机场的出租车问题

摘要

出租车司机在送乘客抵达机场后需要决定是否留在机场排队载客，司机对于当前机场打车需求的准确判断可以提高自己的收益；管理部门对于出租车的良好调度有利于提高乘客的乘车效率，平衡司机之间的收益差距。本文针对四个问题，分别建立了出租车司机的决策模型、机场出租车上客区模型与短途司机优先权分配模型来解决机场的出租车问题。

针对问题一，司机根据个人利益来进行决策，通过对司机进行去留两个决策的收益组成与时间成本进行比较与整理，分析出司机的收益与司机的等待时间有关。分析影响司机等待时间的有关因素，即为司机所能观测到的信息：近期抵达机场的航班信息、蓄车池中的车辆数量以及当前的机场天气状况。从以上影响因素出发，对司机在蓄车池中等待的时间进行预测，通过动态规划可以模拟出乘客队列长度变化，再对乘客队列预测模型进行训练，则可以通过队列长度以及蓄车池中的车辆数量对司机的等待时间进行预测。

针对问题二，我们收集了白云机场 9 月 13、14 日两天的全天航班数据和广州市出租车收费相关的信息。我们分别用 $\mu = 30$ 分钟和 $\mu = 120$ 分钟的正态分布函数来模拟抵达机场航班和即将起飞航班的客流量随时间的变化，结合白云机场乘客抵离行为特征来对乘出租车的旅客流量进行计算。用机场 13 日的客流数据拟合出旅客队列长度随航班信息特征数据的变化曲线，用于预测乘客队列长度。司机将结合蓄车池车辆数量与航班信息输入模型，模型通过预测的出乘客队列长度来计算司机预计等待时间进行决策。将 14 日的数据带入此模型和已知队列长度的决策模型，观察司机决策的相似性，发现 87.92% 的司机做出了与准确结果相同的选择，证明了其合理性。通过对航班信息和蓄车池车辆数目的定量分析，我们得到抵达航班信息对决策的影响比出发航班信息对决策的影响更大，蓄车池中车辆数目对于决策的影响最大。

针对问题三，乘车区变为并行双车道，但上车点位于道路的一侧，所以存在乘客要横穿车道的情况，要保证乘客与车辆安全，就要确保旅客横穿车道的行为和出租车驶入该车道的行为不同时发生。在模拟这一互斥操作的时候我们采用互斥锁技术来确保乘客和司机只有一个对象可以占有车道。对于上车点的位置、泊车位的个数的设置，我们设立出租车通行能力和泊车位空闲率两个指标，通过控制变量的方法模拟不同上车点和不同泊车位个数下指标指数，选择既保证乘客安全又提高上客区通行效果组合——每个车道有 5 个泊车位，最佳上车点位于车道得中部，此时上客区得通行能力为 480 辆/小时，车道最后停泊位得利用效率为 74.43%。

针对问题四，为使短途司机在载客返回市区所得的总收益同直接载客返回市区的司机在相同时间内所获收益相等，整理他们的收益与时间成本表达式可以得到短途载客司机在返回到机场还应等待的时间，此时出租车调度人员通过动态规划算法可以得到使得该司机可以等待相应时间的排名（优先权），该司机就可以直接从该位置排队，从而避免了收益的不均衡，使用该决策对白云机场的出租车在存在短途载客的情况下重新模拟，随机选出 100 个司机对其收益绘制散点图，得到一条比较稳定的直线，计算其方差为 6.54，说明此决策实现了收益均衡。

关键字： 动态规划 正态分布 互斥锁

一、 问题提出与重述

1.1 问题的提出

出租车是大多数乘客下飞机后前往目的地的主要交通工具之一，送客到机场的出租车司机会面临是否要前往蓄车池排队等待载客返回市区的选择。司机需要在时间成本和载客收益之间进行权衡，一个良好的决策可以帮助司机获得更多收益。对在机场等待的出租车司机进行良好的管理有利于提高交通运输效率与机场服务质量。

1.2 问题的重述

(1) 分析影响出租车司机决策的有关因素并探究其影响机理，结合出租车司机的收益、机场客流量变化等影响因素建立出租车司机的决策模型与选择策略。

(2) 通过收集某个城市中的机场及出租车的数据来训练出租车司机的决策模型，从而给出该机场的出租车司机的决策方案，并对模型的合理性和对有关因素的依赖性进行探究。

(3) 由于经常出现出租车排队载客以及乘客排队乘车的情况。某机场“乘车区”现有两条并行车道，管理部门应如何设置“上车点”，并合理安排出租车和乘客，在保证车辆和乘客安全的条件下，使得总的乘车效率最高。

(4) 管理部门拟对部分短途载客再次返回的出租车给予一定的“优先权”，使得这些出租车的收益尽量均衡，试给出一个合理的“优先”安排方案。

二、 问题分析

2.1 问题的分析

问题一分析：

对于影响出租车司机的决策的相关因素，从司机决策的出发点，即司机的个人收益出发，可以通过司机直接观察得到的航班信息、蓄车池中车的数量等信息，推测出部分司机位置的信息，分析司机的已知信息以及未知信息，从而得出影响出租车司机决策的有关因素以及影响机理。对于司机是否进入蓄车池等待载客的决策，可以根据分析得到的影响司机决策的因素来进行建模，通过模拟司机进入蓄车池载客到返程所得收益同司机立即返回市区所得收益进行比较，选择使得司机收益最大的方案。

问题二分析：收集国内某机场及其城市出租车的相关数据来生成该机场出租车司机的决策方案，由于机场的流量数据不易获取，我们可以通过收集机场一天的航班信息，通过对旅客行为模式的分析得到机场一天随时间变化的客流量数据，进而得到乘坐出租车的旅客流量。在得到旅客流量数据后，可以通过流量数据模拟出预测模型的训练数据，用来训练司机的决策模型。最终通过带入新数据检验和定量分析来探究模型的合理性与对相关元素的依赖性。

问题三分析：

问题三要求针对有两条并行车道的机场乘车区进行“上车点”的设置，并合理安排出租车和乘客，在保证安全的前提下尽可能提高效率。首先要明确两条并行车道的设计结构，有并行车道就会存在乘客横穿车道的情况，若想保证乘客与车的安全，就要设计策略使得乘客横穿车道与车的同行不能同时出现在相同的车道上，同时只有一条车道上的车停稳时，才能放乘客进入乘车去。对于上车点的设置，需要考虑同一条车道上所有车距离上车点的距离，使得乘客走向所有车的距离大致相同。若要尽可能提高上客区通行效率，就需要权衡泊车位的数量，使得“上车点”通行能力强并且泊车位空闲率低。

问题四分析：

问题四要求给予某些短途载客再次返回的出租车一定的“优先权”，使得出租车的收益尽量均衡。对于那些跑短途的出租车司机，假设他们在跑完短途回到机场，获得一定的优先权后接单返回市区，计算出这段时间该司机所获总收益，与在相同时间内其他未接到短途工作的司机所得收益进行比较，调整他们的优先权计算方式，使得这两种司机的所得收益尽可能相同。

2.2 模型的假设

(1) 司机从机场直接返回市区途中没有收益。
(2) 广州白云机场不会因意外事件突然停运。
(3) 司机的业务水平大致相同差距不大。

三、 符号说明

表 **1** 符号说明

符号	表示含义
$\overline{c_{city}}$	出租车在市内单位时间的平均收益
$\overline{c_{back}}$	出租车载客从机场到市内的平均收益
c_{city}	出租车在市内单位时间的平均收益
c_{back}	出租车载客从机场到市内的平均收益
t_{wait}	出租车在蓄车池中等待时长
t_{equal}	出租车两种决策收益相同时的 t_{wait} 临界值
t_{back}	出租车从机场返回市区的时长
T_{wait}	出租车在机场等待到完成载客的总时长
$W_{in}^{i}(i=0...n)$	排在第 i 个位置的出租车驶入蓄车池的时间
$W_{out}^{i}(i=0...n)$	排在第 i 个位置的出租车预计接到乘客的时间
I_{back}	出租车直接返回市区的收益
I_{wait}	出租车在机场载客的收益
m_{wait}	排队打车乘客的队列长度

