智能 RGV 的动态调度模型

摘要

本文通过建立智能 RGV 的贪心算法动态调度模型，利用 C++编程进行模型模拟，给出了智能 RGV 在三种不同的具体情况下的调度策略和系统的作业效率。针对第一种具体情况，我们利用 C++中类的抽象特性把 RVG 以及 CNC 抽象为不同的类，每种类含有常量、变量（如位置量，时间量等）、行为函数（如 RGV 的移动行为，上下料行为，等待行为，CNC 的工作行为）。各种类的常量、变量与行为函数会发生改变，在智能 RGV 的状态改变上（即动态调度），我们采用了基于 ERD 算法（尽可能早地调度先到达的作业）和 SPT 算法（尽可能早地调度作业长度短的作业）的智能 RGV 动态调度贪心算法模型。与此同时我们发现在第一种具体情况的第一组数据中，是存在循环规律的，我们通过该循环规律得到了第一种具体情况下的第一组数据的运行规律。随后，我们利用构建的智能 RGV 动态调度贪心算法模型进行编程处理。将题中所给的表 1 中系统作业参数的 3 组数据代入程序，求解出了程序模型下的 RGV 的调度策略和系统的作业效率。最后我们用程序代码运行出的结果与循环规律得出的结果进行比较，从而验证了程序的正确性。

针对第二种具体情况，我们把基于贪心算法的智能 RGV 动态调度模型进行了改进，加入了加工两道工序所必须的常量、变量和行为函数，保证了贪心算法的每步最优性。在建模的过程中，我们发现在第二种具体情况中各台 CNC 的刀具类型对智能调度系统的工作效率有影响。因此为了得到更优的结果，我们讨论了各台 CNC 的刀具类型的选择策略，得到了刀具类型分配的选择模型。针对得到的新模型，我们进行了 C++程序编译并利用题中所给数据求解出了第二种具体情况下智能 RGV 的调度策略和系统工作效率。

针对第三种具体情况，我们在第一种和第二种情况的基础上运用对 CNC 加入了故障和维修的行为，当贪心算法进行最优解判断时，我们对其进行了一次最优解故障判断，从而让 RGV 避开故障 CNC。此外，当 CNC 开始工作时，我们用基于随机函数的算法对 CNC 进行了一次判断，判断 CNC 是否故障，故障以后我们设置了参数使 CNC 会发生维修的行为，接着我们利用题中所给数据求解出了第三种具体情况下的单次智能 RGV 的调度策略和系统工作效率，以及故障发生数，故障发生的节点。除了依照 RGV 和 CNC 的动态行为得到的改进模型，我们还讨论了一种基于固定故障序号的静态改进模型，这种模型提前确定会发生故障的物料序号，能够制定不同故障发生情况下的应急调度，具有较特别的理论价值。

最后，我们给出了模型的优缺点，还给出了模型的评价和展望。

关键词：RGV 贪心算法 单机调度 ERD 算法 SPT 算法 C++

1.问题重述

问题给出了一个智能加工系统的示意图，该智能加工系统由 8 台计算机数控机床、1 辆轨道式自动引导车、1 条 RGV 直线轨道、1 条上料传送带、1 条下料传送带等附属设备组成。RGV 是一种无人驾驶、能在固定轨道上自由运行的智能车。它能够根据指令自动控制移动方向和距离，并能够完成上下料以及清洗物料等任务作业。针对三种具体情况：

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

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

（3）CNC 在加工过程中可能发生故障（据统计：故障的发生概率约为 1%）的情况，每次故障排除（人工处理，未完成的物料报废）时间介于 10~20 分钟之间，故障排除后即刻加入作业序列。要求分别考虑一道工序和两道工序的物料加工作业情况。

完成两项任务：

任务 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；

任务 2：利用表 1 中系统作业参数的 3 组数据分别效检验模型的实用性和算法的有性，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。

2.模型假设与约定

1)将轨道分为四个位置，CNC1 与 CNC2 位于轨道位置 0 处，CNC3 与 CNC4 位于轨道位置 1 处，CNC5 与 CNC6 位于轨道位置 2 处，CNC7 与 CNC8 位于轨道 3 处；

（2）时间的最短间隔为 1S；

（3）除了第三种具体情况中的 CNC 会发生故障以外，其它具体情况中智能加工系统的其余部件均运行正常；

（4）开始时刻 RGV 在轨道位置 0 处，且开始时刻所有的 CNC 都处于空闲状态，没有装载任何物料；

（5）假设第三种具体情况中 CNC 发生故障的概率是 1%，且维修的时间服从均匀分布；

（6）每一个 CNC 发生故障的概率都是相同的，没有差异和记忆性

（7）若 CNC 在这一次作业中会发生故障，则故障发生时刻为开始工作的时刻

（8）约定在第一轮上料过程中，CNC 不会发生故障

3.符号说明及名词定义

符号	单位	含义
position	-	RVG 当前位置
now_cnc	-	RVG 当前目标
rgv_flag	-	RGV 有无熟料的判断参数
$posCalculate$	S	RGV 的移动时间
n	-	加工物序列号
$STEP1$	S	RGV 移动一步的时间
$STEP2$	S	RGV 移动两步的时间
$STEP3$	S	RGV 移动三步的时间
$CLEAN$	S	RGV 进行清洗操作的时间
$move$	-	RGV 移动
$load$	-	RGV 上下料
$clean$	-	RGV 清洗
$wait$	-	RGV 等待
$CNC::n$	-	加工物序列号
$count$	S	CNC 剩余工作时间
$number$	-	CNC 编号

$CNC::position$	-	CNC 位置
$CNC\_WORKTIME$	s	CNC 加工一道工序的时间
$CNC1$	s	奇数 CNC 的上下料操作时间
$CNC0$	s	偶数 CNC 的上下料操作时间
$CNCNUMBER$	-	CNC 的数量
$countdown$	-	为了让 CNC 和 RGV 的时间同步的函数
P	$\%$	RGV 的调度系统的作业效率
M	s	各台 CNC 的总空闲时间之和
M_i	s	编号为 i 的 CNC 总空闲时间
v	-	效率平衡系数
U	-	效率平衡残数
q_1	-	装有第二种刀具类型的 CNC 数量
q_2	-	装有第二种刀具类型的 CNC 数量
T_1	-	CNC 加工第一道工序的时间
T_2	-	CNC 加工第二道工序的时间

4.模型的建立与求解

4.1 针对第一种具体情况的建模以及算法求解

因为智能加工系统的每班次作业时间固定，为 8 小时，而所有的熟料均由 CNC 加工而成，若各台 CNC 的空闲时间总和 M 越小，则系统的作业效率的工作效率越高，且工作效率为

$$P = 100\% - \frac{M}{8h * 8} \qquad (4.1\text{-}1)$$

因而我们的智能加工系统建模算法的目标函数记为

$$\min(f) = \min(M) = \min(\sum M_i)$$

4.1.1 智能加工系统的抽象量化

为了建立第一种具体情况的模型，我们对 RGV 和 CNC#进行了必要的抽象量化

（1） RGV 的抽象量化

	符号类型	符号	说明
RGV	变量	position	RVG 当前位置
		now_cnc	RVG 当前目标
		rgv_flag	RGV 有无熟料的判断参数
		$posCalculate$	RGV 的移动时间
		n	加工物序列号
	常量	$STEP1$	RGV 移动一步的时间
		$STEP2$	RGV 移动两步的时间
		$STEP3$	RGV 移动三步的时间
		$CLEAN$	RGV 进行清洗操作的时间
	行为函数	$move$	RGV 移动
		$load$	RGV 上下料

		clean	RGV 清洗
		wait	RGV 等待

（2）CNC 的抽象量化

	符号类型	符号	说明
CNC	变量	$CNC :: position$	加工物序列号
		count	CNC 剩余工作时间
	常量	*number*	CNC 编号
		$CNC :: position$	CNC 位置
		$CNC\_WORKTIME$	CNC 加工一道工序的时间
		$CNC1$	奇数 CNC 的上下料操作时间
		$CNC0$	偶数 CNC 的上下料操作时间
		CNCNUMBER	CNC 的数量
	行为函数	*countdown*	为了让 CNC 和 RGV 的时间同步

4.1.2 RGV 的行为函数的详细解释

（1） $RGV :: RGV()$

RGV 类构造函数，定义 RGV 类对象时实现成员变量的初始化。
具体成员变量初始化情况：RGV 初始位置 Position=0，初始最优解（RGV 当前操作 CNC 台的编号）now_cnc =1，RGV 初始状态 rgv_flag 空闲，加工总耗时为 0，加工熟料总数为 0。

（2） $void\ RGV::Init(CNC\ *p)$

RGV 第一轮作业情况特殊，仅需考虑 RGV“移动”和“上料”动作。

（3） $int\ RGV::posCalculate(int,int)$

该函数返回 RGV 从位置 1pos1 移动到位置 2pos2 需要的时间。主要由函数 RGV：：move 调用。

（4） $void\ RGV::move(CNC\ *p)$

该函数抽象 RGV 类的“运动”作业。①先用“贪心算法”找到 RGV 下一移动目标的最优解 $next\_ci$；②比较当前对象 now_ci 和最优对象 $next\_ci$，如果当前对象不是最优对象，则移动向最优对象；③判断是否需要等待，如果最优 CNC 工作台当前忙碌（CNC 工作剩余时间 count 不为 0），则等待。

