# 机场出租车运营模型

## 摘要

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。国内多数机场都是将送客（出发）与接客（到达）通道分开的。送客到机场的出租车司机都将会面临两个选择：进入蓄车池等待载客则需要付出相应的时间成本，而选择放空直接返回市区则需要付出空载费用和可能损失潜在的载客收益。在某时间段抵达的航班数量和“蓄车池”里已有的车辆数是司机可观测到的确定信息，如何根据已知信息去做出判断使得利益最大化就十分重要了。

针对问题一，我们采用“单位时间净收益”的评价体系，分别计算A方案与B方案的单位时间净收益，最后选取单位时间净收益更大的方案。在计算A方案的单位时间净收益时，我们着重建立了计算基于算法模拟的排队时间模型。该模型细致地考虑了一段时间内到达的航班信息list、当前排队出租车数L、当前等待上车的乘客数$p_0$对排队时间的影响。该模型的亮点为，对航班到达信息进行逐条模拟，保证了输出的精确度。

针对问题二，选取上海市浦东国际机场及该市的出租车为研究对象，我们深入搜集了上海市浦东国际机场的相关信息，如全天的航班信息表、上海市人口密度地图、上海出租车计价公式等。其中，我们依据上海市人口密度地图，创造性的建立了人口密度与打车里程概率密度的关系模型，使得数据更为贴近实际。在分析模型的依赖性时，我们取一系列的适当的当前等待上车的乘客$p_0$值，然后在每个$p_0$值下，在不同的到达时间t和排队出租车数L下，绘制决策散点图。基于这些散点图，我们分析了模型对相关输入的依赖性以及模型的合理性。

针对问题三，我们首先考虑如何保证车辆和乘客安全，经过分析论证，按照一种规则，我们先得到了一种出租车驶入与驶出的方案，并且在方案中可以恰当地安排乘客上车的时机，使得既保证了乘客的安全，又能尽快地上车，减少乘客的等待时间，提高了乘客的服务舒适感。在此方案下，我们又探讨了泊车区可同时等待载客的出租车的数量对于单位时间驶出率的影响，且由于乘客上车时间的不稳定性也会影响单位时间驶出率。我们建立了一个数学模型，考虑了前述影响因素，算得了不同的泊车位数量所对应的单位时间驶出率，又综合考虑了管理成本等实际因素，得出了一个最优

方案，这样使得司机的收益也会得到提高，同时管理部门所付出的成本也不会浪费太多。在本题中，我们考虑到了现实状况下不确定性所带来的影响，是一个亮点。最后还对于得到的最优方案的每个车位的负荷率及可行的调整方法进行了简单的讨论。

在问题四中，我们基于问题一的模型，引入“插队系数”a 作为“优先权”的衡量标准。插队系数的引入，保证了短途载客司机的“单位时间净收益”与非短途载客司机的“单位时间净收益”相等。因而，插队系数受短途客里程数 x’、当前排队出租车数 L、当前等待上车的乘客数$p_0$、司机前往排队的时间决定。在最后，我们带入了一组常规的数据，对插队系数 a 与短途客里程数 x’的关系进行了拟合，拟合曲线表现良好，表现了模型的合理性。

**关键词：**算法模拟 随机概率 蒙特卡罗 停车模型

# 一、 问题重述

## 1.1 问题背景

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。国内多数机场都是将送客（出发）与接客（到达）通道分开的。送客到机场的出租车司机都将会面临两个选择：

(A) 前往到达区排队等待载客返回市区。出租车必须到指定的“蓄车池”排队等候，依“先来后到”排队进场载客，等待时间长短取决于排队出租车和乘客的数量多少，需要付出一定的时间成本。

(B) 直接放空返回市区拉客。出租车司机会付出空载费用和可能损失潜在的载客收益。

在某时间段抵达的航班数量和“蓄车池”里已有的车辆数是司机可观测到的确定信息。通常司机的决策与其个人的经验判断有关，比如在某个季节与某时间段抵达航班的多少和可能乘客数量的多寡等。如果乘客在下飞机后想“打车”，就要到指定的“乘车区”排队，按先后顺序乘车。机场出租车管理人员负责“分批定量”放行出租车进入“乘车区”，同时安排一定数量的乘客上车。在实际中，还有很多影响出租车司机决策的确定和不确定因素，其关联关系各异，影响效果也不尽相同。

## 1.2 问题提出

请你们团队结合实际情况，建立数学模型研究下列问题：

(1) 分析研究与出租车司机决策相关因素的影响机理，综合考虑机场乘客数量的变化规律和出租车司机的收益，建立出租车司机选择决策模型，并给出司机的选择策略。

(2) 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，并分析模型的合理性和对相关因素的依赖性。

(3) 在某些时候，经常会出现出租车排队载客和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) 机场的出租车载客收益与载客的行驶里程有关，乘客的目的地有远有近，出租车司机不能选择乘客和拒载，但允许出租车多次往返载客。管理部门拟对某些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，试给出一个可行的“优先”安排方案。

## 二、 问题分析

### 2.1 问题(1)的分析

题目要求分析研究与出租车司机决策相关因素的影响机理，综合考虑机场乘客数量的变化规律和出租车司机的收益。我们考虑影响司机决策的因素有：在某时间段抵达的航班的时刻表，“蓄车池”里已有的车辆数 C，司机到达机场时已有的正在排队乘客数量 p0。以这 3 个影响因素作为司机决策模型的输入，从而，建立数学模型根据这些输入的影响机理计算司机采取不同决策下的经济效益，给出经济效益大者对应的策略方案。

### 2.2 问题(2)的分析

按照题目要求，我们所选取的研究对象的数据应该是易于发掘的，应该选取一个较大型的机场来进行研究。而问题(2)会运用到问题(1)中所建立的模型，也就是将模型运用到实际问题中，那么模型中的参数如何合理的给出是本题的关键点。关于模型的依赖性问题，应该是将模型中的相关参数进行变动，观测这种变化对于模型给出的结果的影响是否显著。

### 2.3 问题(3)的分析

根据该问的要求，我们分析知与第一二问关联微乎，需重新建立相对应的模型。以什么为标准去衡量不同“上车点”的方案的优劣就显得至关重要。考虑到出租车的目的是为了尽快的将乘客带出机场以免造成机场人口滞留，我们以出租车内单位时间驶出率为优化目标函数。同时，应确定一个车辆驶入停车位的方式及规则，在能够保证乘客上车时的安全的前提下，使得目标函数最优。又考虑到实际情况中每个乘客的上车时间的浮动是巨大的，不能忽略，因此模型中应该体现出乘客上车时间的随机性。

### 2.4 问题(4)的分析

对于该问题，首先要解决如何定义短途出租车的问题，然后需要设置分配的优先级的方式。由于短途的司机行驶的距离也不尽相同，用一刀切的方式给出一个平均的优先级显然不够合理，因此最终解决问题所得的优先级应该是与距离相关的，需要考虑如何将这个距离体现在优先级的计算中。另一个问题是优先级因该如何刻画，通过哪一种指标来判定有了优先级之后的短途车与长途车的收益是一致的，这些都是应该在模型中解决的问题。

## 三、 模型假设

### 3.1 问题（1）的假设

A1 假设淡季（旺季）的白天（夜间）时间段每趟航班人数一致，且同一趟航班选择打车的乘客在下飞机后经经过相同时间(从航站楼移动至上车地点、取行李等耗时)同时进入排队打车系统

A2 只要上车处旅客量不为零，单位时间发送旅客量恒定，且每辆车的旅客数一致，从而单位时间发送出租车的数量恒定。

A3 假设燃油费与公里数呈线性关系

A4 在距机场相等距离的地点，接到乘客的概率相等。

A5 司机将乘客送到市区目的地之后，立刻往机场方向返回。

### 3.2 问题（2）的假设

A6 假设打车乘客的目的地选择概率与目的地的人口密度成正相关

### 3.3 问题（3）的假设

A7 综合考虑各乘客的个体差异后假设乘客从起点到上车点的步速都是一致的。

A8 鉴于所研究的是乘客上车点分布的最优解，不考虑待乘车乘客和待客出租车的数量

多少的影响，故假设乘客数与出租车数均足够大，即不存在空闲时间。

A9 假设车位大小一致

A10 假设车的启动时间和停车时间分布一致，每辆车在启动时间后以相同的速度匀速行驶

## 四、 模型建立与求解

### 4.1 模型变量解释

| 序号 | 符号 | 符号解释 |
|---|---|---|
| 1 | T | 泊车区第 k 轮车辆全部进入驶离区起到第 k+1 轮车辆全部进入驶离区的时长 |
| 2 | t(j) | 第 j 排的车从所在 T 的起始时刻开始至其进入驶离区的时间 |
| 3 | $t_k^j$ | 表示第 j 排的车处于第 k 个阶段的时间 k=1,2,3 |
| 4 | E | 单位时间驶出率 |
| 5 | $p_0$ | 当前到达上车区但未上车的人数 |
| 6 | L | 当前排队的出租车数目 |
| 7 | list[1...n] | 当前可查询的时刻表 |
| 8 | f(x) | 乘客乘出租车前往的目的地的里程概率密度函数 |
| 9 | h(x) | 由出租车里程收费标准确定的里程数与收益的函数 |
| 10 | g(x) | 油费与公里数的函数 |
| 11 | v | 出租车平均行驶速度 |

| | | |
|---|---|---|
| 12 | xmax | 市区范围内据机场最大距离 |
| 13 | EX | 里程数期望 |

**4.2 问题（1）**

我们建立一个决策模型：

决策模型的输入量：当前排队的出租车数目 L、当前到达上车区但未上车的人数 $p_0$、当前可查询的时刻表 list[1...n]。

已知数据（依赖于数据统计）：

1. 乘客乘出租车前往的目的地的里程概率密度函数 f(x)
2. 由出租车里程收费标准确定的里程数与收益的函数 h(x)
3. 油费与公里数的函数 g(x)
4. 出租车平均行驶速度 v
5. 市区范围内据机场最大距离 xmax

