基于收益最大化的司机决策和效益最优的上车方案研究

摘要

出租车到达机场后，可以选择前往“蓄车池”载客（方案 A），也可以选择空车返回（方案 B）。本文从供应和需求入手，分析了机场出租车载客决策的影响因素及机理，建立了基于收益最大化的司机决策模型并实证分析；利用元胞自动机对两条并行车道的“乘客区”的上车方式进行了仿真，推荐了效益最优的上车方案，并给出了基于收益均衡的短途订单司机的再载客优先分配方案。

第一问：首先归纳影响司机决策的 11 项因素；其次从供应和需求入手，将 11 项因素归纳为 3 个因素：“蓄车池”里已有的车辆数（确定性因素），未来一段时间抵达航班的总乘客数量（基本确定性因素），选择打车的乘客占总乘客的比例（不确定因素），其中不确定因素与司机的经验判断因子相关；然后将两种方案放置在相同的时间长度中，考察两种方案的预期收益，建立基于收益最大化的决策模型。结果表明，在两种极端情况下（见第 4.2.2 节），司机分别选择方案 A 和方案 B，对于第三种一般情况，司机决策受经验判断因子等因素影响。

第二问：以上海市为例确定了车辆的基本成本和收益信息，以虹桥机场 2019 年 9 月 14 日数据为例确定了航班数，利用 GPS 卫星数据确定当日出租车的信息并对模型进行实证分析。结果表明：（1）同一天的不同时间段，司机的决策不一样；（2）同一个时间段，具有不用经验判断因子的司机的决策也不一样。灵敏度分析表明：（1）乘客目的地与机场的距离越远，司机的收益越大，距离越近，司机的收益越小，甚至可能赔本；（2）司机的经验判断因子对结果具有一定的影响，经验越丰富的司机，做出正确选择的几率越大，收益也越多。

第三问：首先根据上车的方向性、相邻上车点的间距大小，乘客行李的多少设计出 6 种备选方案；设置不同方案的运行规则，利用元胞自动机进行仿真，系统运行 3 小时，统计不同方案在单位时间通过的乘客数（效率性），车辆的急刹车和变道频次（安全性），和实际用时与期望用时的比例（满意度），以安全性为底线，优先考虑效率性，兼顾满意度，确定了最优上车方案为**单侧乘车，相邻上车点小间距，乘客分流制**，此时共有 8 个上车点，每小时能疏散 1664.1 名乘客。

第四问：定义短途订单为此订单的收益低于“乘客区”订单的平均收益，设置类似高速公路上应急车道的“蓄车池”优先车道，并提出两种优先安排方案。方案一：区分短途订单的内部差异性，按照“先来后到”原则，对短途订单司机予以优先车道候车载客；方案二：设置以 15 分钟为周期的短途订单差异性排序，短途订单收益与平均收益差距越大，司机优先级越高，在优先车道的序号越靠前。

关键词：收益决策；经验判断因子；元胞自动机；优先级

一、 问题重述

机场陆侧交通系统是衔接机场交通与城市交通的中间环节，出租车是其中重要的衔接交通方式[1]，出租车系统在机场承担了30%至70%左右的旅客集散量[2]，出租车在提高机场服务水平方面发挥了重要作用。

对于送客到机场的出租车司机而言，他面临或者前往到达区排队等待载客返回市区（方案A），或者直接放空返回市区拉客（方案B）的两种选择。因为司机间的竞争性，旅客到达时间、选择交通方式的不确定性，和诸如节假日与工作日的差异性，会导致两种选择各有优劣。微观层面考虑出租车司机的收益，宏观层面考虑机场的乘车效率，本文主要研究如下4个问题：

1. 归纳与出租车司机决策相关的因素及其影响机理，并统筹考虑司机的收益和乘客数量变化，建立出租车司机决策模型，确定其选择策略。

2. 以上海虹桥机场及上海市出租车的相关数据，给出不同时间状态下该机场出租车司机的选择方案，并对模型进行灵敏度分析和有效性检验。

3. 针对出租车排队载客和乘客排队乘车同时存在的情况，在保证安全的前提下，确定“上车点”的个数、间距和上车方式，使得乘车效率最大化。

4. 为保障载客到短途的司机的收益，设计短途旅客司机“插队”的优先排队方案，允许他们往返进行载客，以均衡司机们的收益。

二、问题分析

2.1 问题1的分析

本问要求确定与出租车司机决策相关的因素及其影响机理，统筹考虑司机的收益和乘客数量变化，建立出租车司机决策模型，确定其选择策略。

针对等待载客的方案A和空载返回市区的方案B，司机决策的动机是个人收益的最大化。收益取决于市场的供应和需求，因此以供应和需求为核心，通过向外递进分析确定影响司机决策的因素。

接下来研究两种方案的收益。首先取相同的时间长度为基准，针对两种方案的特性，将时间长度划分为等待期、驶离机场期等不同的阶段；其次结合乘客到达的规律（如服从某种分布），分别计算不同阶段的收入和成本；然后建立以收益最大化为目标的决策模型，确定司机选择策略。

2.2 问题2的分析

本问以上海虹桥机场及上海市出租车的相关数据，给出不同时间状态下该机

场出租车司机的选择方案，并对模型进行灵敏度分析和有效性检验。

根据上海市强生出租车公司基于 GPS 的行驶数据，首先对数据进行经纬度筛选，确定上海虹桥机场在某一天的出租车行驶数据；然后结合相关文献和一些生活经验，可确定模型的相关参数；最后代入到决策模型，对司机给出决策建议。

值得强调的是，因为司机在部分因素上的认知具有个体差异性，所以即使针对同一种状况，不同司机的决策也可能是不同的。因此进一步检验相关参数（如司机的认知能力，乘客目的地的远近等）对模型的影响来对模型稳定性进行检验，并通过数值模拟来反映在不同交通情形下（高峰期和平峰期）不同特点（激进型、平衡型、保守型）司机的决策，验证模型的有效性并扩充模型的有效覆盖面。

2.3 问题 3 的分析

本问要求设置以安全为前提，以效率最大化为目标，确定双车道并行车道“上车点”的个数、间距和上车方式。

考虑到司机行为的不确定性和乘客到来规律的不确定性，采用元胞自动机进行仿真。首先根据文献和推理确定若干种可能的上车方式，确定每一种上车方式对应的元胞运行规则；然后让系统允许一定的时间，统计单位时间的载客数，急刹车和乘客平均等待时间等因素，建立评价模型，确定最优的上车方式；最后针对最优的上车方式，通过控制变量法，对比确定最优的上车点个数和间距。

2.4 问题 4 的分析

等待时间过长而行驶时间片段是司机不愿意看到的，而短途载客司机往往会遇到这一问题。本问为保障载客到短途的司机的收益，设计短途旅客司机“插队”的优先排队方案，允许他们往返进行载客，以均衡司机们的收益。

首先将指定单收益与载客平均收益进行对比，界定短途载客的概念；然后引入优先级指数的概念，针对不同的原则设计不同的优先安排方案；最后并通过对比优先安排方案前后司机单位时间收益的标准差，来验证模型的有效性。标准差越小，标准差减少得越多，说明方案发挥的均衡性越充分。

三、模型假设与符号说明

3.1 模型假设

1. 机场每天的排班表是公开可查的，排班表以天为单位基本稳定。这个假设是为了便于设定司机的判断力具有以天为单位的周期性，且符合主观认知。

2. 所有的航班都满载抵达机场，且潜在客户数就是航班的总容量。首先航

班一定不会超载，其次航班虽然不一定总是满载，但是也会有送亲友到机场的人会返回至市区，也是潜在乘客，假设两部分抵消，并为了计算上的方便性，作此假设。

4. 假设出租车的数量基本不变[3]。这一假设取自文献[3]，一方面因为出租车客运交通方式是准公共产品，市场运营通常受到政府施行的准入规制限制，以致城市出租车的市场投入数量往往在几年内都不会发生较大变化；另一方面也为了确保司机选择不同方案的合理性，因为某个方案现有的出租车（竞争对手）太多，那么另一种方案的竞争对手就少。

5. 所有司机都能清楚地知道当前一段时间内抵达的航班数量和“蓄车池”里已有的车辆数。

6. 乘客拟到达的目的地具服从以机场到城市中心距离为均值的正态分布。因为乘客到底目的地的距离具有不可预知性，本假设的随机性能使得模型具有可操作性，并更能反映、适用于现实情况。

7. 机场具有足够成熟的物联网和物理技术来判断司机的任意一单生意是否属于短途生意。

3.2 符号说明

表 1：本文所使用的主要符号（相关符号在正文中均会予以再次解释说明）

变量名	含义
t_i	在“蓄车池”的相关时期，$i=1,2,3$ 分别表示等候乘客期、驶离机场期和市区行驶期
$t^*(i)$	司机 i 能够接受的等待时间上限
$m(t)$	t 时刻“蓄车池”内待载客的车辆量
β	每辆出租车的平均载客率
$p(t)$	在时间段 $\left(t,t+t^*\right)$ 内，所有抵达机场的航班的乘客总数
$n(t)$	t 时刻“乘车区”的乘客数
$\alpha(t)$	经验判断因子，司机估计的 t 时刻乘坐出租车的乘客比例
$G_A(i)$	司机 i 选择方案 A 的收益
$G_B(i)$	司机 i 选择方案 B 的收益

四、收益最大化视角下的决策模型

4.1 影响因素的确定及影响机理分析

本文围绕供求关系进行影响因素的确定，供应指在“蓄车池”等候载客的出租车数量，反映的是司机愿意载客的程度。需求指前往“乘车区”打车的乘客数量，可假设该数量与航班总人数呈线性关系，不同的司机根据经验判断比例系数，与实际比例系数可能有一定的误差。

