从几何学的角度模拟了1%的故障发生概率，并将其转化为计算机语言写入到仿真环境中，模拟了存在故障隐患的真实车间状况，同样在仿真的过程中验证了调度策略的优化度。

一、问题简述

随着现今科学技术的飞速发展，自动化、智能化的物流服务提升了产业的生产水平及人们的生活质量，其中智能 RGV 小车 (Rail Guided Vehicle) 在过程中扮演了重要的角色。类似地，在生产加工工厂中，通过利用智能 RGV 小车，我们同样可以大大提高工厂的生产力，在短时间内高效率地完成生产任务。然而，在资源有限、设备工作时间等的约束下，如何对调度策略进行优化，成为一个尤为重要的问题。

本次建模便是关于智能 RGV 的动态调度策略的问题，先重述如下：在一个加工系统中，存在 8 台计算机数控机床 (CNC) 、1 台智能 RGV 小车（含有清洗槽和双头机械手臂）、RGV 配套轨道、上料传送带及下料传送带。工作流程中，由 CNC 对生料及半成品进行加工；RGV 负责对 CNC 进行上下料、以及成品的清洗和回收等操作。在实际情况下，CNC 对生料加工需要一定的时间，且 RGV 对奇数编号的 4 台 CNC 比偶数编号的操作速度快，除此之外，RGV 清洗和回收成品也需要一定的时间。CNC 有两种状态：进行加工、等待上料；RGV 有四种状态：等待任务、进行上下料、在轨道上移动、清洗与回收成品。本次研究的关键点包括：1）如何在相同时间内，加工出更多成品？2）如何提高 CNC 的利用率，使每台 CNC 都能在时间上平衡地参与生产？

二、模型的假设

（1）情况一：含有一道工序、不考虑机器故障

针对情况一，即仅有一道工序且不考虑故障的情况下，进行如下简化与假设：

1）对同一台 CNC 的上料和下料操作是连贯进行的，即对某一 CNC 进行下料操作后，上料传送带立刻送入一块生料，紧接着立刻对此台 CNC 进行上料，使其继续工作；

（合理性：根据题设的要求，必须避免机器处于空机状态，且在仅有一道工序时，机械臂的特性可以允许这种操作）

2）智能 RGV 仅在收到请求信号的情况下，才可进行移动，否则将一直处于等待任务的状态而不能随意移动；

（合理性：题目明确指出 RGV 受 CNC 信号的控制，且在实际生产中，如果没有中央系统调控，RGV 只能接受外源信号而不可随处移动）

3）当有熟料需要清洗时，将清洗槽内清洗完成的成品放入下料传送带上，并将孰料放入清洗槽中，以上整个过程记为清洗时间，且对于第一轮操作，虽然没有放入下料传送带的步骤，也记为一次完整的清洗过程；

（合理性：符合题干对于清洗时间的定义且可以帮助简化问题，使每次操作后都计算一次清洗时间）

（2）情况二：含有两道工序、不考虑机器故障

针对情况二，即仅有两道工序且不考虑故障的情况下，可以继承关于情况一的所有假设，同时增添以下条目：

1）CNC 与刀具的匹配是固定的，不可中途更换，且两种刀具的数量不一定保持相同；

（合理性：题设中已指明不可更换刀具，故 CNC 和某一种刀具的匹配是确定的，同时观察待测数据可知，第一和第二道工序有较大的时间差，因此允许不同数量搭配可以更好地优化结果）

2）只有在第一道工序得到的半成品放在机械手臂上时，RGV 才可去响应负责第二道工序 CNC 的请求，否则只能响应第一道工序或处于静止状态；

（合理性：如果机械手臂上没有半成品，那么 RGV 前去负责第二道工序的 CNC 时，只能进行下料操作而不能立刻进行上料操作，这样将导致 CNC 处于空机状态，且再次上料会浪费许多时间）

（3）情况三：仅有一道或两道工序、考虑机器故障

针对情况三，即仅有一道工序且考虑机器故障的情况下，可以继承关于情况

一和二的所有假设，同时增加以下条目：

1）工作时 CNC 发生故障的概率为 1%，以一次加工、仅加工一个工件的过程为单位，考虑 1%的故障发生概率；

（合理性：每次加工过程是连续的，且每次加工仅针对一个工件，在此情况下每次加工的事件是独立的，故可以将每次加工分开考虑）

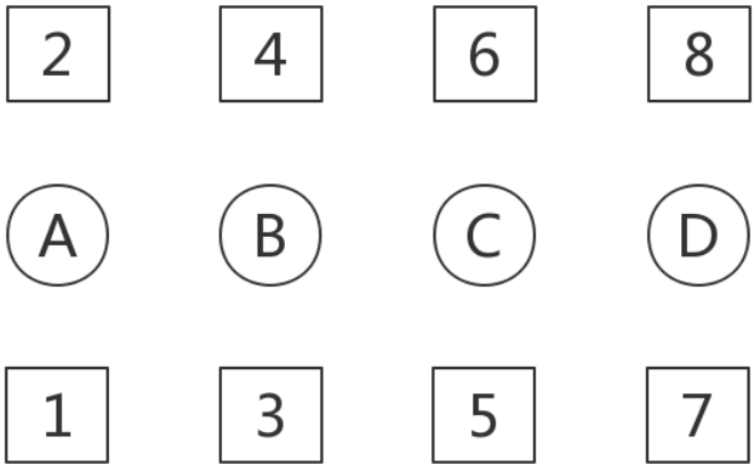
2）故障排解的过程中，故障 CNC 中的废料立刻从系统中移除，即在故障解决后 CNC 属于空机状态，正常等待 RGV 前来上料；

（合理性：符合题设要求，且可保证经过修理的 CNC 正常地处在队列中）

三、模型的建立

（1）情况一：仅有一道工序、不考虑机器故障

分析问题，可将智能 RGV 的工作流程分为两种情况：1）静置于轨道上正在等待任务；2）从某一位置前往某台 CNC 并进行相关操作。



对于后者，如上图所示，智能 RGV 可能的位置有 4 个（标号为 A、B、C、D），CNC 共有 8 台（标号为 1~8），针对 RGV 从某一特定位置前往某一台 CNC 进行操作的过程，可进一步分为三个步骤：

（a） 移动——RGV 在轨道上移动的过程

（b） 上下料——RGV 为目标 CNC 先进行下料，再进行上料

（c） 清洗——RGV 将加工后的熟料放入清洗槽中，并将槽里原有的成品放上下料传送带

可列出以下等式：

$$T_{sum} = T_{move} + T_{deal} + T_{clean}$$

T_{sum}：RGV 从某个位置前往某台 CNC 进行操作所花费的总时间

T_{move}：RGV 在轨道上移动的时间

T_{deal}：RGV 对 CNC 进行先下料后上料的操作时间

T_{clean}：RGV 的清洗工作时间

为了模式化地描述 RGV 从某个位置到某台机器并进行操作的总时间T_{sum}，我们构建一个 4×8 的矩阵 **M**：矩阵的行号对应 RGV 可能的 4 个位置 A、B、C、D，矩阵的列号对应 8 台 CNC。例如，矩阵元素 $\mathrm{T_{A,7}}$ 代表 RGV 从位置 A 开始前往 7 号 CNC 进行操作所需的总时间T_{sum}；矩阵元素 $\mathrm{T_{B,4}}$ 代表 RGV 从位置 B 开始前往 4 号 CNC 进行操作所需的总时间T_{sum}。

此外，对于奇数号 CNC 的T_{deal}，取值为题设表格中“RGV 为 CNC1#，3#，5#，7#一次上下料所需时间”；对于偶数号 CNC 的T_{deal}，取值为题设表格中“RGV 为 CNC2#，4#，6#，8#一次上下料所需时间”；对于T_{clean}，取值为题设表格中“RGV 完成一个物料的清洗作业所需时间”。对于T_{move}，根据 RGV 位置和目标 CNC 的相对位置，分别取表格中“移动 1 个单位”、“移动 2 个单位”、“移动 3 个单位”所需的时间。

由此，可以得到矩阵 **M**：

$$\mathbf{M} = \begin{bmatrix} \mathrm{T_{A,1}} & \mathrm{T_{A,2}} & \mathrm{T_{A,3}} & \mathrm{T_{A,4}} & \mathrm{T_{A,5}} & \mathrm{T_{A,6}} & \mathrm{T_{A,7}} & \mathrm{T_{A,8}} \\ \mathrm{T_{B,1}} & \mathrm{T_{B,2}} & \mathrm{T_{B,3}} & \mathrm{T_{B,4}} & \mathrm{T_{B,5}} & \mathrm{T_{B,6}} & \mathrm{T_{B,7}} & \mathrm{T_{B,8}} \\ \mathrm{T_{C,1}} & \mathrm{T_{C,2}} & \mathrm{T_{C,3}} & \mathrm{T_{C,4}} & \mathrm{T_{C,5}} & \mathrm{T_{C,6}} & \mathrm{T_{C,7}} & \mathrm{T_{C,8}} \\ \mathrm{T_{D,1}} & \mathrm{T_{D,2}} & \mathrm{T_{D,3}} & \mathrm{T_{D,4}} & \mathrm{T_{D,5}} & \mathrm{T_{D,6}} & \mathrm{T_{D,7}} & \mathrm{T_{D,8}} \end{bmatrix}$$

