机场出租车排队问题的优化配置

摘要

出租车在维持城市交通系统的可持续性方面发挥着重要作用，《上海市出租车公司数据报告》中指出上海 2018 年出租车数量 48900 台，平均每万人拥有 20.25 台。随着“互联网+”时代的到来，智能匹配管理系统实现了乘客需求预判、车辆决策调度，使得旅客与出租车双方的利益最大化。同时，根据民航局发布的《2018 年民航机场生产统计公报》，目前，年旅客吞吐量 1000 万人次以上的机场已达 37 个，其中首都机场旅客吞吐量突破 1 亿人次，居国内各机场首位。2018 年，浦东和虹桥两大机场保障年起降航班 771957 架次（浦东机场 504972 架次，虹桥机场 266985 架次）。由此可见，搭乘航班出行已经为国民普遍接受，并且据报告，人次在逐年上升。

本文以上海浦东国际机场为例，就出租车的历史行车数据、机场航班信息等数据，对出租车司机的选择决策进行讨论与分析，进而思考如何设计更优的“多上客点”和“短途返回优先权”等管理方案。

针对问题一，我们给出了可能影响出租车司机决策的相关因素，并且给出了出租车司机的选择决策模型的数学表达。

针对问题二，我们对爬虫爬到的部分出租车单日数据和上海机场单日数据进行分析、处理，给出第一问中一些参数的经验值，带入第一问用来进一步完善模型，并且 python 编程实现了输入航班数和等待出租车数量，输出司机最优决策，即在机场等待接客或直接返回市区。本章最后给出了模型的合理性说明和局限性。

针对问题三，我们考虑到多乘车点和双车道的特殊性，建立数学模型，给出了使得乘车效率最大化的排队乘车模型，以及配套的建筑设计方案。最后还给出了补位时间控制的计算公式。

针对问题四，针对长短途司机收益的差别，给出了“短途优先权”的具体设计方案，达到短途载客再次返回的出租车的收益能够与其他车均衡。用 python 编程，利用动态规划的数学模型实现用户输入约束条件和观察值，给出机场出租车管理系统应该给出的量化优先权。最后，在第一二三问的基础上，考虑到长短途区别，给出了长短途客流与出租车分离的建议，从而优化出租车资源配置。

关键词：优化　动态规划　出租车资源配置　机场调度与设计

1 问题重述

大多数乘客下飞机后要去市区（或周边）的目的地，出租车是主要的交通工具之一。送客到机场的司机会面临两个选择：（A）前往到达区排队等待载客返回市区；（B）直接放空返回市区拉客。司机的等待时间长短取决于排队出租车和乘客的数量，而司机可通过某段时间抵达的航班数量和“蓄车池”车辆数估计出等待时间。随着“互联网+”时代的到来，智能匹配管理系统实现了乘客需求预判、车辆决策调度，能够使得乘客与出租车双方的利益最大化。问题要求搜集相关数据，完成如下问题：

（1）分析影响出租车司机决策的相关因素，建立租车司机的选择决策模型。

（2）收集国内某一机场及其所在城市出租车的相关数据，根据搜集到的数据，对问题一中建立的模型进行进一步的完善，根据模型给出司机的选择方案，并分析模型的合理性。

（3）为解决出租车排队载客和乘客排队乘车的情况，设置两条并行车道，并合理安排出租车和乘客，使总乘车效率最高。

（4）提出一个可行的“优先权”方案，使得某些短途载客再次返回的出租车的收益能够与其他车均衡。

2 模型假设与符号说明

2.1 模型基本假设

1、在机场载客的出租车不能选择乘客和拒载。

2、在某时间段内抵达的航班数量和“蓄车池”里已有的车辆数是司机可观测到的确定信息。

3、不考虑突发情况或极端自然状况导致的绕行、停车、堵车等情况。

4、出租车司机的总收益仅考虑按照出租车收费标准产生的车费和所耗油费，不考虑司机加价、收取小费或者高速桥梁过路费等产生的额外费用。

5、短途载客返回机场“插队”的司机选择的返回里程和去程相同。

6、假设在市区载客的情况在各个时段内比较均衡，时段间有差别，可以将某一时段内单位里程收入视为常数。

7、假设出租车在一段行程中，速度为一个常数。

8、假设出租车在市区不会遇到前往机场的单子，且在机场及市区这两个区域之外接不到单。

2.2 符号说明

符号	说明
x_A	出租车从机场出发载客的平均路程
v_A	出租车从机场出发载客的平均行驶速度
t_w	出租车在机场“蓄车池”排队等候的时间
t_A	出租车在机场排队等候、从机场出发载客的时间总和
I_A	出租车选择在机场排队等待载客返回市区的总收益
x_0	机场距离市区的路程
x_s	出租车在市区载客的平均路程
v_s	出租车在市区载客的平均行驶速度
t_0	出租车放空从机场返回市区的时间
t_s	出租车放空返回市区、在市区载客的时间总和
I_s	出租车选择放空从机场返回市区载客的总收益

c	出租车平均每公里所耗的油价
f	出租车每公里的收费
PPMC *(Profit per Mile in City)*	出租车市区载客平均每公里的收益
t_{taxi}	出租车在机场上车点等待乘客上车所需时间
PPT *(Passenger per Taxi)*	平均每辆出租车搭载乘客的数量
n_{taxi}	当前时间段机场“蓄车池”内排队等候的出租车的数量
n_{plane}	当前时间段到达机场的航班数量
$n_{passenger}$	平均每架航班的乘客数量
r_{taxi}	当前时间段机场选择乘坐出租车离开的乘客比例
l_{taxi}	出租车车头与队伍中前一辆出租车车头之间的距离
v_{taxi}	出租车驶入或驶出上车点的速度
$v_{passenger}$	乘客步行到达上车点的速度
n	每条车道上车点的数量
t_{old}	传统模式（单一上车点）所需等待总时间
t_{new}	新模式（多上车点）所需等待总时间
$\tilde{x}$	短途载客可以再次返回“插队”的出租车的首单路程的阈值
x_{A1}	首单路程小于等于阈值时，首单短途载客平均路程
t_{A1}	首单路程小于等于阈值时，出租车在机场两次排队等候、两次从机场出发载客、首单返回机场的所需时间总和
I_{A1}	首单路程小于等于阈值时，出租车两次载客的总收益
$performance_{A1}$	首单路程小于等于阈值时，单位时间内出租车的收益
x_{A2}	首单路程大于阈值时，长途载客平均路程
t_{A2}	首单路程大于阈值时，出租车在机场排队等候、从机场出发载客的时间总和
I_{A2}	首单路程大于阈值时，出租车长途载客的总收益
$performance_{A2}$	首单路程大于阈值时，单位时间内出租车的收益
turn	出租车“优先插队”所处的排队位次

表 1　基本符号说明

3　问题分析

3.1 问题一的分析

我们主要分析了在某一时间段内，给定抵达机场的航班数量n_{plane}、“蓄车池”里已有的车辆数n_{taxi}、抵达航班中选择乘坐出租车离开的乘客比例r_{taxi}三个因素对出租车司机决策的影响。

如果司机选择方案 A（前往到达区排队等待载客返回市区），司机所耗费的总时间为机场排队等待时间和载客返回市区的时间总和，总收益为机场返回市区的车费减去油费。而如果司机选择方案 B（直接放空返回市区拉客），司机所耗费的总时间空车回市区的时间和市区载客时间的总和，总收益为市区有效载客的车费减去油费。

在当前时间段的条件下，给定三个因素的观测值，通过控制两种决策所需的总时间相同，分别计算并比较两种方案司机的总收益，得出司机在该条件下的最优策略。

3.2 问题二的分析

问题二是在问题一建立的模型的基础上，选择某个特定的城市和机场，根据我们搜集到的出租车 GPS 行车数据、国内航班信息等相关数据，对模型进行求解，给出具体条件下司机的最优选择方案。

