基于蒙特卡罗模拟的机场出租车问题

摘要

出租车已成为大多数机场乘客选择出行的主要交通工具。国内大多数机场“出发”与“到达”通道分开，出租车司机面临着 1) 排队等待接客，2) 直接放空返回市区两种选择。本文以深圳宝安国际机场为例，从影响机场乘客与出租车司机作出双向选择的若干因素出发，构建司机决策模型；针对宝安机场外部车道构造，提出合理的乘落客区域规划方案。

针对问题一，本文从影响司机乘客双向选择的因素（乘客与司机角度）出发，运用蒙特卡罗模拟出中国 45 座主要城市的 892 条航线，估计出各机场各时间段客流量。运用多元线性回归模型，得出司机留下等待接客的概率可写为：$P = \gamma_0 + \delta_1 t + \delta_2 m + \delta_3 weather + \beta_1 T + \beta_2 S_{out} + \eta_t$。计算司机期望收益 $E(\tilde{w}) = Pw_1 + (1-P)w_2$，若 $E(\tilde{w}) > 0$，则司机应选择等待载客，且此情况下应进一步优化策略使得 $E(\tilde{w})$ 最大；反之，司机可选择空载返回市区。

针对问题二，本文以深圳宝安国际机场为例，搜集到深圳市出租车计价规则、深圳市 2016–2018 年天气报告等数据，通过模拟法并验证，得到机场航班准点率及客流量，运用问题一所提出的线性模型（并推广至机器学习算法），训练模型得到最终策略。

针对问题三，本文考虑到多种国内机场的并行两车道分布实际情况（包括：机场内部电梯通道、出发入口、到达出口等各设施的具体分布），结合排队理论，综合考虑到 `M/M/1`，纵列 `M/M/S`，并列 `M/M/S` 三种方案，最终得出符合问题三的高效率设计方案：设置 6 个可同时搭载乘客的出租车候客服务台时，乘客排队等待时间成本与车道设置成本之和构成的总费用最小。

针对问题四，本文参考上海市现行机场司机多次往返载客方案，本文建议启用智能化匹配管理系统、发放电子短途票、追加日往返机场补贴三个方向，弥补出租车司机等待载客的时间成本损失，保障该类司机的“优先权”，同时建议将所需数据持久化，避免决策无数据支持。

本文编程语言为 Python；深圳市天气预测数据来源为深圳市气象局官网；机场准点率信息来源为 VariFlight；问题一、二中所需中国主要城市航线数模拟及各时段旅客流量由蒙特卡洛模拟及剪枝得出。

关键字：蒙特卡洛模拟　机器学习算法　排队论　系统最优原理　边际收益递减

一、 问题重述

出租车已成为大多数机场乘客选择出行的主要交通工具。国内大多数机场“出发”与“到达”通道分开，送客到机场的出租车司机面临着 1) 排队等待接客，2) 直接放空返回市区两种选择。

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的可观测信息。通常司机可依据个人经验判断某季节或某时间段抵达航班的多少和可能乘客数量的多寡，从而决定是否在机场周边接客。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”，安排一定数量的乘客与司机双向匹配。而现实中仍有很多影响出租车司机决策的确定和不确定因素。

(1) 分析与出租车司机决策相关因素的影响机理，即各影响因素对司机的期望收益间的数量关系，建立出租车司机选择决策模型，并给出司机的选择策略。
(2) 基于问题一所得模型，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) 机场落客与载客区周边常会出现出租车排队和乘客排队的情况。某机场“乘车区”现有两条并行车道，应如何设置最佳“上车点”、合理安排出租车和乘客，以保证车辆和乘客安全且使总乘车效率最高?
(4) 机场的出租车载客收益与载客的行驶里程有关，出租车司机可多次往返载客但不可拒载。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（即在司机的时间成本损失上做出一定弥补），试给出一个可行的“优先”安排方案。

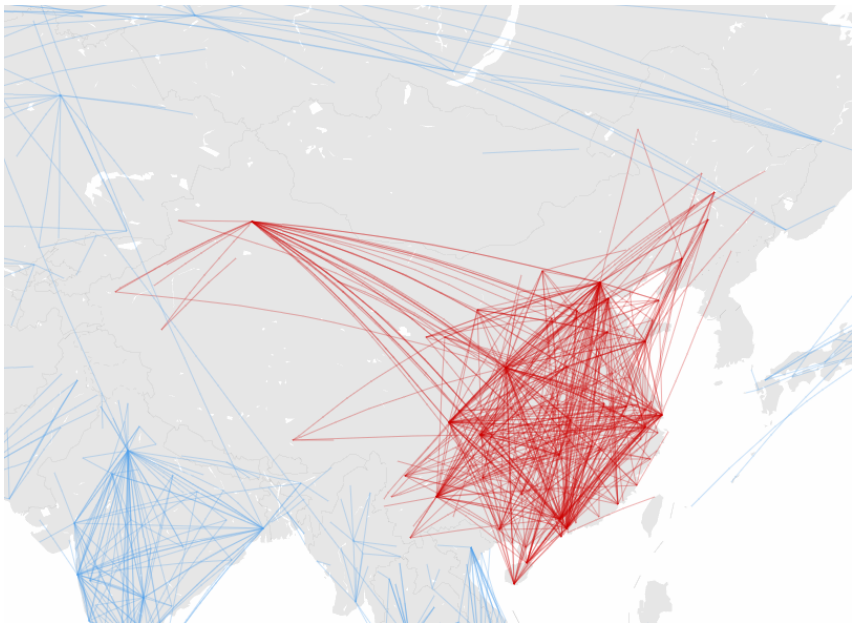
二、 模型假设

2.1 蒙特卡洛模拟数据假设

网络上相关数据过少导致较难通过有监督的机器学习算法进行模型的训练，于是根据中国国内的真实数据进行数据建模，最后抽离出 5 个变量:

1. 城市机场密度（将国内城市进行梯度划分）[1]；
2. 机场停机位数量（将国内机场进行梯度划分）[2]；
3. 飞机时速（统计结果见附录 B的 `settings.py` 文件）；
4. 飞机客容量（由搜索引擎搜索国内各大机场使用飞机型号及搭载乘客人数统计得出）；
5. 机场跑道容量[3]。

(a) 数据建模得出机场位置及航线图



(b) 飞常准世界航线图

图 1　蒙特卡洛模拟与实际数据比较

同时根据一定的规则（见附录 B的 `model_airline.py` 文），规划机场之间的航线，从而得到机场的客容量，由图 1a与图 1b，从图像上来说两者基本符合。因此进一步模拟，产生各大机场的实时数据，我们选取第一梯队城市进行绘图，其中横坐标代表一天的秒数，纵坐标标示客流量，如图 2，通过与能找到的少量真实数据进行比较，数据趋势大致相同，我们假设蒙特卡洛模拟可以模拟出机场总客流量。

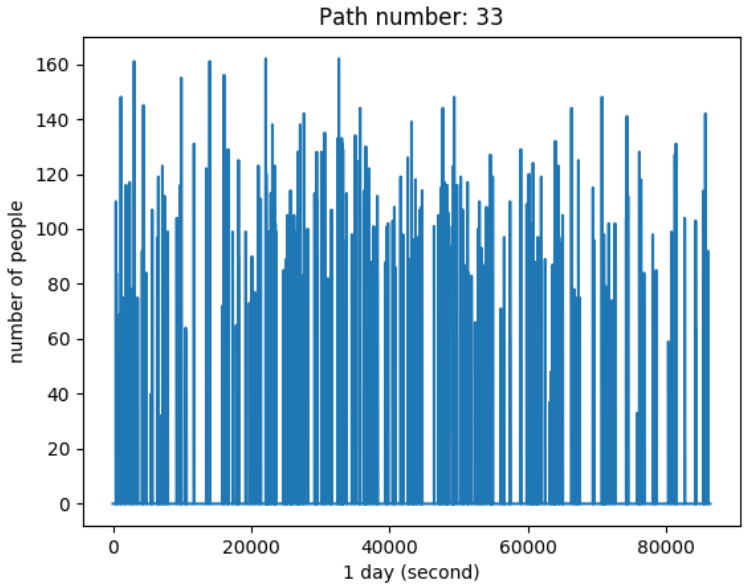


图 2　实时模拟机场总客流量

2.2 多元线性回归的五条基本假设

1. 问题一中预测司机载客概率 P 的模型可写为各解释变量的线性组合，即

$$P = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + u$$
$$= \beta_0 + \sum_{i=1}^{k} \beta_i x_i + u.$$

   其中，$x_i (i = 1, 2, \ldots, k)$ 在本文中为各个解释变量，$k = 6$，$\beta_i (i = 1, 2, \ldots, k)$ 在本文中为模型中各解释变量的系数，u 为残差项；

2. 假设模型含有 n 个观测值，则对于每一个观测值中的 k 个解释变量，均有且仅有一个对应的观测值；
3. 假设各解释变量之间不相关，无完全共线性；
4. 假设残差项关于所有解释变量的条件期望为 0，即

$$E(u|x_1, x_2, \ldots, x_k) = 0.$$

5. 各解释变量满足同方差性，即

$$Var(u|x_1, x_2, \ldots, x_k) = \sigma^2,$$

   其中，σ 是常数。

2.3 解释变量分布假设

1. 假设天气状态/航班延误状态服从两点分布，即

$$Weather = \begin{cases} 1 & \text{航班延误/恶劣或极端天气} \\ 0 & \text{航班正常/天气晴好} \end{cases}$$

2. 假设机场距乘客目的地距离 S_{out} 服从正态分布 $N(10,3)$，单位为 km。

三、 符号说明

3.1 符号说明

符号说明如表 1。

符号	含义
N	机场乘客到达数
N_p	到达旅客中意愿乘出租车的人数
m	乘客行李重量
$weather$	天气状态，二元变量
S_{out}	机场距乘客目的地距离
t_d	司机排队等待时长
T	载客高峰每单平均运营时长
P	司机选择载客的概率
t	乘客到达时间段（乘客打车时段）
$\tilde{w}$	司机收益（随机变量）

表 1　符号说明

四、 问题一的模型建立与求解

4.1 问题一原理

4.1.1 蒙特卡洛模拟方法

定义 1 ▶ 蒙特卡洛模拟[4]

通过生成合适的随机数并且观察符合某些属性来解决问题的方法称为蒙特卡洛模拟（或蒙特卡洛方法、随机抽样、统计试验方法）。

网络上相关数据过少导致较难通过有监督的机器学习算法进行模型的训练，于是根据中国国内的真实数据进行数据建模，最后抽离出 5 个变量：城市机场密度、机场停机位数量、飞机时速、飞机客容量以及机场跑道容量。同时根据一定的规则（见附录 B的 `model_airline.py` 文），规划机场之间的航线，从而得到机场的客容量。根据实际情况进行建模，使用如图 3所视架构，共模拟中国 45 座城市、892 条航线、14422 架飞机，为解题提供充分的训练数据。

