



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	中国计量大学
--------	--------

参赛队号	22103560098
------	-------------

	1.何健郡
队员姓名	2.张滕成
	3.周美琪

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 基于动态规划的 PBS 多目标优化调度问题研究

摘 要：

工业领域中，利用缓存区调序是优化工业流程，节省工业成本的重要手段。随着新能源技术的发展，汽车工业在生产中的地位逐渐提高，而汽车制造涂装-总装缓存调序作为汽车制造流程中极为重要的一部分，其调序区调度优化问题有着重要的研究意义，本文通过分析车辆调度工作流程建立了**基于动态规划的多目标优化调度模型**。

针对问题一，通过分析车辆的调度流程以及相关约束条件，以**每辆车被接车横移机将要运往的目标车道以及该车是否要通过返回道**作为决策变量，建立每一时刻进车道与返回道上车位车辆的时间状态矩阵，以第一辆车被接车横移机处理为初始时刻，利用接车横移机和送车横移机工作时间矩阵，模拟出每一辆车在随时间变化的位置以及状态。由于接车横移机优先处理返回道 10 号位上的车辆，送车横移机优先处理先到 1 号位的车辆，则可通过返回道使用次数对车辆的序列进行迭代改变。对**优化目标一（ z_1 ）**，利用卷积的思想处理，即设置一个卷积核，将其公式化；对**优化目标二（ z_2 ）**，建立 $z_2 = F(L)$ ，其主要通过既定的算法计算得分；对**优化目标三（ z_3 ）**，将每辆车是否使用返回道统计即可得到；对**优化目标四（ z_4 ）**，通过进车道与返车道上每个车位的时间矩阵，利用时间变量，对整个车道调度过程进行递推模拟，将车辆的输出序列进行迭代，最终求得整个调度流程所消耗的时间，以此建立**基于动态规划的多目标优化调度模型**。

考虑到接车横移机与送车横移机工作时选择不同车道所消耗的时间差，利用**灰狼算法**设置迭代次数对模型进行求解，得到附件 1 最高得分为 18.916 分，其调度流程总用时 3056 秒，返回道总共使用 6 次，附件 2 最高得分为 44.469 分，其调度流程总用时 2996s，返回道总共使用 6 次。

针对问题二，取消约束 6 后，接车横移机不再优先处理返车道 10 号位上的车，引入新的决策变量，即**返回道上每辆车到达 10 号位后的等待时间**，接车横移机每次工作时，首先判断返回道 10 号位上是否有车辆，当返回道上 10 号位有车时，接车横移机到达返回道上的时刻需大于返回道上 10 号位上车的等待时间，才会处理返回道上的车，否则处理涂装-PBS 出车口处的车辆。取消约束 7 后，送车横移机将不再优先处理先到达 1 号位的车辆，即在处理完上一辆车后需对当前时刻所有车道 1 号位进行判断，如果 1 号位上的车辆大于 1，则需进行选择，此时将位于 1 号位的每一辆汽车分别模拟当其送出或送往返回道以后输出序列 L ，然后求得当前情况每一辆车的得分，对得分进行排序，重新规定他们的输出顺序。以问题一所建立的模型为基础，添加上述新的决策变量和约束条件建立**基于动态规划的多目标优化调度模型**。

利用灰狼算法设置迭代次数对模型进行求解，得到附件 1 最高得分为 19.459 分，其调度流程总用时 3056 秒，返回道总共使用 8 次；附件 2 最高得分为 45.78 分，其调度流程总用时 3014 秒，返回道总共使用 4 次。与模型一求解结果进行比较，其收敛速度更快，

结果更好。

综上，本文通过分析车道调度流程，利用了较少的决策变量，建立了基于动态规划的多目标优化调度模型，较好的拟合了车道调度流程，利用所给数据计算出最后得分，并通过与理论最优解进行比较，对模型进行了评价。

关键词：动态规划；PBS 时间调度；多目标优化模型；灰狼算法

目 录

一、问题重述.....	4
1.1 问题的背景	4
1.2 问题的提出	4
二、模型假设.....	5
三、符号说明.....	6
四、问题一.....	8
4.1 考虑横移机调度优先级的 PBS 优化问题分析.....	8
4.2 考虑横移机调度优先级的 PBS 优化模型的建立.....	8
4.3 考虑横移机调度优先级的 PBS 优化问题算法设计.....	21
4.4 考虑横移机调度优先级的 PBS 优化问题求解结果.....	23
五、问题二.....	25
5.1 不考虑横移机调度优先级的 PBS 优化问题的分析.....	25
5.2 不考虑横移机调度优先级的 PBS 优化模型建立.....	25
5.3 不考虑横移机调度优先级的 PBS 优化问题算法设计.....	29
5.4 不考虑横移机调度优先级的 PBS 优化模型求解结果.....	29
六、模型评价与推广.....	31
6.1 模型的优点.....	31
6.2 模型的缺点.....	31
6.3 模型的推广.....	31
七、参考文献.....	31
八、附录.....	32

一、问题重述

1.1 问题的背景

一辆汽车在正式上线之前需要经过多道工序，主要由汽车制造厂的焊装车间、涂装车间、总装车间完成。但是，由于每个车间有不同的生产约束，以致生产调度无法按照同一序列连续生产，通常在相邻两个车间之间和车间内部必要的地方设置缓存区，这大大提高了工厂的生产效率。本文着重考虑在涂装车间与总装车间之间建立一个具有调序功能的缓存区，用来存放涂装车间的已完成喷涂的车身，将该缓存区称为 PBS (Painted Body Store)，在该缓存区内调整涂装车间的出车序列以满足总装车间约束的进车序列。

1.2 问题的提出

已知涂装车间的出车序列，要求严格按照 PBS 的约束说明和相关时间数据说明，建立 PBS 优化调度模型，调整总装车间的进车序列，使其尽可能满足生产需求。

针对问题一，需要考虑的优化目标如下，以权重系数从高到低排列：

- 1) 每两辆混动车身之间间隔两辆燃油车身；
- 2) 对出车序列进行分块，每一分块中的四驱车型与两驱车型之比为 1:1；
- 3) 尽可能不使用返回车道；
- 4) 总调度时间尽可能短。

需要考虑的约束条件如下：

- 1) 送车横移机不能运送返回车道上的车身；
- 2) 车身在 PBS 中的移动方向固定不变；
- 3) 接车、送车横移机单次只能运送一个车身；
- 4) 接车、送车横移机完成任意动作后，必须返回中间的起始位置，才可进行下一步动作；
- 5) 接车、送车横移机在执行任何动作的过程中，不可以被打断；
- 6) 当返回道的 10 停车位有车且接车横移机空闲时，接车横移机必须处理 10 停车位上的车身；
- 7) 当若干进车道的 1 停车位有车身且送车横移机空闲时，优先处理最先到达进车道 1 停车位的车身；
- 8) 只要进车道的 1 停车位有车身时，送车横移机就必须为工作状态；
- 9) 进车道和返回道各 10 个停车位，且每个停车位只能容纳 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个停车位移动；
- 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。

针对问题二，在问题一的基础上，去除第 6、7 条约束，建立 PBS 优化调度模型。

二、模型假设

假设 1：假设汽车车型对总装车间的进车序列没有影响；

假设 2：假设接车、送车横移机在装载车身到进车道 10 停车位或返回道 1 停车位的时间忽略不计；

假设 3：假设车身从涂装-PBS 出车口处装载到接车横移机上，以及在 4 车道出口中间位置处，车身由送车横移机上卸载到 PBS-总装接车口，时间均忽略不计；

假设 4：横移机运动时的速度保持一致；

假设 5：假设在整个 PBS 过程中，接车、送车横移机等装置不会出现故障；

假设 6：车辆在车道上从前一车位移动到后一车位的 9s 内始终处于前一车位，到第 9s 时处于后一车位；

三、符号说明

符号	解释说明
ts_1	接车横移机从初始位置把车辆运送到进车道 10 停车位上所消耗的时间矩阵
ts_2	接车横移机从进车道返回中间初始位置所消耗的时间矩阵
ts_1'	送车横移机从初始位置到进车道 1 停车位所消耗的时间矩阵
ts_2'	送车横移机将车辆从进车道 1 停车位运送至总装-PBS 口消耗的时间矩阵
tfd	接车横移机从涂装-PBS 口出发将车身装载运送到进车道后返回初始位置所消耗的时间矩阵
$tfd1$	接车横移机从涂装-PBS 口出发将返回道 10 停车位上的车身运到各进车道 10 停车位所消耗的时间矩阵
tf	送车横移机从总装-PBS 口出发至进车道 1 停车位将车辆运载至返回道 1 停车位后返回初始位置所消耗的时间矩阵
tf_1	送车横移机从总装-PBS 出发至进车道 1 停车位将车辆运载至返回道 1 停车位所消耗的时间矩阵
tf_2	送车横移机从返回道回到初始位置所消耗的时间矩阵
v	车辆使用返回道的次数
$i(v)$	接车横移机从返回道取出 v 次后第 i 辆车
$T_{i(v)}$	第 $i(v)$ 辆车从涂装-PBS 口的出发时刻
$J_{i(v)}$	第 $i(v)$ 辆车属性的矩阵
$H_{i(v)}$	为 0-1 变量, 第 $i(v)$ 辆车进入目标车道的矩阵
$h_{v,k}$	为 0-1 变量, 表示返回道 10 停车位上的车辆被送至目标车道的矩阵
$Td_{i(v)}$	车辆从涂装-PBS 口被送入进车道 10 停车位上所消耗的时间
$TD_{i(v)}$	接车横移机第 v 次处理返回道车辆后, 第 i 辆车出发前的等待时间
$TDS_{i(v)}$	第 $i(v)$ 辆车的目标进车道 10 停车位有车时, 该车在出发前的等待时间
$t_{i(v)}$	接车横移机将第 $i(v)$ 辆汽车送入进车道 10 停车位的时刻
$Tc_{i(v)}$	第 $i(v)$ 辆车被接车横移装载的时刻
$x_{i(v),k}$	0-1 变量, 表示第 $i(v)$ 辆车的目标车道 10 停车位是否有车
$td_{i(v)}$	车辆放入进车道 10 停车位的时刻
$Tct_{i(v),j}$	第 $i(v)$ 辆车经过返回道迭代后, 处于 j 停车位的时间矩阵
$Tend_{i(v)}$	第 $i(v)$ 辆车从 1 停车位到达总装-PBS 口的时刻
L_i	输出序列各车辆的属性
$Te_{i(v)}$	第 $i(v)$ 辆车从 1 号停车位到总装-PBS 口消耗的时间
$D_{k,j,t}$	t 时刻, 第 k 车道的第 j 个位置是第几辆车
$tjs_{i(v+1)}$	处于返回道 1 停车位上的第 $i(v+1)$ 辆车, 开始被处理的时刻
y_t	当前所有进车道上的总车辆数
$d_{k,j,t}$	0-1 变量, 表示当前时刻, 车道上所有车的位置
$dw_{i(v)}$	第 $i(v)$ 辆车在每个停车位上的时间等待矩阵
$t_{i(v)}$	接车横移机将第 $i(v)$ 辆汽车送入进车道的时刻
$Tc_{i(v)}$	第 $i(v)$ 辆车被接车横移装载的时刻
$tsw_{i(v)}$	当车辆位于 1 停车位时, 等待被处理的时间
$S_{j,t}$	判断 t 时刻, 返回道上的每一个停车位是否有车的矩阵

$sw_{i(v)}$	第 $i(v)$ 辆车在每个车位上的时间等待矩阵
$L_{d,i(v)}$	整个出车序列中所有车辆的动力类型
$L_{q,i(v)}$	整个出车序列中所有车辆的驱动类型
num	位于 1 号位的车辆总数
$Tend_{n(v)}$	序列的最后一辆车 n 结束送车的时刻

四、问题一

4.1 考虑横移机调度优先级的 PBS 优化问题分析

首先，对车辆在 PBS 流程中的运动轨迹进行分析，车辆运作流程如图 4.1 所示。

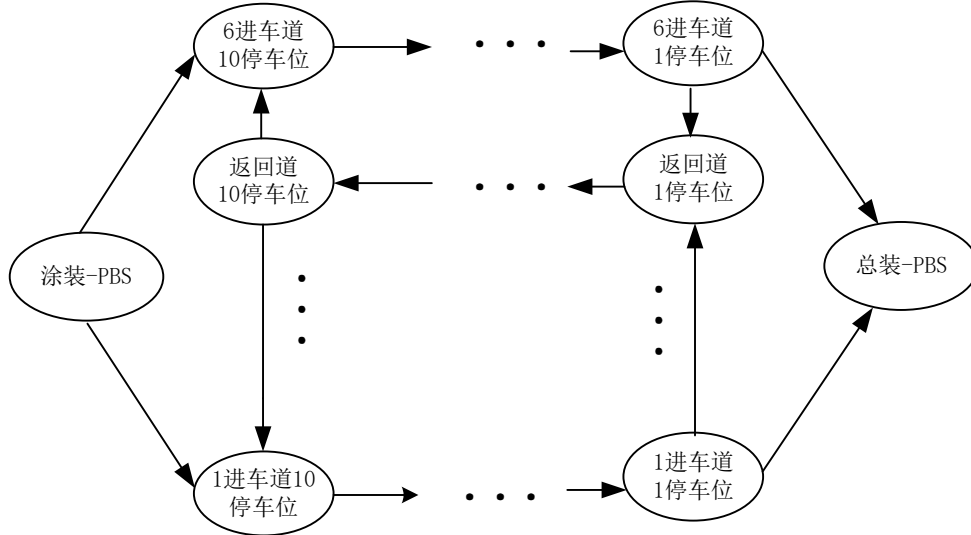


图 4.1 PBS 运作流程图

其次，根据 PBS 相关时间数据分析，可得接车横移机从涂装-PBS 口出发，将车运送到不同进车道上消耗的时间差最大为 9s。所以，只要当车辆被运送到任意进车道后，在不使用返回道的情况下，其送车顺序将与接车顺序一致，而每使用返回道一次，就会改变一次出车顺序，即当某一辆车被送车横移机决定送到返回道上时，整个车辆的出车顺序将会被改变，接车横移机从返回道中取车总共取了 v 次，则整个出车顺序 $I(v) = \{1, 2, 3, \dots, i(v), \dots, n\}$ 就改变了 v 次，而这种改变并非瞬间改变，而是当汽车决定被送车横移机移动到返回道上开始改变，汽车移动到返回道上的 10 号位后，被接车横移机插入队列完成改变。

最后，从优化目标出发，对附件中的生产数据进行分析，考虑满足单目标条件下的扣分情况，以此得到一个参考得分。以附件 1 为例，混动车身与燃油车身的数量之比为 2:1，四驱与两驱车型的数量之比近似 10:1。从第一个目标出发，即 n 个混动车身需要 $(n-1)*2$ 个燃油车身，则有 54 个混动车身可以满足要求，剩余 158 个不满足要求，依据量化逻辑进行扣分，需要扣 $158 \times 1 \times 0.4 = 63.2$ 分，在完全满足其他目标时，可得 36.8 分。从第二个目标出发，即四驱与两驱车型的出车序列为 1:1，则有 29 个两驱车型满足要求，剩余 260 个不满足要求，依据量化逻辑进行扣分，需要扣 $260 \times 1 \times 0.3 = 78$ 分，在完全满足其他目标时，可得 22 分。

4.2 考虑横移机调度优先级的 PBS 优化模型的建立

决策变量：

通过对车辆 PBS 调度流程的分析，可知该问题的优化核心在于车辆的出车序列，它主要受接车横移机在涂装-PBS 进车口选择的车道 $H_{i(v),k}$ 、是否使用返回道 $r_{i(v)}$ 的影响，因

此，将它们设为该问题的决策变量。

其中，

$$H_{i(v),k} = \begin{cases} 0, & \text{第 } i \text{ 辆车不进入 } k \text{ 进车道} \\ 1, & \text{第 } i \text{ 辆车进入 } k \text{ 进车道} \end{cases}, k = 1,2,3,4,5,6 \quad (1)$$

$$r_{i(v)} = \begin{cases} 1, & \text{第 } i \text{ 辆车返回} \\ 0, & \text{第 } i \text{ 辆车不返回} \end{cases} \quad (2)$$