决策思路：

分别计算两种选择下，从到达机场的这个时刻起（即决策时刻），到下次回到机场的这一周期性时间范围内，单位时间内的净收益，选择单位时间内净收益高的方案。

下面分别叙述 A、B 两种方案的单位时间计算方法。

A 方案：单位时间净收益 = （载客离开机场的收益 + 载新一轮客前往机场的收益 - 来回燃油费）/（排队时间 + 往返时间）

1. 载客离开机场的收益：由于里程数与收益的函数 h(x)已知，只需确定离开机场的里程数即可。离开机场的里程数可用里程的数学期望表示，即 EX =$\int_0^{\mathrm{xmax}} f(t)\,dt$ .故离开机场的收益为 h(EX)
2. 载新一轮客前往机场的收益：根据假设 A5，载完旅客到达目的地后，即返回机场。由于离开机场的距离为 EX，则在此条件下，条件概率分布函数为 F(x|x < EX) = P(X < x)/P(X < EX) = $\frac{\int_0^x f(t)\,dt}{\int_0^{EX} f(t)\,dt}$, 故概率密度为$\frac{f(x)}{\int_0^{EX} f(t)\,dt}$，故期望为$\frac{\int_0^{EX} tf(t)\,dt}{\int_0^{EX} f(t)\,dt}$,

记 Φ(x) $=\frac{\int_0^x tf(t)\,dt}{\int_0^x f(t)\,dt}$ ，则载新一轮客前往机场的收益为 h(Φ(EX))

3.来回燃油费：由于最远距离为 EX，则来回燃油费为 2 * g(EX)

4.往返时间：2 * EX / v

5.排队时间：这是本决策模型中的重点部分，将在下面进行详述。

排队模型的建立：

把出租车的队列分为两种状态：空闲状态、工作状态。

在工作状态下，上车点的客人源源不断地被出租车运走，出租车队列不断向前移动。当上车点的乘客数降为零时，队列进入空闲状态。空闲状态下，根据假设 A2，单位工作时间离开的出租车数量是一定的，设单位工作时间离开的出租车数量为 car_per_min，每辆出租车的载客量为 p_per_car。由于排在前面的出租车数目 L 为已知量，若要使排在前方的出租车全部离开，需累积一定的工作状态总时间，即 T = L / car_per_min。对此我们的思路是：模拟整个过程，直到累积工作状态时间达到 T，此时的时间即为总的排队时间。

定义算法如下：

get_wait_time(L, $p_0$, list):

input:

当前排队的出租车数目 L、当前到达上车区但未上车的人数$p_0$、当前可查询的时刻表 list[1…n]

output:等待时间

start：

T = L / car_per_min;  // 需累积工作时间

k = car_per_min * p_per_car; // 旅客人数下降的直线斜率（人/min）

sum_wolk_time = 0;  // 当前累积工作时间

p = $p_0$;  //当前上车点乘客数

t = 0; // 当前时刻

cnt = 0;  // 航班表的遍历下标

while (sum_work_time < T):  // 循环直到 工作时间累计足

    t_zero = t + p / k;  // 若无航班到达，等车乘客数将在 t_zero 时刻降为

```
        0
        if (t_zero > list[cnt].time) { // 等车乘客数还没到达0, 又来了人
                    p = p - (list[cnt].time - t) * k; // 更新等车乘客数
                    temp_sum_work_time = sum_work_time + list[cnt].time  -  t
                                // 预计累计工作时间
        else     //等车乘客数到达0之前，无航班旅客到达
                p = 0;    // 等车乘客数减为零
                temp_sum_work_time = sum_work_time + t_zero  -  t;
                                // 预计累计工作时间

        end if;
        if (temp_sum_work_time >= T) // 工作这么久后，如果累计工作时间已经满
        足
                        return t + (T - sum_work_time);  // 输出等待时间
        end if
         sum_work_time = temp_sum_work_time;  // 更新累计工作时间

                    t = list[cnt].time // 更新当前时间
                    p += list[cnt].time * rate;  // rate为就坐率为定值（百分
        之多少的人坐出租车）
        cnt++;
end while
```

综上，A方案的单位时间收益为

AVG_A = (h(EX) + h(fai(EX)) - 2 * g(x))/(get_wait_time($p_0$,L,list) + 2 * EX / v)　　（1-1）

B方案：单位时间净收益 = （载客回到机场的收益 - 来回燃油费）/ 往返时间

载客回到机场的收益：由于B方案下，并没有载到旅客便离开机场，故载客回机场的里程期望为EX,故载客回到机场的收益为h(EX)

1.来回燃油费：同 A 方案，即 2 * g(EX)

2.往返时间：同 A 方案，即 2 * EX / v

综上，B 方案的单位时间收益为：

AVG_B = (h(EX) - 2*g(EX)) / (2 * EX / v) （1-2）

至此，我们便可给出决策：

a) 若 min{AVG_B，AVG_A} = AVG_A，则选择 A 方案；

b) 若 min{AVG_B，AVG_A} = AVG_B，则选择 B 方案。

### 4.3 问题（2）

由于上海市浦东国际机场的信息获取较易，我们以其为研究对象，运用模型一，我们输入数据，从而得到结果。

数据准备：

由问题（2）的分析知我们需要将上海市浦东国际机场的全天航班时刻表作为决策模型的参数提前输入好，于是在网站上找到浦东机场在 9/14 的所有航班信息，使用 python 语言编写自动抓取网络信息的脚本程序。再对抓取到的信息使用 Java 语言正则表达式提取出需要的信息，获得指定格式的 9/14 全天航班时刻表。这里每个单元格中用空格划分，空格前表示航班的到达时刻，空格后为该时刻同时到达的航班数。

| 浦东机场2019/9/14各时刻到达航班数 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0:5 2 | 8:30 1 | 10:40 3 | 12:50 2 | 14:45 4 | 16:50 2 | 18:35 2 | 20:20 9 | 22:5 4 |
| 0:10 7 | 8:35 1 | 10:45 2 | 12:55 5 | 14:50 5 | 16:55 6 | 18:40 9 | 20:25 4 | 22:10 2 |
| 0:15 14 | 8:40 2 | 10:50 3 | 13:0 7 | 14:55 2 | 17:0 17 | 18:45 4 | 20:30 4 | 22:15 6 |
| 0:20 8 | 8:45 1 | 10:55 6 | 13:5 3 | 15:0 6 | 17:5 4 | 18:50 14 | 20:35 1 | 22:20 16 |
| 0:25 6 | 8:50 5 | 11:0 6 | 13:10 3 | 15:10 1 | 17:10 1 | 18:55 17 | 20:40 2 | 22:25 2 |
| 0:30 14 | 8:55 2 | 11:5 2 | 13:15 2 | 15:15 2 | 17:15 5 | 19:0 22 | 20:45 3 | 22:30 7 |
| 0:35 8 | 9:15 2 | 11:20 3 | 13:25 1 | 15:20 4 | 17:20 8 | 19:5 5 | 20:50 1 | 22:35 6 |
| 0:40 3 | 9:20 2 | 11:25 5 | 13:35 5 | 15:25 2 | 17:25 4 | 19:10 3 | 20:55 7 | 22:40 10 |
| 0:45 4 | 9:25 4 | 11:35 3 | 13:40 9 | 15:35 8 | 17:35 4 | 19:15 4 | 21:0 11 | 22:50 9 |
| 0:50 7 | 9:30 2 | 11:40 3 | 13:45 5 | 15:40 2 | 17:40 10 | 19:20 3 | 21:5 5 | 22:55 5 |
| 0:55 4 | 9:35 11 | 11:50 1 | 13:50 6 | 15:45 1 | 17:45 3 | 19:30 6 | 21:10 11 | 23:0 8 |
| 1:0 7 | 9:40 4 | 11:55 6 | 13:55 12 | 15:50 2 | 17:50 2 | 19:35 5 | 21:15 4 | 23:5 5 |
| 1:5 2 | 9:45 7 | 12:0 2 | 14:0 2 | 15:55 6 | 17:55 14 | 19:40 3 | 21:20 11 | 23:10 8 |
| 1:10 2 | 9:50 5 | 12:5 4 | 14:5 1 | 16:0 3 | 18:0 8 | 19:45 2 | 21:25 2 | 23:15 7 |
| 1:25 3 | 9:55 7 | 12:10 4 | 14:10 2 | 16:5 12 | 18:5 7 | 19:50 11 | 21:30 6 | 23:20 10 |
| 1:30 3 | 10:0 2 | 12:20 2 | 14:15 5 | 16:10 10 | 18:10 8 | 19:55 3 | 21:35 2 | 23:25 9 |
| 1:45 2 | 10:5 4 | 12:25 1 | 14:20 2 | 16:15 2 | 18:15 8 | 20:0 14 | 21:40 7 | 23:35 7 |
| 1:55 2 | 10:15 6 | 12:35 5 | 14:25 4 | 16:20 1 | 18:20 3 | 20:5 4 | 21:50 3 | 23:40 4 |
| 7:45 1 | 10:20 4 | 12:40 1 | 14:30 2 | 16:30 1 | 18:25 4 | 20:10 1 | 21:55 2 | 23:45 8 |
| 8:5 3 | 10:25 4 | 12:45 5 | 14:40 5 | 16:40 5 | 18:30 2 | 20:15 11 | 22:0 4 | 23:50 5 |

图 4.3.1

数据处理：

首先找到上海市浦东机场周边人口分布的密度图像，以机场为起点等角度向周围发出射线，以机场为圆心，作等距离的同心圆，取圆和射线的交点为样本点。按照密度图像上的人口密度等级的标注，统计同一圈样本点上的人口密度等级之和，计算该圈样本点的平均人口密度等级。取横轴为各圈在地图上对应比例尺下的公里数，纵轴为圈上样本点的平均人口密度等级，由之前的计算结果画出的一系列点后用多项式函数拟合该散点图，得到人口密度等级随到机场距离变化的函数，根据假设即可得到打车乘客到距机场某个距离的目的地的可能性的变化函数。将该函数进行归一化处理，便可打车公里数的概率密度函数。

如图 4.3.2 所示，我们取 10 条射线，以 5 公里为步长画同心圆，统计到距浦东机场 75 公里的范围，由于 75 公里处的范围属于上海市的边界，所以这样划分下的统计是合理有效的。

