基于局部搜索的动态调度策略

摘要

作业车间调度问题是现代工业生产系统的关键之一，与传统的车间调度问题相比，本题引入了无人驾驶的智能工作车 RGV 进行上料、下料、清洗操作。机械臂的状态变化与 RGV 小车的作业次序对整个加工系统有着重大的影响，为了提高智能加工系统的工作效率，本文结合传统的车间调度模型，对加工系统中 RGV 小车的动态调度问题进行了研究，并建立了相关模型。

针对任务一，我们首先考虑一道工序的物料加工情况，使用优先规则指派算法，在 SPT（加工时间最短的工序优先）的基础上，引入启发式搜索算法加以改进，不仅考虑当前加工时间最短，还考虑了之后作业的加工时间，以局部过程内的加工时间为代价函数，得到一个精确度相对较高的近似解；为了得到效率更高的可行解，我们使用模拟退火算法进行优化，将上一步中的近似解作为模拟退火算法的初始解，使得求解算法的精度进一步提高。对于两道工序的 RGV 调度问题，每个工件的加工应该按工序顺序执行，且每次执行完第一道工序后，必须立即将机械臂上的中间产品送往下一道工序的加工位置，为此，我们引入工件约束条件，使得修正后的模型能够有效描述两道工序的调度问题；对于故障问题，我们假设故障产生的概率符合二项分布，故障持续的时间符合正态分布，在每次 CNC 加工结束之后进行故障检验，同时在模型中引入故障约束条件，使得故障问题得以在模型中体现。

针对任务二，我们抽取附件表 1 中的三组参数，分别输入已经建立完成的多个 Matlab 仿真调度模型，得出一般情况下单工序调度器与双工序调度器模型，以及故障模型参与下两个调度模型的输出产品数，以及每个生料的上料时间与下料时间，作为最终模型表现得分的评判标准，并完善附件中的表格，并加以结果的分析与反思。

最后，我们对模型进行了检验和中肯的评价。

关键词　优先规则指派算法，局域搜索法，启发式搜索，模拟退火法

一、 问题重述

1.1 问题的背景

随着“工业 4.0”与“中国制造 2025”等概念的提出，制造业对自动化、智能化生产模式及现代化、智能化的物流运输和仓库系统的需求日益增长。RGV 小车作为现代化，智能化的物流运输和仓储系统的关键设备之一，应用于越来越多的生产制造和物流项目中。在题目所给的智能加工系统中，RGV 小车的效率是难点。为了提高 RGV 小车的工作效率，需要对系统中 RGV 小车的动态调度问题进行研究，针对不同的加工情况，建立相关的模型，制定对应的调度策略，提高系统的作业效率，完善整个智能加工系统。