其中程序接口易用可拓展，整体架构清晰科学，并且数据格式统一，适合用来进行机器学习算法的训练。

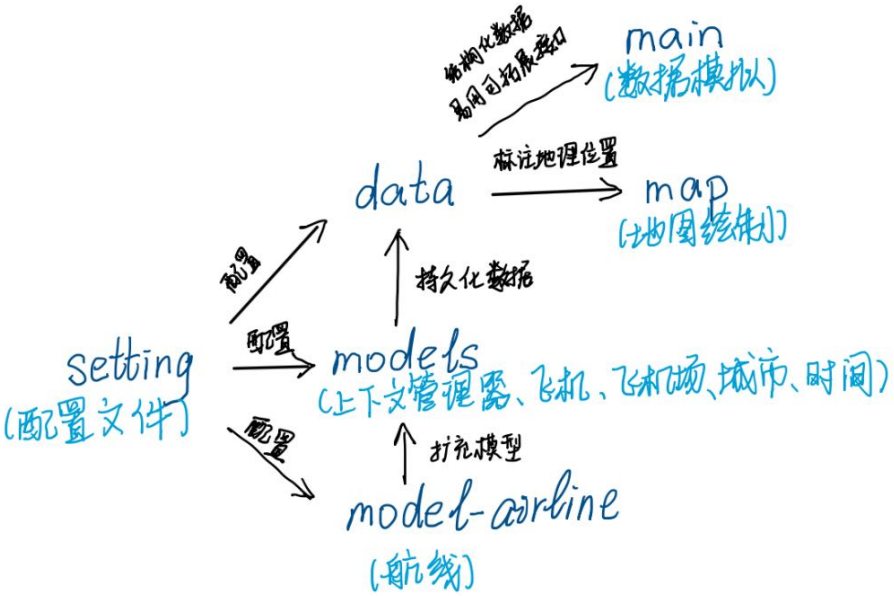


图 3　程序架构

	第一梯队	第二梯队	第三梯队	第四梯队
2018 年旅客吞吐量	>3000 万	1000–3000 万	200–1000 万	<200 万

表 2　中国机场分级依据

4.2 问题一分析

问题一旨在找出影响司机是否在机场选择载客的因素，即研究各因素与司机选择在机场载客返回市区的概率 P 的数量关系。

本文将影响因素拆分为乘客方面与司机方面两个角度。

(1) 乘客方面，即潜在影响乘客选择出租车作为交通方式的因素分为三类：乘客到达时间段（即：乘客打车时段）t，乘客携带行李重量 m，当时天气状况（即航班到达状态：延误/按时抵达）$weather$。由此三因素，进而影响当时乘客到达人数 N 及意愿选乘出租车的人数 N_p。

(2) 司机方面，即影响司机选择在机场载客返回市区概率的因素为：乘客中意愿选乘出租车人数 N_p，司机排队载客的平均等待时长 T，司机的载客路程（即：机场至乘客目的地距离）S_{out}。

同时使用中国范围内的直辖市、地级市、特别行政区、省会城市及各省其他主要知名城市的经纬度坐标，建立起“一城一港（北京机场设为两座航空港），两两相连”的飞行网络，见图 1。按照 2018 年旅客吞吐量，将我国机场大致分为四个梯队：

蒙特卡罗模拟步骤具体如下：

(1) 建立起“两两相连”的航空路线后，剔除始发机场与目的地机场间飞行时间短于 30 分钟的航线。

如图4，O 为始发港，设与其相关联的机场为 T_1、T_2、T_3、T_4，已知四条航线距离 v_1、v_2、v_3、v_4，根据余弦公式 $\cos\theta_{ij} = |v_i \cdot v_j| / \left(|v_i| \cdot |v_j|\right)$，$i,j = 1,\ldots,4$，可得出两两航空港地理位置间的夹角，夹角矩阵为 $\Theta = \{\theta_{ij}\}_{i,j=1,\ldots,4}$ 运用离群值检验，得出临界夹角 α，$0.5° < \alpha < 2°$，将 θ 值小于 α 的航线剔除。本图中，由于 O 港距 T_2 港过近且 θ_{12} 较小，v_2 航线可由 v_1 代替，故将 v_2 剔除。

(2) 将两地距离作为航线保留与否的权重，航线里程越长，权重越大，航线则越易被保留。计算上步剔除后各机场的节点数（即各机场的航线数），得到复杂矩阵 $C_{1\times n}$，检查跨梯队间的机场是否仍存在航线联系，若存在，由于跨梯队间机场航线通常由同梯队联程航线代替，故剔除此类航线联系。

我国大陆三千万级以上机场的机位数及 2018 年旅客吞吐总量如表3所示：

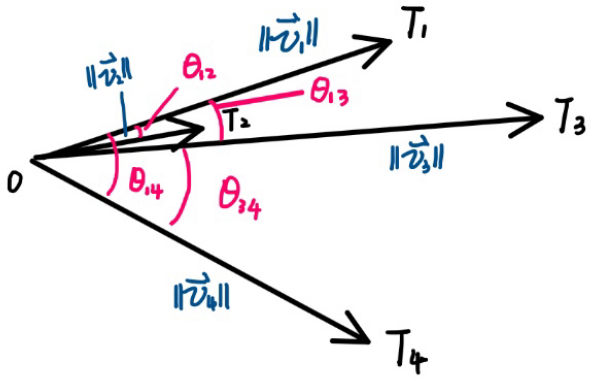


图 4　飞机场节点剪枝策略

IATA 编号	机场名	机位数量 (个)	旅客吞吐量 (万)(2018 年)
XIY	西安咸阳国际机场	127	4465
CTU	成都双流国际机场	178	5291
KMG	昆明长水国际机场	110	4709
CKG	重庆江北国际机场	209	4160
SHA	上海虹桥国际机场	155	4363
PVG	上海浦东国际机场	218	7405
CAN	广州白云国际机场	220	6973
PEK	北京首都国际机场	314	10098
SZX	深圳宝安国际机场	199	4935
HGH	杭州萧山国际机场	127	3824

表 3　中国大陆三千万级机场机位数及年旅客吞吐量

构建最直接影响 P 的因素与 P 间的数量关系，以多元线性关系为例：

$$\begin{aligned} P &= \hat{N_p} + \beta_0 + \beta_1 T + \beta_2 S_{out} + \epsilon_t \\ &= \gamma_0 + \delta_1 t + \delta_2 m + \delta_3 weather + \beta_1 T + \beta_2 S_{out} + \eta_t \end{aligned} \tag{1}$$

其中，β_i 为待估计参数，η_t，ϵ_t 为白噪声误差项。由此，得出司机选择载客回市区

的概率 P。计算司机在此概率测度下的期望收益，即

$$E(\tilde{w}) = Pw_1 + (1 - P)w_2,$$

其中，w_1，w_2 分别为司机载客所得收益及司机空载返回时的收益（为负）。若 $E(\tilde{w}) > 0$，则司机应选择等待载客，且此情况下应进一步优化策略使得 $E(\tilde{w})$ 最大；若 $E(\tilde{w}) < 0$，司机可选择空载返回市区。至此得到司机选择决策模型。

五、 问题二的模型建立与求解

5.1 问题二分析

问题二旨在将问题一得出的司机决策模型，运用到国内某一城市的机场和出租车数据上作为实证分析，并讨论模型的合理性，得出各解释变量对司机选择载客回城的概率 P 的数量关系（正/负相关及具体数值关系）。

本文选择深圳宝安国际机场及出租车数据为例，对问题一中提出的模型做检测。

5.2 解释变量信息汇总

5.2.1 深圳出租车计价规则

一、“红色”出租小汽车运价项目和标准

（一） 起步价：首 2 公里 11.00 元；

（二） 里程价：超过 2 公里部分，每公里 2.40 元；

（三） 返空费：每天的 6 时 23 时，超过 25 公里部分，每公里按上述里程价的 30% 加收返空费：

（四） 夜间附加费：夜间起步价 16 元，每天的 23 时至次日凌晨 6 时，按上述起步价和里程价的 20% 加收夜间附加费；

（五） 候时费：每分钟 0.80 元；

（六） 大件行李费：体积超过 0.2 立方米、重量超过 20 公斤的大件行李，每件 0.50 元。

二、“绿色”出租小汽车运价项目和标准

（一） 起步价：首 1.5 公里 6.00 元；

（二） 里程价：超过 1.5 公里部分，每公里 2.40 元；

（三） 返空费：每日 6 时 23 时，超过 15 公里部分，每公里按上述里程价的 30% 加收返空费；

（四） 夜间附加费：每天的 23 时至次日凌晨 6 时，按上述起步价和里程价的 20% 加收夜间附加费；

（五） 候时费：每分钟 0.50 元；

要素	2018 年 (天)	2017(天)	2016 年 (天)	累年平均值（天）
高温日数	2	7	5	4.3
低温日数	1	0	2	1.1
大风日数	3	2	2	4.4
暴雨日数	8	11	11	9.2
雾日数	0	0	3	3.3
霾日数	20	22	27	72

表 4　2016–2018 年深圳国家基本气象站天气日数

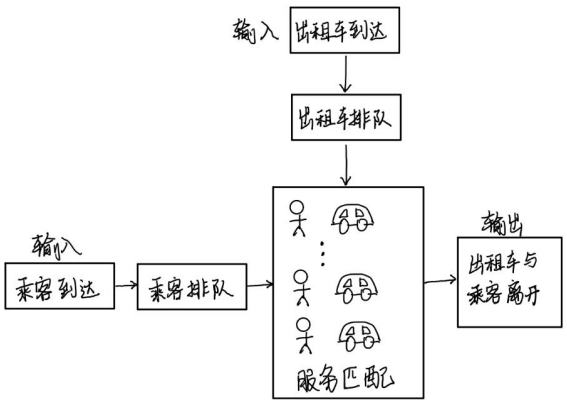


图 5　服务布局系统简图

（六） 大件行李费：体积超过 0.2 立方米、重量超过 20 公斤的大件行李，每件 0.50 元。

因“绿色”清洁能源汽车逐渐普及，故本文采用“绿色”出租车计价规则。

5.2.2 深圳气候概况二元变量——*weather*

下表 4统计了 2016-2018 年深圳市极端天气日数，以此作为计算二元变量 *weather* 取不同状态数时的概率。

由此得

$$weather = \begin{cases} 1, & p = 34/365 = 0.093 \\ 0, & p = 1 - 0.093 = 0.907 \end{cases}.$$

5.2.3 司机等待时长 t_d

假设司机等待时长服从泊松分布，即 $t_d Poisson(30)$，单位为分钟。

对于排队或说等待问题来说，需考虑的因素是排队长度 L 和顾客等待时间 W，无论是从系统的角度还是从顾客的角度来看，两者均是越小越好。对服务布局系统进行了研究，如图 5。其中队列长 L_q 是指系统中正处于排队等待的平均旅客数，队长 L_S 则是指队列长 L_q 与正在接受服务的顾客数之和。等待时间 W_q 则是指顾客从进入系统开始到开始接受服务的平均时间，逗留时间 W_s 是指从顾客进入系统到接受完服务离开系统的平均时间。

用系统中乘客排队的队长 L_s 及其逗留时间 W_s 对系统进行分析，得：

$$L_s = L_q + C_\rho = \frac{1}{C!}\frac{(C\rho)^C\rho}{(1-\rho)^2}P_0 + \frac{\lambda}{\mu}, \tag{2}$$