3.3 问题三的分析

问题三的背景为机场“乘车区”有两条并行车道。为了保证车辆和乘客的安全，我们设计了天桥方便乘客到离机场较远一侧的车道，也避免斑马线对车辆通行产生的阻碍。

提高乘车效率的另一种方法是增加上车点。我们需要分别从司机和乘客的角度，将单一乘车点的传统方案和多乘车点的新方案两种方案的等待时间进行比较，提高双方的体验感。

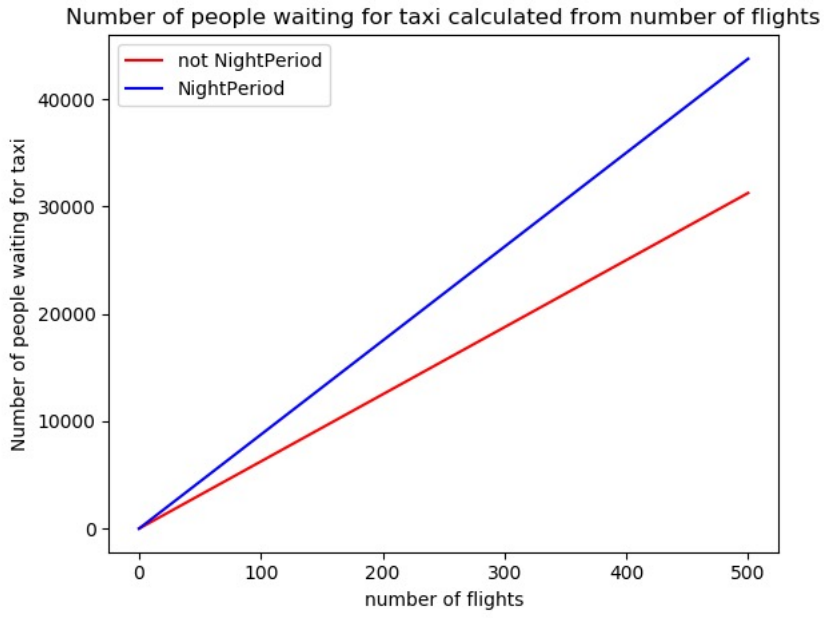
3.4 问题四的分析

我们考虑短途载客再次返回并接单的出租车，和非短途载客的出租车，两者的单位时间收益能够均衡。短途载客但不返回的出租车的收益是否能与非短途载客的出租车均衡，不在我们的考虑范围内。我们的“优先权”策略为：在前一次载客里程数x_{A1}足够短的情况下（小于阈值$\tilde{x}$），再次返回的出租车能够得到优先的排队位次$turn$，排队位次与前一次载客里程数成负相关。最后，我们根据实际数据计算出当天不同时间段的阈值$\tilde{x}$，并且推算出$turn$与x_{A1}的函数关系。

4 问题一的模型建立

4.1 影响出租车司机决策的相关因素分析

某时间段抵达机场的航班数量n_{plane}和“蓄车池”里已有的车辆数n_{taxi}是司机在决策时可以获得的信息。另外，抵达航班中选择乘坐出租车离开的乘客比例r_{taxi}也可能对出租车司机的决策产生影响。其中，一天内不同时间段的航班数量可能不同，选择乘坐出租车的乘客比例可能也不同。比如午夜降落的航班数量可能会少一些，而由于公共交通班次减少，选择乘坐出租车的乘客比较相对白天可能出现上升。



在某一时间段内，给定抵达机场的航班数量n_{plane}、“蓄车池”里已有的车辆数n_{taxi}、抵达航班中选择乘坐出租车离开的乘客比例r_{taxi}三个变量因素的输入值，平均每辆出租车搭载乘客的数量为 *PPT(Passenger per Taxi)*，若当前时间段内需要搭乘出租车的乘客数量大于排队出租车的载客数量总和，则司机的排队时间为排队出租车的数量n_{taxi}与出租车在上车点等待乘客上车所需时间t_{taxi}的乘积；否则司机则需先等满该时间段，再按照剩余排队出租车的数量决定等待时间。

因此，司机在“蓄车池”排队等候的时间t_w可以表示为

$$t_w = \begin{cases} n_{taxi} \cdot t_{taxi}\ ,\ if\ n_{plane} \cdot n_{passenger} \cdot r_{taxi} \geq PPT \cdot n_{taxi} \\ period + \dfrac{PPT \cdot n_{taxi} - n_{plane} \cdot n_{passenger} \cdot r_{taxi}}{PPT} \cdot t_{taxi}, otherwise \end{cases}$$

接下来我们将以此建立出租车司机决策模型，在不同时间段内计算并比较题中所给的 A、B 两种方案司机的收益情况，给出具体条件下司机的最优决策。

4.2 方案 A（前往到达区排队等待载客返回市区）

通过选取机场出发的出租车行驶 GPS 数据，我们可以求出从机场出发载客的平均路程x_A。出租车从机场出发载客的平均行驶速度为v_A，在“蓄车池”排队等候的时间为t_w，因此选择方案 A 司机总计耗费的时间为：

$$t_A = t_w + \frac{x_A}{v_A}$$

设出租车平均每公里所耗的油价为 c，当前时间段出租车每公里的收费为 f，因此选择方案 A 司机的总收益为：

$$I_A = -cx_A + fx_A$$

4.3 方案 B（直接放空返回市区拉客）

机场距离市区的路程为x_0，因此出租车放空从机场返回市区的时间为：

$$t_0 = \frac{x_0}{v_A}$$

出租车在市区载客的平均路程为x_s，平均行驶速度为v_s，则选择方案 B 司机总计耗费时间为：

$$t_s = t_0 + \frac{x_s}{v_s}$$

由于回市区拉客的平均每单时长可能会较低，且在单与单之间会出现一个不等的空车时间，因此为了计算方案 B 中司机的总收益，我们根据当前时间段出租车每公里的收费f以及具体数据集中司机在一段时间内平均每公里有效载客所获的车费值，引入一个量 *PPMC(Profit per Mile in City)* 表示出租车市区载客平均每公里的收益，将选择方案 B 司机的总收益表示为：

$$I_s = -c(x_0 + x_s) + PPMC \cdot x_s$$

4.4 收益比较和方案选择

控制 A、B 两种方案下司机所耗费的总时间相同，我们可以比较在相同时间内两种方案获得收益的多少。即在$t_A = t_s$的约束下，比较I_A、I_s的大小。

由时间约束$t_A = t_s$可推得

$$x_s = \left(t_w + \frac{x_A}{v_A} - \frac{x_0}{v_A}\right) v_s$$

带入 4.3 中的I_s表达式可得

$$I_s = -c(x_0 + x_s) + PPMC \cdot x_s = -cx_0 + (PPMC - c)(t_w + \frac{x_A - x_0}{v_A})v_s$$

若$I_A > I_s$，即

$$t_w < \frac{fx_A - c(x_A - x_0)}{(PPMC - c)v_s} - \frac{x_A - x_0}{v_A}$$

司机应该选择方案 A（前往到达区排队等待载客返回市区）；

若$I_A < I_s$，司机应该选择方案 B（直接放空返回市区拉客）；

若$I_A = I_s$，司机选择任一方案皆可。

5 问题二的模型求解

考虑机场客流量、机场设施的完备程度、以及可获得相关数据量的情况对建模的影响，我们以上海浦东国际机场为例建立了我们的具体模型。

5.1 数据搜集和爬取

5.1.1 出租车行车数据

出租车历史数据主要来源于 CSDN 网站上的数据集，记录了 2007 年 2 月 20 日的上海出租车行驶时每隔 15s 采样的 GPS 信息[1]，特征包括：出租车 ID、时间、经度、纬度、夹角角度、出租车的瞬时速度和出租车载客状态，数据量接近 10 万条。

5.1.2 国内航班数据

机场航班运行数据主要来源于携程，但是由于携程网站上对于历史日期航班数据的缺失情况较为严重（例如我们爬取了 2007 年 2 月 20 日和 2019 年 2 月 20 日抵达上海浦东国际机场的国内航班信息，分别仅有午夜时分抵达的 20 多条航班数据，明显不合常理），我们假设每日抵达上海浦东国际机场的航班数量相同。我们最终爬取的航班数据记录了 2019 年 9 月 12 日抵达上海浦东国际机场的国内航班信息，特征包括：出发地，航空公司，航班号，起飞时间，抵达时间，准点率，最低价，数据量 1027 条。

5.1.3 出租车收费信息

考虑到上海市出租车日间和夜间的收费标准f不同（下表 2）[2]，我们将一天 24 小时分为 4 个时间段，分别为午夜（0:00-6:00）、早上（6:00-12:00）、下午（12:00-18:00）和晚上（18:00-24:00）。之后我们将在这四个时间段下分别讨论司机的选择策略。

	日间(05:00-23:00)	夜间(23:00-05:00)
0-3 公里	14 元	18 元
3-10 公里	2.5 元/公里	3.1 元/公里
15 公里以上	3.6 元/公里	4.7 元/公里

表 2　上海市出租车收费标准

5.1.4 模型建立所需的其他基本信息

经查询可知，2019 年 9 月 12 日上海的油价（92 号汽油）为 6.60 元/升[3]，换算单位后可

[1] 来源：CSDN https://download.csdn.net/download/feng512275/10978231

[2] 来源：上海机场（集团）有限公司网站 https://www.shairport.com/cn/jcjt/index_53191.html

[3] 来源：油价网 http://youjia.chemcp.com/

知出租车平均每公里所耗的油价c为 0.56 元。现在国内民用航空飞机选用的是波音[4]或空客[5]飞机，在两家公司官网查询常见民用飞机的座位数情况后，我们假定平均每架航班的载客数$n_{passenger}$为 250 人，其中日间（早上、下午、晚上）选择乘坐出租车的乘客比例r_{taxi}为 25%，夜间选择乘坐出租车的乘客比例r_{taxi}为 35%。在机场载客的出租车在机场上车点等待乘客上车所需时间t_{taxi}为 30s，每辆出租车搭载乘客的数量 *PPT(Passenger per Taxi)*为 2 人。出租车从机场出发载客的平均行驶速度v_A为 70km/h，在市区内载客的平均行驶速度v_s为 40km/h。