继续分析，在仅有一道工序且不考虑故障的情况下，最开始所有 8 台 CNC 均为空置状态。在按一定次序为 8 台机器进行第一次上料后，8 台机器均按照一定

的时间差依次完成工作，根据假设，在没有信号传入 RGV 的情况下，RGV 保持等待状态而不可移动。为了保证 RGV 的工作效率，在仅有一台 CNC 发出完成信号时，RGV 即立刻前往该 CNC 进行上下料操作。此外，由于存在时间$\mathbf{T_{sum}}$，在此时间段内，可能有多台 CNC 依次发出信号，应该优先处理哪一台 CNC 才能达到较高的工作效率？

由等式可知，影响$\mathbf{T_{sum}}$的因素有移动时间、上下料时间以及清洗时间。由于清洗时间都一致，所以只需要考虑移动时间以及上下料时间。可以提前观察待测的三组数据，对奇数号和偶数号 CNC 进行操作的时间相差值平均为 4.3 秒，而移动不同单位长度所需时间的差值在 8~18 秒之间波动，因而后者将作为主要影响因素，我们以“就近原则”响应不同 CNC 的信号：距离 RGV 最近的 CNC 被优先处理。对于与 RGV 距离相同的两台 CNC，为了保证每台 CNC 都有足够高的工作效率而不至于等待时间过久，我们选取队列中最先发出信号的 CNC 作为优先处理对象。

在上述原则的基础上，如果确定了 RGV 第一轮 8 台 CNC 的操作顺序，那么后续的调度行动也是确定的。至此，问题已经简化为：**初始第一轮操作，RGV 应该按照何种顺序进行依次上料**。我们可以借助刚刚构建的代价矩阵 **M**，以及上述就近原则和最先原则，对第一轮 8 台 CNC 的操作顺序的所有方案进行枚举，以计算机编程的方式计算各个方案生产速度，并保证每台 CNC 都有高且均衡的工作效率。

（2）情况二：含有两道工序、不考虑机器故障

分析问题，与情况一相比，情况二多了第二道工序。根据假设，只有在机械臂上有半成品的情况下，RGV 才能前去负责第二道工序的 CNC 进行上下料。同时我们在继承情况一的就近原则和最先原则的前提下，情况二不仅需要考虑第一轮上料的操作步骤，而且需要考虑装有不同刀具的 CNC 个数之比以及它们的排布位置——在确定这些条件后，RGV 调度策略即可随之确定。

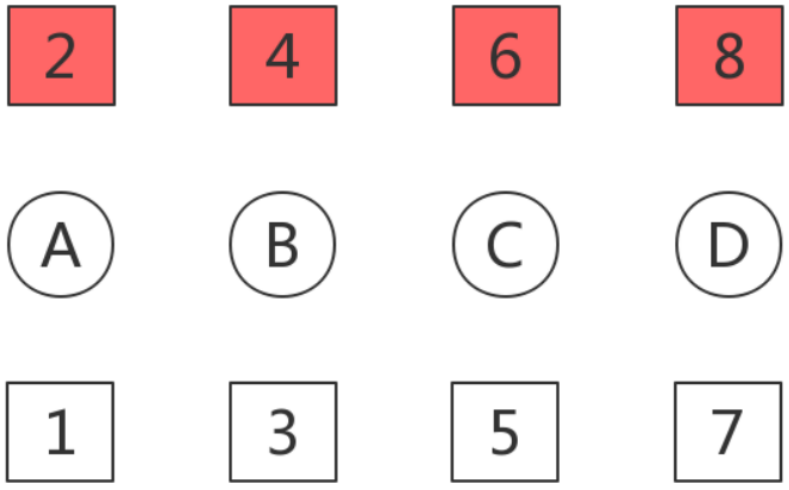
与情况一类似，继续构建一个 4×8 的矩阵，用以表示 RGV 从 4 个位置前往 8 台 CNC 的 32 种情况的用时T_{sum}。与情况一中模型的不同点在于，虽然T_{move}和T_{clean}的取值保持一致，但是T_{deal}根据不同刀具的排布选区题设表格中不同的取

值：“CNC 加工完成一个两道工序物料的第一道工序所需时间”、“CNC 加工完成一个两道工序物料的第二道工序所需时间”。

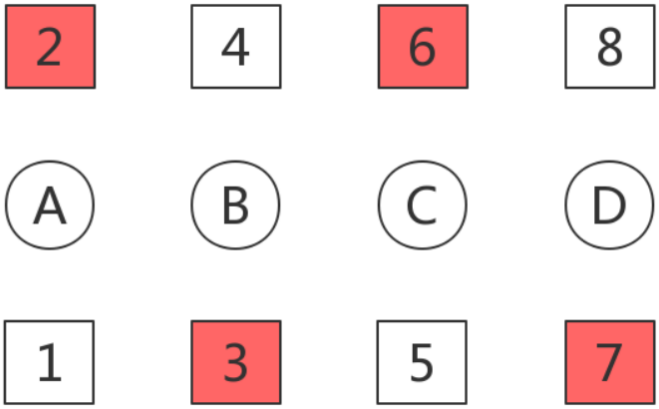
提前观察待测数据，可以发现第一组数据中两道工序的时间之比近似为 1:1，但是第三组数据两道工序的时间之比为 2.5:1。不难得知在两道工序的加工时间之比有较大变化的情况下，最优方案中不同刀具的数量之比必然会有所不同。例如，如果第一道工序加工时间较长，那么可能需要更多的 CNC 负责第一道工序的加工。

于是，关于不同刀具的选择和排布，按分步式概率的方法，方案共有2^8种。继续采用贪心枚举的方法，对每一种确定的刀具选择和排布方案，通过计算机程序模拟 RGV 进行第一轮上料操作的路径，对所有路径取最优的结果，进行打印和比较，挑选出所有结果。

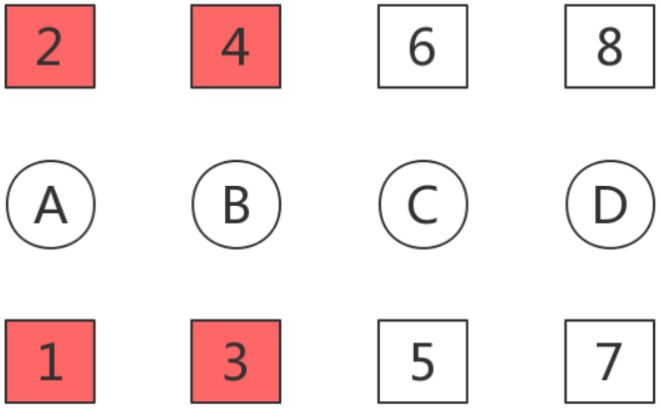
下面是几种可能的刀具选择和排布方案（白色 1 号刀具，红色 2 号刀具）：



（1）



（2）



（3）…

至此，问题已转化为算法问题，我们将通过计算机编程的方法进一步解决问题并进行仿真。详细内容参见“模型的求解”部分。

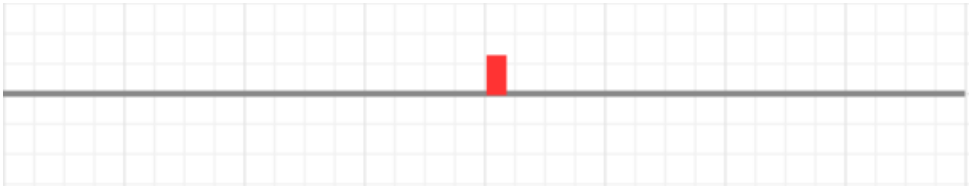
（3）情况三：含有一道或两道工序、考虑机器故障

分析问题，情况三建立在情况一和二的基础上，增加了故障发生这一因素。因此情况三重点解决的问题是，如何在程序中模拟 1%的故障概率事件。

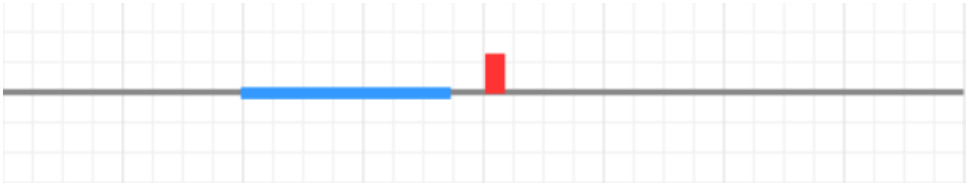
根据先前的假设，我们可以将加工一个工件的一次加工过程视为一次事件，模拟其发生故障的情况。对于模拟这种随机事件，既需要保证故障的发生概率为 1%，同时也要确保发生故障时有一个确定的故障发生时刻。我们采取下述方式建模：