对 PBS 相关时间数据进行处理：

首先，由于接车横移机在运作过程中的速度保持一致，且忽略将车身从接车横移机装载和卸载的时间，那么就可以通过接车横移机将车身从涂装-PBS 口运到各进车道 10 停车位后返回初始位置消耗的时间，得到接车横移机将车放入各进车道所消耗的时间（用列矩阵 ts_1 表示）和接车横移机从进车道 10 停车位返回涂装-PBS 所消耗的时间（用列矩阵 ts_2 表示），即

$$ts_1 = ts_2 = [9,6,3,0,6,9]^T \quad (3)$$

同理可得，送车横移机从总装-PBS 口出发到各进车道 1 停车位消耗的时间 ts_1' 以及从该位置返回到总装-PBS 口消耗的时间 ts_2' 。

接车横移机从涂装-PBS 口出发到返回道 10 停车位将车身装载运送到进车道后返回初始位置的运作流程，如图 4.2 所示。

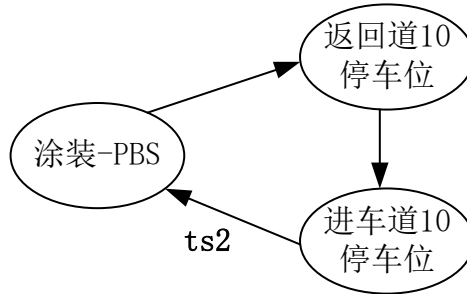


图 4.2 接车横移机在返回道、进车道、涂装-PBS 口的运作流程

已知：接车横移机从涂装-PBS 口出发到返回道 10 停车位将车身装载运送到进车道后返回初始位置所消耗的时间，用列矩阵 tfd 表示，即

$$tfd = [24,18,12,6,12,18]^T \quad (4)$$

可以求得：接车横移机从涂装-PBS 出发将返回道 10 停车位上的车身运到各进车道 10 停车位所消耗的时间，用矩阵 $tfd1$ 表示，即

$$tfd1 = tfd - ts2 = [15,12,9,6,6,9]^T \quad (5)$$

由于送车横移机从总装-PBS 口到 4 进车道 1 停车位的时间忽略不计，所以可以通过送车横移机在返回道、4 进车道、总装-PBS 口的运作流程，得到送车横移机从返回道 1 停车位至总装-PBS 口消耗的时间，该运作流程如图 4.3 所示。

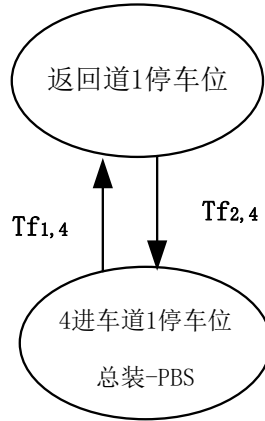


图 4.3 送车横移机在返回道、4 进车道、总装-PBS 口的运作流程

已知：

$$tf_{1,4} + tf_{2,4} = 6 \quad (6)$$

$$tf_{1,4} = tf_{2,4} \quad (7)$$

可以求得：

$$tf_2 = [3,3,3,3,3]^T \quad (8)$$

同理可得，接车横移机至返回道所消耗的时间 tf_2' 。

接车横移机从涂装-PBS 口出发到返回道 10 停车位将车身装载运送到进车道后返回初始位置的运动流程，如图 4.4 所示。

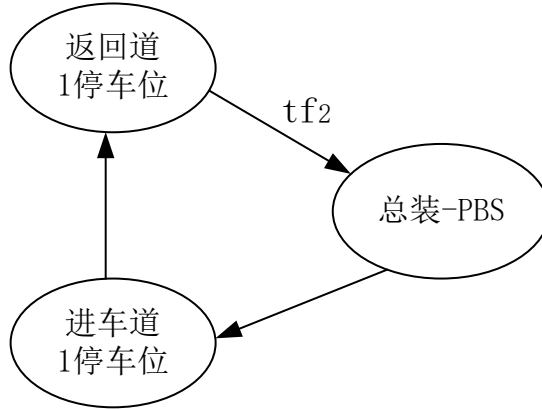


图 4.4 送车横移机在返回道、进车道、总装-PBS 口的运作流程

已知：送车横移机从总装-PBS 口出发至进车道 1 停车位将车辆运载至返回道 1 停车位后返回初始位置所消耗的时间，用列矩阵 tf 表示，即

$$tf = [24,18,12,6,12,18]^T \quad (9)$$

可以求得：送车横移机从总装-PBS 出发至进车道 1 停车位将车辆运载至返回道 1 停车位所消耗的时间，用矩阵 tf_1 表示，即

$$tf_1 = tf - tf_2 = [21,15,9,3,9,15]^T \quad (10)$$

车辆经 PBS 调度的流程：

根据约束条件 3，每辆车只会被接车横移机运往其中某一进车道，即

$$\sum_k^6 H_{i(v),k} = 1 \quad (11)$$

(1) 汽车出发时刻处理:

考虑到所有汽车需按照顺序从涂装-PBS 出车口开始准备出发, 这里从第一辆汽车出发时开始计时, 此时返回道并未被使用, 因此, 第一辆汽车的在涂装-PBS 出车口时刻准备出发时刻可表示为:

$$T_{1(0)} = 0 \quad (12)$$

在接车横移机运送第一辆车至进车道后返回中间初始位置, 准备运送第二辆车时, 第一辆车不会到达进车道 1 停车位, 即返回道仍不会被使用, 所以, 第二辆汽车的出发时刻应当为:

$$T_{2(0)} = T_{1(0)} + H_{1(0)k} \cdot (ts_1 + ts_2) \quad (13)$$

以此类推, 当扩展到第 $i(v)$ 辆车时, 有两种情况:

a) 返回道 10 停车位没有车, 则

$$T_{i(v)} = T_{i-1(v)} + H_{i-1(v),k} \cdot (ts_1 + ts_2) \quad (14)$$

其中, $H_{i-1(v),k}$ 表示将前一辆车需要运送到哪一进车道。

b) 返回道 10 停车位上有车, 则接车横移机在处理完上一辆车后, 需要立刻处理返回道上的车, 当返回道上的车全部处理完后, 再运送涂装-PBS 口处的车。那么, 在计算此时涂装-PBS 口出第 z 辆车的出发时刻时, 则需加上处理返回道上车的时间。即:

$$T_{i(v)} = T_{i-1(v)} + H_{i-1(v),k} \cdot (ts_1 + ts_2) + h_{v,k} \cdot (tfd_1 + tfd_2) \quad (15)$$

其中, h_v 表示接车横移机将返回道 10 停车位的车运送至哪一进车道, tfd_1 表示接车横移机把车辆从返回道运送至对应车道所消耗时间的矩阵。表示从进车道返回中间初始位置所消耗的时间矩阵。

当 $i(v)$ 辆车经过返回道第 v 次动作后, 已经彻底完成迭代, 重新插入队列中。此时, 该车变为第 $i(v+1)$ 辆车, 即

$$J_{i(v+1)} = J_{i(v)} \quad (16)$$

而该车的等待时间为:

$$T_{i(v+1)} = T_{i-1(v)} + H_{i-1(v),k} \cdot (ts_1 + ts_2) + h_v \cdot tf_2' \quad (17)$$

与此同时, 涂装-PBS 口的第 $i(v)$ 辆车变为第 $i+1(v+1)$ 辆车, 则

$$\begin{aligned} T_{i+1(v+1)} &= T_{i(v)} = T_{i-1(v)} + H_{i-1(v),k} \cdot (ts_1 + ts_2) + h_v \cdot (tfd_1 + tfd_2) \\ J_{i+1(v+1)} &= J_{i(v)} \end{aligned} \quad (19)$$

其中, $J_{i(v)}$ 表示第 i 辆车的属性。

$$J_{i(v)} = [j_d, j_q] \quad (20)$$

$$j_d = \begin{cases} 0, & \text{该车为混动车型} \\ 1, & \text{该车为燃油车型} \end{cases} \quad (21)$$

$$j_q = \begin{cases} 0, & \text{该车为四驱车型} \\ 1, & \text{该车为两驱车型} \end{cases} \quad (22)$$

(2) 车辆从开始时刻到进车道上的处理:

当汽车在涂装-PBS 出车口开始准备出发后, 由于接车横移机可以选择等待也可以选择直接开始工作, 因此处理时间由两部分组成, 分别是接车横移机等待时间和接车横移机将车辆从出车口运送到进车道 10 号位的时间。即:

$$Tz_{i(v)} = TD_{i(v)} + TDS_{i(v)} + Td_{i(v)} \quad (23)$$

用 $TD_{i(v)}$ 表示第 i 辆车在接车横移机从返回道接车第 v 次后, 第 i 辆车出发前的等待时间, 当其为 0 时表示接车横移机直接开始工作接车, $TDS_{i(v)}$ 表示的是第 $i(v)$ 辆车目标车道有车, 接车横移机为了将第 $i(v)$ 辆车送往目标车道需等待的时间, $Td_{i(v)}$ 指的是第 $i(v)$ 辆车被运往进车道的的时间。

接车过程中, 考虑到接车横移机对车道的选择问题, 将 $t_{i-1(v)}$ 表示接车横移机将第 $i-1$ 辆汽车送入进车道上的时间, 其可表示为:

$$t_{i-1(v)} = T_{i-1(v)} + TD_{i-1(v)} + Tc_{i(v)} \quad (24)$$

此时, 第 $i(v)$ 辆汽车将要被接车横移机接车的时刻为

$$tc_{i(v)} = t_{i-1(v)} + TD_{i(v)} \quad (25)$$

第 $i(v)$ 辆车将要进入的车道 10 号位是否有车可表示为

$$x_{i,k} = \left\lfloor \frac{D_{k,10,tc_{i(v)}}}{D_{k,10,tc_{i(v)}}+1} \right\rfloor = \begin{cases} 1, & \text{目标进车道 10 停车位有车} \\ 0, & \text{目标进车道 10 停车位没有车} \end{cases} \quad (26)$$

则如果要选择有车的那个车道, 需等待的时间矩阵为

$$TDS_{i(v)} = 9 - H_{i(v),k} \cdot \left\lfloor \frac{D_{k,10,tc_{i(v)}}}{D_{k,10,tc_{i(v)}}+1} \right\rfloor \cdot ts_1 \quad (27)$$

而横移机将车辆运送到进车道上的时间为

$$Td_{i(v)} = H_{i(v),k} \cdot ts_1 \quad (28)$$

(3) 车辆在进车道上的处理:

当汽车放入进车道时, 此时的时刻为

$$td_i = T_{i(v)} + Tz_{i(v)} \quad (29)$$

则

$$H_{i(v),k} \cdot D_{k,10,td_i} = i(v) \quad (30)$$

考虑到当车辆进入进车道后, 其到达 1 停车位的序列已经固定, 因此, 不考虑后续补充车辆, 直接利用车道行进规则, 对 $D_{k,10,td_i}$ 进行迭代, 得到第 $i(v)$ 辆车在每个车位上的

时间矩阵 $Tct_{i(v),j}$ 。

(4) 车辆由送车横移机处理：

首先，考虑返回道是否会堵车，即无法将进车道 1 停车位上的车运至返回道的情形。由于接车横移机会优先处理返回道上的车辆，且将车辆从进车道 1 停车位运送至返回道消耗时间最短的进车道为 4 进车道，所以只有在送车横移机连续将 4 进车道的两辆车送至返回道时，返回道 1 停车位上才可能会有车。该过程的运作流程，如图 4.5 所示。

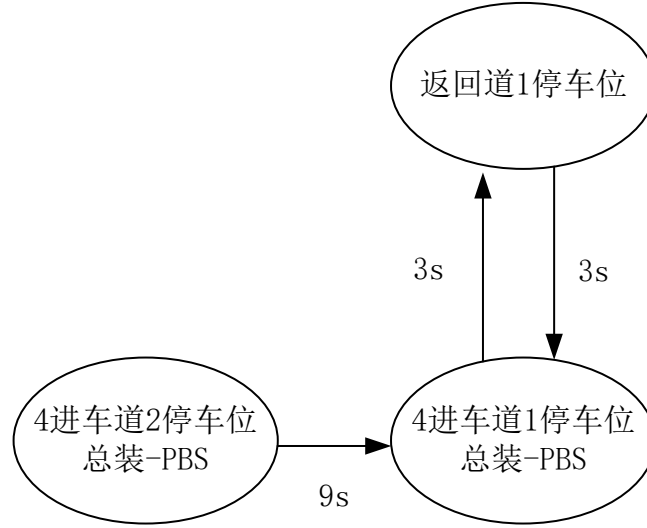


图 4.5 送车横移机处理 4 进车道上车至返回道的运作流程

如图所示，在接车横移机处理完第一辆车后返回初始位置的时刻与 4 进车道 2 停车位上的车到达 1 停车位的时刻，有 3 秒的时间间隔。所以，送车横移机不会连续将 4 进车道的两辆车送至返回道，即返回道 1 停车位上不会一直有车，不存在无法将进车道 1 停车位上的车运至返回道的情形。

其次，考虑是否要将到达进车道 1 停车位的车送回返回道，可分为两种情形。

1) 第 $i(v)$ 辆汽车直接送出，则送出时间

$$Te_{i(v)} = H_{i(v),k} \cdot ts_1 \quad (31)$$

此时，第 $i(v)$ 辆汽车退出程序，其结束时间为：

$$Tend_{i(v)} = T_{i(v)} + Tz_{i(v)} + \sum_j^{10} Tct_{i(v),j} + Te_{i(v)} \quad (32)$$

输出序列属性

$$L_i = J_{i(v)} \quad (33)$$

2) 第 $i(v)$ 辆汽车送往返回道，则送出时间

$$Te_{i(v)} = H_{i(v),k} \cdot tf_1 \quad (34)$$

此时，第 $i(v)$ 辆汽车进入返回道，整个流程结束时间为：

$$Tend_{i(v)} = T_{i(v)} + Tz_{i(v)} + \sum_j^{10} Tct_{i(v),j} + Te_{i(v)} \quad (35)$$

同时，令返回序列

$$J_{v+1} = J_{i(v)} \quad (36)$$

则此时 $v+1$ 辆在返回道 1 车位上的开始时间为

$$tjs_{v+1} = Tend_{i(v)} \quad (37)$$

(5) 车道行进规则:

建立矩阵 $D_{k,j,t}$ 表示在 t 时刻第 k 车道的第 j 个位置是第多少辆车, 如 $D_{1,6,5} = 5$ 表示的是在第 5s 时, 第 1 车道的第 6 车位是第五辆车, 该矩阵随着车辆进入进车道的的时间进行改变迭代。

在初始时刻, 接车横移机还没有开始工作, 即任意 k 车道的 j 停车位都没有车, 则

$$D_{k,j,0} = \begin{bmatrix} 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \end{bmatrix} \quad (38)$$

考虑到当汽车被送入进车道后, 其目前在进车道上的所有车由于送车横移机优先处理先到 1 号位的汽车, 且所有车道移动速度均为 9s, 即接入顺序将与送出顺序一致, 同时第 $i+1(v)$ 辆车将不会影响第 $i(v)$ 辆车的在每个车位上的时间。

因此, 第 $i(v)$ 辆车在 t 时刻进入车道后, 所有的运动都是由目前在进车道上的车辆决定的,

假设第 $i(v)$ 辆车在 t 时刻进入车道, 那么令

$$H_{i(v),k} \cdot D_{k,j,t} = i(v) \quad (39)$$

则目前进车道上总共有

$$y_t = \sum_k^6 \sum_j^{10} d_{k,j,t} \quad (40)$$

除掉第 $i(v)$ 辆车本身还有 $y_t - 1$ 辆车, 即目前正在被送车横移机处理的车为 $J_{i-y_t(v)}$, 则根据其是否被送往返回道, 其处理完的时刻为

$$tend_{i(v)} = \begin{cases} Tend_{i-y_t(v)} + 3, & r_{i(v)} = 1 \\ Tend_{i-y_t(v)} + H_{i(v),k} \cdot ts_2, & r_{i(v)} = 0 \end{cases} \quad (41)$$