5.2 数据预处理

5.2.1 出租车行驶经纬度坐标与实际路程的转换

由于我们获取的出租车数据仅含每隔 15s 采样的经纬度数据，因此我们需要根据 LBS 球面距离公式（Haversine 公式）计算已知经纬度的 a、b 两个位置之间的距离 D。

$$D = R_{earth}\arccos\left(\sin latitude\ a \cdot \sin latitude\ b + \cos latitude\ a \cdot \cos latitude\ b \cdot \cos(longitude\ a - longitude\ b)\right)$$

其中，地球平均半径R_{earth}为 6371.004 km。

5.2.2 *PPMC(Profit per Mile in City)*数值的计算

依据 5.1.3 中的上海市出租车收费标准（见上表 2）可以计算出行驶不同距离出租车司机的收入（车费）。下图 1 对比了日间和夜间车租车车费随距离的变化情况；下图 2 和下图 3 分别对比了不同时间段内，距离相同和车费相同时的出租车单数情况。

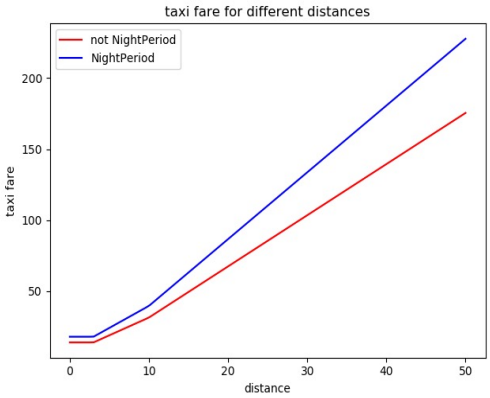


图 1　出租车车费与距离的关系

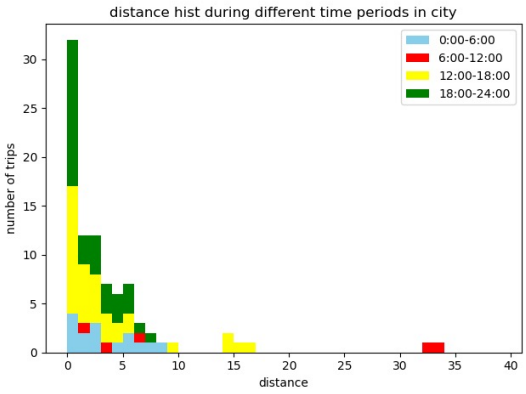


图 2　不同时间段内，相同距离条件下出租车单数比较

[4] 来源：波音中国官网 http://www.boeing.cn/
[5] 来源：空客中国官网 https://www.airbus.com/company/worldwide-presence/china/china-cn.html

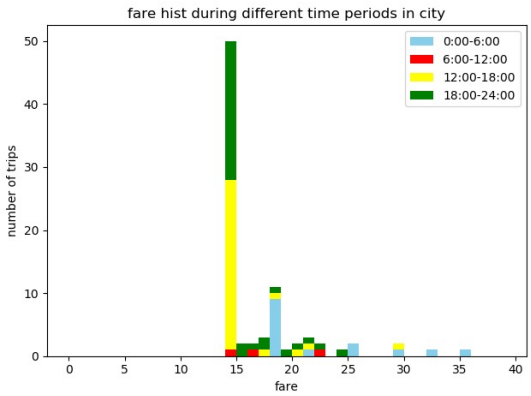


图 3　不同时间段内，相同费用条件下的出租车单数比较

假设在城区内接单是一个比较平稳的情况，*PPMC(Profit per Mile in City)*反映的就是无论载客或空车，单位距离能够获得的平均车费。根据我们获取的数据，可以分四个时间段（0:00-6:00，6:00-12:00，12:00-18：00，18:00-24:00），分别利用下列公式计算出该时间段内的 *PPMC*。

$$PPMC = \frac{\text{该时间段内市区内所有出租车的总收入}}{\text{该时间段内市区内所有出租车行驶的总里程数}}$$

其中，收入（车费）是依据 5.1.3 中的上海市出租车收费标准（见上表 2），由出租车开始载客到空车之间行驶的里程数计算得出；总收入（总车费）为所有收入（车费）的总和；总里程数为市区内每次采样之间的行驶距离总和。

计算出的 *PMCC* 在四个时段（0:00-6:00，6:00-12:00，12:00-18：00，18:00-24:00）的值分别为 1.8198，1.5043，2.3445，2.4277。

5.2.3 出租车从机场出发载客的平均路程x_A的计算

x_A是平均情况下从机场出发的出租车该单载客里程数，从收集到的数据中可以看出x_A不随时间段有显著改变，求出均值为 32.8037。

5.2.4 不同时间段内排队出租车数量和降落航班数量的情况

根据上海浦东国际机场在地图的实际位置，我们选取 31.141735°N-31.157668°N，121.800750°E-121.811835°E 所构成的矩形为我们模型中机场的位置。在不同时间段内，我们分别统计数据集中 GPS 坐标曾经出现在机场所在的矩形内的出租车数量，以此作为排队出租车的数量。午夜（0:00-6:00）、早上（6:00-12:00）、下午（12:00-18:00）和晚上（18:00-24:00）的数量分别为 34、119、104、67。

根据我们爬取的航班数据中的抵达时间，可以直接得到午夜（0:00-6:00）、早上（6:00-12:00）、下午（12:00-18:00）和晚上（18:00-24:00）降落在上海浦东国际机场的国内航班数量分别为 50、192、341、444。

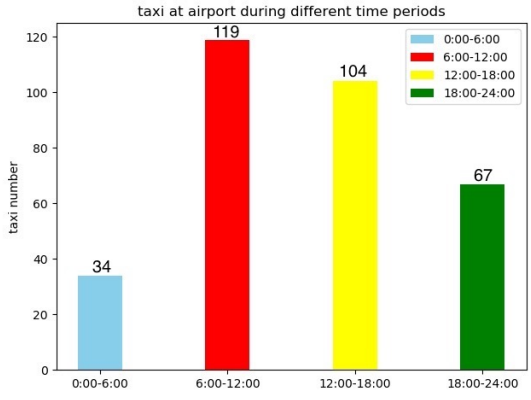


图 4　不同时间段内排队出租车的数量

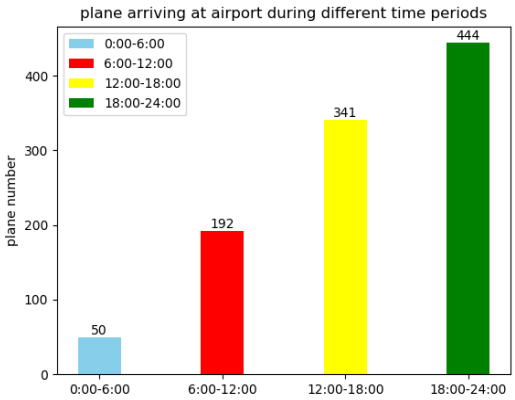


图 5　不同时间段内降落航班的数量

5.2.5 市区位置的确立

我们根据数据集中的所有出租车行车数据的经纬度作 GPS 位置的散点图（下图 6），将中心密集区域 31.233126°N-31.247842°N, 121.484058°E-121.505704°E（图 6 中的蓝色矩形区域，实际地图上为上海东方明珠电视塔附近）选定为我们的市区位置，该位置距离上海浦东国际机场的距离x_0为 44km。

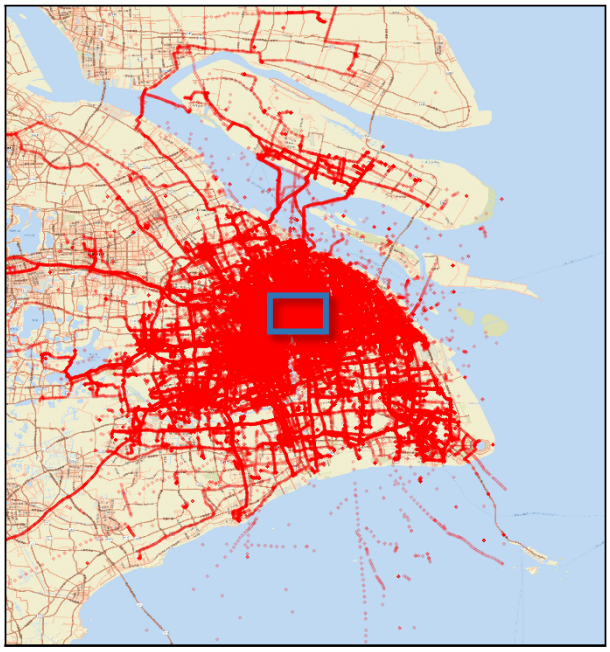


图 6　出租车行车 GPS 位置散点图

5.3 具体条件下出租车司机的最优决策讨论

根据 4.4 部分我们对司机最优决策的讨论，带入我们获取的出租车车费标准和油价等数据可以计算出某一时间段内，司机为了获取更大收益，在机场等待时间的临界值：

$$\frac{fx_A - c(x_A - x_0)}{(PPMC - c)v_s} - \frac{x_A - x_0}{v_A}$$

如果给定司机观测到当前时间段内的航班数量和排队等候出租车数量，我们可以根据 4.1 部分讨论的等候时间公式计算出实际在机场排队排队接单所需的等候时间t_w。将t_w与临界值比较可以给出具体条件下司机的最优策略。