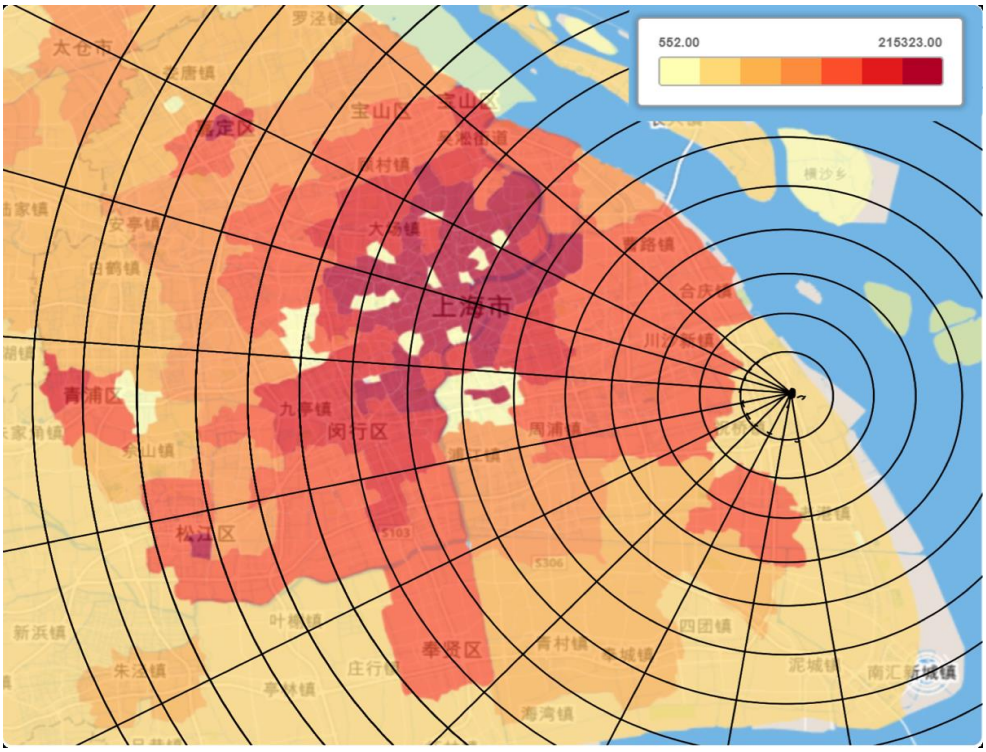
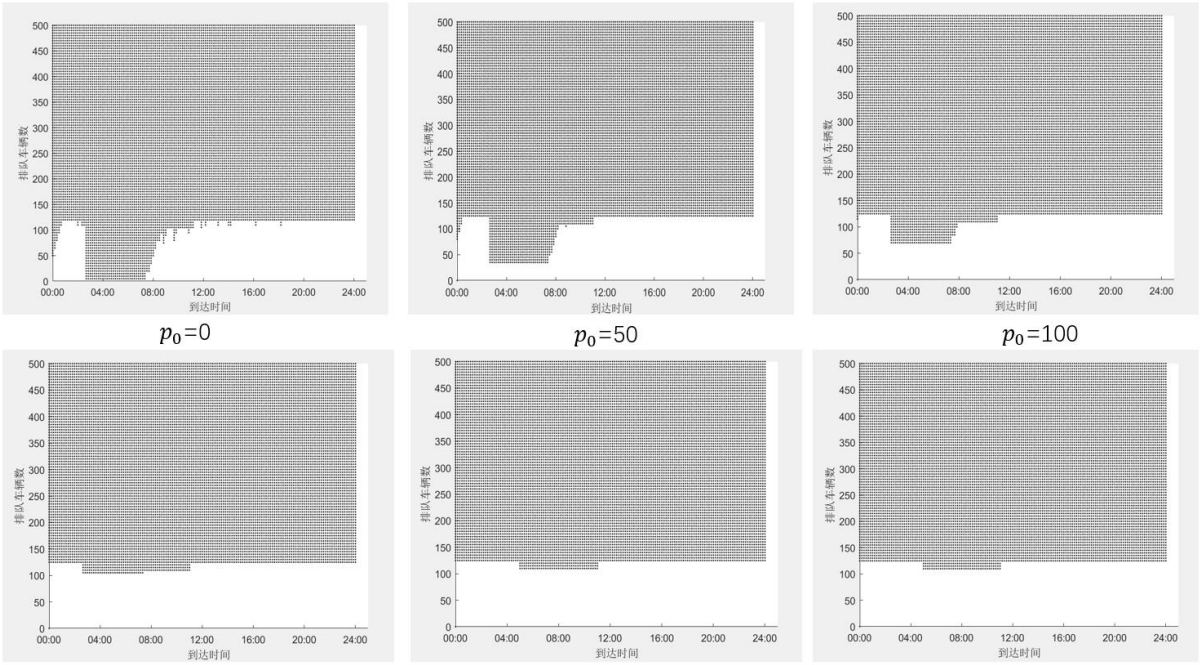



图 4.3.2

通过查找资料得到，出租车的邮费 v = k*x ，k = 7 元/s，出租车的平均速度 v = 50km/h = $\frac{5}{6}$km/min，出租车的有效座位数 s = 2.5 个/车（有效座位数即平均每个出租车上的乘客数），在白天时间，约 15%的乘客会选择出租车，夜间比例接近 45%。

代入（1）问题中式（1-1）和（1-2），可以得到图 4.3.3：




$p_0$=0　　$p_0$=50　　$p_0$=100

$p_0$=150　　$p_0$=200　　$p_0$=250

图 4.3.3

图中各点代表对应横纵坐标下的决策结果，如果该点显示为黑点，则表示该情况下的决策结果为题目中的 B 方案，否则就表示决策结果为 A 方案。故该图像可大致分为上下两个区域，上区域的决策为 B，下区域决策为 A。

通过对改变$p_0$产生的多张图横向对比，这里每张图$p_0$从 0 开始以 50 的步长增加。第一张图$p_0$为 0，则在大约 3：00 至 7：00 时，即使出租的车排队数量为 0，也建议 B 方案，因为此时$p_0$为 0，所以即使该决策司机作为第一个等在乘客出口的司机，出口处也没乘客，且返回查看之前准备好的航班时刻表，发现这段时间没有任何航班抵达，所以要等到下一班航班到来会消耗大量的等待时间。随着$p_0$的增加，则 3:00 到 7：00 会有一部分排队车辆对应到 A 决策，因为此时的$p_0$会使得做 A 决策的司机载到客。此外整体来看，两区域的在 3:00 到 11：00 之间的分界线也会随着$p_0$的增加而上移，结合时刻表发现这是因为$p_0$的增加会使得在降落航班稀疏的时间段内司机的等待却不前移的时间减少，从而可以容忍更多的排队车辆以执行 A 策略。而对于其他密集降落的时间段，有大量的乘客不断到来，司机的等待却不前移的时间几乎为 0，所以即使$p_0$的增加与否不会影响这段时期的决策。然而当$p_0$增加到一定值时，此处为 200，则该分界线不会再上移，而是停在极限情况不再变化，这是因为此时的$p_0$已经满足能够始终为出租车司机提供源源不断的客流，所以达到满载的极限情况，此时$p_0$再增加也不会影响决策。

分别观测各张图，在相同到达时间的情况下，排队车辆数的增加会导致司机的排队时间增加，减少其选择 A 决策带来的收益，从而更倾向 B 决策，当排队车辆数稳定时，随着到达时间的变化，只有在当航空公司的后半夜停运时，才使得模型倾向于建议 B 决策，其余时间对于决策选择的影响基本不大。

大约 7 点之后，开始有航班陆续抵达，则在更多的排队车辆数情况下会开始建议使用 A 决策，表示有更多的乘客抵达，可接受适当延长的出租车排队时间。大约 2 点开始，到达航班减少至 0，则在相同的排队车辆数情况下会由建议使用 A 决策转为使用 B 决策，因为当最后一架降落航班的乘客到达出租车站点时，在这批乘客乘坐出租车驶离之后，没有新航班降落所带来的客流量，所以建议决策 B。综合以上分析之后发现该模型符合人的经验，切合实际情况，有较高的合理性。

### 4.4 问题（3）

根据国家道路标准规定的停车位大小，假定每个车位的长度均为 5 米，宽度为 3.5

米，且在计算人与车间的距离时，将车抽象为停车位的中心点。

首先我们考虑停车位的分布，参见图 4.4.1。将距离乘客出口近的车道记位第一车道，另一条为第二车道，如果我们选择 2j-1 号车位作为乘客的上车点，那么对于该车道中 2j-1 之前的车位，若我们不选择其中某个作为上车点，那么相当于实际生活中乘客选择距离远的车乘坐而不去选择距离近的车乘坐，这与常识不相符合，于是若我们选取 2i-1 处为上车点，则 1,3,…,2j-3 均为上车点。若在优化方案中选择了 m 个停车位，为避免两条车道所选取的车位个数不同而带来的安全隐患，即乘客在车流中穿行的风险，故而我们要求停车位为偶数个，即 m=2i，且这些停车位都是并排等长地安排在两条车道中，如图 4.4.1 所示。


图 4.4.1

接下来我们考虑车辆的驶入驶出方案，若待所有车停稳后才允许乘客上车，那么就必然会浪费位于前面停车位的车辆的时间，所以若允许一辆车满足安全状态即可允许上车，那么就会一定程度上的减少各方的等待时间，而一辆车满足安全的状态可认为是紧跟其后的车辆是停稳的，出于这种考虑，我们设置车辆进入和驶出泊车区的方式为：当上一批的 2i 个泊车位上的车均进入驶离区时，即刻开始允许下一批车进入泊车区，此时泊车区的情况如图 4.4.2；在进入过程中每次只放入并排的两辆车，只有当目前放入的两辆车停稳后，才允许后一排的两辆车进入。

