中国研究生创新实践系列大赛

中国光谷·“华为杯”第十九届中国研究生数学建模竞赛

学　　校	西南交通大学
参赛队号	22106130232
队员姓名	1. 俞建民
	2. 赵松
	3. 冯聪

中国研究生创新实践系列大赛

中国光谷·“华为杯”第十九届中国研究生数学建模竞赛

题 目 涂装–总装缓存区调序调度优化问题

摘 要：

本文主要讨论了汽车制造涂装-总装缓存调序区调度优化问题，通过 PBS 建模对涂装区-PBS 入口进入 PBS 缓冲区的车身序列进行重组排序以适应总装车间的生产序列。通过对整个题目的分析可知，问题分为数学建模和算法求解两个部分，其中 PBS 调序优化建模部分包含了车辆信息与车道匹配建模、车身在车道前进规则建模、横移机优先运送规则建模以及各类车间约束建模等。算法求解部分包括遗传算法的使用，以及遗传算法的改进优化。

对于问题 1 而言：首先，以附件 3 所提供的位置编码数据为切入点，考虑附件 1 中车身固有属性（车型、配置、驱动方式），设定决策变量 x_{it}^{m} 来刻画每个时刻车身所处的车位位置。根据题目所给约束条件和优化目标，建立以车身位移 *OD* 对为下角标，车身属性和时间为上标的决策变量，刻画每一个车身在车位之间的位置转移过程。然后根据题目所给 PBS 约束和时间约束，利用多目标优化建模思维，建立 PBS 优化调度模型。同时需要考虑车身在车道上的前进规则和横移机的运车优先规则，且明确车道上车位与车位之间和车位与横移机之间转移时间长度。利用 PBS 缓冲区对涂装顺序进行重组排序，利用横移机对不同进车道和返回车道的进车和出车车身进行位置分配，从而实现车身顺序序列与总装车间生产序列的最优匹配。对于问题 1 而言，需要考虑约束 6、7 中两个横移机优先运送规则，届时可能会出现总时间 T 增加的情况，所以在建立模型时充分考虑对应算法的运算速度，避免运算时间过长。由于题目给出了固定的评分规则，利用评分来评价模型优化程度，同时给出紧约束和时间说明，所以多目标优化最为合适，建立线性规划模型，也利于求解。通过建模以及算法求解可知，考虑四个优化目标及题目中所给约束，利用遗传算法求解得到最终评分为 13.67，第一辆车身从涂装-PBS 进口进入 PBS 缓冲区，到最后一辆车身经过 PBS-总装出口进入总装车间总共耗时 4022 秒。

对于问题 2 而言：相比于问题 1，问题 2 是在问题 1 的基础上减少了 6、7 中两个约束条件，然后进行 PBS 优化建模。问题 2 的其他约束条件和相关变量设定完全同问题 1。同样考虑涂装-PBS 的进车序列与 PBS-总装出车序列之间的重组排序问题，但是在求解问题 2 的时候，需要仔细思考缺少横移机规则情况下模型的优化能力问题，以及对应算法求解的速度问题。总之，问题 2 和问题 1 的区别在于少约束，同时也是模型的简化，但是所缺少约束对于最后优化目标的影响可能是很大的，这在最后的模型评价和算法改进部分有所考虑。通过建模和算法求解可知，问题 2 相比于问题 1 减少 6、7 两个约束，同样考虑四个

优化目表标，进行遗传算法求解得到最终评分为 26，第一辆车身从涂装-PBS 进口进入 PBS 缓冲区，到最后一辆车身经过 PBS-总装出口进入总装车间总共耗时 3146 秒。

关键词：多目标优化模型；PBS 缓冲区；遗传算法；模型优化

目录

1. 问题重述

1.1 问题背景

缓冲区问题是整个工业生产行业在生产计划制定流程中的一个重要环节，在遵守各车间作业区约束且满足总生产计划的条件下科学合理的安排缓冲区生产序列以保证工厂生产任务的正常生产，并尽可能的减少因为生产序列不匹配而带来的经济损失和资源浪费。目前，我国大多数生产企业的缓冲区的重排序工作主要是依靠 RFID 技术和 OPC 技术，两者具有不便于整合到移动设备中、作用距离短、缺少稳定性和鲁棒性等特点。因此，基于现代技术的缺点，迫切需要建立一种能够依靠自身调序实现生产序列自由转换的缓冲区，从而实现生产序列的合理排列。

缓冲区问题是运筹学和生产计划交叉领域的一类重要研究课题。在过去三十年里，学者们关于此类问题提出了许多成熟的优化模型和求解算法。模型方面，针对虚拟重排序问题，韩建明[1]针对 WBS 车辆颜色切换问题的虚拟重排序模式，构建 0-1 整数规划模型，并使用贪婪规则和束搜索算法求解。算法方面，胡婷婷[2]提出了基于两阶段匹配度的 PBS 缓冲区车身路由调度策略；陈正茂[3]提出了外层算法采用遗传算法，内层采用启发式规则进出缓冲区的组合算法，极大的提高了算法的求解速度。本文进站同样采用遗传算法。Muhl 等人[4]研究汽车在各个生产车间的生产约束和不确定因素，针对缓冲区车辆排序问题，提出一种局部排序方法，并使用启发式算法求解；

对于本文研究的汽车制造涂装-总装缓冲区调序调度优化问题，其实就是缓冲区问题的具体化，考虑到涂装车间与总装车间序列差异较大且各车间的约束不同，会导致生产调度无法按照同一序列连续生产，这就需要在两个车间之间建立一个具有调序功能的缓存区，即 PBS（Painted Body Store，汽车制造涂装-总装缓存调序区），用来将涂装车间的出车序列调整到满足总装车间约束的进车序列。通常此类问题运用多目标优化的方法来建模，然后运用遗传算法、启发式算法等来进行求解。但是建模的效果都比较差，求解的算法思路都大体一致。同时也存在最优化效果不好，精度不高以及求解速度慢等问题。因此，我们希望充分考虑涂装出车序列、PBS 区域调度能力、PBS 调度模式限制，建立 PBS 优化调度模型，利用结合遗传算法的创新算法对每时每刻车辆位置进行标号和输出结果，从而实现车身的实时位置输出，监测生产序列重排序列的合理性。

1.2 需解决的问题

问题一：严格按照 PBS 约束说明及相关时间数据说明，根据涂装出车序列，考虑 PBS 区域调度能力及限制，建立 PBS 优化调度模型，使得总装进车序列尽可能满足总装生产需求。给出将你们的优化调度方案分别应用于附件 1 和附件 2 数据的得分结果，并将使用附件 1 和附件 2 两套数据的调度输出结果按照附件 4 格式分别存入 result11.xlsx 和 result12.xlsx。

问题二：如果去除 PBS 约束说明中第 6、7 两条约束，其余约束不变，根据涂装出车序列，考虑 PBS 区域调度能力及限制，建立 PBS 优化调度模型，使得总装进车序列尽可能满足总装生产需求。给出将你们的优化调度方案分别应用于附件 1 和附件 2 数据的得分结果，并将使用附件 1 和附件 2 两套数据的调度输出结果按照附件 4 格式分别存入 result21.xlsx 和 result22.xlsx。

1.3 PBS 约束说明

1. 送车横移机不能将返回道的车身送入 PBS-总装接车口。
2. 车身在进车道和返回道的移动方向为图中标注方向，不得改变。
3. 接车横移机和送车横移机上同一时刻分别最多有一个车身。
4. 接车横移机和送车横移机在完成任意动作后，必须返回中间初始位置，才可以执行下一步动作。
5. 接车横移机和送车横移机在执行任何动作过程中，均不能被打断。
6. 当返回道 10 停车位有车身，同时接车横移机空闲时，优先处理返回道 10 停车位上的车身。
7. 当若干进车道 1 停车位有车身等候，同时送车横移机空闲时，优先处理最先到达 1 停车位的车身。
8. 如果任意进车道 1 停车位有车身，那么送车横移机不能设置为空闲状态。
9. 进车道和返回道每个时刻最多容纳 10 个车身，每个停车位最多容纳 1 个车身。
10. 同一车道内，多个车身在不同停车位上的移动可以不同步进行。
11. 当某车身所在停车位的下一停车位出现空位时，车身必须立即开始向下一停车位移动。
12. 车身在进车道和返回道不同停车位之间移动的过程中，不能被调度。

1.4 优化目标

1. 混动车型间隔 2 台非混动车型为优，权重系数 0.4。
2. 四驱车型与两驱车型倾向 1:1 出车序列，权重系数 0.3。
3. 返回道使用次数倾向于 0，权重系数 0.2。
4. 倾向总调度时间越短越好，权重系数 0.1。

注：该权重系数用于多目标得分加权，各权重系数相加等于 1。

2. 模型假设与符号说明

2.1 模型假设

根据题目要求，本文在解题前做出如下模型假设：

假设一：题目所给的数据真实可靠；

假设二：假设车身序列能完全输出 PBS 进入总装车间；

假设三：假设除所需分析变量外不再有外生变量的影响；

假设四：第二问假设减少约束 6、7 不会对模型合理性产生影响。

假设五：若车移动动作需要 s 秒，则假设动作在 s 秒末完成。

2.2 符号说明

表 2-1　符号说明

决策变量	
x_{it}^{m}	决策变量，0-1 变量，$x_{it}^{m}=1$表示车身 m 在 t 时刻时在 i 位置上
中间变量	
$y_{(i,\ j)}^{m\ t,t+1}$	0-1 变量，$y_{(i,\ j)}^{m\ t,t+1}=1$表示车身 m 在 t 时刻由位置 i 转到位置 j
$zoneA_t$	$zoneA_t=[StartA_t,\ EndA_t]$表示送车横移机完成任务 $y_{(i,2)}^{m\ t,t+1}$ 的整个时段；$StartA_t$：开始时间，$EndA_t$：结束时间
$zoneB_t$	$zoneB_t=[StartB_t,\ EndB_t]$表示送车横移机完成任务 $y_{(1,\ i)}^{m\ t,t+1}$ 的整个时段；$StartB_t$：开始时间，$EndB_t$：结束时间
a	进车道 1 停车位有车身的车位中，横移机能最先到达的那个车位
Out_t	0-1 变量，Out_t=1 表示 t 时刻有车到达总装-PBS 接车口
$time_m$	车 m 到达总装-PBS 接车口时间
$order_m$	序号，车 m 到达总装-PBS 接车口的顺序序号
h_k	0-1 变量，h_k=1 表示序号为 k 的车辆是混动车型；h_k=0 表示序号为 k 的车辆是非混动车型
W	目标总得分
W_1	目标 1 得分
W_2	目标 2 得分