此时, 在车道上即将被处理的车辆为 $J_{i-y_t+1(v)}$, 首先考虑 $J_{i-y_t+1(v)}$ 的位置, 可表示为

$$w_{i-y_t+1(v),j} = \left\lceil H_{i(v),k} \cdot \frac{D_{k,j,t}}{i} - y_t(v) + 1 \right\rceil \quad (42)$$

考虑到接车横移机移动的最小间隔为 3s, 因此每过 3s 更新一次状态。更新状态时首先要考虑当前时刻所有车的位置

当前时刻所有车的位置, 可表示为

$$d_{k,j,t} = \left\lceil \frac{D_{k,j,t}}{D_{k,j,t+1}} \right\rceil \quad (43)$$

要改变状态需要判断两种情况, 第一种当车位于 1 号车位时, 需等待多久, 送车机来送车, 第二种情况是当车位于其他车位时, 需判断前面是否有车:

第一种情况下, 即位于 1 号位其等待时间应当为

$$tsw_{i-y_t(v)+1} = tend_{i-y_t(v)} - t \quad (44)$$

而第 $t+3$ 时刻时, 如果 $tsw_{i-y_t(v)+1} \leq 3$, 则其将被运往送车机, 退出车道矩阵 $d_{k,j,t+3}$,

该位置被置 0, $tsw_{i-y_t(v)+1} > 3$ 则位置保持不变, 即

$$\begin{cases} d_{k,1,t+3} = d_{k,1,t}, & tsw_{i-y_t(v)+1} \leq 3 \\ D_{k,1,t+3} = D_{k,1,t}, & tsw_{i-y_t(v)+1} > 3 \end{cases} \quad (45)$$

第二种情况下, 位于其他车位的车需考虑前一位置的状态, 即 $j > 1$ 时

$$dk_{k,j>1,t} = d_{k,j,t} - d_{k,j-1,t} \quad (46)$$

如果 $dk_{k,j,t} = 1$ 则表示第 k 车道, 第 j 个车道其前一个车道是空的, 可移动致下一位置,

由于目前只过了 3s, 而移动车道需要 9s, 因此此时车道状态还未改变, 需要连续 3 次更新状态后, 即 9s 后才会发生改变, 此时

$$D_{k,j,t+9} = D_{k,j,t} - D_{k,j,t} \cdot dk_{k,j,t} + D_{k,j-1,t} \cdot dk_{k,j-1,t} \quad (47)$$

以此类推, 假设第 $i(v)$ 辆车经过 $3n$ 秒后被送车横移机送出, 那么第 $i(v)$ 辆汽车被送车横移机送出的时刻

$$tend_{i(v)} = t + 3n \quad (48)$$

以及第 $i(v)$ 辆车在每个车位上的时间等待矩阵

$$dw_{i(v)} = \sum_{t_i}^{tend_{i(v)}} [H_{i(v)} \cdot D_{k,j,t} - i(v)] \quad (49)$$

以第一辆车为例, 其进入第四车道时, 则矩阵

$$D_{k,j,0} = \begin{bmatrix} 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 1,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \end{bmatrix} \quad (50)$$

此时, 计算所有车道上车辆的总数目, 即

$$y_0 = \sum_k^6 \sum_j^{10} d_{k,j,0} = 1 \quad (51)$$

即除去第一辆车外, 整个进车道无车辆, 因此无需考虑前车,

$$D_{k,j,0+3} = D_{k,j,0+6} = \begin{bmatrix} 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 1,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \end{bmatrix} \quad (52)$$

以此类推, 由于只有 4 车道有车, 因此, 当 9s 后需要判断, 并更新状态

$$D_{k,j,0+9} = \begin{bmatrix} 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,1, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \end{bmatrix} \quad (53)$$

直到 81s 后，第一辆车到达 1 号位，即

$$D_{k,j,81} = \begin{bmatrix} 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,1 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \\ 0,0, \dots, 0,0 \end{bmatrix} \quad (54)$$

此时送车横移机没有处理车辆，则第一辆车直接被送往送车横移机，其在 1 号位的等待时间为 0，记录移出时间 $tend_1$ 最后移出车道，即第一辆车在各个车位的时间矩阵为

$$dw_1 = \sum_{t_1}^{tend_1} H_{1(0)} \cdot D_{k,j,t} \quad (55)$$

(6) 返回道行进规则：

返回道行进规则与进车道行进规则类似，建立矩阵 $S_{j,t}$ ，表示在第 t 时刻，返回道上的每一车位是否有车及如果有车，该车的序号。

则在 0 时刻，其应该为：

$$S_{j,0} = [0,0,0, \dots, 0,0] \quad (56)$$

假设此时第 v 辆车刚好进入返车道，则此时时刻为 tjs_v ，而此时的 1 号位置为 v

$$S_{1,tjs_v} = v \quad (57)$$

同样每隔 3s 更新一次状态，当 $tjs_v + 3$ 时刻时，首先判断前一时刻整个返回道上哪一个车位有车

即

$$S_{j,tjs_v} = \left\lfloor \frac{S_{j,tjs_v}}{S_{j,tjs_v} + 1} \right\rfloor \quad (58)$$

此时返车道上总共有 n_{tjs_v} 辆车，则

$$n_{tjs_v} = \sum_j^{10} S_{j,tjs_v} \quad (59)$$

由于返车道只有一条道，则此时返回道上即将被处理的车为第 $v - n_{tjs_v}$ 辆车，有两种情况，第一种情况，当返回道上的车位于 10 号位时，等待被接车横移机处理，此时，接车横移机正在处理第 $i - 1(v - n_{tjs_v})$ 辆车，则返回道上第 $v - n_{tjs_v}$ 辆车需在 10 号位等待

$$tfd_{v-n_{tjs_v}} = Tz_{i-1(v-n_{tjs_v})} + H_{i-1(v-n_{tjs_v})} \cdot ts_2 \quad (60)$$

如果 $tfd_{v-n_{tjs_v}} > 3$ ，则 $tjs + 3_v$ 时刻时， $v - n_{tjs_v}$ 仍在 10 号位，

$$S_{10,tjs_v+3} = S_{10,tjs_v} \quad (61)$$

如果 $tfd_{v-n_{tjs_v}} < 3$ ，则 $tjs + 3_v$ 时刻时， $v - n_{tjs_v}$ 退出返车道

$$S_{10,tjs_v+3} = 0 \quad (62)$$

当位于其他车道时，需要 9s 更新一次状态，首先判断前一车道是否有位置让该车前进

$$sy_{tjs_v} = s_{j,tjs_v} - s_{j+1,tjs_v} \quad (63)$$

如果 $sy_{tjs_v} = 1$ 则表示前一车位空缺，后一位置需在 9s 后移动到前车位置。

即

$$s_{j<10,tjs_v+9} = s_{j<10,tjs_v} - sy_{tjs_v} \cdot s_{j<10,tjs_v} + sy_{tjs_v} \cdot s_{j+1,tjs_v} \quad (j < 10) \quad (64)$$

以此类推，假设第 v 辆车经过 $3n$ 秒后被接车横移机送出，那么第 i 辆汽车被接车横移机送出的时刻

$$stend_v = tjs_v + 3n \quad (65)$$

以及第 v 辆车在每个车位上的时间等待矩阵

$$sw_v = \sum_{tjs_v}^{stend_v} [s_{j,tjs_v} - v] \quad (66)$$

优化目标：

(1) 混动车型间隔 2 台非混动车型为优

输出序列 $L_{d,i}$ 表示的是整个序列里面所有的动力类型将其用 0-1 变量表示，则

$$L_{d,i} = \begin{cases} 0, & \text{该车为燃油} \\ 1, & \text{该车为混动} \end{cases} \quad (67)$$

即优化目标所需要的排列顺序为：

$$1, 0, 0, 1, 0, 0, 1, \dots$$

利用卷积的思想，设置一个核函数

$$ker = [1, 9, 4, 2] \quad (68)$$

让输出序列 $L_{1,i}$ 中的连续四个数与核函数相乘再相加，即

$$zl_i = L_{1,i}ker_1 + L_{1,i+1}ker_2 + L_{1,i+2}ker_3 + L_{1,i+3}ker_4 \quad (69)$$

则如果 $zl_i = 3$ ，则第 i 辆开始连续 4 辆满足混动模型间隔 2 台非混动模型，若 zl_i 为其他值，则不满足，考虑到如果第 i 辆开始连续 4 辆满足混动模型间隔 2 台非混动模型，那么第 $i+1, i+2$ 辆车，不用计算是否满足优化目标 1，为了将 $i+1, i+2, i-1, i-2$ 移出 zl_i ，假设如果 $zl_i = 3$ ，则 $zl_{i+1} = zl_{i+2} = 8$ ，即只要找出序列 zl_i 中除去 8 和 3 以外的所有数，即可得到最后到底有多少辆车没有满足混动模型间隔 2 台非混动模型：

$$z = \{zl_i \neq 3, zl_i \neq 8 | zl_i\} \quad (70)$$

即最后需要扣 $|z|$ 分，那么目标函数 $z1$ 可表示为

$$z1 = 100 - |z| \quad (71)$$

(2) 四驱车型与两驱车型倾向 1:1 出车序列，

输出序列 $L_{q,i}$ 表示的是整个序列里面所有车辆的驱动类型，其用 0-1 变量表示，则

$$L_{q,i} = \begin{cases} 1, & \text{该车辆为两驱} \\ 0, & \text{该车辆为四驱} \end{cases} \quad (72)$$

设得分函数为

$$z2 = 100 - F(L_{2,i}) \quad (73)$$

由于是需要分块计算比例，当只有第一辆车的时候无法计算比例，因此从第二辆车开始则 $F(L_{2,i})$ 的算法流程如下：

Step1: 输入整个序列

Step2: 初始化处理流程，令 $n=2$, $p=1$, $q=0$, $F=0$

Step3: 判断 $L_{2,n}$ 是否等于 $L_{2,n-1}$ ，是则进行下一步，否则进行 Step5

Step4: 记录 $p=p+1$,

Step5: 记录 $q=q+1$

Step6: 判断是否 $p>q$ ，是则 $n=n+1$ ，进行 Step8，否则进行下一步

Step7: $F=F+1$, $n=n+1$, $p=q$, $q=0$

Step8: 判断 n 是否大于总数据量，大于执行下一步，否返回 Step3

Step9: 输出 F

具体算法流程，如图 4.6 所示。

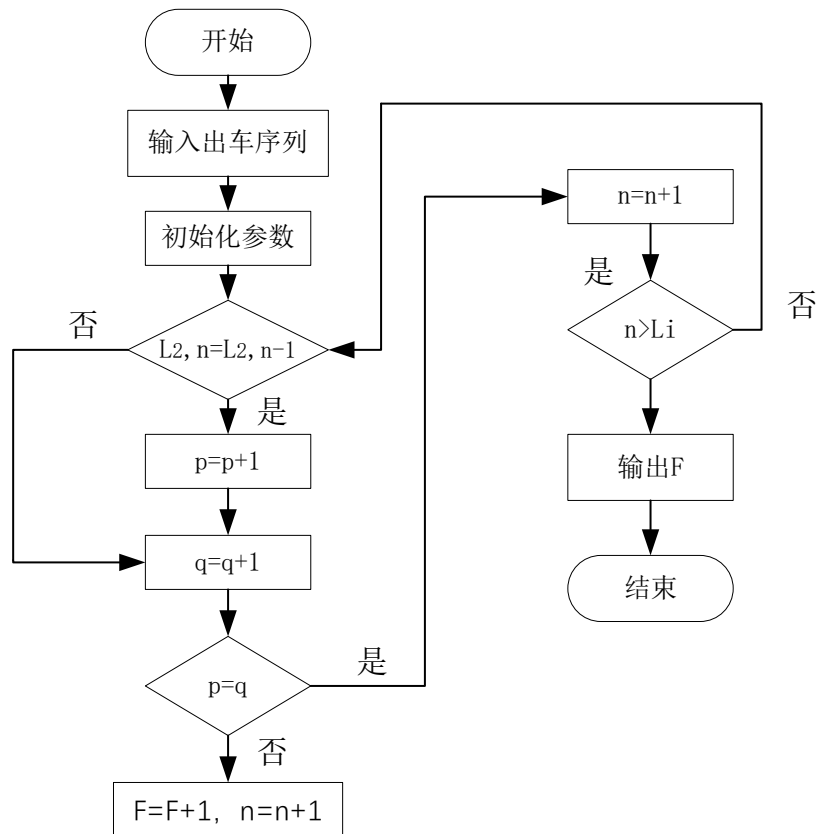


图 4.6 $F(L_{2,i})$ 的算法流程图

(3) 返回道使用次数倾向于 0，权重系数 0.2

$$z3 = 100 - v \quad (74)$$

(4) 总调度时间越短越好

假设出车队列长度为 C ，第一辆车身进入涂装-PBS 出车口时刻记为零，以最后一个车身进入 PBS-总装接车口的时刻 T 为总完成时间，其理论最快完成时间为 $9C+72$ (全部走进车道 4，出车序列与入车序列相同)，其时间惩罚值设置为 $0.01 \times (T - 9C - 72)$ ，最后目标

得分为 100 - 时间惩罚值。计算理论完成最快时间为 2934s，目标函数可表示为

$$z4 = 100 - 0.01 \times (Tend_{n(v)} - 2934) \quad (75)$$

其中 $Tend_{n(v)}$ 表示序列的最后一辆车 n 结束送车的时刻。即整个控制过程完成的时刻。

模型确立:

综上所述，将式 (11)、(13) - (36)、(39) - (49)、(57) - (66) 联立，可得生产线 PBS 优化调度模型为：

$$Z = \max(0.4 \times z1 + 0.3 \times z2 + 0.2 \times z3 + 0.1 \times z4) \quad (76)$$

$$\left. \begin{aligned}
& \sum_k^6 H_{i(v)k} = 1 \\
& T_{i(v)} = T_{i-1(v)} + H_{i-1(v),k} \cdot (ts_1 + ts_2) \\
& sy_{tjs_v} = s_{j,tjs_v} - s_{j+1,tjs_v} \\
& J_{i(v+1)} = J_{i(v)} \\
& T_{i(v+1)} = T_{i-1(v)} + H_{i-1(v),k} \cdot (ts_1 + ts_2) + h_v \cdot tf_2' \\
& tfd_{v-n_{tjs_v}} = Tz_{i-1(v-n_{tjs_v})} + H_{i-1(v-n_{tjs_v})} \cdot ts_2 \\
& Tz_{i(v)} = TD_{i(v)} + TDS_{i(v)} + Td_{i(v)} \\
& t_{i-1(v)} = T_{i-1(v)} + TD_{i-1(v)} + Tc_{i(v)} \\
& tc_{i(v)} = t_{i-1(v)} + TD_{i(v)} \\
& x_{i,k} = \left\lfloor \frac{D_{k,10,tc_{i(v)}}}{D_{k,10,tc_{i(v)}}+1} \right\rfloor \\
& TDS_{i(v)} = 9 - H_{i(v),k} \cdot \left\lfloor \frac{D_{k,10,tc_{i(v)}}}{D_{k,10,tc_{i(v)}}+1} \right\rfloor \cdot ts_1 \\
& Td_{i(v)} = H_{i(v),k} \cdot ts_1 \\
& sw_v = \sum_{tjs_v}^{stend_v} [s_{j,tjs_v} - v] \\
& Tend_{i(v)} = T_{i(v)} + Tz_{i(v)} + \sum_j^{10} Tct_{i(v),j} + Te_{i(v)} \\
& L_i = J_{i(v)} \\
& Te_{i(v)} = H_{i(v),k} \cdot tf_1 \\
& dw_1 = \sum_{t_1}^{tend_1} H_{1(0)} \cdot D_{k,j,t} \\
& tjs_{v+1} = Tend_{i(v)} \\
& D_{k,j,t+9} = D_{k,j,t} - D_{k,j,t} \cdot dk_{k,j,t} + D_{k,j-1,t} \cdot dk_{k,j-1,t} \quad (j > 1) \\
& dk_{k,j>1,t} = d_{k,j,t} - d_{k,j-1,t} \\
& tsw_{i-y_t(v)+1} = tend_{i-y_t(v)} - t \\
& dw_i = \sum_{t_i}^{tend_{i(v)}} [H_{i(v)} \cdot D_{k,j,t} - i(v)] \\
& d_{k,j,t} = \left\lfloor \frac{D_{k,j,t}}{D_{k,j,t}+1} \right\rfloor \\
& w_{i-y_t+1(v),j} = [H_{i(v),k} \cdot D_{k,j,t} / i - y_t(v) + 1] \\
& y_t = \sum_k^6 \sum_j^{10} d_{k,j,t} \\
& H_{i(v),k} \cdot D_{k,j,t} = i(v) \\
& s_{j,tjs_v} = \left\lfloor \frac{s_{j,tjs_v}}{s_{j,tjs_v} + 1} \right\rfloor \\
& \dots
\end{aligned} \right\} s. t. = \tag{77}$$