以下是在给定航班数量、出租车数量的条件下，我们模型可以给出司机最优策略的案例：

（1）当司机在早上 10 点观察到的抵达航班数为 3，机场排队等候的出租车数量为 200，那么根据司机观测数据计算出司机的等待时间t_w为 1.6666h。根据我们建立模型中的公式可以计算出司机等待时间的临界值为 3.3330h，那么司机的最优策略为“前往到达区排队等待载客返回市区”。

（2）当司机在夜间 23 点观察到的抵达航班数为 10，机场排队等候的出租车数量为 200，那么根据司机观测数据计算出司机的等待时间t_w为 1.6667h。根据我们建立模型中的公式可以计算出司机等待时间的临界值为 2.2097h，那么司机的最优策略为“前往到达区排队等待载客返回市区”。

（3）当司机在早上 10 点观察到的抵达航班数为 10，机场排队等候的出租车数量为 400，那么根据司机观测数据计算出司机的等待时间t_w为 3.3333h。根据我们建立模型中的公式可以计算出司机等待时间的临界值为 3.3330h，那么司机的最优策略为“前往到达区排队等待载客返回市区”。

6　问题三的模型建立与求解

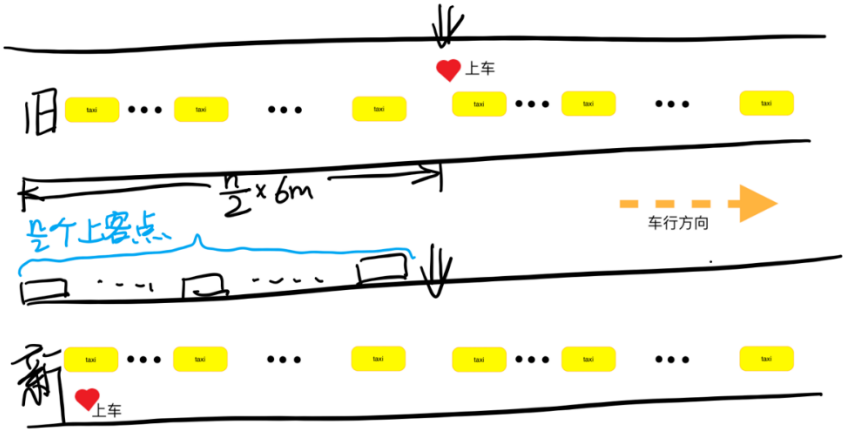


图 7　传统模式和新模式的示意图对比

6.1 考虑上车点已有出租车就位的情况（从乘客角度比较时间）

6.1.1 传统模式：每条车道只有一个上车点

考虑出租车空车已经就位，统一放行乘客，$\frac{l_{taxi}\cdot\frac{n}{2}}{v_{taxi}}$ 表示该乘客上车后到离开机场的车行时间，t_{taxi} 表示乘客装行李入后备箱以及进入车辆的时间。

如果每条车道只有一个上车点，那么队伍中的下一辆出租车需要等前一辆出租车上客完毕，并沿车道驶出“上车点”（同时该车沿车道驶入上车点），这是一个完整的载客过程。因此，该情况下每组乘客总计所需时间为

$$t_{old}=\frac{l_{taxi}\cdot\frac{n}{2}}{v_{taxi}}+t_{taxi}$$

6.1.2 新模式：每条车道有 *n* 个上车点

考虑出租车空车已经就位，统一放行乘客，则 $\frac{l_{taxi}\cdot\frac{n}{2}}{v_{passenger}}$ 表示朝车行方向反方向走的最远（最不幸的）乘客从排队出口到上车点的时间，$\frac{l_{taxi}\cdot n}{v_{taxi}}$ 表示该乘客上车后到离开机场的车行时间，t_{taxi} 表示乘客装行李入后备箱以及进入车辆的时间。

如果每条车道上有多个上车点，那么下一批出租车需要等到离中心位置最远的上车点乘客上车完毕，并驶离上车点后才能开始驶入。因此，该情况下每组乘客平均所需时间为

$$t_{new}=\frac{\frac{l_{taxi}\cdot\frac{n}{2}}{v_{passenger}}+\frac{l_{taxi}\cdot n}{v_{taxi}}+t_{taxi}}{n}$$

6.1.3 从乘客的角度比较新旧模式

作为乘客，肯定希望新的模式能够使自己减少等待时间，即$t_{old}>t_{new}$。我们假定出租车车头与队伍中前一辆出租车车头之间的距离l_{taxi}为 6m，出租车驶入或驶出上车点的速度v_{taxi}为 2m/s，乘客步行到达上车点的速度$v_{passenger}$为 0.6m/s，乘客到达上车点上车所需时间t_{taxi}为 30s。带入t_{old}和t_{new}的表达式，可以解得 *n*>1.2561 或 *n*<-15.9227。因此，我们需要设计两个以上的上车点。

6.2 考虑上车点已有乘客就位的情况（从司机角度比较等待时间）

6.1.1 传统模式：每条车道只有一个上车点

考虑出租车乘客已经就位，司机及时补位，$\frac{l_{taxi}\cdot\frac{n}{2}}{v_{taxi}}$ 表示该乘客上车后到离开机场的车行时间，$t_{taxi}\cdot\frac{n}{2}$ 表示该司机所载乘客以及（该司机进入候客区后）此乘客之前的乘客（共 $\frac{n}{2}$ 名）装行李入后备箱以及进入车辆的时间。

如果每条车道只有一个上车点，下一批乘客

$$t_{old}=\frac{l_{taxi}\cdot\frac{n}{2}}{v_{taxi}}+t_{taxi}\cdot\frac{n}{2}$$

6.1.2 新模式：每条车道有 *n* 个上车点

考虑出租车乘客已经就位，司机及时补位，则 $\frac{l_{taxi}\cdot n}{v_{taxi}}$ 表示朝车行方向反方向走的最远（最不幸的）乘客上车后，车行离开机场的时间，t_{taxi} 表示乘客装行李入后备箱以及进入车辆的时间。

如果每条车道上有多个上车点，下一批出租车司机需要等待的时间t_{new}为上一批乘客中需要走到离中心位置最远的上车点所需的时间、上一批出租车驶离上车点的时间和乘客上车时间的总和。计算公式如下：

$$t_{new}=\frac{l_{taxi}\cdot n}{v_{taxi}}+t_{taxi}$$

6.2.3 从出租车司机的角度比较新旧模式

作为出租车司机，肯定也希望新的模式能够使自己减少等待时间，即$t_{old}>t_{new}$。我们延续 6.1.3 中对一些常数的假定，带入t_{old}和t_{new}的表达式，可以解得 *n*>2.5。因此，我们需

要设计三个以上的上车点。

6.3 配合模型的建筑设计

为了使得乘车效率更高，我们提出了天桥设计，方便两个车道更安全且更有效率地使用：

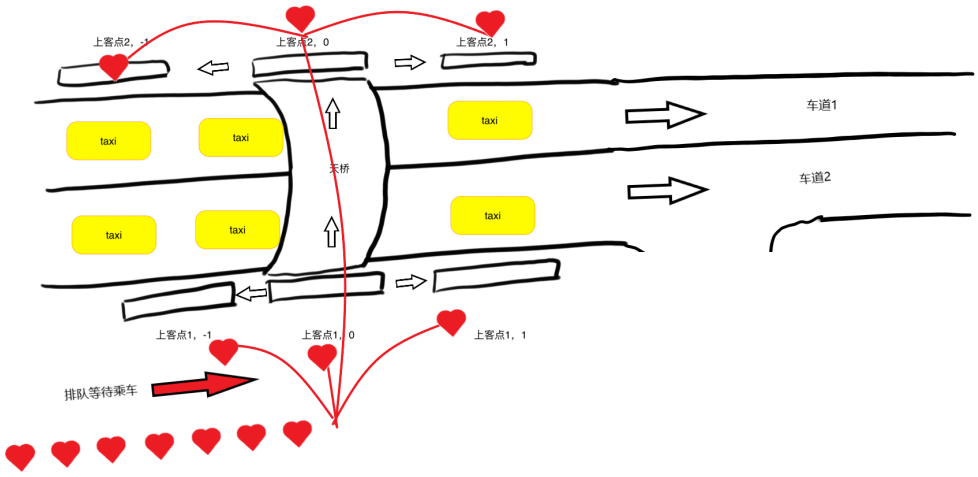


图 8　带有人行天桥、多上车点的机场出租车等候区示意图

如果采取乘客提前补位的策略，则可以保证每当车辆补齐后都可以立即开始装行李出发。

在此之前，需要让乘客提前从排队队首前往各个乘车点。（实际中可以通过观察乘客携带行李数以及乘客运动便捷程度分配乘车点）

我们继续对这一过程进行建模，假设过天桥需要 60s，两个乘车点之间需要 10s 的步行时间，单车道设置 n 个乘车点。需要乘客最少提前t_{walk}时间前往乘车点。

