参赛编号: CS-GXDC-0288

赛题类型:算法分析题

团队人数:5

# 早高峰共享单车潮汐点的群智优化

## 一. 摘要

本作品针对厦门市早高峰时段(7:00-9:00)存在的共享单车潮汐现象-即因停车点爆满存在的车辆”还不进”问题,提出了五种识别潮汐现象集中地区的方案.在后续优化建议中使用图方法将用户从潮汐点引导至邻近共享单车密度较低的停车点,进行”削峰填谷”.并提出了共享单车异常状态以及”乱停车”的数据挖掘方法和现象改进建议.

### 作品主要产出:

1. 五种潮汐点识别方法

2. “削峰填谷” 引导措施

3. 基于数据的共享单车锁具, 车体物理状态异常监测

4. “乱停车” 现象集中点

### 主要使用技术:

DBSCAN, Geohash, 数据挖掘, 最短路算法

### 开发工具及环境:

Python3.7, Microsoft SQL Server

## 二. 问题需求

在早高峰(7:00-9:00), 部分繁忙地段时常发生因共享单车点位爆满导致的共享单车占道或因无法停入造成的被罚款问题. 通过对哈啰单车调度人员的调查发现,目前较为常用的,解决”借不到”, ”还不进”问题的方案是通过调度车将单车淤积严重地区的共享单车运送至临近非潮汐点, 但这无疑是极其耗费物力人力的.

就此问题,我们需要对共享单车订单数据及停车点位坐标进行探索,得出潮汐现象最严重的地区并提供用户引导建议,以减少调度车使用次数,节约成本.

## 三. 数据应用

### 3.1 数据清单

##### 3.1.1 原始文件使用概况

|  |  |  |
| --- | --- | --- |
| 文件名 | 文件说明 | 使用字段及应用说明 |
| gxdc\_dd.csv | 共享单车订单数据 | BICYCLE\_ID: 进行数据挖掘,以此识别行为异常单车并清洗  LATITUDE, LONGITUDE: 自行车坐标. 进行归属停车点分类及”乱停车”现象识别  UPDATE\_TIME:用于配合BICYCLE\_ID进行数据清洗,在可视化中也进行了使用 |
| gxdc\_tcd.csv | 停车点围栏数据 | 围栏定位,用于自行车归属停车点分类和围栏聚类 |
| gxdc\_gj20201221.csv  gxdc\_gj20201222.csv  gxdc\_gj20201223.csv  gxdc\_gj20201224.csv  gxdc\_gj20201225.csv | 行车轨迹数据 | 行车轨迹可视化 |

### 3.2. 数据分析细节

#### 3.2.1 数据完善

##### 3.2.1.1 面积计算

使用给定的围栏多边形坐标点计算围栏面积(单位:平方米).命名为AREA字段

##### 3.2.1.2 geohash(./tools/hashing.py)

geohash的原理是将经纬度编码至字符串,以等长宽方格进行划分, 可大幅缩小距离查找的计算量. 字符串长度越长, 划分精度越高, 优化也越明显,但对于本次竞赛数据,过长的geohash直接加入字典进行查找(使用字典树数据结构可以解决这个问题,但在本作品中没有使用.7位geohash无法覆盖到的自行车数据并不是很多,对于所有数据使用字典树查找反而会增加计算时间)会导致较大的停车点无法被全覆盖以至于在其中部分区域的车辆无法被计入的问题.

在上一小节的面积计算中我们可以得知长度最大的停车围栏(展鸿路\_L\_B10002)约为84米. 使用7位即153\*153方格的geohash编码围栏四角可将整个围栏完全覆盖在内,不会出现因围栏中间不被任何一个顶点编码覆盖而无法将此范围内自行车计入点位的状况.

因仅使用单个编码格时容易出现如图3.2.1.2所示的边界问题

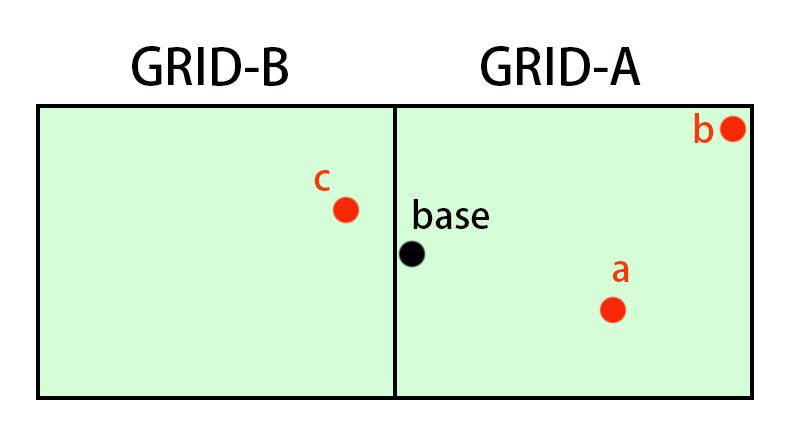


图2.3.2 错误最近点现象: 因geohash划分, 错误地将a识别为离base最近的点,而非c点

所以对于每一个围栏四边形顶点都将其本身编码和相邻的8个编码加入计算过程.