（5） $void\ RGV::load(CNC\ *p)$

该函数抽象 RGV 类的“上下料”作业。①判断当前 CNC 工作台编号的奇偶性，选择奇数号上下料时间 CNC1 还是偶数号上下料时间 CNC2；②完成“上下料”作业的同时，将此时 RGV 中的“加工物序列总数 RGV::n”和“CNC 加工完成一个一道工序的物料所需时间 $CNC\_WORKTIME$”赋给当前 CNC 工作台的“加工物序号 $CNC::n$”和“加工剩余时间 $CNC::count$”。

（6） $void\ RGV::clean(CNC*\ p)$

该函数抽象 RGV 类的“清洗”作业。将“完成清洗时间 CLEAN”同步到“总时间”和所有 CNC“加工剩余时间 $CNC::count$”。

（7） $void\ RGV::wait(CNC*\ p)$

该函数抽象 RGV 类的“等待”作业。将“完成清洗时间 CLEAN”同步到“总时间”和所有 CNC“加工剩余时间 $CNC::count$”。

4.1.3 CNC 的行为函数的详细解释

（8） RGV::RGV()

RGV 类构造函数，定义 RGV 类对象时实现成员变量的初始化。

具体成员变量初始化情况：RGV 初始位置 Position=0，初始最优解（RGV 当前操

作 CNC 台的编号）now_cnc=1，RGV 初始状态 rgv_flag 空闲，加工总耗时为 0，加工熟料总数为 0。

（9）void RGV::Init(CNC *p)

RGV 第一轮作业情况特殊，仅需考虑 RGV“移动”和“上料”动作。

（10）int RGV::posCalculate(int pos1, int pos2)

该函数返回 RGV 从位置 1pos1 移动到位置 2pos2 需要的时间。主要由函数 RGV：：move 调用。

（11）void RGV::move(CNC *p)

该函数抽象 RGV 类的“运动”作业。①先用“贪心算法”找到 RGV 下一移动目标的最优解 next_cnc；②比较当前对象 now_cnc 和最优对象 next_cnc，如果当前对象不是最优对象，则移动向最优对象；③判断是否需要等待，如果最优 CNC 工作台当前忙碌（CNC 工作剩余时间 count 不为 0），则等待。

（12）void RGV::load(CNC *p)

该函数抽象 RGV 类的“上下料”作业。①判断当前 CNC 工作台编号的奇偶性，选择奇数号上下料时间 CNC1 还是偶数号上下料时间 CNC2；②完成“上下料”作业的同时，将此时 RGV 中的“加工物序列总数 RGV::n”和“CNC 加工完成一个一道工序的物料所需时间 CNC_WORKTIME”赋给当前 CNC 工作台的“加工物序号 CNC::n”和“加工剩余时间 CNC::count”。

（13）void RGV::clean(CNC* p)

该函数抽象 RGV 类的“清洗”作业。将“完成清洗时间 CLEAN”同步到“总时间”和所有 CNC“加工剩余时间 CNC::count”。

（14）void RGV::wait(CNC* p)

该函数抽象 RGV 类的“等待”作业。将“完成清洗时间 CLEAN”同步到“总时间”和所有 CNC“加工剩余时间 CNC::count”。

4.1.4 智能 RGV 的动态调度策略与算法

对于智能 RGV 的动态调动策略，我们选择采用基于 ERD 算法和 SPT 算法的贪心算法求取智能 RGV 动态调度策略的局部最优解。

当 RGV 空闲时，选择判断当前的目标 CNC_m 是否为局部最优解，即判断 当前的 M_i 是否为最小值，判断情况根据基于 ERD 算法和 SPT 算法的贪心算法有两种抽象情况

（1）先来先服务法（ERD 算法）

尽可能早地调度先到达的作业，以充分利用各作业到达时刻之间的时间段。

如：当前目标 CNC_m 的工作剩余时间为 $count_m$，比较目标 CNC_j 的工作剩余时间为 $count_j$，我们选择工作剩余时间为 $min\{count_{m,}count_j\}$ 的目标。

（2）短作业优先法（SPT 算法）

尽可能早地调度作业长度短的作业(简称短作业)，使得其完成时间对后续

作业完成时间的影响较小, 从而获得较小的作业完成时间之和。
如：当 RGV 能够同时响应奇、偶 CNC 的上下料操作时，∵CNC1<CNC0 ，∴RGV 优先响应奇数 CNC

但是当考虑实际情况时，需要考虑基于结合 ERD 算法和 SPT 算法的贪心算法，具体实际情况有如下四种：

（1）RGV 的当前目标 CNC_m 和比较目标 CNC_j 均为工作状态，若不等式

$$max\{count_m, t_{p,m}\} < max\{count_j, t_{p,j}\} \quad (4.1.4\text{-}1)$$

成立，则当前目标 CNC_m 为局部最优解，否则选取比较目标 CNC_j 为当前目标

（式中：$count_m$ 表示当前目标 CNC_m 工作剩余时间，$count_j$ 表示比较目标工作剩余时间，$t_{p,m}$ 表示 RGV 从目前位置 p 到达 CNC_m 位置的时间，$t_{p,j}$ 表示 RGV 从目前位置 p 到达 $count_j$ 位置的时间）

（2） RGV 的当前目标 CNC_m 和比较目标 CNC_j 均为空闲状态，
若不等式

$$t_{p,m} < t_{p,j} \quad (4.1.4\text{-}2)$$

（式中参数含义同（1））成立，则当前目标 CNC_m 为局部最优解，否则选取比较目标 CNC_j 为当前目标

（3） RGV 的当前目标 CNC_m 为工作状态，比较目标 CNC_j 为等待状态，
若不等式

$$max\{count_m, t_{p,m}\} < t_{p,j} \quad (4.1.4\text{-}3)$$

（式中参数含义同（1））成立，则当前目标 CNC_m 为局部最优解，否则选取比较目标 CNC_j 为当前目标

（4） RGV 的当前目标 CNC_m 为等待状态，比较目标 CNC_j 为工作状态，
若不等式 ：

$$t_{p,m} < max\{count_j, t_{p,j}\} \quad (4.1.4\text{-}4)$$

（式中参数含义同（1））成立，则当前目标 CNC_m 为局部最优解，否则选取比较目标 CNC_j 为当前目标

每次空闲时，将当前目标 CNC_m 与其余七个 CNC 目标进行比较，得出最优的当前目标。

对于以上算法过程，我们选取第一种具体情况和第一组数据进行算法模拟。

以第 1 组数据为例说明：

以给编号 1 的 CNC 上料完后为时间起点，t=0。此时 C1 需要 560s 后才能加工完毕。到第二个起点所花费的时间为：小车路程所花费的时间 3×20s+46s=106s；小车给其他 CNC 上下料所花费时间 3×28s+4×31s=208s；总时间 208s+106s=314s。t=314s 时，小车回到 1 位置，等待 C1 加工完毕。t=560s 时，小车给 C1 上下料，到第二个起点时，t=588s。

此后小车到第三个循环起点的时间为：小车移动花费时间 106s;给其他 CNC 上下料时间 208；清洗花费时间 8×25=200；总时间 514<560,可见，循环时间应为 560s+28s=588s。标准循环次数 n 满足：

$$588n < 8\times 60\times 60 - 2$$

$$n < 48.98$$

取 n=48，第 48 次循环结束时，t=588×48=28224.，到完成 C8 的熟料清洗时，t=28224+25×8(清洗时间)+3×28＋4×31（上下料时间）+3×20（RGV 移动时间）=28692<28820，即完成最后的熟料清洗。注意，由于第一次循环没有获得熟料，所以还需要减去 8 个。