根据文献[4-8]，结合日常判断，本文整理出影响供应和需求的相关因素，如下图所示。

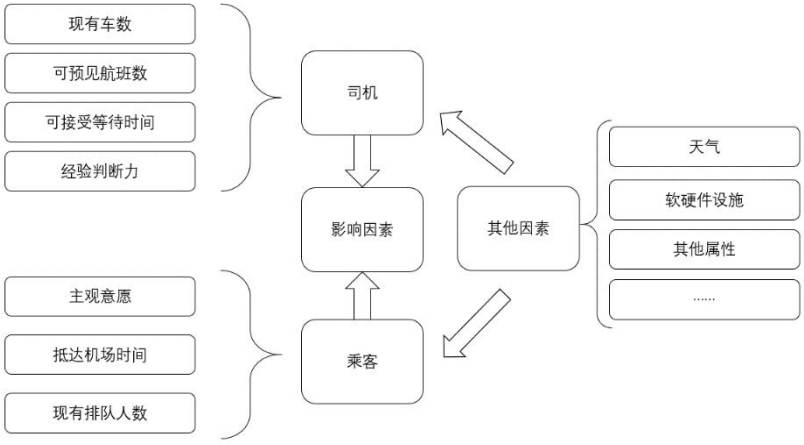

图 1 影响供应和需求的相关因素图

供应方面各因素的影响机理如下：

（1）“蓄车池”现有的车辆数。“蓄车池”现有的车辆数越大，说明出租车对乘客的供应相对充足，司机选择前往“蓄车池”载客的意愿越低。

（2）未来一段时间抵达航班的数量。未来一段抵达航班的数量越多，乘客对于出租车的需求量越大，司机选择前往“蓄车池”载客的意愿越高。

（3）司机可接受的等待时间。司机可接受的等待时间越长，其选择前往“蓄车池”载客的意愿越高。

（4）司机对于愿意乘车人数比例的判断。司机会根据个人经验来判断当时愿意乘车的人数占比，此比例越高，其选择前往“蓄车池”载客的意愿越高。

需求方面各因素的影响机理如下：

（5）乘客的主观意愿。不同乘客对于交通方式的选择不同，部分乘客会乘坐地铁等其他交通工具，使选择乘坐出租车的人数减少。

（6）乘客抵达机场的时间。若飞机抵达机场的时间为夜晚，乘客更愿意选择乘坐出租车；若飞机抵达机场的时间为市区交通高峰期，乘客更倾向地铁。

（7）“乘车区”当前的排队人数，排队人数较少，乘客愿意选择乘坐出租车

的意愿越高。

除此之外，天气、机场的软硬件服务设施（如“蓄车池”的容量，工作人员的态度）、当天的属性（节假日还是工作日等）等也都会通过影响供求关系从而影响司机的决策。

如果假设司机是个理性人，那么供求关系决定了司机的决策。实际上由于司机的经验判断力与实际可能有一定的差异，因此司机眼中的需求和实际的需求是存在差异的，因此总结上述多项因素，影响司机决策的因素主要可以归纳为三种：

第一种：当前时刻“蓄车池”里已有的车辆数，且这是一个确定性信息。

第二种：未来一段时间抵达航班的总乘客的数量，根据假设 7，这也是一个确定性信息。

第三种：当前时刻选择打车的乘客占总乘客的比例，这是一个不确定信息，与司机的个人经验判断有关。

分析认为，一切因素的变化，都可以归纳为对司机经验判断能力的变化，从而影响供需情况，影响司机的决策。

在图 1 的 10 多项因素中，“蓄车池”现有的车辆数可以归纳为第一种，未来一段时间抵达航班的数量可以归纳为第二种；其他所有因素可以归纳到第三种因素中，因为具有不确定性，这些不确定性会影响司机认知中的“有效乘客数”，尽管这个有效乘客不一定是实际的，但是在司机的认知和判断中，这就是真实有效的乘客数，司机是基于这个有效乘客数来进行决策的。

4.2 决策模型的建立

4.2.1 两种方案时长的公平化处理

上图表明，不同方案各有所长，然而时间、车费、自由等不具有可比性，因此需要将两种方案放置在同一个标准下处理。有两种处理方法：

（1）时间保持一致，考察方案收益的大小：以方案 A 完成第一个订单所花费的时间为准，将方案 B 多出来的时间纳入到考核中，看同样时长下，哪种方案收益更大。

（2）收益保持一致，考察方案耗时的长短：以方案 A 完成第一个订单的收益（收入-成本）为准，计算方案 B 得到同等收益所需要的时间，看收益相同情况下，哪种方案耗时更短。

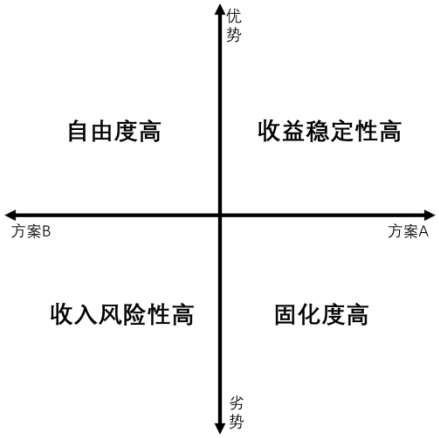

图 2 两种方案的优势劣势

相对于时间，收益需要估计的参数更多（包括单位时间行驶成本、单位时间等待成本等），因此本文采用第（1）中处理方法，示意图如下。

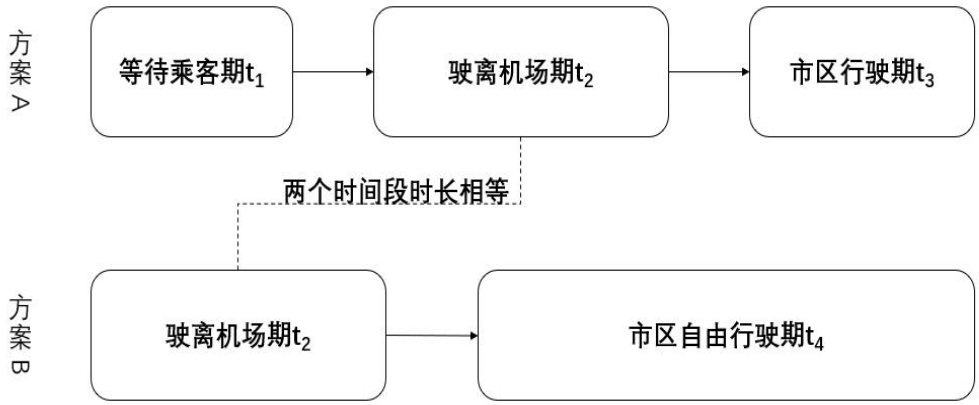

图 3：两种方案的时间对比图

如上图所示，方案 A 要经历三个阶段：

（1）等候乘客期t_1：机场出租车司机为了载客前往“蓄车区”排队，等待乘客上车所消耗的时间；

（2）驶离机场期t_2：车辆载到客后从机场到市区所要消耗的时间；

（3）市区行驶期t_3：司机到达市区后送车上乘客到达指定目的地所消耗的时间。

为了确保时间的可比性，将方案 B 的时间划分为两个阶段：

（1）驶离机场期t_2：时间数值上与方案 1 完全相同，但是唯一不同的是方案二中的车辆在返回时未乘坐乘客，司机需自行承担空载损失；

（2）市区自由行驶期$t_4 = t_1 + t_3$：车辆返回市区后在市区随意接客的时间。

4.2.2 决策模型的建立

用 $m(t)$ 表示 t 时刻“蓄车池”内待载客的车辆量，用 β 表示平均载客率，用 $\alpha(t)$ 表示 t 时刻愿意前往“乘客区”乘坐出租车的乘客占航班总乘客的比例， $\alpha(t)$ 既与时间相关，也与司机个人的经验相关。

用 $n(t)$ 表示 t 时刻“乘车区”的乘客数，用 t^* 表示司机可接受的等待时间上限，用 $p(t)$ 表示在时间段 $\left(t,t+t^*\right)$ 内，所有抵达机场的航班的乘客总数，则在时间段 $\left(t,t+t^*\right)$ 内需要在“乘车区”乘车的乘客数为 $n(t)+\alpha(t)p(t)$。

分三种情形考虑司机的决策：

情形 1：当前“乘车区”状态为有人无车，这种情况发生于第一个司机到达的时候，因为无需付出等待时间成本，此时司机应选择方案 A。

情形 2：当前“乘车区”状态为有人有车，但待载客车辆（不含自己）的预载容量大于或等于在时间段 $\left(t,t+t^*\right)$ 内需要在“乘车区”乘车的乘客数，即

$$\beta m(t) \geq n(t)+\alpha(t)p(t)\,,$$

此时司机应选择方案 B，因为在司机的等待时间上限内，前方的车辆足以带走所有的乘客。

情形 3：当前“乘车区”状态为有人有车，但待载客车辆（不含自己）的预载容量小于在时间段 $\left(t,t+t^*\right)$ 内需要在“乘车区”乘车的乘客数，即

$$\beta m(t) < n(t)+\alpha(t)p(t)\,,$$

此时司机面临方案 A 和方案 B 的取舍，需要进一步分析两种方案的收益。

如果选择方案 A，司机的**收入** F_A 为

$$F_A = f_1 + f_2 + f_3\,,$$

其中 f_1 为出租车的起步价，为定值； f_2 为路程费用，与路程成正比； f_3 为等待费用，主要指在堵车、等红绿灯时产生的费用，例如在某城市，每等候 3 分钟收费 2.8 元。