依据如上假设，最耗时到达的乘车点应该为天桥对面的逆车行方向最远一个。到达费时：$(t_{walk})max = 60 + 10 * \frac{n}{2}$。

7　问题四的模型建立与求解

7.1　“优先权”方案相关因素分析

我们把“优先权”设计为当司机从机场接的第一单的里程数x_{A1}小于某一阈值$\tilde{x}$时，该司机被认定为第一单短途载客，他返回机场接第二单时可以获得一个优先的排队位次$turn$，该优先排队位次与前一次载客里程数x_{A1}成负相关。我们计算的是第一单短途载客并返回的出租车，可能会获得的平均收益，因此第二次载客里程为从机场载客里程的均值x_A。我们想要使收益均衡的另一方是非短途载客的出租车，载客里程数是大于阈值$\tilde{x}$的均值x_{A2}。

7.2　A 类司机（短途载客并返回的出租车）

短途载客并返回的出租车花费的时间是：第一次排队等待载客的时间，加上往返第一次载客里程的时间，加上返回后排队等待载客的时间，再加上接到第二次载客里程的时间。设第一次短途载客里程为x_{A1}，第二次返回机场后得到的优先位次为$turn$，则花费时间t_{A1}的表达式为：

$$t_{A1} = t_w + \frac{2x_{A1} + x_A}{v_A} + turn \cdot t_{taxi}$$

短途载客并返回的收益为：短途载客的收费，加上第二次载客的收费，减去短途载客并返回的油费，再减去第二次载客的油费。如下式：

$$I_{A1} = (-2c + f)x_{A1} + (f - c)x_A$$

7.3 B 类司机（非短途载客的出租车）

非短途载客的出租车不会返回机场。因此他花费的时间是：排队等待载客的时间，加上非短途载客回市区的时间。设非短途载客里程为x_{A2}，从机场出发载客的平均速度为v_A，则花费时间t_{A2}的表达式为：

$$t_{A2} = t_w + \frac{x_{A2}}{v_A}$$

非短途载客的收益为：非短途载客的收费，减去非短途载客的油费。如下式：

$$I_{A2} = fx_{A2} - cx_{A2}$$

7.4 C 类司机（短途载客但不返回的出租车）

和非短途载客的出租车指标计量方式相同。

7.5 收益比较和“优先权”方案选择

7.5.1 阈值计算

我们考虑的短途载客并返回的出租车，一定是即使没有“优先权”，返回机场接第二单也能比不返回的收益大。因此，当返回与不返回的收益相等时，我们可以确定允许返回机场接第二单的“短途阈值”。四个时间段两者的单位时间收益-里程图如下：

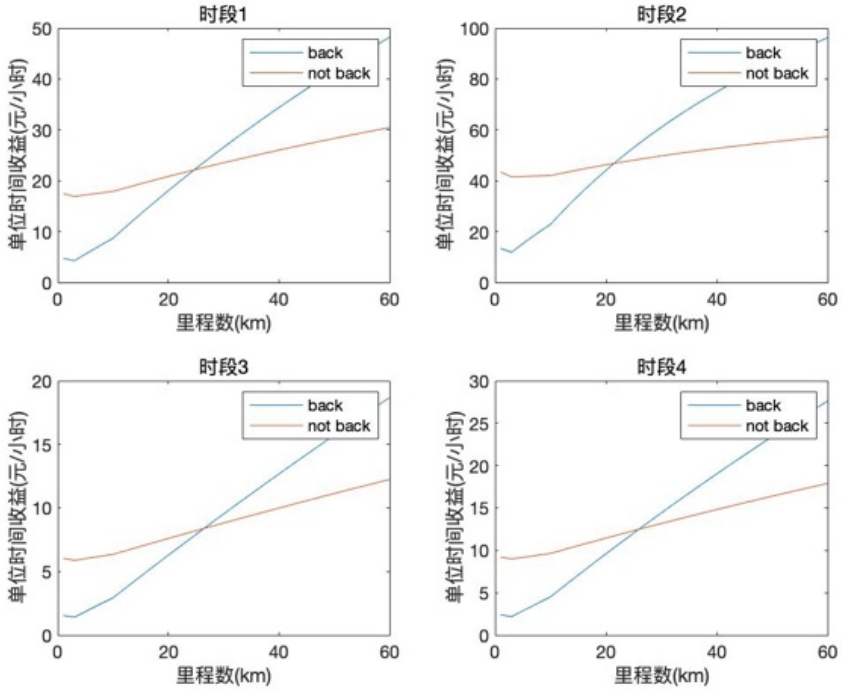


图 9　单位时间收益-里程图

数学方程如下：

$$\frac{(f-c)\tilde{x}}{t_w+\frac{\tilde{x}}{v_A}}=\frac{(-2c+f)\tilde{x}+(f-c)x_A}{2t_w+\frac{2\tilde{x}+x_A}{v_A}}$$

求解该方程可以得出四个时间段的短途阈值$\tilde{x}$分别为 25、22、26、26km。

7.5.2 优先权方案

我们考虑短途载客并返回的出租车，和非短途载客的出租车的单位时间收益相等，得到短途里程x_{A1}和返回后的优先排队位次$turn$两者的关系。如下式：

$$\frac{I_{A1}}{t_{A1}}=\frac{I_{A2}}{t_{A2}}$$

最后解得优先排队位次$turn$和短途载客里程x_{A1}的函数关系式为：

$$turn=\frac{t_w\cdot[(f-c)(x_A-x_{A2})+(f-2c)x_{A1}]-\frac{f\cdot x_{A1}\cdot x_{A2}}{v_A}}{(f-c)x_{A2}\cdot t_{taxi}}$$

7.5.3 具体条件下的优先权方案

我们实现了可以利用动态规划的数学模型实现用户输入约束条件和观察值，给出机场出租车管理系统应该给出的量化优先权。

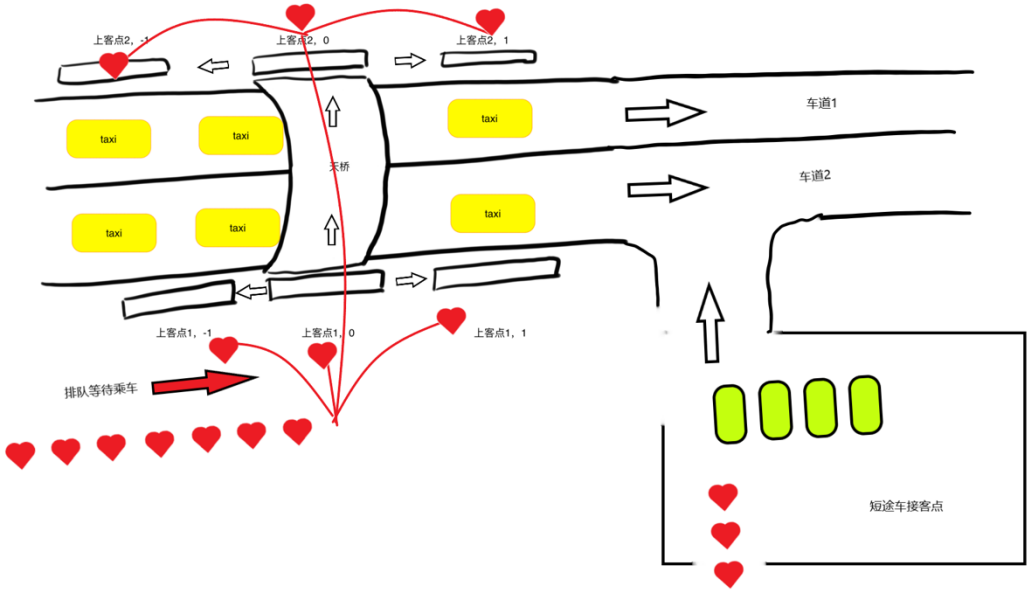
```
## Policy4.py
please input the hour of Day:
10
please input the distance for this trip:
(should be smaller than threshold26)
10
please input the approximate waiting time:
2
your turn to go should be: 269

Process finished with exit code 0
```

7.6 设计长短途分离的候车方案

由于综合考虑到第三问中长短途司机的接单距离未知性和收入不均衡性，我们提出抗议将出租车等候区域划分为长途和短途。分别排队，分别离开。短途车仍然可以享受相对优先，返回后等待系统叫号，叫号后即可接客，即刻离开。

这样可以在不打消短途司机积极性的前提下给司机选择权。同时分散客流，使得短途车的乘客可以比较快地离开。



8 模型的评价

8.1 模型的优点

（1）在问题一、二中，我们考虑到白天和夜间出租车收费不同、选择乘坐出租车的乘客比例不同等因素，将全天 24 小时划分为四个时间段进行讨论。类似地，在问题四中，我们的“优先权方案”同样分四个时间段进行讨论，给出了四个短途返回的临界值。

（2）在问题一、二中，对于司机观测到的航班数量和机场排队出租车数量，我们的模型可以给出预计的等待时间表达式，后期可以制作成交互软件，方便出租车司机在不同条件下灵活地选择最优方案。类似地，在问题四中，司机可以根据第一单短途的不同距离在第二次接单时得到不同的优先排队位次，使司机的收益最大化。

（3）在问题三设计机场出租车的等候区时，双向考虑了司机和乘客的体验感（双方都要减少等待时间），使设计更具人性化。

8.2 模型的不足

（1）我们用于构建模型的出租车行车数据量较少，仅占上海市出租车市场的十分之一左右。我们收集的航班数据与出租车行车数据的日期不匹配，没有考虑季节、节假日等因素对航班数量、出租车数量的影响，也没有考虑随着时间推移出租车市场增加出租车投放量的情况。

（2）我们仅以航班抵达时间信息来估计乘坐出租车的乘客数量，忽略了实际情况下飞机延误时间、乘客提取托运行李所需时间等，因此在某一时间段内的乘坐出租车乘客数量不准确。

9 参考文献

[1] Ji Y, Cao Y, Du Y, et al. Comparative Analyses of Taxi Operations at the Airport[J]. Transportation Research Procedia, 2017, 25:2227-2237.