5.2.4 行李重量 m

大部分民航公司的免费托运行李额为 20kg，而实际旅程中乘客携带行李重量层次不一，且尚无直接渠道得到旅客行李托运信息，因此本文认为 m 是不易观测变量，建议使用其他代理变量加入到回归式中。

m 对民航公司是显式直接以观测变量，本文建议在实际问题解决中考虑该变量对总概率 P 的影响。

六、 问题三的模型建立与求解

6.1 问题三分析

6.1.1 排队模型

定义 2 ▶ 排队模型

排队系统主要由输入过程、排队规则和服务机构构成。输入过程，即顾客到来的时间规律，出租车辆到达的时间规律常被视为服从泊松分布；排队规则，即顾客以何种规则排队，可分为损失制、等待制、混合制；服务机构，可分为单服务台、多服务台并/串联、混合型；服务规则多样，有先到先得、后到先得、随机服务、优先服务等规则。其表示方式为：`X/Y/Z/A/B/C`。其中，

- `X`表示顾客到达流或顾客到达间隔时间的分布；
- `Y`表示服务时间的分布；
- `Z`表示服务台数目；
- `A`表示系统容量限制；
- `B`表示顾客源数目；
- `C`表示服务规则。

定义 λ 为单位时间车辆平均到达率，μ 为单位时间系统服务率，定义 ρ 为服务强度，$\rho = \lambda/\mu$。$\rho < 1$，则系统服务率大于车辆平均到达率满足服务要求，反之不能满足。

`M/M/1`（单点式排队系统）排队模型认为车辆的到达服从泊松分布，出租车服务时间服从负指数分布，仅有一个服务台服务。布局形式如图 6。`M/M/S`模型则有多个服务台服务。

- 单点式出租车排队服务系统是指一列乘客等候上车的队伍对应一个上车点的布局形式；
- 多点纵列式出租车排队服务系统属于面向乘客的带有多个服务台和一个公共队伍的排队系统；
- 多点并列式出租车排队服务系统与多点纵列式类似，但系统内各个上车点的布置形式呈并列式。

6.1.2 三种常见的服务布局对比

详见表格 5

	优点	缺点	适用环境
单点式（如图 6）	造价低； 安全，乘客与出租车同时在各自的等候空间进行排队。	等候时间长，只有当一个乘客服务结束离开排队系统后，后面的乘客才能接受服务。	客源较少的小型枢纽站的出租车上客区或者路侧出租车服务点。
多点纵列式（如图 7）	增加了上车服务点，提高了系统的服务效率。	多个服务点的出租车同时驶离各自的上车点时，车辆之间易产生干扰与冲突。	机场等枢纽的纵向距离较大的交通枢纽。
多点并列式（如图 8）	分散客流，乘客离站效率提高。	容易产生客流干扰及人车冲突。	上客区纵向距离较短的传统枢纽。

表 5　不同出租车排队服务系统的优缺点

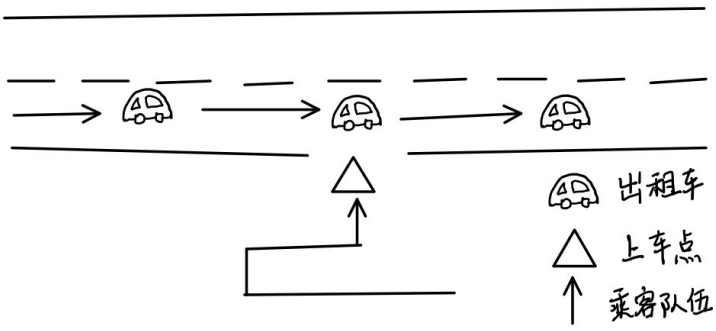


图 6　M/M/1 出租车排队服务布局

6.2 排队模型建立

根据排队论的相关理论，对枢纽内多点式出租车上客区这一排队系统建立排队模型。当上客区排队系统处于全忙期，且系统的服务强度 $\rho < 1$ 时，系统达到稳定状态且不会形成无限排队的现象，在此基础上对系统的输入与输出进行建模。

在系统达到稳态时，C 个服务台工作，系统中出租车乘客数为 n 的概率如下：

$$P_0(C) = \left[\sum_{k=0}^{C-1} \frac{1}{k!}\left(\frac{\lambda}{\mu}\right)^k + \frac{1}{C!}\frac{1}{(1-\rho)}\left(\frac{\lambda}{\mu}\right)^C\right]^{-1} \tag{3}$$

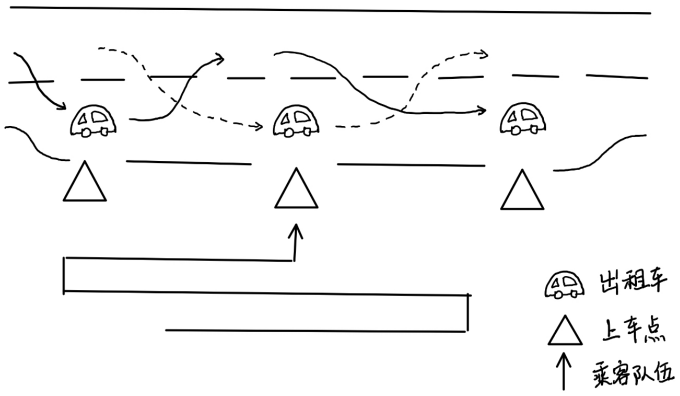


图 7　纵列 M/M/S 出租车排队服务布局

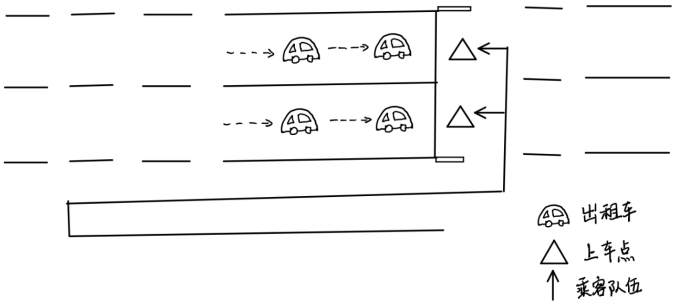


图 8　并列 M/M/S 出租车排队服务布局

$$P_n(C)=\begin{cases}(\lambda/\mu)^k P_0(C)/n, & n=1,2,\ldots,C\\ \left(C!C^{n-C}\right)^{-1}(\lambda/\mu)P_0(C), & n=C+1\end{cases} \tag{4}$$

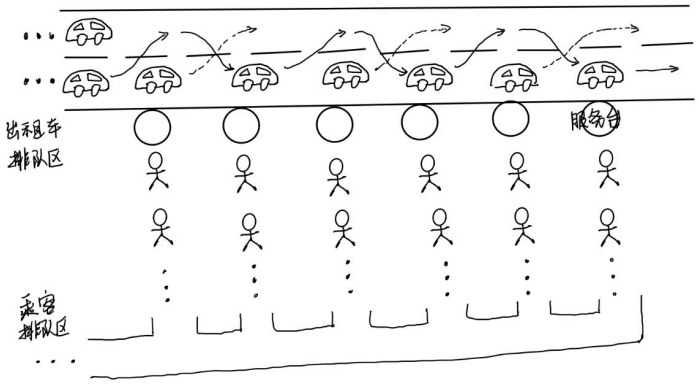


图 9　问题三的解决方案设计简图

用系统中乘客排队的队长 L_s 及其逗留时间 W_s 对系统进行分析，得：

$$L_s = L_q + C_\rho = \frac{1}{C!}\frac{(C\rho)^C\rho}{(1-\rho)^2}P_0 + \frac{\lambda}{\mu}, \tag{5}$$

$$E(W_s) = \frac{P_n(C)}{C\mu(1-\rho)^2} = \frac{n\mu}{n!(n\mu-\lambda)^2}\left(\frac{\lambda}{\mu}\right)^n P_0(C). \tag{6}$$

6.3 排队系统优化

利用排队系统的费用决策模型对排队系统进行优化设计。假设乘客等待时间的总费用为 $Z_1 = \alpha L_s$，上车点建设成本为 $Z_2 = \beta C$，其中 α 为每个乘客单位时间的等待时间成本，β 为单个服务台的服务时间成本与单个上车点的建设费用。当则需要满足两者之和最小才能使得系统进一步优化，即：

$$\min: Z(C) = Z_1 + Z_2 = \alpha L_s(C) + \beta C, \quad Z(C-1) \le Z(C) \le Z(C+1) \tag{7}$$

6.4 结论

在解决本问题时，对国内各交通枢纽已有的服务布局进行了调研，并总结出了最常见的三种进行分析讨论。并以深圳机场及题目要求为例，对多点式的排队服务系统进行了评析。利用排队论中的费用决策模型对排队系统进行优化，设计方案如图 9。经过调查与计算得，当上客区的上车点数为 6，即设置 6 个可同时搭载乘客的出租车候客服务台时，乘客排队等待时间成本与车道设置成本之和构成的总费用最小。

七、 问题四的模型建立与求解

7.1 问题四分析

问题四旨在提出对某些往返于机场短途载客的出租车司机们的“优先权”方案，本文参考上海浦东国际机场现使用的出租车智能匹配管理系统，考虑定位技术与绩效补贴两方向，同时为短途载客司机们提供“优先权”。

7.2 大型交通枢纽出租车智能匹配管理系统

7.2.1 Beckmann 交通分配数学模型

交通分配模型的变量与参数表示如下：

- x_a：路段 a 的交通流量，组成向量为 $x = (\dots, x_a, \dots)$；
- t_a：路段 a 的交通阻抗；
- $t_a(x_a)$：以流量为自变量的阻抗函数；

- f_k^{rs}：点对 (r,s) 间第 k 条路径的交通流量；
- C_k^{rs}：点对 (r,s) 的第 k 条路径阻抗；
- u_{rs}：(r,s) 的最小阻抗；
- $\delta_{a,k}^{rs} = \begin{cases} 1 & \text{路段 } a \text{ 在 } (rs) \text{ 间的第 } k \text{ 条路径} \\ 0 & \text{其他情况;} \end{cases}$
- W_{rs}：(r,s) 间的所有路径集合；
- q_{rs}：(r,s) 间的交通量。

定理 1 ▶ 系统最优原理

在交通网络中的交通量应按某种方式分配，以使网络中所有交通元的总阻抗最小。

系统最优原理的目标函数是使得网络中所有用户阻抗最小，即：

$$\min: Z(x) = \sum_a x_a t_a(x_a)$$

$$\text{s.t.} \begin{cases} \sum_k f_k^{rs} = q_{rs} & \forall r,s; \\ f_k^{rs} > 0 & \forall r,s,k; \\ x_a = \sum_{r,s} \sum_k f_k^{rs} \delta_{a,k}^{rs} & \forall a. \end{cases} \tag{8}$$

该系统称为系统最优模型（SO, System Optimization）。

7.2.2 常见通信系统

- GSM（Global System for Mobile Communications），提供短信。
- GPRS（General Packet Radio System），介于 2G 与 3G 间的通信技术。它具有保持永远在线，按数据流量计费，自如切换、高速传输等优点。
- GPS（Global Positioning System），广泛应用、高精度。
- GIS（Geographic Information System），空间数据处理技术；与 GPS、RS 统称 3S 系统。

7.3 短途往返出租车司机“优先权”设定

7.3.1 发放“智能短途票”