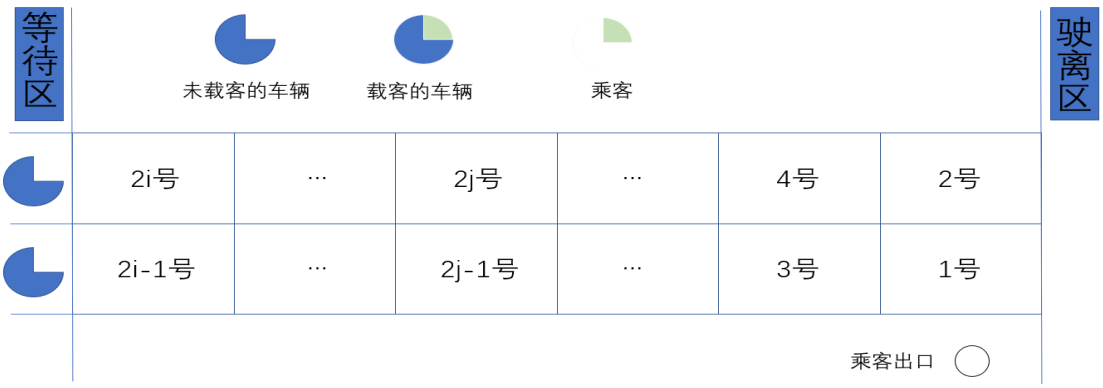



图 4.4.2

乘客进入泊车区的方式为：如果乘客想进入第一排车位中的车时，待 3，4 号位车停下后，此时才开始从出口中放行 2 名乘客进入第一排车位中的车，如图 4.4.3；每次只允许两个人进入，当 2j+1，2j+2 号车位的车停稳后，才允许 2j-1，2j 号车位的乘客进入泊车区，且认为二人事先已无冲突的选好了要上的车，直接走向相应车位，如图 4.4.4。

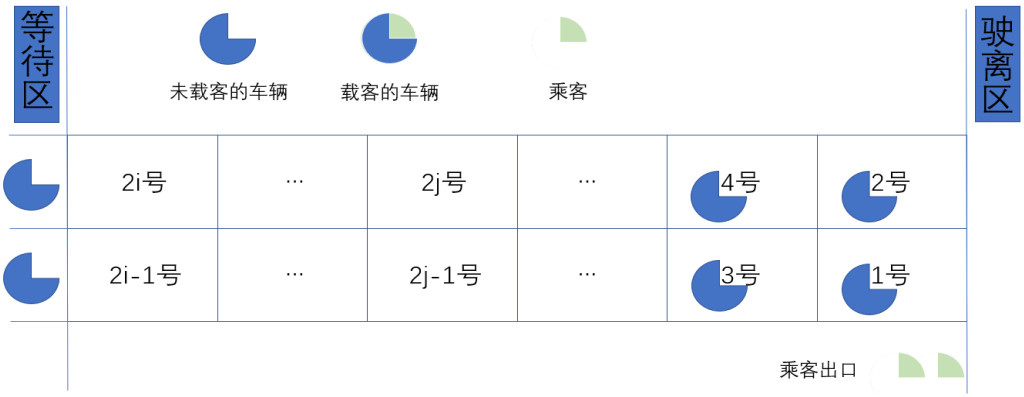



图 4.4.3

（ 此时 3，4 号停车位停稳，两个乘客进入泊车区，准备上 1，2 号车）

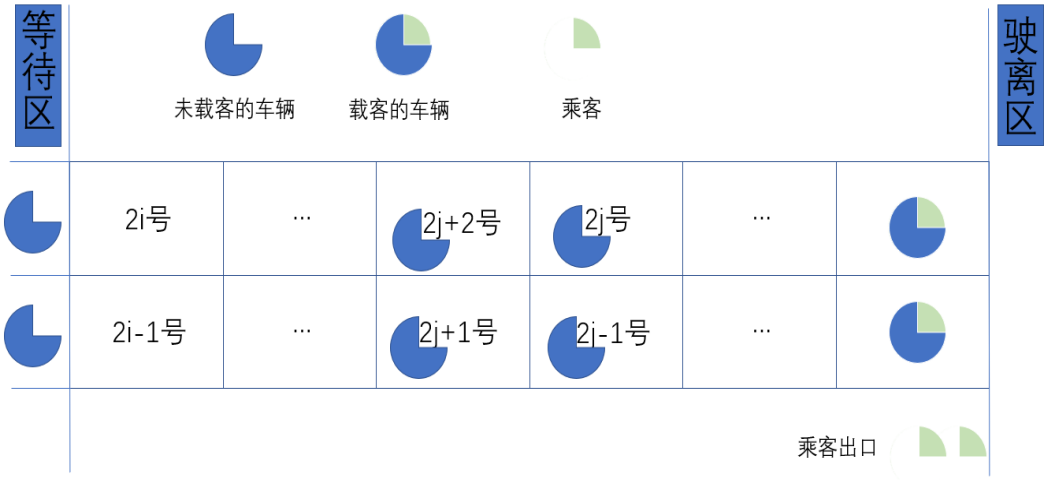



图 4.4.4

（此时 2j+2,2j+1 号停车位已停稳，两个乘客进入泊车区，准备上 2j,2j-1 号车,1，2 号停车位的车已完成上车，准备驶离）

为了后续叙述简单，将每辆车分为三个阶段，第一阶段为车辆在被允许进入泊车区至其达到对应车位且停稳，第二阶段为乘客上车，第三阶段为前一辆车到达驶离区到该车辆到达驶离区。

计算 T 的算法为（T 为泊车区第 k 轮车辆全部进入驶离区起到第 k+1 轮车辆全部进

入驶离区的时长）：

对于设置了 2i 个车位的方案，T＝t(i)。由于上文事先设置好车位的布局方式，对于每次放行进入泊车区的并排的两辆车，根据假设知这两辆车启动时间和行驶速度相同，则他们是并排同时到达对应的车位，所以两条车道的最后一排车位也是同时停好两辆车；而同时两条车道上的车辆的驶出是相互独立的，即某排的车辆是否可以进入阶段三，只取决于其所在车道位于该车之前的车辆是否完成阶段三，而与并排车辆的状态无关，又因为单车道的最后一排车位上的车辆完成阶段三的先后顺序决定了 T，故而我们只需给出一条车道的算法，然后取两条车道的时长的最大值即可。

最后一辆车是否进入阶段三不仅取决于该车的阶段二是否完成，还取决于该车之前的各车辆的状态，故而我们可以由前至后的依次计算出各排车的从周期开始到完成阶段三的时长，计算至最后一辆车即可，这个工作可由计算机进行循环来完成。在进行推导过程中我们注意到 i=1 时，i=2 时及 i>2 时的表达式是不同的，所以应分情况讨论：

a) i=1

此时 T 就是第一辆车三个状态的时间之和，即 T=t(1)= $t_1^1 + t_2^1 + t_3^1$，如图 4.4.5 ：

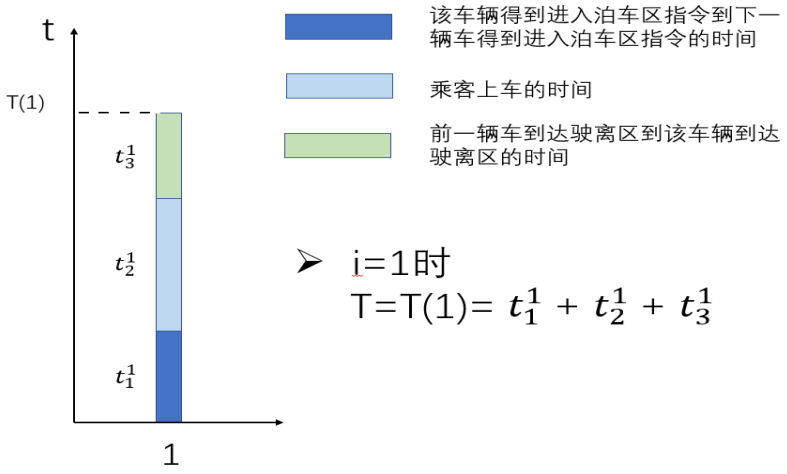



图 4.4.5

b) i=2

此时第一辆车由于要等待第二辆车停稳才可以进入第二阶段，图 4.4.6 为时间示意图，由图 4.4.6 我们可知：

t(1)= $t_1^1 + t_2^1 + t_3^1 + t_1^2$

t(2)= $t_1^1 + t_1^2 + t_3^2 + \max(t_2^1 + t_3^1, t_2^2)$

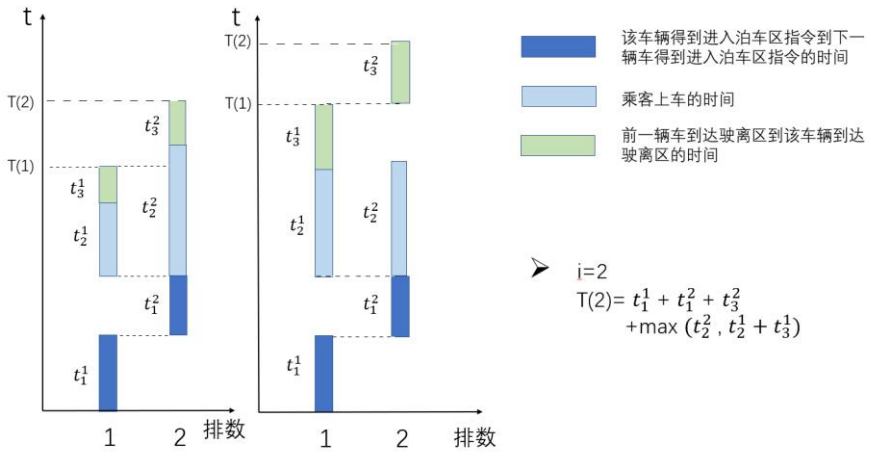



图 4.4.6

c) i>2

相较于上一种情况，第 j 辆车多了一个等待后一辆车停稳的时间，图 4.4.7 为此情形的时间示意图，由图可得：

t(1)= $t_1^1 + t_1^2 + t_2^1 + t_3^1$

t(2)= $t_1^1 + t_1^2 + t_3^2 + \max(t_2^2 + t_1^3, t_2^1 + t_3^1)$

t(j)= $\sum_{n=1}^{j} t_1^n + t_3^j + \max\{T(j-1) - \sum_{n=1}^{j} t_1^n \ , t_2^j + t_1^{j+1}\}$

T=t(i)= $\sum_{j=1}^{i} t_1^j + t_3^i + \max\{T(i-1) - \sum_{j=1}^{i} t_1^j \ , t_2^i\}$