1）假设有一条长度为 L 的长线段：

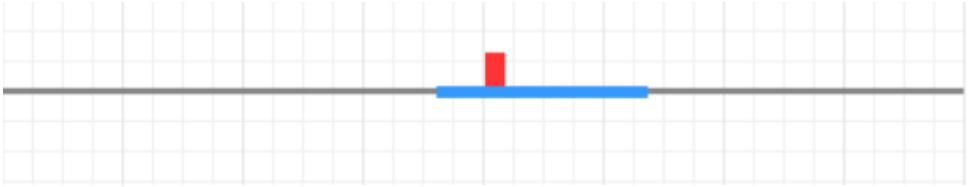
2）长线段的正中央有一段单位长度的红色区域：



3）一个长度为 x 的短线段在长线段内部滚动，当它随机停止时可以处于任意位置。如图，若红色区域未出现在短线段中，则代表未发生故障；



4）若红色区域出现在短线段中，则代表发生故障，且红色区域在短线段中的相对位置，作为故障在工作时间内发生的相对时刻：



为了保证红色区域出现在短线段中（发生故障）的概率$P = x/(L - x)$为 1%，我们可以得到等式：

$$\frac{x}{L - x} = 0.01$$

化简得：

$$L = 101x - 1$$

因此，我们令 x 取值为加工对应 CNC 的加工时间，由此计算出 L 的长度，通过计算机编程模拟上述随机事件，即可完成对于故障的模拟。

情况三实际上是情况一和情况二的再次仿真，附加条件是增加了故障事件的模拟以及相关调度。可以通过这种仿真再次验证情况一和情况二最终方案的优化

程度，也是对实际生产情况的一种再现。

四、模型的求解

1 模型仿真算法框架：

```
int main(){
    dfs(0,8);
    int m_res = 0;
    int m_i = 0;
    //int tpath[8]={1,3,5,7,8,6,4,2};
    for(int i=0;i<40320;i++){
        int result = Work(i);
        if (result > m_res) {
            m_res = result;
            m_i = i;
        }
    }
    cout<<"Max Result:"<<m_res<<endl;
    cout<<"Order:"<<endl;
    for(int j=0;j<8;j++){
        cout<<patharray[m_i][j]+1<<' ';
    }
    cout<<endl;
    system("Pause");
```

上料的所有可能顺序恰为 1-8 的全排列，共 8！=40320 种排列，每一种排列对应一个上料顺序。对每一种上料顺序计算 8 小时产生的孰料数，找出最大值所对应的最大孰料数以及其对应的上料顺序（可能不止一个）。

首先使用深度遍历算法产生 1-8 的全排列，图中为函数 void dfs(index,n)；函数的实现方式为常规递归算法，具体算法可参照附录。产生的所有排列存储在一个路径数组:patharray[40321][8]中。图中 Work(int i)函数为仿真的核心函数，参数 i 代表初始上料顺序在 patharray 中的索引。

2 仿真核心：

- 状态数组：

由于要模拟的系统较为复杂，为了保证正确星，采用了大量的状态数组标识工厂的状态。

```
bool first[8] = {true};
int CNCcount[8] = {0};//CNC工作计数
stuff_record stuff[500];
int stuffcount = 0;
int record[8];//记录CNC的物件编号
```

```
struct stuff_record{
    int cnc;
    int up;
    int down;
};
```

```
int state[8]={0};
int timeleft[8]={0};
int stime[8]={-1,-1,-1,-1,-1,-1,-1,-1};
int location=0;
```

变量的含义如下：

变量	含义
First	CNC 是否为第一次上料
CNCcount	CNC 完成的任务数
Stuff	所有工件的信息
Stuff_record	一个工件的信息，包括为其加工的 CNC 编号，开始和结束加工的时间。
Record	CNC 上正在加工的工价编号
Stuffcount	对工件进行计数和编号
State	CNC 是否为空闲状态（0 为空闲）
Timeleft	CNC 完成当前工作还需的时间
Stime	CNC 上当前加工的工件的开始上料时间。
Location	RGV 当前的位置

- 时间代价矩阵：

首先需要存储从 i(1-4)位置到达 j(1-8)编号的 CNC 并进行一次上下料操作所需要的时间，该时间是 RGV 决策时的主要参考量。

```
int COST[4][8]=
    {
        (RGV2CNC1),(RGV2CNC2),(RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),(RGV2CNC1+MOVE2),(RGV2CNC2
        +MOVE2),(RGV2CNC1+MOVE3),(RGV2CNC2+MOVE3),
        (RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),(RGV2CNC1),(RGV2CNC2),(RGV2CNC1+MOVE1),(RGV2CNC2
        +MOVE1),(RGV2CNC1+MOVE2),(RGV2CNC2+MOVE2),
        (RGV2CNC1+MOVE2),(RGV2CNC2+MOVE2),(RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),(RGV2CNC1),
        (RGV2CNC2),(RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),
        (RGV2CNC1+MOVE3),(RGV2CNC2+MOVE3),(RGV2CNC1+MOVE2),(RGV2CNC2+MOVE2),(RGV2CNC1+MOVE1),
        (RGV2CNC2+MOVE1),(RGV2CNC1),(RGV2CNC2)
    };
```

- 决策规则：

1）如果当前没有空闲的 CNC，则等待直至一个 CNC 完成。该规则的时间时通过一个数组 timeleft[8]记录每一个 CNC 完成当前物料还需的时间，从中选出最小的那个即为最早空闲出来的 CNC。

2）如果有空闲的 CNC，则使用一个容器存储空闲 CNC 的编号。如果这些空闲的 CNC 中有从未上过料的个体（即为新 CNC），则其应优先上料，且在有多个的情况下按照预生成的初始化上料顺序上料。若容器中所有 CNC 都不是新 CNC，则按照贪心规则选择下一个目标。

3）贪心规则的算法如下：

```
int NextDecision(int local, vector<int> vec, vector<int> svec){
    int m_i=0,m_d=10000;
    for(int i=0;i<vec.size();i++){
        if(COST[local][vec[i]] < m_d){
            m_i = vec[i];
            m_d = COST[local][vec[i]];
        }
    }
    for(int i=0;i<vec.size();i++){
        if(COST[local][vec[i]] == m_d){
            if(svec[i] < svec[m_i]){
                m_i = vec[i];
            }
        }
    }
    return m_i;
}
```

贪心的择优条件为时间代价矩阵对应值最小（对应第一个 for 循环）。如果结果有多个，则选择比较早开始工作的 CNC（对应第二个 for 循环），函数返回结果为目标 CNC 的编号（0-7）。

- 时间的模拟：

```
int gap = COST[location][target];
//前往目标并上料
TCount += gap;
stime[target] = TCount;
state[target] = 1;
location = (target)/2;
stuff[stuffcount].cnc = target+1;//上料，编号stuffcount;
int laststuff = record[target];
record[target] = stuffcount;
stuff[stuffcount].up = TCount;
stuffcount++;
```

时间的模拟式程序的一大关键，即对变量 TCount 的操作，一切判断都离不开对 TCount 的计算判断，其标志着整个工厂运行时间。在上下料过程中，新 CNC 和旧 CNC 有些许不同，新 CNC 由于之前没有孰料，所以到达新 CNC 时不会有清洗和下料过程。该规则的模拟借助了一个数组 first[8]表示该编号的 CNC 是否为第一次上料，如若不是，则会对应到某一个编号的物料的完成。

```
if(!first[target]){
    //清洗下料
    TCount += RGVCLEAN;
    Count += 1;
    CNCcount[target]++;
    stuff[laststuff].down = TCount;
}
else{
    first[target] = false;
}
```

终止条件：

根据题目要求，8 小时为一个工期，所以程序结束的条件即为工厂时间大于或等于 8*3600 = 28800s.程序终止后，返回本次工期期间产生的孰料个数。

```
if(TCount >= 28800) return Count;
```

上述模型针对模型一进行设计，可以直接对模型一进行仿真求解，最终题目要求的结果存储在 stuff[500]这个数组中。模型二和模型三只需对模型一进行一定的修改即可完成

3 模型二的求解：

模型二与模型一的区别在于两种工序产生了一个刀具的分配问题，以及在进行生产的时候对工件有次序限制，所以模型的修改主要集中在刀具的分配以及次序的控制上。

- 刀具的分配：

总共有 8 台机器，则分配共有 2^8 种情况（全部为一种刀具的包括在内）。分配再次使用一个递归算法进行模拟，结构存储在 typearray[256][8]这个二维数组中。

```
void TypeDivide(int index, int n=8){
    if(index == 8){
        for(int i=0;i<n;i++){
            typearray[row][i] = b[i];
        }
        row ++;
        return ;
    }
    else{
        b[index] = 1;
        TypeDivide(index+1, n);
        b[index] = 2;
        TypeDivide(index+1,n);
    }
}
```