1.2 问题的相关信息

1.2.1 附件 1 给出了智能加工系统的组成与作业流程：

（1）智能加工系统的构成：

智能加工系统由 8 台 CNC、1 台带机械手和清洗槽的 RGV、1 条 RGV 直线轨道、1 条上料传送带和 1 条下料传送带等附属设备构成。

（2）智能加工系统的说明：

① 现有八台 CNC 可以对工件进行加工，CNC 状态包括：闲置、工作、等待、损坏。

② 当 CNC 处于闲置状态时，向 RGV 发出工作请求，由 RGV 确定其作业序列，为其完成上料，下料，清洗等操作；

③ 当 CNC 处于工作状态时，RGV 可以执行其他 CNC 的工作请求；

④ 当 CNC 处于等待状态，表明 RGV 没能即刻赶来处理请求；

⑤ 当 CNC 处于损坏状态，无法进行工作，需要进行修复。

（3）智能加工系统的作业流程：

图 1 给出了智能加工系统的作业流程图。

1.2.2 任务 2 提供了智能加工系统作业参数的 3 组数据表。

智能加工系统作业参数的 3 组数据表如表 1 所示。

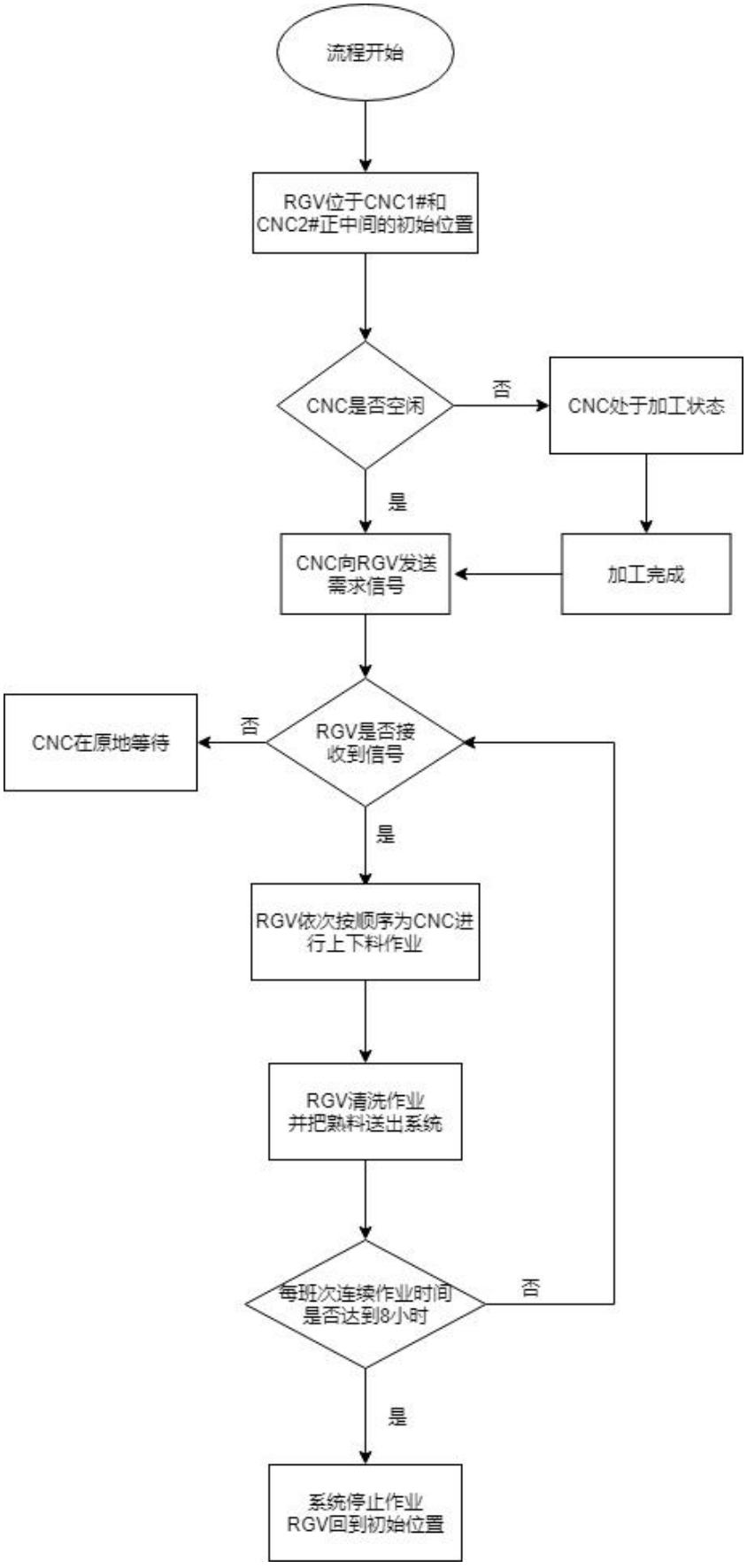

流程开始
RGV位于CNC1#和CNC2#正中间的初始位置
CNC是否空闲
否
CNC处于加工状态
是
CNC向RGV发送需求信号
加工完成
CNC在原地等待
否
RGV是否接收到信号
是
RGV依次按顺序为CNC进行上下料作业
RGV清洗作业并把熟料送出系统
每班次连续作业时间是否达到8小时
否
是
系统停止作业RGV回到初始位置

图 1

时间单位：秒

系统作业参数	第 1 组	第 2 组	第 3 组
RGV 移动 1 个单位所需时间	20	23	18
RGV 移动 2 个单位所需时间	33	41	32
RGV 移动 3 个单位所需时间	46	59	46
CNC 加工完成一个一道工序的物料所需时间	560	580	545
CNC 加工完成一个两道工序物料的第一道工序所需时间	400	280	455
CNC 加工完成一个两道工序物料的第二道工序所需时间	378	500	182
RGV 为 CNC1#，3#，5#，7#一次上下料所需时间	28	30	27
RGV 为 CNC2#，4#，6#，8#一次上下料所需时间	31	35	32
RGV 完成一个物料的清洗作业所需时间	25	30	25

表 1：智能加统作业参数的 3 组数据表

1.3 问题的提出

针对下面的三种具体情况：

（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。

完成下列两项任务：

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。

二、 问题分析

2.1 任务 1 的分析

根据题目与附件提供的相关信息，我们绘出单工序模型下机械臂的状态转移图（见图 2）与双工序模型下机械臂的状态转移图（见图 3）。

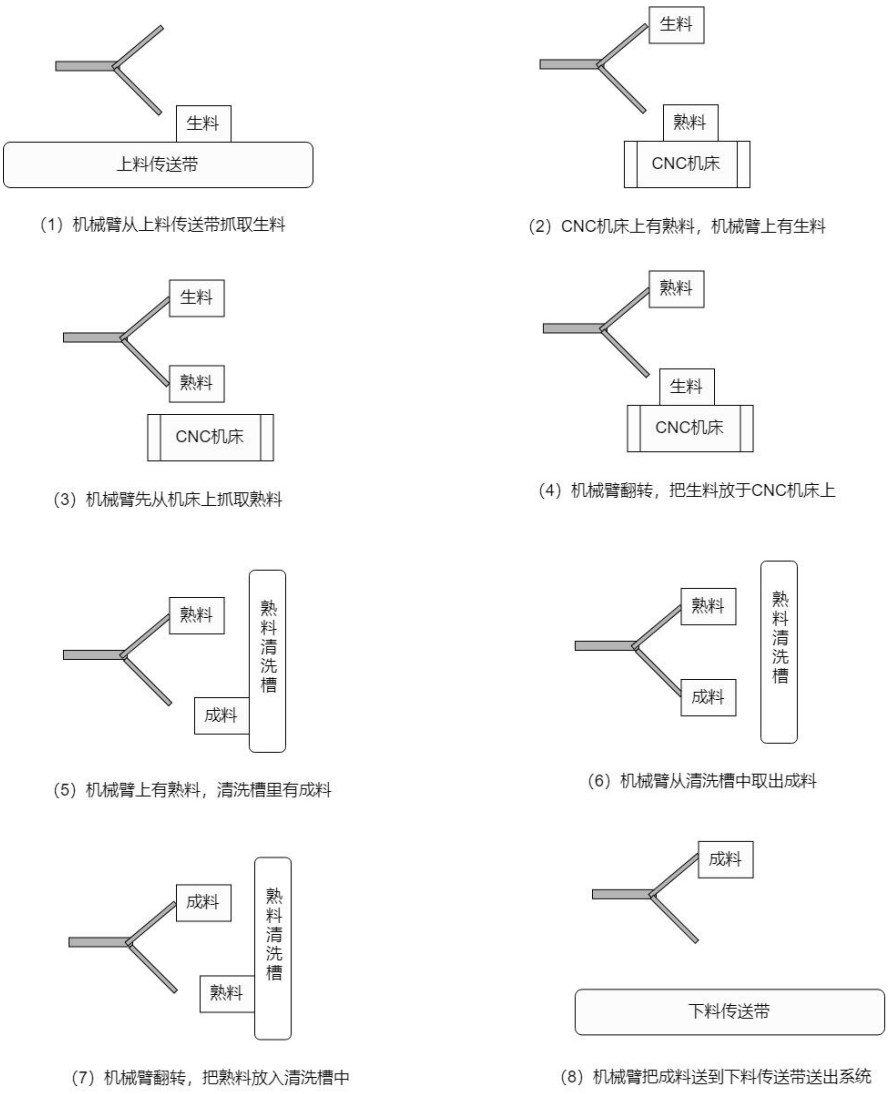
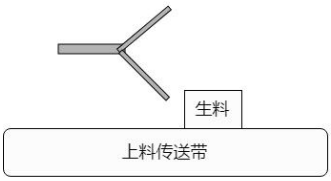
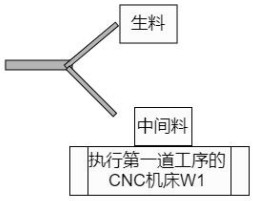