则 8h 总共获得清洗完成的熟料为 8×48=384 个。上述 RGV 的调度策略如下图所示

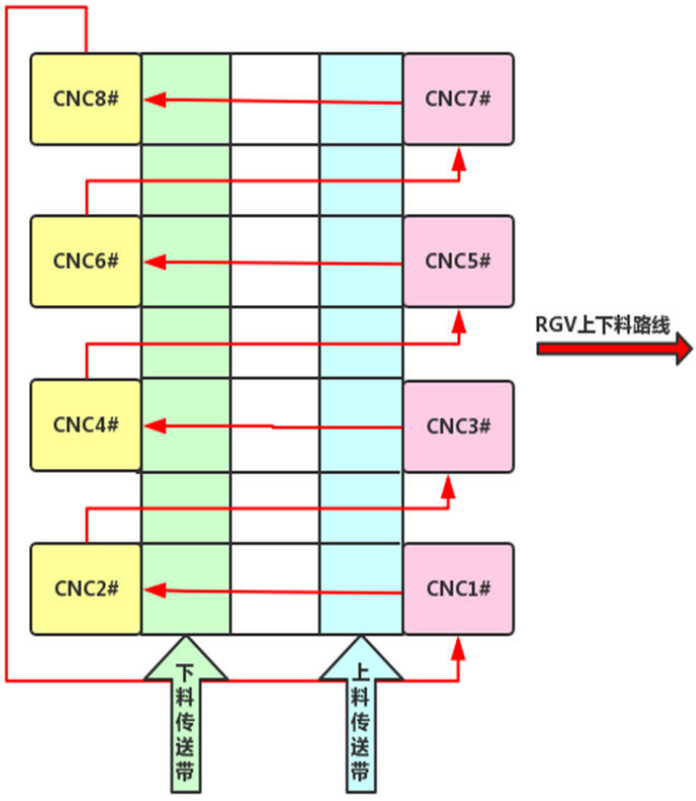


图 1

4.1.5 智能 RGV 的动态调度策略的代码实现

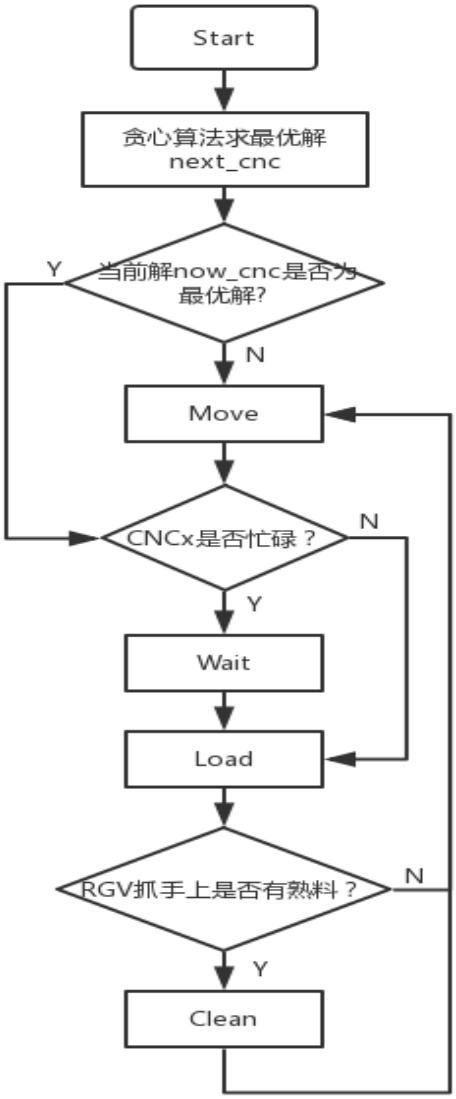


图 2

4.1.6 模型求解与分析

我们通过 C++针对上述模型进行了编程，程序附于目录。将题中所给的三组数据代入程序，我们可以求解出对应三种具体情况的智能 RGV 的动态调度方案，我们将所求得的调度方案写在了题中要求的 EXCEL 表格中，并利用求得的调度方案，做出了一部分调度情况的甘特图，效果如下图所示：

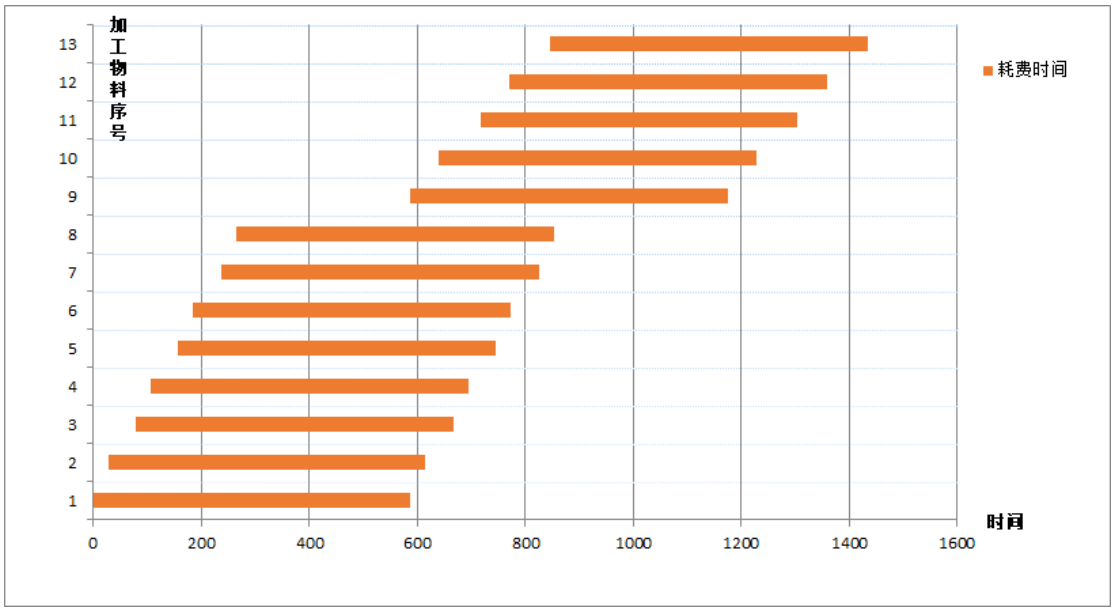


图 3

如甘特图所示，我们将得到的调度方案与 4.1.4 推理中得到的调度方案进行比较，发现两者基本一样，从而验证了程序模型的正确性。

4.2 第二种具体情况的求解

4.2.1 第二种具体情况的抽象量化

在第二种具体情况中我们所用基本模型沿用 4.1 中所用模型。针对第二种具体情况，主要发生的变化在于将物料从生料加工为熟料的工序从一道变成了两道，每一个 CNC 只能加工一道工序，在 RGV 应答 CNC 时，其上可能没有物料，也可能有加工过第一道工序的半成品物料。因此我们需要对第二种具体情况的改变之处进行新的抽象量化。具体添加的常量、变量与函数如下表格所示：

4.1.2 RGV 的成员函数的详细解释

（15）RGV::RGV()

RGV 类构造函数，定义 RGV 类对象时实现成员变量的初始化。

具体成员变量初始化情况：RGV 初始位置 Position=0，初始最优解（RGV 当前操作 CNC 台的编号）now_cnc=1，RGV 初始状态 rgv_flag 空闲，加工总耗时为 0，加工熟料总数为 0。

（16）void RGV::Init(CNC *p)

RGV 第一轮作业情况特殊，仅需考虑 RGV“移动”和“上料”动作。

（17）int RGV::posCalculate(int pos1, int pos2)

该函数返回 RGV 从位置 1pos1 移动到位置 2pos2 需要的时间。主要由函数 RGV：：move 调用。

（18）void RGV::move(CNC *p)

该函数抽象 RGV 类的“运动”作业。①先用“贪心算法”找到 RGV 下一移动目标的最优解 next_cnc；②比较当前对象 now_cnc 和最优对象 next_cnc，如果当前对象不是最优对象，则移动向最优对象；③判断是否需要等待，如果最优 CNC 工作台当前忙碌（CNC 工作剩余时间 count 不为 0），则等待。

（19）void RGV::load(CNC *p)

该函数抽象 RGV 类的“上下料”作业。①判断当前 CNC 工作台编号的奇偶性，选择奇数号上下料时间 CNC1 还是偶数号上下料时间 CNC2；②完成“上下料”作业的同时，将此时 RGV 中的“加工物序列总数 RGV::n”和“CNC 加工完成一个一道工序的物料所需时间 CNC_WORKTIME”赋给当前 CNC 工作台的“加工物序号 CNC::n”和“加工剩余时间 CNC::count”。

（20）void RGV::clean(CNC* p)

该函数抽象 RGV 类的“清洗”作业。将“完成清洗时间 CLEAN”同步到“总时间”和所有 CNC“加工剩余时间 CNC::count”。

（21）void RGV::wait(CNC* p)

该函数抽象 RGV 类的“等待”作业。将“完成清洗时间 CLEAN”同步到“总时间”和所有 CNC“加工剩余时间 CNC::count”。

4.1.3 CNC 的成员函数的详细解释

（22）CNC::CNC(int num, int pos)

CNC 类构造函数，要求在初始化 CNC 类对象时必须进行显性初始化。显性初始化方便用户使用程序时进行查错，且初始化后对象的属性不会在程序运行时发生改变，提高了程序的稳定性。

具体成员变量初始化情况：CNC 初始编号 number=0；CNC 初始位置 position=0；CNC 剩余加工时间 count；CNC::n 当前加工无序列号。

（23）void CNC::countdown(int temp)

该函数的作用是同步 CNC 类与 RGV 类的时间变化。输入形参 temp 为调用该函数时变化的时间值。

4.2.2 刀具类型分配的选择模型

在第二种情况中，我们需要首先确定各台 CNC 的刀具类型，CNC 所有的刀具

类型有 2^8-2 种情况（总共有八台 CNC，每台机器有两种刀具情况，排列组合得共有 2^8，但是所有 CNC 均为同一种刀具的情况显然不成立，所以减去得到 2^8-2 种情况）。这个数字并不大，可以运用穷举法确定最优的刀具类型，但是当 CNC 的数量增加时，情况总数呈现指数变化，这种情况下，穷举法显然不具有普遍性。因此我们制定了一套使得智能加工系统的作业效率较高的选择模型。