4.3 考虑横移机调度优先级的 PBS 优化问题算法设计

采用灰狼算法对问题一所建立的动态规划模型进行求解，即通过模拟狼群捕猎对模型进行优化来解决动态调度问题。

灰狼算法，即通过构建适应度函数来模拟狼群中的等级制度，将适应度最高的个体定义为狼王，记为 α ，其次记为 β_1 ，再次记为 β_2 ，其他群狼定义为最低等级的狼，记为 δ 。狼群在适应度最高的狼 α ， β_1 ， β_2 带领下调整自己的位置，使种群不断逼近最优解，从而获得一个近似最优解。

通过对车辆调度过程分析，可以发现该模型主要通过接车横移机调整所进入的车道序号，是否进入返回道来控制车身运动，因此以每辆车所进入的车道序号，是否进入返回道作为控制方案，然后对时间进行模拟获得得分。本文通过先随机生成控制方案，然后模拟流程计算结果，随着迭代次数的增加，每次迭代狼群均会往头狼的方向移动，使得解决方案逐渐优化，从而获得最优解。

实际上，考虑到在对 PBS 相关时间数据进行处理过程中，接车横移机与送车横移机将车辆从在运往返回道时，虽然他们的来回时间都是 $[24, 18, 12, 6, 12, 18]$ ，但实际上车辆到达车位的时间并不相同，如图 4.7 所示。

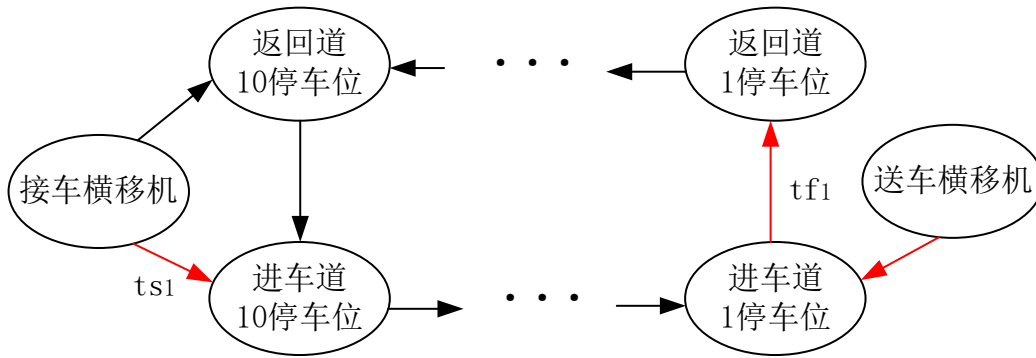


图 4.7 车辆经过返回道回到进车道 10 停车位的运作流程

为在车辆运输过程中运用 1 次返回道时的情形，即接车横移机从涂装-PBS 口出发将车辆运到进车道 10 停车位，而后车辆通过进车道停车位不断后移，到达进车道 1 停车位时，送车横移机从总装 PBS 口出发，将车辆从进车道 1 停车位运送到返回道 1 停车位，而后经过返回道停车位不断前移，到达返回道 10 停车位时，接车横移机从涂装-PBS 口出发，将车辆从返回道 10 停车位运至进车道 10 停车位。

通过上述对 PBS 相关时间数据的分析，可以得到车辆经过返回道 1 次，横移机在调度过程中消耗的时间，即

$$ts_1 + tf_2 = [30, 21, 12, 3, 15, 24]^T$$

由此，可以根据上述过程消耗的总时间最短为优，设定车道 C_i 的选择优先级，为

$$C_4 > C_3 > C_5 > C_2 > C_6 > C_1$$

通过对各优化目标的权重分析，应优先满足权重较高的目标，即每两辆混动车之间要间隔两辆燃油车，四驱与两驱趋向 1:1 的出车序列。通过对附件所给车辆的数据分析，得分比重较大的燃油和四驱的车型数量较少，应当依据车道的选择优先级较高的处理，因此通过上述策略启发最先开始的随机生产方案，使得更容易得到最优解。

设计算法步骤如下：

Step1: 设置迭代次数

Step2: 通过启发策略随机生成 40 个狼群位置即控制方案

Step3: 初始化最开始的时间 $t=0$ ，输入车辆，车道时间矩阵，

Step4: 判断 t 时刻的输入车辆是否为最后一辆车，否，则进行下一步；是，则进行 step6

Step5: $t=t+3$ ，通过控制方案，更新当前输入车辆，车道时间矩阵，返回 step4

Step6: $t=t+3$ ，通过控制方案，更新当前输入车辆，车道时间矩阵

Step7: 判断 t 时刻车道已经处理完成，是，则计算适应度函数；否，则进行 step6

Step8: 通过适应度函数，更新 α ， β_1 ， β_2 的位置

Step9: δ 灰狼依据 α ， β_1 ， β_2 更新位置

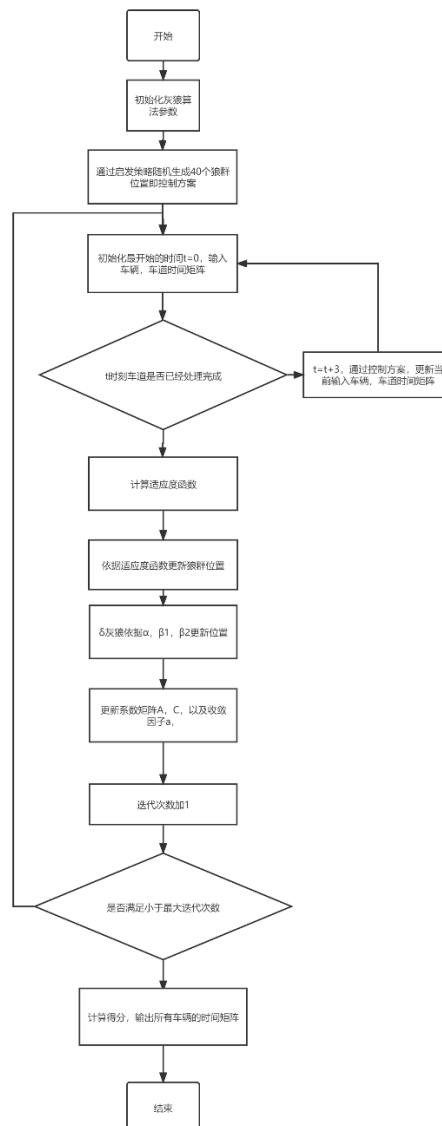
Step10: 更新系数矩阵 A ， C ，以及收敛因子 a ，

Step11: 迭代次数加 1

Step12: 是否满足小于最大迭代次数，是，进行下一步，否，返回 Step3

Step13: 计算得分，输出所有车辆的时间矩阵

算法具体流程图如下：



4.4 考虑横移机调度优先级的 PBS 优化问题求解结果

根据上述算法，运用 Python 对模型进行求解，求解结果如下表所示：

附件 1：

优化后总调度时间： 3056s

优化后返回道使用次数： 6 次

优化目标	理想得分	优化得分
1	36.8	-34.4
2	22	24.60
3	20	18.8
4	10	9.916
总得分	88.8	18.916


















附件 2：

优化后总调度时间： 2996s

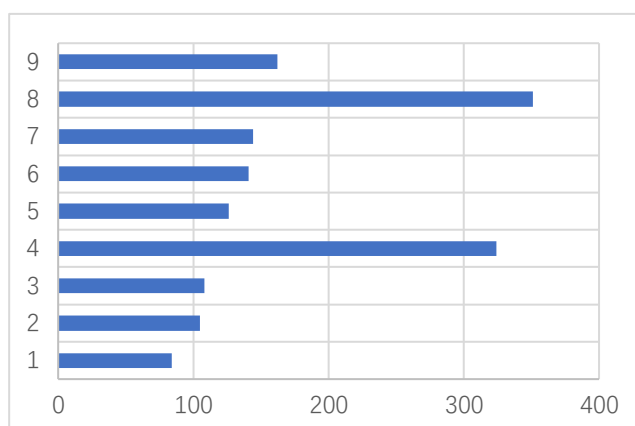
优化后返回道使用次数： 6 次

优化目标	理想得分	优化得分
1	36.8	-10.8
2	22	24.6
3	20	18.8
4	10	10.039
总得分	88.8	44.439

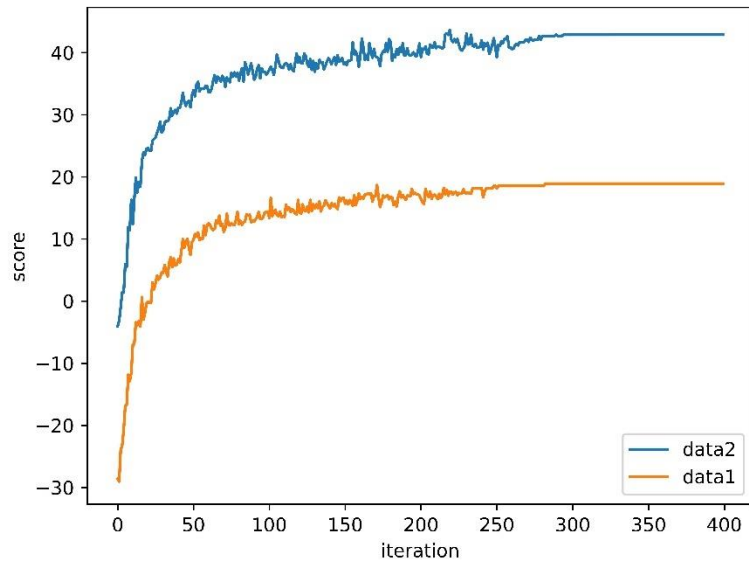
第 1250 秒车道上所有车辆所处的位置（以附件 1 为例）：

	10	9	8	7	6	5	4	3	2	1
进车道 6										
进车道 5										
返回道										
进车道 4										
进车道 3										
进车道 2										
进车道 1										

车辆以进车序列经过 PBS 调度后到达总装-PBS 口的时刻（第 1-9 辆车），如下图所示（横坐标：表当前时刻，纵坐标：表进车序列）：



该问题的迭代优化图，如下：



通过对上图观察可以发现，附件 1 和附件 2 的求解结果在经过 300 次左右迭代后，其变化趋势趋于平稳。为了防止模型陷入局部最优解，对模型进行多次迭代。通过综合考量，可以确定此结果为当前最优解。

五、问题二

5.1 不考虑横移机调度优先级的 PBS 优化问题的分析

在问题一的基础上，减少两条约束，即返回道 10 停车位和进车道 1 停车位上的车辆不需要立刻处理。针对约束条件 6，取消该约束时，将会出现在接车横移机空闲时，返回道 10 停车位和涂装-PBS 出车口都有车等待处理。针对约束条件 7，取消该约束时，汽车被送车横移机处理的控制流程就会发生改变，即不止一个车道的 1 停车位上有待处理的车，则送车横移机需要判断优先处理顺序。

5.2 不考虑横移机调度优先级的 PBS 优化模型建立

决策变量：

与问题一相比，在本问题中，不考虑横移机调度的优先级，即返回道 10 停车位上的车辆就可以停留。因此，增加决策变量 th_v 表示处于返回道 10 停车位上车辆的等待时间。

约束条件：

对比问题一，以下约束发生改变：

(1) 车辆出发时刻的约束：

当返回道上 10 号位没有车时，且在接送上一辆车后还未返回初始位置，与问题一一样，这里不予赘述；当返回道上 10 号位有车时，即不再是优先处理返回道，那么返回道上 10 号位的车将增加新的变量，即等待时间

当 $S_{10, T_{i-1(v)} + H_{i-1(v),k} \cdot ts_1} = v$ 时，此时，增加一新变量 th_v 表示返回道上的第 $i(v)$ 辆车在 10 号位的等待时间，如果等待时间大于接车横移机处理第 $i-1(v)$ 辆车的时间，即

$$stend_v + th_v > T_{i-1(v)} + H_{i-1(v),k} \cdot ts_2 \quad (78)$$

则处理涂装-PBS 出车口上的汽车，其出发时刻为

$$T_{i(v)} = T_{i-1(v)} + H_{i-1(v),k} \cdot ts_1 \quad (79)$$

如果等待时间大于接车横移机处理第 $i-1$ 车的时间，即

$$stend_v + th_v < T_{i-1(v)} + H_{i-1(v),k} \cdot ts_2 \quad (80)$$

需处理完上一辆车后，处理返回道的车，全部处理完后，再处理该车。

此时由于返回道上有车，在接车横移机从返回道接车第 v 次时，第 $i(v)$ 辆车的等待时间需加上优先处理返回道上的时间，即：

$$T_{i(v)} = T_{i-1(v)} + H_{i-1(v),k} \cdot ts_1 + (tfd_1 + tfd_2) \cdot h_v + th_v \quad (81)$$

其中 h_v 表示接车横移机第 v 次接车是否到某一车道，考虑到这里要优先处理返回道上的车，第 v 次车辆已经彻底迭代完成，重新插入队列中，变为第 $i(v+1)$ 辆车，即

$$J_{i(v+1)} = J_v \quad (82)$$

该车的等待时间应该为

$$T_{i(v+1)} = T_{i-1(v)} + H_{i-1(v),k} \cdot ts_1 + tfd_1 \cdot h_v + th_v \quad (83)$$

与此同时，原来的第 $i(v)$ 辆车变为第 $i+1(v+1)$ 辆车，即

$$J_{i+1(v+1)} = J_{i(v)} \quad (84)$$

其出发时刻可表示

$$T_{i+1(v+1)} = T_{i(v)} = T_{i-1(v)} + H_{i-1(v),k} \cdot ts_1 + (tfd_1 + tfd_2) \cdot h_v + th_v \quad (85)$$

(2) 汽车被送车横移机处理的时间

汽车在被送车横移机处理时有两种状态： $r_{i(v)}$ 表示是否被送往返回道，与此同时，还需要判断返回道上的 1 号位是否有车，此时时刻为

$$tr_{i(v)} = T_{i(v)} + T_{z_{i(v)}} + \sum_j^{10} T_{ct_{i(v),j}} \quad (86)$$

则当 $s_{1,tr_{i(v)}}=0$, $r_{i(v)} = 1$ 时才能将汽车送往返回道

$$Te_{i(v)} = H_{i(v),k} \cdot tf_1 \quad (87)$$

此时, 第 $i(v)$ 辆汽车进入返回道, 整个流程结束时间为:

$$Tend_{i(v)} = T_{i(v)} + T_{z_{i(v)}} + \sum_j^{10} T_{ct_{i(v),j}} + Te_{i(v)} \quad (88)$$

当 $s_{1,tr_{i(v)}} \neq 0$ 时, 还需要加上 $v-1$ 辆汽车运送到 2 车道的的时间

$$sw_{v-1} = \sum_{tjs_{v-1}}^{stend_{v-1}} [s_{1,t} - v] \quad (89)$$

结束时间为

$$Tend_{i(v)} = T_{i(v)} + T_{z_{i(v)}} + \sum_j^{10} T_{ct_{i(v),j}} + Te_{i(v)} + sw_{v-1} \quad (90)$$

最后, 令返回序列

$$J_{v+1} = J_{i(v)} \quad (91)$$

则此时 $v+1$ 辆在返回道 1 车位上的开始时间为

$$tjs_{v+1} = Tend_{i(v)} \quad (92)$$

如果 $r_{i(v)} = 0$, 则第 i 辆汽车直接送出, 则送出时间

$$Te_{i(v)} = H_{i(v),k} \cdot ts_1 \quad (93)$$

此时, 第 i 辆汽车退出程序, 其结束时间为:

$$Tend_{i(v)} = T_{i(v)} + T_{z_{i(v)}} + \sum_j^{10} T_{ct_{i(v),j}} + Te_{i(v)} \quad (94)$$