图 2 单工序模型下机械臂的状态转移图

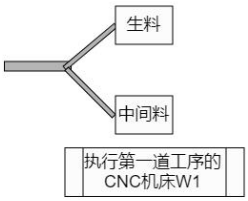
由图可知，机械臂的两个前爪可以旋转，RGV 每次使用一个前爪取出 CNC 机床上已经加工好的熟料，然后转动机械臂，将另一个前爪上的生料放上 CNC，使得 CNC 能连续工作；每次清洗时，CNC 的一个前爪取出清洗槽内已经处理完毕的成料，然后转动机械臂，将另一个前爪上的熟料放入清洗槽内，这种方式使得 RGV 一次可以执行两道操作，明显提高了工作效率。



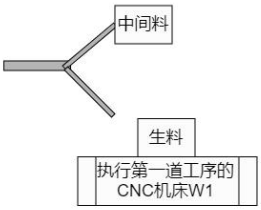
（1）机械臂从上料传送带抓取生料



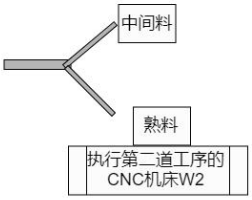
（2）CNC机床W1上有中间料，机械臂上有生料



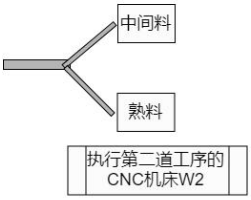
（3）机械臂先从机床W1上抓取中间料



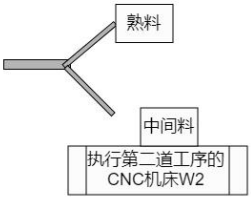
（4）机械臂翻转，把生料放于机床W1上



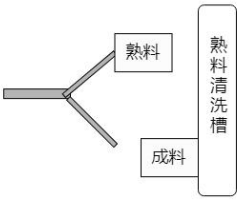
（5）CNC机床W2上有熟料，机械臂上有中间料



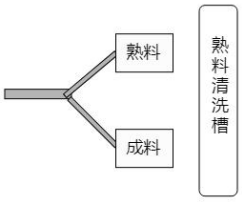
（6）机械臂从W2 上抓取熟料



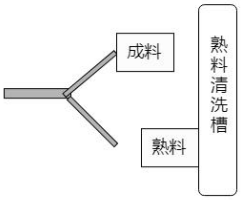
（7）机械臂翻转，把中间料放于机床W2上



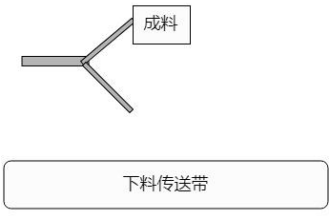
（8）机械臂上有熟料，清洗槽里有成料



（9）机械臂从清洗槽中取出成料



（10）机械臂翻转，把熟料放入清洗槽中



（11）机械臂把成料送到下料传送带送出系统

图 3 双工序模型下机械臂的状态转移图

由图可知，与单工序的状态转移图相比，双工序模型引入了中间料这一概念，中间料是指生料进行了第一次加工之后得到的中间产品，当 RGV 从机床上取出中间料之后，机械臂的一个前爪已被中间料占用，此时 RGV 必须到达执行第二次加工的机床位置，完成中间料的第二道工序并取出这个机床上已经加工完毕的熟料，否则，RGV 的下一次执行任务只有一只前爪可用，不利于整体效率的提升。

基于状态转移图，我们首先针对一道工序的物料加工作业情况，建立了单工序无故障 RGV 调度模型，并在此基础上进行修正，得到了双工序无故障模型、考虑故障的单工序 RGV 调度模型、以及考虑故障的双工序 RGV 调度模型。

2.2 任务 2 的分析

根据任务 1 中建立的 RGV 动态模型，利用表 1 中系统作业参数的 3 组数据分别经验模型的实用性和算法的有效性，完成附件 2 的 EXCEL 表格。

三、 基本假设

1. 不考虑传送带延迟时间；
2. 不考虑各工序之间的间隔时间；
3. 不考虑物料掉落，机器损坏，车间断电等小概率事件；
4. 各机床发生故障的概率相对独立，且故障时间满足正态分布；
5. CNC 向 RVG 发送请求的过程不消耗时间(或者总体时间极小不考虑)；

四、 符号说明

符号	说明
I	工件集合
M	机床集合
V	工序集合
i	工件代号
m	机床代号
J_i	工件 i 的工序集合，用{j}来表示
v	工序代号，用（i，j）表示

S_m	第 m 号机床的工序执行序列， 用{v1，v2,……,vm}表示； 调度序列为：{S_1，S_2，……，S_m}
g	工序执行代号，(i，m，v)表示第 i 个工件在 m 台机床上进行 v 工序
G	RGV 的工序执行序列，{g}
P	CNC 发生故障的概率
T_{mi}	第 i 次 CNC 发生故障 （发生在第 m 个 CNC）的时间
T_{si}	第 i 次 CNC 发生故障 （发生在第 m 个 CNC）的时刻
ts	对应工序的加工开始时刻
tl	对应工序的加工持续时间