➢ 次序控制：

次序控制的核心在于加入一个 bool 变量标识当前 RGV 的状态：RGVstate。控制的部位如下：

```
if( (RGVstate==0) && freeCNC1.empty()){
        //所有CNC都在工作，无上下料需求
        //找到最快完成工作的CNC，时间跟进
        int min_time=1000000,min_i=0;
        for(int i=0;i<8;i++){…
        }
        PCount += 1;
        //时间流逝……
        TCount += min_time;

    }
else{
        if(RGVstate==0){…
        }
        else{
            //寻找可用的2类机的位置
            //如果没有闲置的2类机，则等待：
            if(freeCNC2.empty()){…
            }
            //如果有空闲的二类机，寻找最近的那个
            else{…
            }
        }
}
```

若 RGVstae 为 0，表示当前 RGV 上无物料（包括生料、孰料、半成品），此时 RGV 下一个目标必然为第一道工序的 CNC；若其为 1，则代表 RGV 上有一个半成品，则其下一个目标必然为第二道工序的 CNC。通过在上下料步骤中修改 RGVstate 的值来模拟工序规则。

4 模型三的求解：

模型三在模型一和模型二的基础上加上了随机故障。故障发生后，故障的 CNC 无法继续工作，且其上的工件作废。模型三对模型一和模型二的修改主要集中在故障在工作过程中的发生以及工价作废所产生的影响。

- 故障的发生：

对于故障是否发生以及发生的时刻，使用 C++的 stdlib.h 和 time.h 产生随机数模拟，模拟的思想已经在前面阐述。

```
int Breakdown(int len)
{
    int L = 101*len-1;
    int x =rand()%(L+1);
    //if return value x is -1,there is no error;
    //else, it is the exact time when the error happens.
    x=(x<=28000&&x>=(28000-len+1))?(28000-x):(-1);
    return x;
}
```

函数的参数 len 表示模拟的时 len 长度时间种的故障，函数的返回值 x 若为 -1 则没有故障，否则为发生故障在 len 中的时间。

对于故障解决的时间，使用一个随机函数产生类模拟产生，使用时间系统作种子，保证随机性。

```
class RandomNumber{
public:
    RandomNumber(){
        srand(time(0));
    }
    int get(int begin = 0, int end = 1){
        return rand()%(end-begin+1)+begin;
    }
};
```

故障产生在 CNC 工作时，所以 Breakdown 函数的调用应该放在上下料模拟之后。如果发生故障则记录故障的信息。故障使用一个结构存储。

```
int b = Breakdown(CNCT);
if(b >= 0){…
}
stuffcount++;
```

```
struct break_info{
    int st;
    int cnc;
    int start;
    int solve;
};
break_info breaks[1000];
int breaktime = 0;
```

- 工件作废：

如果发生故障，则正在加工的工件作废，所以在故障被修复后，CNC 上无完成该加工工序的工件（熟料或半成品），所以相当于一个新 CNC，所以只需修改 first 数组，视其为新 CNC 即可。

```
if(b >= 0) first[target] = true;
```

五、结果分析与检验

根据上文设计的算法，分别将给定的三组数据带入问题一、二、三中，得到了各自的最优解。在此部分中，我们将对所得的数据加以说明以及相应的分析。

问题一：

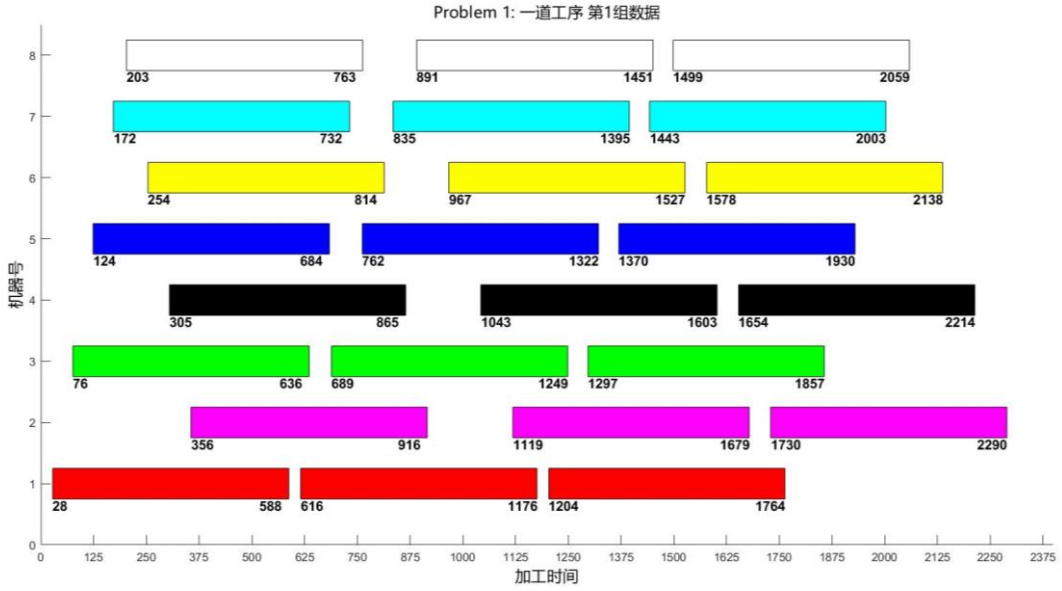
以第 1 组数据为例进行说明。经过计算得到，共有 5 种最优路径如下图所示（数字代表 CNC 的编号）：

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

在填表过程中，我们选择其中一条最优路径 1 3 4 7 8 6 4 2 进行数据填写，得到了相应情况下 8 小时内加工的情况，部分截图如下（NA 表示 8 小时内未完成）：

368	2	28003	28639		
369	1	28056	28692		
370	3	28129	28765		
371	5	28202	NA		
372	7	28275	NA		
373	8	28331	NA		
374	6	28462	NA		
375	4	28538	NA		
376	2	28614	NA		
377	1	28667	NA		
378	3	28740	NA		

从表中可以看出，在 8 小时内共完成加工 370 件（剩余部分在 8 小时内上料，但未在 8 小时内完成），并且为使得算法的流程更加清晰，利用所得数据通过 Matlab 制作了该流程对应的甘特图，具体如下：



（其中每种颜色唯一代表每一种机器，矩形块左下角的数字代表该物件加工开始的时间，右下角的数字代表该物件完成加工的时间，矩形的长度表示物件加工所需的时间）。以上部分是针对第一份数据的结构分析。

对于另外两组数据，由于结果相似，故不做重复分析，通过图片给出所得结果，且由于第二、三组数据做出的甘特图与第一组相似，故在此不再分析。对于第二组数据，共 4 种最优路径，选取 1 3 5 7 8 6 4 2 的路径为例，得到在 8 小时内共能加工 353 件物料：

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

351	4	27926	28594		
352	2	28014	28682		
353	1	28074	28742		
354	3	28157	NA		
355	5	28240	NA		

对于第三组数据，与第二组相同共有 4 种最优路径，与第二组数据选择相同的路径，得出在 8 小时内共加工完成物料 380 件：

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

378	3	27988	28608		
379	5	28058	28678		
380	7	28128	28748		
381	8	28185	NA		
382	6	28311	NA		

问题二：

针对问题二，将分别对三组数据进行分析说明。

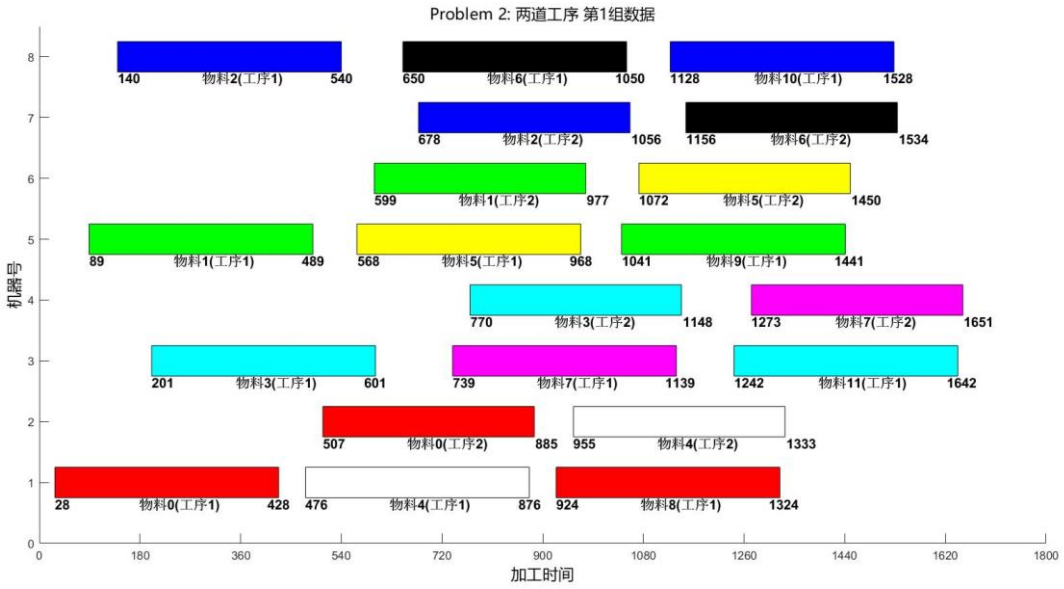
1、第一组数据：将数据带入算法模型中，我们共得到 7 组最优路径，分别如下：1 5 3 7、1 5 8 3、3 7 6 1、4 7 5 1、4 7 6 1、2 5 3 7、2 5 8 3（每一组数据表示该种情况下处理第一道工序的 CNC 的编号）。选取 1 5 8 3 作为 1 号 CNC 的情况进行分析，得到以下结果：