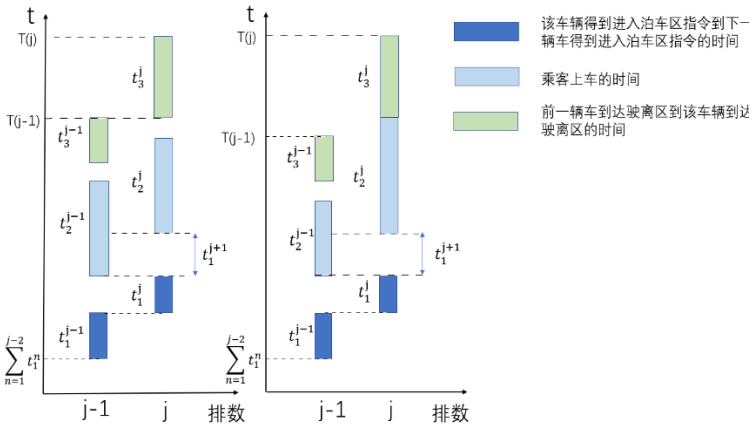



图 4.4.7

对于乘客上车的随机时间，我们由参考文献[6]可得到下表中的频率分布图，应用蒙特卡罗方法将数据进行处理，得到下表 4.4.8：

| 时间（s） | 频率 | 累积频率 | 时间（s） | 频率 | 累积频率 |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| 3-6 | 0.031111 | 0.031111 | 51-54 | 0 | 0.804444 |
| 6-9 | 0.084444 | 0.115556 | 54-57 | 0.031111 | 0.835555 |
| 9-12 | 0.08 | 0.195556 | 57-60 | 0.017778 | 0.853333 |
| 12-15 | 0.088889 | 0.284444 | 60-63 | 0.031111 | 0.884444 |
| 15-18 | 0.04 | 0.324444 | 63-66 | 0.008889 | 0.893333 |
| 18-21 | 0.044444 | 0.368889 | 66-69 | 0.013333 | 0.906666 |
| 21-24 | 0.071111 | 0.44 | 69-72 | 0.035556 | 0.942222 |
| 24-27 | 0.044444 | 0.484444 | 72-75 | 0.013333 | 0.955555 |
| 27-30 | 0.048889 | 0.533333 | 75-78 | 0.013333 | 0.968889 |
| 30-33 | 0.04 | 0.573333 | 78-81 | 0.008889 | 0.977778 |
| 33-36 | 0.044444 | 0.617778 | 81-84 | 0.004444 | 0.982222 |
| 36-39 | 0.04 | 0.657778 | 84-87 | 0.008889 | 0.991111 |
| 39-42 | 0.022222 | 0.68 | 87-90 | 0.004444 | 0.995555 |
| 42-45 | 0.048889 | 0.728889 | 90-93 | 0 | 0.995555 |
| 45-48 | 0.031111 | 0.76 | 93-96 | 0 | 0.995555 |
| 48-51 | 0.044444 | 0.804444 | 96-99 | 0.004444 | 1 |

表 4.4.8

我们将 3 对应 0，6 对应 0.031111，9 对应 0.115556，12 对应 0.195556，以此类推，可以得到一个单增的折线图，计算机随机生成一个介于 0、1 之间的数 rand，通过判断 rand 位于哪个小区间段内，通过线性插值的方式可得到一个随机的乘客上车时间 M（rand）。线性插值的公式为：

$$M(rand) = y_1 + \frac{x-x_1}{x_2-x_1}(y_2 - y_1),\ rand \in (x_1, x_2),$$

其中$x_1$在表 4.4.8 中对应$y_1$，$x_2$在表 4.4.8 中对应$y_2$。

为了得到每个阶段的时间，我们需要通过下面的值来算出：

查得：车辆的启动时间时间满足均值为 5s 的均匀分布模型，人的平均步速为 1.5m/s，停车场的行驶速度为不超过 1.39m/s，于是求得车辆每走过一个车位的时间为 3.6s，则有

计算第一车道时，$t_1^j = 5 + \frac{\sqrt{(j-1)^2 25 + 3.5^2}}{1.5}$，$t_2^j = M(rand)$，$t_3^j = 5 + \frac{\sqrt{j^2 25 + 3.5^2}}{1.5}$

计算第二车道时，$t_1^j = 5 + \frac{\sqrt{(j-1)^2 25 + 4*3.5^2}}{1.5}$，$t_2^j = M(rand)$，$t_3^j = 5 + \frac{\sqrt{j^2 25 + 4*3.5^2}}{1.5}$

至此我们可以建立模型如下：

优化目标：对于每一个 i，对两条车道分别求解，可得 T1，T2，两者中最大值为 T，在 i 的变化范围内找到最大的 E=8000/T 对应的 i。

T 的表达式为：

i=1　T=t(1)= $t_1^1 + t_2^1 + t_3^1$

i=2　T=t(2)= $t_1^1 + t_1^2 + t_3^2 + \max(t_2^1 + t_3^1, t_2^2)$

i>2　t(1)= $t_1^1 + t_1^2 + t_2^1 + t_3^1$

t(2)= $t_1^1 + t_1^2 + t_3^2 + \max(t_2^2 + t_1^3, t_2^1 + t_3^1)$

t(j)= $\sum_{n=1}^{j} t_1^n + t_3^j + \max\{T(j-1) - \sum_{n=1}^{j} t_1^n \ , t_2^j + t_1^{j+1}\}$

T=t(i)= $\sum_{j=1}^{i} t_1^j + t_3^i + \max\{T(i-1) - \sum_{j=1}^{i} t_1^j \ , t_2^i\}$

其中 0=<i<=14

模型的求解由计算机编程实现，代码见附录。

为更恰当的讨论 T 的变化情况，现定义效率提高比例 $e_i$=$\frac{T_{i-1}-T_i}{T_{i-1}}$.

我们输入数据（参考文献[6]），为了消除由于乘客上车时长随机分布导致的 T 计算结果不稳定性的影响，我们将模拟每次有 8000 辆车待通过泊车区载客的情况，模拟 1000 次，得到的平均 T 就具有普适意义，得到的结果如下表 4.4.9 及图 4.4.10、图 4.4.11 所示：

| i | E | $e_i$ | i | E | $e_i$ |
|---|---|---|---|---|---|
| 1 | 3.84E-02 | | 8 | 8.51E-02 | 2.74E-02 |
| 2 | 5.52E-02 | 4.39E-01 | 9 | 8.68E-02 | 1.95E-02 |
| 3 | 6.56E-02 | 1.88E-01 | 10 | 8.84E-02 | 1.86E-02 |
| 4 | 7.22E-02 | 9.99E-02 | 11 | 8.96E-02 | 1.37E-02 |
| 5 | 7.68E-02 | 6.42E-02 | 12 | 9.06E-02 | 1.14E-02 |
| 6 | 8.02E-02 | 4.43E-02 | 13 | 9.16E-02 | 1.14E-02 |
| 7 | 8.28E-02 | 3.30E-02 | 14 | 9.25E-02 | 9.54E-03 |

表 4.4.9

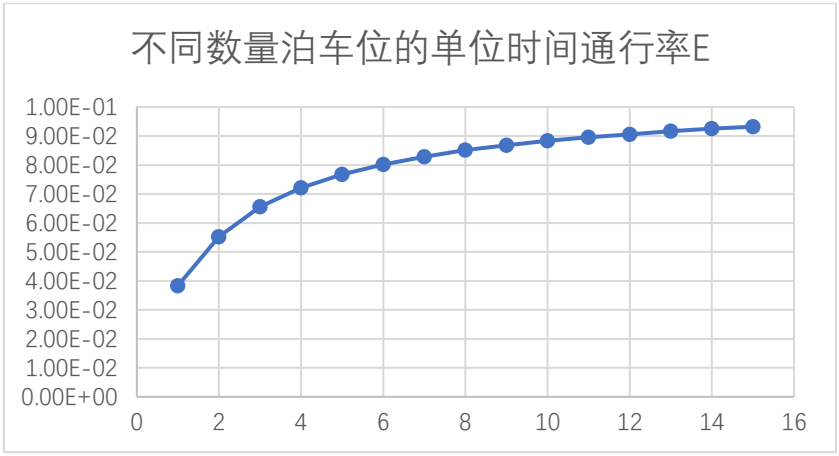



图 4.4.10

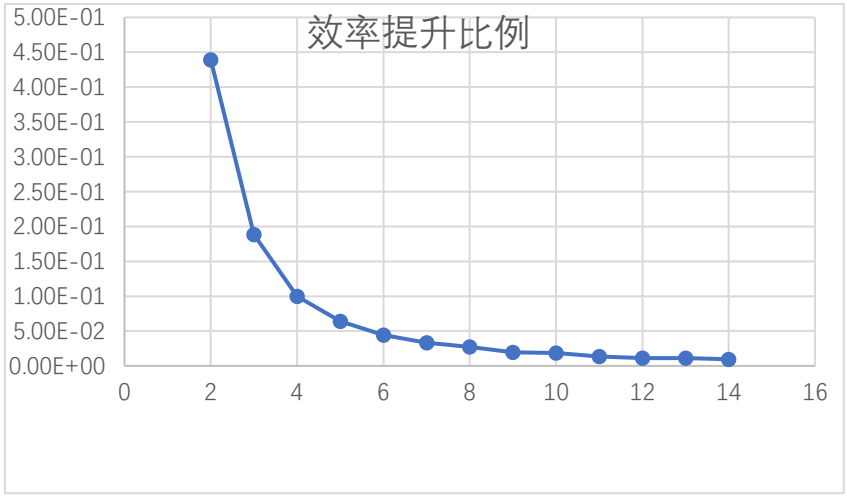



图 4.4.11

由上述结果可知，随着车位数的增加，E 在增大，但我们注意到，E 增大的幅度也在逐步减小，即增加车位数所带来效率提高比例慢慢变小；同时，管理部门为其付出的管理成本将会增加，乘客的体验也会变差。观测图 4.4.1，我们发现 i＝5 时，$e_6$比$e_5$小，且 i=6 之后的 e 在较低数值上逐渐缓慢下降，体现出增加车位引起的通行固定车辆数所需时间的降低幅度在减缓，考虑到现实情况下，增加车位带来额外的管理难度以及乘客到达停车位的距离变长等因素，选择 5 个泊车位可以使得单位时间驶出率最优。

为进一步讨论 i=5 时每个车位的利用效率情况，现定义第 j 个车位的负荷率 $=\frac{t(j-1)-\sum_{n=1}^{j-2} t_1^n}{T}$，负荷率见下图 4.4.12

| 车位号 | 负荷率 | 车位号 | 负荷率 |
|---|---|---|---|
| 1 | 0.339756798899513 | 6 | 0.38415133505335886 |
| 2 | 0.35775026322212916 | 7 | 0.4005848271720382 |

| | | | |
|---|---|---|---|
| 3 | 0.35274355958355025 | 8 | 0.4064023686237848 |
| 4 | 0.36567808111901157 | 9 | 0.42578689025296307 |
| 5 | 0.37607507823060493 | 10 | 0.43026486221599974 |