司机的成本 C_A 为

$$C_A = c_1 \times t_t + c_2 \times t_2 + c_3 \times t_3\,,$$

其中c_1表示单位时间的等待费用，c_2表示驶离机场期单位时间的平均行驶费用，c_3表示市区行驶期单位时间的平均行驶费用，因为在驶离机场期的速度大于在市区行驶的速度，所以两者的费用不同，t_1,t_2,t_3分别表示等待乘客期、驶离机场期和市区行驶期的速度。

因此司机选择方案 A 的收益

$$G_A = F_A - C_A = \sum_{i=1}^{3}\left(f_i - c_i t_i\right).$$

如果选择方案 B，司机没有确切的收入，司机在(t_1+t_3)的时长内，以一定的概率获得收入，用$\overline{f}$表示当地司机单位时间的期望薪资，则司机的预期收入

$$F_B = \overline{f} \times (t_1 + t_3)，$$

司机的成本C_B为

$$C_B = c_2 \times t_2.$$

因此司机选择方案 B 的预期收益

$$G_B = F_B - C_B = \overline{f}(t_1 + t_3) - c_2 t_2.$$

综上，司机的决策模型和选择策略为

- 若$G_A - G_B = \sum_{i=1}^{3} f_i + \left(\overline{f} - c_1\right)t_1 + \left(\overline{f} - c_3\right)t_3 > 0$，则选择方案 A；
- 若$G_A - G_B = \sum_{i=1}^{3} f_i + \left(\overline{f} - c_1\right)t_1 + \left(\overline{f} - c_3\right)t_3 < 0$，则选择方案 B；
- 若$G_A - G_B = \sum_{i=1}^{3} f_i + \left(\overline{f} - c_1\right)t_1 + \left(\overline{f} - c_3\right)t_3 = 0$，则选择方案 A 或方案 B 均可。

需要特别强调的是，方案 A 具有乘客目的地不可预知的随机性，方案 B 具有是否能成功载到客人以及空车率的不确定性，上述决策模型是具有一定的风险的，因此司机的个人经验和判断力就显得尤为重要。

五、司机决策的实证分析及灵敏度检验

5.1 数据来源及预处理

5.1.1 航班数和出租车数的确定

首先利用 GPS 卫星地图确定上海市虹桥机场的经纬度范围，再通过对强生出租车公司数据中存放经纬度的两列数据进行筛选[①]。

上海虹桥机场位于东经（131.33°，131.35°），北纬（31.17°，31.22°），筛选出此范围内的出租车数据，并认为这些出租车均位于机场内。强生巡游出租车规模约占上海市巡游出租车保有量的 25%，故将任意时刻在机场内的强生出租车数量乘以 4 倍即可估计出整个机场内的出租车数量。

根据假设 1，上海虹桥机场的航班数量及其时刻基本保持稳定，不考虑因为天气等因素导致的航班延期。根据上海机场(集团)有限公司网站[②]提供的数据，确定了 2019 年 9 月 14 日各时间段的航班数量；并根据 GPS 卫星地图确定了对应时刻的出租车数，如下表所示。

表 2：各时间段的航班数量

时间段	航班数量	出租车数
7：50—9：50	44	4832
9：50—11：50	85	5956
11：50—13：50	89	6376
13：50—15：50	96	3224
15：50—17：50	92	4
17：50—19：50	99	0
19：50—21：50	100	3044

5.1.2 司机期望时薪和单位时间成本的确定

上海市出租车的起步价格为 14 元/3 公里，超起租里程为 3-15 公里时，单价为 2.5 元/公里，超运距加价距离为 15 公里起步，单价为 3.6 元/公里；低速等候费为每 4 分钟/公里，夜间加价上浮 30%[9]；上海市虹桥机场距市中心约为 15 公里，路上有 8 个红绿灯，红绿灯时长为 3 分钟；假设遇到红绿色为相互独立事件，则由概率论的知识可知，平均会遇到的红绿灯个数为 3.4 个。

① 数据来源网站：https://pan.baidu.com/wap/init?surl=I328htw_iRGovbwka7NVmw&tdsourcetag=s_pcqq_aiomsg&qq-pf-to=pcqq.group，提取码 n3r1。

② 数据来源网站：https://www.shanghaiairport.com/。

假设除早高峰、晚高峰外，在时间段 7：50-9：50 或 17：50-19：50 内，道路畅通无阻；早、晚高峰时，堵车时间各为一个小时。司机行驶时间为 12 小时，早高峰和晚高峰时间之和为 4 小时，而其他时间段不堵车，故堵车时长的期望为 $1/3$h，因此司机的期望单位时间收入为 64 元/h。

根据文献[1-3,5]的调查结论和部分主观经验，本文假设出租车在郊区行驶速度为 70km/h，油耗为 4.70L/100km；在市区行驶的速度为 50km/h，油耗为 4.11L/100km。车开启并处于等待时耗油量为 1L/h。根据查询资料，可得出租车所用的 93 号汽油油价为 5.75 元/L。

因此，计算可得出租车的等待时间成本 $c_1 = 5.75$ 元/L，驶离机场期的时间成本为 $c_2 = 18.92$ 元/L，市区行驶的时间成本为 $c_3 = 11.82$ 元/L。

5.1.3 不同类型司机的量化

依据司机个人的性格特点，将司机分为激进型、平衡型、保守型。不同类型的司机依据个人经验对于愿意乘车人数比例的判断不同，激进型司机对于此比例的判断大于平衡型且大于保守型。下表给出了不同类型司机在各个时间段对愿意乘坐出租车的乘客数比例的估计值及对应时刻的供应和需求量，如下表所示。

表 3：不同类型司机在不同时段对供需量的判断

时间段	激进型			平衡型			保守型		
	$\alpha(t)$	需求	供应	$\alpha(t)$	需求	供应	$\alpha(t)$	需求	供应
7：50—9：50	0.8	4224	6040	0.4	5100	6040	0.2	1056	6040
9：50—11：50	0.9	9180	7445	0.5	5340	7445	0.3	3060	7445
11：50—13：50	0.9	9612	7970	0.5	5760	7970	0.3	3204	7970
13：50—15：50	0.9	10368	4030	0.5	5520	4030	0.3	3456	4030
15：50—17：50	0.9	9936	5	0.5	5940	5	0.3	3312	5
17：50—19：50	0.9	10692	0	0.5	6000	0	0.3	3564	0
19：50—21：50	0.9	10800	3805	0.5	6180	3805	0.3	3600	3805
21：50—23：50	0.9	11124	7280	0.5	5100	7280	0.3	3708	7280

5.2 实证分析：司机的决策

5.2.1 模型求解

根据第 4.2.2 节中的分析，在情形 1 和情形 2 中，司机分别选择方案 A 和方案 B。

对于情形 3，根据第 4 章的模型和第 5.1 节的数据，计算得到不同时刻不同类型的司机的决策，如下表所示。

表 4：情形 3 下司机的决策

时间段	激进型	平衡型	保守型
7：50—9：50	B	B	B
9：50—11：50	A	B	B
11：50—13：50	A	B	B
13：50—15：50	A	A	B
15：50—17：50	A	A	A
17：50—19：50	A	A	A
19：50—21：50	A	A	A
21：50—23：50	A	A	A

5.2.2 结果分析

表 3 表明，不同类型的司机在不同时刻对供应量和需求量的预计具有较大的差异，例如中午 11:50-13:50，激进型司机预计的需求量高出保守型司机 6408 人。

而表 4 的结果表明，尽管司机的判断力不一样，但是在本文的决策模型下，司机的决策几乎保持一致，没有出现过差异性，说明本模型具有良好的可用性，对不同类型的司机均具有可靠的参考价值。造成结果的无差异性，也可能是司机估计的愿意乘坐出租车的乘客占总乘客的比例差距仍不够大，**因为在表 3 中，相邻类别的司机的判断比例差值一般为 0.1，极少数为 0.2，区分度不够**，在下文的灵敏度分析中，讲进一步考虑同一个时刻，不同判断比例对司机决策的影响。