240	3	27539	28000	4	28031	28517
241	1	27643	28104	2	28135	28621
242	5	27779	28240	6	28271	28757
243	8	27886	28347	7	28375	NA
244	3	28000	28461	4	28492	NA

从表中得到，在 8 小时内总共完成了 242 件物料的加工，进一步可以得到每个 CNC 的使用率如下：

```
Use Rate:
1th: 59
2th: 58
3th: 59
4th: 58
5th: 60
6th: 59
7th: 59
8th: 60
```

与问题一相同，为了对于整体流程有更加深刻的了解，采用甘特图的方式对部分流程进行的可视化描绘如下（其中每种颜色唯一代表一个部件，矩形左右角的时间分别为该道工序开始加工的时间与完成加工的时间，下方文字表示该物料的编号以及当前所处的工序）：



2、第二组数据：将数据带入模型中，与上一组数据类似，能够得到 5 种最优路径分别为：6 3 1 8、6 4 1 8、6 3 2 8、2 4 5 7、6 2 4 8（每一组数据表示该种情况下处理第一道工序的 CNC 的编号）。选取 2 4 5 7 这一种分组作为例子进行分析，得到结果如下：

200	7	27279	27832	8	27867	28450	
201	4	27420	27973	1	28066	28649	
202	2	27578	28131	3	28184	28767	
203	5	27714	28267	6	28302	NA	
204	7	27832	28385	8	28420	NA	

总共在 8 小时内完成了 202 件物料的加工，同样得到每一个 CNC 的使用率分别为：

```
Use Rate:
    1th: 50
    2th: 51
    3th: 50
    4th: 51
    5th: 51
    6th: 52
    7th: 51
    8th: 52
```

由于与第一组数据相同，该情况也为 4：4 分割，因此对应的甘特图与第一组数据类似，不再额外分析说明。

3、第三组数据：在计算前不妨进行猜测，针对于 455：182 的两部工序时间，合理的猜测该种情况下处理两道工序的 CNC 应当为不平衡分割。代入数据得到，共有 10 种最优路径：1 2 4 6 7、1 2 4 6 8、3 6 1 8 4、1 5 6 8 4、 4 1 6 7

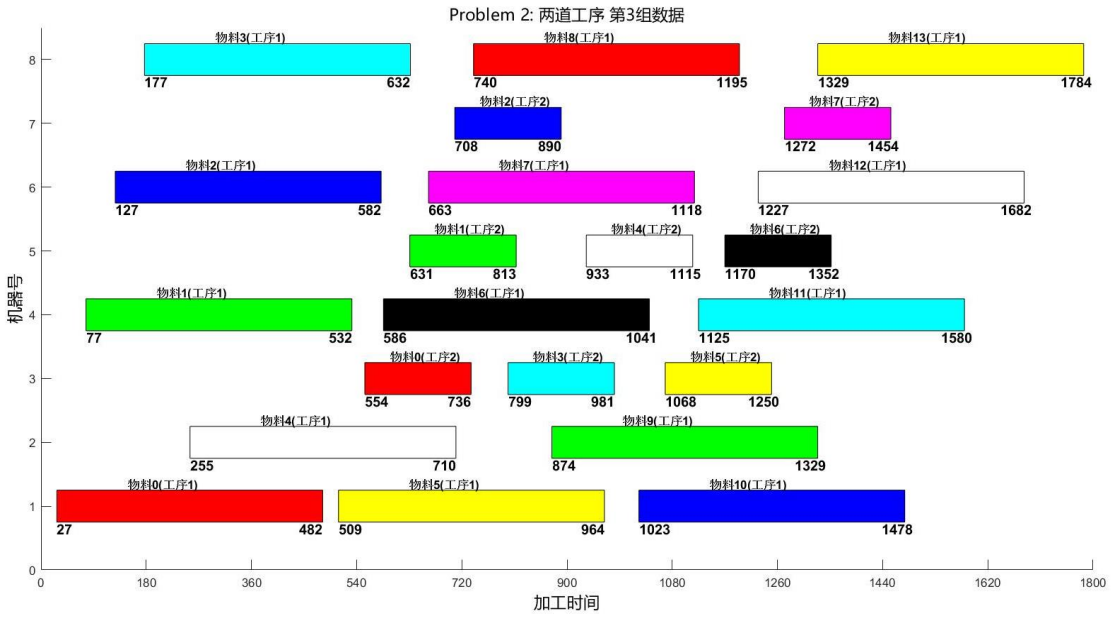
8、2 3 4 6 7、2 3 4 6 8、2 5 4 6 7、2 6 8 4 5、2 4 7 8 6（处理两道工序的 CNC 比例为 5：3，与猜想相符）。以 1 2 4 6 7 的分组情况为例，得到的数据如下所示，在 8 小时内共完成了 240 件物料的加工：

237	4	27406	27989	5	28034	28411	
238	6	27508	28091	7	28136	28744	
239	8	27610	28193	3	28252	28540	
240	2	27744	28327	5	28386	28642	
241	1	27887	28470	3	28515	NA	

通过数据能够得到各个 CNC 的使用率：Use Rate:

1th: 46
2th: 46
3th: 91
4th: 46
5th: 45
6th: 46
7th: 46
8th: 91

对于 5：3 的 CNC 分组，由于 CNC 的使用不再具有一定的规律性，因此甘特图的流程可视化对于理解流程十分重要（甘特图图示与第一组数据中相同）。从图中看出，由于第二道工序所需时间较短，因此在处理第二道工序的 CNC 数量更少的情况下，使用率也更高，通过对这 3 个第二道工序 CNC 的重复利用，来满足 5 个第一道工序 CNC 的需求，从而实现了整个流程。



问题三：

对于一道工序的情况，以第一组数据为例，得到以下的结果：在 8 小时的时间内共完成 366 个物料的加工（表格上），并通过随机数模拟得到了故障发生的时间和对应的 CNC 编号（表格下）。

363	5	28350	29007	
364	7	28423	29087	
365	8	28479	29100	
366	6	28559	28742	
367	4	28645	NA	
368	1	28718	NA	

10	3	972	1910
56	1	4554	5492
102	2	8167	9105
148	4	11757	12695
194	6	15347	16285
240	8	18945	19883
286	7	22555	23493
332	5	26148	27086

对于两道工序的情况，同样以第一组数据为例，得到以下的结果：以下两个表格分别表示，在 8 小时的时间内共完成 242 个物料的加工，以及通过随机数模拟得到的故障发生的时间和对应的 CNC 编号。

237	5	27342	27784	7	27832	28312	
238	8	27446	27888	2	28069	28549	
239	3	27563	28641	2	28731	27611	
240	1	27667	28122	6	28186	28666	
241	5	27784	28239	7	28287	28767	
242	8	27888	28343	2	28524	28003	
243	1	28122	28577	6	28641	NA	
244	5	28239	28694	7	28742	NA	

23	7	2660	3799
66	4	7658	8797
109	2	12650	13789
152	6	17646	18785
195	7	22629	23768
238	4	27611	28750

以上就是全部三个问题的数据结果以及相应的分析。

六、模型的评价、推广

本次建模提供的其实是一种处理实际问题的思路，以数学的方法模式化地描述问题，转化为计算机语言并利用 c++编程的良好特性，再现了车间生产中的调度场景，在仿真的同时求解得到最优的调度方案。