传统纸质短途票发放给从机场接客且能在一个小时内或规定时间返回机场再次接客的出租车司机，以便该司机下一次返回接客时无需排队，节省等待时间成本。而纸质短途票存在票面日期被随意修改、伪造等问题，人工不易管理。

现今国内其他机场可参考上海浦东机场推行的智能化匹配管理系统，实现短途票电子到账；每部出租车均需安装 GPS，以便清楚直观地看到司机是否在一定时间内“短途”，避免弄虚作假，且提高了出租车的运营效率。

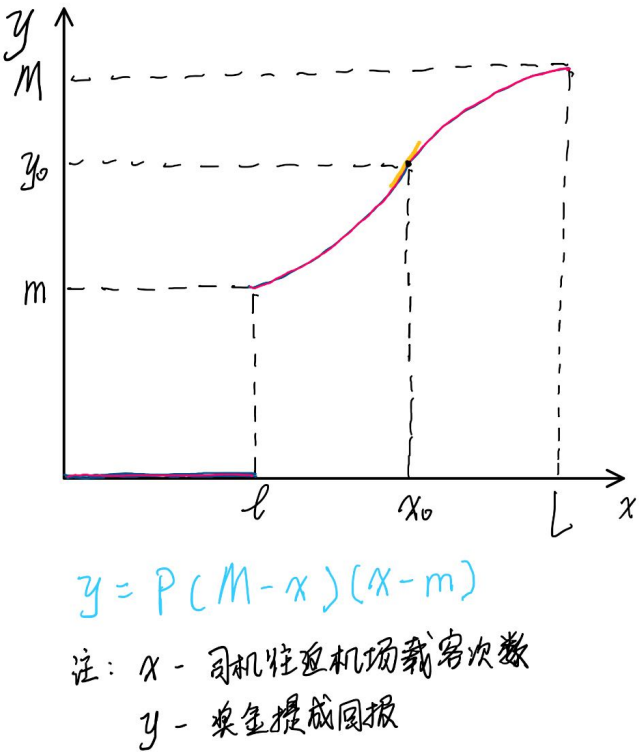


图 10　司机日往返机场次数与提成回报的关系

7.3.2 实行多次短途司机的奖励政策

奖励政策具体草拟如下：

1. 设置司机当日往返机场载客的下界次数 l 与上限 L，上下界具体数值需依据该机场实际承载旅客量、机场周边路况等因素设置。
2. 依据边际回报递减原理，令司机往返载客次数与奖金提成回报关系满足 logistic 函数。如图 10所示，m, M 分别为奖金数的上下界，l, L 分别为当日往返机场载客的下上限次数，点 (x_0, y_0) 为该函数段中导数最大的点。
3. 导数最大即意味着边际收益最大，因此可推断出大多数司机的日往返机场载客数趋于 x_0 次，此时大部分司机的回报为 y_0。此方案一方面能用奖金激励司机短途载客，保障了机场乘车乘客的接机需求；另一方面因接机次数存在上限及边际收益回报递减，能有效地控制司机日往返机场次数，保障了机场周边道路交通的通畅性。

本文建议采取电子短途票与短途接送奖励政策的同步推进，且此种方案不受机场类型、地域的影响，适宜在全国大范围推广。

八、 模型评价与推广

优点：

在问题一中，我们从出租车司机的角度出发，运用线性模型，同时兼顾到了出租车司机的利益与机场客流的疏通能力。基于蒙特卡罗模拟，综合天气等因素，较为完善地对机场乘客的数量变化规律进行了预测。

在问题二中，我们利用了深圳机场及出租车的相关数据，建立了服务布局系统，且对模型的合理性及相关因素的依赖性进行了良好评估，

在问题三中，我们建立多服务台排队模型，对国内现有的服务布局系统进行了优缺点对比。同时利用排队系统的费用决策模型对排队系统进行优化设计，通过绘图方式，清晰、直观地给出了问题的解决方案。

在问题四中，我们给出的方案是具有实用性的，因为我们对国内各大型机场都进行了调研并总结他们处理该类问题的优良办法，从而给出了可直接用于现实环境下的多种方案。

缺点：

研究中将枢纽内出租车上客区的排队系统进行了简化，仅参考了上客区的乘客排队情况。而现实中，出租车排队服务系统通常存在双端排队模式。因此，希望在后续的研究中可以综合考虑多个因素，如出租车排队情况、候客出租车停车位的设置情况等。在此基础上对排队系统进行优化，进而提高出租车乘客的离站效率。

推广：

本模型虽然存在一些不足之处，但是得到的结果还是较为合理的。本模型还可以用于火车站、大型商场等区域。在指标的方面再考虑全面，得到的结果会更让人满意。本模型基于蒙特卡罗模拟，在数据库表现方面良好，之后结合大数据、互联网 + 等，可解决网约车匹配等多方面问题。

参考文献

[1] MODOOD. Administrative-divisions-of-china[EB/OL]. 2018. https://github.com/modood/Administrative-divisions-of-China.

[2] PENTADIINE. 机场（搭乘空中交通及供飞机起降的设施）[EB/OL]. 2019. https://baike.baidu.com/item/ /74273.

[3] W_OU. 跑道容量[EB/OL]. 2018. https://baike.baidu.com/item/ /5320286.

[4] WEISSTEIN, W. E. Monte carlo method[EB/OL]. 2019. http://mathworld.wolfram.com/MonteCarloMethod.html.

[5] FANAEE-T H, GAMA J. Event labeling combining ensemble detectors and background knowledge[J/OL]. Progress in Artificial Intelligence, 2013:1-15. http://dx.doi.org/10.1007/s13748-013-0040-3.

[6] PEDREGOSA F, VAROQUAUX G, GRAMFORT A, et al. Scikit-learn: Machine learning in Python[J]. Journal of Machine Learning Research, 2011, 12:2825-2830.

[7] 杨洋. “打的软件” 对出租车司机行为决策及运营绩效的影响研究—技术效率与配置效率的双重视角[J]. 2016.

[8] 云亮, 罗孝羚, 蒋阳升, 等. 枢纽里站出租车休假排队系统服务台数优化模型[J]. 2015.

[9] 魏中华, 王琳, 邱实. 基于排队论的枢纽内出租车上客区服务台优化[J]. 2017.

[10] 孙宝芸, 张悦, 张丽萍. 高速公路停车收费系统中排队模型的比较分析[J]. 2015.

[11] 李强. 出租车智能管理系统[D]. [出版地不详]: 兰州理工大学, 2011.

[12] 车勇. 基于多人合乘模式的出租车智能调度管理系统设计与研究[D]. [出版地不详]: 同济大学; 同济大学软件学院, 2008.

九、 附录

A 国内主要机场 2019 年 6 月–8 月准点率信息

(1) 表 6: 中国大陆 *TOP* 级（承载客流量超 *3000* 万，*2019* 年 *8* 月）机场准点出发信息表

(2) 表 7: 中国大陆 *TOP* 级（承载客流量超 *3000* 万，*2019* 年 *7* 月）机场准点出发信息表

(3) 表 8: 中国大陆 *TOP* 级（承载客流量超 *3000* 万，*2019* 年 *6* 月）机场准点出发信息表

(4) 表 9: 中国大陆 *TOP* 级（承载客流量 *1000* 万–*3000* 万，*2019* 年 *8* 月）机场准点出发信息表

(5) 表 10: 中国大陆 *TOP* 级（承载客流量超 *1000–3000* 万，*2019* 年 *7* 月）机场准点出发信息表

(6) 表 11: 中国大陆 *TOP* 级（承载客流量超 *1000–3000* 万，*2019* 年 *6* 月）机场准点出发信息表

(7) 表 12: 中国大陆 *TOP* 级（承载客流量 *200* 万–*1000* 万，*2019* 年 *8* 月）机场准点出发信息表

(8) 表 13: 中国大陆 *TOP* 级（承载客流量 *200* 万–*1000* 万，*2019* 年 *7* 月）机场准点出发信息表

(9) 表 14: 中国大陆 *TOP* 级（承载客流量 *200* 万–*1000* 万，*2019* 年 *6* 月）机场准点出发信息表

排行	IATA 编号	机场名	航班离港数	离港准点率	较去年同期增长	平均出发延误时间（分钟）
1	XIY	西安咸阳国际机场	15085	78.45%	3.88%	28.65
2	KMG	昆明长水国际机场	15644	74.31%	19.48%	30.10
3	CKG	重庆江北国际机场	13622	73.97%	7.30%	32.17
4	CAN	广州白云国际机场	20032	71.44%	19.41%	32.57
5	CTU	成都双流国际机场	15420	70.89%	1.52%	37.92
6	SHA	上海虹桥国际机场	11030	66.03%	-1.67%	40.22
7	SZX	深圳宝安国际机场	14646	64.50%	19.10%	41.56
8	PEK	北京首都国际机场	24812	64.46%	7.89%	39.54
9	PVG	上海浦东国际机场	19413	64.00%	8.68%	40.94
10	HGH	杭州萧山国际机场	11334	60.09%	7.48%	48.38

表 6　中国大陆 TOP 级（承载客流量超 3000 万，2019 年 8 月）机场准点出发信息表

排行	IATA 编号	机场名	航班离港数	离港准点率	较去年同期增长	平均出发延误时间（分钟）
1	XIY	西安咸阳国际机场	15208	73.58%	-4.25%	31.31
2	KMG	昆明长水国际机场	15666	73.27%	1.60%	32.25
3	CAN	广州白云国际机场	19862	68.97%	10.31%	34.67
4	SHA	上海虹桥国际机场	11500	68.94%	3.61%	37.76
5	CKG	重庆江北国际机场	13694	68.76%	-2.57%	36.93
6	PVG	上海浦东国际机场	20458	68.26%	10.34%	33.65
7	CTU	成都双流国际机场	15429	67.37%	8.65%	39.91
8	SZX	深圳宝安国际机场	14559	63.04%	1.52%	43.50
9	HGH	杭州萧山国际机场	11575	57.88%	13.86%	49.15
10	PEK	北京首都国际机场	24672	53.12%	-8.58%	53.27

表 7　中国大陆 TOP 级（承载客流量超 3000 万，2019 年 7 月）机场准点出发信息表

排行	IATA 编号	机场名	航班离港数	离港准点率	较去年同期增长	平均出发延误时间（分钟）
1	XIY	西安咸阳国际机场	13828	77.54%	-6.08%	28.21
2	CTU	成都双流国际机场	14554	77.08%	6.53%	31.06
3	KMG	昆明长水国际机场	14149	76.47%	-2.00%	30.00
4	CKG	重庆江北国际机场	12495	73.12%	-8.97%	31.97
5	SHA	上海虹桥国际机场	10829	70.90%	-4.24%	34.59
6	PVG	上海浦东国际机场	19365	69.34%	-1.61%	32.34
7	CAN	广州白云国际机场	18581	67.08%	3.70%	37.88
8	PEK	北京首都国际机场	23466	63.36%	1.93%	37.90
9	SZX	深圳宝安国际机场	13480	60.70%	-9.78%	50.54
10	HGH	杭州萧山国际机场	10797	52.86%	-20.54%	54.68