5.3 灵敏度分析

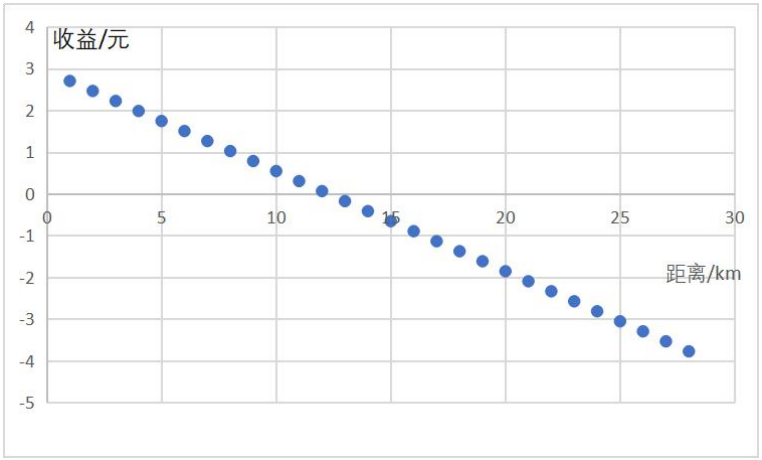

图 4 距离因子的灵敏度分析

图 4 给出了乘客目的地的远近对司机决策的影响。由图可知乘客目的地与机场的距离越远，司机的收益越大，距离越近，司机的收益越小，甚至可能赔本。

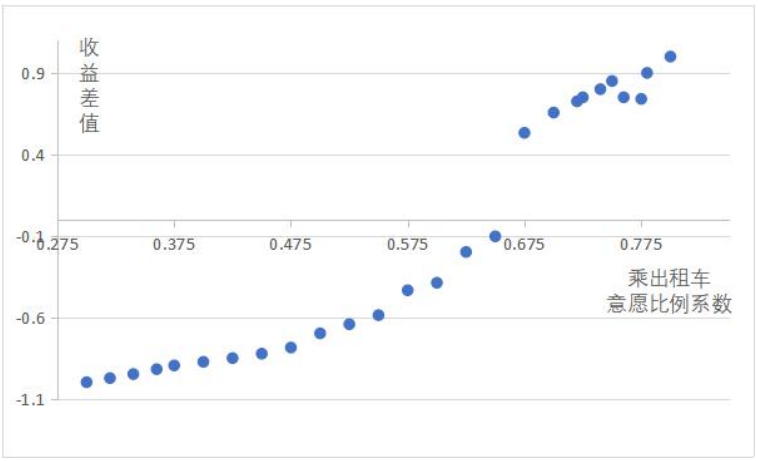

图 5 比例因子的灵敏度分析

图 5 给出了在 9:50-11:50 的时段内不同比例因子对司机决策的影响。据上表分析可知：比例因子越大，司机越愿意去选择 A 方案。司机的经验判断因子对结果具有一定的影响，经验越丰富的司机，做出正确选择的几率越大，收益也越多。

六、基于元胞自动机的最优上车方式推荐

6.1 六种备选方案的介绍

通过参考国内外一些大型机场的“乘车区”布局和一些组内想法，本文根据单侧上车还是双侧上车、相邻上车点间距大小，乘客行李的多少，提出了六种并行车道的上车规则。

第一种，两侧乘车，乘客分流制：在这种规则下，一部分乘客直接在右车道的右侧排队等候上车，一部分乘客在左车道的左侧排队等候上车；且根据乘客行李的多少，分别多行李乘客和单行李乘客。本文的判断标准为，如果需要开后备箱放行李，无论多少人乘车，都属于多行李乘客；如果不需要开后备箱放行李，均属于单行李乘客。第一种方案的“乘车区”示意图如下所示。

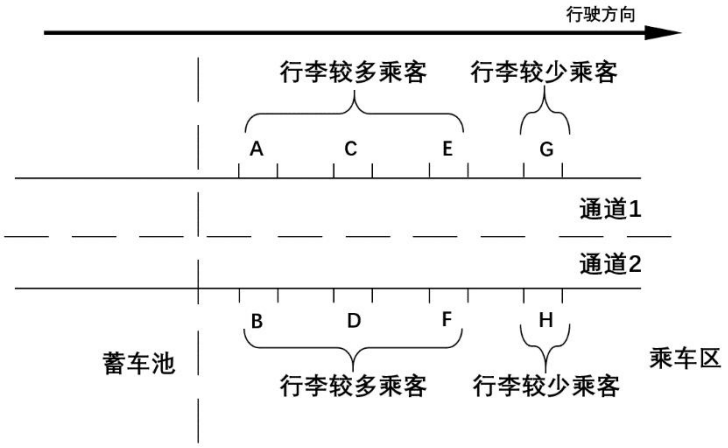

图 4 第一种上车方案的乘车示意图

第二种，两侧乘车，乘客不分流制：除不区分乘客的行李外，其他同第一种规则。

第三种，单侧乘车，相邻上车点大间距，乘客分流制：在这种规则下，所有乘客都在右车道的右侧排队等候上车；相邻上车点间距较大，前方车辆上客时不影响后方车辆启动；乘客分流的含义同上。

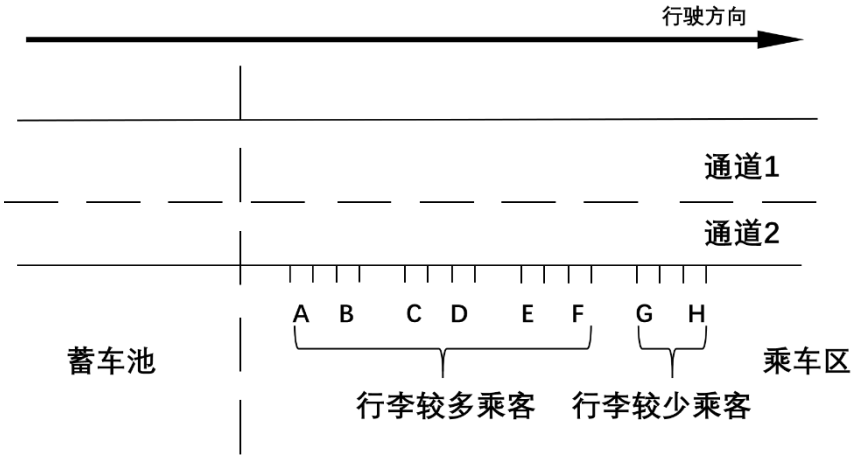

图 5：第三种上车方案的乘车示意图

第四种，单侧乘车，相邻上车点大间距，乘客不分流制：除乘客不分流外，其他同第三种。

第五种，单侧乘车，相邻上车点小间距，乘客分流制：在这种规则下，相邻上车点间距较小，前方车辆上客时影响后方车辆启动，此时约定单号上车口车辆离开时直行，双号上车口车辆变道驶离；其他同第三种规则。

第六种，单侧乘车，相邻上车点小间距，乘客不分流制：除乘客不分流外，其他同第五种。

根据第 2.3 节的分析， 首先假设共有 8 个上车点，乘客分流时，多行李乘客上车点为 6 个，单行李乘客上车点为 2 个；如果是双侧上车，各上车点平均给两侧。

为了更好地模拟乘车区的模型，本文选用元胞自动机来进行系统的仿真。

6.2 元胞自动机仿真

6.2.1 基本结构和假设

本文只关注交通本身，不考虑其他因素，因此对乘车区做如下假设：

- 没有行人闯入交通。
- 乘车区公路笔直。
- 没有其他因素（如天气或气温）影响交通。

对驾驶员做出如下假设：

- 所有驾驶员遵守相同的规则。
- 驾驶员换道时会在优先让行直到行驶车辆。
- 两辆车作为元胞连接时仍存在最短换道距离，驾驶员能进行换道。

这些假设是模型有效的重要保证，同时也是合理可解释的。

本文模型包含两条车道，每一行代表一个车道，车道长度为 19 个元胞的长度。每个元胞表示一辆车。

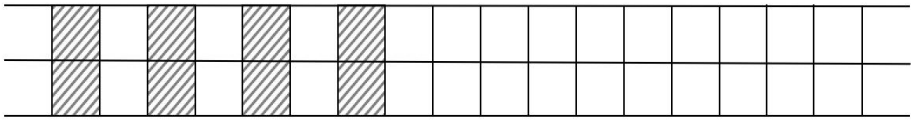

图 6：车道车辆模拟图

车辆在离散时间间隔中移动，每一步每辆车将向前移动一个整数元胞。假设车辆变道用时 2.5s，由于单位时间的车速大小是元胞的个数，所以这样一种假设会使得我们速度的取值比较有限。针对车辆的行驶情况，一辆轿车的长度为 1 个元胞，单位时间最多前进 2 个元胞。

由于车与车之间应该有安全车距，且应满足最小换道距离。为了保证相邻车辆存在的合理性,我们假设车虽然占据了 1 或者 2 个元胞,但实际上并没有把它们完全占据满。为了使得元胞和速度的大小有意义,我们假设每个元胞的长度为 4.5m,出租车长为 4m。利用这些信息,可知出租车的最大车速为 4.5*2=9（m/s），折算后等于 9*3.6=32.4（km/s），符合一般“乘客区”的车速上限要求。

在基本结构中,我们将建立变道的准则,首先将右行车道准则进行精确描述。

6.2.2 向左车道换道准则

在右车道的车辆 V 将会向左车道移动,如果：

- 车辆 V 的驾驶员希望超越正前方的车辆；
- 车辆 V 的前方有足够的空间允许它变道，这意味着：

- V 的左边没有车辆；
- 如果 V 变道,其左后方最近的车辆不会撞到它；
- 如果 V 变道,它不会撞到其左前方最近的车辆。

通过引入下列参数,可以让上述表达数学化：

- V：所考虑的车辆代号；
- v：车辆 V 的车速；
- D：车辆 V 的车头与同车道前面的车尾之间的距离；
- V_{expect}：车辆 V 的期望速度；
- V_{lb}：车辆V 左边车道后面的车；
- v_{lb}：车辆V 左边车道后面的车的速度；
- D_{lb}:车辆V 的尾部和后面车辆的头部的距离；
- V_{tb} 、v_{tb}、D_{tb}:表示前方的车辆，含义类似。

下图给出了向左变道的示意图，利用这些已经定义过的概念，右车道的车将会变道到左车道，如果满足以下条件:

- 车辆 V 的驾驶员希望比他正前方的车跑得快：$v_{\text{expect}} > v_{\text{tb}}$ (这里我们假设 V 和前面的车速相同)；
- 如果 V 变道，他左后方的车辆不会撞到他：$D_{\text{lb}} > (v_{\text{lb}} - v_{\text{tb}})t$；
- 如果 V 变道，他不会撞到他左前方的车：$D_{\text{tb}} > (v_{\text{tb}} - v_{\text{lb}})t$。