仿真所使用的贪心算法具有不易受干扰、稳定性强的特点；且其计算复杂度低，计算速度快。但是最优调度问题是一个 NP 问题，贪心算法智能得到局部的最优解。不过根据仿真的结果，贪心算法的效果可观，与极限效率相差无几。以问题 1 的第一组数据为例，本算法在一个工期产生了 371 个工件，与极限 411 相差了 40 个，时间利用率达 90%，在只有一个 RGV 的情况下，这个利用率是足够高的。

但是，此模型仍然有改进的方向，由于贪心算法只会选择时间代价最小的目标，但是实际运行中可能会通过一次让步而使得下一步节省更多的时间。这种情况可以使用遗传算法进行模拟，遗传算法模拟的核心算法可见附录及附加材料。同时，本问题中 RGV 的移动必须以 CNC 的需求为前提，但是 RGV 可以通过提前移动而利用等待的时间，从而增大流程的效率。

七、参考文献

[1] Michael Negnevitsky（著），陈薇（译），人工智能 智能系统指南，机械工业出版社

[2] 白其峥（主编），数学建模案例分析，海军出版社，2000 年北京出版

八、附录

Problem1.cpp

```
#include<stdio.h>
#include<iostream>
#include<string.h>
#include<vector>
#define MOVE1 (20)
#define MOVE2 (33)
#define MOVE3 (46)
#define CNCT (560)
#define CNCT1 (400)
#define CNCT2 (378)
#define RGV2CNC1 (28)
#define RGV2CNC2 (31)
#define RGVCLEAN (25)
using namespace std;
int patharray[40321][8];
int row=0;
int TCount=0,Count=0,PCount=0;
int COST[4][8]=
    {

(RGV2CNC1),(RGV2CNC2),(RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),(RGV2CNC1+MOV
E2),(RGV2CNC2+MOVE2),(RGV2CNC1+MOVE3),(RGV2CNC2+MOVE3),

(RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),(RGV2CNC1),(RGV2CNC2),(RGV2CNC1+MOV
E1),(RGV2CNC2+MOVE1),(RGV2CNC1+MOVE2),(RGV2CNC2+MOVE2),

(RGV2CNC1+MOVE2),(RGV2CNC2+MOVE2),(RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),(
RGV2CNC1),(RGV2CNC2),(RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),
```

```
(RGV2CNC1+MOVE3),(RGV2CNC2+MOVE3),(RGV2CNC1+MOVE2),(RGV2CNC2+MOVE2),(
RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),(RGV2CNC1),(RGV2CNC2)
    };
int NextDecision(int local, vector<int> vec, vector<int> svec){
    int m_i=0,m_d=10000;
    for(int i=0;i<vec.size();i++){
        if(COST[local][vec[i]] < m_d){
            m_i = vec[i];
            m_d = COST[local][vec[i]];
        }
    }
    for(int i=0;i<vec.size();i++){
        if(COST[local][vec[i]] == m_d){
            if(svec[i] < svec[m_i]){
                m_i = vec[i];
            }
        }
    }
    return m_i;
}
int visit[100];//用来判断这个数字是否被访问过 如 visit[2]=1;说明数字 2
被访问过
int a[100];//0-(n-1)存放数字
void dfs(int index,int n)
{
    if(index==n)//终止条件
    {
        for(int i=0;i<n;i++)
```

```
        {
            patharray[row][i] = a[i]-1;
        }
        row++;
        return;
    }
    for(int i=1;i<=n;i++)//判断数字是否被访问过
    {
        if(visit[i]==0)
        {
            visit[i]=1;
            a[index]=i;
            dfs(index+1,n);
            visit[i]=0;
        }
    }
}


struct stuff_record{
    int cnc;
    int up;
    int down;
};
bool first[8] = {true};
int CNCcount[8] = {0};//CNC 工作计数
stuff_record stuff[500];
int stuffcount = 0;
int record[8];//记录 CNC 的物件编号
```

```
int Work(int path){
    stuffcount = 0;
    int state[8]={0};
    int timeleft[8]={0};
    int stime[8]={-1,-1,-1,-1,-1,-1,-1,-1};
    int location=0;
    TCount = 0; Count = 0; PCount = 0;
    for(int i=0;i<8;i++){
        state[i]=0;
        CNCcount[i] = 0;
        stime[i] = -1;
        timeleft[i] = 0;
        record[i] = -1;
        first[i] = true;
    }
    //初始序列：
    vector<int> initvec;
    for(int i=0;i<8;i++){
        initvec.push_back(patharray[path][7-i]);
    }
    while(1){
        //计算每个 CNC 完成工作还需要的时间；公式为：剩余时间=当前时间
-开始时间；
        for(int i=0;i<8;i++){
            if(stime[i] == -1){
                state[i] = 0;
                continue;
            }
            int tmp = TCount-stime[i];
```

```
        timeleft[i] = tmp>CNCT ? 0 : (CNCT-tmp);
    }

    //检查各个CNC的状态
    for(int i=0;i<8;i++){
        if(timeleft[i] == 0){
            state[i] = 0;
        }
        else {
            state[i]=1;
        }
    }
    //检查是否有上下料需求
    vector<int> freeCNC;
    for(int i=0;i<8;i++){
        if(state[i]==0) freeCNC.push_back(i);
    }
    if(freeCNC.empty()){
        //所有CNC都在工作，无上下料需求
        //找到最快完成工作的CNC，时间跟进
        int min_time=CNCT,min_i=0;
        for(int i=0;i<8;i++){
            if(timeleft[i]<min_time){
                min_time = timeleft[i];
                min_i = i;
            }
        }
        PCount += 1;
        //时间流逝.....
```

```
        TCount += min_time;
    }
    else{
        //从队列里寻找下一个去的位置
        //初始化时，要按顺序来，之后再使用 NextDecison
        int target;
        if(!initvec.empty()){
            target = initvec[initvec.size()-1];
            initvec.pop_back();
        }
        else{
            vector<int> startvec;
            for(int i=0;i<freeCNC.size();i++){
                startvec.push_back(stime[freeCNC[i]]);
            }
            target = NextDecision(location, freeCNC,startvec);
        }
        int gap = COST[location][target];
        //前往目标并上料
        TCount += gap;
        stime[target] = TCount;
        state[target] = 1;
        location = (target)/2;
        stuff[stuffcount].cnc = target+1;//上料，编号 stuffcount;
        int laststuff = record[target];
        record[target] = stuffcount;
        stuff[stuffcount].up = TCount;
        stuffcount++;
        if(!first[target]){
```

```
                //清洗下料
                TCount += RGVCLEAN;
                Count += 1;
                CNCcount[target]++;
                stuff[laststuff].down = TCount;
            }
            else{
                first[target] = false;
            }
            if(TCount >= 28800) return Count;
        }

    }
}
bool ispath(int path,int tp[8]){
    for(int i=0;i<8;i++){
        if(tp[i] != patharray[path][i]+1) return false;
    }
    return true;
}
int main(){
    dfs(0,8);
    int m_res = 0;
    int m_i = 0;
    int tpath[8]={1,3,5,7,8,6,4,2};
    for(int i=0;i<40320;i++){
        int result = Work(i);
        //cout<<i<<"th is:"<<result<<endl;
        if (result > m_res) {
```

```
            m_res = result;
            m_i = i;
        }
    }
    cout<<"Max Result:"<<m_res<<endl;
    cout<<"Order:"<<endl;
    for(int j=0;j<8;j++){
        cout<<patharray[m_i][j]+1<<' ';
    }
    cout<<endl;
    system("Pause");
}
```