表 8　中国大陆 TOP 级（承载客流量超 3000 万，2019 年 6 月）机场准点出发信息表

排行	IATA 编号	机场名	航班离港数	离港准点率	较去年同期增长	平均出发延误时间（分钟）
1	HAK	海口美兰国际机场	6620	76.39%	18.40%	28.04
2	NNG	南宁吴圩国际机场	4942	75.80%	14.73%	26.43
3	TNA	济南遥墙国际机场	5932	75.67%	11.25%	27.34
4	WUH	武汉天河国际机场	8778	74.93%	12.18%	27.13
5	DLC	大连周水子国际机场	7250	74.60%	6.24%	30.73
6	KWE	贵阳龙洞堡国际机场	7519	74.26%	15.26%	31.46
7	URC	乌鲁木齐地窝堡国际机场	8774	74.05%	-9.00%	28.04
8	KHN	南昌昌北国际机场	4458	73.62%	15.04%	31.75
9	TYN	太原武宿国际机场	4597	73.45%	18.32%	33.70
10	LHW	兰州中川国际机场	5552	73.02%	26.45%	30.32
11	CGO	郑州新郑国际机场	9432	72.14%	12.93%	34.02
12	CGQ	长春龙嘉国际机场	4244	70.41%	36.49%	37.76
13	SHE	沈阳桃仙国际机场	6370	69.46%	26.43%	37.69
14	CSX	长沙黄花国际机场	8811	68.92%	6.50%	34.65
15	HRB	哈尔滨太平国际机场	6310	68.80%	23.59%	37.57
16	HFE	合肥新桥国际机场	4136	68.07%	24.90%	37.36
17	SYX	三亚凤凰国际机场	4736	67.82%	8.19%	37.37
18	FOC	福州长乐国际机场	4679	67.19%	5.69%	34.77
19	TAO	青岛流亭国际机场	8138	64.79%	16.84%	38.17
20	SJW	石家庄正定国际机场	3835	64.40%	9.56%	40.41
21	HET	呼和浩特白塔国际机场	5725	64.12%	0.40%	39.79
22	WNZ	温州龙湾国际机场	3705	63.60%	26.05%	36.08
23	ZUH	珠海金湾国际机场	4066	62.18%	8.00%	44.65
24	XMN	厦门高崎国际机场	8137	58.02%	5.44%	44.61
25	NKG	南京路口国际机场	9696	57.82%	6.49%	49.55
26	TSN	天津滨海国际机场	7204	57.60%	-5.97%	49.87
27	NGB	宁波栎社国际机场	3546	55.88%	10.11%	42.51

表 9　中国大陆 TOP 级（承载客流量 1000 万–3000 万，2019 年 8 月）机场准点出发信息表

排行	IATA 编号	机场名	航班离港数	离港准点率	较去年同期增长	平均出发延误时间（分钟）
1	DLC	大连周水子国际机场	7293	76.22%	2.92%	28.53
2	HAK	海口美兰国际机场	6552	74.95%	7.92%	28.74
3	LHW	兰州中川国际机场	5423	72.91%	14.70%	31.72
4	NNG	南宁吴圩国际机场	4917	72.27%	7.80%	30.24
5	WUH	武汉天河国际机场	8730	71.92%	5.36%	31.92
6	KHN	南昌昌北国际机场	4382	71.90%	3.86%	32.99
7	TNA	济南遥墙国际机场	5920	70.50%	-1.51%	31.53
8	KWE	贵阳龙洞堡国际机场	7476	68.15%	-2.17%	35.20
9	SYX	三亚凤凰国际机场	4835	67.32%	1.13%	37.21
10	SHE	沈阳桃仙国际机场	6456	66.94%	6.39%	40.21
11	CGO	郑州新郑国际机场	9448	66.14%	7.85%	38.95
12	URC	乌鲁木齐地窝堡国际机场	8617	65.92%	-1.31%	32.99
13	CSX	长沙黄花国际机场	8895	65.71%	-0.32%	38.71
14	CGQ	长春龙嘉国际机场	4341	64.22%	9.62%	43.20
15	HRB	哈尔滨太平国际机场	6422	63.63%	-4.68%	41.48
16	HFE	合肥新桥国际机场	4123	63.41%	7.94%	43.83
17	FOC	福州长乐国际机场	4700	63.21%	13.49%	39.08
18	TAO	青岛流亭国际机场	8371	61.11%	3.67%	41.08
19	TYN	太原武宿国际机场	4665	60.88%	-7.41%	48.09
20	ZUH	珠海金湾国际机场	3987	60.39%	-4.19%	46.57
21	SJW	石家庄正定国际机场	3776	56.08%	-8.18%	59.10
22	HET	呼和浩特白塔国际机场	5603	55.85%	-10.50%	49.32
23	NKG	南京禄口国际机场	9838	53.68%	0.88%	54.82
24	XMN	厦门高崎国际机场	8288	50.92%	-8.83%	47.47
25	WNZ	温州龙湾国际机场	3857	50.89%	2.40%	49.25
26	NGB	宁波栎社国际机场	3726	49.50%	-0.89%	52.18
27	TSN	天津滨海国际机场	7357	47.67%	-25.72%	68.20

表 10　中国大陆 TOP 级（承载客流量超 1000–3000 万，2019 年 7 月）机场准点出发信息表

排行	IATA 编号	机场名	航班离港数	离港准点率	较去年同期增长	平均出发延误时间（分钟）
1	URC	乌鲁木齐地窝堡国际机场	7497	84.78%	4.83%	19.70
2	DLC	大连周水子国际机场	6563	82.86%	7.66%	20.89
3	TNA	济南遥墙国际机场	5181	76.74%	-4.38%	26.18
4	LHW	兰州中川国际机场	5040	76.67%	-3.50%	27.27
5	KHN	南昌昌北国际机场	3834	76.56%	2.20%	27.42
6	HAK	海口美兰国际机场	5864	76.35%	-1.25%	28.65
7	NNG	南宁吴圩国际机场	4493	75.21%	-0.76%	27.49
8	SYX	三亚凤凰国际机场	4383	72.72%	1.09%	31.09
9	KWE	贵阳龙洞堡国际机场	6473	72.32%	-6.19%	33.35
10	CSX	长沙黄花国际机场	7848	71.95%	-6.46%	32.49
11	WUH	武汉天河国际机场	7957	70.70%	-6.14%	31.44
12	CGO	郑州新郑国际机场	8257	69.99%	-4.83%	36.57
13	SHE	沈阳桃仙国际机场	5569	69.14%	-1.33%	36.59
14	HRB	哈尔滨太平国际机场	5855	68.78%	1.01%	34.60
15	TAO	青岛流亭国际机场	7543	68.47%	4.75%	33.30
16	HFE	合肥新桥国际机场	3552	68.45%	3.43%	38.56
17	TYN	太原武宿国际机场	4151	67.34%	-7.07%	37.08
18	SJW	石家庄正定国际机场	3470	66.93%	3.43%	38.50
19	CGQ	长春龙嘉国际机场	3867	65.73%	0.63%	42.05
20	TSN	天津滨海国际机场	6529	65.66%	2.40%	37.93
21	HET	呼和浩特白塔国际机场	4755	65.56%	-6.35%	38.51
22	FOC	福州长乐国际机场	4033	64.24%	-5.01%	37.05
23	ZUH	珠海金湾国际机场	3569	60.30%	-10.39%	50.38
24	XMN	厦门高崎国际机场	7603	60.13%	-0.82%	38.55
25	WNZ	温州龙湾国际机场	3535	54.97%	-10.16%	47.69
26	NGB	宁波栎社国际机场	3299	52.55%	-17.87%	49.48
27	NKG	南京禄口国际机场	8862	52.48%	-24.14%	55.68

表 11　中国大陆 TOP 级（承载客流量超 1000–3000 万，2019 年 6 月）机场准点出发信息表

排行	IATA 编号	机场名	航班离港数	离港准点率	较去年同期增长	平均出发延误时间（分钟）
1	UYN	榆林榆阳国际机场	1120	84.73%	-1.22%	19.16
2	KHG	喀什国际机场	961	81.58%	-8.79%	18.17
3	JHG	西双版纳国际机场	2050	81.21%	6.90%	24.57
4	LJG	丽江三义国际机场	2608	80.98%	5.97%	22.58
5	XNN	西宁曹家铺国际机场	3556	76.69%	12.39%	30.18
6	LXA	拉萨空嘎国际机场	1801	76.28%	3.96%	28.37
7	KWL	桂林两江国际机场	3154	75.86%	9.49%	25.98
8	BHY	北海国际机场	795	74.72%	24.98%	27.96
9	DYG	张家界荷花国际机场	1385	74.58%	10.26%	31.73
10	ZHA	湛江国际机场	1135	74.43%	10.61%	27.34
11	INC	银川河东国际机场	4109	73.48%	11.32%	32.66
12	HLD	呼伦贝尔东山国际机场	1641	71.87%	-3.01%	33.08
13	YCU	运城关公国际机场	1030	70.26%	7.00%	32.36
14	YIH	宜昌三峡国际机场	1267	70.16%	17.53%	34.94
15	BAV	包头东河国际机场	883	70.02%	22.01%	31.84
16	DSN	鄂尔多斯伊金霍洛国际机场	1173	69.28%	2.83%	35.60
17	ZYI	遵义新洲国际机场	903	68.29%	33.89%	35.34
18	YNT	烟台蓬莱国际机场	3913	64.88%	21.46%	38.40
19	WEH	威海国际机场	1141	64.77%	1.16%	38.23
20	SWA	揭阳潮汕国际机场	2512	64.17%	6.78%	33.82
21	XUZ	徐州观音国际机场	1049	61.99%	31.13%	43.91
22	JJN	泉州晋江国际机场	2519	60.54%	16.25%	39.32
23	YTY	扬州泰州国际机场	1064	58.83%	45.69%	48.34
24	MIG	绵阳南郊国际机场	1533	58.25%	25.60%	46.39
25	LYI	临沂沭埠岭国际机场	904	57.78%	-6.56%	46.60
26	NTG	南通新东国际机场	1313	50.15%	19.69%	55.64
27	NAY	北京南苑国际机场	1997	42.94%	-34.11%	61.87
28	WUX	苏南硕放国际机场	2574	42.78%	1.61%	61.45
29	CZX	常州奔牛国际机场	1388	37.14%	14.98%	64.36