输出序列属性

$$L_i = J_{i(v)} \quad (95)$$

考虑约束六, 即存在一种情况 1 号位有不止一辆汽车, 需要送车横移机做出选择, 此时汽车的车道行进规则改变。

(3) 车道行进规则

当前时刻所有车的位置, 可表示为

$$d_{k,j,t} = \left\lfloor \frac{D_{k,j,t}}{D_{k,j,t+1}} \right\rfloor \quad (96)$$

此时, 位于 1 号位的车辆总数可表示为:

$$num = \sum_k^6 d_{k,1,t} \quad (97)$$

当 $num \leq 1$ 时, 其等待时间与问题一相同, 不改变, 当 $num > 1$ 时, 随着时间的推进考虑送车横移机选择策略,

当 $t+3$ 时，目前正在被送车横移机处理的车为 $J_{i-y_t(v)}$ ，则当前时刻输出的序列为 $L_{i-y_t(v)}$ 作为判断依据，对目前正在 1 道的车进行模拟，选择得分最高的序列，然后处理该车。

则需判断的车辆有

$$i - y_t(v) - 1, i - y_t(v) - 2, \dots, i - y_t(v) - num \quad (98)$$

其得分分别为

$$Z_{i-y_t(v)-1}, Z_{i-y_t(v)-2}, \dots, Z_{i-y_t(v)-num} \quad (99)$$

需处理的车辆为

$$sx_m = \max\{Z_{i-y_t(v)-1}, Z_{i-y_t(v)-2}, \dots, Z_{i-y_t(v)-num}\} \quad (100)$$

令目前需要处理的车辆 $i - y_t(v) - 1 = m$ ，其他车辆次序依次改变。

模型确立：

综上所述，联立模型一，得到不考虑横移机调度优先级的 PBS 优化模型为：

$$Z = \max(0.4 \times z1 + 0.3 \times z2 + 0.2 \times z3 + 0.1 \times z4) \quad (101)$$

$$\begin{aligned}
& \sum_k^6 H_{i(v)k} = 1 \\
& T_{i(v)} = T_{i-1(v)} + H_{i-1(v),k} \cdot (ts_1 + ts_2) \\
& Tz_{i(v)} = TD_{i(v)} + TDS_{i(v)} + Td_{i(v)} \\
& TDS_{i(v)} = 9 - H_{i(v),k} \cdot \left\lfloor \frac{D_{k,10,tc_{i(v)}}}{D_{k,10,tc_{i(v)}}+1} \right\rfloor \cdot ts_1 \\
& Te_{i(v)} = H_{i(v),k} \cdot ts_1 \\
& stend_{i(v)} + th_{i(v)} > T_{i-1(v)} + H_{i-1(v),k} \cdot ts_2 \\
& T_{i(v)} = T_{i-1(v)} + H_{i-1(v),k} \cdot ts_1 \\
& T_{i(v+1)} = T_{i-1(v)} + H_{i-1(v),k} \cdot (ts_1 + ts_2) + h_v \cdot tf_2' \\
& J_{i(v+1)} = J_v \\
& tc_{i(v)} = t_{i-1(v)} + TD_{i(v)} \\
& D_{k,j,t+9} = D_{k,j,t} - D_{k,j,t} \cdot dk_{k,j,t} + D_{k,j-1,t} \cdot dk_{k,j-1,t} \quad (j > 1) \\
& dw_i = \sum_{t_i}^{tend_{i(v)}} [H_{i(v)} \cdot D_{k,j,t} - i(v)] \\
& d_{k,j,t} = \left\lfloor \frac{D_{k,j,t}}{D_{k,j,t}+1} \right\rfloor \\
& s_{j,tjs_v} = \left\lfloor \frac{s_{j,tjs_v}}{s_{j,tjs_v} + 1} \right\rfloor \\
& tr_{i(v)} = T_{i(v)} + Tz_{i(v)} + \sum_j^{10} Tct_{i(v)j} \\
& Te_{i(v)} = H_{i(v),k} \cdot tf_1 \\
& Tend_{i(v)} = T_{i(v)} + Tz_{i(v)} + \sum_j^{10} Tct_{i(v)j} + Te_{i(v)} \\
& sw_{i(v-1)} = \sum_{tjs_{i(v-1)}}^{stend_{i(v-1)}} [s_{1,t} - v] \\
& Tend_{i(v)} = T_{i(v)} + Tz_{i(v)} + \sum_j^{10} Tct_{i(v)j} + Te_{i(v)} + sw_{v-1} \\
& J_{i(v+1)} = J_{i(v)} \\
& tjs_{i(v+1)} = Tend_{i(v)} \\
& Te_{i(v)} = H_{i(v),k} \cdot ts_1 \\
& Tend_{i(v)} = T_{i(v)} + Tz_{i(v)} + \sum_j^{10} Tct_{i(v)j} + Te_{i(v)} \\
& L_i = J_{i(v)} \\
& num = \sum_k^6 d_{k,1,t} \\
& \dots
\end{aligned}
\quad s. t. = \left\{ \begin{array}{l} \end{array} \right. \quad (102)$$

5.3 不考虑横移机调度优先级的 PBS 优化问题算法设计

该题算法与问题一模型的求解过程相同，也是采用灰狼算法对根据问题 2 所建立的动态规划模型进行求解，与问题一不同的是，由于取消了约束 6 和约束 7，除了问题一涉及到的决策变量：横移机接送每辆车进入的车道序号，每辆车是否进入返回道以外，还有新的决策变量，每辆进入返回道的车辆在 10 号位的等待时间，而送车横移机遇到多辆车在 1 号位等待时，将一号位的每一辆车模拟送出整个序列的得分，得分最高的先行处理。

模型二主要体现在决策变量的增加，以及车道行进规则的改变，其整个的算法流程与问题一的求解相同。

该算法设计步骤如下：

Step1: 初始化灰狼算法参数

Step2: 通过启发策略随机生成 40 个狼群位置即控制方案

Step3: 初始化最开始的时间 $t=0$ ，输入车辆，车道时间矩阵，

Step4: 判断 t 时刻车道是否已经处理完成，是，进行 step6，进行下一步，否，进行下一步

Step5: $t=t+3$ ，通过控制方案，更新当前输入车辆，车道时间矩阵

Step6: 计算适应度函数

Step6: 通过适应度函数，获得每头狼的生存价值。

Step7: 更新狼群的位置

Step8: 依据适应度函数， α ， β_1 ， β_2 更新位置，

Step9: 迭代次数加 1

Step10: 是否满足小于最大迭代次数，是，进行下一步，否，返回 Step3

Step11: 计算得分，输出所有车辆的时间矩阵

流程图与问题一相似，这里不重复展示。

5.4 不考虑横移机调度优先级的 PBS 优化模型求解结果

通过上述算法分析，利用 Python 进行求解，得到以下结果：

附件 1:

优化后总调度时间： 3056s

优化后返回道使用次数： 8 次

优化目标	理想得分	优化得分
1	36.8	-32.8
2	22	24
3	20	18.4
4	10	9.859
总得分	88.8	19.459



















附件 2:

优化后总调度时间： 3014s

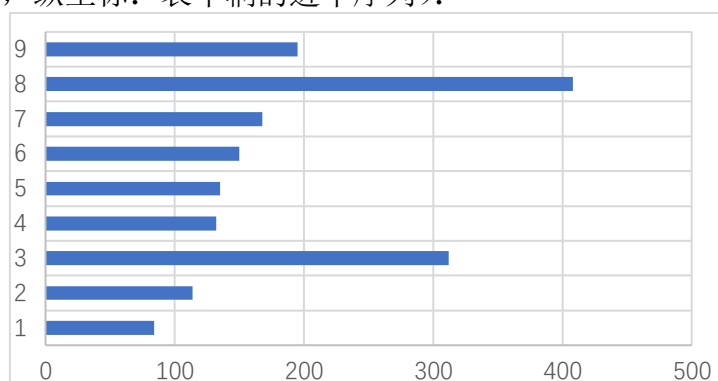
优化后返回道使用次数： 4 次

优化目标	理想得分	优化得分
1	36.8	-8
2	22	24.60
3	20	19.2
4	10	9.907
总得分	88.8	45.78

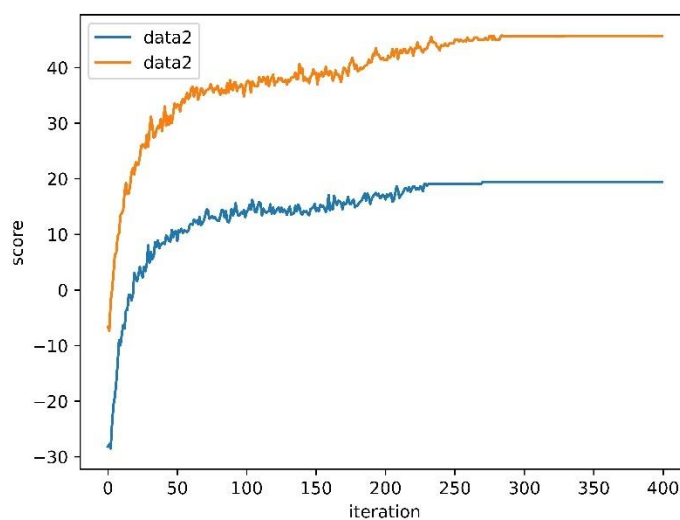
第 1250 秒车道上所有车辆所处的位置（以附件 1 为例）：

	10	9	8	7	6	5	4	3	2	1
进车道 6										
进车道 5										
返回道										
进车道 4										
进车道 3										
进车道 2										
进车道 1										

车辆以进车序列经过 PBS 调度后到达总装-PBS 口的时刻（第 1-9 辆车），如下图所示（横坐标：表当前时刻，纵坐标：表车辆的进车序列）：



该问题的迭代优化图，如下：



通过对上图观察可以发现，附件 1 和附件 2 的求解结果在经过 250-300 次迭代后，其变化趋势趋于平稳。与问题 1 的求解结果相比较，可发现模型二比模型一能够更快的收敛。同时，通过比较模型一和模型二的得分情况，可发现无论是附件 1 还是附件 2，模型二的得分均高于模型一，即模型二优于模型一。

六、模型评价与推广

6.1 模型的优点

- (1) 模型采用较为成熟的数学理论来建立，因此所得结果较为可靠。
- (2) 该模型建立流程清晰，详细易懂，便于今后对模型进行进一步的优化推广。
- (3) 该模型的建立运用了启发式算法，有效降低了模型求解时的困难。

6.2 模型的缺点

- (1) 模型虽然综合考虑了很多因素，但由于建模时间的限制，可能忽略了其他实际应用中可能对生产调度产生的影响，在实际应用中具有一定的局限性。
- (2) 该模型具有启发式算法的局限性，无法得到全局最优解。

6.3 模型的推广

该模型是有关生产线缓存区优化调度问题的一个典型案例，可广泛应用于流水工作车间。此外，也可以该模型为基础，进行简单的修改，推广应用于公交车、人力资源等调度问题。

七、参考文献

- [1] 陈广阳. 汽车生产线缓冲区设计及排序问题研究[D]. 华中科技大学, 2007.
- [2] 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.

八、附录

问题 1:

```
import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt

data_1 = pd.read_excel('附件 1.xlsx')
data_2 = pd.read_excel('附件 2.xlsx')
data_1 = np.array(data_1)
data_2 = np.array(data_2)
data_1[:,2] = np.where(data_1[:,2]=='燃油',0,1)
data_1[:,3] = np.where(data_1[:,3]=='两驱',0,1)
data_2[:,2] = np.where(data_2[:,2]=='燃油',0,1)
data_2[:,3] = np.where(data_2[:,3]=='两驱',0,1)

class GWO():
    def __init__(self,dim=6000,maxiter=200,size=40,lb=0,ub=0.999999999):
        self.dim = dim
        self.X = np.random.uniform(low=lb,high=ub,size=(size,dim))
        self.alpha_wolf_pos = None
        self.beta_wolf_1_pos = None
        self.beta_wolf_2_pos = None
        self.maxiter = maxiter
        self.size = size
        self.lb = lb
        self.ub = ub
        self.gen_best_y = np.zeros(shape=(self.maxiter+1,))
        self.gen_best_X = np.zeros(shape=(self.maxiter+1,dim))
        self.gen_min_y = None
        self.gen_min_x = None
        self.time_in_out = np.array([18, 12, 6, 0, 12, 18]) / 3
        self.time_fan = np.array([24, 18, 12, 6, 12, 18]) / 3
        self.timeChedaoToHyj = self.time_in_out / 2
        self.time_fanToChedao = np.array([4, 3, 2, 1, 1, 2])
        self.time_0 = 3
        self.gold1 = None
        self.gold2 = None
        self.gold3 = None
        self.gold4 = None
```

```

def fitness(self,data):
    self.curY = np.zeros(shape=(self.size,))
    for i in range(self.size):
        ans, outLine, needNunberFan, costtime = self.genX(maxiter=self.dim/2, data=data, renwu=self.X[i], cur=i)
        curGold, self.gold1, self.gold2, self.gold3, self.gold4 = self.getGold(outLine, needNunberFan, data, costtime)
        self.curY[i] = -1*curGold

def getCurWolf(self,t):
    index = self.curY.argsort()
    self.alpha_wolf_pos = self.X[index[0]]
    self.beta_wolf_1_pos = self.X[index[1]]
    self.beta_wolf_2_pos = self.X[index[2]]

def getNewX(self,t):
    r_1_a = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    r_1_b_1 = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    r_1_b_2 = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    r_2_a = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    r_2_b_1 = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    r_2_b_2 = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    a = 2*(1-t/self.maxiter)
    A_a = 2*a*r_1_a - a
    A_b_1 = 2*a*r_1_b_1 - a
    A_b_2 = 2 * a * r_1_b_2 - a
    C_a = 2*r_2_a
    C_b_1 = 2*r_2_b_1
    C_b_2 = 2*r_2_b_2
    distance_a = np.abs(C_a * self.alpha_wolf_pos - self.X)
    distance_b_1 = np.abs(C_b_1 * self.beta_wolf_1_pos - self.X)
    distance_b_2 = np.abs(C_b_2 * self.beta_wolf_2_pos - self.X)
    X_a = self.alpha_wolf_pos - A_a * distance_a
    X_b_1 = self.beta_wolf_1_pos - A_b_1 * distance_b_1
    X_b_2 = self.beta_wolf_2_pos - A_b_2 * distance_b_2
    new_X = (X_a+X_b_1+X_b_2)/3
    new_X = np.where(new_X<self.lb,self.lb,new_X)
    new_X = np.where(new_X>self.ub,self.ub,new_X)
    return new_X

def chedaoOneToOne(self,x, result, time_i):
    xAfter0 = x[:, 3:-1]
    xIndex = np.where(xAfter0 != -1)
    curVar = (xIndex[0].reshape(-1, 1) + 1) * 10 + 10 - (xIndex[1].reshape(-1, 1) + 3) // 3
    curVar = curVar.reshape(-1, 1)