四、 模型建立与求解

4.1 问题一的建模与解答

4.1.1 影响司机决策的相关因素

要分析影响出租车司机决策的相关因素，首先要明确司机进行决策的根本依据，即出租车司机始终以个人收益最大化为决策目标，来决定是否留在机场等待载客。因此要分析影响司机进行决策的相关因素，就要对影响司机收益的因素进行分析。比较司机到达机场后的两个决策：立即返程和在蓄车池等待调度。其中在立即返程得情况下司机在到达市区前的收益为 0, 到达市区后平均单位时间收益为 $\overline{c_{city}}$。在蓄车池中等待调度的出租车获得的收入为机场载客平均收益 $\overline{c_{back}}$，但是消耗了在蓄车池等待乘客的时间 t_{wait}。由于回到市区以后，都按市区均价计量收入，因此，当在机场等待乘客的司机返回市区时，两种决策对司机盈利带来的影响也随之消失。分别计算两种决策下司机的收益，公式如下：

$$I_{back} = \overline{c_{city}} * t_{wait} \tag{1}$$

其中，I_{back} 为出租车直接返回市区的收益，$\overline{c_{city}}$ 为出租车在市区单位时间内的平均收益，t_{wait} 为出租车在蓄车池的等待时长。

$$I_{wait} = \overline{c_{back}} \tag{2}$$

其中，I_{wait} 为出租车在机场载客的收益，$\overline{c_{back}}$ 为出租车在机场载客的平均收益。

观察上述两个决策的收益，由于市区单位时间收益 $\overline{c_{city}}$, 出租车机场载客平均收益 $\overline{c_{back}}$ 都是固定的，所以对决策产生影响的只有出租车司机在蓄车池中的等待时间 t_{wait}。要想探究影响出租车司机收益的因素，也就是探究影响出租车司机在蓄车池中等待载客时长的因素。由题目可知：司机在到达机场时可以直接观测到的信息有某时间段抵达的航班数量和“蓄车池”里已有的车辆数，我们从司机直接观测到的数据来分析有关因素对司机决策造成的影响：

(1) 司机到达机场近期内出发以及抵达的航班数量。该因素对司机决策产生的影响在于司机可以通过观测到的 i 小时内到达机场的航班的情况来推测目前排队和即将到来等待出租车的乘客的大致数量。在 i 小时内到达机场的航班数量越多，就说明旅客对于出租车的需求量越大，则出租车空等的情况就会减少，出租车司机的等待时间也相应有所减少。同样，在 i 小时内出发的航班数量越多，就说明出租车对于旅客的需求量越大，出租车空等的概率加大，出租车的等待时间也相应增加。

(2) 当前蓄车池中等待的出租车数量 n。该因素对司机决策产生的影响在于当前蓄车池中等待的出租车数量决定了司机在蓄车池中等待时间 t_{wait} 的下限，由于蓄车池的排队策略采用“先来后到”的策略调度出租车，所以该出租车司机至少要等在他之前的 n 个出租车全部驶离之后才可以进行载客。所以当蓄车池中等待出租车数量 n，出租车司机的等待时间也会相应减少。

(3) 当前机场的天气状况。该因素对司机决策产生的影响在于司机可以通过判断天气状况来推断该时间段内到达的旅客的打车倾向。如果该时间段的天气为雨，则会导致私家车出行减少，司机会推断由于受到该天气影响，到达机场的旅客打出租车的概率会有所提升，从而使得决策更偏向于留在机场等待载客。

综上所述，影响出租车司机决策的主要因素有据司机到达机场近期内出发以及抵达的航班数目、当前蓄车池中等待的出租车数量 n 和当前机场天气状况，他们通过影响司机在蓄车池中的预计等待时间来影响司机决策。

4.1.2 出租车司机选择决策模型

通过之前对影响司机决策的因素的分析，可以对司机做出决策的过程进行模拟，从而建立出租车司机选择决策模型，为出租车司机提供最佳的选择策略。由上文分析可得，当 $I_{back} = I_{wait}$，即 $\overline{c_{city}} * t_{wait} = \overline{c_{back}}$ 时，两个决策的收益相同，设该情况下的临界等待时间为 t_{equal}，t_{equal} 可由则可以将决策简化为：

(1) 当 $t_{wait} < t_{equal}$ 时，选择留在机场等待载客；

(2) 当 $t_{wait} > t_{equal}$ 时，选择立刻返程；

(3) 当 $t_{wait} = t_{equal}$ 时，两种决策获益相当，任选其一。

其中 t_{equal} 可以根据具体的城市与机场的数据计算得到，所以模拟出租车司机的决策只需要模拟出租车司机在蓄车池中的预计等待时间告诉时间，在让司机做出决策。

1. t_{wait} 的模拟

t_{wait} 是出租车在蓄车池中等待的时长，设 W_{in} 为出租车驶入蓄车池时刻，W_{out} 为出租车接到乘客驶出蓄车池时刻，则有

$$t_{wait} = W_{out} - W_{in} \tag{3}$$

出租车驶入蓄车池时间是确定的，所以对 t_{wait} 的预测就是对 W_{out} 的预测。对于 W_{out} 的模拟，我们采用动态规划的算法，即分阶段求解决策。

采用动态规划算法，首先要判断 W_{out} 的求解是否具有最优子结构的特征。最优子结构指问题的最优解包含子问题的最优解，针对于 W_{out}，我们假设该出租车司机前面有 $n-1$ 名司机，则该司机若进入蓄车池，他将排在第 n 名，第 i 名出租车司机在蓄车池中的预计离开时间为 $W_{out}^{i}(i=1...n)$，则该司机预计离开时间为 W_{out}^{n}。由于蓄车池采用的是“先来后到”的排队方式，所以该司机只有当他前面的一个司机出发后他才能出发，也就是说若想使得 W_{out}^{n} 取到尽可能早的最优解，就要保证 W_{out}^{n-1} 有最优解，这就说明 W_{out} 的求解符合动态规划的最优子结构要求。

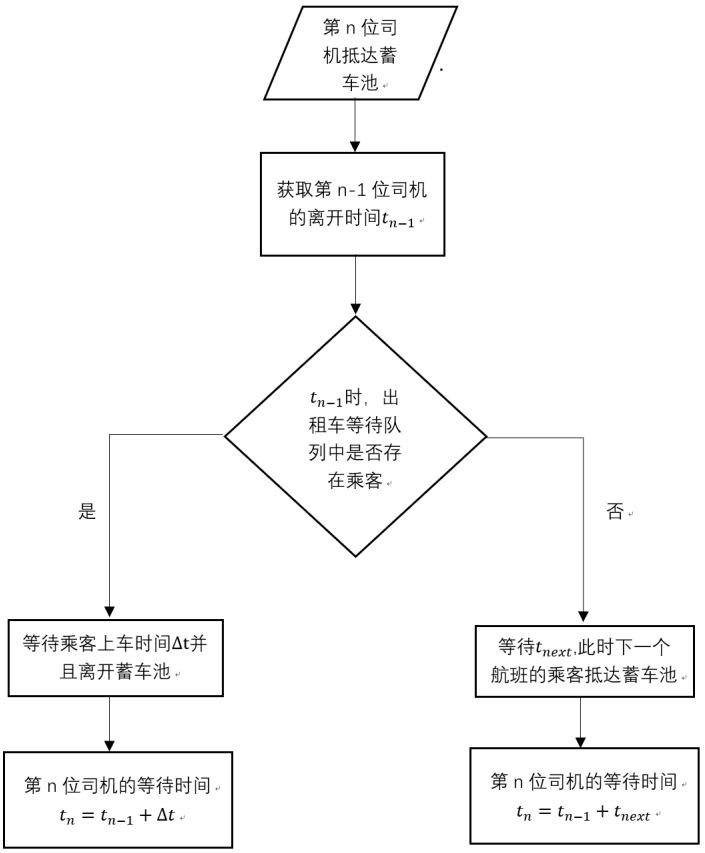


图 **1**　最优子结构图

W_{out} 满足最优子结构，就说明 W_{out}^{n} 可以通过 W_{out}^{n-1} 进行表达。对于当前出租车司机，在他前一名司机（也就是第 n-1 名）接到客人之前，他们的等待时间是相同的。当第 n-1 名司机接到乘客驶离后，该司机面临两种情况：