W_3	目标 3 得分
W_4	目标 4 得分
z	0-1 变量， $z \in \{0,1\}$
d_k	0-1 变量， d_k =1 表示目标 1 扣 1 分

参数与集合

M	车身集合
m	车身， $m \in M$
I	位置总集合， $I = \{I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup \{0,1,2,3\}\}$
I_i	车道 i 的位置集合, i =1,2,3,…,7
T	时间戳集合
t	时间戳， $t \in T$
OD	可行位移集合 $OD = OD_l \cup OD_o$
OD_l	车身在车道上的可行位移集合
OD_o	车身在车道和横移机的可行位移集合
(i, j)	OD 对， $(i, j) \in OD$
I'	进车道 1 停车位集合， $I' = \{11, 21, 31, 41, 51, 61\}$
H	（序号，混动车型）对集合
E	（序号，混动车型）对集合

参数

S	$S = \{S_{11}, S_{21}, S_{31}, S_{41}, S_{51}, S_{61}, S_{110}, S_{210}, S_{310}, S_{410}, S_{510}, S_{610}\}$ $= \{18,12,6,0,12,18,18,12,6,0,12,18\}$ ， 进车过程、出车过程分别对每条车道消耗的时间
R	$R = \{R_{11}, R_{21}, R_{31}, R_{41}, R_{51}, R_{61}, R_{110}, R_{210}, R_{310}, R_{410}, R_{510}, R_{610}\}$ $= \{24,18,12,6,12,18,24,18,12,6,12,18\}$ 车进/出返回道横移机分别耗时,计算可得 $\{R_{71}, R_{710}\} = \{6,6\}$
B	无穷大量

其中：

$I_1 = \{11,12,13,14,15,16,17,18,19,110\}$

$$OD_l = \begin{Bmatrix} (110,19),(19,18),(18,17),(17,16),(16,15),(15,14),(14,13),(13,12),(12,11), \\ (210,29),(29,28),(28,27),(27,26),(26,25),(25,24),(24,23),(23,22),(22,21), \\ (310,39),(39,38),(38,37),(37,36),(36,35),(35,34),(34,33),(33,32),(32,31), \\ (410,49),(49,48),(48,47),(47,46),(46,45),(45,44),(44,43),(43,42),(42,41), \\ (510,59),(59,58),(58,57),(57,56),(56,55),(55,54),(54,53),(53,52),(52,51), \\ (610,69),(69,68),(68,67),(67,66),(66,65),(65,64),(64,63),(63,62),(62,61), \\ (71,72),(72,73),(73,74),(74,75),(75,76),(76,77),(77,78),(78,79),(79,710) \end{Bmatrix}$$

$$OD_o = \begin{Bmatrix} (0,1),(710,1), \\ (1,110),(1,210),(1,310),(1,410),(1,510),(1,610), \\ (11,2),(21,2),(31,2),(41,2),(51,2),(61,2), \\ (2,3),(2,71) \end{Bmatrix}$$

3. 问题分析

3.1 问题 1 分析

问题 1 属于运筹学多目标优化问题和生产计划排序问题的交叉领域问题，解决此类问题一般通过多目标线性规划数学建模，然后通过遗传算法、启发式算法求解。当然，也可以看成排队网络优化问题进行建模，利用算法进行模型求解和优化。但本题需要考虑的开放性约束较多，这里构思解决此类问题方法需要改进。首先对问题 1 进行分析可知，所需建模区域主要由涂装、PBS（Painted Body Store，汽车制造涂装-总装缓存调序区）、总装区域组成，涂装顺序不一定满足总装顺序，所以 PBS 调序区就需要对生产顺序进行重新编组排序。首先车身按照涂装区域涂装顺序，从涂装-PBS 入口进入 PBS 调序区，这里需要设定选择策略，来确定车身进入 PBS 调序区的顺序序列，这里有很多方法，例如 Mont-Carlo 法、随机森林等。但是这里需要基于车身的车型、配置和驱动方式等属性，建立车身和车道的匹配程度模型。从而确定初始车身序列进入 PBS 涂装区的顺序。考虑到题目要求和车间约束，设定决策变量 x_{it}^{m}，

i 表示车位编号，m 表示车身顺序编号，t 表示离散时间。每一辆车在每一个时刻点会对应一个具体位置，相应就会匹配到车位编码，同时车位编码也对应到每一辆具体车身的属性。

再设定中间变量 $y_{(i,\ j)}^{m\ t,t+1}$，i 表示该车当前车位编号，j 表示该车下一位置的车位编号，m 表示车身顺序编号，t 表示离散时间。$y_{(i,\ j)}^{m\ t,t+1}$ 表示车身从当前位置是否前进到下一位置的状态转移过程，取值为 1 时表示车身 m 从 i 位置移动到 j 位置，否则没有。其中 (i,j) 是包含位置编码的 OD 对。本题所使用的的多目标规划模型包含 4 个优化目标，题目所给约束 12 个，其余还有一些隐含约束。在第五部分，我们完整给出了 PBS 多目标调序优化模型，并在第三部分对每个变量作出了明确的定义和区分。通过模型对车身序列作出重组排序动作，然后以适应生产计划的序列通过 PBS-总装出口进入总装车间。对于整个多目标优化建模而言，不仅要考虑时间推移步长的大小，还要考虑横移机优先运送规则，所以这里增加了一些新的变量，详见第三部分。整个问题一被分成了两个部分，第一部分是建立 PBS 调度优化模型，这里直接采用多目标优化线性规划模型，同时进行模型的对比和评价。第二部分则是根据第一部分建立的多目标优化模型进行编程实现，同时进行算法的评价和改进。

涂装-PBS-总装区域的车身序列重组过程分为两个选择策略：一个是车身进入 PBS 缓冲区域是对进车道的选择策略，另一个是出站是车身对 PBS-涂装和返回车道的选择策略。这里进站选择策略采用进站启发式规则，按照优先级由高到底的顺序进行，如下所示：