```

```

result[xAfter0[xIndex[0], xIndex[1]].astype(int), time_i * 3:time_i * 3 + 3] = curVar
xIndex = np.where(x[:, :3] != -1)
result[x[xIndex[0], xIndex[1]].astype(int), time_i * 3:time_i * 3 + 3] = (xIndex[0].reshape(-1,
1) + 1) * 100 + 10

xIndex = np.where(x[:, -1] != -1)
result[x[xIndex[0], -1].astype(int), time_i * 3:time_i * 3 + 3] = (xIndex[0].reshape(-1, 1) + 1) * 10 + 1
return result

def fanOneToOne(self, x, result, time_i): ###x 表示每个车道车位上对应的车的编号, -1 为没车
    curVar = x[0]
    if curVar != -1:
        result[int(curVar), time_i * 3:time_i * 3 + 3] = 710 ###更新反车道 10 车位

    curVar = x[1:]
    curVarIndex = np.where(curVar != -1)[0] ###找到有车的车位
    result[curVar[curVarIndex].astype(int), time_i * 3:time_i * 3 + 3] = 70 + 9 - curVarIndex.reshape(-1,
1) // 3 ###更新反车道 1-9 车位

    return result

def Outputresult(self, x):
    x_Index = np.where(x != 3)[1].max()
    return x[:, :x_Index + 3]

def getGold(self, x, number, data, costTime):
    l = len(x)
    data = data[:, 2:]
    x_list = []
    for i in range(l):
        x_list.append(data[x[i], 1])
    item2 = self.z2(x_list)
    x_list = []
    for i in range(l):
        x_list.append(data[x[i], 0])
    item1 = self.z1(x_list)
    result = (100-item1)*0.4+(100-item2)*0.3+(100-number)*0.2+0.1*(100-(costTime-2934)*0.01)
    return result, (100-item1)*0.4, (100-item2)*0.3, (100-number)*0.2, 0.1*(100-(costTime-2934)*0.01)

def z1(self, data):
    result = 0
    cc = 0
    data = np.array(data)
    a = np.where((data == 1))
    for i in range(1, len(a[0])):
        if a[0][i] - a[0][i - 1] == 3:

```

```

        result = result + 1
    else:
        cc = cc + 1
    rrr = cc
    return rrr

def z2(self,list1):
    result = [0]
    rr = []
    flag = 0
    j = 1
    rrr = 0
    R = 0
    for i in range(1, len(list1)):
        if list1[i] == list1[i - 1]:
            flag += 1
        else:
            flag = 0
        result.append(flag)
    result.append(0)
    for i in range(1, len(result)):
        if result[i] <= result[i - 1]:
            rr.append(result[i - 1] + 1)

    while j < len(rr):
        if rr[j] == rr[j - 1]:
            rrr = rrr + 1

        else:
            R = R + 1
        j = j + 2
    if len(rr) % 2:
        R = R + 1
    return R

def genX(self,renwu,maxiter, data,cur):
    renwu = renwu.reshape(-1,2)
    m, n = data.shape ###数据长度
    outputLine = [] ###输出队列
    inputHyjToChedaoGang = [] ###刚刚从输入横移机放入车道
    outputHyjTofanGang = [] ###刚刚从输出横移机放入反车道
    result = np.zeros(shape=(m, int(maxiter * 3)))
    chedaoIndex = np.zeros(shape=(6, 28)) - 1 ###车道上车的编号，没车就是-1
    fanIndex = np.zeros(28) - 1 ###返回道上车位的编号，没车为-1

```

点

```
chedaoState = np.zeros(shape=(6, 28)) ###车道路上有无车, 有车1, 没车0
fanState = np.zeros(28) ###反车道上的状态, 有车1, 没车0
fanNumber = 0 ###使用返回道次数
resultHyjStarEnd = np.zeros(shape=(m, 5)) ###0 还没出, 1 在横移机上, 2 在车道上, 3 在输出横移机, 4 在终
resultHyjStarEnd[:, 0] = 1
ifStar = np.zeros(m)
ifStar[0] = 1 ###
if_fanStar = 0 ###返回车道的第10车位状态, 0 为空, 1 为满
ifInputHyj = 0 ###入口横移机是否在工作, 0 为空, 1 为满
ifOutputHyj = 0 ###出口横移机是否在工作, 0 为空, 1 为满
timeFanArrive = 0 ###返回道到达时间
fromWhere = 0 ###输入横移机上的车的来源, 0 表示PBS, 1 表示返回道
arriveIndex = [] ###到达最后一个车位的车道顺序
outOrInput = 0 ###横移机准备把车送出去还是送回返回道, 0 表示去返回道, 1 表示送出去
inputHyjCar = -1 ###初始化输入横移机上的车的编号, -1 表示没有车
OutHyjCar = -1 ###初始化输出横移机上的车编号, 没车表示-1
PbsCarCur = 0 ###初始化PBS->输入横移机上的编号, 第一辆为0
arrive10Chewei = np.zeros(m)
i = 0
while i < maxiter:
    ifRunChedao = np.zeros(shape=(6, 27)) ###可移动的车道,1 表示可移动, 0 表示不可移动,这得对约束的理解
    ifRunFan = np.zeros(27) ###返回道是否可移动,1 表示可移动, 0 表示不可移动
    if ifInputHyj == 0: ###如果输入横移机空闲
        ifRunChedao_0 = chedaoState[:, :3] ###10 车位上的车道状态
        ifRunChedao_0 = np.where(ifRunChedao_0 == 1) ###10 车位上有车的索引
        needQueren = []
        canRunChedao = [3,2,4,1,5,0] ###可选择车道
        for j in range(6):
            if j in ifRunChedao_0[0]:
                canRunChedao.remove(j) ###剩下一定能走的车道
                needQueren.append(j) ###需要计算时间确认的车道
            ###这里根据来源不同进行不同的计算可选择车道
            ###计算可选择车道这里有问题
            # if if_fanStar == 1: ###10 号车位返回道有车
            #     if needQueren:
            #         for j in range(len(needQueren)):
            #             if time_fanToChedao[needQueren[j]] > 3 - ifRunChedao_0[needQueren[j], 1]:
            #                 canRunChedao.append(needQueren[j])
        if PbsCarCur < m:
            if if_fanStar == 0: ###当返回道的10 车位为空时,
                #canRunChedao = [3,2,4,1,5,0] ###输入横移机可选择的部分
                canRunChedao.append('等待')
```

```

chooseChedao = renwu[i,0] * len(canRunChedao) ### 根据既定任务选择车道
chooseChedao = canRunChedao[int(chooseChedao)] ### 选择进哪个车道

if if_fanStar == 1:###返回车道不为空
    chooseChedao = renwu[i,0] * len(canRunChedao)    ###根据既定任务选择车道
    chooseChedao = canRunChedao[int(chooseChedao)]    ###选择进哪个车道

if chooseChedao != '等待': ### 如果不选择等待
    timeInputHyjNeed = self.time_in_out[chooseChedao] ### 所选择的车道需要花的时间
    timeInputHyjWorkCur = i ### 输入横移机开始工作的时间
    if PbsCarCur < m: ### 仓库内还有车
        if if_fanStar == 0: ### 如果返回道 10 车位没车
            ifInputHyj = 1 ### 立刻可以拿车，将横移机置为工作状态
            fromWhere = 0 ### 记录数据来源，从仓库来的数据
            result[PbsCarCur, i * 3:(i + 1) * 3] = 1 ### 记录数据
            inputHyjCar = PbsCarCur ### 输入横移机从仓库拿车，更新输入横移机上车的序号
            PbsCarCur += 1 ### 下一辆仓库的出车序号

if if_fanStar == 1:
    fromWhere = 1 ##### 记录数据来源，从反车道来的数据
    ifInputHyj = 0 ### 此时还没装车，为了方便处理所以将输入横移机的工作状态置为 0，实际此时为 1

if i >= timeFanArrive + 1: ### 输入横移机到达反车道 10 车位
    inputHyjCar = fanIndex[0] ### 输入横移机此时携带的车的序号
    result[int(inputHyjCar), i * 3:(i + 1) * 3] = 1 ### 记录结果
    fanIndex[0] = -1 ### 车被搬走，所以返回道 10 车位没有车，用-1 表示
    fanState[0] = 0 ### 车被搬走，返回道 10 车位没车，0 表示没车
    if_fanStar = 0 ### 车被搬走，返回道 10 车位没车，0 表示没车
    timeInputHyjWorkCur = i ### 输入横移机从返回道接受车的时间点
    ifInputHyj = 1 ### 输入横移机在工作

if ifInputHyj == 1: ### 如果输入横移机在工作
    if fromWhere == 0: ### 如果从仓库来的
        if inputHyjCar != -1:
            result[int(inputHyjCar), i * 3:(i + 1) * 3] = 1 ### 记录结果
        if i == (timeInputHyjWorkCur + timeInputHyjNeed / 2): ### 如果输入横移机送到目标车道
            inputHyjToChedaoGang.append(chooseChedao) ### 输入横移机刚刚把车放在指定车道
            chedaoState[chooseChedao, 0] = 1 ### 将所选择车道的 10 车位置为 1
            chedaoIndex[chooseChedao, 0] = inputHyjCar ### 更新车道上的车的序号
            result[inputHyjCar, i * 3:(i + 1) * 3] = (1 + chooseChedao) * 100 + 10 ### 记录结果
            inputHyjCar = -1 ### 输入横移机此时不再有车

```

```

if i >= timeInputHyjNeed + timeInputHyjWorkCur: ###输入横移机回到原来位置时
    ifInputHyj = 0 ###输入横移机工作转态置为0
    timeFanArrive = i ###修改的地方#####

if fromWhere == 1:
    if i == timeInputHyjWorkCur + self.time_fanToChedao[int(chooseChedao)]:
        chedaoState[chooseChedao, 0] = 1 ###从返回道的10车位来的车，由输入横移机放到选择的车道中
        chedaoIndex[chooseChedao, 0] = inputHyjCar ###更新被选择的车道上车的编号
        inputHyjToChedaoGang.append(chooseChedao) ###输入横移机刚刚把车放在指定车道
        result[int(inputHyjCar), i * 3 : i * 3 + 3] = chooseChedao * 100 + 10 ###更新结果
        inputHyjCar = -1 ###输入横移机上没有车

    if i >= self.time_fan[chooseChedao] - 1 + timeInputHyjWorkCur: ###输入横移车回到开始的位置时
        ifInputHyj = 0
        timeFanArrive = i ###修改的地方#####

if ifOutputHyj == 0 and len(arriveIndex) != 0: ###如果输出横移机空闲，且1车位内有车
    timeOutputHyjWorkCur = i ###输出横移机开始工作的时间
    timeOutputHyjNeed = self.time_in_out[arriveIndex[0]] / 2 ###输出横移机需要到达指定车道所花的时间
    timeOutputHyjChedaoToFan = self.time_fanToChedao[arriveIndex[0]] ###输出横移机从指定车道送到返回
    道的时间
    ifOutputHyj = 1 ###输出横移机工作状态置为1

if ifOutputHyj == 1:
    if i == timeOutputHyjWorkCur + timeOutputHyjNeed: ###输出横移机到达1车位
        ###到达1车位后对输出还是回返回道进行判断
        if fanState[25:].any() == 1:
            outOrInput = 1

        if fanState[25:].any() != 1:
            if renwu[i, 1] <= 0.5:
                outOrInput = 1
            if renwu[i, 1] > 0.5:
                outOrInput = 0
            ###0表示去返回道，1表示直接出去

result[int(chedaoIndex[int(arriveIndex[0]), -1]), i * 3 : i * 3 + 3] = 2 ###车到了输出横移机身上，记录结果
OutHyjCar = chedaoIndex[int(arriveIndex[0]), -1] ###输出横移机上面车的序号
chedaoState[int(arriveIndex[0]), -1] = 0 ###车道状态改变
chedaoIndex[int(arriveIndex[0]), -1] = -1 ###车道上对应车位的车的序号置为-1 (-1表示没有车)
arriveIndex.pop(0) ###到达最后一个车位的车道顺序里去除被接走的车

```

```

if i >= timeOutputHyjWorkCur + timeOutputHyjNeed:
    ###outOrInput 这个规则及约束还没写
if outOrInput == 0: ###把车送到返回道
    if i < timeOutputHyjWorkCur + timeOutputHyjNeed + timeOutputHyjChedaoToFan: ###车在输出横移
机上
        result[int(OutHyjCar), i * 3:i * 3 + 3] = 2

    if i == timeOutputHyjWorkCur + timeOutputHyjNeed + timeOutputHyjChedaoToFan: ###把车从输出横
移机放到返回道
        arrive10Chewei[int(OutHyjCar)] += 1 ###车到达返回道，这表明车已用掉一次返回的机会
        result[int(OutHyjCar), i * 3:i * 3 + 3] = 71
        outputHyjTofanGang.append(27) ###刚刚把车从输出横移机放到反车道
        fanState[-1] = 1 ###返回车道的1车位放车
        fanIndex[-1] = OutHyjCar ###返回车道上的1车位上车的序号
        OutHyjCar = -1 ###输出横移机不再有车
        fanNumber += 1

    if i == timeOutputHyjWorkCur + timeOutputHyjNeed + timeOutputHyjChedaoToFan + 1: ###输出横移
机结束工作
        ifOutputHyj = 0
        ###输出横移机送出去
        if outOrInput == 1:
            if i < timeOutputHyjWorkCur + 2 * timeOutputHyjNeed: ###输出横移机还未到达指定车道，车还在横
移机上
                result[int(OutHyjCar), i * 3:i * 3 + 3] = 2

            if i == timeOutputHyjWorkCur + 2 * timeOutputHyjNeed: ###输出横移机成功把车送出去
                result[int(OutHyjCar), i * 3:i * 3 + 3] = 3 ###更新记录
                outputLine.append(int(OutHyjCar)) ###输出队列添加上刚刚送出去的车
                ifOutputHyj = 0 ###输出横移机结束工作
                OutHyjCar = -1 ###输出横移机上没有车，置为-1

        ###车道能否移动判断矩阵
        for j in range(6):
            for j_index in range(9):
                if j_index == 0:
                    if chedaoState[j, -1] == 0: ###1 车位没车的话，这条道一定可以走
                        ifRunChedao[j] = 1
                        break

                if j_index > 0:
                    if chedaoState[j, 27 - j_index * 3:30 - j_index * 3].any() == 0: ###当前车位没车的话，后面的车一定可
以往前走

```



```

        ifRunChedao[j, :27 - j_index * 3] = 1
        break

ifRunChedao = np.append(ifRunChedao, np.zeros(shape=(6, 1)), axis=1) ###1 车位不能通过车道往前走
###返回道可否移动的判断矩阵
for j in range(9):
    if j == 0:
        if fanState[0] == 0: ###返回道 10 车位没车时，返回道都可移动
            ifRunFan[:] = 1
            break

    if j > 0:
        if fanState[j * 3 - 2 : j * 3 + 1].any() == 0: ###返回道当前车位没车时，小于该车位的车位可移动
            ifRunFan[j * 3:] = 1
            break

ifRunFan = np.append(np.zeros(1), ifRunFan) ###返回道 10 车位的车不可通过返回道移动

### 车道状态改变
haveCarCanGo = ifRunChedao * chedaoState ###有车且能走的车
if inputHyjToChedaoGang:
    haveCarCanGo[int(chooseChedao), 0] = 0 ###刚刚到车道的车不能走
    inputHyjToChedaoGang.pop() ###取消刚刚到车道这个‘刚刚’的状态
haveCarCanGoIndexOldChedao = np.where(haveCarCanGo == 1)[0]
haveCarCanGoIndexOldChewei = np.where(haveCarCanGo == 1)[1]
haveCarCanGoIndexNewChewei = haveCarCanGoIndexOldChewei + 1
chedaoState[haveCarCanGoIndexOldChedao, haveCarCanGoIndexNewChewei] = 1
chedaoState[haveCarCanGoIndexOldChedao, haveCarCanGoIndexOldChewei] = 0

### 调整车道上车的序号
chedaoIndex[haveCarCanGoIndexOldChedao, haveCarCanGoIndexNewChewei], chedaoIndex[
    haveCarCanGoIndexOldChedao, haveCarCanGoIndexOldChewei] = \
    chedaoIndex[haveCarCanGoIndexOldChedao, haveCarCanGoIndexOldChewei], chedaoIndex[
        haveCarCanGoIndexOldChedao, haveCarCanGoIndexNewChewei]
result = self.chedaoOneToOne(chedaoIndex, result, i) ###记录车道上车的位置
chedaoLastChewei = chedaoState[:, -1] ###1 车位上的状态
chedaoLastChewei = np.where(chedaoLastChewei == 1)[0] ###1 车位的有车车道

for j in chedaoLastChewei:
    if j not in arriveIndex: ###已经在到达 1 车位的队列则不再添加
        arriveIndex.append(j) ###到达 1 车位的队列添加新到达的

### 反车道状态改变
haveFanCarCanGo = ifRunFan * fanState ###反车道上有车且能走的车