五、模型的建立与求解

5.1 模型的建立

5.1.1 动态调度模型的整体符号约束与说明

在不考虑故障的情况下，每道工序由 g =（ i，m，v，ts，tl）确定，i 代表工件代号，m 代表 CNC 代号，v 代表工序代号（只考虑一道工序时，v 固定为 1；考虑两道工序加工时，v = 1 或 2），ts 代表该工序的加工开始时刻，tl 代表该工序的加工持续时间。

每台 CNC 的加工序列为：

$$S_{\mathrm{k}}=\{g_1,g_2,.....,g_n\},1<=k<=8,g_n=(i_n,m_n,v_m,t_{sn},t_{\mathrm{ln}})\text{且}\mathrm{m_n}=\mathrm{k};$$

RGV 的作业序列为：

$$G=\{\mathrm{g_1},\mathrm{g_2},.....,\mathrm{g_n},.....\};$$

智能 RGV 动态调度策略的解为 G，目标函数是模型在 8 小时内产生的成料数量 P。

$$P_{\max}=Total(G)_{\max}$$

CNC 的加工序列应满足调度问题中常见要求[1]：工件约束条件和机器约束条件，工件约束条件是指每个工件的工序顺序是固定的，必须先执行前一道工序，才能执行下一道工序；机器约束条件是指每台机器一次只能加工一个工序。此外，在本题中，CNC 上的生料是通过传送带进行输送，因此，我们提出传送带约束条件：RGV 的工作序列中，当前工序的开始时间与前一个序列的结束时间之间的时间间隔应当不小于 RGV 在两个位置之间的移动时间。

5.1.2 单工序无故障模型的约束条件

对单工序无故障模型，由于每个工件只需要执行一道工序，所以不考虑工件约束条件，模型约束条件可以表示为：

$$t_{sa}+t_{la}\le t_{sb},\ \ i_a,i_b,t_{sa},t_{sb},t_{lb}\in g_a,\ \ i_a<i_b,\ \ g_a\in S_k \cdots\cdots(1)$$

$$t_{s(a+1)}-(t_{sa}+t_{la})=dist(m_a,m_{a+1}),\ \ t_{sa},t_{la},m_a\in g_a,t_{s(a+1)},m_{a+1}\in g_{a+1} \cdots\cdots(2)$$

其中，（1）式表示机器约束条件，（2）式表示传送带约束条件。

5.1.3 双工序无故障模型的约束条件

对于双工序无故障的情况，我们需要考虑每个工件的工序执行顺序，只有执行了前一道工序后，才能执行后一道工序，因此我们加入工序约束条件修正模型：

$$t_{sa}+t_{la}\le t_{sb},\ \ i_a,v_a,t_{sa},t_{sb}\in g_a,\ \ i_b,v_b,t_{sb}\in g_b,\ \ i_a=i_b,v_a<v_b \cdots\cdots(3)$$

5.1.4 考虑故障模型的约束条件

假设每台 CNC 在加工过程中有 P 的概率发生故障，故障时刻用 T_{si} 表示，且故障持续时间 T_{mi}（i = 1,2,3,.....）符合正态分布，则有故障约束条件：

$$\forall T_{mi}\forall(g_k=(i_k,m_k,v_k,t_{sk},t_{lk}))，\ \ (m=m_k\Rightarrow t_{sk}<T_{mi}\vee t_{sk}>T_{mi}+T_{si})\cdots\cdots(4)$$

5.2 无故障的单工序动态调度模型的建立与求解

5.2.1 优先规则指派算法

对于单工序无故障的智能调度模型，我们首先使用优先规则指派算法[2]进行调度，在 RGV 的每个决策时刻，使用代价函数 $f(n)$ 对每个可执行的决策进行估值，再选择执行最小代价的任务，通过每次寻找局部最优解，使得求解序列 G 接近于全局最优解。在这一过程中，代价函数 $f(n)$ 的适用程度直接决定了近似解的好坏，我们引入 A*算法的思想，既考虑了当前时刻选取任务的执行时间，也考虑当前选取任务对 RGV 下一步决策的影响，具体表示为：

$$f(n)=\lambda*t(n)+\delta*h(n)$$

其中，$t(n)$ 表示 RGV 当前位置到执行决策 n 所需时间，$h(n)$ 表示假设 RGV 执行决策 n 完毕后，对当前任务队列中其余任务执行的拓展代价数值，同样以两者坐标点之间的移动时间，与相应的操作时间作为参考标准。两者分别具有不同的权重参数 λ、δ，参数数值的选取依据其对模型输出的影响，整体数值之和作为执行当前任务的代价值 f_n。

假设单工序系统系统处于图 4 所在的状态，CNC2#、CNC5#、CNC6#、CNC7#同时向 RGV 小车发送需求信号：

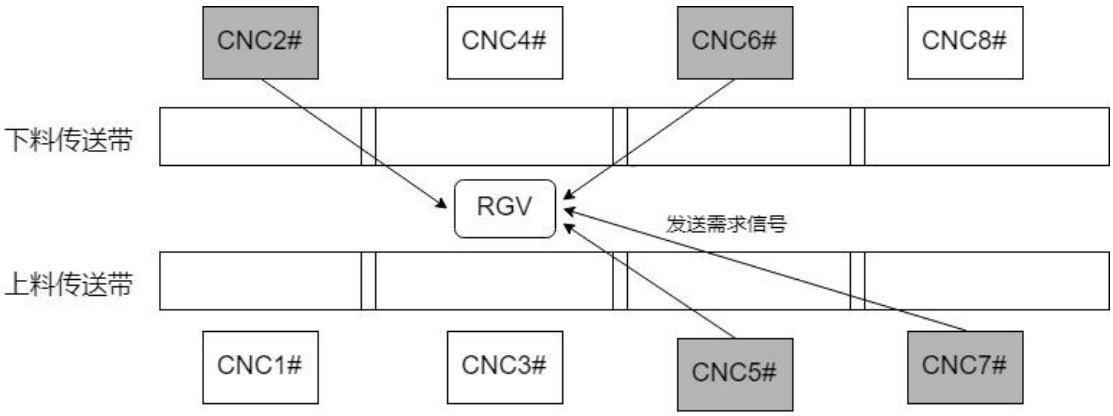


图 4

在此实例中，RGV 小车移动的决策队列为｛CNC2#，CNC5#，CNC6#，CNC7#｝，以计算 $f(CNC2)$ 为例，此处假设参数 λ、δ 皆为 1。RGV 从初始位置移动到 CNC2#前需要向左移动 1 个单位距离的时间，那么对应的 $t(CNC2\#)$ 为 20+31=51s；当 RGV 执行完第一步后，根据最短时间指派，RGV 接下来会执行 CNC5# 和 CNC6# 的请求，耗时分别为 33+28=61s，0+31=31s，所以 $h(CNC2\#)$ =61+31=92s；$f(CNC2)$ =51+92=143s。