（1）如果进站车身与某进车道 1 停车位停放车身属性一致，则选择进入该车道；

（2）假设无法匹配到（1）中所描述车道，就随机选择任意 10 停车位空闲的进车道；

（3）如果（1）、（2）中所描述的车道都找不到，那么该车身暂时停放在横移机上，等待符合描述车道出现为止。

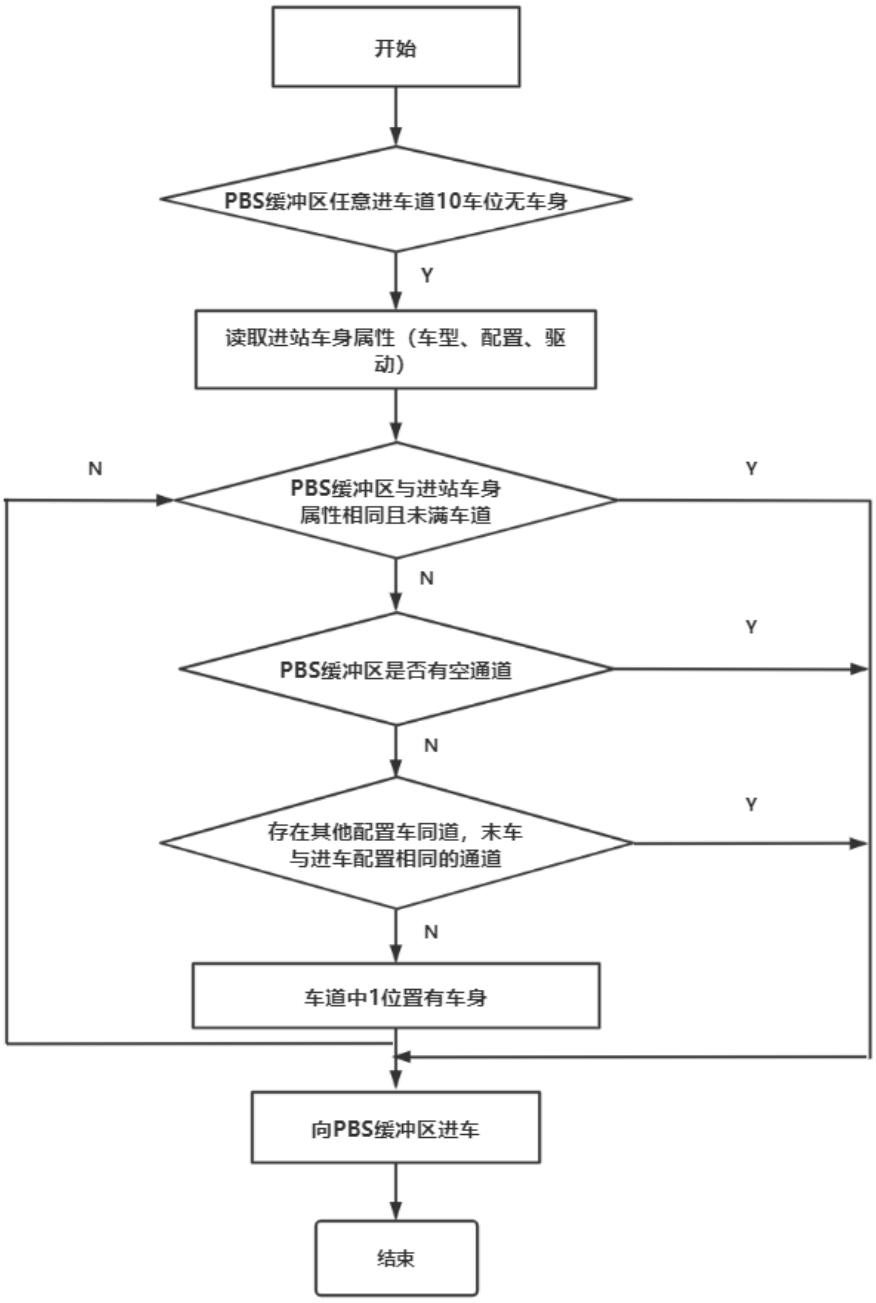


图 3-1 PBS 进站调度方法

出站启发式规则（按照优先级由高到低的顺序）：
（1）依据四个优化目标扣分原则进行车身出站顺序选择；
（2）如果不符合（1）中出站条件，则进入返回车道。

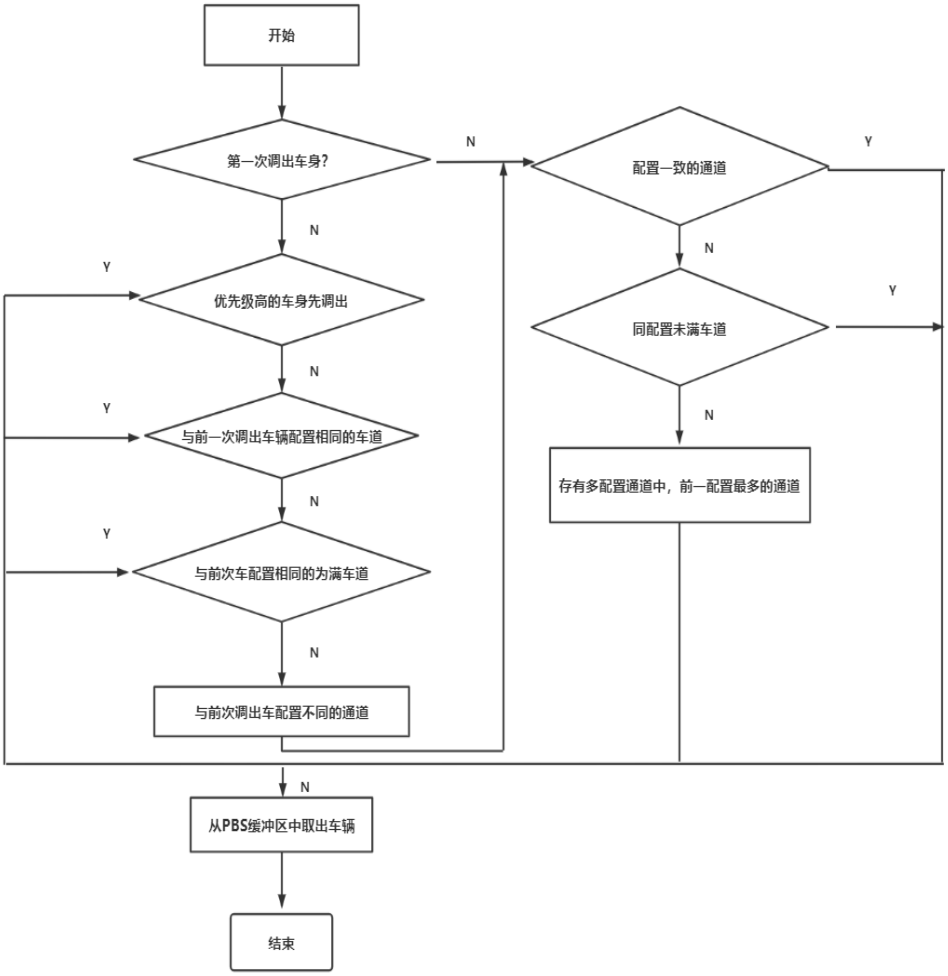


图 3-2 PBS 出站调度方法

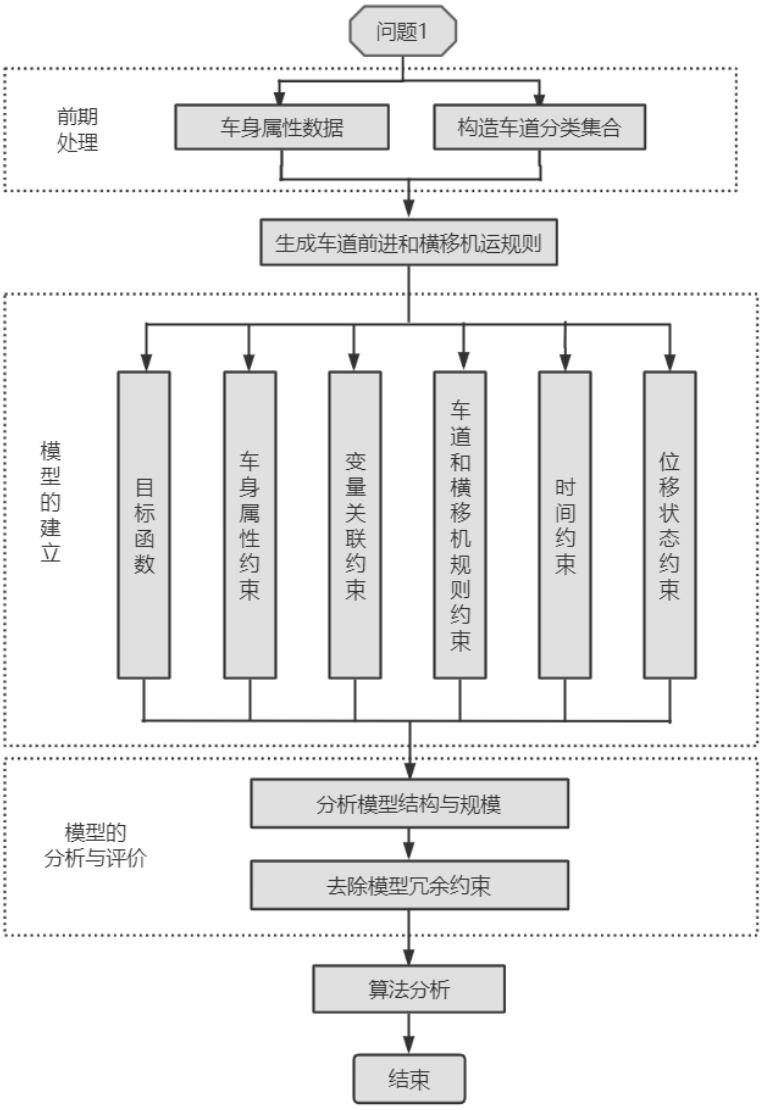


图 3-3 问题解决方案

图 3-4 优化目标 1 建模图例

优化目标 1 建模思路：将最终出车顺序序号 1,2,3,…,318，并根据每一个车身的配置不同（混动和非混动）编号 0 或 1，其中 0 代表非混动车型，1 代表混动车型。通过数学模

型可以用决策变量表示出每辆车的出车顺序序号。根据序号建立（序号，混动车型）集合，剔除其中非混动元素，得到混动元素的（序号，1）集合。再用后一元素序号减去前一元素序号得到两混动车辆的间隔。当间隔为 3 时，说明混动车辆之间的非混动车辆数为 2，此时不扣分。当间隔为其他值时，均扣一分。

优化目标 2 思路：将最终出车序列编号 1,2,3,…,318，并根据每一个车身的配置不同（两驱和四驱）编号 2 或 4，其中 2 代表两驱车型，4 代表四驱车型。针对出车序列，按照题中要求的规则进行分块，分块示例如图所示，每一块都有奇数个车身或者偶数个车身组成，含奇数个车身的组块不符合规则，需要扣分，如图块①；含偶数个车身的组块中，只有可以在中间将块分割成两个同类型的块才可以不扣分，如图中的块②，否则要扣分。

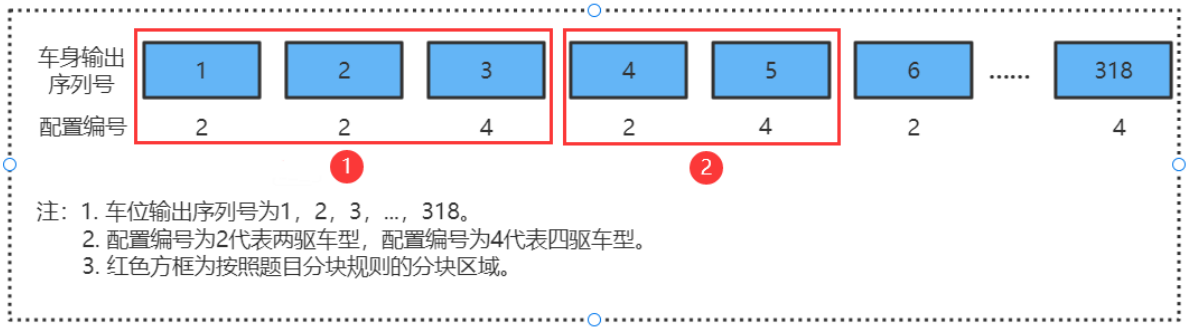


图 3-5　优化目标 2 建模图例

3.2 问题 2 分析

问题 2 与问题 1 相似，属于同一类问题，问题 2 仅仅在问题 1 的基础上减少了 6、7 两个约束变量，而其他约束不变，车道和横移机优先规则不变，优化目标也没有改变。这里依然与问题 1 一样，采用多目优化的方法，建立多目标调序优化线性规划模型，利用四个优化目标对总评分进行寻优。最后生成以最优目标函数，即最优得分。对应的车身生产序列为总装生产计划的车身序列。问题 2 的决策变量和中间变量设置形式依然需要与问题 1 保持一致，因为问题 2 的求解方法和求解模型基本一致。但是在问题 2 的结果分析中需要与问题 1 进行比较说明，评估约束 6、7 的重要程度，其对于整个车身生产序列调序的影响程度和影响结果。