定义装有第一种刀具的 CNC 数量为 q_1,装有第二种刀具的 CNC 数量为 q_2,，T_1 为 CNC 加工第一道工序的时间，T_2 为 CNC 加工第二道工序的时间。

接着考虑刀具的分配方案，当分配刀具处于极端状态时，系统效率显然是低下的，比如最极端的情况，所有 CNC 均只装配同一种刀具，系统效率为 0。又因为第一道工序的用时和第二道工序的用时不同，所以平均分配显然也会导致效率降低，使得加工工序用时更大的 CNC 工作次数少，导致系统工作效率降低。只有为加工时长更长的工序分配更多一些的 CNC，才能使得加工时长更短的工序完成后，更快地进行系统调度，减少 CNC 的总空闲时间 M。

为此我们定义 v 为效率平衡系数，U 为效率平衡残数，U 越小时系统效率越大。我们有

$$v = \frac{q_2 * T_1}{q_1 * T_2} \qquad (4.2.2\text{-}1)$$

$$U = |v - 1| \qquad (4.2.2\text{-}2)$$

目标函数为 $F = min\{U\}$，

并满足约束条件：$q_1 + q_2 = 8$（$1 \le q_1, q_2 \le 7$，且 q_1, q_2 为整数）

T_1，T_2 为定值

这个问题显然是可解的，我们对第二组数据进行举例说明

$$v = \frac{280 * q_2}{500 * q_1}, q_1 + q_2 = 8 \quad (1 \le q_1, q_2 \le 7，且 q_1, q_2 为整数)，$$

$$U = |(1 - 280 * (8 - q_1) / 500 * q_1| \qquad (4.2.2\text{-}3)$$

根据函数 U 的单调性，我们可以求得 q_2=5，q_1=3 时,U 最小。所以得到装有第一种刀具的 CNC 数量应为 5 个，装有第二种刀具的 CNC 数量应为 3 个。

此外对于装有第一种刀具 CNC 和装有第二种刀具 CNC 的位置，第一步我们先采用 SPT 算法，将花费时间更少的工序安排在上下料时间更短的奇数 CNC 上加工。则由第一步，对同样的第二组数据，安装不同刀具的 CNC 有如下两种相对位置

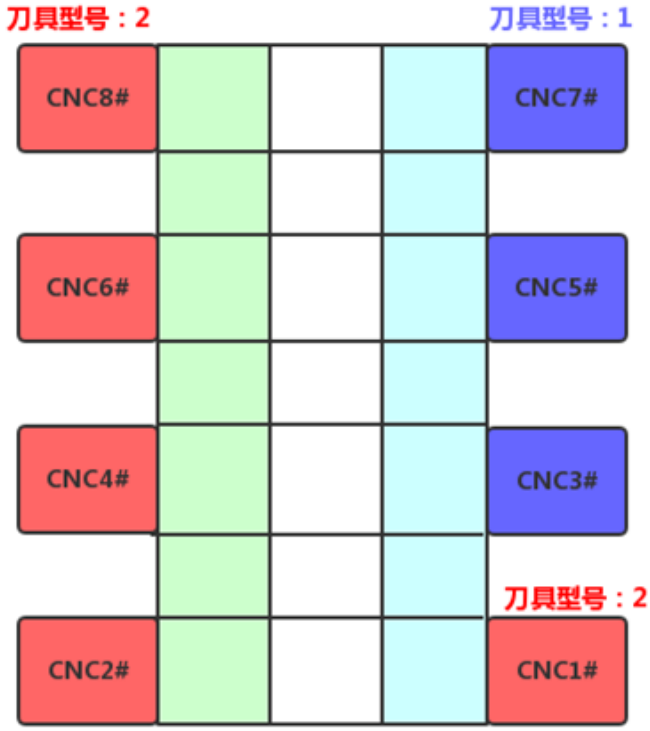

刀具型号：2
刀具型号：1
CNC8#
CNC7#
CNC6#
CNC5#
CNC4#
CNC3#
刀具型号：2
CNC2#
CNC1#

图 4


刀具型号：2
刀具型号：1
CNC8#
CNC7#
CNC6#
CNC5#
CNC4#
CNC3#
刀具型号：2
CNC2#
CNC1#

图 5

其中 1 代表这一台 CNC 装配第一种刀具，2 代表这一台 CNC 装配第二种刀具。然后我们计算从装配第一种刀具的各个 CNC 位置分别到装配第二种刀具的各个 CNC 位置的时间和。在我们所举的例子中，第一种情况下，时间和

$$S1=\sum(6STEP1+4STEP2+2STEP3)=\left(6*23s+4*41s+2*59s\right)=420s \quad (4.2.2\text{-}4)$$

（其中 STEP1、STEP2、STEP3 表示 RGV 走 1、2、3 步所花的时间）同样地，我们得到 S2=420s,S1=S2 则认为两者效率相同。此外若 S1<S2 则，第一种相对位置的刀具安排比第二种相对位置的刀具安排更优，反正第二种相对位置的刀具安排更优。

通过以上的选择模型我们选择了较优的安排策略，用该安排策略对我们的程序赋值，可以得到较优的安排策略与工作效率。我们将其作为结果存在了附件的 EXCEL 表格当中。

4.2.3 第二种具体情况的算法实现

如图所示：

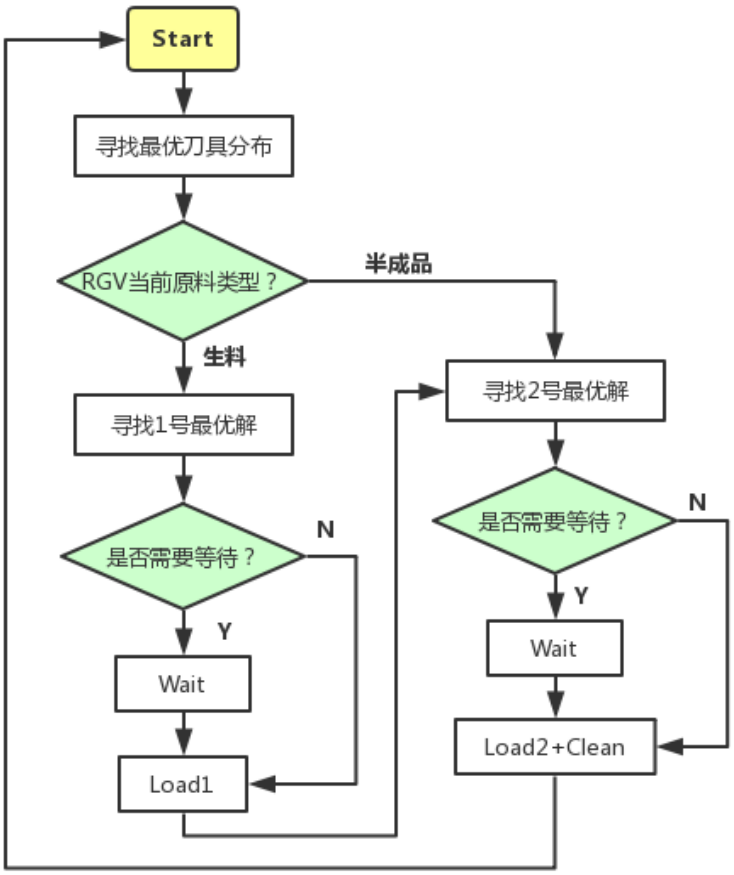


图 6

4.3 第三种具体情况的求解

4.3.1 故障判断和维修时间的确定模型

针对第三种具体情况，我们首先确定故障判断和维修时间的实现。
对于故障判断，我们选择为每一个 CNC 工作的时刻生成一个从 0 到 100 的随机数 a，如果生成的随机数 a<1，即该时刻会发生故障。
而对于维修时间，我们同样采用了随机数的方法，生成了一个从 600 到 1200 的随机数 b，b 代表该次维修所花的秒数。

4.3.2 基于智能 RGV 动态过程的改进模型

在这个模型中，我们对 RGV 与 CNC 赋予了更多的智能，使得它们能够识别故障行为。具体过程为：当利用贪心算法确定最优解时，我们会进行最优解是否故障的判断，如果判断为真，则跳过该解，重新进行利用贪心算法确定优解的步骤。如果判断为假，则按照 4.1,4.2 的模型继续进行调度。当调度过程进行至上下料完成时刻，我们会对 CNC 进行一次故障判断，如果判断为真，则进入维修操作，如果判断为假，则重新进行利用贪心算法确定优解的步骤。算法流程图如下图所示：