（1）乘客队伍中还有需要打车的乘客，则司机前往接人；

（2）乘客队伍中已空，则该司机需要等到下一个航班到达后才会等到乘客。

用 t_n^{n-1} 来表示第 n-1 位司机离开到第 n 个司机离开所用时间，t_{aboard} 表示乘客上车所需时间，t_{next} 表示距离下一辆航班抵达后第一位旅客到达的时间，m_{wait} 表示乘客排队的队列长度，所以 W_{out} 的状态转移方程可描述如下：

$$W_{out}^{n} = W_{out}^{n-1} + t_n^{n-1} \begin{cases} t_n^{n-1} = t_{aboard} & m_{wait} \neq 0 \\ t_n^{n-1} = t_{next} & m_{wait} = 0 \end{cases} \tag{4}$$

由此状态转移方程就可以推导出该出租车司机预计离开蓄车池的时刻，对于方程中需要判断的 m_{wait} 即排队乘客人数，则需要司机根据航班抵达状况进行推断，得到当前大概抵达乘客数量和预计到来乘客数量。

2. 乘客等待队列长度 m_{wait} 的估算

分析影响当前排队旅客人数的因素，乘客等待队列长度主要受到近期已经抵达机场的航班信息和近期预计起飞的航班信息的影响。其中近期已经抵达机场的航班信息主要影响到达等待队列的乘客人数，近期抵达航班越多，到达等待队列打车的旅客越多。近期预计起飞的航班信息主要影响到出租车流量，近期预计起飞的航班越多，则到达的机场的出租车越多，旅客等待出租车的时间就会越少。所以可以根据近期起飞航班信息以及出发航班信息对 m_{wait} 的值进行预测。

3. 司机的决策流程

通过上述分析，可以建立司机决策模型，预测当前出乘客等待队列的长度，从而估算出司机的预计等待时间，帮助司机决定是否在停车场等待载客。司机的决策流程图如下：

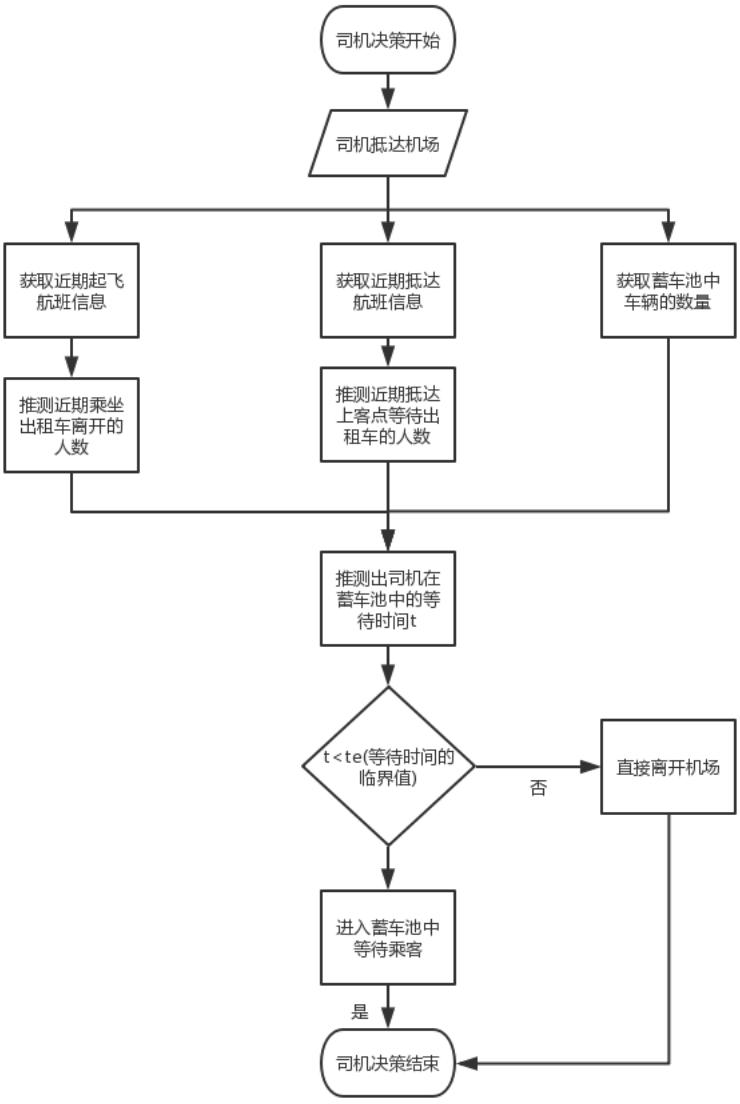


图 **2**　司机决策流程图

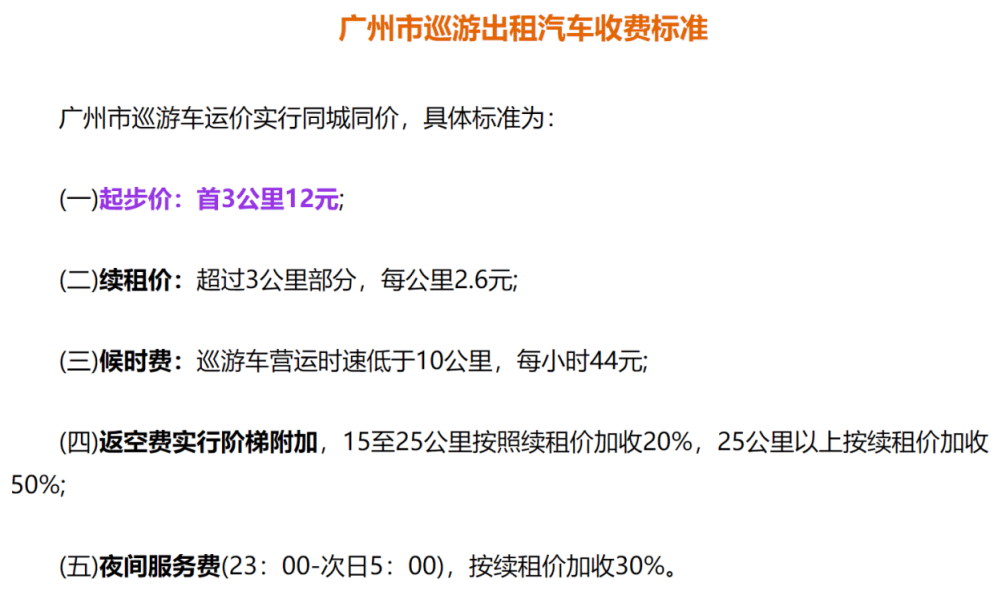
4.2 问题二的建模和求解

问题二要求收集国内某一机场及其所在城市出租车的相关数据，给出该机场出租车司机的选择方案。问题一已对司机进行决策的策略进行建模，所以只需要收集到建模所需数据带入模型进行训练即可。

4.2.1 广州白云机场相关数据收集

我们选择广州的白云机场作为机场的数据来源，通过从白云机场的官方网站上获取到的 9 月 13 日、9 月 14 日的全部航班信息，包括到达航班的到达时间信息、航站楼信息、机型信息和出发航班的出发时间信息、航站楼信息、机型信息，整理得到一共 2500

余条航班信息，详情见附录。我们还搜集到了建模所需关于广州市的出租车收费方案，根据《广州市发展改革委广州市交通委员会关于优化调整巡游出租车运价的通知》，广州市出租车收费如下图所示：



广州市巡游出租汽车收费标准

广州市巡游车运价实行同城同价，具体标准为：

(一)**起步价：首3公里12元**;

(二)**续租价：**超过3公里部分，每公里2.6元;

(三)**候时费：**巡游车营运时速低于10公里，每小时44元;

(四)**返空费实行阶梯附加**，15至25公里按照续租价加收20%，25公里以上按续租价加收50%;