4. 模型的建立与求解

4.1 数学模型的建立

$$Max\ W = 0.4W_1 + 0.3W_2 + 0.2W_3 + 0.1W_4 \tag{4-1}$$

$$W_1 = 100 - \sum_k d_k \tag{4-2}$$

$$W_2 = 100 - \sum_{n=1}^{318} \lambda_{2n-1} - \sum_{n=1}^{318} \lambda_{2n} + \eta \tag{4-3}$$

$$W_3 = 100 - \sum_{m\in M} \sum_{t\in T} x_{71t}^m \tag{4-4}$$

$$W_4 = 100 - 0.01\times\left(t_{\max} - 9C - 72\right) \tag{4-5}$$

$$0 \le \sum_{m\in M} \sum_{i\in I} x_{it}^m \le B\left(1-z\right) \qquad t\in N,\ B\text{ 为无穷大量} \tag{4-6}$$

$$t_{\max} = zt \qquad t\in N \tag{4-7}$$

$$y_{(i,j)}^{m\ t,t+1} = x_{it}^m \times x_{j\ t+1}^m \tag{4-8}$$

$$y_{(710,2)}^{m\ t,t+1} = 0 \qquad \forall m\in M, t\in T \tag{4-9}$$

$$y_{(i,j)}^{m\ t,t+1} = 0 \qquad (i,j)\notin OD,\ \forall m\in M, t\in T \tag{4-10}$$

$$0 \le \sum_{t=t+1}^{t+8} y_{(i,j)}^{m\ \tau,\tau+1} \le B\left(1-y_{(i,j)}^{m\ t,t+1}\right) \qquad \forall m\in M, t\in T,\ B\text{为无穷大量} \tag{4-11}$$

$$\sum_{m\in M} x_{1t}^m \le 1 \qquad \forall t\in T \tag{4-12}$$

$$\sum_{m\in M} x_{2t}^m \le 1 \qquad \forall t\in T \tag{4-13}$$

$$Z_{k,k+1}^{m\ 4,2} = 1 \tag{4-14}$$

$$Z_{0.5\lambda_n,0.5\lambda_n+1}^{m\ 2,4} = 1 \tag{4-15}$$

$$\sum_{m\in M} x_{it}^m \le 1 \tag{4-16}$$

$$zone\ A_t = [t - \left(0.5S_i \times y_{(2,3)}^{m\ \ t+0.5s_i,t+0.5s_i+1} + 0.5R_i \times y_{(2,71)}^{m\ \ t+0.5R_i,t+0.5R_i+1}\right) \times y_{(i,2)}^{m\ \ t,t+1},$$
$$t + \left(0.5S_i \times y_{(2,3)}^{m\ \ t+0.5s_i,t+0.5s_i+1} + 0.5r_i \times y_{(2,71)}^{m\ \ t+0.5R_i,t+0.5R_i+1}\right) \times y_{(i,2)}^{m\ \ t,t+1}] \quad \forall t \in T, i \in \{51,61\} \tag{4-17}$$

$$zoneA_t = [t - \left(0.5S_i \times y_{(2,3)}^{m\ \ t+0.5s_i,t+0.5s_i+1} - 0.5(R_i - 6) \times y_{(2,71)}^{m\ \ t+0.5(R_i+6),t+0.5(R_i+6)_i+1}\right) \times y_{(i,2)}^{m\ \ t,t+1}, \quad \forall t \in T,$$
$$t + \left(0.5S_i \times y_{(2,3)}^{m\ \ t+0.5s_i,t+0.5s_i+1} + 0.5(R_i + 6) \times y_{(2,71)}^{m\ \ t+0.5(R_i+6)_i,t+0.5(R_i+6)_i+1}\right) \times y_{(i,2)}^{m\ \ t,t+1}] \quad i \in \{11,21,31,41\} \tag{4-18}$$

$$StartA_{t+1} - EndA_t \geq 0 \qquad \forall t \in T \tag{4-19}$$

$$zone\ B_t = [t - \left(0.5S_i \times y_{(0,1)}^{m\ \ t-0.5s_i,t-0.5s_i+1} + 0.5R_i \times y_{(710,1)}^{m\ \ t-0.5(R_i-6),t-0.5(R_i-6)+1}\right),$$
$$t + \left(0.5S_i \times y_{(0,1)}^{m\ \ t-0.5s_i,t-0.5s_i+1} + 0.5R_i \times y_{(710,1)}^{m\ \ t+0.5(R_i-6),t+0.5(R_i-6)+1}\right)] \quad \begin{matrix} \forall t \in T \\ i \in \{510,610\} \end{matrix} \tag{4-20}$$

$$zone\ B_t = [t - \left(0.5S_i \times y_{(0,1)}^{m\ \ t-0.5s_i,t-0.5s_i+1} + 0.5(R_i + 6) \times y_{(710,1)}^{m\ \ t-0.5R_i,t-0.5R_i+1}\right), \quad \forall t \in T$$
$$t + \left(0.5S_i \times y_{(0,1)}^{m\ \ t-0.5s_i,t-0.5s_i+1} + 0.5(R_i - 6)_i \times y_{(710,1)}^{m\ \ t-0.5R_i,t-0.5R_i+1}\right)] \quad i \in \{110,210,310,410\} \tag{4-21}$$

$$StartB_{t+1} - EndB_t \geq 0 \qquad \forall t \in T \tag{4-22}$$

$$y_{(710,1)}^{m\ \ t+R_{710},t+R_{710}+1} = \sum_{m \in M} x_{710,t}^m \times \left(1 - \sum_{m \in M} x_{1t}^m\right) \qquad \forall t \in T \tag{4-23}$$

$$a = \min\left(S_i \times \sum_{m \in M} x_{it}^m\right) \qquad \forall t \in T, i \in I' \tag{4-24}$$

$$1 - \sum_{m \in M} x_{2t}^m \leq y_{(a,2)}^{m\ \ t+0.5R_a,t+0.5R_a+1} \leq 1 \qquad \forall t \in T, i \in I' \tag{4-25}$$

$$x_{it}^m \times \left(1 - x_{jt}^m\right) \leq \sum_{\tau=t+1}^{t+9} y_{(i,j)}^{m\ \ \tau,\tau+1} \leq 1 \qquad \begin{matrix} \forall t \in T, m \in M \\ (i,j) \in OD_l \end{matrix} \tag{4-26}$$

$$Out_t = \sum_{t \in T} x_{3,t}^m \qquad \forall t \in T \tag{4-27}$$

$$time_m = \sum_{t \in T} \left(x_{3,t}^m \times t\right) \qquad \forall m \in M \tag{4-28}$$

$$order_m = \sum_{t=1}^{time_m} Out_t \qquad \forall m \in M \tag{4-29}$$

$$H = \{(1,h_1),(2,h_2),...,(318,h_{318})\} \tag{4-30}$$

$$E = H \setminus \{h = 0 | H\}, \quad (order_k, h_k) = e_k \in E \qquad k \in N \tag{4-31}$$

$$d_k=\begin{cases}0,\ \text{当} order_{k+1}-order_k=3\\ 1,\ \text{其他}\end{cases} \qquad k\in N \qquad (4\text{-}32)$$

式(4-1)为问题的目标函数，目标函数为车型排序、出车序列、返回道使用次数、总调度时间四个目标得分的加权平均值。

式(4-2)为目标一：按照混动车型间隔 2 台非混动车型为优规则的得分。

式(4-3)为目标二：按照安排四驱车型与两驱车型倾向 1:1 出车序列规则的得分。

式(4-4)为目标三：按照返回道使用次数倾向于 0 规则的得分。

式(4-5)表示目标四：按照总调度时间越短越好规则的得分。

式(4-6)和式(4-7)表示当所有的位置都没有车身时，时间t达到最大值$t_{\max}$。

式(4-8)说明了中间变量$y_{(i,\ j)}^{m\ t,t+1}$的含义，若车身m在t时刻由位置i转到位置j，则$y_{(i,\ j)}^{m\ t,t+1}=1$。

式(4-9)表示 PBS 约束 1：送车横移机不能将返回道的车身送入 PBS-总装接车口。

式(4-10)表示 PBS 约束 2：车身在t到$t+1$时刻的位移只能由位置i移动到固定的下一位置j。

式(4-11)表示 PBS 车身移动过程：任意车身在进车道/返回道中，从某一停车位移动至后一停车位，消耗时间为 9 秒。

式(4-12)和式(4-13)表示 PBS 约束 3：接车横移机和送车横移机上同一时刻分别最多有一个车身。

(4-14)表示由附件 1 可以确定出车序列第一个位置是两驱车型，分块规则确定为 2→4，为相邻两个模块之间的转移过程，每个模块首尾位置分别为两驱车型和四驱车型。

(4-15)表示当分块区间长度λ_n为偶数时，$\frac{\lambda_n}{2}$位置处相邻两模块转移过程为 4→2 时，不扣分。

式(4-16)表示每个停车位每个时刻最多有一个车身。

式(4-17) 、(4-18) 、(4-19) 、(4-20) 、(4-21) 、(4-22)表示 PBS 约束 4 和 PBS 约束 5。约束 4：接车横移机和送车横移机在完成任意动作后，必须返回中间初始位置，才可以执行下一步动作。约束 5：接车横移机和送车横移机在执行任何动作过程中，均不能被打断。其中式(4-17) 、(4-18) 、(4-20) 、(4-21)的含义为：若t时刻送车横移机、接车横移机有动作，则动作进行的时间区间分别为$zoneA_t$、$zoneB_t$。式(4-19)与(4-22)则说明了每两个相邻动作的时间区间不能重合。六个约束结合便能实现 PBS 约束 4 和 PBS 约束 5。

式(4-23)表示 PBS 约束 6：当返回道 10 停车位有车身，同时接车横移机空闲时，优先处理返回道 10 停车位上的车身。当$\sum_{m\in M}x_{1t}^{m}=0$，结合 PBS 约束 5、6，可保证$t$时刻接车横移机空闲。

式(4-24)定义了变量a。进车道 1 停车位有车身的车位中，横移机能最先到达的那个车位。

式(4-25)表示 PBS 约束 7：当若干进车道 1 停车位有车身等候，同时送车横移机空闲时，优先处理最先到达 1 停车位的车身。同时满足 PBS 约束 8：当任意进车道 1 停车位有车身时，送车横移机不能空闲。式(4-25)表明当若干进车道 1 停车位有车身等候时，横移机在执行动作需要的时间内必须完成一次动作。

式(4-26)表示 PBS 约束 10 和 11。约束 10：同一车道内，多个车身在不同停车位上的移动可以不同步进行。约束 11：当某车身所在停车位的下一停车位出现空位时，车身必须立即开始向下一停车位移动。同一车道内，多个车身在不同停车位上的移动可以不同步进

行。式(4-26)说明当前一个位置 i 在某时刻存在车辆，且下个位置 j 不存在车辆时，9 秒内车辆必移动一次。当前一个位置 i 在某时刻存在车辆，且下个位置 j 也存在车辆时，9 秒内车辆移动与否都可。当前一个位置 i 在某时刻不存在车辆，无论下个位置 j 是否存在车辆，9 秒内位置 i 和 j 之间都不会出现移动车辆的动作。

式(4-27)和式(4-28)分别定义了中间变量 Out_t 、$time_m$ 的定义。Out_t 是 0-1 变量，Out_t =1 表示 t 时刻有车到达总装-PBS 接车口。$time_m$ 表示车 m 到达总装-PBS 接车口时间。