```

```

if outputHyjTofanGang:
    haveFanCarCanGo[-1] = 0 ###刚刚到反车道的车不能动
    outputHyjTofanGang.pop() ###取消刚刚到反车道的这个‘刚刚的’状态
haveCarCanGoIndexOldfan = np.where(haveFanCarCanGo == 1)[0] ###反车道上有车的车位
haveCarCanGoIndexNewfan = haveCarCanGoIndexOldfan - 1 ###反车道上的车移动后的车位
fanState[haveCarCanGoIndexOldfan] = 0
fanState[haveCarCanGoIndexNewfan] = 1

if fanState[0] == 1: ###返回道10 车位有车时
    if_fanStar = 1 ###返回道10 车位修改为有车

fanIndex[haveCarCanGoIndexOldfan], fanIndex[haveCarCanGoIndexNewfan] = \
    fanIndex[haveCarCanGoIndexNewfan], fanIndex[haveCarCanGoIndexOldfan] ###调整反车道上车的序号
if fanState[0] == 1:
    if_fanStar = 1
if fanState[0] == 0:
    if_fanStar = 0

if inputHyjCar != -1:
    result[int(inputHyjCar), i * 3:i * 3 + 3] = 1

if OutHyjCar != -1:
    result[int(OutHyjCar), i * 3:i * 3 + 3] = 2

result = self.fanOneToOne(fanIndex, result, i) ###记录返回道上车的位置
result[outputLine, i * 3:i * 3 + 3] = 3 ###记录车出去后，车所在的位置

if i > 750:
    if len(np.where(result[:, i * 3] != 3)[0]) == 0:
        result = result[:, :(i + 1) * 3]
        break

if i == maxiter-2:
    #print(0)
    #print(i)
    self.X[cur] = np.random.uniform(low=self.lb, high=self.ub, size=(self.dim,))
    renwu = self.X[cur]
    renwu = renwu.reshape(-1, 2)
    i = 0
    outputLine = [] ###输出队列
    inputHyjToChedaoGang = [] ###刚刚从输入横移机放入车道
    outputHyjTofanGang = [] ###刚刚从输出横移机放入反车道
    result = np.zeros(shape=(m, int(maxiter * 3)))
    chedaoIndex = np.zeros(shape=(6, 28)) - 1 ###车道上车的编号，没车就是-1

```

```

fanIndex = np.zeros(28) - 1 ###返回道上车位的编号，没车为-1
chedaoState = np.zeros(shape=(6, 28)) ###车道上有无车，有车1，没车0
fanState = np.zeros(28) ###反车道上的状态，有车1，没车0
fanNunber = 0 ###使用返回道次数
resultHyjStarEnd = np.zeros(shape=(m, 5)) ###0 还没出，1 在横移机上，2 在车道上，3 在输出横移机，4
在终点
resultHyjStarEnd[:, 0] = 1
ifStar = np.zeros(m)
ifStar[0] = 1 ###
if_fanStar = 0 ###返回车道的第10 车位状态，0 为空，1 为满
ifInputHyj = 0 ###入口横移机是否在工作，0 为空，1 为满
ifOutputHyj = 0 ###出口横移机是否在工作，0 为空，1 为满
timeFanArrive = 0 ###返回道到达时间
fromWhere = 0 ###输入横移机上的车的来源，0 表示 PBS，1 表示返回道
arriveIndex = [] ###到达最后一个车位的车道顺序
outOrInput = 0 ###横移机准备把车送出去还是送回返回道，0 表示去返回道，1 表示送出去
inputHyjCar = -1 ###初始化输入横移机上的车的编号，-1 表示没有车
OutHyjCar = -1 ###初始化输出横移机上的车编号，没车表示-1
PbsCarCur = 0 ###初始化 PBS->输入横移机上的编号，第一辆为0
arrive10Chewei = np.zeros(m)

i += 1

return result, outputLine, fanNunber, result.shape[1]

```

```

def run(self, data):
    t1 = time.time()
    for i in range(self.maxiter):
        self.fitness(data)
        self.gen_best_y[i] = np.min(self.curY)
        self.gen_best_X[i, :] = self.alpha_wolf_pos
        self.getCurWolf(i)
        self.X = self.getNewX(i)
        #if i > 10 and self.gen_best_y[i] > self.gen_best_y[i-10]-0.0001:
        #break
        if i % 10 == 0:
            print(i)
            t2 = time.time()
            if t2-t1 > 6000:
                break

    self.fitness(data)
    self.gen_best_y[i+1] = np.min(self.curY)

```

```

self.gen_best_y = self.gen_best_y[:i+1]
self.gen_best_X[i+1, :] = self.alpha_wolf_pos
self.gen_best_X = self.gen_best_X[:i+1,:]
index = np.argmin(self.curY)
best_x = self.X[index]
best_y = self.curY[index]
gen_min_y_index = self.gen_best_y.argmax()
self.gen_min_y = self.gen_best_y[gen_min_y_index]
self.gen_min_x = self.gen_best_X[gen_min_y_index]
return best_x,best_y

```

```

model = GWO(maxiter=400)
best_x,best_y = model.run(data_1)
plt.plot(model.gen_best_y)
plt.show()

```

```

result, outputLine, fanNunber,costTime=model.genX(maxiter=model.dim/2,data=data_1,cur=22,renwu=model.gen_min_x)

```

```

result = pd.DataFrame(result)
result.to_excel('附件 1 的结果.xlsx')

```

```

model = GWO(maxiter=400)
best_x,best_y = model.run(data_2)
plt.plot(model.gen_best_y)
plt.show()

```

```

result, outputLine, fanNunber,costTime=model.genX(maxiter=model.dim/2,data=data_1,cur=22,renwu=model.gen_min_x)

```

```

result = pd.DataFrame(result)
result.to_excel('附件 2 的结果.xlsx')

```

问题 2

```

import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt

```

```

data_1 = pd.read_excel('附件 1.xlsx')
data_2 = pd.read_excel('附件 2.xlsx')
data_1 = np.array(data_1)
data_2 = np.array(data_2)
data_1[:,2] = np.where(data_1[:,2]=='燃油',0,1)
data_1[:,3] = np.where(data_1[:,3]=='两驱',0,1)
data_2[:,2] = np.where(data_2[:,2]=='燃油',0,1)
data_2[:,3] = np.where(data_2[:,3]=='两驱',0,1)

```

```

class GWO():
    def __init__(self,dim=10000,maxiter=200,size=40,lb=0,ub=0.999999999):
        self.dim = dim
        self.X = np.random.uniform(low=lb,high=ub,size=(size,dim))
        self.alpha_wolf_pos = None
        self.beta_wolf_1_pos = None
        self.beta_wolf_2_pos = None
        self.maxiter = maxiter
        self.size = size
        self.lb = lb
        self.ub = ub
        self.gen_best_y = np.zeros(shape=(self.maxiter+1,))
        self.gen_best_X = np.zeros(shape=(self.maxiter+1,dim))
        self.gen_min_y = None
        self.gen_min_x = None
        self.time_in_out = np.array([18, 12, 6, 0, 12, 18]) / 3
        self.time_fan = np.array([24, 18, 12, 6, 12, 18]) / 3
        self.timeChedaoToHyj = self.time_in_out / 2
        self.time_fanToChedao = np.array([4, 3, 2, 1, 1, 2])
        self.time_0 = 3
        self.gold1 = None
        self.gold2 = None
        self.gold3 = None
        self.gold4 = None

    def fitness(self,data):
        self.curY = np.zeros(shape=(self.size,))
        for i in range(self.size):
            ans, outLine, needNunberFan, costtime = self.genX(maxiter=self.dim/4, data=data, renwu=self.X[i], cur=i)
            curGold, self.gold1, self.gold2, self.gold3, self.gold4 = self.getGold(outLine, needNunberFan, data, costtime)
            self.curY[i] = -1*curGold

    def getCurWolf(self,t):
        index = self.curY.argsort()

```

```

self.alpha_wolf_pos = self.X[index[0]]
self.beta_wolf_1_pos = self.X[index[1]]
self.beta_wolf_2_pos = self.X[index[2]]

```

```
def getNewX(self,t):
```

```

    r_1_a = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    r_1_b_1 = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    r_1_b_2 = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    r_2_a = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    r_2_b_1 = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    r_2_b_2 = np.random.uniform(low=0,high=1,size=(self.size,self.dim))
    a = 2*(1-t/self.maxiter)
    A_a = 2*a*r_1_a-a
    A_b_1 = 2*a*r_1_b_1-a
    A_b_2 = 2 * a * r_1_b_2 - a
    C_a = 2*r_2_a
    C_b_1 = 2*r_2_b_1
    C_b_2 = 2*r_2_b_2
    distance_a = np.abs(C_a * self.alpha_wolf_pos - self.X)
    distance_b_1 = np.abs(C_b_1 * self.beta_wolf_1_pos - self.X)
    distance_b_2 = np.abs(C_b_2 * self.beta_wolf_2_pos - self.X)
    X_a = self.alpha_wolf_pos - A_a * distance_a
    X_b_1 = self.beta_wolf_1_pos - A_b_1 * distance_b_1
    X_b_2 = self.beta_wolf_2_pos - A_b_2 * distance_b_2
    new_X = (X_a+X_b_1+X_b_2)/3
    new_X = np.where(new_X<self.lb,self.lb,new_X)
    new_X = np.where(new_X>self.ub,self.ub,new_X)
    return new_X

```

```
def chedaoOneToOne(self,x, result, time_i):
```

```

    xAfter0 = x[:, 3:-1]
    xIndex = np.where(xAfter0 != -1)
    curVar = (xIndex[0].reshape(-1, 1) + 1) * 10 + 10 - (xIndex[1].reshape(-1, 1) + 3) // 3
    curVar = curVar.reshape(-1, 1)
    result[xAfter0[xIndex[0]], xIndex[1]].astype(int), time_i * 3:time_i * 3 + 3 = curVar
    xIndex = np.where(x[:, :3] != -1)
    result[x[xIndex[0]], xIndex[1]].astype(int), time_i * 3:time_i * 3 + 3 = (xIndex[0].reshape(-1,
                                                                                               1) + 1) * 100 + 10
    xIndex = np.where(x[:, -1] != -1)
    result[x[xIndex[0]], -1].astype(int), time_i * 3:time_i * 3 + 3 = (xIndex[0].reshape(-1, 1) + 1) * 10 + 1
    return result

```

```
def fanOneToOne(self,x, result, time_i): ###x 表示每个车道车位上对应的车的编号, -1 为没车
```

```

    curVar = x[0]

```

```

if curVar != -1:
    result[int(curVar), time_i * 3:time_i * 3 + 3] = 710 ###更新反车道 10 车位

curVar = x[1:]
curVarIndex = np.where(curVar != -1)[0] ###找到有车的车位
result[curVar[curVarIndex].astype(int), time_i * 3:time_i * 3 + 3] = 70 + 9 - curVarIndex.reshape(-1,
1) // 3 ###更新反车道 1-9 车位

return result

def Outputresult(self,x):
    x_Index = np.where(x != 3)[1].max()
    return x[:, :x_Index + 3]

def getGold(self,x, number, data, costTime):
    l = len(x)
    data = data[:, 2:]
    x_list = []
    for i in range(l):
        x_list.append(data[x[i], 1])
    item2 = self.z2(x_list)
    x_list = []
    for i in range(l):
        x_list.append(data[x[i],0])
    item1 = self.z1(x_list)
    result = (100-item1)*0.4+(100-item2)*0.3+(100-number)*0.2+0.1*(100-(costTime-2934)*0.01)
    return result,(100-item1)*0.4,(100-item2)*0.3,(100-number)*0.2,0.1*(100-(costTime-2934)*0.01)

def z1(self,data):
    result = 0
    cc = 0
    data = np.array(data)
    a = np.where((data == 1))
    for i in range(1, len(a[0])):
        if a[0][i] - a[0][i - 1] == 3:
            result = result + 1
        else:
            cc = cc + 1
    rrr = cc
    return rrr

def z2(self,list1):
    result = [0]
    rr = []
    flag = 0

```

```

j = 1
rrr = 0
R = 0
for i in range(1, len(list1)):
    if list1[i] == list1[i - 1]:
        flag += 1
    else:
        flag = 0
    result.append(flag)
result.append(0)
for i in range(1, len(result)):
    if result[i] <= result[i - 1]:
        rr.append(result[i - 1] + 1)

while j < len(rr):
    if rr[j] == rr[j - 1]:
        rrr = rrr + 1

    else:
        R = R + 1
    j = j + 2
if len(rr) % 2:
    R = R + 1
return R

```

```

def genX(self,renwu,maxiter, data,cur):
    renwu = renwu.reshape(-1,4)
    m, n = data.shape  ###数据长度
    outputLine = []  ###输出队列
    inputHyjToChedaoGang = []  ###刚刚从输入横移机放入车道
    outputHyjTofanGang = []  ###刚刚从输出横移机放入反车道
    result = np.zeros(shape=(m, int(maxiter * 3)))
    chedaoIndex = np.zeros(shape=(6, 28)) - 1  ###车道上车的编号，没车就是-1
    fanIndex = np.zeros(28) - 1  ###返回道上车位的编号，没车为-1
    chedaoState = np.zeros(shape=(6, 28))  ###车道上有无车，有车1，没车0
    fanState = np.zeros(28)  ###反车道上的状态，有车1，没车0
    fanNumber = 0  ###使用返回道次数
    resultHyjStarEnd = np.zeros(shape=(m, 5))  ###0 还没出，1 在横移机上，2 在车道上，3 在输出横移机，4 在终
点
    resultHyjStarEnd[:, 0] = 1
    ifStar = np.zeros(m)
    ifStar[0] = 1  ###
    if_fanStar = 0  ###返回车道的第10车位状态，0 为空，1 为满
    ifInputHyj = 0  ###入口横移机是否在工作，0 为空，1 为满

```



```

ifOutputHyj = 0 ###出口横移机是否在工作, 0 为空, 1 为满
timeFanArrive = 0 ###返回道到达时间
fromWhere = 0 ###输入横移机上的车的来源, 0 表示 PBS, 1 表示返回道
arriveIndex = [] ###到达最后一个车位的车道顺序
outOrInput = 0 ###横移机准备把车送出去还是送回返回道, 0 表示去返回道, 1 表示送出去
inputHyjCar = -1 ###初始化输入横移机上的车的编号, -1 表示没有车
OutHyjCar = -1 ###初始化输出横移机上的车编号, 没车表示-1
PbsCarCur = 0 ###初始化 PBS->输入横移机上的编号, 第一辆为0
arrive10Chewei = np.zeros(m)
i = 0
while i < maxiter:
    ifRunChedao = np.zeros(shape=(6, 27)) ###可移动的车道,1 表示可移动, 0 表示不可移动,这得对约束的理解
    进行修改
    ifRunFan = np.zeros(27) ###返回道是否可移动,1 表示可移动, 0 表示不可移动
    if ifInputHyj == 0: ###如果输入横移机空闲

        ifRunChedao_0 = chedaoState[:, :3] ###10 车位上的车道状态
        ifRunChedao_0 = np.where(ifRunChedao_0 == 1) ###10 车位的索引
        needQueren = []
        canRunChedao = [3,2,4,1,5,0] ###可选择车道
        for j in range(6):
            if j in ifRunChedao_0[0]:
                canRunChedao.remove(j) ###剩下一定能走的车道
                needQueren.append(j) ###需要计算时间确认的车道
            ###这里根据来源不同进行不同的计算可选择车道
            ###计算可选择车道这里有问题
            # if if_fanStar == 1: ###10 号车位返回道有车
            #     if needQueren:
            #         for j in range(len(needQueren)):
            #             if time_fanToChedao[needQueren[j]] > 3 - ifRunChedao_0[needQueren[j], 1]:
            #                 canRunChedao.append(needQueren[j])
        if PbsCarCur < m:
            if if_fanStar == 0: ###当返回道的 10 车位为空时,
                #canRunChedao = [3,2,4,1,5,0] ###输入横移机可选择的部分
                #canRunChedao.append('等待')
                chooseChedao = renwu[i,2] * len(canRunChedao) ###根据既定任务选择车道
                chooseChedao = canRunChedao[int(chooseChedao)] ###选择进哪个车道

            if if_fanStar == 1:###返回车道不为空
                chooseChedao = renwu[i,2] * len(canRunChedao) ###根据既定任务选择车道
                chooseChedao = canRunChedao[int(chooseChedao)] ###选择进哪个车道