表 12　中国大陆 TOP 级（承载客流量 200 万–1000 万，2019 年 8 月）机场准点出发信息表

排行	IATA 编号	机场名	航班离港数	离港准点率	较去年同期增长	平均出发延误时间（分钟）
1	UYN	榆林榆阳国际机场	1106	83.82%	-5.71%	18.66
2	LXA	拉萨空嘎国际机场	1846	82.18%	15.50%	22.98
3	KHG	喀什国际机场	909	81.94%	4.08%	18.68
4	JHG	西双版纳国际机场	2063	81.52%	0.43%	23.27
5	LJG	丽江三义国际机场	2601	78.16%	-1.65%	26.36
6	XNN	西宁曹家铺国际机场	3477	73.71%	3.61%	32.94
7	INC	银川河东国际机场	4129	73.14%	6.10%	32.56
8	DSN	鄂尔多斯伊金霍洛国际机场	1152	71.88%	7.92%	37.43
9	DYG	张家界荷花国际机场	1358	71.31%	2.51%	35.75
10	KWL	桂林两江国际机场	3204	70.48%	1.64%	34.16
11	BHY	北海国际机场	820	70.00%	-0.97%	28.74
12	ZHA	湛江国际机场	1144	69.67%	-1.31%	29.99
13	YIH	宜昌三峡国际机场	1256	69.25%	14.47%	33.27
14	HLD	呼伦贝尔东山国际机场	1503	68.44%	-12.54%	38.05
15	YCU	运城关公国际机场	1004	64.84%	-2.55%	43.71
16	WEH	威海国际机场	1166	64.75%	3.30%	35.86
17	ZYI	遵义新洲国际机场	905	64.09%	-1.02%	39.48
18	BAV	包头东河国际机场	894	63.04%	6.58%	39.73
19	SWA	揭阳潮汕国际机场	2579	61.38%	5.38%	36.12
20	LYI	临沂沭埠岭国际机场	909	60.99%	0.38%	44.56
21	YNT	烟台蓬莱国际机场	3947	55.09%	-3.40%	48.04
22	XUZ	徐州观音国际机场	1073	55.04%	12.16%	51.97
23	MIG	绵阳南郊国际机场	1549	52.91%	30.72%	49.03
24	YTY	扬州泰州国际机场	1027	51.90%	15.74%	56.66
25	JJN	泉州晋江国际机场	2658	49.36%	-8.88%	49.64
26	NAY	北京南苑国际机场	1942	44.85%	-27.72%	65.09
27	NTG	南通新东国际机场	1388	44.34%	5.70%	60.69
28	WUX	苏南硕放国际机场	2616	37.73%	-10.95%	68.30
29	CZX	常州奔牛国际机场	1393	33.05%	-8.01%	66.71

表 13　中国大陆 TOP 级（承载客流量 200 万–1000 万，2019 年 7 月）机场准点出发信息表

排行	IATA 编号	机场名	航班离港数	离港准点率	较去年同期增长	平均出发延误时间（分钟）
1	UYN	榆林榆阳国际机场	1015	91.82%	-0.29%	10.92
2	LXA	拉萨空嘎国际机场	1710	86.58%	8.79%	17.20
3	KHG	喀什国际机场	835	84.55%	3.51%	19.25
4	HLD	呼伦贝尔东山国际机场	918	81.25%	-5.34%	20.72
5	JHG	西双版纳国际机场	1723	81.05%	-6.83%	24.85
6	INC	银川河东国际机场	3617	80.32%	-1.18%	22.47
7	LJG	丽江三义国际机场	2310	80.30%	-4.87%	23.24
8	DSN	鄂尔多斯伊金霍洛国际机场	997	79.88%	1.75%	20.95
9	XNN	西宁曹家铺国际机场	2972	77.26%	-6.60%	26.08
10	KWL	桂林两江国际机场	2772	75.08%	-4.18%	28.87
11	BAV	包头东河国际机场	707	74.22%	5.94%	29.70
12	YIH	宜昌三峡国际机场	1143	72.18%	-6.32%	26.30
13	BHY	北海国际机场	768	72.06%	-4.97%	31.03
14	DYG	张家界荷花国际机场	1269	71.79%	-9.44%	31.73
15	ZHA	湛江国际机场	1071	70.67%	-6.00%	31.84
16	WEH	威海国际机场	1093	69.90%	12.14%	29.30
17	ZYI	遵义新洲国际机场	883	66.44%	-5.52%	41.08
18	SWA	揭阳潮汕国际机场	2163	66.42%	-5.94%	31.75
19	LYI	临沂沭埠岭国际机场	895	63.55%	-1.06%	38.95
20	YCU	运城关公国际机场	933	62.38%	-16.94%	41.74
21	YNT	烟台蓬莱国际机场	3432	60.86%	13.66%	39.08
22	NAY	北京南苑国际机场	1854	59.71%	-2.27%	41.00
23	JJN	泉州晋江国际机场	2577	59.46%	4.86%	39.57
24	XUZ	徐州观音国际机场	1021	59.16%	-5.74%	46.93
25	MIG	绵阳南郊国际机场	1465	51.40%	27.43%	48.46
26	NTG	南通新东国际机场	1227	49.80%	2.57%	54.44
27	YTY	扬州泰州国际机场	956	48.12%	-15.90%	63.51
28	WUX	苏南硕放国际机场	2337	36.55%	-35.41%	73.93
29	CZX	常州奔牛国际机场	1274	35.83%	-27.29%	72.51

表 14　中国大陆 TOP 级（承载客流量 200 万–1000 万，2019 年 6 月）机场准点出发信息表

B 蒙特卡洛模拟

主文件 (main.py)

```
# -*- encode: utf-8 -*-
from random import random, choice, choices
from tqdm import tqdm

from models import Context, Aircraft, Airport, City, Time
from data import context, cities, airline, t
from settings import randbool, randgauss, path_number, aircraft_number, people_number_ratio


population = {}
for city in cities:
    for airport in city.airports:
        population[airport] = [0] * t.DAY*t.HOUR*t.MINUTE


for i in tqdm(range(t.DAY*t.HOUR*t.MINUTE), ascii=True):
# for i in range(t.DAY*t.HOUR*t.MINUTE):
    t.elapse()

    paths = choices(list(airline), k=path_number(t.is_day()))
    for path in paths:
        aircrafts = []

        if randbool(): path=path[::-1]
        number = aircraft_number(t.is_day())
        path_ = choice(path[0].airports), choice(path[1].airports)

        if (len(list(path_[0].filter(t.is_active))) >= number) \
            and (path_[1].capacity-path_[1].aircraft_number > number) \
            and number != 0:
            times = path_[0].aircraft_leave_for_times(path_[1], number)
            for key, val in times.items():
                aircrafts.append(key)
                t.sleep(key, val)

        for aircraft in aircrafts:
            population[path_[1]][i] = round(people_number_ratio() * aircraft.capacity)
```

配置文件（settings.py）

```
# -*- encode: utf-8 -*-
from decimal import Decimal
from typing import Callable
from random import random, randint, choice, choices, gauss


def minmax(min_:[int, float], value:[int, float], max_:[int, float]) -> [int, float]:
    '''Specify the range.'''
    if min_ > value:
        return min_
    elif value > max_:
        return max_
    else:
        return value

# ======================== models.py ========================
def longitude() -> Decimal:
    '''纬度'''
    a = 3. + 51/60
    b = 53. + 33/60
    return Decimal(random_(a, b))

def latitude() -> Decimal:
    '''经度'''
    a = 73. + 33/60
    b = 135. + 5/60
    return Decimal(random_(a, b))

def random_(a:float, b:float) -> float:
    '''Random from `a` to `b`.
    '''
    return (b-a)*random() + a

with open('region.tsv', 'r') as f:
    regions = [line.split('\t') for line in f.readlines()]
def latlng() -> Decimal:
    '''经度, 纬度'''
    while True:
        city, lat, lng = choice(regions)
        if city.endswith('市') or '香港' in city or '澳门' in city or '台湾' in city:
            break
    return Decimal(lat), Decimal(lng)


```

```
language = 'zh'

translation = {
    'airport': {
        'zh': '机场',
        'en': ' airport '
    },
    'longitude': {
        'zh': longitude,
    },
    'latitude': {
        'zh': latitude,
    },
    'latlng':{
        'zh': latlng,
    },
    'location': {
        'zh': (35.7, 104.3),
    }
}


# ========================= map.py =========================
time_limit = 30 * 60
distance_to_weights = lambda x: x ** .5


# ========================= data.py =========================
def airport_capacity() -> Callable:
    weight = (10, 27, 29)
    capacity = (gauss(178, 21), gauss(55, 6), gauss(30, 2))

    func, = choices(capacity, weights=weight)
    return max(0, round(func))


data_file = 'data.pickle'

city_number = 45

city_capacity = lambda: max(1, round(gauss(5, 1)))
aircraft_capacity = lambda: max(1, round(gauss(150, 30)))
aircraft_speed = lambda: max(1, gauss(500, 50))
aircraft_capacity_ratio = lambda: minmax(0, gauss(0.5, 0.1), 1)
people_number_ratio = lambda: minmax(0, gauss(0.7, 0.1), 1)
```

```
90
91
92  # ======================== map.py ========================
93  html_file = 'world_map_airports.html'
94
95
96  # ======================== main.py ========================
97  def randbool() -> bool:
98      '''Random boolean.
99      '''
100     return bool(randint(0, 1))
101
102 def randgauss(mu:float, sigma:float) -> float:
103     '''Random gauss distribution.
104     '''
105     return minmax(0.0, gauss(mu, sigma), 1.0)
106
107 def path_number(is_day:bool=True) -> int:
108     '''Return path number with different `is_day`.
109     '''
110     mu, sigma = (2, 1) if is_day else (1, 1)
111     return max(0, round(gauss(mu, sigma)))
112
113 def aircraft_number(is_day:bool=True) -> int:
114     '''Return aircraft number with different `is_day`.
115     '''
116     mu, sigma = (2, 1) if is_day else (1, 1)
117     return max(0, round(gauss(mu, sigma)))
```

根据数据库绘制地图（map.py）

```
1  # -*- encode: utf-8 -*-
2  # https://python-visualization.github.io/folium/quickstart.html
3  import folium
4  from folium import plugins
5
6  from data import cities, airline
7  from settings import html_file, translation, language
8
9
10 m = folium.Map(location=translation['location'][language],
11     zoom_start=4,
12     tiles='Stamen Terrain',)
13
```

```
# City -> Airport -> Aircraft
for city in cities:
    folium.Marker(city.coordinate[::-1],
        popup='<i>{}</i>'.format(city.airports),
        tooltip=city.name
    ).add_to(m)
    for airport in city.airports:
        folium.Marker(airport.coordinate[::-1],
            popup='<i>{}</i>'.format(airport),
            tooltip=airport.name,
            icon=folium.Icon(color='red', icon='info-sign')
        ).add_to(m)

# Airline
attr = {
    'text': '\u2708' + ' '*16,
    'repeat': True,
    'offset': 8,
    'font_size': 8,
}

for path in airline:
    line = folium.PolyLine(
        [path[i].coordinate[::-1] for i in range(len(path))],
        weight=0.5,
    ).add_to(m)

    plugins.PolyLineTextPath(line, **attr).add_to(m)

m.save(html_file)
```

持久化数据（data.py）

```
# -*- encode: utf-8 -*-
# cities -> airports -> aircrafts
import dill
import os
from typing import Tuple

from models import Context, Aircraft, Airport, City, Time
from model_airline import Airline, Symmetric
from settings import data_file, city_number, city_capacity, airport_capacity, aircraft_capacity,
    aircraft_speed, aircraft_capacity_ratio, people_number_ratio, \
    time_limit, distance_to_weights
```

```
11
12
13  def read_data(file:str) -> Tuple[Context, City, Time, Airline]:
14      '''Read data from 'file'.
15
16      Arguments:
17      ==========
18          file : String
19
20      Returns:
21      ========
22          Tuple[Context, City, Time, Airline]
23      '''
24      with open(file, 'rb') as f:
25          return dill.load(f)
26
27  def write_data(file:str, context:Context, cities:City, airline:Airline, t:Time):
28      '''Write date to 'file'.
29
30      Arguments:
31      ==========
32          file : String
33          context: Context
34          cities : City
35          airline : Airline
36          t: Time
37
38      Returns:
39      ========
40          None
41      '''
42      with open(file, 'wb') as f:
43          dill.dump((context, cities, airline, t), f)
44
45  def drop_data(file:str):
46      '''Drop data from 'file'.
47
48      Arguments:
49      ==========
50          file : String
51
52      Returns:
53      ========
54          None
55      '''
```

```
    os.remove(file)


# Cache data
if os.path.exists(data_file):
    context, cities, airline, t = read_data(data_file)
else:
    context = Context()
    t = Time()
    cities = [City(city_capacity()) for i in range(city_number)]

    for city in cities:
        city.init_airport(airport_capacity, aircraft_capacity, aircraft_speed,
                          aircraft_capacity_ratio, people_number_ratio)

    airline = Airline(cities, time_limit, distance_to_weights)

    write_data(data_file, context, cities, airline, t)
```