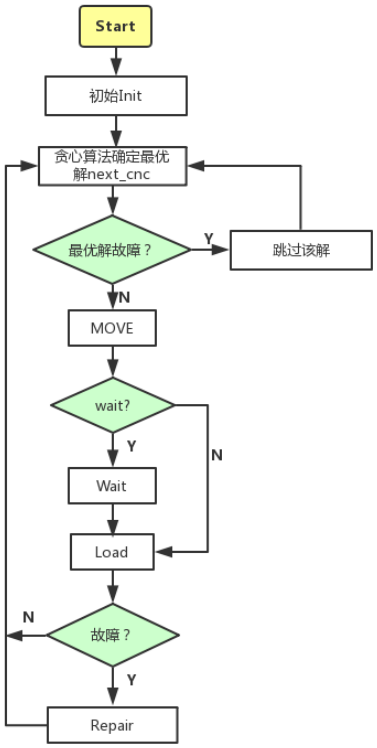


图 7

对于这种模型，我们进行了 C++的代码实现，得到了 RGV 的调度策略和系统的作业效率。针对结果，我们发现故障的发生基本服从泊松过程，且故障发生之后调

度策略会发生极大的变化，而产量也会下降，与我们的预期相同。

4.3.3 基于固定故障序号的静态改进模型

在这个模型中，我们选择在模型运行前先选择将会发生故障的序列号。具体操作过程如下：
首先我们取一个数组 a[i]（1≤i≤o)，因为发生故障下的熟料总产量小于无故障状态下的总产量，所以取 O 为无故障情况下系统熟料总产量，如第一组数据第一种情况下该数组为 a[i](1≤i≤383）。然后我们对数组中的每一个元素进行故障判断，具体步骤如下生成一个从 0 到 100 的随机数 a，如果生成的随机数 a<1，即该序号对应的加工物料会发生故障。将会发生的故障的序号从小到大排列我们可以得到一个故障序号向量如 P=（25，31，47，56，120，160，197）
当加工物料序号小于 25 时，我们的模型运行完全按照原有模型运行，当故障发生的时刻，我们将当时发生的状态代入故障发生时特有的模型进行计算，故障完全排除以后，将故障排除末时刻各个 CNC 以及 RGV 的状态记录下来，代入原有模型。加工物料继续开始计数，直到计数到达故障序号向量下一个值的时刻，重复上一次发生故障所进行的操作，循环进行这些步骤，直到时间耗尽，从而得到系统的调度过程。这种模型的优势在于，可以依照该模型，制定不同故障发生情况下的应急调度。但是时间有限，在这种想法上，我们没有付诸于程序实践。

5.模型评价

5.1 模型的优点

（1）在数据处理方面，我们的模型通过 C++宏定义常量，当宏定义常量如小车移动的距离发生改变时，只需更改宏定义常量的赋值即可正常运行，提高了程序的适用性，并且拟合出的结果合理有效。

（2）模型的抗干扰能力比较强。我们的局部最优模型采用贪心算法使得 RGV 在任何情况下都有相对应的判断流程和对应行动。如果采取固定路线，在 CNC 平台下出现故障的情况下，RGV 可能会出现等待，破坏原来的循环算法，导致后续一系列的拖延、滞后行为，降低系统运行效率。

（3）模型部分输出图像为甘特图，其通过条状图来显示项目，进度和其他时间相关的系统进展的内在关系随着时间进展的情况，直观表明计划何时进行，进展与要求的对比。便于管理者弄清项目的剩余人物，评估工作进度。

（4）模型简单易懂，模型原则方便理解，抛弃无用复杂的模型公式，实现“简单的模型解决复杂的问题”。

（5）模型全部采用 C++类封装，程序比 matlab 等其他语言实现的安全性和稳定性更高，可移植性更强。

5.2 模型的缺点

（1）采取的算法结构使以局部最优为基础原则，但是从整体来看并不能判断是否为最优路径和模型，导致拟合的结果也不能判断是否最优，不排除更优解的存在

（2）算法在第二问中对刀具的最佳位置分布选取经验化，即单纯地认为某种分布式排列最优，实际上不知道也无法证明最优排列的存在。

（3）没有实现文件流代码，不能将生成数据直接读入 Excel 表，但可以通过 Excel 排序得到附件格式的数据排列。

5.3 模型的展望

由于时间受限，模型的具体功能也受到了限制，如果能够给予更多时间，模型将能够实现以下更多的功能。

（1）不单单只适用于直线型往返路线，如果面对环形路线，贪心算法在原理上能够完成局部最优解。

（2）如果增加 RGV 的数目，这样显然增加了效率，同样地，贪心算法依旧适用于此类条件，但是要注意，在评价时间消耗条件中要增加小车堵塞的时间计算。

6.参考文献

【1】吴焱明, 刘永强,张栋，赵韩.基于遗传算法的 RGV 动态调度研究.合肥:合肥工业大学出版社，2012，89-93

【2】李凯. 含作业到达时间的单机调度问题的改进算法.合肥：合肥工业大学出版社,2008，20-23

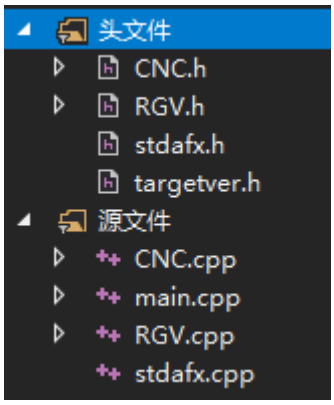
【3】曹阳明，周亚勤，杨建国，刘凯强. 考虑刀具约束的作业车间调度研究.上海：东华大学机械工程学院，2017，103-106

【4】甘婕，考虑性能可靠度约束的单机调度与视情维修的集成 优化模型.太原：太原科技大学工业与系统工程研究所，2017，123-134

附录

运行平台：VS2017

本程序全部采用 C++编写，正确运行程序请在 VS 平台新建一个 Windows 控制台应用程序，附录代码用“/**/”分割，存储成不同类型文件。



①第一种情况（无故障）：

```
/*----------------------------------------stdafx.h----------------------------------------*/
#pragma once
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <iostream>
#include <Windows.h>
#include <ctime>
#include <stdlib.h>
/*----------------------------------------targetver.h----------------------------------------*/
#pragma once
#include <SDKDDKVer.h>
/*----------------------------------------CNC.h----------------------------------------*/
#pragma once
class CNC
{
private:
	int number;	//CNC 编号
	int position; //CNC 位置
	int count;	//CNC 剩余工作时间
	int n;		//加工物序列号
public:
	CNC(int num, int pos);
	void countdown(int tem);
	friend class RGV;
};
/*----------------------------------------CNC.cpp----------------------------------------*/
#include "stdafx.h"
#include "CNC.h"
```

```
using namespace std;
CNC::CNC( int num, int pos)
{
	number = (num > 0) ? num : ERROR;
	position = (pos >= 0) ? pos : ERROR;
	count = 0;
	n = 0;
}
void CNC::countdown(int temp)
{
	if (count > temp) {
		count -= temp;
	}
	else {
		count = 0;}
}
/*-------------------------------------RVG.h-------------------------------------*/
#pragma once
#include "CNC.h"
class RGV
{
private:
	int position; //rgv 当前位置
	int now_cnc;	//当前目标
	int rgv_flag;	//rgv_flag,1 有熟料,0 无

public:
	int time; //总用时
	int sum; //加工熟料总数
	RGV();
	void Init(CNC *p); //第一轮初始化
	int posCalculate(int pos1, int pos2); //计算 RGV 移动时间
	void move(CNC *p); //RGV 移动
	void load(CNC *p); //RGV 上下料
	void clean(CNC *p);//RGV 清洗
	void wait(CNC *p);//RGV 等待
};
/*-------------------------------------RVG.cpp-----------------------------------------*/
#include "stdafx.h"
#include "RGV.h"
#define STEP1 20
#define STEP2 33
#define STEP3 46
#define CNC_WORKTIME 560
```

```
#define CNC1 28
#define CNC0 31
#define CLEAN 25
#define CNCNUMBER 8
using namespace std;
int n = 0;        //加工物序列号
RGV::RGV()
{
    position = 0;
    now_cnc = 1;
    rgv_flag = 0;
    time = 0;
    sum = 0;
}
void RGV::Init(CNC *p)
{
    CNC* ptr = p;
    for (int i = 0; i < CNCNUMBER; i++, now_cnc++) {
        int temp = posCalculate(position, (p + now_cnc - 1)->position);
        if (temp) {
            time += temp;
            CNC*ptr1 = p;
            for (int i = 0; i < CNCNUMBER; i++, ptr1++) {
                ptr1->countdown(temp);
            }
        }
        load(p);
        position = (p + now_cnc - 1)->position;
    }
}
int RGV::posCalculate(int pos1, int pos2)//RGV 移动时间计算
{
    switch (abs(pos1 - pos2))
    {
    case 3: return STEP3;
    case 2: return STEP2;
    case 1: return STEP1;
    case 0: return 0;
    default: return ERROR;
    }
}

void RGV::move(CNC *p)//RGV 移动
{
```

```
    CNC* ptr = p;
    int time1 = 10000;//最少时间
    int time2 = 0;       //当前时间
    int next_cnc = 1; //最优解
    for (int i = 0; i < CNCNUMBER; i++, ptr++)
    {
        int postime = posCalculate(position, ptr->position);
        time2 = ptr->count + postime;//CNC 工作剩余时间 + RGV 移动到当前位置
时间
        if (time1 > time2) {
            time1 = time2;
            next_cnc = ptr->number;
        }
        else continue;
    }

    if (now_cnc != next_cnc) {//当前对象不是最优对象，移动
        int temp = posCalculate(position, (p + next_cnc - 1)->position);
        if (temp) {
            time += temp;
            CNC*ptr1 = p;
            for (int i = 0; i < CNCNUMBER; i++, ptr1++) //所有 CNC 剩余时间 - rgv
移动时间
            {
                ptr1->countdown(temp);
            }
        }
        position = (p + next_cnc - 1)->position;
        now_cnc = next_cnc;
    }
    //cout << "position" << position << '\t' << "now_cnc" << now_cnc << endl;
//test
    wait(p);
}
void RGV::load(CNC *p)//RGV 上下料
{
    CNC* ptr = p + now_cnc - 1; //此处数组下标和 CNC 编号要注意！
    int temp = 0;

    cout << time << '\t' << '\t' << ptr->n << '\t' << '\t' << ++n << '\t' << now_cnc <<
endl;
    if ((ptr->number % 2) == 1) {
        temp = CNC1;
    }
```

```
    else {
        temp = CNC0;
    }
    time += temp;
    CNC*ptr1 = p;
    for (int i = 0; i < CNCNUMBER; i++, ptr1++) //所有 CNC 剩余时间 - rgv 移动时间
    {
        ptr1->countdown(temp);
    }
    ptr->n = n;
    ptr->count = CNC_WORKTIME;
    rgv_flag = 1;
}
void RGV::clean(CNC* p)//RGV 清洗
{
    sum++;
    int temp = CLEAN;
    time += temp;
    CNC*ptr1 = p;
    for (int i = 0; i < CNCNUMBER; i++, ptr1++) //所有 CNC 剩余时间 - rgv 移动时间
    {
        ptr1->countdown(temp);
    }
}
void RGV::wait(CNC* p)
{
    CNC* ptr = p + now_cnc - 1;
    if (ptr->count) {
        int temp = ptr->count;
        time += temp;
        CNC*ptr1 = p;
        for (int i = 0; i < CNCNUMBER; i++, ptr1++) //所有 CNC 剩余时间 - rgv 移动
时间
        {
            ptr1->countdown(temp);
        }
        ptr->count = 0;
    }
    else return;
}
/*-------------------------main.cpp---------------------------------*/
#include "stdafx.h"
#include "RGV.h"
#include "CNC.h"
```