[2] Gonzales E J , Yang C , Morgul E F , et al. Modeling Taxi Demand with GPS Data from Taxis and Transit[J]. Choice Models, 2014.

[3] Smeltink J W, Soomer M J, Waal P R D, et al. An Optimisation Model for Airport Taxi Scheduling⋆[J]. 2008.
[4] Costa D . Performance and Design of Taxi Services at Airport Passenger Terminals[J]. Master in Ctis, 2009.
[5] 吴娇蓉, 李铭, 梁丽娟. 综合客运枢纽出租车上客点管理模式和效率分析[J]. 交通信息与安全, 2012, 30(4):18-23.
[6] 姜启源, 谢金星, 叶俊. 数学模型.第 4 版[M]. 高等教育出版社, 2011.

附录　代码部分

代码思路：

DecisionNo1.py 第一二问的模型用代码做出可交互版本，帮助到达机场的司机依据当前时间，观测到的航班数和当前排队车辆数来决定是进入机场排队还是返回市区区域

　　calcNumPeople 以航班数和当前时段为输入估计乘车人数

　　calcTw 以人数，出租车数量，饥饿等待时间（当没有乘客在排队，而出租车却有剩余）为输入估计司机排队接客的等待时间

　　decide 做决定

　　calcXa 从机场出发的平均载客里程

　　calcPPMC 根据时段返回相应的 PPMC

distSumInCity.py 计算不同时段下市区出租车发生的里程数之和

　　distanceSumInCity

inCityIncome.py 计算不同时段下市区出租车发生的车费之和

　　calcInCityCarryDist

　　calcInCityIncome

LonLat2Dist.py 从经纬度计算距离

drawGraphs.py 画出论文中需要的部分图示

regressionWithThresholdDistanceForQueuingPolicy.py 根据不同临界值计算临界值左右的距离均值

filterDepArrLinesFromFile.py 从所有文件中抽取机场出发的数据

taxiQueuing.py 计算不同时段下在机场排队出租车的平均值

Policy4.py 第四问的模型实现（未完成）

howManyPeopleInACabFromAirportToCity.py 计算不同时段下在机场乘坐出租车的人数

　　drawCabPassengerNumHist 画出直方图

utils.py 共享函数

　　fare 根据出租车公司规定计算不同时段的阶梯车费

　　get_filelist 文件读写相关函数

```
1.  """
2.  DecisionNo1.py
3.  """
4.  from utils import fare
5.  from numpy import array
6.
7.
8.  def calcNumPeople(numFlights, NightPeriod=False):
9.      """
10.     suppose the percentage of passengers from an arrival flight who is going
     to take a cab is 25%
11.     suppose the percentage increase of passengers from an arrival flight, du
    ring NightPeriod, who is going to take a cab is 10%
12.     """
13.     return numFlights * 250 * (0.25 + int(NightPeriod) * 0.1)
```

```
14.
15.
16. def calcTw(numPeople, numTaxi, hungryTime=0, TloadPassenger=(1 / 120)):
17.     """
18.     on average 2 person in one cab
19.     assume the time for each cab to run after reaching the head of the queue
     is 30 second (1/120 hour)
20.     define hungryTime as the time the cab has to wait till there is new pass
    engers to come to the parking lot,
21.     only happen when hungry of passengers
22.     """
23.     if numPeople > 2 * numTaxi:
24.         return numTaxi * TloadPassenger
25.     else:
26.         return hungryTime + numTaxi * TloadPassenger
27.
28.
29. def decide(tw, hour, Xa=32.8, PPMC=1.2048954796495943, Va=70, Vs=40, X0=44,
    c=0.56):
30.     """
31.     tw: waiting hours calculated by experience and observation (given as inp
    ut)
32.     Xa: the distance between the airport and where the passenger is headed
33.     PPMC: expected income per mile in city area
34.     Va, Vs: speed of cab in rural area (from airport to city) and in city ar
    ea
35.     X0: distance from airport to the center of the city
36.     c: cost of oil per mile
37.     """
38.     NightPeriod = hour < 5 or hour >= 23
39.     Fa = fare(Xa, NightPeriod)
40.     expectedTw = (X0 - Xa) / Va + ((c * (X0 - Xa) + Fa) / (PPMC - c)) / Vs
41.     print("Max endurable waiting time:", expectedTw)
42.     print("Expected waiting time calculated from your inputs:", tw)
43.     if tw < expectedTw:
44.         print("Stay At Airport!")
45.     else:
46.         print("Go To City!")
47.
48.
49. def calcXa(TimePeriod):
50.     """
51.     Taxi Distance departing from airport:
52.     [ 5.7522861 , 12.70583893, 36.2344826 , 31.66968362, 33.68858315,
```

```
       39.86210384, 19.20838719, 43.82756287, 46.66738947,  2.16244103,
       32.28587466, 25.9675955 , 26.81828985, 46.33381685, 41.65848003,
       36.64776083, 46.48312939, 25.25266788, 34.95457476, 37.4034993 ,
       42.98161996, 37.00201414, 49.34189856, 48.79110121, 62.90995982,
       36.01832898, 51.36597839, 31.92488509, 41.8439101 , 38.88536259,
        4.37704512,  1.05851537,  1.07810724,  0.71190109,  2.20795834,
       41.55370761, 33.01905746, 26.4317773 , 42.3122994 , 21.23612477,
       38.69043218, 33.73861671, 41.74722684, 39.21889462, 34.57166724,
       36.26480028,  1.22316519,  3.90013579, 27.8863642 , 20.29901287,
       46.52290695, 44.1639085 , 39.38744551, 27.94120026, 51.20579324,
       40.69499251, 47.38605861, 33.12139269, 37.0515469 , 30.80837309,
       35.52436541, 35.14999693, 41.22600298,  7.10265778, 40.24289292,
       45.08590153, 46.35742046, 36.43361243, 36.00296785, 40.36565771,
       41.64899155,  0.93598353, 30.88057538, 31.49763758, 48.75461939,
       38.76133913, 31.83685927, 42.0370264 , 39.643243  , 43.60449062,
       40.5935389 ,  2.50847149, 42.93978407, 35.75856251, 36.9608126 ]
    mean: 32.8036864231462
    Not sensitive to TimePeriod!!!
    """
    return 32.8036864231462


def calcPPMC(TimePeriod):
    """
    PPMC @ airport
    ## inCityIncome.py
        TimePeriod 1
        330.8134554199392
        TimePeriod 2
        280.86369463847456
        TimePeriod 3
        684.301355754978
        TimePeriod 4
        517.5485014292086

    ## distSumInCity.py
        TimePeriod 1
        181.78146607852437
        TimePeriod 2
        186.7074013288872
        TimePeriod 3
        291.8755629493137
        TimePeriod 4
        213.18291336295698
```

```
    """
    PPMCDict = [1.8198414973561567, 1.5042986653953263, 2.3444969110819733, 2.4277203705723385]
    return PPMCDict[int(TimePeriod) - 1]


def isNightPeriod(hour):
    return hour >= 23 or hour < 5


if __name__ == '__main__':
    print("By Default:\ndecide(tw, Xa, PPMC, Va, Vs, NightPeriod=False, X0=44, c=0.56)")

    numFlight = int(input("please input num of flights arriving at the airport:"))
    numTaxi = int(input("please input num of taxi waiting at airport:"))
    hourOfTime = int(input("please input the time for this decision:\n (simply input the hour ~)"))
    NightPeriod = False
    if hourOfTime >= 23 or hourOfTime < 5:
        NightPeriod = True
    TimePeriod = hourOfTime / 6 + 1

    decide(calcTw(calcNumPeople(numFlight, NightPeriod=NightPeriod), numTaxi), hourOfTime, PPMC=calcPPMC(TimePeriod))
    # tw = array(
    #     [0.4167, 0.25, 1.0, 1.0, 1.0, 1.1667, 1.1667, 1.3333, 0.8333, 0.16667, 1.5, 1.3333, 0.91667, 0.3333, 0.5833,
    #      0.8333, 1.1667, 1.8333, 1.3333, 0.6667, 0.4167, 1.0, 1.5, 0.9167]) * 5
    # for i in range(24):
    #     decide(tw=tw[i], hour=i, PPMC=calcPPMC(i // 6 + 1))
```

```
"""
distSumInCity.py
"""
from utils import fare
from LonLat2Dist import *
import matplotlib.pyplot as plt  # 可视化绘制
import numpy as np

allSums = []
```

```
10.
11.
12. def distanceSumInCity(TimePeriod=None):
13.     with open('inCityAll.csv') as ff:
14.         cab = 0
15.         lines = ff.readlines()
16.         lines = [i.strip().split(',') for i in lines]
17.         lastLonLat = None
18.         sum4indv = 0
19.         for arr in lines:
20.             if TimePeriod and int(arr[1].split(' ')[1][0:2]) / 6 + 1 != TimeP
    eriod:
21.                 continue
22.             if arr[0] != cab:
23.                 cab = arr[0]
24.                 sums.append(sum4indv)
25.                 sum4indv = 0
26.                 lastLonLat = None
27.             else:
28.                 thisLonLat = (float(arr[2]), float(arr[3]))
29.                 if lastLonLat and thisLonLat:
30.                     sum4indv += get_distance_hav(lastLonLat[0], lastLonLat[1]
    , thisLonLat[0], lastLonLat[1])
31.                 lastLonLat = thisLonLat
32.
33.
34. bins = range(0, 101, 10)
35. colors = ["SkyBlue", "red", "yellow", "green"]
36. labels = ["0:00-6:00", "6:00-12:00", "12:00-18:00", "18:00-24:00"]
37.
38. for TimePeriod in range(1, 5):
39.     sums = []
40.     print("TimePeriod", TimePeriod)
41.     distanceSumInCity(TimePeriod)
42.     # print(sums)
43.     allSums.append(sums)
44.     print(sum(sums))
45.
46. plt.hist(allSums, bins=bins, color=colors, histtype="bar", rwidth=10,
47.          stacked=True, label=labels)
48. plt.xlabel("distance")
49. plt.ylabel("number of trips")
50. plt.title("distance hist during different time periods in city")
51. plt.legend()
```

```
52. plt.show()
```

```
1.  """
2.  drawGraphs.py
3.  """
4.  import matplotlib.pyplot as plt  # 可视化绘制
5.  import numpy as np
6.  from DecisionNo1 import calcNumPeople
7.
8.  fig, ax = plt.subplots()
9.  ind = np.linspace(0, 500, 10)
10.
11. fday, fnight = [], []
12. for i in ind:
13.     fday.append(calcNumPeople(i, False))
14.     fnight.append(calcNumPeople(i, True))