(五)**夜间服务费**(23：00-次日5：00)，按续租价加收30%。

图 **3**　出租车收费方案

结合广州市出租车收费策略和我们搜集到的其他信息，我们查找到广州市出租车日均营业额为 1000 元，由于广州市出租车分白班夜班两段工作时间，每班工作时间为 8 小时，可以得到广州市内出租车平均每小时收入 $\overline{c_{city}}$=￥63、白云机场距离市区距离 $38.8km$、白云机场乘坐出租车到市区的价格 $\overline{c_{back}}$=￥100，白云机场乘坐出租车返回市区的平均用时 $t_{back}=45$ 分钟等，在获取到白云机场和出租车的数据后，我们可以对问题一中描述的决策模型进行实现。

4.2.2 白云机场的出租车决策方案

将白云机场的 $\overline{c_{back}}$、$\overline{c_{city}}$、t_{back} 带入公式 (1) 中，可得到白云机场出租车司机的等待乘客时间临界值 t_{equal} 为 95 分钟，说明出租车司机最多可以容忍在白云机场等待 95 分钟的时间即 $t_{wait}\leq 95$ 分钟，否则他将立刻返回市区。t_{wait} 的计算方法可以通过动态规划的方法进行判断，在上一文中已详细说明，要想得到出租车在蓄车池等待时长 t_{wait}，就要预测出排队打车乘客队列长度 m_{wait} 的值，对于乘客队列长度 m_{wait} 的预测，我们需要根据白云机场的旅客行为特征来对队列长度预测模型进行训练。

1. 已知队列长度下的出租车决策模拟

（1）对白云机场一天的客流量与出租车流量的模拟

根据已获取到的白云机场的一天的航班信息，我们可以对白云机场一天的客流量与出租车流量进行模拟。

对客流量模拟：当一架航班抵达机场时，他所携带的乘客为这架飞机的客容量乘以它的上座率，对于这一批旅客，他们出站的时间各不相同，我们采用正态分布函数来对

这批旅客出站的时间进行模拟，通过阅读论文[1] 我们可以得到旅客从下机到出站的平均时间为 30 分钟，所以可设旅客的到达规律遵循 $\mu = 30$ 分钟的正态分布, 这样就可以模拟一架航班的旅客抵达后的客流量变化。对于即将起飞的航班，由于机场建议乘客于飞机起飞前两小时左右抵达机场，所以乘客抵达机场遵循 $\mu = 120$ 分钟的正态分布。广州白云机场 9 月 13 号客流量随时间的变化如图所示：

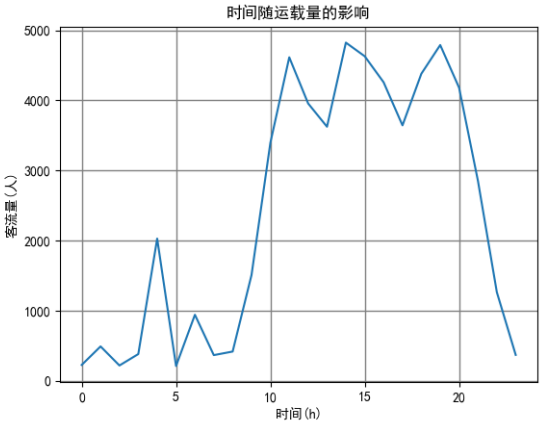


图 **4**　白云机场客流量随时间的变化

对出租车流量模拟：通过抵达和即将起飞的航班信息我们可以模拟出机场客流量随时间的变化，而出租车流量变化可以通过客流量变化反映出来。在白云机场旅客抵离方式选择研究[1] 这篇论文中，作者对白云机场的旅客抵达和离开机场的选择进行了调查，统计出前往机场的旅客选择出租车方式的概率 P_{arrive} 为 9.54%，离开机场的旅客选择出租车方式的概率 P_{leave} 为 13.56%，所以可以通过计算出的抵达和离开机场的旅客流量来估算出出租车的流量，即

$$\begin{cases} F^{p}_{arrive} * P^{t}_{arrive} = F^{t}_{arrive} \\ F^{p}_{leave} * P^{t}_{leave} = F^{t}_{leave} \end{cases} \tag{5}$$

其中 F^{p}_{arrive}、P^{t}_{arrive}、F^{t}_{arrive} 分别为到达客流量、到达乘客选择出租车的概率、到达出租车流量。

（2）已知队列长度下的司机决策模拟

在根据机场航班信息模拟出白云机场一天的客流量变化和出租车流量变化后，我们先假设司机已知当前队列长度对其决策进行模拟，来获得队列长度的变化信息，从而获取训练队列长度预测模型的数据。司机在已知当前队列长度的情况下对 t_{wait} 的预测是准确的，所以其做出的决策也是正确的，现通过目前模拟出来的机场一天的流量数据结合司机已知队列长度下的决策策略对这一天所有到达机场的司机的决策进行模拟，同时记录乘客队列长度随时间的变化，为队列长度 m_{wait} 的预测模型提供训练数据。

2. 乘客等待队列长度 m_{wait} 的预测模型

问题一中我们分析了影响 m_{wait} 的因素主要有近期已经抵达机场的航班信息数据和近期预计起飞的航班信息数据，在对已知乘客队列长度 m_{wait} 下的出租车决策模拟的过程中我们获得了有关乘客队列长度随时间变化的数据，通过计算近期各个时间点的已经抵达机场的航班信息数据和近期预计起飞的航班信息数据，我们分别采用 RELU 函数以及正态分布函数度量近期起飞航班以及抵达航班对乘客等待队列得影响。

其中 RELU 函数表达式为：

$$v_m = \begin{cases} v_{max} & n_f >= n_e \\ n_f * v_{max}/n_v & 0 \leq n_f < n_e \end{cases} \tag{6}$$

其中，n_f 表示 1-2 小时内将要起飞得航班数，n_e 表示航班数影响得临界值，v_max 表示蓄车池最大得通行速率。

对于近期已经抵达机场的航班信息数据和近期预计起飞的航班信息数据的计算方法如下：对于近期已经抵达机场的航班信息数据，我们采用 $\mu = 30$ 分钟的正态分布来对不同时间抵达机场的航班分配权重，再进行求和，就会得出出租车乘客等待队列得近期来人；对于近期预计起飞的航班信息数据，由于出租车接客驶离的速度有一个上限，即当存在足够多的出租车时，出租车离开的速率会维持一个稳定的最大值，在达到这个最大值之前，出租车服务的速度与到达机场的出租车数量呈正相关，所以我们选择 RELU 函数作为表达于出租车载客的速率随即将起飞的航班数量的变化如图所示：

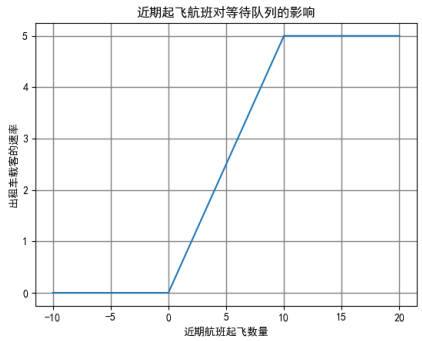


图 **5**　即将起飞航班权重的 **RELU** 曲线

因此，根据近期抵达航班信息估算出等待出租车队列新增乘客，根据将要起飞航班信息估算处出租车流量从而得出，等待出租车队列流失乘客速率，根据上述两者便可得出当前乘客等待队列得人数，表达式如下：

$$m_{wait} = \sum \int_0^t \frac{1}{\sigma\sqrt{2\pi}} exp^{-\frac{(x-\mu)^2}{2\sigma^2}} - \delta * v_m + k$$

其中，m_{wait} 表示当前乘客等待队列得长度，μ 以及 σ 分别表达抵达航班到达等待队列人数与时间有关得正态分布得均值以及方差，$\delta t * v_m$ 表示近期乘客队列离开得人数。

3. 司机基于队列长度预测模型的决策方案的合理性验证

在训练出队列长度预测模型之后，司机就可通过观测到的近期航班到达以及驶离的情况来对当前队列长度做出预估，再结合即将到达的航班信息来对未来流量预测，就可以推测出大致的等候时间，进而进行决策。为验证司机决策方案的合理性，现通过对已知队列长度决策算法和队列长度预测决策算法两种决策方案在 9 月 14 日上的数据进行模拟，通过观察两种方案中司机决策的相似性来判断此方法的合理性。两种方法在新的数据上的模拟如下图：