数据库建模---航空数据结构（models.py）

```
# -*- encode: utf-8 -*-
from decimal import Decimal
from faker import Faker
from math import radians, cos, sin, asin, sqrt
from typing import Callable, Iterable, List, Tuple, Dict

from settings import language, translation


faker = Faker(language)
faker.latlng = translation['latlng'][language]


# Phantom class
class Context: pass
class Aircraft: pass
class Airport: pass
class City: pass


class Context:
    def __init__(self):
        ''' Initialize Context.
```

```
24
25          Arguments:
26          ==========
27              None
28
29          Returns:
30          ========
31              None
32          '''
33          self.dict = dict()
34
35      def __getitem__(self, key:object) -> object:
36          '''Return self[key].
37          '''
38          return self.dict.get(key)
39
40      def __setitem__(self, key:object, value:object):
41          '''Set self[key] to value.
42          '''
43          self.dict[key] = value
44
45      def __repr__(self) -> str:
46          '''Return repr(self).
47          '''
48          def not_start_end_with_(string:str, fix:str='_') -> bool:
49              return not (string.startswith(fix) or string.endswith(fix))
50          return str(list(filter(not_start_end_with_, dir(self))))
51
52
53  class Aircraft:
54      def __init__(self, capacity:int, speed:int):
55          ''' Initialize Aircraft.
56
57          Arguments:
58          ==========
59              capacity: Integer
60              speed: Integer
61
62          Returns:
63          ========
64              None
65          '''
66          self.capacity = capacity
67          self.people_number = 0
68          self.speed = speed
```

```
69
70          self.name = faker.numerify('#'*9)
71
72      def __repr__(self) -> str:
73          '''Return repr(self).
74          '''
75          return '<Aircraft {} ({}/{})>'.format(self.name, self.people_number, self.capacity)
76
77      def boarding(self, number:int=-1) -> bool:
78          '''Boarding the aircraft.
79
80          Arguments:
81          ==========
82              number: Integer
83
84          Returns:
85          ========
86              Boolean
87          '''
88          if number > self.capacity-self.people_number:
89              return False
90          elif number < 0:
91              self.people_number = self.capacity
92          else:
93              self.people_number += number
94          return True
95
96      def boarding_by_ratio(self, ratio:float) -> bool:
97          '''Board the aircraft by ratio from 0 to 1.
98
99          Arguments:
100         ==========
101             ratio: Float
102
103         Returns:
104         ========
105             Boolean
106         '''
107         if not 0<ratio<1:
108             return False
109         return self.boarding(round(ratio*self.capacity))
110
111     def disembarking(self, number:int=-1) -> bool:
112         '''Disembarking the aircraft.
113
```

```
        Arguments:
        ==========
            number: Integer

        Returns:
        ========
            Boolean
        '''
        if number > self.people_number:
            return False
        elif number < 0:
            self.people_number = 0
        else:
            self.people_number -= number
        return True


class Airport:
    def __init__(self, capacity:int):
        ''' Initialize Airport.

        Arguments:
        ==========
            capacity: Integer

        Returns:
        ========
            None
        '''
        if isinstance(capacity, Callable):
            print(capacity)
        self.capacity = capacity
        self.aircrafts = [None] * self.capacity
        self.aircraft_number = 0

        self.name = faker.city_name() + translation['airport'][language]

    def __repr__(self) -> str:
        '''Return repr(self).
        '''
        return '<Airport: {} ({}/{})>'.format(self.name, self.aircraft_number, self.capacity)

    def __iter__(self) -> iter:
        '''Return iter(self).
        '''
```

```
159         for aircraft in self.aircrafts:
160             if aircraft is not None:
161                 yield aircraft
162 
163     def init_coordinate(self, coordinate:Tuple[Decimal]):
164         ''' Initialize the coordinate of Airport.
165 
166         Arguments:
167         ==========
168             coordinate: Tuple[Decimal, Decimal]
169 
170         Returns:
171         ========
172             None
173         '''
174         assert len(coordinate)==2, 'Length Error'
175         latitude = faker.coordinate(coordinate[0])
176         longitude = faker.coordinate(coordinate[1])
177         self.coordinate = (latitude, longitude)
178 
179     def init_aircraft(self, aircraft_capacity:Callable, aircraft_speed:Callable,
            aircraft_capacity_ratio:Callable, people_number_ratio:Callable):
180         ''' Initialize the aircraft of Airport.
181 
182         Arguments:
183         ==========
184             aircraft_capacity: Callable
185             aircraft_speed: Callable
186             aircraft_capacity_ratio: Callable
187             people_number_ratio: Callable
188 
189         Returns:
190         ========
191             None
192         '''
193         assert isinstance(aircraft_capacity, Callable), 'Type Error'
194         assert isinstance(aircraft_speed, Callable), 'Type Error'
195         assert isinstance(aircraft_capacity_ratio, Callable), 'Type Error'
196         assert isinstance(people_number_ratio, Callable), 'Type Error'
197 
198         for idx in range(round(aircraft_capacity_ratio()*self.capacity)):
199             self.aircrafts[idx] = Aircraft(aircraft_capacity(), aircraft_speed())
200             self.aircrafts[idx].boarding_by_ratio(people_number_ratio())
201             self.aircraft_number += 1
202 
```

```
203     def aircraft_leave_for_times(self, airport:Airport, number:int=1) -> Dict[Aircraft, int]:
204         '''Calculate the times of the aircrafts leaving for `airport`.
205 
206         Arguments:
207         ==========
208             airport: Airport
209             number: Integer
210 
211         Returns:
212         ========
213             Dict[Aircraft, Integer]
214         '''
215         distance = self.distance_from(airport)
216         out = self.aircraft_out(number)
217         if airport.aircraft_in(out):
218             times = [round(distance/o.speed*Time.HOUR*Time.MINUTE) for o in out]
219             return dict(zip(out, times))
220         return {}
221 
222     def aircraft_out(self, number:int=1) -> List[Aircraft]:
223         '''Pop the aircrafts out from Airport.
224 
225         Arguments:
226         ==========
227             number: Integer
228 
229         Returns:
230         ========
231             List[Aircraft, ...]
232         '''
233         if number > self.aircraft_number:
234             return []
235         self.aircraft_number -= number
236         result = [None] * number
237         ith = 0
238         for idx, aircraft in enumerate(self.aircrafts):
239             if aircraft is not None:
240                 result[ith] = aircraft
241                 ith += 1
242                 self.aircrafts[idx] = None
243                 if ith >= number:
244                     break
245         return result
246 
247     def aircraft_in(self, aircrafts:List[Aircraft]) -> bool:
```

```
248         '''Put the aircrafts into Airport.
249
250         Arguments:
251         ==========
252             aircrafts: List[Aircraft, ...]
253
254         Returns:
255         ========
256             Boolean
257         '''
258         length = len(aircrafts)
259         if length > self.capacity-self.aircraft_number:
260             return False
261         self.aircraft_number += length
262         ith = 0
263         for idx, aircraft in enumerate(self.aircrafts):
264             if aircraft is None:
265                 self.aircrafts[idx] = aircrafts[ith]
266                 ith += 1
267                 if ith >= length:
268                     break
269         return True
270
271     def getattr_from_aircraft(self, key:str) -> list:
272         '''Get the attribution from all of the aircrafts.
273
274         Arguments:
275         ==========
276             key: String
277
278         Returns:
279         ========
280             List[Object, ...]
281         '''
282         attrs = (getattr(aircraft, key, None) for aircraft in self.aircrafts)
283         return list(filter(lambda x: x is not None, attrs))
284
285     def distance_from(self, place:[Airport, City]) -> float:
286         '''Calculate the distance between `self` and `place` (unit: km).
287
288         Arguments:
289         ==========
290             place: [Airport, City]
291
292         Returns:
```

```
        ========
            Float
        '''
        lat1, lon1 = map(radians, self.coordinate)
        lat2, lon2 = map(radians, place.coordinate)

        dlat, dlon = lat1-lat2, lon1-lon2
        a = sin(dlat/2)**2 + cos(lat1)*cos(lat2)*sin(dlon/2)**2
        return 2*asin(sqrt(a)) * 6371

    def filter(self, condition:Callable) -> iter:
        ''' Filter iteration. Or filter(condition, self).
        '''
        for aircraft in self:
            if condition(aircraft):
                yield aircraft


class City:
    def __init__(self, capacity:int):
        ''' Initialize the City.

        Arguments:
        ==========
            capacity: Integer

        Returns:
        ========
            None
        '''
        self.capacity = capacity
        self.airports = [None] * self.capacity

        self.name = faker.city()
        self.coordinate = faker.latlng()

    def __repr__(self) -> str:
        '''Return repr(self).
        '''
        return '<City: {} ({})>'.format(self.name, self.capacity)

    def init_airport(self, airport_capacity:[Iterable, Callable], aircraft_capacity:Callable,
        aircraft_speed:Callable, aircraft_capacity_ratio:Callable, people_number_ratio:Callable):
        ''' Initialize the airports of City.