式(4-29)说明了中间变量 $order_m$ 的定义：车 m 到达总装-PBS 接车口的顺序序号。式(4-29)的字面意思是 $order_m$ 代表时间 $time_m$ 时刻，累计到达总装-PBS 接车口的车辆数。而 $time_m$ 为车 m 到达总装-PBS 接车口时间。因此 $order_m$ 即为车 m 到达总装-PBS 接车口的顺序序号。

通过式(4-27)、(4-28)、(4-29)可以得到每个车辆 m 的编号，即车辆 m 与序号 $order_m$ 的一一对应关系。

式(4-30)列出了集合 H 包含的元素。$(k,h_k)\in H$，其中 k 为车的序号，h_k 则为序号 k 对应的车辆的混动型号。每辆车是否为混动在附件 1、附件 2 中均已给出。

式(4-31)定义了集合 E。E 为集合 H 去掉非混动元素后剩下的元素，即混动元素。

式(4-32)说明了二元变量 d_k 的含义。当 $order_{k+1}-order_k=3$ 时，d_k =0，机当两个混动车辆的序号相差 3 时，不扣分。其他情况都扣 1 分。结合式(4-2)即可实现优化目标 1。

4.2 问题 1 的求解

表 4-1　改进遗传算法参数含义

参数符号	对应含义
$size\_pop$	染色体种群数目
max_iter	最大迭代次数
$size\_pop$	染色体种群数目
max_iter	最大迭代次数
p_select	选择概率阙值
$prob\_mut\_single$	单点变异概率
$prob\_mut\_reverse$	逆转变异概率
$prob\_crossover$	交叉概率
$numberAdjust$	调整道数目
$Objectvie1\_value$	个体的目标函数 1 的值
$Objectvie2\_value$	个体的目标函数 2 的值
$Input\_matrix$	输入矩阵（包括车辆颜色编码和对应的总装序号）
$Output\_matrix$	输出矩阵（包括总装序号在涂装车间的排列，以及对应的进入缓冲区对应的调整道编号）

表 4-2 改进遗传算法伪代码

名称：	基于重排序的改进遗传代码
输入	待排产序列 线性缓冲区调整道数目 待排产序列中车辆对应的涂装颜色信息
输出	上游车间的计划排产序列 排产序列对应的线性缓冲区入站调整道编号
1：	参数初始化
2：	种群初始化
3：	序列重排序
4：	计算所有序列对应的优化目标 1 的值
5：	计算所有序列对应的优化目标 2 的值
6：	计算所有序列对应的优化目标 3 的值
7：	计算所有序列对应的优化目标 4 的值
8：	If 任一序列的优化目标 1 取值为 0
9：	将种群序列按照优化目标函数 1 的加权值升序排列，计算每个序列被选择的概率
10：	Else 根据优化目标函数 1 的值计算相对适应度，然后计算每个序列被选择的概率
11：	End
12：	进行选择操作
13：	将两个序列完成交叉操作
14：	将两个序列完成变异操作
15：	将两个序列完成逆转变异操作得到子代序列
16：	If 子代序列数目小于初始群体数目
17：	重复 12、13、14、15 步骤
18：	End
19：	If 终止条件不满足
20：	重复步骤 3 至 18
21：	End

（1）数据处理

对附件中两套数据内提供的所有信息进行 0，1 编码，其中，车型 A 为 0，B 为 1；动力燃油为 0，混动为 1；驱动两驱为 0，四驱为 1。

（2）求解过程

本文采用遗传算法求解。遗传算法（GA）是一种受生物进化启发的学习方法，它不再是从一般到特殊或从简单到复杂地搜索假设，而是通过变异和重组当前已知的最好假设来生成后续的假设。

（3）遗传算法流程

模型输入：本模型以涂装车间出车序列的选择车道为变量，即对于 318 个车身，每个车身选择的车道，所以一共有 318 个变量，每个变量的取值范围为{0，1，2，3，4，5}。最后的输入为长度为 318 的序列，例如[3, 2, 4, 3, 0, 2, …]。

模型输出：上述输入经过模型之后，得到模型对该序列的评分，即由题目中四个优化

目标量化所确定的得分。

由于计算硬件资源有限，确定输入种群数为 4，迭代次数为 4，变异概率为 0.2，首先输入序列初始化，计算初始序列的种群适应度，根据适应度的大小，通过选择、交叉、变异的方式进行更新，然后重新计算种群适应度，以此循环，直到达到迭代次数。迭代结果如图 4-2 所示。其中上方的图，横坐标表示迭代次数，纵坐标表示每个个体的适应度值；下方的图，横坐标表示迭代次数，纵坐标表示种群中适应度最高的个体的适应度值。最后，输出适应度最高的序列个体，并输入模型中，输出最终的目标分数，以及每个车身在每个时刻的位置列表，行表示时间，列表示每个车身，其中的值表示车身位置，-1 表示车身未进入 PBS 缓冲区，或者车身已出 PBS 缓冲区。

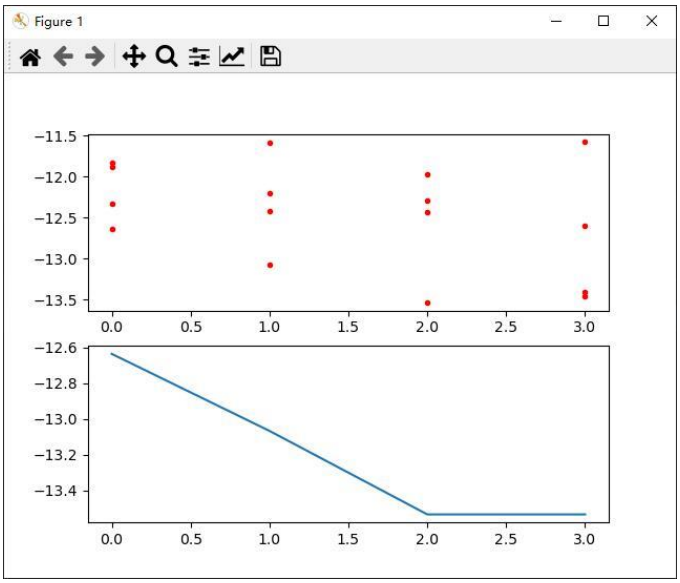


图 4-1　问题 1 迭代结果图

4.3 问题 2 的求解

通过分析，问题 2 与问题 1 的求解过程基本一致，只是少了题中 6 和 7 的约束条件，故只需修改模型中的约束条件，即删去约束 6 和 7。

另外，算法流程与问题 1 一致。种群数目为 2，迭代次数为 2，变异概率为 0.2。其余过程不变，在此不多赘述。最后迭代结果图如图 4-2 所示。

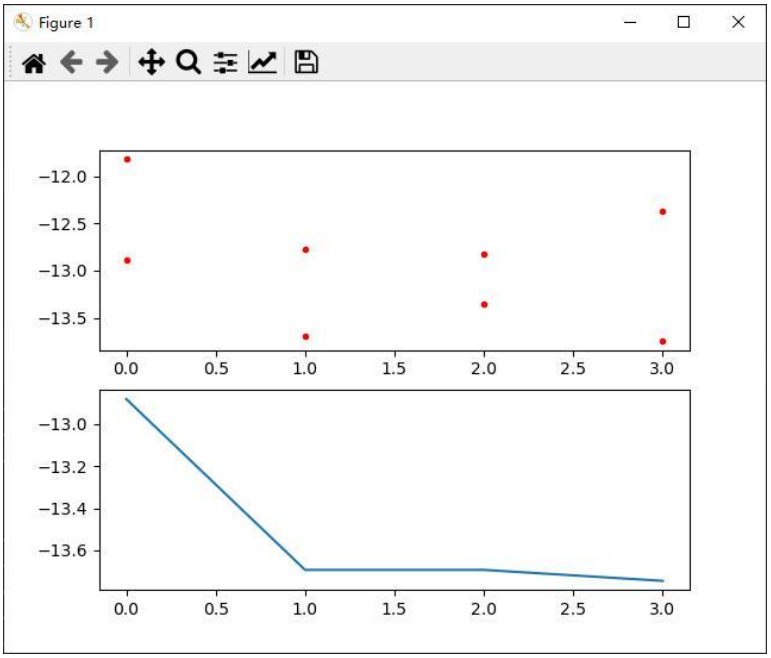


图 4-2　问题 2 迭代结果图

5. 结果分析

	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
269	-1	-1	-1	-1	2	22	63	65	17	39	610	0	-1	-1	-1	-1
270	-1	-1	-1	-1	2	21	63	65	17	38	610	0	-1	-1	-1	-1
271	-1	-1	-1	-1	2	21	63	65	17	38	610	0	-1	-1	-1	-1
272	-1	-1	-1	-1	3	21	63	65	17	38	610	0	-1	-1	-1	-1
273	-1	-1	-1	-1	-1	21	63	65	17	38	69	0	-1	-1	-1	-1
274	-1	-1	-1	-1	-1	21	63	65	17	38	69	410	-1	-1	-1	-1
275	-1	-1	-1	-1	-1	21	63	65	17	38	69	410	1	-1	-1	-1
276	-1	-1	-1	-1	-1	21	62	64	16	38	69	410	1	0	-1	-1
277	-1	-1	-1	-1	-1	21	62	64	16	38	69	410	1	0	-1	-1
278	-1	-1	-1	-1	-1	2	62	64	16	38	69	410	1	0	-1	-1
279	-1	-1	-1	-1	-1	2	62	64	16	37	69	410	1	0	-1	-1
280	-1	-1	-1	-1	-1	2	62	64	16	37	69	410	1	0	-1	-1
281	-1	-1	-1	-1	-1	2	62	64	16	37	69	410	1	0	-1	-1
282	-1	-1	-1	-1	-1	2	62	64	16	37	68	49	1	0	-1	-1
283	-1	-1	-1	-1	-1	2	62	64	16	37	68	49	610	0	-1	-1
284	-1	-1	-1	-1	-1	3	62	64	16	37	68	49	610	0	-1	-1
285	-1	-1	-1	-1	-1	-1	61	63	15	37	68	49	610	0	-1	-1
286	-1	-1	-1	-1	-1	-1	61	63	15	37	68	49	610	0	-1	-1
287	-1	-1	-1	-1	-1	-1	61	63	15	37	68	49	610	0	-1	-1
288	-1	-1	-1	-1	-1	-1	61	63	15	36	68	49	610	0	-1	-1
289	-1	-1	-1	-1	-1	-1	61	63	15	36	68	49	610	0	-1	-1
290	-1	-1	-1	-1	-1	-1	61	63	15	36	68	49	610	0	-1	-1
291	-1	-1	-1	-1	-1	-1	61	63	15	36	67	48	610	0	-1	-1
292	-1	-1	-1	-1	-1	-1	61	63	15	36	67	48	69	0	-1	-1
293	-1	-1	-1	-1	-1	-1	61	63	15	36	67	48	69	1	-1	-1
294	-1	-1	-1	-1	-1	-1	2	62	14	36	67	48	69	1	0	-1
295	-1	-1	-1	-1	-1	-1	2	62	14	36	67	48	69	1	0	-1
296	-1	-1	-1	-1	-1	-1	2	62	14	36	67	48	69	1	0	-1
297	-1	-1	-1	-1	-1	-1	2	62	14	35	67	48	69	1	0	-1
298	-1	-1	-1	-1	-1	-1	2	62	14	35	67	48	69	210	0	-1