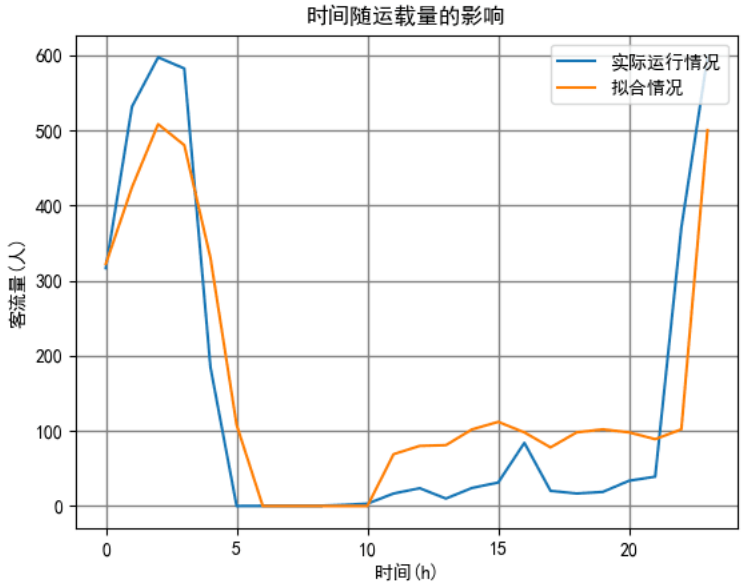


图 **6**　两种决策方案在新数据上的表现

根据上图，可以得出，拟合情况大致反映了实际情况下，等待队列客流量人数随时间的变化。同时在拟合情况下，87.92% 的司机做出了与实际情况相同的选择，进一步证实了该方案的合理性。

4. 司机选择方案对相关因素的依赖性

（1）司机选择方案对出发/抵达航班信息的依赖性

在掌握不可靠、不完全的信息的情况下做出相对较为优秀的决策有着重要的现实意义，我们在测试过程中对一部分数据进行了篡改以此观察当司机获取信息与实际情况不一致时决策方案的可靠性。下图反映了我们有关机场航班信息得测试数据：

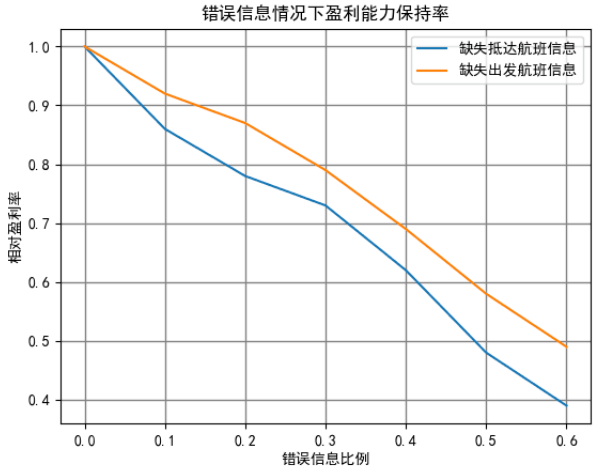


图 **7**　篡改出发/抵达航班信息

从图中我们发现，相较于出发航班信息，抵达信息对决策的影响更大，这也提示了机场管理者对司机进行必要的信息提示对降低机场出租车滞留率，提升出租车司机盈利

能力有着较大的帮助。当信息损失到达 30% 时，司机的营收能力降低到了理想情况的 70% 80%。当司机信息严重不足或错误较多时，可能会导致较为严重的堵车或者游客滞留情况。

（2）司机选择方案对蓄车池中车辆数的依赖性

错误的蓄车池数目信息通知会导致司机对当前乘客需求信息的错误估计。当蓄车池中给出信息和实际车辆数量信息相差达到 30% 以上时，司机的平均等待时间出现了大幅上升。这提示了机场方面应当加强对蓄车池信息的维护和及时公开。司机对蓄车池中车辆数得依赖如下图所示：

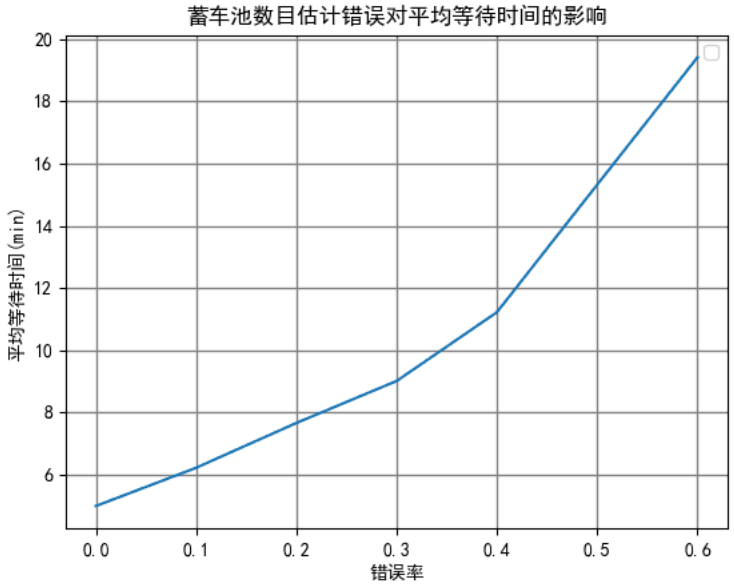


图 **8**　篡改出发/抵达航班信息

由以上分析可得决策对于相关因素的依赖性：抵达航班信息对决策的影响比出发航班信息对决策的影响更大，蓄车池中车辆数目对于决策的影响最大。

4.3 问题三的建模与求解

查阅资料可知[3] 双车位乘车区的结构如图所示，即两个车道都有相同数量的泊车位，蓄车池中的出租车按顺序从单车道驶入，在泊车区前分成两组车队分别进入两个车道进行泊车侯客，为保证乘客与车的安全，且尽可能提高乘车效率，管理部门需要对“上车点”，出租车和乘客的管理策略以及泊车位的数量进行设计。

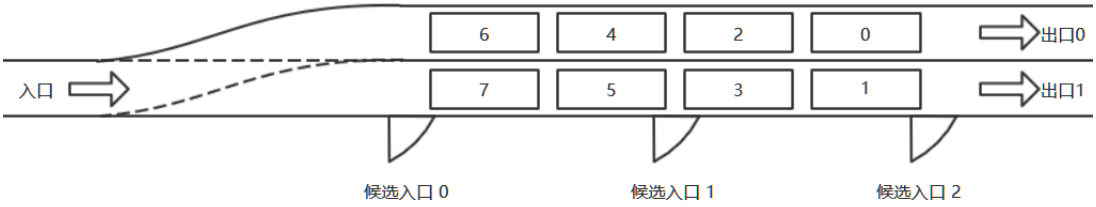


图 **9**　双车位乘车区结构示意图

4.3.1 双车道乘车区的设计

1. 双车道上车点的设计

图 8 中的 3 个候选入口即为上车点，队列中的乘客只能从上车点进入车道乘车，上车点的位置不同，旅客从上车点移动到出租车的时间也会有所不同，进而会影响乘车的效率，现根据上车点相对于泊车位的位置预设三个预选上车点，从中选择使得乘车效率最高的上车点。

2. 泊车位数量的设计

如图 8 所示，双车道每一个车道都有相同数量的泊车位，当出租车驶入乘车区时，它们会依次停在相应的泊车位中等待乘客乘车，对于每一个车道上的泊车位数量，需要我们通过计算不同泊车位下的乘车表现来选择出使得乘车效率最高的泊车位个数。现定义两个用于比较不同策略下的模型表现的指标：