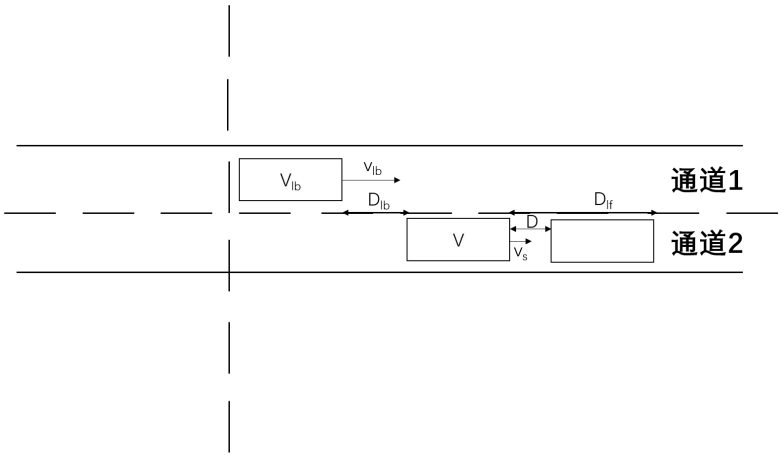

图 7：向左变道示意图

6.3模型评价准则的选取

在元胞自动机进行仿真时，将载客效率、安全性和人们的满意度作为评价准

则，来检验模型的运行能力，并评价其优劣。

- 载客效率：单位时间送走乘客的数量
- 安全性：急刹车率和换道率。
- 满意度：行驶一段给定距离，预计时间与实际用时的比例，这里几既包括了司机，也包括了乘客的满意度。

6.4 模型评价结果

	方案 1	方案 2	方案 3	方案 4	方案 5	方案 6
载客效率	0.166720697	0.206287999	0.157195696	0.152122969	0.462258468	0.398289357
危险度	0.75	0.000296707	0.601336898	0.587206266	0.135324779	0.136750405
满意度	0.556834304	0.755922443	0.728810831	0.681190035	0.702140516	0.789274974

由上表可知，方案 5 的载客效率最高；方案 2 安全性最高；方案 6 满意度最高。综合载客效率、安全性以及满意度三种因素权衡考虑，我们发现方案 5 和 6 都是较为优秀的设置模型，其载客效率、安全性以及乘客满意度都很高。但是考虑到机场的实际情况属于人员拥堵地段，如何疏通乘客显得最为迫切且实际，如何使机场人员流动效率变高显得更为重要。所以本文认为在实际设置时，应选择载客效率最高的方案 5，以解决机场的实际需求，达到最好的疏通效果。

在理想条件下，本文认为上车口设立越多越好，但是实际问题中上车口会消耗大量的资金和人力资源，这些费用都需要机场自己去承担，所以根据生活经验与实际分析，我们建议上车口设立为 6~8 个最为优良。

七、短途载客的补贴：优先权的优化规划

第 5.3 节的分析表明，机场出租车的短途运营虽也可盈利，但是平均盈利低于市区正常盈利，这在一定程度上会制约出租车司机前往机场载客的积极性。本问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。

假设如果给予某辆出租车优先权，那么有足够成熟的物联网和物理技术将该出租车的载客顺序调整到方案制定的位置，即模型不需要考虑短途载客司机再次返回后，如何从队列的末端调整到前端。

7.1 短途概念的界定

短途不是一个确切的概念，“优先权”的初衷就是使短途出租车的收益尽量均

衡，因此等待的时间也是很重要的因素。

用 $p(i)$ 表示编号为 i 的司机在“蓄车池”接单的单位时间收益，定义为

$$p(i)=\frac{G(i)}{t_1(i)+t_2(i)+t_3(i)},$$

其中 $F(i)$ 为司机 i 该订单的收入，$t_1(i)$ 表示司机 i 的等待时间，t_2 表示司机 i 驶离机场的时间，t_3 表示司机 i 市区行驶的时间。

用 $\overline{p}$ 表示全体司机的在“蓄车池”接单平均的单位时间收益，现实中可以通过抽样调查确定，则

$$\overline{p}=\frac{1}{n}\sum_{i=1}^{n}p(i),$$

其中 n 为抽样调查的司机的总数。

定义：一单生意被认定为短途生意，若满足即 $p(i)<\overline{p}$，即对于司机 i 在“蓄车池”完成的某一单生意而言，这单生意的单位时间收益小于平均收益。

本文约定，如果司机完成了一单短途生意，如果他选择立马返回机场继续载客，则可享有优先级；如果司机中途接了其他生意，则不享有优先级。

7.2 优先安排方案的确定

假设存在优先级插队车道（类似于高速公路上的应急车道，普通车辆不得进入该车道，否则予以相关处罚，或者采取识别，经识别通过的车辆才能被放行进入该车道。），考虑到乘客和司机的安全性问题，插队车道的出口是当前排队队列的下一名乘客可乘坐的出租车对应的位置，如下图所示。

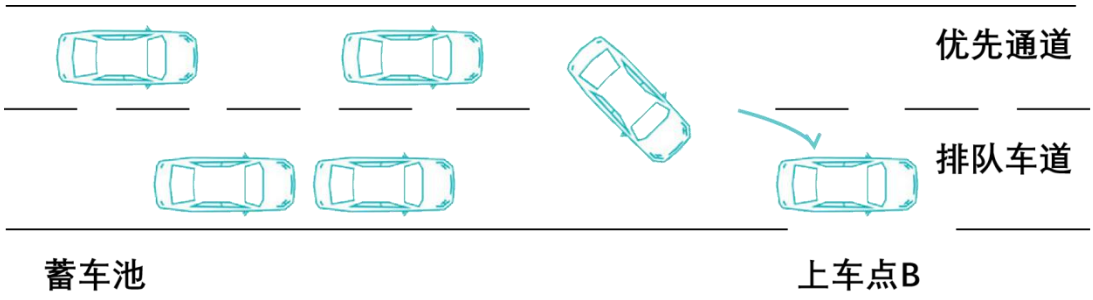

图 8：优先方案示意图

本文给出了两种优先安排方案。

7.2.1 方案 1：先到先安排

在这种方案下，不区分优先权的级别，按照“先来后到”的原则进行优先安排。

当有优先权的车辆来时，在优先车道排队，然后按照“优先车道、排队车道、优先车道、排队车道”逐一交替的方式载客，如下图所示。

7.2.2 方案 2：优先权的量化

在这种方案下，优先权是有高低之分的，优先权越高的，越优先安排。首先定义优先权指数，如果多个司机都完成了短途生意，应该对他们的优先权进行排序。定义司机 i 的优先权指数

$$r(i) = \overline{p} - p(i)\,,$$

且 $r(i)$ 越大，优先级越高。

为了节约空间，提高“蓄车池”利用率，仍假设只有一条优先车道。确定一个时间间隔，如 15 分钟，即优先车道每 15 分钟开放一次进入，在 15 分钟内，所有优先级的车辆在“蓄车池”进行排队，根据优先权指数的高低进行排队进入优先权插队车道。

这里设置的时间间隔（如 15 分钟）是为了一方面是为了给有优先权的车辆进行排序，另一方面是为了控制等待时间，如果时间间隔太长，则会提高车辆的排队时间，而如果时间间隔太短，则优先级较高的车辆难以体现出差异，当时间间隔趋向于 0 时，方案 2 趋向于方案 1。

7.3 优先方案效果的度量

优先方案的初衷使为了使得短途出租车的收益均量均衡。可利用元胞自动机进行再次仿真，计算所有司机的单位时间收益。其中短途司机将两次订单合成一个订单计算单位时间收益。

通过对比优先方案前后的单位时间收益标准差，即可确定优先方案的有效性；进一步，还可以通过控制变量法，确定方案 2 中的最优时间间隔。

受时间限制，本文没有办法进行仿真模拟，故仅提供方案思路。

八、优缺点分析

本文建立了收益最大化视角下的决策模型、基于元胞自动机的最佳设置模型以及优先权的优化模型。

8.1 优点分析

（1）逻辑思维系统全面，即从宏观上对整体的系统效率进行考量，也从微观上对司机个体的行为进行判断

（2）对乘客的到达，司机的判断力假设成与时间相关的变量，更符合实际情况。

（3）模型参数来源主要取决于文献，结果真实可靠。

（4）设定符合现实的交通规则，采用元胞自动机仿真对系统进行模拟，较好地还原了现实真实情况中的随机性，并重点衡量效率和安全指标。

（5）对“短途”概念进行了精确量化，并设计了多种优先安排方案，具有较强的可操作性，且不需要大幅度修改现有的“蓄车池”系统，可供管理部门参考。

8.2 缺点分析

（1）部分模型参数源于参考文献，但是这些文献研究的并不完全是同一个城市，对模型结果可能产生一定的误差。

（2）在寻找机场数据时本文于 9 月 14 搜寻了 9 月 14 日的最新数据，由于时间问题导致 21:50 至次日 7:50 的数据还没上传，如果最开始找 9 月 13 日或之前的数据，结果会更加全面，但并不影响模型的使用。

（3）在用元胞自动机对上车方式进行评价时，首先固定了上车点，然后进行模拟，可能对结果有一定的误差。

九、参考文献

[1] 黎冬平,晏克非,程林结,许明明.机场出租车上客区的服务水平模型[J].哈尔滨工业大学学报,2011,43(04):126-130.

[2] 黄岩,王光裕.虹桥机场 T2 航站楼出租车上客系统组织管理优化探讨[J].城市道桥与防洪,2014(12):7-9+35-36.

[3] 白竹,王健,胡晓伟.城市出租车系统运营效率评价研究[J].交通运输系统工程与信息,2014,14(03):227-233.