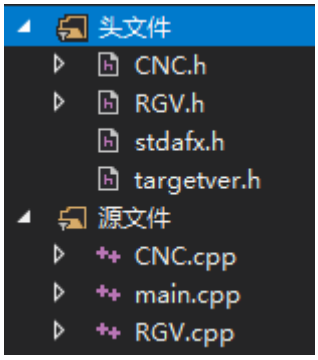
```
#define CNCNUMBER 8
#define TIME 28800
using namespace std;
int main()
{
	RGV rgv;
	CNC cnc[CNCNUMBER] = {
		CNC(1,0), CNC(2,0), CNC(3,1), CNC(4,1),
		CNC(5,2), CNC(6,2), CNC(7,3), CNC(8,3) };
	CNC *pCNC = cnc;

	cout <<"时间" << '\t' << "下料开始" << '\t' << "上料开始" << '\t' << "CNC#" <<
endl;

	/*第一轮
	仅需考虑 RGV“移动”和“上下料”动作*/
	rgv.Init(pCNC);

	/*第(n+1)轮
	RGV 循环“移动”，“上下料”和“清洗”动作*/
	while (rgv.time <= TIME)
	{
		rgv.move(pCNC);
		rgv.load(pCNC);
		rgv.clean(pCNC);
	}
	cout << "生成熟料总数：" << rgv.sum << endl;
	system("pause");
	return 0;
}
```

②第二种情况（无故障）：




```
/*----------------------------------------CNC.h----------------------------------------*/
#pragma once
```

```
class CNC
{
private:
	int number;	//CNC 编号
	int position; //CNC 位置
	int knife;	//CNC 刀具型号,1 第一道工序，2 第二道工序
	int count;	//CNC 剩余工作时间
	int n;		//加工物序列号
public:
	CNC(int num, int pos, int kni);
	void countdown(int temp);
	friend class RGV;
};
/*----------------------------------------CNC.cpp----------------------------------------*/
#include "stdafx.h"
#include "CNC.h"
CNC::CNC(int num, int pos, int kni)
{
	number = (num > 0) ? num : ERROR;
	position = (pos >= 0) ? pos : ERROR;
	knife = (kni == 1 || kni == 2) ? kni : ERROR;
	count = 0;
	n = 0;
}
void CNC::countdown(int temp)
{
	if (count > temp) {
		count -= temp;
	}
	else {
		count = 0;
	}
}
/*----------------------------------------RGV.h----------------------------------------*/
#pragma once
#include "CNC.h"
using namespace std;
class RGV
{
private:
	int position; //rgv 当前位置
	int now_cnc;//rgv 当前 CNC 对象
	int rgv_flag; //rgv_flag,RGV 当前状态，1 空闲，2 持有半成品
	int rgv_n;		//RGV 当前半成品号
```

```
public:
	int time; //总用时
	int sum; //加工熟料总数
	RGV();
	int find(CNC* p); //寻找最优解
	void Init(CNC *p); //第一轮初始化
	int posCalculate(int pos1, int pos2); //计算 RGV 移动时间
	void move(CNC *p); //RGV 移动
	void load(CNC *p); //RGV 上下料
	void wait(CNC *p);//RGV 等待
};
/*----------------------------------------RGV.cpp----------------------------------------*/
#include "stdafx.h"
#include "RGV.h"
#define STEP1 20
#define STEP2 33
#define STEP3 46
#define CNC_WORKTIME1 400
#define CNC_WORKTIME2 378
#define CNC1 28
#define CNC0 31
#define CLEAN 25
#define CNC_NUMBER 8
#define TIME 28800
using namespace std;
int n = 0;//加工物序号
RGV::RGV()
{
	position = 0;
	now_cnc = 1;
	rgv_flag = 1;
	rgv_n = 0;
	time = 0;
	sum = 0;
}
void RGV::Init(CNC *p)
{
	CNC*ptr = p;
	for (int i = 0; i < CNC_NUMBER; i++, ptr++, now_cnc++)
	{
		if (ptr->knife == 1) {
			int temp = posCalculate(position, (p + now_cnc - 1)->position);
			if (temp) {
				time += temp;
```

```
                CNC*ptr1 = p;
                for (int i = 0; i < CNC_NUMBER; i++, ptr1++) //同步 RGV 和 CNC 的时
间
                {
                    ptr1->countdown(temp);
                }
            }
            load(p);
            position = (p + now_cnc - 1)->position;
            //cout << "position" << position << '\t' << "now_cnc" << now_cnc <<
endl;   //test
            rgv_flag = 1;
        }
        else continue;
    }
}
int RGV::posCalculate(int pos1, int pos2)//RGV 移动时间计算
{
    switch (abs(pos1 - pos2))
    {
    case 3: return STEP3;
    case 2: return STEP2;
    case 1: return STEP1;
    case 0: return 0;
    default: return ERROR;
    }
}
int RGV::find(CNC *p)
{
    CNC* ptr = p;
    int next_cnc = 1;//最优解
    int min_time1 = 1000;
    int now_time1 = 0;
    if (rgv_flag == 1) {//空闲
        for (int i = 0; i < CNC_NUMBER; i++, ptr++) {//注意！！空闲既可选择刀具 1，
也可选择刀具 2
            int postime = posCalculate(position, ptr->position);
            now_time1 = (ptr->count > postime) ? ptr->count : postime;
            if (min_time1 > now_time1) {
                min_time1 = now_time1;
                next_cnc = ptr->number;
            }
            else continue;
        }
```

```
    }
    else {//半成品
        for (int i = 0; i < CNC_NUMBER; i++, ptr++) {
            if (ptr->knife == 2) {
                int postime = posCalculate(position, ptr->position);
                now_time1 = (ptr->count > postime) ? ptr->count : postime;
                if (min_time1 > now_time1) {
                    min_time1 = now_time1;
                    next_cnc = ptr->number;
                }
            }
            else continue;
        }
    }
    return next_cnc;
}
void RGV::move(CNC *p)//RGV 移动
{
    int next_cnc = find(p);
    if (next_cnc != now_cnc) {//当前对象不是最优对象，移动
        int temp = posCalculate(position, (p + next_cnc - 1)->position);
        if (temp) {
            time += temp;
            CNC*ptr1 = p;
            for (int i = 0; i < CNC_NUMBER; i++, ptr1++) //所有 CNC 剩余时间 - rgv
移动时间
            {
                ptr1->countdown(temp);
            }
        }
        position = (p + next_cnc - 1)->position;
        now_cnc = next_cnc;
        //cout << "position" << position << '\t' << "now_cnc" << now_cnc << endl;
//test
    }
    wait(p);
}
void RGV::load(CNC *p)
{
    CNC* ptr = p + now_cnc - 1; //此处数组下标和 CNC 编号要注意！

    int count_temp = 0;
    if (ptr->knife == 1) {
        cout << time << '\t' << rgv_flag << '\t' << ptr->n << '\t' << '\t' << ++n << '\t' <<
```

```
'\t' << now_cnc << endl;
            count_temp = CNC_WORKTIME1;
            rgv_flag = 2;
            rgv_n = ptr->n;//半成品序列号
            ptr->n = n;
        }
        else {
            cout << time << '\t' << rgv_flag << '\t' << ptr->n << '\t' << '\t' << rgv_n << '\t'
<< '\t' << now_cnc << endl;
            count_temp = CNC_WORKTIME2 + CLEAN;
            rgv_flag = 1;
            if ((TIME - time) > CNC0 && (TIME - time) > CNC1) {
                  sum = ptr->n;
            }else
                 sum = (ptr->n - 1);
            ptr->n = rgv_n;
        }
        ptr->count = count_temp;
        int temp = 0;
        if ((ptr->number % 2) == 1) {
            temp = CNC1;
        }
        else {
            temp = CNC0;
        }
        time += temp;
        CNC*ptr1 = p;
        for (int i = 0; i < CNC_NUMBER; i++, ptr1++) //所有 CNC 剩余时间  - rgv 移动时
间
        {
            ptr1->countdown(temp);
        }
}
void RGV::wait(CNC *p)//RGV 等待
{
        CNC* ptr = p + now_cnc - 1;
        if (ptr->count) {
            int temp = ptr->count;
            time += temp;
            CNC*ptr1 = p;
            for (int i = 0; i < CNC_NUMBER; i++, ptr1++) //所有 CNC 剩余时间  - rgv 移动
时间
            {
                ptr1->countdown(temp);
```

```
            }
        }
        else return;
}
/*----------------------------------------main.cpp----------------------------------------*/
#include "stdafx.h"
#include "RGV.h"
#include "CNC.h"
#define CNC_NUMBER 8
#define TIME 28800
using namespace std;
int main()
{
        RGV rgv;
        CNC cnc[CNC_NUMBER] = {
            CNC(1,0,1), CNC(2,0,2), CNC(3,1,1), CNC(4,1,2),
            CNC(5,2,1), CNC(6,2,2), CNC(7,3,1), CNC(8,3,2) };
        CNC *pCNC = cnc;

        cout << "时间" << '\t' <<"工序"<< '\t' << "下料开始" << '\t' << "上料开始" << '\t'
<< "CNC#" << endl;

        rgv.Init(pCNC);

        while (rgv.time <= TIME)
        {
            rgv.move(pCNC);
            rgv.load(pCNC);
        }
        cout << "生成熟料总数：" << rgv.sum << endl;
        system("pause");
         return 0;
}
```