（1）出租车通行能力 P_{pass}，表示单位时间内蓄车池驶出出租车数量。

（2）泊位空闲率 R_{free}，表示每一个泊位空闲的比例（空闲即泊位上没有出租车）。

对于这两个指标，应当调整泊位数量来使得出租车通行能力尽可能提高，泊位空闲率尽可能降低，最终实现旅客的乘车效率最高。

3. 基于安全性的出租车和乘客的调度策略

如图 8 所示，上车点都位于出口 1 的车道一侧，当乘客要前往出口 0 车道乘车时就必须要横穿一条车道。为保证乘客安全，就必须要设定策略对穿过马路的出租车调度不能和乘客穿过马路的调度相冲突。也就是当有乘客横穿车道 1 去车道 2 乘车的时候，车道 1 不允许车队驶入；当有车驶入车道 1 的时候，不允许乘客横穿车道去车道 2 乘车。

此外，处于车辆以及乘客的安全，车辆之间需要保持 1.5m 以上的间距、车速不得超过 2m/s、车道内所用车辆停止稳定，乘客才能进入上客区上车。

4.3.2 双车道乘车区调度模型

1. 乘客上车所用时间模拟

查阅论文[3] 得到机场乘客上车所用时间的特征：乘客上车所用的时间近似服从均值为 32.8 s 的指数分布。所以可以依概率对旅客上车的时间进行随机，使得每个旅客的上车时间各不相同，但在整体上符合指数分布特征。

2. 双车道乘车区数据

表 **2**　双车道乘车去数据

车身长	道路宽度	行人步速	车速	等待区长度	停车稳定时间
4m	3m	1.4m/s	2m/s	3m	1.5s

3. 对具有互斥性的旅客横穿车道 1 与出租车驶入车道 1 的模拟

对于这两个操作的互斥性，在模拟的时候采用互斥锁的技术，来确保同一时间只能使出租车或者乘客中的一个对象来获取过路权，只有到当一个对象完成对车道的占有释放了互斥锁的时候，另一个对象才可以获取该车道的互斥锁，从而保证模拟中两个操作的互斥性。

4.3.3 不同上车点对上车区通行能力的影响

在上车点人数以及蓄车池中车数足够多的情况下，我们分别在双车道五泊车位的条件下，仿真 3 种不同入客口对应通车情况，结果如下表：

表 **3**　不同上车点下的通行能力

	车道数	泊位数	通行能力 (辆/小时)
候选入口 0	2	5	435
候选入口 1	2	5	480
候选入口 2	2	5	430

根据表格中的数据我们可以发现，候选入口 1 的通行能力明显优于其余两种入口。分析可知，选择位于车道中间的候选入口 1，缩短了行人距离车道两端点的距离，从而降低了乘客的上车时间。因此，管理人员应该在车道中间 (候选入口 1) 处设置上车点，这样可以增加上车区的通行能力。

4.3.4 车道泊位数对上车区通行能力的影响

在时间跨度为 1 小时的条件下，分别仿真多种车道泊位数情况下，双车道上车区的通行能力，结果如下表：

表 **4** 不同上车点下的通行能力

泊位数	最后泊位利用率	通行能力 (辆/小时)	边际通行能力
1	91.93%	272	76
2	83.37%	348	36
3	83.41%	384	36
4	80.33%	420	60
5	74.43%	480	24
6	71.53%	504	35
7	66.67%	539	5
8	66.23%	544	5
9	66.23%	549	0

分析上表，可以发现在一定范围内，上车区的通行能力随着泊位数的增加而不断表达，而泊位利用率随之减小，并且在每条车道泊位数为 7 时，上车区的边际通行能力急剧下降。同时，由于泊位数的增加，将给管理人员带了维护上客区秩序带来极大的难度，因此车道泊位数不宜过大。

综合上述，在保证乘客安全以及车辆通行效率的情况下，应该选择将上车点设置在车道中部 (候选入口 1)，并且每条车道提供 5 个车辆停泊位，并且乘客只能等车道上车辆停稳的情况下，方能进入车道。

4.4 问题四的建模与求解

4.4.1 短途载客司机的优先权分配方案

1. 短途司机与普通司机的收益比较

要想让短途载客的出租车司机所得收益和其他司机收益尽量均衡，以接到短途乘客的司机最终返回到市区的时间为基准，对接到短途乘客的司机在此期间的收益和普通机场到市区的司机的收益进行表达：

设接到短途乘客的司机将乘客从机场送到短途目的地所用时间为 t_{short}，接到短途乘客的司机返回机场在机场获得优先权后的等待时间为 $t_{priority}$, 该司机从最开始到达机场一直到回到市区所用时间为 T_{short}，则 T_{short} 的表达式为：

$$T_{short} = t_{wait} + 2t_{short} + t_{priority} + t_{back} \tag{7}$$

设短途司机在 T_{short} 时间内的收益为 I_{short}，短途收益为 c_{short}，则 I_{short} 的表达式为：

$$I_{short} = c_{short} + c_{back} \tag{8}$$

对于接到普通回市区乘客的司机来说，他们在 T_{short} 时间内的时间组成如下：

$$T_{short} = t_{wait} + t_{back} + t_{city} \tag{9}$$

其中 t_{city} 为他们返回市区后在市区载客的时间，他们在 T_{short} 时间内的收益为：

$$I_{normal} = c_{back} + c_{city} * t_{city} \tag{10}$$

2. 短途载客司机的优先权分配方案

根据公式 (8)、(9)、(10)、(11) 可得当短途载客的出租车司机在 T_{short} 时间内的所得收益和其他司机收益相同时，有 $T_{short} = T_{short}$、$I_{short} = I_{normal}$，可得到如下关系：

$$c_{short} = c_{city} * t_{city} \tag{11}$$

$$t_{city} = 2t_{short} + t_{priority} \tag{12}$$

联立公式（12）、（13）可得到 $t_{priority}$ 的表达式：

$$t_{priority} = \frac{c_{short}}{c_{city}} - 2t_{short} \tag{13}$$

通过上式我们可以计算出短途司机获得优先权后的等待时间 $t_{priority}$，从而使得机场出租车调度人员可以通过查找当前出租车队伍中各个出租车的预计载客时刻可以将该短途司机置于等待时间为 $t_{priority}$ 的队伍位置进行排队，若 $t_{priority} \leq 0$ 就说明该司机进行的短途载客的预计收益已小于普通司机收益，则将该短途司机出租车置于队伍首位，无需进行更多等待。

4.4.2 短途载客司机优先权分配方案的检验

根据上述优先权处理策略，现对机场出租车的载客情况进行模拟，对于司机载到的乘客，我们采用 10% 的比例设置短途旅客，再把短途旅客的短途里程设置为 15-25km 的随机数，管理员对于各出租车预计离开时间的估算采用问题二中动态规划的方法，现对一天的乘客的出租车出行进行模拟，为了方便比较两种司机的收益，现取短途载客最长时间，即短途距离为 25km 的司机的 $T_{short} = xmin$ 做为比较出租车司机收益的时间长度，计算出各个司机在机场的这段时间内的收益，随机取其中的 100 名司机绘制收益散点图如下图所示：

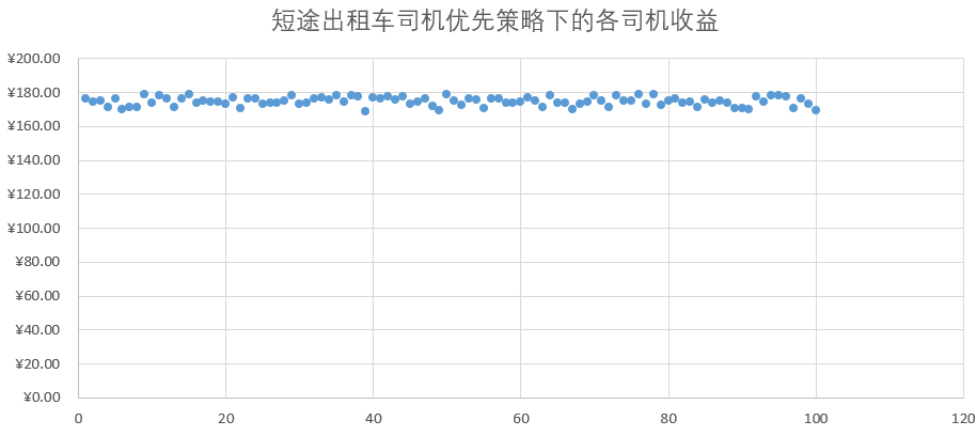


图 **10**　短途载客司机优先权方案的司机收益

由散点图可以看出，采用此策略管理机场出租车，可以使机场所有的出租车都具有基本一致的收益，计算该收入散点的方差为 6.54，在可以接受的范围之内，说明该策略可以良好的均衡短途载客的出租车和其他出租车之间的收益差距。