图 5-1　部分车身位置列表

由问题一的建模和算法求解分析可知，不同的车道选择序列，经过模型后得到的优化目标分数不同，并且随着迭代次数增加，分数逐渐增大（图中取的是分数的负数，故逐渐减小）。由于种群数目和迭代次数的限制，最终结果并不是最优解或者较优解，但已经能够看到分数逐渐增大的趋势。当种群数目和迭代次数足够大时，结果应该会更好。

由问题二的建模和算法求解分析，与问题一的结果比较，可以发现，问题二的最终结果要好一些，我们认为，当去除约束 6 和约束 7 时，模型的限制减少，通过模型后的出车序列更满足四项目标要求，所以最终分数更高。

6. 模型评价

6.1 模型优点

1. 通过查阅相关文献和对赛题背景的分析，本文构建了一个多目标整数规划模型。和单目标规划模型相比，一方面，该多目标整数规划模型能够针对离散化问题进行精准刻画和描述，可以对移动物体在某一时刻所处的静态状态和移动的动态过程进行结合；另一方面，该多目标整数规划模型可以在全局的角度针对不同权重的目标对整体的调度方案进行优化，能够为汽车生产企业提供高效快捷的调度优化策略和优化方案。

2. 通过对已知数据和未知变量的提前合理分类处理有利于建立模型，本文对所有停车位按照通道分成 7 个集合，将位移分成在通道停车位上的移动和没有在通道上的停车位上的移动两个集合等。在没有对数据分类处理的情况下建立模型是非常困难的，很难用数学表达式描述所有的目标和约束条件，求解的时候也会很复杂。

6.2 模型不足

1. 本文建立的模型适用于本问题，变量维数比较高，通用性和普适性有待提高，对于复杂度较高的问题在描述和求解上相对困难。

2. 本文建立的模型参数较多，可读性有所欠缺。

参考文献

[1]韩建明. 混合型汽车装配线的重排序方法研究[D].东南大学,2015.
[2]胡婷婷. PBS 缓冲区车身路由调度策略的研究与应用[D].合肥工业大学,2017.
[3]陈正茂. 基于排序缓冲区的多车间关联排序研究[D]. 华中科技大学, 2008.
[4]Muhl E, Charpentier P, Chaxel F. Optimization of physical flows in an automotive manufacturing plant: some experiments and issues. Engineering Applications of Artificial Intelligence, 2003, 16(4): 293-305.

附录

```python
import numpy as np
from dataload import dl
import random
from openpyxl import Workbook  # 导入库


class Car:
    def __init__(self, idx, cx, dl, qd):
        self.index = idx
        self.cx = cx
        self.dl = dl
        self.qd = qd


class Line:
    def __init__(self, idx):
        self.index = idx
        self.position()
        self.cx = -1
        self.dl = -1
        self.qd = -1

    # 生成车位
    def position(self):
        self.P = []
        for i in range(10):  # 位置0-9
            p = Position(10*self.index+i)
            self.P.append(p)

    # 更新
    def update(self):
        for i in range(1, 10):
            # idx = 10 * self.index + i
            self.P[i].update(self.P[i-1].state, self.P[i-1].qs)

    # def update1(self):
    #     for i in range(9):
    #         # idx = 10 * self.index + i
    #         self.P[i].update1(self.P[i+1].cd, self.p[i+1].state)

```

```
class Linef:
    def __init__(self):
        self.index = 7
        self.position()
        self.cx = -1
        # self.dl = -1
        # self.qd = -1

    def position(self):
        self.P = []
        for i in range(10):  # 位置0-9
            p = Posf(10*self.index+i)
            self.P.append(p)

    def update(self):
        for i in range(9):
            self.P[i].update(self.P[i+1].state, self.P[i+1].qs)


class Position:
    def __init__(self, idx):
        self.index = idx
        self.cd = 0
        self.state = 0
        self.qs = 0
        self.cx = -1
        self.dl = -1
        self.qd = -1

    def update(self, state, qs):
        if self.state > 0:
            if state > 0:
                if qs == 1:
                    if self.cd == 0:
                        self.qs = 1
                        self.cd = 9
            else:
                if self.cd == 0:
                    self.qs = 1
                    self.cd = 9


class Posf:
    def __init__(self, idx):
        self.index = idx
        self.cx = -1
        self.dl = -1
        self.qd = -1
        self.cd = 0
        self.state = 0
        self.qs = 0

    def update(self, state, qs):
        if self.state > 0:
            if state > 0:
                if qs == 1:
                    if self.cd == 0:
                        self.qs = 1
                        self.cd = 9
            else:
                if self.cd == 0:
                    self.qs = 1
                    self.cd = 9


class AGV1:
    def __init__(self, idx):
        self.index = idx
        self.cx = -1
        self.dl = -1
        self.qd = -1
        self.cost = [18, 12, 6, 0, 12, 18]
        self.fetch_cost = [24, 18, 12, 6, 12, 18]
        self.cd = 0
        self.state = 0
        self.memory = 0
        self.start = 0
```

```
        # self.remember = 0
        self.fetch_time = 0

    def send_to(self, end):    # end表示去哪个车道
        self.end = end
        self.cd = self.cost[end]  # 更新cd
        self.memory = self.cd

    def send_to1(self):
        self.end = 3


    def fetch(self, end):
        self.fetch_time = 3
        self.end = end
        self.cd = self.fetch_cost[end]

    # def unload(self):
    # def update(self):
    #     if self.cd


class AGV2:
    def __init__(self, idx):
        self.index = idx
        self.cx = -1
        self.dl = -1
        self.qd = -1
        self.qc = 0
        self.cost = [18, 12, 6, 0, 12, 18]
        self.fetch_cost = [24, 18, 12, 6, 12, 18]
        self.cd = 0
        self.state = 0
        self.memory = 0
        self.start = -1
        self.fetch_time = 0

    def send_to(self, start):  # start表示从哪个车道取车,num表示车号
        self.start = start
```

```
        # self.state = num
        self.cd = self.cost[start]  # 更新cd
        self.memory = self.cd

    def send_to1(self):
        self.start = 3
        # self.state = num

    def fetch(self, start):
        self.start = start
        if start < 3:
            self.fetch_time = (3 - start) * 3
        elif start > 3:
            self.fetch_time = (start - 2) * 3
        self.cd = self.fetch_cost[start]
        self.qc = 1


    # def fetch(self):
    #     self
    # def update(self):
    #     self.state = 0


class Env:
    def __init__(self, car_num, l_num, rank):
        self.car_num = car_num
        self.p_num = 74
        self.l_num = l_num
        self.create_item()
        self.t = 0
        self.rank = list(rank)

    def create_item(self):
        self.lines = []
        for i in range(1, self.l_num+1):  # 车道1-6
            l = Line(i)
            self.lines.append(l)
        self.linef = Linef()
```