③第三种情况——一道工序（有故障）：

第三情况的代码建立在之前程序的基础上，有增无减，重复部分不再体现，只显示改动部分。

```
/*----------------------------------------CNC.h----------------------------------------*/
class CNC
{
private:
        int flag;          //CNC 当前状态，1 故障，0 正常
};
/*----------------------------------------CNC.cpp----------------------------------------*/
CNC::CNC( int num, int pos)
```

```
{
	flag = 0;
}
void CNC::countdown(int temp)
{
	if (count > temp) {
		count -= temp;
	}
	else {
		count = 0;
		if (flag) {
			flag = 0; //恢复正常
			cout << "CNC#" << number << "恢复正常" << endl;
		}
	}
}
/*----------------------------------------RGV.h----------------------------------------*/
class RGV
{
public:
	int t_time; //总用时
	int scr_times; //报废次数

	void repair(CNC *p);//RGV 故障修复
}
/*----------------------------------------RGV.cpp----------------------------------------*/
RGV::RGV()
{
	t_time = 0;
	scr_times = 0;
}
void RGV::Init(CNC *p)
{
	CNC* ptr = p;
	for (int i = 0; i < CNCNUMBER; i++, now_cnc++)		{
		int temp = posCalculate(position, (p + now_cnc - 1)->position);
		if (temp) {
			t_time += temp;
			CNC*ptr1 = p;
			for (int i = 0; i < CNCNUMBER; i++, ptr1++)			{
				ptr1->countdown(temp);
			}
		}
		load(p);
```

```
            position = (p + now_cnc - 1)->position;
        }
}
void RGV::move(CNC *p)//RGV 移动
{
        CNC* ptr = p;
        int time1 = 10000;//最少时间
        int time2 = 0;        //当前时间
        int next_cnc = 1;
        for (int i = 0; i < CNCNUMBER; i++, ptr++)
        {
            if (!(ptr->flag)) {
                int postime = posCalculate(position, ptr->position);
                time2 = ptr->count + postime;//CNC 工作剩余时间  + RGV 移动到当前
位置时间
                if (time1 > time2) {
                    time1 = time2;
                    next_cnc = ptr->number;
                }
            }
            else continue;
        }
}
void RGV::repair(CNC *p)
{
        CNC*ptr = p + now_cnc - 1;
        double rand_num    = rand() % 100;          //生成随机数
        if (rand_num < 1.0) { //故障概率 0.01
            scr_times++;
            cout << t_time << '\t' << n << "号物料报废" << '\t' << "CNC#" <<
ptr->number << "发生故障" << endl;
            ptr->flag = 1;//故障
            int repair_time = rand() % 600 + 600;
            ptr->count = repair_time;
        }
        else return;
}
/*---------------------------------------main.cpp-------------------------------------*/
int main()
{
        CNC cnc[CNCNUMBER] = {
            CNC(1,0), CNC(2,0), CNC(3,1), CNC(4,1),
            CNC(5,2), CNC(6,2), CNC(7,3), CNC(8,3) };
        CNC *pCNC = cnc;
```

```
    cout <<"时间" << '\t' << "下料开始" << '\t' << "上料开始" << '\t' << "CNC#" <<
endl;
    while (rgv.t_time <= TIME)
    {
        rgv.move(pCNC);
        rgv.load(pCNC);
        rgv.repair(pCNC);
        rgv.clean(pCNC);
    }
    cout << "生成熟料总数：" << rgv.sum-rgv.scr_times << endl;
    cout << "发生故障次数" << rgv.scr_times << endl;
}
```