15. ax.plot(ind, fday, c='r', label='not NightPeriod')
16. ax.plot(ind, fnight, c='b', label='NightPeriod')
17. ax.set_ylabel('Number of people waiting for taxi')
18. ax.set_xlabel('number of flights')
19. ax.set_title('Number of people waiting for taxi calculated from number of fl
    ights')
20. plt.legend()
21. plt.show()
```

```
"""
filterDepArrLinesFromFile.py
"""
import sys
import os

from utils import *


def filterDepArrLinesFromFile(path):
    with open(path, 'r+') as f:
        flag = -1  # 是否从下客到上客
        depFromA = False  # 从机场出发,且单还未结束
        lines = f.readlines()
        lines = [i.strip().split(',') for i in lines]
        for arr in lines:
            if arr[6] != flag:
                flag = int(arr[6])
                # pudong
                if flag > 0 and (31.141735 - 5e-6) < float(arr[3]) < (31.157668 + 5e-6) \
                        and (121.800750 - 5e-6) < float(arr[2]) < (121.811835 + 5e-6):
                    # hongqiao
                    # if flag == 1 and 31.168064 < arr[3] < 31.219602 and 121.322830 < arr[2] < 121.355780:
                    depFromA = True
                    print(','.join([str(i) for i in arr]))
                elif flag == 0 and depFromA:
                    print(','.join([str(i) for i in arr]))
                    depFromA = False  # 本单结束
                elif flag == 0:
                    depFromA = False  # 本单结束


if __name__ == '__main__':
    # filterDepArrLinesFromFile(sys.argv[1])
    list = get_filelist('.', [])
    for e in list:
        if e.split('.')[-1] == 'txt':
            # print(e)
            filterDepArrLinesFromFile(e)
```

```
"""
howManyPeopleInACabFromAirportToCity.py
"""
import matplotlib.pyplot as plt  # 可视化绘制
import numpy as np


def drawCabPassengerNumHist(hour=None):
    allPNs = []
    with open("filtered(from airport to city).csv") as f:
        lines = f.readlines()
        lines = [i.strip().split(',') for i in lines]
        for arr in lines:
            if int(arr[1].split(' ')[1][0:2]) == hour:
                allPNs.append(arr[6])
    # print("number of taxi at airport at hour", hour, "is ", len(allPNs))
    print(len(allPNs))
    return allPNs


if __name__ == '__main__':
    # tmp = np.array(drawCabPassengerNumHist())
    # fig, ax = plt.subplots()
    # plt.hist(tmp)
    # plt.show()
    for i in range(24):
        drawCabPassengerNumHist(i)
```

```
"""
inCityIncome.py
"""
from utils import fare
from pandas import DataFrame
from LonLat2Dist import *
import matplotlib.pyplot as plt  # 可视化绘制
import numpy as np

allSums = []


def calcInCityIncome(TimePeriod=None):
    with open('inCity.csv') as ff:
        lines = ff.readlines()
```

```
        lines = [i.strip().split(',') for i in lines]
        lastLonLat = None
        NightPeriod = None
        for arr in lines:
            hour = int(arr[1].split(' ')[1][0:2])
            if TimePeriod and hour / 6 + 1 != TimePeriod:
                continue
            if int(arr[6]) == 0:
                thisLonLat = (float(arr[2]), float(arr[3]))
                if lastLonLat and thisLonLat:
                    dist = get_distance_hav(lastLonLat[0], lastLonLat[1], th
isLonLat[0], lastLonLat[1])
                    sums.append(fare(dist, NightPeriod))
                lastLonLat = None
            else:
                NightPeriod = hour < 5 or hour >= 23
                lastLonLat = (float(arr[2]), float(arr[3]))


def calcInCityCarryDist(TimePeriod=None):
    with open('inCity.csv') as ff:
        lines = ff.readlines()
        lines = [i.strip().split(',') for i in lines]
        lastLonLat = None
        NightPeriod = None
        for arr in lines:
            hour = int(arr[1].split(' ')[1][0:2])
            if TimePeriod and hour / 6 + 1 != TimePeriod:
                continue
            if int(arr[6]) == 0:
                thisLonLat = (float(arr[2]), float(arr[3]))
                if lastLonLat and thisLonLat:
                    dist = get_distance_hav(lastLonLat[0], lastLonLat[1], th
isLonLat[0], lastLonLat[1])
                    sums.append(dist)
                lastLonLat = None
            else:
                NightPeriod = hour < 5 or hour >= 23
                lastLonLat = (float(arr[2]), float(arr[3]))


bins = range(0, 40, 1)
colors = ["SkyBlue", "red", "yellow", "green"]
labels = ["0:00-6:00", "6:00-12:00", "12:00-18:00", "18:00-24:00"]
```

```
58.
59. for TimePeriod in range(1, 5):
60.     sums = []
61.     print("TimePeriod", TimePeriod)
62.     # calcInCityCarryDist(TimePeriod)
63.     calcInCityIncome(TimePeriod)
64.     # print(sums)
65.     print(sum(sums))
66.     allSums.append(sums)
67.
68. plt.hist(allSums, bins=bins, color=colors, range=(0, 40),
69.          histtype="bar", rwidth=10,
70.          stacked=True, label=labels)
71. # plt.xlabel("distance")
72. plt.xlabel("fare")
73. plt.ylabel("number of trips")
74. # plt.title("distance hist during different time periods in city")
75. plt.title("fare hist during different time periods in city")
76.
77. plt.legend()
78. plt.show()
```

```
1.  """
2.  LonLat2Dist.py
3.  """
4.  # coding=utf-8
5.  # import MySQLdb
6.  import sys
7.
8.  from math import sin, asin, cos, radians, fabs, sqrt
9.
10. EARTH_RADIUS = 6371  # 地球平均半径，6371km
11.
12.
13. def hav(theta):
14.     s = sin(theta / 2)
15.     return s * s
16.
17.
18. def get_distance_hav(lat0, lng0, lat1, lng1):
19.     "用 haversine 公式计算球面两点间的距离。"
20.     # 经纬度转换成弧度
```

```
    lat0 = radians(lat0)
    lat1 = radians(lat1)
    lng0 = radians(lng0)
    lng1 = radians(lng1)

    dlng = fabs(lng0 - lng1)
    dlat = fabs(lat0 - lat1)
    h = hav(dlat) + cos(lat0) * cos(lat1) * hav(dlng)
    distance = 2 * EARTH_RADIUS * asin(sqrt(h))
    return distance


if __name__ == '__main__':
    with open(sys.argv[1]) as f:
        lines = f.readlines()
        flag = False
        dep, arr = None, None
        for line in lines:
            line = line.strip().split(',')
            if line[6] == '1':
                dep = float(line[2]), float(line[3])
            elif dep is not None:
                arr = float(line[2]), float(line[3])
                print(get_distance_hav(dep[0], dep[1], arr[0], arr[1]))
                dep, arr = None, None
```

```
"""
regressionWithThresholdDistanceForQueuingPolicy.py
"""
import numpy as np  # 快速操作结构数组的工具
import matplotlib.pyplot as plt  # 可视化绘制
from sklearn.linear_model import LinearRegression  # 线性回归
import numpy as np

# 样本数据集，第一列为x，第二列为y，在x和y之间建立回归模型

Xthreshold2meanForShortDist = []
Xthreshold2meanForLongDist = []

"""
机场出发的单子的里程数
"""
```

```
allDists = np.array([5.7522861, 12.70583893, 36.2344826, 31.66968362, 33.68858315,
                     39.86210384, 19.20838719, 43.82756287, 46.66738947, 2.16244103,
                     32.28587466, 25.9675955, 26.81828985, 46.33381685, 41.65848003,
                     36.64776083, 46.48312939, 25.25266788, 34.95457476, 37.4034993,
                     42.98161996, 37.00201414, 49.34189856, 48.79110121, 62.90995982,
                     36.01832898, 51.36597839, 31.92488509, 41.8439101, 38.88536259,
                     4.37704512, 1.05851537, 1.07810724, 0.71190109, 2.20795834,
                     41.55370761, 33.01905746, 26.4317773, 42.3122994, 21.23612477,
                     38.69043218, 33.73861671, 41.74722684, 39.21889462, 34.57166724,
                     36.26480028, 1.22316519, 3.90013579, 27.8863642, 20.29901287,
                     46.52290695, 44.1639085, 39.38744551, 27.94120026, 51.20579324,
                     40.69499251, 47.38605861, 33.12139269, 37.0515469, 30.80837309,
                     35.52436541, 35.14999693, 41.22600298, 7.10265778, 40.24289292,
                     45.08590153, 46.35742046, 36.43361243, 36.00296785, 40.36565771,
                     41.64899155, 0.93598353, 30.88057538, 31.49763758, 48.75461939,
                     38.76133913, 31.83685927, 42.0370264, 39.643243, 43.60449062,
                     40.5935389, 2.50847149, 42.93978407, 35.75856251, 36.9608126])

for Xthreshold in np.arange(round(min(allDists)) + 1, round(max(allDists)), 0.5):
    above = np.mean(allDists[allDists < Xthreshold])
    below = np.mean(allDists[allDists >= Xthreshold])
    Xthreshold2meanForShortDist.append([Xthreshold, Xthreshold ** 2, Xthreshold ** 3, above])
    Xthreshold2meanForLongDist.append([Xthreshold, Xthreshold ** 2, Xthreshold ** 3, below])