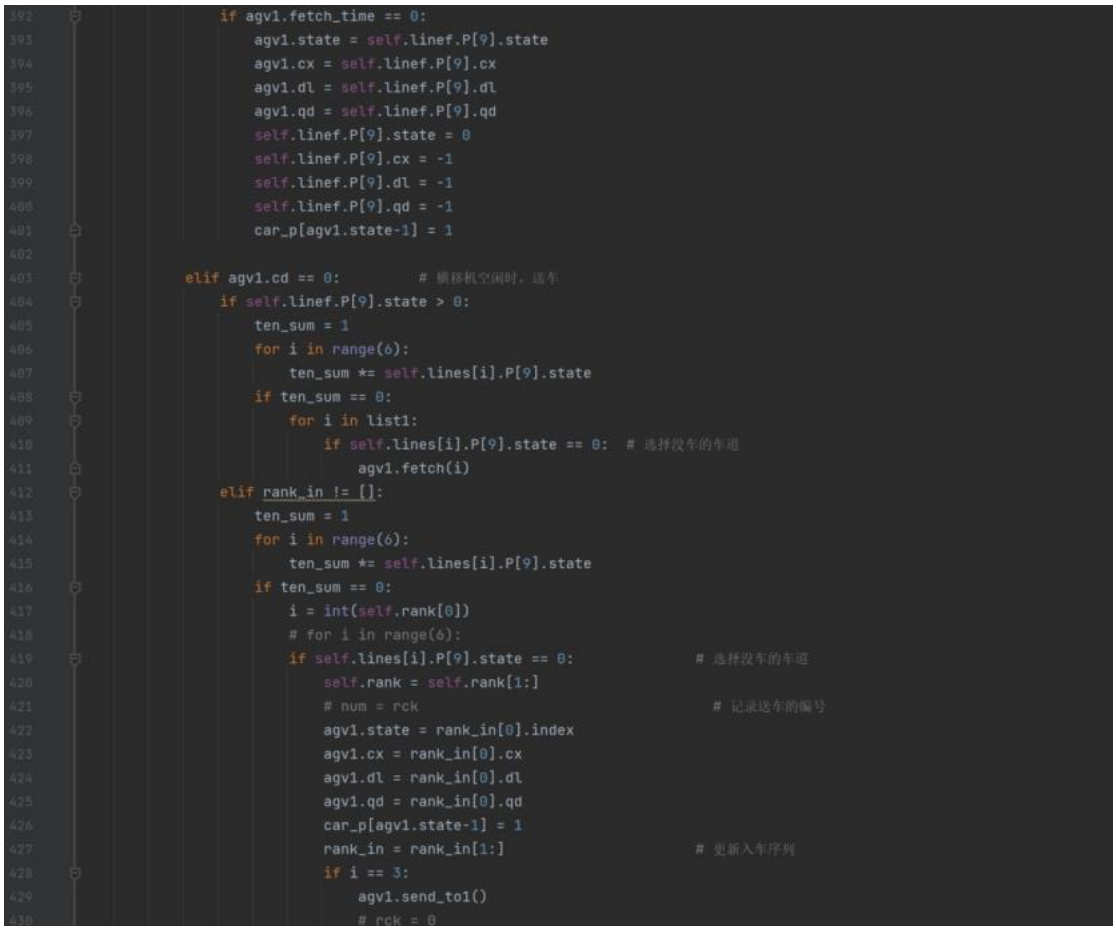
```
        self.cars = []
        filePath = '1.csv'
        with open(filePath) as f:
            data = np.loadtxt(filePath, str, delimiter=',', skiprows=1)
        data = dl(data)
        for i in range(data.shape[0]):
            xl = data[i]
            car = Car(xl[0], xl[1], xl[2], xl[3])
            self.cars.append(car)
        self.agv1 = AGV1(1)
        self.agv2 = AGV2(2)  # 两个横移机
        self.rck = 0
        self.cck = 0  # 入车口和出车口

    # def update_item(self):
    #     self.agv1.update()

    def step(self):
        wb = Workbook()  # 创建一个xlsx文件
        ws = wb.active  # 激活工作表
        flag = 0
        ret = 0
        count = 0  # 统计出车数
        p_sum = 0
        rck = self.rck
        cck = self.cck
        rank_out_car = np.zeros([self.car_num, 4])
        for m in range(len(self.lines)):
            for n in range(10):
                p_sum += self.lines[m].P[n].state
        for n in range(10):
            p_sum += self.linef.P[n].state
        p_sum = p_sum + rck + cck + self.agv1.state + self.agv2.state
        rank_in = self.cars
        agv1 = self.agv1
        agv2 = self.agv2
        rank_out = []
        result = []
        car_p = np.zeros(self.car_num, int)
```

```
        # car_p = list(car_p)
        for i in range(self.car_num):
            car_p[i] = -1
        # test = np.ones(318)
        # lines = self.lines

        while rank_in != [] or p_sum != 0:
            for i in range(self.car_num):
                if car_p[i] == 3:
                    car_p[i] = -1
            self.t += 1
            cck = 0
            list1 = [3, 2, 4, 1, 5, 0]
            if rck == 0 and rank_in != []:
                rck = rank_in[0].index
                car_p[rck-1] = 0
            if agv2.fetch_time > 0:
                agv2.fetch_time -= 1
                if agv2.fetch_time == 0:  # 把车返回的路上取车
                    k = agv2.start
                    agv2.state = self.lines[k].P[0].state
                    agv2.cx = self.lines[k].P[0].cx
                    agv2.dl = self.lines[k].P[0].dl
                    agv2.qd = self.lines[k].P[0].qd
                    self.lines[k].P[0].state = 0
                    self.lines[k].P[0].cx = -1
                    self.lines[k].P[0].dl = -1
                    self.lines[k].P[0].qd = -1
                    car_p[agv2.state - 1] = 2

            if agv2.cd == 0:  # 横移车空闲时
                one_sum = 0
                for i in range(6):
                    one_sum += self.lines[i].P[0].state
                if one_sum > 0:  # 判断车道末尾是否有车
                    for i in list1:
                    # for i in range(6):
                        if self.lines[i].P[0].state > 0:       # 选择有车的车道
                            num = self.lines[i].P[0].state          # 记录出车的编号
```