[4] 李旭宏,唐怀海,吴炼,朱彦东.综合客运枢纽车道边通行能力分析[J].公路交通科技,2009,26(04):128-132.

[5] 耿中波,宋国华,赵琦,高永,万涛,辛然.基于 VISSIM 的首都机场出租车上客方案比选研究[J].中国民航大学学报,2013,31(06):55-59.

[6] 胡日聪.多目标决策在城市出租车规划中的应用[J].黑龙江科技信息,2008(29):62-63.

[7] 赵桂香.基于出租车承载行为的优化决策调度方法研究[D].重庆交通大学,2015.

[8] 黎明,沈锦发,张旭.空载出租车在马尔科夫决策中的平稳分布[J].生产力研究,2017(09):121-123+154+161.

[9] 唐梦斐.上海市出租车经营模式研究[D].华中科技大学,2015.

附录

1. 基于元胞自动机的最佳设置模型中第一种方案模型代码：

```
from random import *
car1=[[0,0,0,0],]                                    #car[0 车的有无，1 车的速度，2 送客时间,3 乘客数】
for i in range(14):
    car1.append([0,0,0,0])
for i in range(4):                                   #前 15 为行驶区，后 4 为接客口
    car1.append([1, 0, 0, 0])
car2=car1
sum=[]                                               #总人数
time=[]                                              #时间序列
stop=[]                                              #停车数
time1=[]                                             #总时间
a=[]                                                 #道路密度

def speed(car):                                      #判断速度改变
    if car[1][0] and car[0][0]:                      #判断位于位置 2 的辆车是否需要减速
        car[1][1] = 0
        if car[1][1] != 0:
            stop.append(1)
    for i in range(19):
        if car[i][0]    and car[i-1][0]:             #如果前面有车，速度降至 0
            if car[i][0]!=0:
                stop.append(1)
            car[i][1]=0
        if car[i][0] and car[i-1][0] !=1:            #如果前面没车，前前面有车，速度为 1
            car[i][1]=1
        if car[i][0] and car[i - 1][0] != 1 and car[i-2]!=1:
            car[i][1]=2                              #如果前前面也没车，速度为 2
```

```
def move(car):                                          #车移动
    for i in range(19):
        if car[i][1]:
            car[i][2]+=1                                #时间增加
            x=i-car[i][1]                               #目的地 x
            if x<0:                                     #如果开出去了，则在
时间里加入数据
                time.append(car[i][2])
                sum.append(car[i][3])
            else:
                car[x]=car[i]
            car[i]=[0,0,0,0]                            #清除数据
    b=0
    for i in range(15):
        if car1[i][0]==1:
            b+=1
        if car2[i][0]==1:
            b+=1
    a.append(b/15)

def baggage(x):
    t=0
    for i in range(x):
        if random()<=0.75:
            t+=4
    return t

def PessengerUp(car,x):
    for i in range(4):
        ti=[]
        if x <= 0.1:
            n = 3
            t=7
            if x-i!=0:
```

```
                t = 7 + baggage(2)
        if x >= 0.1 and x <= 0.3:
            n = 2
            t=15
            if x-i!=0:
                t = 15 + baggage(1)
        if x >= 0.3:
            n = 1
            t=24
            if x-1!=0:
                t = 24 + 4
        ti.append(t)
        car[15+i][0]=1
        car[15+i][1]=1
        car[15+i][2]=t
        car[15+i][3]=n
    time1.append(max(ti))
    return max(ti)

def start(car,x):
    t=PessengerUp(car,x)                    #获取乘客上车时间
    for i in range(t):
        speed(car)                          #速度改变
        move(car)                           #车动
    for i in range(4):
        car[15+i][1]=1


num=int(input("元胞活动次数为："))
for i in range(num):
    start(car1,1)
    start(car2,2)

x=0
for i in sum:
```

```
    x+=i
t=0
for i in time1:
    t+=time1[i]
t=t-time[0]+time1[0]-time[num-1]+time1[num-1]
print("停车数为：",len(stop))
print("总共送走游客：",x)
print("总共耗时：",t)
print("每次送客时间分别为：",time)
print("每次上车的人分别为：",sum)
print("道路密度为：",a)
```

2.基于元胞自动机的最佳设置模型中第二种方案模型代码：

```
from random import *
car1=[[0,0,0,0],]                                    #car[0 车的有无，1 车的速度，2 送客时间,3 乘客数】
for i in range(14):
    car1.append([0,0,0,0])
for i in range(4):                                   #前 15 为行驶区，后 4 为接客口
    car1.append([1, 0, 0, 0])
car2=car1
sum=[]                                               #总人数
time=[]                                              #时间序列
stop=[]                                              #停车数
time1=[]                                             #总时间
a=[]                                                 #道路密度

def speed(car):                                      #判断速度改变
    if car[1][0] and car[0][0]:                      #判断位于位置 2 的辆车是否需要减速
        if car[1][1] != 0:
            stop.append(1)
        car[1][1] = 0
        stop.append(1)
```

```
    for i in range(19):
        if car[i][0]    and car[i-1][0]:                    #如果前面有车，速度降至 0
            car[i][1]=0
            if car[i][1] != 0:
                stop.append(1)
        if car[i][0] and car[i-1][0] !=1:                   #如果前面没车，前前面有车，
速度为 1
            car[i][1]=1
        if car[i][0] and car[i - 1][0] != 1 and car[i-2]!=1:
            car[i][1]=2                                     #如果前前面也没车，
速度为 2

def move(car):                                              #车移动
    for i in range(19):
        if car[i][1]:
            car[i][2]+=1                                    #时间增加
            x=i-car[i][1]                                   #目的地 x
            if x<0:                                         #如果开出去了，则在
时间里加入数据
                time.append(car[i][2])
                sum.append(car[i][3])
            else:
                car[x]=car[i]
            car[i]=[0,0,0,0]                                #清除数据
    b=0
    for i in range(15):
        if car1[i][0]==1:
            b+=1
        if car2[i][0]==1:
            b+=1
    a.append(b/15)

def baggage(x):
    t=0
    for i in range(x):
```

```
            if random()<=0.75:
                t+=4
        return t

def PessengerUp(car):
        for i in range(4):
            x=random()
            ti=[]
            if x<=0.1:
                n=3
                t=7+baggage(3)
            if x>=0.1 and x<=0.3:
                n=2
                t=15+baggage(2)
            if x>=0.3:
                n=1
                t=24+baggage(1)
            ti.append(t)
            car[15+i][0]=1
            car[15+i][1]=1
            car[15+i][2]=t
            car[15+i][3]=n
        time1.append(max(ti))
        return max(ti)

def start(car):
        t=PessengerUp(car)                                  #获取乘客上车时间
        for i in range(t):
            speed(car)                                      #速度改变
            move(car)                                       #车动
        for i in range(4):
            car[15+i][1]=1



num=int(input("元胞活动次数为："))
```

```
for i in range(num):
    start(car1)
    start(car2)


x=0
for i in sum:
    x+=i
t=0
for i in time1:
    t+=time1[i]
t=t-time[0]+time1[0]-time[num-1]+time1[num-1]
print("停车数为：",len(stop))
print("总共送走游客：",x)
print("总共耗时：",t)
print("每次送客时间分别为：",time)
print("每次上车的人分别为：",sum)
print("道路密度为：",a)
```