图 4.4.12

由于乘客上车时间的不确定性与差异性，每个车位的负荷率较低，若能减少上车时间较长的乘客的出现概率，则可以使得车位的负荷率提升，可以通过在实际生活中安排工作人员帮助乘客在上车过程中搬运大件行李，从而达到目的。

**4.5 问题（4）**

仍然利用问题（1）中模型，由于该优先级是由机场给出的，所以我们认为模型中的$P_0$是已知量，当前排队的出租车数以及时刻表都是已知的，

我们首先对短途车进行定义，即设置一个短途地判断标准$x_0$，当车辆接到客送达目的地的公里数 x<$x_0$时，即认为该车为短途车。对于一个短途出租车，给其优先级的目的是缩短其返程回来再次进入蓄车池排队接客的等待时间，让其能尽快地接到下一个客人，用缩短的等待的时间以及再次接到客人的收益，去平衡掉其第一次由于接到短途较之接到长途的司机的损失，从而使得接到短途与接到长途的收益是平衡的。而缩短等待时间可以等效考虑成在其再次接客时由于分配的优先级引起的前面的车辆数较真实值减少所节省的排队等待时间，故而这个优先级可由再次接客时将前面实际排队车辆数 L 的缩短为 aL 的比例系数 a 刻画。如此计算的优先级参数 a 具有实时性，更加灵活，对于问题的解决更具有普适性。

模型建立：

输入：$p_0$　$x'$　t（短途车返回准备进入蓄车池的时刻）　L

由收益平衡公式：

$$\frac{h(x')-2g(x')+h\left(y(x')\right)}{\frac{2x'}{v}+time(p_0,aL,t)}=\frac{\mathrm{h(EX\_L)}-2\mathrm{g}(EX\_L)+\mathrm{h}(\varphi(EX\_L))}{\frac{2*\mathrm{EX\_L}}{v}+time(p_0,L,t)}$$

解出$time(p_0,aL,t)$的值，将$\hat{a}$在[0,1]区间内遍历，然后比较$time(p_0,\hat{a}L,t)$和$time(p_0,aL,t)$，找到使得这两个值最接近的$\hat{a}$即为所求 a 的值。

接下来我们输入一组确定的数据以验证模型的可行性和可计算性：

根据查到的资料，可知在上海浦东机场对于$x_0$的定义为 25 km，此外可取以下可靠

数据: $p_0$ = 150 人，L = 200 辆，t = 600min（表示上午 10：00）

接着给定一个短途车的接客里程数$x'$，便可计算出要分配给该车的插队系数 a。由此计算出一系列的点后，便可拟合出一个 a 关于接客里程数$x'$的函数曲线，如下图。

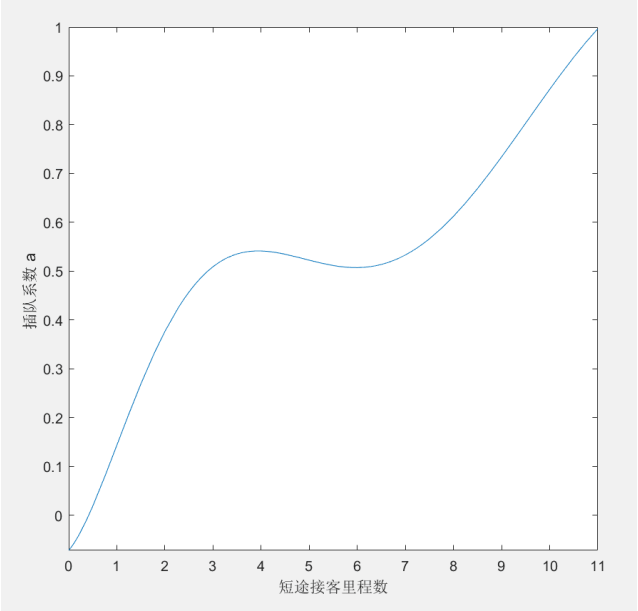



图 4.5.1

该函数的表达式为 a = 0.000000018142049594289056233965338311119*x^8 - 0.000001779305858530639803153586969708 4*x^7 + 0.000070058916392224079546305337551 36*x^6 - 0.001407396411531562510979775915132*x^5 + 0.015078884082165644811568405714297*x^4 - 0.080866472547221068811751365501 55*x^3 + 0.165079759967172196022389130121 16*x^2 + 0.117358936226380605738484064204 39*x - 0.071843426455635073235583831774 42

由此可知，对于机场来说，只要事先设置好通过监测设备动态记录的数据$p_0$、L、t，然后对于任一个返回的短途车，例如通过该短途车上安装的 GPS 系统得到该短途的里程数$x'$，则可计算出一个 a，从而确定了该车的排队位置，因此该方案是可行的。

## 五、 模型评价及改进

我们在问题（1）中所建立的模型中所使用地打车距离概率分布图是由人口密度图而推演所得，认为出租车地行驶路线为直线，与实际问题有一定的出入，这个误差是由

于无法得到准确的打车距离概率分布图而造成的，若能得到该图，则模型可进一步优化。

问题（2）中我们所使用的数据如油费、平均的速度等，都是取平均值带入模型中进行求解，但是这样未免有些粗糙，所以若能够得到更为精确的数据的相关信息，如变化曲线、概率分布图，则可以求出更加准确的解，也使得模型的实用性得到了提升。

我们在问题（3）在假定每个司机作出反应的时间是相同的，而实际生活中却是有一定的偏差的，故而这个偏差就会影响到最终得到的优化方案，我们可以将(3)中模型进行改进，即$t_k^j$中 k=1,2,3 时均是带有随机数的值，仍可以通过大量的数据模拟取得到一个在考虑反应偏差时间的情况下的最优解。

## 六、 参考文献


[1]中国人口密度.http://bcl.carto.com/viz/583c2602-9244-11e3-9d1b-0e49973114de/embed_map?title=true&description=true&search=false&shareable=true&cartodb_logo=true&layer_selector=true&legends=true&scrollwheel=true&sublayer_options=1%7C1%7C1%7C1&sql=SELECT%20.

[2]张泉峰. 首都机场接续运输协调保障技术研究与实现[D].电子科技大学,2015.

[3]各航空公司机型分布数量.[DB/OL].2019.https://wenku.baidu.com/view/79d3e33166ec102de2bd960590c69ec3d5bbdbc5.html

[4] 浦东收费与里程关系.[EB/OL]. https://www.shanghaiairport.com/pdjc/jcjt/index_43742.html

[5] 胡稚鸿,董卫,曹流,高忠,陆志勇,吕俊,黄宏标,顾非凡.大型交通枢纽出租车智能匹配管理系统构建与实施[J].创新世界周刊,2019(07):90-95.

[6]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.

[7] 每辆车平均搭载的旅客数.https://bbs.feeyo.com/thread-5931815-1-1.html.


## 七、 附录

附录 1 网页时刻表 python 抓取

```
import re
import urllib.request
from bs4 import BeautifulSoup, Tag

attr_list = ["机场", "起飞/到达", "航线名", "机型", "每周班期", "参考准点率", "航空公司/航班号", "价格排序", "出发时间"]
parsed = []


class Data(object):

    @staticmethod
    def parse(html_element):
        func_dict = {}
        func_dict[9] = Data.week_parser
        func_dict[15] = Data.price_parser

        data = []

        i = 0
        attrs = html_element.contents
        for attr in attrs:
            if type(attr) is Tag:
                data.append(func_dict.get(i, Data.default_parser)(attr))
            i += 1

        return ','.join(data) + '\n'

    @staticmethod
    def default_parser(attr_element):
```

```
    res = []
    for mini in attr_element.children:
        if mini.string is not None:
            tmp = re.sub('\s', '', mini.string)
            if tmp is not '':
                res.append(tmp)
    return ';'.join(res)

@staticmethod
def week_parser(attr_element):
    res = []
    for mini in attr_element.children:
        if type(mini) is Tag:
            _class = mini.attrs.get('class', [''])
            res.append('1' if _class[0] != 'none' else '0')
    return ''.join(res)

@staticmethod
def price_parser(attr_element):
    '<[^>]*>'
    res = []
    for mini in attr_element.children:
        text = mini.text if type(mini) is Tag else mini.string
        tmp = re.sub('\s', '', text)
        if tmp is not '':
            res.append(tmp)
    return ';'.join(res)
```

```
if __name__ == '__main__':
    print(','.join(attr_list))
    base_url = "https://flights.ctrip.com/schedule/departairport-
pudong/inmap-{0}.html"
    for page_num in range(1, 47):
        print("Parsing "+ str(page_num))
        url = base_url.format(page_num)
        resp = urllib.request.urlopen(url)
        html = resp.read()
        bs = BeautifulSoup(html, "html.parser")
        form = bs.find('tbody', id='fltl')
        page_num = 0

        entries = form.contents
        for entry in entries:
            if type(entry) is Tag:
                parsed.append(Data.parse(entry))
    with open('data.csv', 'w',encoding='utf8') as f:
        f.writelines(parsed)
```