Problem2.cpp

```
int patharray[40321][8];
int typearray[256][8];
int row=0;

int b[10];
void TypeDivide(int index, int n=8){
    if(index == 8){
        for(int i=0;i<n;i++){
            typearray[row][i] = b[i];
        }
        row ++;
        return ;
    }
    else{
        b[index] = 1;
```

```
        TypeDivide(index+1, n);
        b[index] = 2;
        TypeDivide(index+1,n);
    }
}
struct stuff_record{
    int flag;//flag = 1 孰料
    int cnc1;
    int up1;
    int down1;
    int cnc2;
    int up2;
    int down2;
};

int TCount=0,Count=0,PCount=0;
int CNCcount[8]={0};
bool first[8]={true};
stuff_record stuff[500];
int stuffcount = 0;
int record[8];//记录 CNC 的物件编号

int Work(int div,int path,int len){
    stuffcount = 0;
    int state[8];
    int timeleft[8];
    int stime[8]={-1,-1,-1,-1,-1,-1,-1,-1};
    int location=0;
    int type[8];
```

```
    TCount = 0; Count = 0; PCount = 0;
    //标记分工情况
    int type[8];
    for(int i=0;i<8;i++){
        state[i]=0;
        CNCcount[i] = 0;
        stime[i] = -1;
        timeleft[i] = 0;
        record[i] = -1;
        first[i] = true;
        type[i] = typearray[div][i];
    }
    //初始化序列
    vector<int> initvec;
    for(int i=0;i<len;i++){
        initvec.push_back(patharray[path][len-1-i]);
    }
    bool RGVstate=0;    int lastrough=-1;
    while(1){
            for(int i=0;i<8;i++){
            if(stime[i] == -1){
                state[i] = 0;
                continue;
            }
            int tmp = TCount-stime[i];
            if (type[i] == 1) timeleft[i] = tmp>CNCT1 ? 0 : (CNCT1-
tmp);
            else timeleft[i] = tmp>CNCT2 ? 0 : (CNCT2-tmp);
        }
```

```
//检查各个CNC的状态
for(int i=0;i<8;i++){
    if(timeleft[i] == 0) state[i] = 0;
    else state[i] = 1;
}
//检查是否有上下料需求
vector<int> freeCNC1;//第一类的空闲
vector<int> freeCNC2;//第二类的空闲
for(int i=0;i<8;i++){
    if(state[i]==0) {
        if(type[i] == 1) freeCNC1.push_back(i);
        else freeCNC2.push_back(i);
    }
}
if( (RGVstate==0) && freeCNC1.empty()){
    //所有CNC都在工作，无上下料需求
    //找到最快完成工作的CNC，时间跟进
    int min_time=1000000,min_i=0;
    for(int i=0;i<8;i++){
        if( (type[i] == 1) && (timeleft[i]<min_time) ){
            min_time = timeleft[i];
            min_i = i;
        }
    }
    PCount += 1;
    //时间流逝.....
    TCount += min_time;
}
else{
```

```
            if(RGVstate==0){
                //从队列里寻找下一个去的1类机的位置
                int target;
                if(!initvec.empty()){
                    target = initvec[initvec.size()-1];
                    initvec.pop_back();
                }
                else {
                    vector<int> startvec;
                    for(int i=0;i<freeCNC1.size();i++){
                        startvec.push_back(stime[freeCNC1[i]]);
                    }
                    target          =          NextDecision(location,
freeCNC1,startvec);//tag1
                }
                int gap = COST[location][target];
                //前往目标
                TCount += gap;
                stime[target] = TCount;
                state[target] = 1;
                location = (target)/2;
                stuff[stuffcount].cnc1  =  target+1;// 上 料 ， 编 号
stuffcount;
                int laststuff = record[target];
                record[target] = stuffcount;
                stuff[stuffcount].up1 = TCount;
                stuffcount++;
                if(!first[target]){
                    //取下半成品的情况
```

```
                    //Count += 1;
                    CNCcount[target]++;
                    stuff[laststuff].down1 = TCount;
                    RGVstate = 1;//RGV 手拿半成品
                    lastrough  =  laststuff;//记录处于 RGV 手中的半成品
工件
                }
                else{
                    first[target] = false;
                    //RGV 手中无半成品
                }

            }

            else{
                //寻找可用的 2 类机的位置
                //如果没有闲置的 2 类机，则等待：
                if(freeCNC2.empty()){
                    int min_time=CNCT2,min_i=0;
                    for(int i=0;i<8;i++){
                        if(       (type[i]       ==       2)       &&
(timeleft[i]<min_time) ){
                            min_time = timeleft[i];
                            min_i = i;
                        }
                    }
                    PCount += 1;
                    //时间流逝.....
                    TCount += min_time;
```

```
                }
                //如果有空闲的二类机，寻找最近的那个
                else{
                    vector<int> startvec;
                    for(int i=0;i<freeCNC2.size();i++){
                        startvec.push_back(stime[freeCNC2[i]]);
                    }

                    int target = NextDecision(location,
freeCNC2,startvec);
                    int gap = COST[location][target];

                    //前往目标
                    TCount += gap;
                    stime[target] = TCount;
                    state[target] = 1;
                    location = (target)/2;
                    RGVstate = 0;

                    int laststuff = record[target];
                    record[target] = lastrough;
                    stuff[lastrough].cnc2 = target+1;
                    stuff[lastrough].up2 = TCount;

                    if(!first[target]){
                        TCount += RGVCLEAN;
                        Count += 1;
                        CNCcount[target]++;
                        stuff[laststuff].down2 = TCount;
```

```
                    }
                    else{
                        first[target] = false;
                    }
                    if(TCount >= 28800) return Count;
                }
            }
        }
    }
}
bool allsame(int t[8]){
    for(int i=1;i<8;i++){
        if(t[i] != t[0]) return false;
    }
    return true;
}
bool ispath(int path,vector<int> tp){
    for(int i=0;i<tp.size();i++){
        if(tp[i] != patharray[path][i]+1) return false;
    }
    return true;
}
int main(){
    //CNC 种类分配
    TypeDivide(0,8);
    int move[4]={0,MOVE1,MOVE2,MOVE3};
    int cnc[2] = {RGV2CNC1,RGV2CNC2};
    int cncp[2] = {CNCT1,CNCT2};//工序
    int m_res = 10000000;
```

```
int m_i = 0,m_j=0;
vector<int> finalans;
for(int i=0;i<256;i++){
    if(allsame(typearray[i])) continue;
    m_res = 0;
    m_j=0;
    //记录第一道工序的CNC
    vector<int> vec1;
    for(int j=0;j<8;j++){
        if(typearray[i][j] == 1) vec1.push_back(j);
    }
    //针对第一道工序的CNC寻找初始化路径
    row = 0;
    dfs(0,vec1.size(),vec1);
    for(int j=0;j<row;j++){
        //第j种路径的情况
        int pathlen = vec1.size();//j路径的具体路径
        int result = Work(i,j,pathlen);
        if (result > m_res) {
            m_res = result;
            m_i = i;
            m_j = j;
        }
        vector<int> tp = {1,5,8,3};
        if(i==86 && ispath(j,tp) ){
        for(int p=0;p<8;p++){
            cout<<patharray[i][p]+1<<' ';
        }
        cout<<endl;
```

```
            for(int i=0;i<stuffcount;i++){
                cout<<i+1<<"th:                    "<<stuff[i].cnc1<<"
"<<stuff[i].up1<<" "<<stuff[i].down1;
                cout<<stuff[i].cnc2<<"             "<<stuff[i].up2<<"
"<<stuff[i].down2<<endl;
            }
        }
        }
        vector<int> vecres;
        finalans.push_back(m_res);


    }

    int max_ans=0;
    for(int i=0;i<finalans.size();i++){
        if(finalans[i] > max_ans) max_ans = finalans[i];
    }
    cout<<"Final Answer:"<<max_ans<<endl;
    system("Pause");
}
```