执行完毕后，得到解 $G1$。

5.2.2 搜索优先规则指派算法最佳估价参数 λ、δ

为了得到关于估价函数两个变量函数对应的最优参数对，并且考虑到参数取值权重变化的非直观性，我们采取步长为 0.1，范围基于[1, 2]的循环取求解值法，充分获取两个变量双方对于各自的权重比值范围，对最佳参数进行逼近。

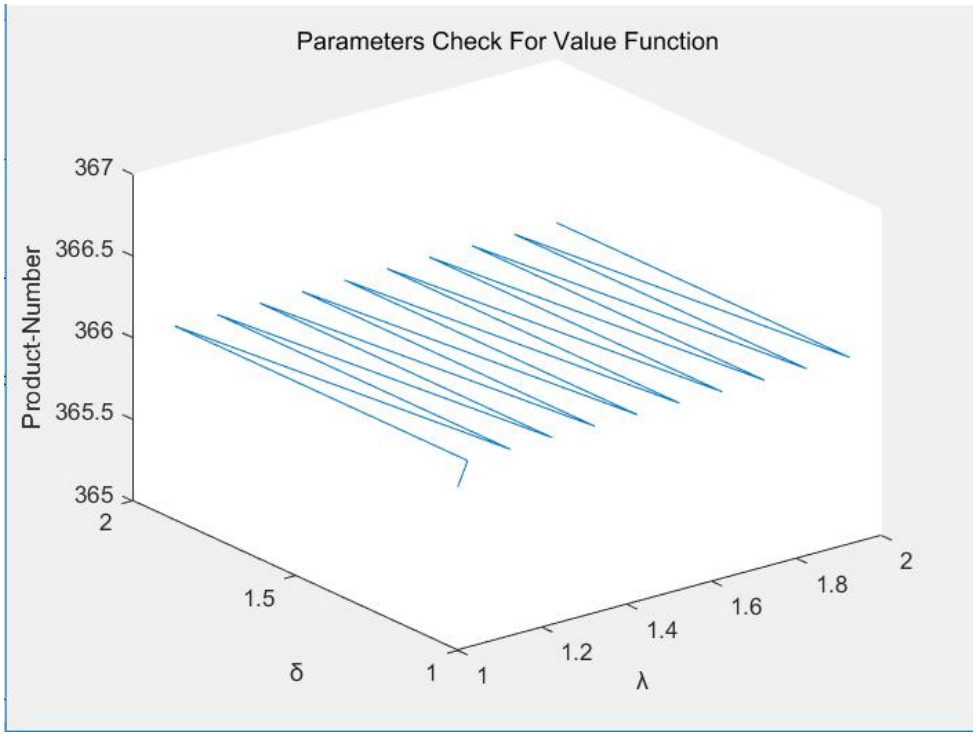


图 5

可以明显地由图 5 看到，三维图的 z 轴作为输出产品数，是参数性能的检测标准，然而位于[1, 2]区间的产出数基本没有大幅度，说明权重概念对于估价值并非关键因素；亦或是参数对(1, 1)一开始就位于最佳参数的附件位置。根据上述所得，此处约定所建模型中均取参数 λ、$\delta=1$

5.2.3 模拟退火算法

为了进一步优化， 得到更接近于最优解的近似解，我们使用模拟退火法[3]（算法流程见图 6），并对流程图里的内容进行阐述。

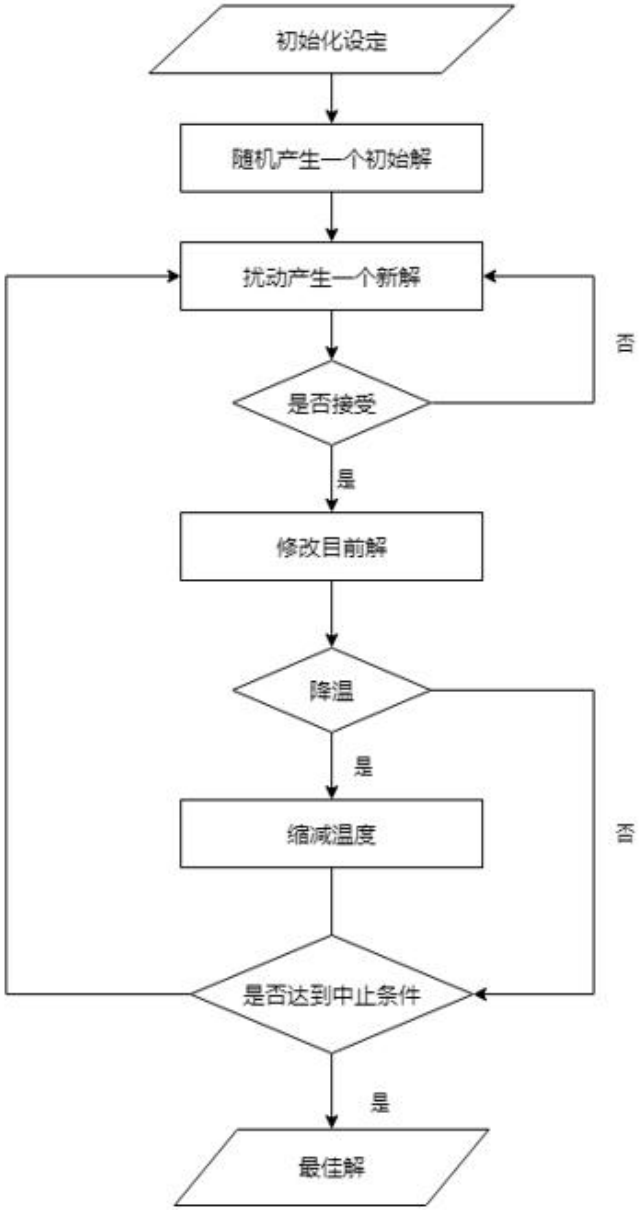


图 6 模拟退火算法流程图

(1) 初始解的产生

将优先规则指派算法的结果 G1 作为模拟退火法的初始解。

(2) 新解的产生

在该算法中，我们指定了两种新解的产生方式：

1)在 G1 中选择一个随机点 g_k，更改工序 g_k 的机床序号;