```

```
337         Arguments:
338         ==========
339             airport_capacity: [Iterable, Callable]
340             aircraft_capacity: Callable
341             aircraft_speed: Callable
342             aircraft_capacity_ratio: Callable
343             people_number_ratio: Callable
344 
345         Returns:
346         ========
347             None
348         '''
349         assert isinstance(airport_capacity, (Iterable, Callable)), 'Type Error'
350         assert isinstance(aircraft_capacity, Callable), 'Type Error'
351         assert isinstance(aircraft_speed, Callable), 'Type Error'
352         assert isinstance(aircraft_capacity_ratio, Callable), 'Type Error'
353         assert isinstance(people_number_ratio, Callable), 'Type Error'
354         if isinstance(airport_capacity, Iterable):
355             assert len(airport_capacity)==self.capacity, 'Length Error'
356 
357         if isinstance(airport_capacity, Callable):
358             airport_capacity = [airport_capacity() for idx in range(self.capacity)]
359         airport_capacity = iter(airport_capacity)
360         for idx in range(self.capacity):
361             self.airports[idx] = Airport(next(airport_capacity))
362             self.airports[idx].init_coordinate(self.coordinate)
363             self.airports[idx].init_aircraft(aircraft_capacity, aircraft_speed,
                    aircraft_capacity_ratio, people_number_ratio)
364 
365     def getattr_from_airport(self, key:str) -> list:
366         '''Get the attribution from all of the airports.
367 
368         Arguments:
369         ==========
370             key: String
371 
372         Returns:
373         ========
374             List[Object, ...]
375         '''
376         attrs = (getattr(airport, key, None) for airport in self.airports)
377         return list(filter(lambda x: x is not None, attrs))
378 
379     def distance_from(self, place:[Airport, City]) -> float:
380         '''Calculate the distance between `self` and `place` (unit: km).
```

```
381
382         Arguments:
383         ==========
384             place: [Airport, City]
385
386         Returns:
387         ========
388             Float
389         '''
390         lat1, lon1 = map(radians, self.coordinate)
391         lat2, lon2 = map(radians, place.coordinate)
392
393         dlat, dlon = lat1-lat2, lon1-lon2
394         a = sin(dlat/2)**2 + cos(lat1)*cos(lat2)*sin(dlon/2)**2
395         return 2*asin(sqrt(a)) * 6371
396
397
398 class Time:
399     MINUTE, HOUR, DAY, MONTH, YEAR = 60, 60, 24, 30, 12
400
401     def __init__(self):
402         ''' Initialize the Time.
403
404         Arguments:
405         ==========
406             None
407
408         Returns:
409         ========
410             None
411         '''
412         self.now = 0
413         self.day = (6, 0, 0), (18, 0, 0)
414         self.dict = dict()
415
416     def __repr__(self):
417         ''' Return repr(self)
418         '''
419         return '{}-{:0>2d}-{:0>2d} {:0>2d}:{:0>2d}:{:0>2d}'.format(*self.ymdhms())
420
421     def elapse(self, second:[int, Tuple[int]]=1):
422         ''' Time elapsed.
423
424         Arguments:
425         ==========
```

```
426             second: [Integer, Tuple[Integer, ...]]
427
428         Returns:
429         ========
430             None
431         '''
432         del_keys = []
433         if not isinstance(second, int):
434             second = self.to_second(second)
435         self.now += second
436         for key in self.dict:
437             self.dict[key][0] -= second
438             if self.dict[key][0] < 0:
439                 if isinstance(self.dict[key][1], Callable):
440                     self.dict[key][1]()
441                 del_keys.append(key)
442         for key in del_keys:
443             del self.dict[key]
444
445     def year_month_day_hour_minute_second(self) -> Tuple[int]:
446         '''Convert the second (self.now) to (year, month, day, hour, minute, second).
447
448         Arguments:
449         ==========
450             None
451
452         Returns:
453         ========
454             Tuple[Integer, ...]
455         '''
456         now = self.now
457         second = now % self.MINUTE
458         now //= self.MINUTE
459         minute = now % self.HOUR
460         now //= self.HOUR
461         hour = now % self.DAY
462         now //= self.DAY
463         day = now % self.MONTH
464         now //= self.MONTH
465         month = now % self.YEAR
466         now //= self.YEAR
467         year = now
468         return year, month, day, hour, minute, second
469
470     def ymdhms(self) -> Tuple[int]:
```

```
        '''Return self.year_month_day_hour_minute_second.
        '''
        return self.year_month_day_hour_minute_second()

    def to_second(self, value:Tuple[int]) -> int:
        '''Convert the (year, ..., second) to second.

        Arguments:
        ==========
            value: Tuple[Integer]

        Returns:
        ========
            Integer
        '''
        length = len(value)
        if length > 6:
            raise AssertionError('Length Error')
        elif length < 6:
            return self.to_second((0,)*(6-length)+value)

        para = 1
        paras = [self.MINUTE, self.HOUR, self.DAY, self.MONTH, self.YEAR]
        now = para * value[-1]
        for idx in range(len(paras)):
            para *= paras[idx]
            now += para * value[-idx-2]
        return now

    def is_day(self) -> bool:
        '''Whether it is daytime.

        Arguments:
        ==========
            None

        Returns:
        ========
            None
        '''
        _, _, _, h, m, s = self.ymdhms()
        now = self.to_second((h, m, s))
        day = self.to_second(self.day[0]), self.to_second(self.day[1])
        return day[0] < now < day[1]

```

```
516     def is_night(self) -> bool:
517         '''Whether it is night.
518 
519         Arguments:
520         ==========
521             None
522 
523         Returns:
524         ========
525             None
526         '''
527         return not self.is_day()
528 
529     def set_day(self, day:Tuple[Tuple]):
530         '''Ste the rule of judging the daytime.
531 
532         Arguments:
533         ==========
534             day: Tuple[Tuple]
535 
536         Returns:
537         ========
538             None
539         '''
540         assert len(day)==2, 'Length Error'
541         assert isinstance(day[0], Tuple), 'Type Error'
542         assert isinstance(day[1], Tuple), 'Type Error'
543 
544         self.day = day
545 
546     def sleep(self, key:object, second:int=1, action:Callable=lambda:None):
547         '''`key` sleeps for `second`.
548 
549         Arguments:
550         ==========
551             key: Object
552             second: Integer
553             action: Callable
554 
555         Returns:
556         ========
557             None
558         '''
559         self.dict.setdefault(key, [0, None])
560         self.dict[key][0] += second
```

```
561             self.dict[key][1] = action
562
563     def is_active(self, key:object) -> bool:
564         ''' Whether 'key' is active.
565
566         Arguments:
567         ==========
568             key: Object
569
570         Returns:
571         ========
572             Boolean
573         '''
574         return key not in self.dict
```

数据库建模---航线数据结构 (model-airline.py)

```
1   # -*- encode: utf-8 -*-
2   from typing import List, Tuple, Callable
3   import numpy as np
4
5   from models import Aircraft, Airport, City, Time
6   from settings import time_limit, time_limit, distance_to_weights
7
8
9   # Phantom class
10  class Airline: pass
11  class Symmetric: pass
12
13
14  class Airline:
15      def __init__(self, cities:List[City], time_limit:int, distance_to_weights:Callable):
16          ''' Initialize Airline.
17
18          Arguments:
19          ==========
20              cities : List[City, ...]
21              time_limit: Integer (second)
22              distance_to_weights: Callable
23
24          Returns:
25          ========
26              None
27          '''
```

```
28          self.cities = cities
29          self.city_number = len(cities)
30          self.time_limit = time_limit
31          self.distance_to_weights = distance_to_weights
32          self.distances = Symmetric(self.city_number)
33
34          for ith in range(self.city_number):
35              self.distances[ith, ith] = None
36              for jth in range(ith+1, self.city_number):
37                  self.distances[ith, jth] = self.distance_from_ith_jth(ith, jth)
38
39      def __repr__(self) -> str:
40          '''Return repr(self).
41          '''
42          return '<Airline with {} path(s)>'.format(len(self))
43
44      def __len__(self) -> int:
45          '''Return len(self).
46          '''
47          return self.distances.nonnone()
48
49      def __iter__(self) -> iter:
50          '''Return iter(self).
51          '''
52          for ith in range(len(self.distances)):
53              for jth in range(ith+1, len(self.distances)):
54                  if self.distances[ith, jth] is not None:
55                      yield self.cities[ith], self.cities[jth]
56
57      def remove_path_by_time_limit(self):
58          '''Remove path by `time_limit`.
59
60          Arguments:
61          ==========
62              None
63
64          Returns:
65          ========
66              None
67          '''
68          for ith in range(self.city_number):
69              for jth in range(ith+1, self.city_number):
70                  distance = self.distance_from_ith_jth(ith, jth)
71                  for airport in self.cities[ith].airports:
72                      for aircraft in airport.aircrafts:
```

```
                        if aircraft is None:
                            break
                        time = distance/aircraft.speed*Time.MINUTE*Time.HOUR
                        if time < self.time_limit:
                            self.distances[ith, jth] = None
                            break
                    if self.distances[ith, jth] is None:
                        break

    def normalize_distances_by_distance_to_weights(self):
        '''Normalize `self.distances` by `distance_to_weights`.

        Arguments:
        ==========
            None

        Returns:
        ========
            None
        '''
        for ith in range(self.city_number):
            for jth in range(ith+1, self.city_number):
                if self.distances[ith, jth] is not None:
                    self.distances[ith, jth] = self.distance_to_weights(self.distances[ith, jth])


    def distance_from_ith_jth(self, ith:int, jth:int) -> float:
        '''Calculate the distance from `ith` city and `jth` city.

        Arguments:
        ==========
            ith: Integer
            jth: Integer

        Returns:
        ========
            Float
        '''
        assert all((0<=ith,jth<self.city_number)), 'Length Error'

        return self.cities[ith].distance_from(self.cities[jth])


class Symmetric:
    def __init__(self, length:int):
```

```
118         '''Symmetric matrix with initial value zero.
119
120         Arguments:
121         ==========
122             length: Integer
123
124         Returns:
125         ========
126             None
127         '''
128         self.length = length
129         self.value = [[0.0]*(i+1) for i in range(length)]
130
131     def __repr__(self) -> str:
132         '''Return repr(self).
133         '''
134         return '<SymmetricMatrix: ({0}x{0})>'.format(self.length)
135
136     def __iter__(self) -> iter:
137         '''Return iter(self).
138         '''
139         for row in self.value:
140             for ele in row:
141                 yield ele
142
143     def __len__(self) -> int:
144         '''Return len(self).
145         '''
146         return self.length
147
148     def __getitem__(self, index:[int, Tuple[int]]) -> float:
149         '''Return self[index].
150         '''
151         assert isinstance(index, (int, Tuple)), 'Type Error'
152         if isinstance(index, int):
153             return self.__getitem__(self.idx_1d_to_2d(index))
154         assert len(index)==2, 'Type Error'
155
156         ith, jth = index
157         if ith < jth:
158             return self.__getitem__((jth, ith))
159         return self.value[ith][jth]
160
161     def __setitem__(self, index:[int, Tuple[int]], value:float):
162         '''Set self[index] to value.
```

```
        '''
        assert isinstance(index, (int, Tuple)), 'Type Error'
        if isinstance(index, int):
            return self.__setitem__(self.idx_1d_to_2d(index), value)
        assert len(index)==2, 'Length Error'

        ith, jth = index
        if ith < jth:
            return self.__setitem__((jth, ith), value)
        self.value[ith][jth] = value

    @property
    def T(self) -> Airline:
        '''Transpose.'''
        return self

    @property
    def shape(self) -> Tuple[int]:
        '''Shape of 'self'.'''
        return self.length, self.length

    def nonnone(self) -> int:
        '''Count the number of non-None elements.
        '''
        count = 0
        for value in self:
            if value is not None:
                count += 1
        return count

    def densify(self) -> np.matrix:
        '''Densify the symmetric matrix.

        Arguments:
        ==========
            None

        Returns:
        ========
            numpy.matrix
        '''
        result = np.zeros((self.length, ) * 2)
        for ith in range(self.length):
            result[ith, ith] = self.value[ith][ith]
            for jth in range(ith+1, self.length):
```

```
                result[ith, jth] = self.value[jth][ith]
                result[jth, ith] = self.value[jth][ith]
        return result

    def idx_1d_to_2d(self, idx:int) -> Tuple[int]:
        '''Convert 1d 'idx' to 2d.

        Arguments:
        ==========
            idx: Integer

        Returns:
        ========
            Tuple[Integer, Integer]
        '''
        row = idx % self.length
        col = idx // self.length
        return row, col



if __name__ == '__main__':
    from data import cities

    airline = Airline(cities, time_limit, distance_to_weights)
```