五、 模型的评价

5.1 模型的优点

(1) 模型一充分考虑了包括航班信息、天气、蓄车池中车辆数等多个因素对司机决策的影响，考虑较为周全。

(2) 模型三对通过仿真的方式，较为准确的计算出不同泊位数、不同上车点下，上车区的通行能力。

(3) 模型四采用蒙特卡洛算法，指定短途司机的里程数，结果更加接近真实情况。

5.2 模型的缺点

(1) 模型二中只获取到广州白云机场两天的航班信息，数据量可能不够。

5.3 模型的应用以及推广

(1) 本文提出的短途载客司机优先权分配方案，充分考虑了短途司机的里程以及返程时间，从而可以较为精准的分配短途司机优先级，使得大部分司机收益均衡，具有很高的实用价值。

(2) 本文提出的上车点在车道中部、双车道 5 泊位的上客区模型，既能在一定程度上提高上客区的车辆通行效率，同时也便于管理人员监管，可以在部分机场推广试点。

参考文献

[1] 林思睿. 机场出租车运力需求预测技术研究 [D]. 电子科技大学,2018.

[2] 郭振义. 白云机场旅客抵离方式选择研究 [D]. 华南理工大学,2015.

[3] 孙健. 基于排队论的航空枢纽陆侧旅客服务资源建模与仿真 [D]. 中国矿业大学 (北京),2017.

附录 A　出租车司机的决策模型

```
import datetime
import time

from copy import deepcopy
# import math
import numpy as np
from scipy.stats import norm

import conf
import dataloader
import role
import json

# import matplotlib.pyplot as plt
arriveData = dataloader.arriveData()
departData = dataloader.departData()


def genInputTaxiSequence() -> list:
    pass


def genDepartList(count: int, timeOffset: int):
    con = conf.Conf()
    np.random.seed(0)
    res = list()
    while len(res) < count:
        t = int(np.random.normal(con.departMiu, con.departSigma, 1)[0])
        if t > con.departLeft and t < con.departRight:
            res.append(int(t + timeOffset))
    return res


def genArriveList(count: int, timeOffset: int):
    con = conf.Conf()
    np.random.seed(0)
    res = list()
    while len(res) < count*con.arriveLRate:
        t = int(np.random.normal(con.arriveLMiu, con.arriveLSigma, 1)[0])
        if t > con.arriveLLeft and t < con.arriveLRight:
            res.append(int(t + timeOffset))
    while len(res) < count:
        t = int(np.random.normal(con.arriveHMiu, con.arriveHSigma, 1)[0])
        if t > con.arriveHLeft and t < con.arriveHRight:
            res.append(int(t+timeOffset))
    return res


with open('./plane.json', 'r') as r:
    planeModule = json.load(r)


def mode2people(src: str):
    return planeModule[src]
```

```
def genDepartTimeLine():
    con = conf.Conf()
    timeLine = list()
    for i in departData:
        timeLine.extend(
            genDepartList(
                int(mode2people(i['机型'])*con.departRate/con.taxiAvgPeople),
                int(time.mktime(i['计划出发时间'].timetuple()))/60
            )
        )
    timeLine.sort()
    return timeLine


def genArriveTimeLine():
    con = conf.Conf()
    timeLine = list()
    for i in arriveData:
        timeLine.extend(
            genArriveList(
                int(mode2people(i['机型'])*con.arriveRate/con.taxiAvgPeople),
                int(time.mktime(i['计划到达时间'].timetuple()))/60
            )
        )
    timeLine.sort()
    return timeLine


def minuteStamp(src: datetime.datetime)->int:
    return int(time.mktime(src.timetuple())/60)


class Simulator:
    def __init__(self):
        # 航班信息
        self.arriveData = arriveData
        self.departData = departData
        # 时间线
        self.departTimeLine = genDepartTimeLine()
        self.arriveTimeLine = genArriveTimeLine()
        self.conf = conf.Conf()
        self.keyTime = None

    def train(self):
        self.log = dict()
        self.driverLog = list()
        self.logList = list()
        self.logListPassenger = list()
        self.log['报告产生时间'] = time.strftime(
            "%Y-%m-%d %H:%M:%S", time.localtime(time.time()))
        self.log['total passenger'] = len(self.arriveTimeLine)
        self.log['total taxi'] = len(self.departTimeLine)
        self.sentPerHour = [0]*24
        # 统计需求2
        self.driverALLProfits = 0
        self.driverALLTimes = 0

        self.keyClipWindowDepartLeft = 0
```

```
        self.keyClipWindowDepartRight = 0
        self.keyClipWindowArriveLeft = 0
        self.keyClipWindowArriveRight = 0
        self.keyArriveTimeLine = 0
        self.keyDepartIimeLine = 0
        self.passengerQuery = list()
        self.carQuery = list()
        self.tempCarQuery = list()
        for self.keyTime in range(
                minuteStamp(self.conf.startDate),
                minuteStamp(self.conf.endDate),
                1):
            if self.keyTime % 60 == 0:
                proc = ((self.keyTime+1 - minuteStamp(self.conf.startDate)) /
                        (minuteStamp(self.conf.endDate) -
                         minuteStamp(self.conf.startDate)))*100
                print("%s\n[%.2f%%]" % (self.keyTime, proc), end="\r")
            self.update(self.driverDetermineKnown)
            # self.update(self.driverDetermine)
            res = dict()
            res['time'] = time.strftime(
                "%Y-%m-%d %H:%M:%S", time.localtime(self.keyTime*60))
            res['passenger'] = len(self.passengerQuery)
            res['taxi in ready'] = len(self.tempCarQuery)
            res['taxi at depart'] = len(self.carQuery)
            self.logList.append(res)
        self.log['各列表历史记录'] = self.logList
        self.log['接单司机单位时间收入'] = self.driverLog
        self.log['司机盈利状况'] = self.driverALLProfits / \
            self.driverALLTimes * 60
        self.log['带客司机平均等待时间'] = sum(
            [i['costTime'] for i in self.driverLog]) / len(self.driverLog)
        self.log['上车乘客等待时间'] = self.logListPassenger
        self.log['上车乘客平均等待时间'] = \
            sum([i['costTime'] for i in self.logListPassenger]) / \
            len(self.logListPassenger)
        self.log['乘客运量统计']= self.sentPerHour
        with open('log.json', 'w') as w:
            json.dump(self.log, w)

    def update(self, choiceFunction):
        # 更新航班信息提示指针
        while self.keyClipWindowArriveLeft < len(self.arriveData) and \
            minuteStamp(
            self.arriveData[self.keyClipWindowArriveLeft]
                ['计划到达时间']) < self.keyTime +\
                self.conf.clipWindowArriveLeft:
            self.keyClipWindowArriveLeft += 1
        while self.keyClipWindowArriveRight < len(self.arriveData) and \
                minuteStamp(
                    self.arriveData[self.keyClipWindowArriveRight]
                    ['计划到达时间']) < self.keyTime + \
                self.conf.clipWindowArriveRight:
            self.keyClipWindowArriveRight += 1
        while self.keyClipWindowDepartLeft < len(self.departData) and \
                minuteStamp(
                    self.departData[self.keyClipWindowDepartLeft]
                    ['计划出发时间']) < self.keyTime + \
```

```
            self.conf.clipWindowDepartLeft:
        self.keyClipWindowDepartLeft += 1
    while self.keyClipWindowDepartRight < len(self.departData) and \
            minuteStamp(
                self.departData[self.keyClipWindowDepartRight]
                ['计划出发时间']) < self.keyTime + \
            self.conf.clipWindowDepartRight:
        self.keyClipWindowDepartRight += 1

    # 更新到达乘客信息
    while self.keyArriveTimeLine < len(self.arriveTimeLine) and \
            self.arriveTimeLine[self.keyArriveTimeLine] <= self.keyTime:
        if len(self.passengerQuery) < self.conf.maxWaitingQueue:
            self.passengerQuery.append(role.Passenger(
                self.arriveTimeLine[self.keyArriveTimeLine]))
        self.keyArriveTimeLine += 1
    # 更新到达出租车信息
    while self.keyDepartIimeLine < len(self.departTimeLine) and \
            self.departTimeLine[self.keyDepartIimeLine] <= self.keyTime:
        self.tempCarQuery.append(role.Taxi(
            self.departTimeLine[self.keyDepartIimeLine]))
        self.keyDepartIimeLine += 1
    # 推进蓄车池队列
    for p, i in enumerate(self.tempCarQuery[:]):
        if i.loadTime + \
            self.conf.processTime['depart2choice'] \
                <= self.keyTime:
            if choiceFunction():
                self.carQuery.append(deepcopy(i))
            self.tempCarQuery.remove(i)
    # 载客
    if self.keyTime % self.conf.processTime['waitPassenger'] == 0:
        maxCon = min(len(self.carQuery), len(
            self.passengerQuery), self.conf.patch)
        # 有载客行为
        if maxCon > 0:
            for i in range(maxCon):
                lastCar = self.carQuery[0]
                lastPassenger = self.passengerQuery[0]
                self.sentPerHour[
                    time.localtime(self.keyTime*60).tm_hour
                ]+= maxCon
                # 统计需求
                self.driverLog.append(
                    {
                        "time": lastCar.loadTime,
                        "costTime": self.keyTime - lastCar.loadTime,
                        "profitsPerHour":
                        110/(45+(self.keyTime - lastCar.loadTime))*60
                    }
                )
                self.logListPassenger.append(
                    {
                        "time": lastPassenger.loadTime,
                        "costTime": self.keyTime-lastPassenger.loadTime
                    }
                )
```

```
                costTime = self.keyTime - lastCar.loadTime+self.conf.toCity
                self.driverALLProfits += self.conf.averageCash - \
                    self.conf.averageProfits * \
                    (costTime) / 60
                self.driverALLTimes += costTime

                del self.carQuery[0]
                del self.passengerQuery[0]
            else:
                # 无出租车
                if len(self.carQuery) == 0:
                    for p, i in enumerate(self.passengerQuery[:]):
                        if i.waitTime is None:
                            i.waitTime = self.keyTime
                        else:
                            if self.keyTime - i.waitTime > \
                                    self.conf.maxWaitTime:
                                self.passengerQuery.remove(i)
                # 无乘客
                else:
                    # maxT = max(self.conf)
                    for i in self.carQuery[:]:
                        if i.waitTime is None:
                            i.waitTime = self.keyTime
                        else:
                            if i.waitTime + \
                                    self.conf.maxWaitTime < self.keyTime:
                                for _ in range(self.conf.patch):

                                    # 统计需求
                                    lastCar = self.carQuery[0]
                                    costTime = self.keyTime - lastCar.loadTime+self.conf.toCity
                                    self.driverALLProfits -= \
                                        self.conf.averageProfits * \
                                        (costTime) / 60
                                    self.driverALLTimes += costTime

                                    del self.carQuery[0]
                                break

    def driverDetermine(self) -> bool:
        arriveData, departData = self.clipData()
        inCounter = 0
        for i in arriveData:
            inCounter +=\
                mode2people(i['机型']) * self.conf.arriveRate / \
                self.conf.taxiAvgPeople * norm.cdf(
                    (self.keyTime-minuteStamp(i['计划到达时间']) -
                     self.conf.arriveLMiu)/self.conf.arriveLSigma
                )
        outCounter = 0
        for i in departData:
            outCounter +=\
                mode2people(i['机型']) * self.conf.departRate / \
                self.conf.taxiAvgPeople
        maxOutCounter = \
            (
                self.conf.clipWindowDepartRight - self.conf.clipWindowDepartLeft
```

```
            ) / \
            self.conf.processTime['waitPassenger'] * self.conf.patch
        outCounter = max(outCounter, maxOutCounter)
        if (inCounter - outCounter > len(self.carQuery)):
            return True
        else:
            return True

    def driverDetermineKnown(self) -> bool:
        # 乘客已在队列中
        if len(self.passengerQuery) > len(self.carQuery):
            waitTime = self.keyTime % self.conf.processTime['waitPassenger'] +\
                (len(self.carQuery)+1)//self.conf.patch * \
                self.conf.processTime['waitPassenger']
        else:
            waitTime = max(
                self.keyTime % self.conf.processTime['waitPassenger'] +
                (len(self.carQuery)+1)//self.conf.patch *
                self.conf.processTime['waitPassenger'],
                self.arriveTimeLine
                [
                    self.keyArriveTimeLine +
                    len(self.carQuery) -
                    len(self.passengerQuery)
                ]
            )
        if waitTime > self.conf.defaultWaitTime:
            return False
        else:
            return True

    def clipData(self):
        arriveData = self.arriveData[
            self.keyClipWindowArriveLeft:
            self.keyClipWindowArriveRight
        ]
        departData = self.departData[
            self.keyClipWindowDepartLeft:
            self.keyClipWindowDepartRight
        ]
        return arriveData, departData


if __name__ == '__main__':
    exp = Simulator()
    exp.train()
```