2).在 G 中选择一个随机点 g_k，并把它和之前某个工序做替换。 更改之后，我们舍弃更改点之后的工序，而选择从这个点开始，重新用优先规则指派算法产生后续的工序，若这个解满足约束条件，则将它作为新解 G'，否则，继续扰动直至新解产生。

(3) 接受准则

当新解 G'可行并且产生成料数量大于当前解时，接受新解，否则，以概率 $P2=\exp(\Delta c/T)$ 接受这个解，其中 Δc 是新解产生成料数量与当前解产生成料数量之差，T 为温度。

(4) 参数设计

经过初始温度的效果测试，以及每一步退火的步长数值的校正，我们为提高求解精度，使用初始温度 $T=100$，步长为 1，迭代次数位于 200~500 步之间的参数集，有目的地选择其中生成最优产出的合理解路径。

5.3 无故障的双工序动态调度模型的建立与求解

对双工序无故障的智能调度模型，设执行第一道工序的 CNC 集合为 W1，执行第二道工序的集合为 W2。与单工序无故障模型相比，双工序无故障模型在每个时刻的决策过程更为复杂，当 RGV 完成生料第一次加工后的上下料作业之后，机械臂的前爪位置被中间元件占用，此时 RGV 不能进行 W1 类的上下料作业，而只能将中间元件作为 W2 类 CNC 的生料，进行上下料操作，因此，我们更改代价函数:

$$f(n)=t(n)+h(n)+Z(v_n,v_{n-1})\text{，Z 为优先级指标}$$

考虑决策时刻的前一个工序，若 vn-1=vn，说明在该决策下，RGV 先后进行了相同的工序，显然是不符合约束条件的，所以此时 Z 的值应该足够大的负数，保证该决策不会被调度算法接受：

$$Z(v_n,v_{n-1})=\begin{cases}-100, & v_n=v_{n-1}\\ 0, & v_n\neq v_{n-1}\end{cases}$$

随后进行与单工序无故障模型相同的求解算法，得到近似解 G3。

5.4 考虑故障的模型的建立与求解

再考虑故障问题，我们假设每次加工过程发生故障的概率相互独立，且满足二项分布，故障影响时间满足正态分布，在每次 RGV 决策过程后，进行故障检测，若发生故障，则记录故障产生时刻与故障产生时间，在这段时间内，请求序列中不包括发生故障的 CNC，直至故障恢复

六、模型的检验与评价

我们使用极值分析的方法来验证单工序模型的正确性，在极限情况下，不考虑 RGV 运输生料和成料的时间，假设 8 台 CNC 都在进行不间断的加工，那么每个产品耗时为 $T_{加工}+T_{上下料}$，极限情况下的成料产量表示为：

$$P_{单工序极限}=\frac{8h*|M|}{T_{加工时间}+T_{上下料时间}+T_{清洗时间}}$$

同理，对于双工序模型的极限情况，我们假设处理第一道工序的 W1 类机床极限工作，并且 W2 类机床能将 M1 加工产生的中间料全部加工为成料，那么极限情况下的成料产量为：

那么，结合数据，得到极限值和测试结果的对比表格，见表 2:

$$P_{双工序极限}=\frac{8h*|W1|}{T_{第一道工序加工时间}+T_{上下料时间}*2+T_{清洗时间}}$$

（单位：数量）

	极限值	优先规则指派	模拟退火法
单工序无故障	384	375	378
双工序无故障	276	215	221
单工序有故障	357	328	331
双工序有故障	235	187	199

表 2

	极限值	优先规则指派	模拟退火法
单工序无故障	1.00	0.976	0.984
双工序无故障	1.00	0.778	0.801
单工序有故障	1.00	0.919	0.927
双工序有故障	1.00	0.795	0.846

表 3

表 3 是处理后的结果，从表中可以看出，算法的求解结果均在合理范围以内，客观上验证了算法的正确性，并在优先规则指派算法的基础上，进一步优化了近似解，符合我们的预期。

七、参考文献

[1] 曾凡丽，作业车间调度问题的高效率算法，华中科技大学，2011

[2] 邓泽林，求解车间作业调度问题的启发式算法， 华中科技大学， 2003.

[3] 冯玉蓉. 模拟退火算法的研究及其应用[D]. 昆明理工大学, 2005.