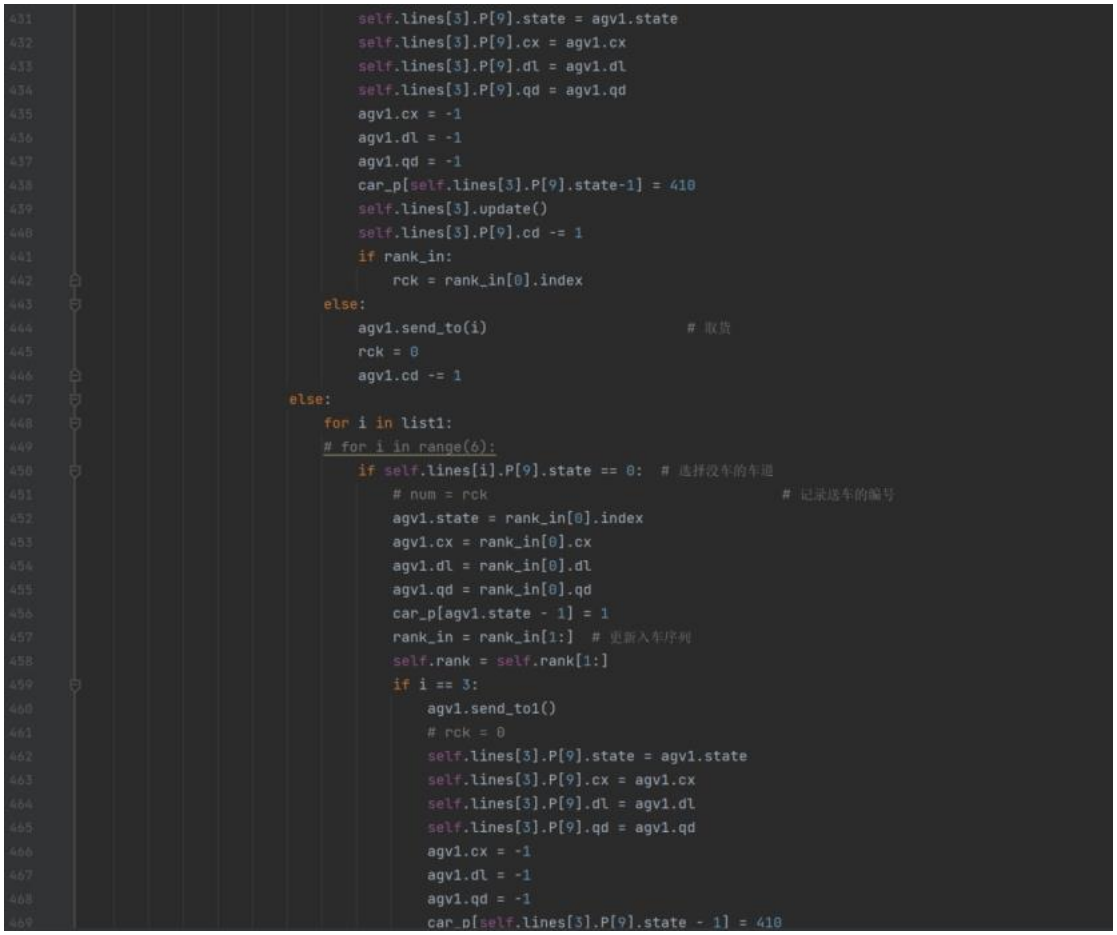
```
                        # rank_out.append(num)                    # 更新出车序列
                        if flag == 1:
                            ret += 1
                            agv2.fetch(i)
                            if i == 3:
                                agv2.state = self.lines[i].P[0].state
                                agv2.cx = self.lines[i].P[0].cx
                                agv2.dl = self.lines[i].P[0].dl
                                agv2.qd = self.lines[i].P[0].qd
                                self.lines[i].P[0].state = 0
                                self.lines[i].P[0].cx = -1
                                self.lines[i].P[0].dl = -1
                                self.lines[i].P[0].qd = -1
                                car_p[agv2.state] = 2
                        if i == 3:
                            agv2.send_to1()                         # 出车过程
                            cck = num
                            rank_out_car[count][0] = self.lines[i].P[0].state
                            rank_out_car[count][1] = self.lines[i].P[0].cx
                            rank_out_car[count][2] = self.lines[i].P[0].dl
                            rank_out_car[count][3] = self.lines[i].P[0].qd
                            self.lines[i].P[0].state = 0
                            self.lines[i].P[0].cx = -1
                            self.lines[i].P[0].dl = -1
                            self.lines[i].P[0].qd = -1
                            car_p[int(num)-1] = 3
                            count += 1
                        else:
                            agv2.send_to(i)                         # 出车过程
                            agv2.cd -= 1
                            break
                # elif agv2.cd == agv2.memory/2:
                #     k = agv2.start
                #     agv2.state = self.lines[k].P[0].state
                #     self.lines[k].P[0].state = 0
                #     agv2.cd -= 1

                else:
                    agv2.cd -= 1
```

```
            if agv2.cd == 3:  # 把车放到返回车道上
                if agv2.qc == 1:
                    self.linef.P[0].state = self.agv2.state  # 卸货
                    self.linef.P[0].cx = self.agv2.cx
                    self.linef.P[0].dl = self.agv2.dl
                    self.linef.P[0].qd = self.agv2.qd
                    self.agv2.state = 0
                    self.agv2.cx = -1
                    self.agv2.dl = -1
                    self.agv2.qd = -1
                    car_p[self.linef.P[0].state - 1] = 71
                    agv2.qc = 0
            if agv2.cd == 0:
                cck = agv2.state
                rank_out_car[count][0] = cck
                rank_out_car[count][1] = agv2.cx
                rank_out_car[count][2] = agv2.dl
                rank_out_car[count][3] = agv2.qd
                agv2.state = 0
                agv2.cx = -1
                agv2.dl = -1
                agv2.qd = -1
                car_p[cck-1] = 3
                count += 1
            if agv2.cd == agv2.memory / 2:    # 走到一半上货
                k = agv2.start
                agv2.state = self.lines[k].P[0].state
                agv2.cx = self.lines[k].P[0].cx
                agv2.dl = self.lines[k].P[0].dl
                agv2.qd = self.lines[k].P[0].qd
                self.lines[k].P[0].state = 0
                self.lines[k].P[0].cx = -1
                self.lines[k].P[0].dl = -1
                self.lines[k].P[0].qd = -1
                car_p[agv2.state-1] = 2
                agv2.memory = -1

        # 更新每个车道位置的状态和cd
        for i in range(6):
```

```
            self.lines[i].update()
            # self.lines[i].update1()
            for n in range(10):
                if self.lines[i].P[n].cd > 0:
                    self.lines[i].P[n].cd -= 1
                    if self.lines[i].P[n].cd == 0:
                        self.lines[i].P[n-1].state = self.lines[i].P[n].state
                        self.lines[i].P[n-1].cx = self.lines[i].P[n].cx
                        self.lines[i].P[n-1].dl = self.lines[i].P[n].dl
                        self.lines[i].P[n-1].qd = self.lines[i].P[n].qd
                        self.lines[i].P[n].state = 0
                        self.lines[i].P[n].cx = -1
                        self.lines[i].P[n].dl = -1
                        self.lines[i].P[n].qd = -1
                        self.lines[i].P[n].qs = 0
                        k = self.lines[i].P[n-1].index
                        car_p[self.lines[i].P[n-1].state-1] = k + 1
        for n in range(10):
            if self.linef.P[n].cd > 0:
                self.linef.P[n].cd -= 1
                if self.linef.P[n].cd == 0:
                    self.linef.P[n+1].state = self.linef.P[n].state
                    self.linef.P[n+1].cx = self.linef.P[n].cx
                    self.linef.P[n+1].dl = self.linef.P[n].dl
                    self.linef.P[n+1].qd = self.linef.P[n].qd
                    self.linef.P[n].cx = -1
                    self.linef.P[n].dl = -1
                    self.linef.P[n].qd = -1
                    self.linef.P[n].state = 0
                    self.linef.P[n].qs = 0
                    k = self.linef.P[n + 1].index
                    if n < 9:
                        car_p[self.linef.P[n + 1].state-1] = k + 1
                    else:
                        car_p[self.linef.P[n + 1].state-1] = k % 10 + 1 + k // 10 * 100

        if agv1.fetch_time > 0:
            agv1.cd -= 1
            agv1.fetch_time -= 1
```



```
            if agv1.fetch_time == 0:
                agv1.state = self.linef.P[9].state
                agv1.cx = self.linef.P[9].cx
                agv1.dl = self.linef.P[9].dl
                agv1.qd = self.linef.P[9].qd
                self.linef.P[9].state = 0
                self.linef.P[9].cx = -1
                self.linef.P[9].dl = -1
                self.linef.P[9].qd = -1
                car_p[agv1.state-1] = 1

        elif agv1.cd == 0:          # 横移机空闲时，送车
            if self.linef.P[9].state > 0:
                ten_sum = 1
                for i in range(6):
                    ten_sum *= self.lines[i].P[9].state
                if ten_sum == 0:
                    for i in list1:
                        if self.lines[i].P[9].state == 0:  # 选择没车的车道
                            agv1.fetch(i)
            elif rank_in != []:
                ten_sum = 1
                for i in range(6):
                    ten_sum *= self.lines[i].P[9].state
                if ten_sum == 0:
                    i = int(self.rank[0])
                    # for i in range(6):
                    if self.lines[i].P[9].state == 0:              # 选择没车的车道
                        self.rank = self.rank[1:]
                        # num = rck                                  # 记录送车的编号
                        agv1.state = rank_in[0].index
                        agv1.cx = rank_in[0].cx
                        agv1.dl = rank_in[0].dl
                        agv1.qd = rank_in[0].qd
                        car_p[agv1.state-1] = 1
                        rank_in = rank_in[1:]                      # 更新入车序列
                        if i == 3:
                            agv1.send_to1()
                            # rck = 0
```



```
                        self.lines[3].P[9].state = agv1.state
                        self.lines[3].P[9].cx = agv1.cx
                        self.lines[3].P[9].dl = agv1.dl
                        self.lines[3].P[9].qd = agv1.qd
                        agv1.cx = -1
                        agv1.dl = -1
                        agv1.qd = -1
                        car_p[self.lines[3].P[9].state-1] = 410
                        self.lines[3].update()
                        self.lines[3].P[9].cd -= 1
                        if rank_in:
                            rck = rank_in[0].index
                    else:
                        agv1.send_to(i)                        # 取货
                        rck = 0
                        agv1.cd -= 1
                else:
                    for i in list1:
                    # for i in range(6):
                        if self.lines[i].P[9].state == 0:  # 选择没车的车道
                            # num = rck                                          # 记录送车的编号
                            agv1.state = rank_in[0].index
                            agv1.cx = rank_in[0].cx
                            agv1.dl = rank_in[0].dl
                            agv1.qd = rank_in[0].qd
                            car_p[agv1.state - 1] = 1
                            rank_in = rank_in[1:]  # 更新入车序列
                            self.rank = self.rank[1:]
                            if i == 3:
                                agv1.send_to1()
                                # rck = 0
                                self.lines[3].P[9].state = agv1.state
                                self.lines[3].P[9].cx = agv1.cx
                                self.lines[3].P[9].dl = agv1.dl
                                self.lines[3].P[9].qd = agv1.qd
                                agv1.cx = -1
                                agv1.dl = -1
                                agv1.qd = -1
                                car_p[self.lines[3].P[9].state - 1] = 410
```

```
                                    self.lines[3].update()
                                    self.lines[3].P[9].cd -= 1
                                    if rank_in:
                                        rck = rank_in[0].index
                                else:
                                    agv1.send_to(i)  # 取货
                                    rck = 0
                                    agv1.cd -= 1
                                break
            else:
                agv1.cd -= 1                                        # 其它时间cd-1
                if agv1.cd == agv1.memory / 2:
                    k = agv1.end
                    self.lines[k].P[9].state = self.agv1.state  # 卸货
                    self.lines[k].P[9].cx = self.agv1.cx
                    self.lines[k].P[9].dl = self.agv1.dl
                    self.lines[k].P[9].qd = self.agv1.qd
                    self.agv1.state = 0
                    self.agv1.cx = -1
                    self.agv1.dl = -1
                    self.agv1.qd = -1
                    self.lines[k].P[9].cd = 9
                    car_p[self.lines[k].P[9].state-1] = (k+1)*100+10
                    agv1.memory = -1

                p_sum = 0
                for m in range(len(self.lines)):
                    for n in range(10):
                        p_sum += self.lines[m].P[n].state
                for n in range(10):
                    p_sum += self.linef.P[n].state
                p_sum = p_sum + rck + cck + agv1.state + agv2.state

                # print(self.t, agv1.state, agv1.end, agv2.state, agv2.start, rck, cck, p_sum)
                # result.append(car_p)
                ws.append(list(car_p.T))
                # print(car_p)

        dl_rank = rank_out_car[..., 2]
```

```
        qd_rank = rank_out_car[..., 3]
        m = -1
        n = -1
        cost1 = 0
        for i in range(len(dl_rank)):
            if dl_rank[i] == 1:
                if m == -1:
                    n = i
                    m = n
                else:
                    m = n
                    n = i
                    if n - m != 3:
                        cost1 += 1
        sum1 = 100 - cost1
        # 计算目标2的值
        cost2 = 0
        m = 0
        first = qd_rank[0]
        if first == 1:
            mark = -1
        else:
            mark = 1
        for i in range(len(qd_rank) - 1):
            if qd_rank[i] - qd_rank[i + 1] == mark:
                x, y = 0, 0
                for j in range(m, i + 1):
                    if qd_rank[j] == 1:
                        x += 1
                    else:
                        y += 1
                if x != y:
                    cost2 += 1
                m = i + 1
        x, y = 0, 0
        for j in range(m, len(qd_rank)):
            if qd_rank[j] == 1:
                x += 1
            else:
```

```
                y += 1
            if x != y:
                cost2 += 1
            sum2 = 100 - cost2
            # 计算目标3的值
            sum3 = 100 - ret
            # 计算目标4的值
            best = 9*self.car_num + 72
            cost4 = 0.01*(self.t - best)
            sum4 = 100-cost4
            res_score = 0.4*sum1 + 0.3*sum2 + 0.2*sum3 + 0.1*sum4
            # wb.save('result11.xlsx')
            return rank_in, rank_out, rank_out_car, wb, res_score


def start(y):
    env = Env(318, 6, y)
    in_list, out_list, cars, result, score = env.step()
    return score


def test(x):
    env_test = Env(318, 6, x)
    in_list, out_list, cars, wb, score = env_test.step()
    return wb, score


# if __name__ == "__main__":
#     rank_num = 0
#     env1 = Env(318, 6, rank_num)
#     # for i in range(10):
#     #     print(env.lines[0].P[i].index)
#     filePath = '1.csv'
#     with open(filePath) as f:
#         data = np.loadtxt(filePath, str, delimiter=',', skiprows=1)
#     data = dl(data)
#     # print(data)
#     in_list, out_list, cars, result, score = env1.step()
#     # print(result)
```

```
#     a = np.asarray(result)
#     print(a.shape)
#     np.savez('my_npz_file.npz', result=a)
#     nums = []
#     cx = []
#     dl = []
#     qd = []
#     # for i in range(318):
#     #     nums.append(cars[i][0])
#     #     cx.append(cars[i][1])
#     #     cx.append(cars[i][1])
#     #     cx.append(cars[i][1])
#
#     # a.tofile('sample.csv', sep=',')
#
#     # print(in_list, len(out_list), out_list, score)
#     print(score)
#
#     # for i in range(318):
#     #     print(cars[i])

```

```python
import numpy as np
from Flow import start
from Flow import test


def schaffer(p):
    '''
    This function has plenty of local minimum, with strong shocks
    global minimum at (0,0) with value 0
    '''
    print(p)
    return -start(p)


# %%
from sko.GA import GA
a = list(np.zeros(318))
b = list(np.ones(318)*5)
ga = GA(func=schaffer, n_dim=318, size_pop=2, max_iter=4, prob_mut=0.2, lb=a, ub=b, precision=1)
best_x, best_y = ga.run()
print('best_x:', best_x, '\n', 'best_y:', best_y)

# %% Plot the result
import pandas as pd
import matplotlib.pyplot as plt

Y_history = pd.DataFrame(ga.all_history_Y)
fig, ax = plt.subplots(2, 1)
ax[0].plot(Y_history.index, Y_history.values, '.', color='red')
Y_history.min(axis=1).cummin().plot(kind='line')
plt.show()
result, test_re = test(best_x)
print(test_re)
result.save('result11.xlsx')
# print(a.shape)
# a = a.T
# a.tofile('sample.csv', sep=',')


# wb = Workbook()  # 创建一个xlsx文件
```

```python
# ws = wb.active  # 激活工作表
# ws.append(a)
# wb.save('result11.xlsx')
```