附录 2 问题 3 模拟 Java 代码

```
package contest;

import java.util.Map;

public class CarPosition {
   private Path path_1;
   private Path path_2;
   private int number = 0; // 总车位数
   private Port port;
```

```
    public CarPosition(Port port) {
        this.port = port;
        path_1 = new Path(port, 1);
        path_2 = new Path(port, 2);
    }

    public int getPositions() {
        return number;
    }

    public void construct(int x) {
        path_1.build(x);
        path_2.build(x);
        number = 2 * x;
    }

    public void generateRandomTime() {
        path_1.generateRandomTime();
        path_2.generateRandomTime();
    }

    public Map<Integer, Double> getPath1BusyTime() {
        return path_1.getBusyTime();
    }

    public Map<Integer, Double> getPath2BusyTime() {
        return path_2.getBusyTime();
    }

    public double getUsedTime() {
        if (path_1.getUsedTime() > path_2.getUsedTime()) {
//            System.out.print(path_1.getUsedTime());
//            System.out.print(path_1.getUsedTime());
```

```
//            System.out.print(path_1.getUsedTime());
            return path_1.getUsedTime();
        } else {
//            System.out.print(path_2.getUsedTime());
//            System.out.print(path_2.getUsedTime());
//            System.out.print(path_2.getUsedTime());
            return path_2.getUsedTime();
        }
    }
}

package contest;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Path {
    private List<Position> positions = new ArrayList<Position>();
    private Map<Integer, Double> busyTime = new HashMap<Integer, Double>();
    private Port port;
    private double usedTime = 0;//该车道用时
    private double run = 5.0;//启动时间
    private double t0 = 3.6;//车走一个车位时间
    private double C1 = 3.5;//车道宽
    private double C2 = 5;//停车位
    private double V0 = 1.5;//人平均步速
    private int index;//轨道编号，1 号靠近出口
    double k = positions.size() + 1; //该车道车位数

    public Path(Port port, int x) {
```

```
        this.port = port;
        this.index = x;
    }

    public Map<Integer, Double> getBusyTime() {
        return busyTime;
    }

    public void generateRandomTime() {
        for (Position position : positions) {
            position.calculate();
            position.getIn();
            int positionIndex = position.getNumber();
            double humanTime = Math.sqrt(positionIndex*positionIndex*C2*C2 + (index)*(index)
* C1*C1)/V0;
            busyTime.put(positionIndex, /*busyTime.get(positionIndex) +*/ position.getTime() +
humanTime + run + (k - positionIndex) * t0 + (positionIndex + 1)*t0);
        }
    }

    public void build(int x) {
        if (positions.size() == 0)
            for (int i = 0; i < x; i++) {
                positions.add(new Position(port, i));
                busyTime.put(i, 0.0);
            }
    }

    //一轮的时间
    public double getUsedTime() {

        double human1 = Math.sqrt(C2*C2 + (index)*(index) * C1*C1)/V0;//人走到第 1 排车
的路上用时
```

```
        double human0 = Math.sqrt((index)*(index) * C1*C1)/V0;
        if (positions.size() == 1) {
            usedTime = human0 + 1*t0 + positions.get(0).getTime() + (k)*t0;
        } else {
            if (human1 + 2*t0 + positions.get(1).getTime() < human0 + t0 +
positions.get(0).getTime() + run) {
                //k*t0 + (k-1) * t0 是前两排车进入到车位的时间
                usedTime = k*t0 + (k-1) * t0 +    human0 +    positions.get(0).getTime() + run + run;
            } else {
                usedTime = k*t0 + (k-1) * t0 +    human1 + 2*t0 + positions.get(1).getTime() + run;
            }
        }

        if (positions.size() > 2)
            for (int i = 2; i < positions.size(); i++) {
//                double human_i = C * Math.sqrt(1 + (index - 1)*3 + i * i);
//                double human_i1 = C * Math.sqrt(1 + (index - 1)*3 + (i-1)*(i-1));
                double human_i = Math.sqrt(i*i*C2*C2 + (index)*(index) * C1*C1)/V0;
                double human_i1 = Math.sqrt((i - 1)*(i - 1)*C2*C2 + (index)*(index) * C1*C1)/V0;
                if (human_i + (i + 1) * t0 + positions.get(i).getTime() < human_i1 + i * t0 +
positions.get(i - 1).getTime() + run) {
                    //(k - i) * t0  是进入到车位时间
                    usedTime += run; //(i + 1) * t0  是从车位出来时间
                } else {
                    usedTime += human_i + (i + 1) * t0 + positions.get(i).getTime() - (human_i1 + i *
t0 + positions.get(i - 1).getTime() + run) + run;
                }
            }
        return usedTime;
    }
}

package contest;
```

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class Port {

   private List<Double> calculateList = new ArrayList<Double>();
   private CarPosition carPositions = new CarPosition(this);

   public void constructSystem() {
      File file = new File("input.txt");
      String line;
      BufferedReader br = null;
      try {
         br =    new BufferedReader(new FileReader(file));
         while ((line = br.readLine()) != null) {
            String[] stringArray = line.split("\\s");
            calculateList.add(Double.parseDouble(stringArray[8]));
//             System.out.println(calculateList.get(0));
         }
      }
      catch (Exception e) {
         try {
            br.close();
         } catch (IOException e1) {
            e1.printStackTrace();
```

```
        }
    }
}

public List<Double> getList() {
    return calculateList;
}

public void simulate(int x) {//x 为单车道车位数量
    int waitingCar = 8000;
    long useTime = 0;
    long onceTime = 0;
    List<Double> busyTimePath_1 = new ArrayList<Double>();
    List<Double> busyTimePath_2 = new ArrayList<Double>();
    List<Double> busyRatio_1 = new ArrayList<Double>();
    List<Double> busyRatio_2 = new ArrayList<Double>();
    for (int i = 0; i < x; i++) {
        busyRatio_1.add(0.0);
        busyRatio_2.add(0.0);
        busyTimePath_1.add(0.0);
        busyTimePath_2.add(0.0);
    }
    for (int i = 0; i < 1000; i++) {

        while(waitingCar > 0) {
            waitingCar -= carPositions.getPositions();
            carPositions.construct(x);
            carPositions.generateRandomTime();

            for (int j = 0; j < carPositions.getPath1BusyTime().size(); j++) {
                Map<Integer, Double> map = carPositions.getPath1BusyTime();
                busyTimePath_1.set(j, busyTimePath_1.get(j) + map.get(j));
            }
```

```
            for (int j = 0; j < carPositions.getPath2BusyTime().size(); j++) {
                Map<Integer, Double> map = carPositions.getPath2BusyTime();
                busyTimePath_2.set(j, busyTimePath_2.get(j) + map.get(j));
            }

            useTime += carPositions.getUsedTime();
            onceTime += carPositions.getUsedTime();
        }
        for (int j = 0; j < x; j++) {
            busyRatio_1.set(j, busyRatio_1.get(j) +    busyTimePath_1.get(j)/onceTime);
            busyRatio_2.set(j, busyRatio_2.get(j) +    busyTimePath_2.get(j)/onceTime);
        }
        waitingCar = 8000;
        onceTime = 0;
        for (int j = 0; j < x; j++) {
            busyTimePath_1.set(j, 0.0);
            busyTimePath_2.set(j, 0.0);
        }
    }
    BufferedWriter br = null;
    File file = new File("output.txt");
    try {
        br = new BufferedWriter(new FileWriter(file));
    } catch (IOException e) {
        e.printStackTrace();
    }

    System.out.println("平均时间：" + useTime/1000.0);
    System.out.println("1 车道各车位忙闲率：");
    for (int i = 0; i < x; i++) {
//            Map<Integer, Double> map = carPositions.getPath1BusyTime();
        System.out.println("1 车道" + i +"号  " + busyRatio_1.get(i)/1000.0);
    }
```

```
        System.out.println("2 车道各车位忙闲率：");
        for (int i = 0; i < x; i++) {
//            Map<Integer, Double> map = carPositions.getPath2BusyTime();
            System.out.println("2 车道" + i +"号  " + busyRatio_2.get(i)/1000.0);
        }
        try {
            br.write(useTime/1000.0 + "\n");
            br.newLine();
            for (int i = 0; i < x; i++) {
                br.write(busyRatio_1.get(i)/1000.0 + "\n");
                br.newLine();
            }
            for (int i = 0; i < x; i++) {
                br.write(busyRatio_2.get(i)/1000.0 + "\n");
                br.newLine();
            }
            br.newLine();



        } catch (IOException e) {
            e.printStackTrace();
        } finally{
            try {
                br.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        };
    }

    public static void main(String[] args) {
        Port port = new Port();
```

```
        port.constructSystem();
        port.simulate(4);
    }

}

package contest;

public class Position {
    private double randomTime = 0;
    private Port port;
    private int number;//下标

    public Position(Port port, int x) {
        this.port = port;
        this.number = x;
    }

    public int getNumber() {
        return number;
    }

    public double calculate() {
        double random = Math.random();
        double result = 0;
//        System.out.println("random    " + random);
        for (int i = 1; i < port.getList().size(); i++) {
            if (random >= port.getList().get(i - 1) && random < port.getList().get(i)) {
//                    System.out.println("res  " + ((random - port.getList().get(i - 1))/(random - port.getList().get(i)) ));
                result = ((random - port.getList().get(i - 1))/(port.getList().get(i) - port.getList().get(i - 1)) )*
                        3 + 3 * i;
```

```
            }
        }
//        System.out.println("result    " + result);
        return result;
    }

    public void getIn() {
        randomTime = this.calculate();
    }

    public double getTime() {
        return randomTime;
    }

}
```

附录 3 抓取信息数据处理

```
package test;

public class pair implements Comparable<pair>{
    private time time = new time();
    private int number;
    @Override
    public int compareTo(pair o) {
        if(this.time.getHour() > o.time.getHour() || (this.time.getHour() ==
o.time.getHour()&&this.time.getMinute() >= o.time.getMinute())){
            return 1;
        }
        return -1;
    }

    public pair(int hour, int minute, int number) {
        time.setHour(hour);
```

```
        time.setMinute(minute);
        this.setNumber(number);
    }

    public int getNumber() {
        return number;
    }

    public time getTime() {
        return time;
    }

    public void setNumber(int number) {
        this.number = number;
    }
}
```

```
package test;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class produceData {
```

```
    public produceData() throws IOException {
        File file = new File("datatxt.txt");
        File outFile = new File("output.txt");
        String line;
        BufferedReader br = null;
        BufferedWriter writer = null;
        Map<String, Integer> countMap = new HashMap<String, Integer>();
        List<pair> timeOrderList = new ArrayList<pair>();

        try {
            br =    new BufferedReader(new FileReader(file));
            while ((line = br.readLine()) != null) {
//                System.out.println(line);
                String[] firstStrings = line.split(",");
                String[] endTimeString = firstStrings[1].split(";");
                if (endTimeString.length > 1 && countMap.containsKey(endTimeString[1])) {
                    countMap.put(endTimeString[1], countMap.get(endTimeString[1]) + 1);
                    System.out.println(endTimeString[1]);
                }
                if (endTimeString.length > 1 && !countMap.containsKey(endTimeString[1])) {
                    countMap.put(endTimeString[1], 1);
                    System.out.println(endTimeString[1]);
                }
            }

            for(String keyString : countMap.keySet()) {
                String[] keyStrings = keyString.split(":");
//                System.out.print(countMap.get(keyString).intValue());
                pair pair = new pair(Integer.parseInt(keyStrings[0]), Integer.parseInt(keyStrings[1]),
countMap.get(keyString).intValue());
                timeOrderList.add(pair);