Problem32.cpp

```
int patharray[40321][8];
int typearray[256][8];
int row=0;

int COST[4][8]=
```

```
    {

(RGV2CNC1),(RGV2CNC2),(RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),(RGV2CNC1+MOV
E2),(RGV2CNC2+MOVE2),(RGV2CNC1+MOVE3),(RGV2CNC2+MOVE3),

(RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),(RGV2CNC1),(RGV2CNC2),(RGV2CNC1+MOV
E1),(RGV2CNC2+MOVE1),(RGV2CNC1+MOVE2),(RGV2CNC2+MOVE2),

(RGV2CNC1+MOVE2),(RGV2CNC2+MOVE2),(RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),(
RGV2CNC1),(RGV2CNC2),(RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),

(RGV2CNC1+MOVE3),(RGV2CNC2+MOVE3),(RGV2CNC1+MOVE2),(RGV2CNC2+MOVE2),(
RGV2CNC1+MOVE1),(RGV2CNC2+MOVE1),(RGV2CNC1),(RGV2CNC2)
    };
class RandomNumber{
public:
    RandomNumber(){
        srand(time(0));
    }
    int get(int begin = 0, int end = 1){
        return rand()%(end-begin+1)+begin;
    }
};

int Breakdown(int len)
{
    //srand((int)time(0));
    int L = 101*len-1;
    // RandomNumber r;
```

```
    // int x=r.get(0,L+1);
    int x =rand()%(L+1);
    //if return value x is -1,there is no error;
    //else, it is the exact time when the error happens.
    x=(x<=28000&&x>=(28000-len+1))?(28000-x):(-1);

    return x;
}
struct break_info{
    int st;
    int cnc;
    int start;
    int solve;
};
break_info breaks[1000];
int breaktime = 0;

struct stuff_record{
    int flag;//flag = 1 孰料
    int cnc1;
    int up1;
    int down1;
    int cnc2;
    int up2;
    int down2;
};

int TCount=0,Count=0,PCount=0;
int CNCcount[8]={0};
```

```
bool first[8]={true};
stuff_record stuff[500];
int stuffcount = 0;
int record[8];//记录CNC的物件编号

int Work(int div,int path,int len){
    stuffcount = 0;
    int state[8];
    int timeleft[8];
    int stime[8]={-1,-1,-1,-1,-1,-1,-1,-1};
    int location=0;
    breaktime = 0;
    TCount = 0; Count = 0; PCount = 0;
    //标记分工情况
    int type[8];
    for(int i=0;i<8;i++){
        state[i]=0;
        CNCcount[i] = 0;
        stime[i] = -1;
        timeleft[i] = 0;
        record[i] = -1;
        first[i] = true;
        type[i] = typearray[div][i];
    }
    //初始化序列
    vector<int> initvec;
    for(int i=0;i<len;i++){
        initvec.push_back(patharray[path][len-1-i]);
    }
```

```
bool RGVstate=0;//0:表示RGV上是空的，下一个应取半成品；1：表示RGV上有一个半成品，需要去2类机
int lastrough=-1;
while(1){
    //计算每个CNC完成工作还需要的时间；公式为：剩余时间=当前时间-开始时间；
    for(int i=0;i<8;i++){
        if(stime[i] == -1){
            state[i] = 0;
            continue;
        }

        int tmp = TCount-stime[i];
        if (type[i] == 1) timeleft[i] = tmp>CNCT1 ? 0 : (CNCT1-tmp);
        else timeleft[i] = tmp>CNCT2 ? 0 : (CNCT2-tmp);
    }

    //检查各个CNC的状态
    for(int i=0;i<8;i++){
        if(timeleft[i] == 0) state[i] = 0;
        else state[i] = 1;
    }

    //检查是否有上下料需求
    vector<int> freeCNC1;//第一类的空闲
    vector<int> freeCNC2;//第二类的空闲
```

```
for(int i=0;i<8;i++){
    if(state[i]==0) {
        if(type[i] == 1) freeCNC1.push_back(i);
        else freeCNC2.push_back(i);
    }
}

if( (RGVstate==0) && freeCNC1.empty()){
    //所有CNC都在工作，无上下料需求
    //找到最快完成工作的CNC，时间跟进
    int min_time=1000000,min_i=0;
    for(int i=0;i<8;i++){
        if( (type[i] == 1) && (timeleft[i]<min_time) ){
            min_time = timeleft[i];
            min_i = i;
        }
    }
    PCount += 1;
    //时间流逝......
    TCount += min_time;

}
else{
    if(RGVstate==0){
        //从队列里寻找下一个去的1类机的位置
        int target;
        if(!initvec.empty()){
            target = initvec[initvec.size()-1];
            initvec.pop_back();
```

```
                }
                else {
                    vector<int> startvec;
                    for(int i=0;i<freeCNC1.size();i++){
                        startvec.push_back(stime[freeCNC1[i]]);
                    }

                    target          =          NextDecision(location,
freeCNC1,startvec);//tag1
                }
                int gap = COST[location][target];
                //前往目标
                TCount += gap;
                stime[target] = TCount;
                state[target] = 1;
                location = (target)/2;

                stuff[stuffcount].cnc1  =  target+1;// 上 料 ， 编 号
stuffcount;
                int laststuff = record[target];
                record[target] = stuffcount;
                stuff[stuffcount].up1 = TCount;

                int b = Breakdown(CNCT);
                if(b >= 0){
                    // cout<<"Break here!"<<endl;
                    // system("Pause");
                    breaks[breaktime].cnc = target+1;
                    breaks[breaktime].st = stuffcount;
```

```
                    breaks[breaktime].start = TCount+b;



                    RandomNumber rs;
                    //int breaklen = 600 + rand()%600;
                    int breaklen = rs.get(600,1200);
                    breaks[breaktime].solve = breaks[breaktime].start
+ breaklen;

                    stime[target] += (breaklen + b - CNCT1);
                    stuff[stuffcount].flag = 1;//报废的物料
                    stuff[stuffcount].down1 = breaks[breaktime].start;

                    breaktime++;
                    first[target] = true;

                }

                stuffcount++;

                if(!first[target]){
                    //取下半成品的情况
                    //Count += 1;
                    CNCcount[target]++;
                    stuff[laststuff].down1 = TCount;

                    RGVstate = 1;//RGV 手拿半成品
                    lastrough  =  laststuff;//记录处于 RGV 手中的半成品
工件
```

```
                }
                else{
                    first[target] = false;
                    //RGV 手中无半成品
                }

            }

            else{
                //寻找可用的 2 类机的位置
                //如果没有闲置的 2 类机，则等待：
                if(freeCNC2.empty()){
                    int min_time=CNCT2,min_i=0;
                    for(int i=0;i<8;i++){
                        if(       (type[i]       ==       2)       &&
(timeleft[i]<min_time) ){
                            min_time = timeleft[i];
                            min_i = i;
                        }
                    }
                    PCount += 1;
                    //时间流逝......
                    TCount += min_time;
                }
                //如果有空闲的二类机，寻找最近的那个
                else{
                    vector<int> startvec;
                    for(int i=0;i<freeCNC2.size();i++){
                        startvec.push_back(stime[freeCNC2[i]]);
```

```
                    }

                    int      target      =      NextDecision(location,
freeCNC2,startvec);
                    int gap = COST[location][target];

                    //前往目标
                    TCount += gap;
                    stime[target] = TCount;
                    state[target] = 1;
                    location = (target)/2;
                    RGVstate = 0;

                    int laststuff = record[target];
                    record[target] = lastrough;
                    stuff[lastrough].cnc2 = target;
                    stuff[lastrough].up2 = TCount;

                    int b = Breakdown(CNCT);
                    if(b >= 0){
                        // cout<<"Break here!"<<endl;
                        // system("Pause");
                        breaks[breaktime].cnc = target+1;
                        breaks[breaktime].st = stuffcount;
                        breaks[breaktime].start = TCount+b;

                        RandomNumber rs;
                        //int breaklen = 600 + rand()%600;
                        int breaklen = rs.get(600,1200);
```

```
                        breaks[breaktime].solve                         =
breaks[breaktime].start + breaklen;

                        stime[target] += (breaklen + b - CNCT2);
                        stuff[stuffcount].flag = 1;//报废的物料
                        stuff[stuffcount].down2                         =
breaks[breaktime].start;

                        breaktime++;
                    }

                    if(!first[target]){
                        TCount += RGVCLEAN;
                        Count += 1;
                        CNCcount[target]++;
                        stuff[laststuff].down2 = TCount;
                    }
                    else{
                        first[target] = false;
                    }

                    if(b >= 0) first[target] = true;

                    if(TCount >= 28800) return Count;
                }
            }
        }
    }
}
```

```
bool allsame(int t[8]){
    for(int i=1;i<8;i++){
        if(t[i] != t[0]) return false;
    }

    return true;
}

bool ispath(int path,vector<int> tp){
    for(int i=0;i<tp.size();i++){
        if(tp[i] != patharray[path][i]+1) return false;
    }
    return true;
}
```

Gentel.m

```
clear;
axis([0,2400,0,8.5]);%x 轴 y 轴的范围
set(gca,'xtick',0:125:2400) ;%x 轴的增长幅度
set(gca,'ytick',0:1:8.5) ;%y 轴的增长幅度
xlabel('加工时间','FontName','微软雅黑','Color','k','FontSize',14)
ylabel('机器号','FontName','微软雅黑','Color','k','FontSize',14,'Rotation',90)
title('Problem 1: 一道工序 第 1 组数据','fontname','微软雅黑','Color','k','FontSize',14);
n_bay_nb = 8;%机器数目
n_task_nb = 24;%任务数目
%x 轴 对应于每个工序的开始时间
n_start_time=[28 76 124 172 203 254 305 356 616 689 762 835 891 967
```

```
1043 1119 1204 1297 1370 1443 1499 1578 1654 1730];
%length 对应于在 x 轴方向的长度即每个工序的持续时间
n_duration_time =[560 560 560 560 560 560 560 560 560 560 560 560 560
560 560 560 560 560 560 560 560 560 560 560];
%y 轴 对应于机器号
n_bay_start=[1 3 5 7 8 6 4 2 1 3 5 7 8 6 4 2 1 3 5 7 8 6 4 2];
%可以根据机器号选择使用哪一种颜色
n_job_id=[0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7];
rec=[0,0,0,0];
color=['r','g','b','c','w','y','k','m'];
for i =1:n_task_nb
  rec(1) = n_start_time(i); %矩形的横坐标
  rec(2) = n_bay_start(i)-0.25;  %矩形的纵坐标
  rec(3) = n_duration_time(i);  %矩形的长
  rec(4) = 0.5; %矩形的宽
  txt1=sprintf('%d',n_start_time(i));%标注开始时间
  txt2=sprintf('%d',n_start_time(i)+560);%标注结束时间
  rectangle('Position',rec,'LineWidth',0.5,'LineStyle','-
','FaceColor',color(n_job_id(i)+1));
  text(n_start_time(i),(n_bay_start(i)-
0.35),txt1,'FontWeight','Bold','FontSize',12);
  text(n_start_time(i)+490,(n_bay_start(i)-
0.35),txt2,'FontWeight','Bold','FontSize',12);
end
```