```

```

if chooseChedao != '等待': ###如果不选择等待
    timeInputHyjNeed = self.time_in_out[chooseChedao] ###所选择的车道需要花的时间
    timeInputHyjWorkCur = i ###输入横移机开始工作的时间
    if PbsCarCur < m: ###仓库内还有车
        if renwu[i,0] >0.5 : ###指定任务状态
            ifInputHyj = 1 ###立刻可以拿车，将横移机置为工作状态
            fromWhere = 0 ###记录数据来源，从仓库来的数据
            result[PbsCarCur, i * 3:(i + 1) * 3] = 1 ###记录数据
            inputHyjCar = PbsCarCur ###输入横移机从仓库拿车，更新输入横移机上车的序号
            PbsCarCur += 1 ###下一辆仓库的出车序号

        if renwu[i,0] <0.5 : ###接返回车道的车
            fromWhere = 1 #####记录数据来源，从反车道来的数据
            ifInputHyj = 1 ###此时还没装车，为了方便处理所以将输入横移机的工作状态置为1

if i == timeFanArrive + 1: ###输入横移机到达反车道 10 车位
    inputHyjCar = fanIndex[0] ###输入横移机此时携带的车的序号
    result[int(inputHyjCar), i * 3:(i + 1) * 3] = 1 ###记录结果
    fanIndex[0] = -1 ###车被搬走，所以返回道 10 车位没有车，用-1 表示
    fanState[0] = 0 ###车被搬走，返回道 10 车位没车，0 表示没车
    if_fanStar = 0 ###车被搬走，返回道 10 车位没车，0 表示没车
    timeInputHyjWorkCur = i ###输入横移机从返回道接受车的时间点
    fromWhere = 1 ###车从返回道来
    ifInputHyj = 1 ###输入横移机在工作

if ifInputHyj == 1: ###如果输入横移机在工作
    if fromWhere == 0: ###如果从仓库来的
        if inputHyjCar != -1:
            result[int(inputHyjCar), i * 3:(i + 1) * 3] = 1 ###记录结果
        if i == (timeInputHyjWorkCur + timeInputHyjNeed / 2): ###如果输入横移机送到目标车道
            inputHyjToChedaoGang.append(chooseChedao) ###输入横移机刚刚把车放在指定车道
            chedaoState[chooseChedao, 0] = 1 ###将所选择车道的 10 车位置为 1
            chedaoIndex[chooseChedao, 0] = inputHyjCar ###更新车道上的车的序号
            result[inputHyjCar, i * 3:(i + 1) * 3] = (1 + chooseChedao) * 100 + 10 ###记录结果
            inputHyjCar = -1 ###输入横移机此时不再有车

        if i >= timeInputHyjNeed + timeInputHyjWorkCur: ###输入横移机回到原来位置时
            ifInputHyj = 0 ###输入横移机工作转态置为 0
            timeFanArrive = i ###修改的地方#####

    if fromWhere == 1:
        if i == timeInputHyjWorkCur + self.time_fanToChedao[int(chooseChedao)]:
            chedaoState[chooseChedao, 0] = 1 ###从返回道的 10 车位来的车，由输入横移机放到选择的车道中

```

```

chedaoIndex[chooseChedao, 0] = inputHyjCar ###更新被选择的车道上车的编号
inputHyjToChedaoGang.append(chooseChedao) ###输入横移机刚刚把车放在指定车道
result[int(inputHyjCar), i * 3:i * 3 + 3] = chooseChedao * 100 + 10 ###更新结果
inputHyjCar = -1 ###输入横移机上没有车

if i >= self.time_fan[chooseChedao] - 1 + timeInputHyjWorkCur: ###输入横移车回到开始的位置时
    ifInputHyj = 0
    timeFanArrive = i ###修改的地方#####

if ifOutputHyj == 0 and len(arriveIndex) != 0: ###如果输出横移机空闲，且1车位内有车
    timeOutputHyjWorkCur = i ###输出横移机开始工作的时间
    choosefan = renwu[i,3]*len(arriveIndex)
    choosefan = int(choosefan)
    timeOutputHyjNeed = self.time_in_out[choosefan] / 2 ###输出横移机需要到达指定车道所花的时间
    timeOutputHyjChedaoToFan = self.time_fanToChedao[choosefan] ###输出横移机从指定车道送到返回道的
时间
    ifOutputHyj = 1 ###输出横移机工作状态置为1

if ifOutputHyj == 1:
    if i == timeOutputHyjWorkCur + timeOutputHyjNeed: ###输出横移机到达1车位
        ###到达1车位后对输出还是回返回道进行判断
        if fanState[25:].any() == 1:
            outOrInput = 1

        if fanState[25:].any() != 1:
            if renwu[i,1] <=0.5:
                outOrInput = 1
            if renwu[i,1] >0.5:
                outOrInput = 0 ###0表示去返回道，1表示直接出去

    result[int(chedaoIndex[int(arriveIndex[0]), -1]), i * 3:i * 3 + 3] = 2 ###车到了输出横移机身上，记录结果
    OutHyjCar = chedaoIndex[int(arriveIndex[0]), -1] ###输出横移机上面车的序号
    chedaoState[int(arriveIndex[0]), -1] = 0 ###车道状态改变
    chedaoIndex[int(arriveIndex[0]), -1] = -1 ###车道上对应车位的车的序号置为-1 (-1表示没有车)
    arriveIndex.pop(0) ###到达最后一个车位的车道顺序里去除被接走的车

if i >= timeOutputHyjWorkCur + timeOutputHyjNeed:
    ###outOrInput 这个规则及约束还没写
    if outOrInput == 0: ###把车送到返回道
        if i < timeOutputHyjWorkCur + timeOutputHyjNeed + timeOutputHyjChedaoToFan: ###车在输出横移
机上
            result[int(OutHyjCar), i * 3:i * 3 + 3] = 2

```

if i == timeOutputHyjWorkCur + timeOutputHyjNeed + timeOutputHyjChedaoToFan: ###把车从输出横移机放到返回道

```
arrive10Chewei[int(OutHyjCar)] += 1 ###车到达返回道，这表明车已用掉一次返回的机会
result[int(OutHyjCar), i * 3:i * 3 + 3] = 71
outputHyjTofanGang.append(27) ###刚刚把车从输出横移机放到反车道
fanState[-1] = 1 ###返回车道的1车位放车
fanIndex[-1] = OutHyjCar ###返回车道上的1车位上车的序号
OutHyjCar = -1 ###输出横移机不再有车
fanNumber += 1
```

if i == timeOutputHyjWorkCur + timeOutputHyjNeed + timeOutputHyjChedaoToFan + 1: ###输出横移机结束工作

```
ifOutputHyj = 0
###输出横移机送出去
```

```
if outOrInput == 1:
```

if i < timeOutputHyjWorkCur + 2 * timeOutputHyjNeed: ###输出横移机还未到达指定车道，车还在横移机上

```
result[int(OutHyjCar), i * 3:i * 3 + 3] = 2
```

if i == timeOutputHyjWorkCur + 2 * timeOutputHyjNeed: ###输出横移机成功把车送出去

```
result[int(OutHyjCar), i * 3:i * 3 + 3] = 3 ###更新记录
outputLine.append(int(OutHyjCar)) ###输出队列添加上刚刚送出去的车
ifOutputHyj = 0 ###输出横移机结束工作
OutHyjCar = -1 ###输出横移机上没有车，置为-1
```

###车道能否移动判断矩阵

```
for j in range(6):
```

```
for j_index in range(9):
```

```
if j_index == 0:
```

```
if chedaoState[j, -1] == 0: ###1 车位没车的话，这条道一定可以走
    ifRunChedao[j] = 1
    break
```

```
if j_index > 0:
```

if chedaoState[j, 27 - j_index * 3:30 - j_index * 3].any() == 0: ###当前车位没车的话，后面的车一定可以往前走

```
ifRunChedao[j, :27 - j_index * 3] = 1
break
```

ifRunChedao = np.append(ifRunChedao, np.zeros(shape=(6, 1)), axis=1) ###1 车位不能通过车道往前走
###返回道可否移动的判断矩阵

```
for j in range(9):
```

```

if j == 0:
    if fanState[0] == 0: ###返回道 10 车位没车时，返回道都可移动
        ifRunFan[:] = 1
        break

if j > 0:
    if fanState[j * 3 - 2: j * 3 + 1].any() == 0: ###返回道当前车位没车时，小于该车位的车位可移动
        ifRunFan[j * 3:] = 1
        break

ifRunFan = np.append(np.zeros(1), ifRunFan) ###返回道 10 车位的车不可通过返回道移动

### 车道状态改变
haveCarCanGo = ifRunChedao * chedaoState ###有车且能走的车
if inputHyjToChedaoGang:
    haveCarCanGo[int(chooseChedao), 0] = 0 ###刚刚到车道的车不能走
    inputHyjToChedaoGang.pop() ###取消刚刚到车道这个‘刚刚’的状态
haveCarCanGoIndexOldChedao = np.where(haveCarCanGo == 1)[0]
haveCarCanGoIndexOldChewei = np.where(haveCarCanGo == 1)[1]
haveCarCanGoIndexNewChewei = haveCarCanGoIndexOldChewei + 1
chedaoState[haveCarCanGoIndexOldChedao, haveCarCanGoIndexNewChewei] = 1
chedaoState[haveCarCanGoIndexOldChedao, haveCarCanGoIndexOldChewei] = 0

### 调整车道上车的序号
chedaoIndex[haveCarCanGoIndexOldChedao, haveCarCanGoIndexNewChewei], chedaoIndex[
    haveCarCanGoIndexOldChedao, haveCarCanGoIndexOldChewei] = \
    chedaoIndex[haveCarCanGoIndexOldChedao, haveCarCanGoIndexOldChewei], chedaoIndex[
        haveCarCanGoIndexOldChedao, haveCarCanGoIndexNewChewei]
result = self.chedaoOneToOne(chedaoIndex, result, i) ###记录车道上车的位置
chedaoLastChewei = chedaoState[:, -1] ###1 车位上的状态
chedaoLastChewei = np.where(chedaoLastChewei == 1)[0] ###1 车位的有车

for j in chedaoLastChewei:
    if j not in arriveIndex: ###已经在到达 1 车位的队列则不再添加
        arriveIndex.append(j) ###到达 1 车位的队列添加新到达的

### 反车道状态改变
haveFanCarCanGo = ifRunFan * fanState ###反车道上有车且能走的车
if outputHyjTofanGang:
    haveFanCarCanGo[-1] = 0 ###刚刚到反车道的车不能动
    outputHyjTofanGang.pop() ###取消刚刚到反车道的这个‘刚刚的’状态
haveCarCanGoIndexOldfan = np.where(haveFanCarCanGo == 1)[0] ###反车道上有车的车位
haveCarCanGoIndexNewfan = haveCarCanGoIndexOldfan - 1 ###反车道上的车移动后的车位
fanState[haveCarCanGoIndexOldfan] = 0

```

```

fanState[haveCarCanGoIndexNewfan] = 1

if fanState[0] == 1: ###返回道10 车位有车时
    if_fanStar = 1 ###返回道10 车位修改为有车

fanIndex[haveCarCanGoIndexOldfan], fanIndex[haveCarCanGoIndexNewfan] = \
    fanIndex[haveCarCanGoIndexNewfan], fanIndex[haveCarCanGoIndexOldfan] ###调整反车道上车的序号
if fanState[0] == 1:
    if_fanStar = 1
if fanState[0] == 0:
    if_fanStar = 0

if inputHyjCar != -1:
    result[int(inputHyjCar), i * 3:i * 3 + 3] = 1

if OutHyjCar != -1:
    result[int(OutHyjCar), i * 3:i * 3 + 3] = 2

result = self.fanOneToOne(fanIndex, result, i) ###记录返回道上车的位置
result[outputLine, i * 3:i * 3 + 3] = 3 ###记录车出去后, 车所在的位置

if i > 750:
    if len(np.where(result[:, i * 3] != 3)[0]) == 0:
        result = result[:, :(i + 1) * 3]
        print('运行成功')
        break

if i == maxiter-2:
    #print(0)
    #print(i)
    print('运行失败')
    self.X[cur] = np.random.uniform(low=self.lb, high=self.ub, size=(self.dim,))
    renwu = self.X[cur]
    renwu = renwu.reshape(-1, 4)
    i = 0
    outputLine = [] ###输出队列
    inputHyjToChedaoGang = [] ###刚刚从输入横移机放入车道
    outputHyjTofanGang = [] ###刚刚从输出横移机放入反车道
    result = np.zeros(shape=(m, int(maxiter * 3)))
    chedaoIndex = np.zeros(shape=(6, 28)) - 1 ###车道上车的编号, 没车就是-1
    fanIndex = np.zeros(28) - 1 ###返回道上车位的编号, 没车为-1
    chedaoState = np.zeros(shape=(6, 28)) ###车道上有无车, 有车1, 没车0
    fanState = np.zeros(28) ###反车道上的状态, 有车1, 没车0
    fanNumber = 0 ###使用返回道次数

```

resultHyjStarEnd = np.zeros(shape=(m, 5)) ###0 还没出, 1 在横移机上, 2 在车道上, 3 在输出横移机, 4 在终点

```
resultHyjStarEnd[:, 0] = 1
ifStar = np.zeros(m)
ifStar[0] = 1 ###
if_fanStar = 0 ###返回车道的第10车位状态, 0 为空, 1 为满
ifInputHyj = 0 ###入口横移机是否在工作, 0 为空, 1 为满
ifOutputHyj = 0 ###出口横移机是否在工作, 0 为空, 1 为满
timeFanArrive = 0 ###返回道到达时间
fromWhere = 0 ###输入横移机上的车的来源, 0 表示 PBS, 1 表示返回道
arriveIndex = [] ###到达最后一个车位的车道顺序
outOrInput = 0 ###横移机准备把车送出去还是送回返回道, 0 表示去返回道, 1 表示送出去
inputHyjCar = -1 ###初始化输入横移机上的车的编号, -1 表示没有车
OutHyjCar = -1 ###初始化输出横移机上的车编号, 没车表示-1
PbsCarCur = 0 ###初始化 PBS->输入横移机上的编号, 第一辆为0
arrive10Chewei = np.zeros(m)
```

```
i += 1
```

```
return result, outputLine, fanNumber, result.shape[1]
```

```
def run(self, data):
    t1 = time.time()
    for i in range(self.maxiter):
        self.fitness(data)
        self.gen_best_y[i] = np.min(self.curY)
        self.gen_best_X[i, :] = self.alpha_wolf_pos
        self.getCurWolf(i)
        self.X = self.getNewX(i)
        #if i > 10 and self.gen_best_y[i] > self.gen_best_y[i-10]-0.0001:
        #break
        if i % 10 == 0:
            print(i)
            t2 = time.time()
            if t2-t1 > 6000:
                break

    self.fitness(data)
    self.gen_best_y[i+1] = np.min(self.curY)
    self.gen_best_y = self.gen_best_y[:i+1]
    self.gen_best_X[i+1, :] = self.alpha_wolf_pos
    self.gen_best_X = self.gen_best_X[:i+1, :]
    index = np.argmin(self.curY)
```

```

best_x = self.X[index]
best_y = self.curY[index]
gen_min_y_index = self.gen_best_y.argmin()
self.gen_min_y = self.gen_best_y[gen_min_y_index]
self.gen_min_x = self.gen_best_X[gen_min_y_index]
return best_x,best_y

```

```

model = GWO(maxiter=400,dim=10000)
best_x,best_y = model.run(data_1)
plt.plot(model.gen_best_y)
plt.show()

```

```

result, outputLine, fanNunber,costTime=model.genX(maxiter=model.dim/4,data=data_1,cur=22,renwu=model.gen_min_x)
result = pd.DataFrame(result)
result.to_excel('result21.xlsx')

```

```

model = GWO(maxiter=400,dim=10000)
best_x,best_y = model.run(data_2)
plt.plot(model.gen_best_y)
plt.show()

```

```

result, outputLine, fanNunber,costTime=model.genX(maxiter=model.dim/4,data=data_1,cur=22,renwu=model.gen_min_x)
result = pd.DataFrame(result)
result.to_excel('result22.xlsx')

```