3.基于元胞自动机的最佳设置模型中第三种方案模型代码：

```
from random import *
car1=[[0,0,0,0,0,0],]                          #car[0 车的有无，1 车的速度，2 送客时间,3 乘客数,4,提升后速度，5 拐弯安全】
for i in range(14):                            #补充剩余的 14 个
    car1.append([0,0,0,0,0,0])
for i in range(4):                             #补充 2 个车子(初始状态即可想要转弯且能转弯）
    car1.extend(([1,0,0,0,0,0],[0,0,0,0,0,0]))
car2=[]
for i in range(19+2):
    car2.append([0,0,0,0,0,0])
sum=[]                                         #总人数
time=[]                                        #时间序列
stop=[]                                        #停车数
time1=[]                                       #总时间
a=[]                                           #道路密度
```

```
def speed(car):                                          #判断速度改变
    if car[1][0] and car[0][0]:                          #判断位于位置 2 的辆车是否需要减速
        if car[1][1] != 0:
            stop.append(1)
        car[1][1] = 0
        stop.append(1)
    for i in range(2,19+2):
        if car[i][0]    and car[i-1][0]:                 #如果前面有车，速度降至 0
            car[i][1]=0
        if car[i][0] and car[i-1][0] !=1:                #如果前面没车，前前面有车，速度为 1
            car[i][1]=1
        if car[i][0] and car[i - 1][0] != 1 and car[i-2]!=1:
            car[i][1]=2                                  #如果前前面也没车，速度为 2
        #预测到超车道的速度
        if car2[i-1][0]==0and car2[i-2][0]==0:
            car[i][4]=2
        if i<15+2:                                       #判断后面的车是否可以安全转车
            if car2[i+1][0]==0 and car2[i+2][0]==0 and car2[i+3][0]==0 and car2[i+4][0]==0:
                car1[i][5]=1
        if i==15+2:
            if car2[16+2][0]==0 and car2[17+2][0]==0 and car2[18+2][0]==0:
                car1[15+2][5]=1
        if i==16+2:
            if car2[17+2][0]==0 and car2[18+2][0]==0:
                car1[16+2][5]=1
        if i==17+2:
            if car2[18+2][0] == 0:
                car1[17+2][5] = 1
        if i==18+2:
```

```
            car1[18+2][5+2] = 1


def move(car):                                          #车移动
    for i in range(19+2):
        if car[i][1]:
            if car[i][4]==2 and car[i][1] !=0 and car[i][5]==1:                  #转
车道
                if i==18+2:
                    car2[18+2] = car[i]   # 换车
                    car2[18+2[2] += 1   # 换的车时间增加
                else:
                    car2[i+1]=car[i]                        #换车
                    car2[i+1][2] += 1                       #换的车时间增加
            else:
                car[i][2]+=1                                 #时间增加
                x=i-car[i][1]                               #目的地 x
                if x<0:                                      #如果开出去了，
则在时间里加入数据
                    time.append(car[i][2])
                    sum.append(car[i][3])
                else:
                    car[x]=car[i]
            car[i]=[0,0,0,0,0,0]                             #清除数据

    for i in range(19+2):                                   #二车道移动
        if car2[i][1]:
            car2[i][2]+=1                                    #时间增加
            x=i-car2[i][1]                                  #目的地 x
            if x<0:                                          #如果开出去了，则在
时间里加入数据
                time.append(car2[i][2])
                sum.append(car2[i][3])
            else:
                car2[x]=car2[i]
```

```
                car2[i]=[0,0,0,0]                                    #清除数据

        b=0
        for i in range(15):
                if car1[i][0]==1:
                        b+=1
                if car2[i][0]==1:
                        b+=1
        a.append(b/15)


def baggage(x):
        t=0
        for i in range(x):
                if random()<=0.5:
                        t+=4
        return t

def PessengerUp(car):
                x=random()
                y=random()
                ti=[]
                if x<=0.1:
                        n=3
                        t=7+12
                if x>=0.1 and x<=0.3:
                        n=2
                        t=15+8
                if x>=0.3:
                        n=1
                        t=24+4
                car[15][0]=1
                car[15][1]=1
                car[15][2]=t
                car[15][3]=n
```

```
x = random()
if x <= 0.1:
        n = 3
        t = 7 + baggage(3)
if x >= 0.1 and x <= 0.3:
        n = 2
        t = 15 + baggage(2)
if x >= 0.3:
        n = 1
        t = 24 + baggage(1)
ti.append(t)
car[17][0] = 1
car[17][1] = 1
car[17][2] = t
car[17][3] = n

x = random()
if x <= 0.1:
        n = 3
        t = 7 + baggage(3)
if x >= 0.1 and x <= 0.3:
        n = 2
        t = 15 + baggage(2)
if x >= 0.3:
        n = 1
        t = 24 + baggage(1)
ti.append(t)
car[19][0] = 1
car[19][1] = 1
car[19][2] = t
car[19][3] = n
time1.append(max(ti))
return max(ti)
```

```
def start(car):
    t=PessengerUp(car)                          #获取乘客上车时间
    for i in range(t):
        speed(car)                              #速度改变
        move(car)                               #车动
    for i in range(4):
        car[15+i][1]=1


num=int(input("元胞活动次数为："))
for i in range(num):
    start(car1)


x=0
for i in sum:
    x+=i
t=0
for i in time1:
    t+=time1[i]
t=t-time[0]+time1[0]-time[num-1]+time1[num-1]
print("停车数为：",len(stop))
print("总共送走游客：",x)
print("总共耗时：",t)
print("每次送客时间分别为：",time)
print("每次上车的人分别为：",sum)
print("道路密度为：",a)
print("分")
```

4.基于元胞自动机的最佳设置模型中第四种方案模型代码：

```
from random import *
car1=[[0,0,0,0,0,0,0],]                         #car[0 车的有无，1 车的速度，2 送客时间,3 乘客数,4,提升后速度，5 拐弯安全】
for i in range(14):                             #补充剩余的 14 个
    car1.append([0,0,0,0,0,0,0])
```

```
for i in range(4):                                    #补充 2 个车子(初始状态即可想要转弯且能转弯）
    car1.extend(([1,0,0,0,0,0],[0,0,0,0,0,0]))
car2=[]
for i in range(19+2):
    car2.append([0,0,0,0,0,0])
sum=[]                                                #总人数
time=[]                                               #时间序列
stop=[]                                               #停车数
time1=[]                                              #总时间
a=[]                                                  #道路密度

def speed(car):                                       #判断速度改变
    if car[1][0] and car[0][0]:                       #判断位于位置 2 的辆车是否需要减速
        if car[1][1] != 0:
            stop.append(1)
        car[1][1] = 0
        stop.append(1)
    for i in range(2,19+2):
        if car[i][0]    and car[i-1][0]:              #如果前面有车，速度降至 0
            car[i][1]=0
        if car[i][0] and car[i-1][0] !=1:             #如果前面没车，前前面有车，速度为 1
            car[i][1]=1
        if car[i][0] and car[i - 1][0] != 1 and car[i-2]!=1:
            car[i][1]=2                               #如果前前面也没车，速度为 2
        #预测到超车道的速度
        if car2[i-1][0]==0and car2[i-2][0]==0:
            car[i][4]=2
        if i<15+2:                                    #判断后面的车是否可以安全转车
            if car2[i+1][0]==0 and car2[i+2][0]==0 and car2[i+3][0]==0 and car2[i+4][0]==0:
```

```
                car1[i][5]=1
        if i==15+2:
            if car2[16+2][0]==0 and car2[17+2][0]==0 and car2[18+2][0]==0:
                car1[15+2][5]=1
        if i==16+2:
            if car2[17+2][0]==0 and car2[18+2][0]==0:
                car1[16+2][5]=1
        if i==17+2:
            if car2[18+2][0] == 0:
                car1[17+2][5] = 1
        if i==18+2:
            car1[18+2][5] = 1


def move(car):                                          #车移动
    for i in range(19+2):
        if car[i][1]:
            if car[i][4]==2 and car[i][1] !=0 and car[i][5]==1:                  #转
车道
                if i==18+2:
                    car2[18+2] = car[i]    #  换车
                    car2[18+2][2] += 1    #  换的车时间增加
                else:
                    car2[i+1]=car[i]                            #换车
                    car2[i+1][2] += 1                           #换的车时间增加
            else:
                car[i][2]+=1                                     #时间增加
                x=i-car[i][1]                                   #目的地 x
                if x<0:                                          #如果开出去了，
则在时间里加入数据
                    time.append(car[i][2])
                    sum.append(car[i][3])
                else:
                    car[x]=car[i]
            car[i]=[0,0,0,0,0,0]                                 #清除数据
```

```
    for i in range(19+2):                                   #二车道移动
        if car2[i][1]:
            car2[i][2]+=1                                    #时间增加
            x=i-car2[i][1]                                  #目的地 x
            if x<0:                                          #如果开出去了，则在
时间里加入数据
                time.append(car2[i][2])
                sum.append(car2[i][3])
            else:
                car2[x]=car2[i]
            car2[i]=[0,0,0,0]                                   #清除数据

    b=0
    for i in range(15):
        if car1[i][0]==1:
            b+=1
        if car2[i][0]==1:
            b+=1
    a.append(b/15)


def baggage(x):
    t=0
    for i in range(x):
        if random()<=0.75:
            t+=4
    return t

def PessengerUp(car):
        x=random()
        ti=[]
        if x<=0.1:
            n=3
            t=7+baggage(3)
```

```
if x>=0.1 and x<=0.3:
        n=2
        t=15+baggage(2)
if x>=0.3:
        n=1
        t=24+baggage(1)
ti.append(t)
car[15][0]=1
car[15][1]=1
car[15][2]=t
car[15][3]=n

x = random()
if x <= 0.1:
        n = 3
        t = 7 + baggage(3)
if x >= 0.1 and x <= 0.3:
        n = 2
        t = 15 + baggage(2)
if x >= 0.3:
        n = 1
        t = 24 + baggage(1)
ti.append(t)
car[17][0] = 1
car[17][1] = 1
car[17][2] = t
car[17][3] = n

x = random()
if x <= 0.1:
        n = 3
        t = 7 + baggage(3)
if x >= 0.1 and x <= 0.3:
        n = 2
        t = 15 + baggage(2)
```

```
            if x >= 0.3:
                n = 1
                t = 24 + baggage(1)
            ti.append(t)
            car[17+2][0] = 1
            car[17+2][1] = 1
            car[17+2][2] = t
            car[17+2][3] = n

            time1.append(max(ti))
            return max(ti)

def start(car):
    t=PessengerUp(car)                          #获取乘客上车时间
    for i in range(t):
        speed(car)                              #速度改变
        move(car)                               #车动
    for i in range(4):
        car[15+i][1]=1


num=int(input("元胞活动次数为："))
for i in range(num):
    start(car1)


x=0
for i in sum:
    x+=i
t=0
for i in time1:
    t+=time1[i]
t=t-time[0]+time1[0]-time[num-1]+time1[num-1]
print("停车数为：",len(stop))
print("总共送走游客：",x)
```

```
print("总共耗时：",t)
print("每次送客时间分别为：",time)
print("每次上车的人分别为：",sum)
print("道路密度为：",a)
```