附录

具体 matlab 源程序见参考资料

1. 基于 Matlab 解决任务的仿真系统，使用任务队列储存并每次拿取队列头部作为目标，时间属性任务修改的时间序列系统(以单工序调度器为例)

```
%% 以启发式优先规则指派算法为核心的，单工序处理调度器函数 -- Matlab 实现 %%
% 输入参数：R1,R2,R3 分别为 RGV 机器人移动 1,2,3 个单位所需时间
% n 为一台 CNC 处理一道工序物料所需时间，T1,T2 为 RGV 分别为奇数，偶数 CNC 上下料时间
% W 为清洗一个熟料的时间
function [origin_count, produce_count, point_arr, time_arr, produce_arr, cncs_path, rgv_path] =
GreedySchedulerOne(R1, R2, R3, n, T1, T2, W, p1, p2)
origin_count = 0; % 记录所用生料数量
produce_count = 0; % 记录生成成品数量
cnc_state = zeros(8, 1); % 表示 8 台 CNC 当前状态，0 空闲，1 工作
rgv_pos = zeros(1, 1); % 记录 RGV 的位置
point_arr = rgv_pos; % 记录 RGV 位置移动的数组
time_arr = rgv_pos;
produce_arr = time_arr;
queue = java.util.LinkedList(); % 当前 CNC 需求任务队列
```

```
% 定义当前启发式全局解路径的存储结构，rgv_path 为 RGV 小车移动路径的收集，cncs_path 为所有的 CNC 机器的历史服务结构
rgv_path = java.util.LinkedList();
cncs_path = [];
for i=1:8
temp_queue = java.util.LinkedList();
cncs_path = [cncs_path, temp_queue];
end

% 定义数组结构体：struct('demand', 'order', 'type', 'val') -> [a1,a2,a3,a4], 0 in demand for begin, 1 for end
for i=1:8
task = [0, i, mod(i,2)~=0, abs(rgv_pos - ((i + (mod(i, 2)~=0))/2-1))];
queue.push(task);
end
```

```
time = 0;
temp_count = 1;
count = 0;
while ~queue.isEmpty() && time <= 28.8
% 记录半小时生成产品数，输出用于描述数据趋势
if time/1.8 >= temp_count
time_arr = [time_arr, time];
produce_arr = [produce_arr, produce_count];
temp_count = temp_count + 1;
end
% 启发式搜索的核心，每次操作收取新任务，以及更新每个任务估价，排序
[cnc_state, queue] = CheckStateAndRecover(cnc_state, queue, time, rgv_pos);
[cnc_state, queue] = ReceiveDemandAndSort(time, cnc_state, rgv_pos, queue, R1, R2, R3, T1, T2, p1, p2);
top = queue.poll();
rel_dis = abs(rgv_pos - ((top(2) + top(3))/2-1));
switch rel_dis
case {1}
time = time + R1;
case {2}
time = time + R2;
case {3}
time = time + R3;
end
rgv_pos = ((top(2) + top(3))/2-1);
if rel_dis ~= 0
point_arr = [point_arr, rgv_pos];
end
```

```
if top(1) == 0
% 上料过程
temp_T = 0;
if top(3) == 1
temp_T = T1;
else
temp_T = T2;
end
origin_count = origin_count + 1;
[cncs_path, rgv_path] = GainInitResult(cncs_path, rgv_path, origin_count, top(2), top(3), time,
temp_T+n);
time = time + temp_T;
cnc_state(top(2)) = time + n;
[cnc_state, count] = LossJudger(cnc_state, time, count, produce_count);
else
% 拿取熟料过程
temp_T = 0;
if top(3) == 1
temp_T = T1;
else
temp_T = T2;
end
produce_count = produce_count + 1;
% 放下生料过程
origin_count = origin_count + 1;
[cncs_path, rgv_path] = GainInitResult(cncs_path, rgv_path, origin_count, top(2), top(3), time,
temp_T+W+n);
time = time + temp_T;
cnc_state(top(2)) = time + n;
[cnc_state, count] = LossJudger(cnc_state, time, count, produce_count);
% 清洗熟料过程
time = time + W;
end
% 无请求，RGV 待机情况
while queue.isEmpty() && time <= 28.8
time = time + 0.001;
[cnc_state, queue] = CheckStateAndRecover(cnc_state, queue, time, rgv_pos);
[cnc_state, queue] = ReceiveDemandAndSort(time, cnc_state, rgv_pos, queue, R1, R2, R3, T1, T2, p1,
p2);
end
end
time_arr = [time_arr, time];
produce_arr = [produce_arr, produce_count];
disp(count);
```

```
end
```

2. 根据启发式优先规则指派法的估价函数

```
%% 确定某个 CNC 处理次序的启发式贪心估价函数，指标为小车运行当前任务，以及假设小车执行该任务其后的各项任务的总体时间 %%

function [queue] = Valuation(rgv_pos, queue, R1, R2, R3, T1, T2, p1, p2)
for i=0:queue.size()-1
result = 0;
temp = queue.get(i);
% 首先加上上下料处理时间指标
if temp(3) == 1
result = result + T1;
else
result = result + T2;
end
% 基于 RGV 机器人到当前任务直线距离的朴素贪心时间指标
switch abs(rgv_pos - ((temp(2) + (mod(temp(2), 2)~=0))/2-1))
case {1}
result = result + R1;
case {2}
result = result + R2;
case {3}
result = result + R3;
end
result = result * p1; % 参数指标权重赋予
% 基于假设执行当前任务，对之后的所有任务执行时间综合影响的启发式指标
temp_rgv_pos = ((temp(2) + (mod(temp(2), 2)~=0))/2-1);
temp_result = 0;
temp_queue = queue.clone();
temp_queue.remove(i);
% 按照距离排序其余其他任务，找出假设运行该点任务之后的最优执行路径，将路径总执行时间作为评估值
temp_queue = BubbleSort(temp_queue, temp_rgv_pos);
while ~temp_queue.isEmpty()
tmp = temp_queue.poll();
tmp_point = ((tmp(2) + (mod(tmp(2), 2)~=0))/2-1);
switch abs(temp_rgv_pos - tmp_point)
case {1}
temp_result = temp_result + R1;
case {2}
temp_result = temp_result + R2;
case {3}
temp_result = temp_result + R3;
end
temp_rgv_pos = tmp_point;
```

```
temp_queue = BubbleSort(temp_queue, temp_rgv_pos);
end
queue.set(i, [temp(1), temp(2), temp(3), result + p2 * temp_result]);
end
```

```
end
```