            }
```

```
//          static class TimeComparator implements Comparator<Object> {
//              public int compare(Object object1, Object object2) {// 实现接口中的方法
//              pair p1 = (pair) object1; // 强制转换
//              pair p2 = (pair) object2;
//              return new Double(p1.price).compareTo(new Double(p2.price));
//              }
//          }


          Collections.sort(timeOrderList);

          writer = new BufferedWriter(new FileWriter(outFile));
//          for (String keyString : countMap.keySet()) {
//              writer.write(keyString + "    " + countMap.get(keyString));
//              writer.newLine();
//          }
          for (int i = 0; i < timeOrderList.size(); i++) {
              writer.write(timeOrderList.get(i).getTime().getHour()          +          ":"          +
timeOrderList.get(i).getTime().getMinute() +" "+ timeOrderList.get(i).getNumber());
              writer.newLine();
          }

      }
      catch (Exception e) {
      } finally {
          try {
              if (br != null) {
                  br.close();
              }
              if (writer != null) {
                  writer.close();
              }
          } catch (IOException e1) {
```

```
                e1.printStackTrace();
            }
        }
    }

    public static void main(String[] args) throws IOException {
        produceData producing = new produceData();
    }
}

package test;

public class time {
    private int hour;
    private int minute;
    public int getHour() {
        return hour;
    }
    public void setHour(int hour) {
        this.hour = hour;
    }
    public int getMinute() {
        return minute;
    }
    public void setMinute(int minute) {
        this.minute = minute;
    }

}+
```

附录四 问题 4 的一个计算示例(某些函数模块在附录三中)

```
t = 600;
p0 = 150;
```

```
x0 = 25;
L = 200;
v = 5/6;

EX_L = ex_l(x0)

right = (h_x(EX_L, t) - 2 * g_x(EX_L) + h_x(fi(EX_L), t))/(2 * EX_L / v + get_wait_time(p0, L, get_list(t), t));


A = [];
cnt = 1;
for x_p = 0:1:25 % 拟合 a 与 x_p 关系


    min = 10000000000000000000000000000000000000000000000000;
    a = 0;
    for i = 0:0.001:1
        left = (h_x(x_p, t) - 2 * g_x(x_p) + h_x(fi(x_p), t))/(2 * x_p / v + get_wait_time(p0, i * L, get_list(t), t));
        delta = abs(left - right);
        if (delta < min)
            min = delta;
            a = i;
        end
    end

    A(1, cnt) = a;

    cnt = cnt + 1;
end

X_P = 0 : 1 : 25;
```

```
% 拟合为多项式曲线
p = polyfit(X_P, A, 8);
% 显示概率密度函数
poly2sym(vpa(p))

% 画图
fplot(poly2sym(p), [0 11]);
```

附录五 里程概率密度函数拟合

```
%%%%%%%%%%%%
% 根据不同距离的平均人流量等级，拟合出打车距离的概率密度函数
%%%%%%%%%%%%
function res = f_x()

% 到机场的距离(km)样本点
X = [0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75];

% 相应距离的人流量等级
Y = [0 1.0000 2.8000 3.5000 3.1000 2.9000 2.8000 3.3000 4.2500 4.2857 2.5714 3.0000 2.6667
2.8000 2.0000 2.0000];

% 拟合为多项式曲线
p = polyfit(X, Y, 10);

% 概率密度函数的定义域：[a, b]
a = 0;
b = 75;

% 将曲线标准化为概率密度函数
p = p / diff(polyval(polyint(p),[a b]));

% 显示概率密度函数
```

```
% poly2sym(vpa(p))

%  画图
% fplot(poly2sym(p), [a b]);

%p(1,12) = 0;
res = p;
```

附录六  排队模型的依赖性分析

```
function avg = A(p0,L,t)
    EX = 38.6306;
    v = 5/6;
    var = get_wait_time(p0, L, get_list(t), t);
    avg = (h_x(EX, t) + h_x(fi(EX),t) - 2 * g_x(EX))/(2 * EX / v + var);
end
function avg = B(t)
    EX = 38.6306;
    v = 5 / 6;
    avg = (h_x(EX, t) - g_x(EX)) / (2 * EX / v);
end
function ch = choose(p0,L,t)
A_v = A(p0, L, t);
B_v = B(t);
if (A_v>B_v)
    ch = 'A';
    return;
else
    ch = 'B';
    return;
end
end
function res = ex_l(x0)
global f
a = x0;
```

```
b = 75;
xf = f;
xf(1, 12) = 0;
res = diff(polyval(polyint(xf),[a b])) / diff(polyval(polyint(f),[a b]));
end
%%%%%%%%%%%%
% 根据不同距离的平均人流量等级，拟合出打车距离的概率密度函数
%%%%%%%%%%%%
function res = f_x()

% 到机场的距离(km)样本点
X = [0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75];

% 相应距离的人流量等级
Y = [0 1.0000 2.8000 3.5000 3.1000 2.9000 2.8000 3.3000 4.2500 4.2857 2.5714 3.0000 2.6667
2.8000 2.0000 2.0000];

% 拟合为多项式曲线
p = polyfit(X, Y, 10);

% 概率密度函数的定义域：[a, b]
a = 0;
b = 75;

% 将曲线标准化为概率密度函数
p = p / diff(polyval(polyint(p),[a b]));

% 显示概率密度函数
% poly2sym(vpa(p))

% 画图
% fplot(poly2sym(p), [a b]);
```

```
%p(1,12) = 0;
res = p;
function res = fi(x)
global f
a = 0;
b = x;
xf = f;
xf(1, 12) = 0;
res = diff(polyval(polyint(xf),[a b])) / diff(polyval(polyint(f),[a b]));
end
function val = g_x(x)
k = 7;
val = k * x;
end
function list = get_list(currentTime)
    in = csvread('time_table.csv');
    walkMin = 44.9;
    pPerPlane = 176.977;
    list(1, 1) = 0;
    list(1, 2) = 0;
    % currentTime = hour * 60 + min;
    cnt = 1;
    for i = 1:size(in, 1)
        time = in(i, 1) * 60 + in(i, 2);
        %if (currentTime - walkMin < time & time <= currentTime + 180)
        if (currentTime - walkMin < time)
            list(cnt, 1) = time - currentTime + walkMin;
            list(cnt, 2) = pPerPlane * in(i, 3);
            cnt = cnt + 1;
        end
    end
```

```
end
function wait_time = get_wait_time(p0,L,list,t)
    car_per_min = 1.5633;     %
    wait_time = 0;
    p_per_car = 1.5;     % 工作状态，平均每个车载的人

    if (t < 5 * 60 || t > 11 * 60) % 晚上
        rate = 0.45;
    else
        rate = 0.15;
    end

    T = L    / car_per_min;
    k = car_per_min * p_per_car;
    sum_t = 0;
    cnt = 1;
    p = p0;
    t = 0;

    while (sum_t < T)
        t_zero = t + p / k;
        if (t_zero > list(cnt, 1))
            p = p - (list(cnt, 1) - t) * k;
            temp_sum_t = sum_t + list(cnt, 1) - t;
            if (temp_sum_t >= T)
                wait_time = t + (T - sum_t);
                return;
            end
            sum_t = temp_sum_t;
            t = list(cnt, 1);
            p = p + list(cnt, 2) * rate;
        else
            temp_sum_t = sum_t + t_zero - t;
```

```
            if (temp_sum_t >= T)
                wait_time = t + (T - sum_t);
                return;
            end
            sum_t = temp_sum_t;
            t = list(cnt, 1);
            p = list(cnt, 2) * rate;
        end
        cnt = cnt + 1;
        if (cnt > size(list, 1))
            wait_time = t + get_wait_time(p, L - sum_t * k / p_per_car, list, t);
            return;
        end
    end

end
function val = h_x(x,t)
    if (t < 5 * 60 || t > 11 * 60) %  晚上
        if (x <= 3)
            val = 18;
        elseif (x <= 15)
            val = 18 * 3.1 * (x - 3);
        else
            val = 18 * 3.1 * 12 + 4.7 * (x - 15);
        end
    else   %  白天
        if (x <= 3)
            val = 14;
        elseif (x <= 15)
            val = 14 * 2.5 * (x - 3);
        else
            val = 14 * 2.5 * 12 + 3.6 * (x - 15);
        end
```

```
    end
end
clear;

p0 = [0 25 50 75 100 125 150 175 200 225 250 275];

global f
f = f_x();
L = 0:5:500;
t = 0:10:24 * 60;
%L = 0:40:500;
%t = 0:20:24 * 60;
[LL,tt] = meshgrid(L,t);

%w_t = get_wait_time(0, LL, get_list(tt));
%mesh(LL,tt,w_t);title('Meshplot');
%w_t = zeros(size(t,2), size(L,2));
%L_A;
%t_A;
%L_B;
%t_B;
res_cnt_A = 1;
res_cnt_B = 1;
for n = 1:size(p0, 2)
    w_t = zeros(size(t,2), size(L,2));
    clear L_A t_A L_B t_B
    for i = 1:size(t,2)
        for j = 1:size(L,2)
            %A_v = A(p0(1, n), L(1, j), t(1, i));
            %B_v = B(t(i));
            %w_t(i,j) = A_v / (A_v + B_v);
            ch = choose(p0(1, n), L(1, j), t(1, i));
```

```
            if (ch == 'A')
                L_A(1, res_cnt_A) =    L(1, j);
                t_A(1, res_cnt_A) =    t(1, i);
                res_cnt_A = res_cnt_A + 1;
            else
                L_B(1, res_cnt_B) =    L(1, j);
                t_B(1, res_cnt_B) =    t(1, i);
                res_cnt_B = res_cnt_B + 1;
            end
        end

    end
    %mesh(LL,tt,w_t);title('Meshplot');
    %contour(LL,tt,w_t);title('Meshplot')
    %view(135, 45);
    figure(n);
    %scatter(t_A, L_A,1.5,'w');
    %hold on;
    scatter(t_B, L_B,1.5,'k')
    %hold off;
    ylabel('排队车辆数');
    xlabel('到达时间');
    %zlabel('选择 A 方案的倾向');
    set(gca,'xtick',[0 240 480 720 960 1200
1440],'xticklabel',{'00:00','04:00','08:00','12:00','16:00','20:00','24:00'}) ;
end
```