附录 B　机场出租车上客区模型

```
import scipy.stats as stats
import json


def calc(n=5):
    n = n
```

```
carSpeed = 2
humanSpeed = 1.4
carLength = 4
carMargin = 1.5
waitingLength = 3
roadWidth = 3
conTime = 1
safeLock = [False, False]
time = [0, 0]
timeOnStop = [[0] * n, [0] * n]
prepareTime = [[0] * n, [0] * n]
prepared = [[False] * n, [False] * n]
state = [0, 0]
counter = 0
# 0 无状态 1 车辆进入 2 停止 3 车辆驶出
t = (n * (carLength + carMargin) - carMargin + waitingLength) / carSpeed
incomeTime = [t, t + 1]
outTime = (n * (carLength + carMargin) - carMargin) / carSpeed
keyTime = 0
position = list()
for i in range(n):
    for j in range(2):
        position.append([2+5.5*i, j % 2*3])
locaiton = [(position[0][0]+position[-1][0])/2, 0]
locaiton = [position[-1][0]+2, 0]
while keyTime < 60*60:
    for i in range(2):
        # 更新人员信息
        if (i == 0 and safeLock[0] and safeLock[1]) or \
                (i == 1 and safeLock[1]):
            for j in range(n):
                while True:
                    res = stats.expon(scale=1/0.0305).rvs()
                    if res < 72:
                        break
                if prepared[i][j] is False:
                    prepareTime[i][j] = int(
                        (abs(position[2 * j + i][0] - locaiton[0]) +
                         abs(position[2 * j + i][1] - locaiton[1]) +
                           res)/humanSpeed+keyTime
                    )
                    prepared[i][j] = True

        # 更新车体信息
        if state[i] == 0:
            time[i] = keyTime
            state[i] = 1
            continue
        elif state[i] == 1:
            if time[i] + incomeTime[i] <= keyTime:
                state[i] = 2
                time[i] = keyTime
                safeLock[i] = True
            continue
        elif state[i] == 2:
            for j in range(n):
                if not prepared[i][j]:
                    break
```

```
                for j in range(1, n):
                    prepareTime[i][j] = max(
                        prepareTime[i][j - 1] + 2, prepareTime[i][j])
                if keyTime >= prepareTime[i][-1]:
                    for j in range(n):
                        timeOnStop[i][j] += prepareTime[i][j]-time[i]
                        prepared[i][j] = False
                    # print(prepareTime)
                    state[i] = 3
                    time[i] = keyTime
                    safeLock[i] = False
                    break
                continue
            elif state[i] == 3:
                if time[i] + outTime <= keyTime:
                    time[i] = keyTime
                    state[i] = 1
                    counter += n
                    continue
        keyTime += 1
        # print(keyTime)
    res = dict()
    res['车位数'] = n
    res['一小时过车计数'] = counter
    res['停滞时间'] = timeOnStop
    return res


if __name__ == '__main__':
    res = dict()
    for i in range(5,6):
        res[i] = calc(i)
    with open('./res.json', 'w') as w:
        json.dump(res, w)
```