3. 模拟退火法的实现(以单工序调度器为例)

```
%% 模拟退火算法优化单工序调度器所得解函数 -- Matlab 实现 %%

function [optimal_result, optimal_produce_num] = SimulatedAnnealingScheduler(init_result,
init_num, init_temp, iterate_time, R1, R2, R3, T1, T2, n, W)
optimal_result = init_result;
optimal_produce_num = init_num;
% 在指定的迭代步数中进行循环，进行随机干扰获得近似解，判断是否接受该新解
for i=1:iterate_time
% 均与分布产生随机数，表示选择的路径点下标
index = round(0 + (optimal_result.size()-2)*rand(1));
% 以随机下标作为基准点，向前考虑与其他不同编号的 CNC 机器路径点进行交换，以 RGV 路径生成的约束作为条件
different_index = [];
history_records = [];
for k=index-1:-1:0
if ~ismember(optimal_result.get(k), history_records)
history_records = [history_records, optimal_result.get(k)];
different_index = [different_index, k];
end
end
constrain_flag = false;
while ~constrain_flag && ~isempty(different_index)
% 以均匀分布，随机选择该数组中的单个元素进行交换
len = length(different_index);
ran_index = round(1 + (len-1)*rand(1));
tmp = different_index(ran_index);
% 取出后的节点则去除
different_index(ran_index) = [];
temp_result = optimal_result.clone();
[temp_result] = SwapCNCID(index, tmp, temp_result);
% 截取临时处理队列中的前 1-tmp 位，进行队列后段的舍弃(每一个节点都改变都直接影响到全局路径的生成)
temp_result = java.util.LinkedList(temp_result.subList(0, tmp+1));
% 进行队列生成规则与约束条件的检测与校验
ttmp = temp_result.get(tmp);
constrain_flag = true;
for j = tmp-1:-1:0
```

```
jmp = temp_result.get(j);
if ttmp(2) == jmp(2) && ttmp(4) + ttmp(5) > jmp(4) % 倘若不满足约束条件，则重新随机寻找基准点前的可
替换下标
constrain_flag = false;
end
end
% 成功找到符合约束条件的变化后路径起始节点段，进行启发式拓展
if constrain_flag == true
% 进行启发式优先规则的新路径拓展生成
[new_path, new_product_num] = HeuristicExpand(temp_result, R1, R2, R3, T1, T2, n, W);
% 拿取新解的产品数，进行评估与取舍
if new_product_num <= optimal_produce_num
% 新解产出小于等于之前解的产出，则以一定概率接受此较差的解
accept_pro = exp((new_product_num - optimal_produce_num) / init_temp);
rand_temp = rand(1);
if rand_temp <= accept_pro
optimal_result = new_path;
optimal_produce_num = new_product_num;
end
else
% 新解产出比之前解产出大，果断接受新解
optimal_result = new_path;
optimal_produce_num = new_product_num;
end
% 退火的关键 -- 温度减小
init_temp = init_temp - init_temp / 100;
end
end
if init_temp <= 0
break;
end
end
```

```
end
```

4. 面向模拟退火随机初始序列的优先规则拓展模块

```
%% 抽取启发式规则生成路径算法，独立出为特定的模拟退火算法随机初值路径以启发式规则生成的新路径解函数 %%

function [front_path, produce_count] = HeuristicExpand(front_path, R1, R2, R3, T1, T2, n, W)
% 依据已经得到的前段路径，恢复当前系统各项状态值
lastNode = front_path.getLast();
time = lastNode(4);
origin_count = 0;
```

```
produce_count = 0;
count = 0;
rgv_pos = (lastNode(2) + (mod(lastNode(2), 2)~=0))/2-1;
cnc_state = zeros(1, 8);
queue = java.util.LinkedList();
```

```
for i=0:front_path.size()-1
% 恢复当前系统 RGV 小车位置，已生成的成品数，和投入的生料数，以及每个 CNC 位于当前时间点上的状态
temp = front_path.get(i);
origin_count = origin_count + 1;
if temp(4) + temp(5) < time
produce_count = produce_count + 1;
cnc_state(temp(2)) = 0;
else
cnc_state(temp(2)) = temp(4) + temp(5);
end
end
```

```
% 进行基于当前系统情况的启发式任务执行，记录新生成的路径
% 首先进行任务队列的初始化
for i=1:8
if cnc_state(i) == 0
task = [0, i, mod(i,2)~=0, abs(rgv_pos - ((i + (mod(i, 2)~=0))/2-1))];
queue.push(task);
end
end
% 然后进行循环处理
while ~queue.isEmpty() && time < 28.8
[cnc_state, queue] = CheckStateAndRecover(cnc_state, queue, time, rgv_pos);
[cnc_state, queue] = ReceiveDemandAndSort(time, cnc_state, rgv_pos, queue, R1, R2, R3, T1, T2, 1,
1);
top = queue.poll();
rel_dis = abs(rgv_pos - ((top(2) + top(3))/2-1));
switch rel_dis
case {1}
time = time + R1;
case {2}
time = time + R2;
case {3}
time = time + R3;
end
rgv_pos = ((top(2) + top(3))/2-1);
if top(1) == 0
% 上料过程
```

```
temp_T = 0;
if top(3) == 1
temp_T = T1;
else
temp_T = T2;
end
origin_count = origin_count + 1;
new_rgv_node = [origin_count, top(2), top(3), time, temp_T+n];
front_path.addLast(new_rgv_node);
time = time + temp_T;
cnc_state(top(2)) = time + n;
[cnc_state, count] = LossJudger(cnc_state, time, count, produce_count);
else
% 拿取熟料过程
temp_T = 0;
if top(3) == 1
temp_T = T1;
else
temp_T = T2;
end
produce_count = produce_count + 1;
% 放下生料过程
origin_count = origin_count + 1;
new_rgv_node = [origin_count, top(2), top(3), time, temp_T+n+W];
front_path.addLast(new_rgv_node);
time = time + temp_T;
cnc_state(top(2)) = time + n;
[cnc_state, count] = LossJudger(cnc_state, time, count, produce_count);
% 清洗熟料过程
time = time + W;
end
% 无请求，RGV 待机情况
while queue.isEmpty() && time <= 28.8
time = time + 0.001;
[cnc_state, queue] = CheckStateAndRecover(cnc_state, queue, time, rgv_pos);
[cnc_state, queue] = ReceiveDemandAndSort(time, cnc_state, rgv_pos, queue, R1, R2, R3, T1, T2, 1,
1);
end
end
end
```