④第三种情况——两道工序（有故障）：

```
/*----------------------------------------CNC.h----------------------------------------*/
RGV::RGV()
{
    rgv_flag = 1;
    rgv_n = 0;
    t_time = 0;
    scr_times = 0;
}
void RGV::Init(CNC *p)
{
    CNC*ptr = p;
    for (int i = 0; i < CNC_NUMBER; i++, ptr++, now_cnc++)
    {
        if (ptr->knife == 1) {
            int temp = posCalculate(position, (p + now_cnc - 1)->position);
            if (temp) {
                t_time += temp;
                CNC*ptr1 = p;
                for (int i = 0; i < CNC_NUMBER; i++, ptr1++) //同步 RGV 和 CNC 的时
间
                {
                    ptr1->countdown(temp);
                }
            }
            load(p);
            position = (p + now_cnc - 1)->position;
            //cout << "position" << position << '\t' << "now_cnc" << now_cnc <<
endl;   //test
            rgv_flag = 1;
        }
        else continue;}
```

```
}
int RGV::find(CNC *p)
{
    CNC* ptr = p;
    int next_cnc = 1;//最优解
    int min_time1 = 1000;
    int now_time1 = 0;
    if (rgv_flag == 1) {//空闲
        for (int i = 0; i < CNC_NUMBER; i++, ptr++) {//注意！！空闲既可选择刀具 1，
也可选择刀具 2
            if (!(ptr->flag)) {
                int postime = posCalculate(position, ptr->position);
                now_time1 = (ptr->count > postime) ? ptr->count : postime;
                if (min_time1 > now_time1) {
                    min_time1 = now_time1;
                    next_cnc = ptr->number; }
            }
            else continue;
        }
    }
    else {//半成品
        for (int i = 0; i < CNC_NUMBER; i++, ptr++) {
            if (!(ptr->flag)) {
                if (ptr->knife == 2) {
                    int postime = posCalculate(position, ptr->position);
                    now_time1 = (ptr->count > postime) ? ptr->count : postime;
                    if (min_time1 > now_time1) {
                        min_time1 = now_time1;
                        next_cnc = ptr->number;
                    }}}
            else continue;
        }
    }
    return next_cnc;
}
void RGV::move(CNC *p)//RGV 移动
{
    int next_cnc = find(p);
    if (next_cnc != now_cnc) {//当前对象不是最优对象，移动
        int temp = posCalculate(position, (p + next_cnc - 1)->position);
        if (temp) {
            t_time += temp;
            CNC*ptr1 = p;
            for (int i = 0; i < CNC_NUMBER; i++, ptr1++) //所有 CNC 剩余时间 - rgv
```

```
移动时间
                {
                    ptr1->countdown(temp);
                }
            }
}
void RGV::load(CNC *p)
{
    CNC* ptr = p + now_cnc - 1; //此处数组下标和 CNC 编号要注意！

    int count_temp = 0;
    if (ptr->knife == 1) {
        cout << t_time << '\t' << rgv_flag << '\t' << ptr->n << '\t' << '\t' << ++n << '\t'
<< '\t' << now_cnc << endl;
        count_temp = CNC_WORKTIME1;
        rgv_flag = 2;
        rgv_n = ptr->n;//半成品序列号
        ptr->n = n;
    }
    else {
        cout << t_time << '\t' << rgv_flag << '\t' << ptr->n << '\t' << '\t' << rgv_n <<
'\t' << '\t' << now_cnc << endl;
        count_temp = CNC_WORKTIME2 + CLEAN;
        rgv_flag = 1;
        if ((TIME - t_time) > CNC0 && (TIME - t_time) > CNC1) {
            sum = ptr->n;
        }else
            sum = (ptr->n - 1);
        ptr->n = rgv_n;
    }
    ptr->count = count_temp;

    int temp = 0;
    if ((ptr->number % 2) == 1) {
        temp = CNC1;
    }
    else {
        temp = CNC0;
    }
    t_time += temp;
    CNC*ptr1 = p;
    for (int i = 0; i < CNC_NUMBER; i++, ptr1++) //所有 CNC 剩余时间 - rgv 移动时
间
    {
```

```
            ptr1->countdown(temp);
        }
}
void RGV::repair(CNC *p)
{
        CNC*ptr = p + now_cnc - 1;
        double rand_num = rand() % 100;          //生成随机数
        if (rand_num < 1.0) { //故障概率 0.01
            scr_times++;
            cout << t_time << '\t' << n << "号物料报废" << '\t' << "CNC#" <<
ptr->number << "发生故障" << endl;
            ptr->flag = 1;//故障
            int repair_time = rand() % 600 + 600;
            ptr->count = repair_time;
        }
        else return;
}

/*---------------------------------------CNC.cpp---------------------------------------*/
CNC::CNC(int num, int pos, int kni)
{
        number = (num > 0) ? num : ERROR;
        position = (pos >= 0) ? pos : ERROR;
        knife = (kni == 1 || kni == 2) ? kni : ERROR;
        count = 0;
        n = 0;
        flag = 0;
}
void CNC::countdown(int temp)
{
        if (count > temp) {
            count -= temp;
        }
        else {
            count = 0;
            if (flag) {
                flag = 0; //恢复正常
                cout << "CNC#" << number << "恢复正常" << endl;
            }
        }
}
/*---------------------------------------RGV.h---------------------------------------*/
class RGV
{
```

```
private:
    int rgv_flag;  //rgv_flag,RGV 当前状态，1 空闲，2 持有半成品
    int rgv_n;          //RGV 当前半成品号
public:
    int t_time; //总用时
    int scr_times; //报废次数

    void repair(CNC *p);//RGV 故障修复
};
/*---------------------------------------RGV.cpp---------------------------------------*/
RGV::RGV()
{
    rgv_flag = 1;
    rgv_n = 0;
    t_time = 0;
    scr_times = 0;
}
void RGV::Init(CNC *p)
{
    CNC*ptr = p;
    for (int i = 0; i < CNC_NUMBER; i++, ptr++, now_cnc++)
    {
        if (ptr->knife == 1) {
            int temp = posCalculate(position, (p + now_cnc - 1)->position);
            if (temp) {
                t_time += temp;
                CNC*ptr1 = p;
                for (int i = 0; i < CNC_NUMBER; i++, ptr1++) //同步 RGV 和 CNC 的时
间
                {
                    ptr1->countdown(temp);
                }
            }
            load(p);
            position = (p + now_cnc - 1)->position; rgv_flag = 1;
        }
        else continue;
    }
}
int RGV::find(CNC *p)
{
    CNC* ptr = p;
    int next_cnc = 1;//最优解
    int min_time1 = 1000;
```

```
    int now_time1 = 0;
    if (rgv_flag == 1) {//空闲
        for (int i = 0; i < CNC_NUMBER; i++, ptr++) {//注意！！空闲既可选择刀具 1，
也可选择刀具 2
            if (!(ptr->flag)) {
                int postime = posCalculate(position, ptr->position);
                now_time1 = (ptr->count > postime) ? ptr->count : postime;
                if (min_time1 > now_time1) {
                    min_time1 = now_time1;
                    next_cnc = ptr->number;
                }
            }
            else continue;
        }
    }
    else {//半成品
        for (int i = 0; i < CNC_NUMBER; i++, ptr++) {
            if (!(ptr->flag)) {
                if (ptr->knife == 2) {
                    int postime = posCalculate(position, ptr->position);
                    now_time1 = (ptr->count > postime) ? ptr->count : postime;
                    if (min_time1 > now_time1) {
                        min_time1 = now_time1;
                        next_cnc = ptr->number;
                    }
                }
            }
            else continue;
        }
    }

    //cout << "最优解" << next_cnc << endl;//test
    return next_cnc;
}

void RGV::move(CNC *p)//RGV 移动
{
    int next_cnc = find(p);
    if (next_cnc != now_cnc) {//当前对象不是最优对象，移动
        int temp = posCalculate(position, (p + next_cnc - 1)->position);
        if (temp) {
            t_time += temp;
            CNC*ptr1 = p;
            for (int i = 0; i < CNC_NUMBER; i++, ptr1++) //所有 CNC 剩余时间 - rgv
```

```
移动时间
                {
                    ptr1->countdown(temp);
                }
            }
            position = (p + next_cnc - 1)->position;
            now_cnc = next_cnc;

            //cout << "position" << position << '\t' << "now_cnc" << now_cnc << endl;
//test
        }
        wait(p);
}

void RGV::load(CNC *p)
{
        CNC* ptr = p + now_cnc - 1; //此处数组下标和 CNC 编号要注意！

        int count_temp = 0;
        if (ptr->knife == 1) {
            cout << t_time << '\t' << rgv_flag << '\t' << ptr->n << '\t' << '\t' << ++n << '\t'
<< '\t' << now_cnc << endl;
            count_temp = CNC_WORKTIME1;
            rgv_flag = 2;
            rgv_n = ptr->n;//半成品序列号
            ptr->n = n;
        }
        else {
            cout << t_time << '\t' << rgv_flag << '\t' << ptr->n << '\t' << '\t' << rgv_n <<
'\t' << '\t' << now_cnc << endl;
            count_temp = CNC_WORKTIME2 + CLEAN;
            rgv_flag = 1;
            if ((TIME - t_time) > CNC0 && (TIME - t_time) > CNC1) {
                  sum = ptr->n;
            }else
                sum = (ptr->n - 1);
            ptr->n = rgv_n;
        }
        ptr->count = count_temp;

        int temp = 0;
        if ((ptr->number % 2) == 1) {
            temp = CNC1;
        }
```

```
    else {
        temp = CNC0;
    }
    t_time += temp;
    CNC*ptr1 = p;
    for (int i = 0; i < CNC_NUMBER; i++, ptr1++) //所有 CNC 剩余时间 - rgv 移动时
间
    {
        ptr1->countdown(temp);
    }
}

void RGV::wait(CNC *p)//RGV 等待
{
    CNC* ptr = p + now_cnc - 1;
    if (ptr->count) {
        int temp = ptr->count;
        t_time += temp;
        CNC*ptr1 = p;
        for (int i = 0; i < CNC_NUMBER; i++, ptr1++) //所有 CNC 剩余时间 - rgv 移动
时间
        {
            ptr1->countdown(temp);
        }
    }
    else return;
}

void RGV::repair(CNC *p)
{
    CNC*ptr = p + now_cnc - 1;
    double rand_num = rand() % 100;          //生成随机数
    if (rand_num < 1.0) { //故障概率 0.01
        scr_times++;
        cout << t_time << '\t' << n << "号物料报废" << '\t' << "CNC#" <<
ptr->number << "发生故障" << endl;
        ptr->flag = 1;//故障
        int repair_time = rand() % 600 + 600;
        ptr->count = repair_time;
    }
    else return;
}
/*----------------------------------------main.cpp----------------------------------------*/
int main()
```

```
{
    RGV rgv;
    CNC cnc[CNC_NUMBER] = {
        CNC(1,0,1), CNC(2,0,2), CNC(3,1,1), CNC(4,1,2),
        CNC(5,2,1), CNC(6,2,1), CNC(7,3,1), CNC(8,3,1) };
    CNC *pCNC = cnc;

    cout << "时间" << '\t' <<"工序"<< '\t' << "下料开始" << '\t' << "上料开始" << '\t'
<< "CNC#" << endl;
    while (rgv.t_time <= TIME)
    {
        rgv.move(pCNC);
        rgv.load(pCNC);
        rgv.repair(pCNC);
    }
    cout << "物料消耗总数：" << rgv.sum << endl;
    cout << "发生故障次数：" << rgv.scr_times << endl;
}
```