5.基于元胞自动机的最佳设置模型中第五种方案模型代码：

```
from random import *
car1=[[0,0,0,0,0,0],]                                    #car[0 车的有无，1 车的速度，2 送客时间,3 乘客数,4,提升后速度，5 拐弯安全】
for i in range(14):                                      #补充剩余的 14 个
    car1.append([0,0,0,0,0,0])
for i in range(4):                                       #补充 4 个车子(初始状态即可想要转弯且能转弯）
    car1.extend(([1,0,0,0,0,0],[1,0,0,0,0,0]))
car2=[]
for i in range(23):
    car2.append([0,0,0,0,0,0])
sum=[]                                                   #总人数
time=[]                                                  #时间序列
stop=[]                                                  #停车数
time1=[]                                                 #总时间
a=[]                                                     #道路密度

def speed(car):                                          #判断速度改变
    if car[1][0] and car[0][0]:                          #判断位于位置 2 的辆车是否需要减速
        if car[1][1] != 0:
            stop.append(1)
        car[1][1] = 0
        stop.append(1)
    for i in range(2,19+4):
        if car[i][0]    and car[i-1][0]:                 #如果前面有车，速度降至 0
            car[i][1]=0
        if car[i][0] and car[i-1][0] !=1:                #如果前面没车，前前面有车，速度为 1
```

```
            car[i][1]=1
        if car[i][0] and car[i - 1][0] != 1 and car[i-2]!=1:
            car[i][1]=2                                    #如果前前面也没车，
速度为 2
        #预测到超车道的速度
        if car2[i-1][0]==0and car2[i-2][0]==0:
            car[i][4]=2
        if i<15+4:                                         #判断后面的车是否
可以安全转车
            if car2[i+1][0]==0 and car2[i+2][0]==0 and car2[i+3][0]==0 and
car2[i+4][0]==0:
                car1[i][5]=1
        if i==15+4:
            if car2[16+4][0]==0 and car2[17+4][0]==0 and car2[18+4][0]==0:
                car1[15+4][5]=1
        if i==16+4:
            if car2[17+4][0]==0 and car2[18+4][0]==0:
                car1[16+4][5]=1
        if i==17+4:
            if car2[18+4][0] == 0:
                car1[17+4][5] = 1
        if i==18+4:
            car1[18+4][5] = 1


def move(car):                                             #车移动
    for i in range(19+4):
        if car[i][1]:
            if i%2==0 and car[i][1] !=0 and car[i][5]==1:          #转车道
                if i==18+4:
                    car2[18+4] = car[i]    #  换车
                    car2[18+4][2] += 1    #  换的车时间增加
                else:
                    car2[i+1]=car[i]                       #换车
                    car2[i+1][2] += 1                      #换的车时间增加
```

```
                else:
                    car[i][2]+=1                                    #时间增加
                    x=i-car[i][1]                                   #目的地 x
                    if x<0:                                         #如果开出去了，
则在时间里加入数据
                        time.append(car[i][2])
                        sum.append(car[i][3])
                    else:
                        car[x]=car[i]
                car[i]=[0,0,0,0,0,0]                                #清除数据

    for i in range(19+4):                                           #二车道移动
        if car2[i][1]:
            car2[i][2]+=1                                           #时间增加
            x=i-car2[i][1]                                          #目的地 x
            if x<0:                                                 #如果开出去了，则在
时间里加入数据
                time.append(car2[i][2])
                sum.append(car2[i][3])
            else:
                car2[x]=car2[i]
            car2[i]=[0,0,0,0]                                       #清除数据

    b=0
    for i in range(15):
        if car1[i][0]==1:
            b+=1
        if car2[i][0]==1:
            b+=1
    a.append(b/15)


def baggage(x):
    t=0
    for i in range(x):
```

```
            if random()<=0.75:
                t+=4
        return t

def PessengerUp(car):
        x=random()
        ti=[]
        for i in range(8):
            if i <4:
                if x<=0.1:
                    n=3
                    t=7
                if x>=0.1 and x<=0.3:
                    n=2
                    t=15
                if x>=0.3:
                    n=1
                    t=24
                else:
                    if x <= 0.1:
                        n = 3
                        t = 7 + 12
                    if x >= 0.1 and x <= 0.3:
                        n = 2
                        t = 15 + 8
                    if x >= 0.3:
                        n = 1
                        t = 24 + 4
            ti.append(t)
            car[15+i][0]=1
            car[15+i][1]=1
            car[15+i][2]=t
            car[15+i][3]=n

        time1.append(max(ti))
```

```
    return max(ti)

def start(car):
    t=PessengerUp(car)                                  #获取乘客上车时间
    for i in range(t):
        speed(car)                                      #速度改变
        move(car)                                       #车动
    for i in range(4):
        car[15+i][1]=1



num=int(input("元胞活动次数为："))
for i in range(num):
    start(car1)



x=0
for i in sum:
    x+=i
t=0
for i in time1:
    t+=time1[i]
t=t-time[0]+time1[0]-time[num-1]+time1[num-1]
print("停车数为：",len(stop))
print("总共送走游客：",x)
print("总共耗时：",t)
print("每次送客时间分别为：",time)
print("每次上车的人分别为：",sum)
print("道路密度为：",a)
```

6.基于元胞自动机的最佳设置模型中第五种方案模型代码：

```
from random import *
car1=[[0,0,0,0,0,0],]                                  #car[0 车的有无，1 车的速度，2 送客时间,3 乘客数,4,提升后速度，5 拐弯安全】
for i in range(14):                                     #补充剩余的 14 个
```

```
    car1.append([0,0,0,0,0,0])
for i in range(4):                                  #补充 4 个车子(初始状态
即可想要转弯且能转弯）
    car1.extend(([1,0,0,0,0,0],[1,0,0,0,0,0]))
car2=[]
for i in range(23):
    car2.append([0,0,0,0,0,0])
sum=[]                                              #总人数
time=[]                                             #时间序列
stop=[]                                             #停车数
time1=[]                                            #总时间
a=[]                                                #道路密度

def speed(car):                                     #判断速度改变
    if car[1][0] and car[0][0]:                     #判断位于位置 2 的辆车是
否需要减速
        if car[1][1] != 0:
            stop.append(1)
        car[1][1] = 0
        stop.append(1)
    for i in range(2,19+4):
        if car[i][0]    and car[i-1][0]:            #如果前面有车，速度降至 0
            car[i][1]=0
        if car[i][0] and car[i-1][0] !=1:           #如果前面没车，前前面有车，
速度为 1
            car[i][1]=1
        if car[i][0] and car[i - 1][0] != 1 and car[i-2]!=1:
            car[i][1]=2                             #如果前前面也没车，
速度为 2
        #预测到超车道的速度
        if car2[i-1][0]==0and car2[i-2][0]==0:
            car[i][4]=2
        if i<15+4:                                  #判断后面的车是否
可以安全转车
            if car2[i+1][0]==0 and car2[i+2][0]==0 and car2[i+3][0]==0 and
```

```
car2[i+4][0]==0:
                    car1[i][5]=1
            if i==15+4:
                if car2[16+4][0]==0 and car2[17+4][0]==0 and car2[18+4][0]==0:
                    car1[15+4][5]=1
            if i==16+4:
                if car2[17+4][0]==0 and car2[18+4][0]==0:
                    car1[16+4][5]=1
            if i==17+4:
                if car2[18+4][0] == 0:
                    car1[17+4][5] = 1
            if i==18+4:
                car1[18+4][5] = 1


def move(car):                                          #车移动
    for i in range(19+4):
        if car[i][1]:
            if i%2==0 and car[i][1] !=0 and car[i][5]==1:                #转车道
                if i==18+4:
                    car2[18+4] = car[i]    #  换车
                    car2[18+4][2] += 1    #  换的车时间增加
                else:
                    car2[i+1]=car[i]                            #换车
                    car2[i+1][2] += 1                           #换的车时间增加
            else:
                car[i][2]+=1                                     #时间增加
                x=i-car[i][1]                                   #目的地 x
                if x<0:                                          #如果开出去了，
则在时间里加入数据
                    time.append(car[i][2])
                    sum.append(car[i][3])
                else:
                    car[x]=car[i]
            car[i]=[0,0,0,0,0,0]                                 #清除数据
```

```
    for i in range(19+4):                                   #二车道移动
        if car2[i][1]:
            car2[i][2]+=1                                    #时间增加
            x=i-car2[i][1]                                  #目的地 x
            if x<0:                                          #如果开出去了，则在
时间里加入数据
                time.append(car2[i][2])
                sum.append(car2[i][3])
            else:
                car2[x]=car2[i]
            car2[i]=[0,0,0,0]                                   #清除数据

    b=0
    for i in range(15):
        if car1[i][0]==1:
            b+=1
        if car2[i][0]==1:
            b+=1
    a.append(b/15)


def baggage(x):
    t=0
    for i in range(x):
        if random()<=0.75:
            t+=4
    return t

def PessengerUp(car):
    x=random()
    ti=[]
    for i in range(8):
        if x<=0.1:
            n=3
```

```
            t=7+baggage(3)
        if x>=0.1 and x<=0.3:
            n=2
            t=15+baggage(2)
        if x>=0.3:
            n=1
            t=24+baggage(1)
        ti.append(t)
        car[15+i][0]=1
        car[15+i][1]=1
        car[15+i][2]=t
        car[15+i][3]=n

    time1.append(max(ti))
    return max(ti)

def start(car):
    t=PessengerUp(car)                          #获取乘客上车时间
    for i in range(t):
        speed(car)                              #速度改变
        move(car)                               #车动
    for i in range(4):
        car[15+i][1]=1


num=int(input("元胞活动次数为："))
for i in range(num):
    start(car1)


x=0
for i in sum:
    x+=i
t=0
for i in time1:
```

```
        t+=time1[i]
t=t-time[0]+time1[0]-time[num-1]+time1[num-1]
print("停车数为：",len(stop))
print("总共送走游客：",x)
print("总共耗时：",t)
print("每次送客时间分别为：",time)
print("每次上车的人分别为：",sum)
print("道路密度为：",a)
```