```

```
# 生成 X 和 y 矩阵
Xthreshold2meanForShortDist = np.array(Xthreshold2meanForShortDist)
Xthreshold2meanForLongDist = np.array(Xthreshold2meanForLongDist)
# X = Xthreshold2meanForShortDist[:, :-1] #三次拟合

X = Xthreshold2meanForShortDist[:, 0:1]  # 一次拟合
yShort = Xthreshold2meanForShortDist[:, -1]
yLong = Xthreshold2meanForLongDist[:, -1]

# ========线性回归========
modelShort = LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
modelShort.fit(X, yShort)  # 线性回归建模
print('#' * 30)
print("短程平均关于临界值:")
print('系数矩阵:\n', modelShort.coef_, modelShort.intercept_)
print('线性回归模型:\n', modelShort)

modelLong = LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
modelLong.fit(X, yLong)  # 线性回归建模
print('#' * 30)
print("长程平均关于临界值:")
print('系数矩阵:\n', modelLong.coef_, modelLong.intercept_)
print('线性回归模型:\n', modelLong)

# 使用模型预测
# predicted = model.predict(X)
# 绘制散点图 参数：x 横轴 y 纵轴
plt.scatter(X[:, 0], yShort, marker='x', c='r')
plt.scatter(X[:, 0], yLong, marker='x', c='b')

x = np.linspace(round(min(allDists)), round(max(allDists)), 100)

# y = modelShort.coef_[2] * x ** 3 + modelShort.coef_[1] * x ** 2 + modelShort.coef_[0] * x + modelShort.intercept_
y = modelShort.coef_[0] * x + modelShort.intercept_
plt.plot(x, y, color="red", linewidth=1.5)

# y = modelLong.coef_[2] * x ** 3 + modelLong.coef_[1] * x ** 2 + modelLong.coef_[0] * x + modelLong.intercept_
y = modelLong.coef_[0] * x + modelLong.intercept_

plt.plot(x, y, color="blue", linewidth=1.5)
```

```
# plt.scatter(X[:, 1], y, marker='x', c='g')
# plt.scatter(X[:, 2], y, marker='x', c='b')
# plt.plot(X, predicted, c='r')

# 绘制 x 轴和 y 轴坐标
plt.xlabel("xThreshold")
plt.ylabel("X(short/Long) mean")

# 显示图形
plt.show()
```

```
"""
taxiQueuing.py
"""
import sys
import os
import numpy as np
from utils import *
import matplotlib.pyplot as plt  # 可视化绘制

taxiSet1, taxiSet2, taxiSet3, taxiSet4 = set(), set(), set(), set()
taxiSetMap = {1: taxiSet1, 2: taxiSet2, 3: taxiSet3, 4: taxiSet4}


def grepQueuingAtAirport(path):
    with open(path, 'r+') as f:
        lines = f.readlines()
        lines = [i.strip().split(',') for i in lines]
        for arr in lines:
            # pudong
            if (31.141735 - 5e-6) < float(arr[3]) < (31.157668 + 5e-6) \
                    and (121.800750 - 5e-
6) < float(arr[2]) < (121.811835 + 5e-6):
                # hongqiao
                # if flag == 1 and 31.168064 < arr[3] < 31.219602 and 121.32
2830 < arr[2] < 121.355780:
                i = int(arr[1].split(' ')[1][0:2]) // 6
                taxiSetMap[i + 1].add(arr[0])
                # print(','.join([str(i) for i in arr]))


if __name__ == '__main__':
```

```
    # filterDepArrLinesFromFile(sys.argv[1])
    list = get_filelist('.', [])
    for e in list:
        if e.split('.')[-1] == 'txt':
            # print(e)
            grepQueuingAtAirport(e)
    taxiHist = [(k, len(v)) for k, v in taxiSetMap.items()]
    print(taxiHist)

    taxiHist = [(1, 34), (2, 119), (3, 104), (4, 67)]
    fig, ax = plt.subplots()
    ind = np.arange(1, 5, 1)
    width = 0.35  # the width of the bars
    rects1 = ax.bar(taxiHist[0][0], taxiHist[0][1], width, color='SkyBlue', label='0:00-6:00')
    rects2 = ax.bar(taxiHist[1][0], taxiHist[1][1], width, color='red', label='6:00-12:00')
    rects3 = ax.bar(taxiHist[2][0], taxiHist[2][1], width, color='yellow', label='12:00-18:00')
    rects4 = ax.bar(taxiHist[3][0], taxiHist[3][1], width, color='green', label='18:00-24:00')
    ax.set_ylabel('taxi number')
    ax.set_title('taxi at airport during different time periods')
    plt.xticks(ind, ('0:00-6:00', '6:00-12:00', '12:00-18:00', '18:00-24:00'))
    ax.legend()
    plt.show()
```

```
"""
utils.py
"""
import os
import numpy as np
import matplotlib.pyplot as plt  # 可视化绘制


def fare(S, NightPeriod=False):
    if NightPeriod:
        if S < 3:
            return 18
        elif S < 10:
            return 18 + (S - 3) * 3.1
```

```
        else:
            return 18 + 7 * 3.1 + 4.7 * (S - 10)
    else:
        if S < 3:
            return 14
        elif S < 10:
            return 14 + (S - 3) * 2.5
        else:
            return 14 + 7 * 2.5 + 3.6 * (S - 10)


def get_filelist(dir, Filelist):
    newDir = dir
    if os.path.isfile(dir):
        Filelist.append(dir)
    elif os.path.isdir(dir):
        for s in os.listdir(dir):
            newDir = os.path.join(dir, s)
            get_filelist(newDir, Filelist)
    return Filelist


if __name__ == '__main__':
    fig, ax = plt.subplots()
    ind = np.linspace(0, 50, 100)
    fday, fnight = [], []
    for i in ind:
        fday.append(fare(i, False))
        fnight.append(fare(i, True))
    ax.plot(ind, fday, c='r', label='not NightPeriod')
    ax.plot(ind, fnight, c='b', label='NightPeriod')
    ax.set_ylabel('taxi fare')
    ax.set_xlabel('distance')
    ax.set_title('taxi fare for different distances')
    plt.legend()
    plt.show()
```