在经过./tools/hashing.py的计算后,我们可以得到两个临时结果:

1. 每一个自行车订单对应的geohash编码

2. 每一个围栏对应的36个可能重复的geohash值(每一个顶点都有9个geohash值,但如果围栏四个顶点在在同一个geohash编码格中,那么这36个geohash就是9个geohash重复四次), 将其反向映射,得到每个geohash编码关联的围栏FID(FID是根据index加的,和FENCE\_ID作用一致但是更加简短),使用{编码:[FID, FID…]}格式的字典(在其后表达为grid\_fence,在find\_nearest.py中有体现)进行存储.

单个停车围栏相关的最近点生成代码如下(使用python-geohash库)

1. **def** point\_grids(point):
2. """
3. :type point: Tuple or List, format: (Latitude, Longitude)
4. :rtype: List, related geohash grids to current point
5. """
6. center = geohash.encode(point[0], point[1], 7)
7. # 将center扩展为九宫格,避免"错误最近点"现象出现
8. nears = geohash.expand(center)
9. **return** [center] + nears
11. **def** fence\_grids(points):
12. """
13. :type points: List[(Float, Float)], with 4 tuples: p1, p2, p3, p4: position or 4 corner points of a polygon(fence)
14. :rtype: List, related geohash grids to current bike fence.
15. """
16. p1, p2, p3, p4 = points
17. p1\_g, p2\_g, p3\_g, p4\_g = point\_grids(p1), point\_grids(p2), point\_grids(p3), point\_grids(p4)
18. **return** list(set(p1\_g + p2\_g + p3\_g + p4\_g))

##### 3.2.1.3 中心点, 面积, 长宽计算

根据顶点坐标计算围栏中心点, 面积, 长宽

##### 3.2.1.4. 最近停车点的查找(./tools/find\_nearest.py)

将每一个自行车点位对应的geohash作为key,在字典中查询对应的围栏FID,并计算该点位与这些围栏的中心点距离,找出最小值所对应的FID.使用该FID的顶点坐标,计算点位与围栏边界的距离(若在围栏内则为0), 记为DISTANCE; 计算点位与最近围栏中心点的距离,记为CENTER\_DISTANCE

##### 3.2.1.5 加入日期字段

原始的gxdc\_dd.csv包括12.21-12.25五天的数据.我们注意到12.25是西方圣诞节, 在这一天的共享单车流向或许会与另外几天不同,所以在自行车数据中单独加入”日”维度,命名为DAY,方便后续可视化和数据分类

##### 3.2.1.6 数据完善小结

**对于自行车数据**

新增字段: GRID, BID, NEAREST\_FENCE, DISTANCE, DAY

GRID: 自行车点位对应的geohash

BID: 根据index形成的编号, 使每条数据都具有独立性

NEAEREST\_FENCE: 最近的停车栅栏对应的FID

DISTANCE: 到最近停车围栏(边缘)的距离

DAY: 当前所属的日(21, 22, 23, 24, 25)

部分数据的NEAREST\_FENCE 和DISTANCE距离为-1,意味着该点位周围153-306米内没有任何停车栅栏.在实际生活中可将其作为”乱停车”数据的一部分

**对于围栏数据**

新增中心点坐标LATITUDE, LONGITUDE. 将原有的五个坐标解包为10个维度(LATITUDE\_0, LATITUDE\_1等),方便读取

新增FID字段(type: int),功能同FENCE\_ID

新增ROAD字段:当前停车点所属的街道名

新增LENGTH, WIDTH字段:当前停车点长宽

新增AREA字段:当前停车点的面积

#### 3.2.2 数据清洗

##### 3.2.2.1 锁具异常行为检测及异常数据清理

使用Microsoft SQL Server进行数据预览时, 发现共有4136辆车存在连续锁车, 连续开车等锁具异常(原因可能时锁具损坏,电量不足等)现象

以BICYCLE\_ID为001ca978928d0e762aaede9118e3c7e6的自行车为例:

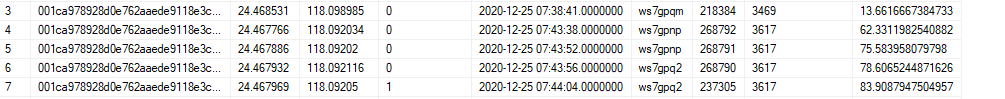


图3.2.2.1.1 开/关锁异常数据示例

7时43分的四个连续开锁数据经matplotlib可视化后, 可以看出该车是在异常数据发生时是持续移动的,而从静止状态转为移动状态的起始数据点是第一个开锁数据 (7时38分)故可以去除由7:43:38开始的开锁记录(保留7:38:41的记录和7:44的关锁记录),并将该车加入异常列表

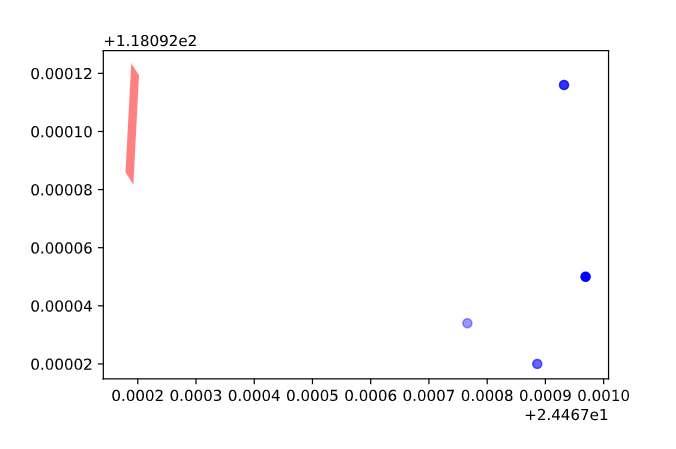


图3.2.2.1.2 异常数据可视化.红色方框为编号3617的围栏,自行车轨迹点按时间先后顺序由浅至深

同理,对于连续的关锁数据,仅保留最后一条,即由移动状态转为静止状态的数据.

车体物理异常状态也可用SQL进行检测:若开-关间隔过短(在SQL中为min<x,x可以是1, 2, 3等较短的分钟数)且多次进行, 则该车有可能因损坏而被用户放弃,需要进行检查.

##### 3.2.2.2 去除7点前与9点后的数据

经数据预览发现原始自行车订单数据中存在的订单时间为早6时至10时, 而本次比赛需要研究的是早高峰时段(7:00-9:00)之间存在潮汐现象的停车围栏,故使用SQL直接移除7:00之前和9:00之后的数据

### 3.3 数据安全保障

## 4. 算法分析

数据预处理即数据的完善与清洗详见3.2.1数据完善与3.2.2数据清洗

在此阶段由于数据显示需求量与代码量的减少和对数据图形化需求的增加, 我们将Python Script + SQL进行的数据挖掘与清洗转至基于Python的Jupyter Notebook以便进行可视化.

经过3.2.1和3.2.2的预处理后,我们得到了两个csv文件: bikes\_data.csv 与 fence\_position.csv

两个csv的字段与字段说明如下表所述

|  |  |  |  |
| --- | --- | --- | --- |
| 表名 | 字段名 | 数据类型 | 字段说明 |
| bikes\_data.csv | DAY | INT | 数据所在的日(21-25) |
| BID | INT(UNIQUE) | 唯一订单代号 |
| BICYCLE\_ID | STRING | 自行车ID |
| LATITUDE | FLOAT | 当前纬度 |
| LONGITUDE | FLOAT | 当前经度 |
| LOCK\_STATUS | INT(BOOL) | 开/关锁状态(0开1关) |
| UPDATE\_TIME | STRING | 数据发生时间 |
| GRID | STRING | 坐标geohash编码 |
| NEAREST\_FENCE | INT | 离坐标点最近的围栏编码 |
| DISTANCE | FLOAT | 离最近围栏的距离 |
| MKTIME | FLOAT | 数据发生时间离era的秒数 |
| FENCE\_ID | STRING | 围栏名 |
| fence\_position.csv | LATITUDE\_X | FLOAT | 围栏顶点经度（X：0-4） |
| LONGITUDE\_X | FLOAT | 围栏顶点纬度（X：0-4） |
| LATITUDE | FLOAT | 围栏中心点经度 |
| LONGITUDE | FLOAT | 围栏中心点纬度 |
| ROAD | STRING | 围栏所在路名 |
| AREA | FLOAT | 围栏面积 |
| FID | INT | 围栏唯一标识（作用同FENCE\_ID） |
| LENGTH | FLOAT | 围栏长度 |
| WIDTH | FLOAT | 围栏宽度 |
|  |  |  |

### 4.1 数据的进一步挖掘与可视化

我们在前面之前提出过疑问:12.25为西方圣诞节, 那么这一天的早高峰时段的开关锁地区分布是否会与其他日期不同? 为此我们使用了matplotlib进行了简单可视化

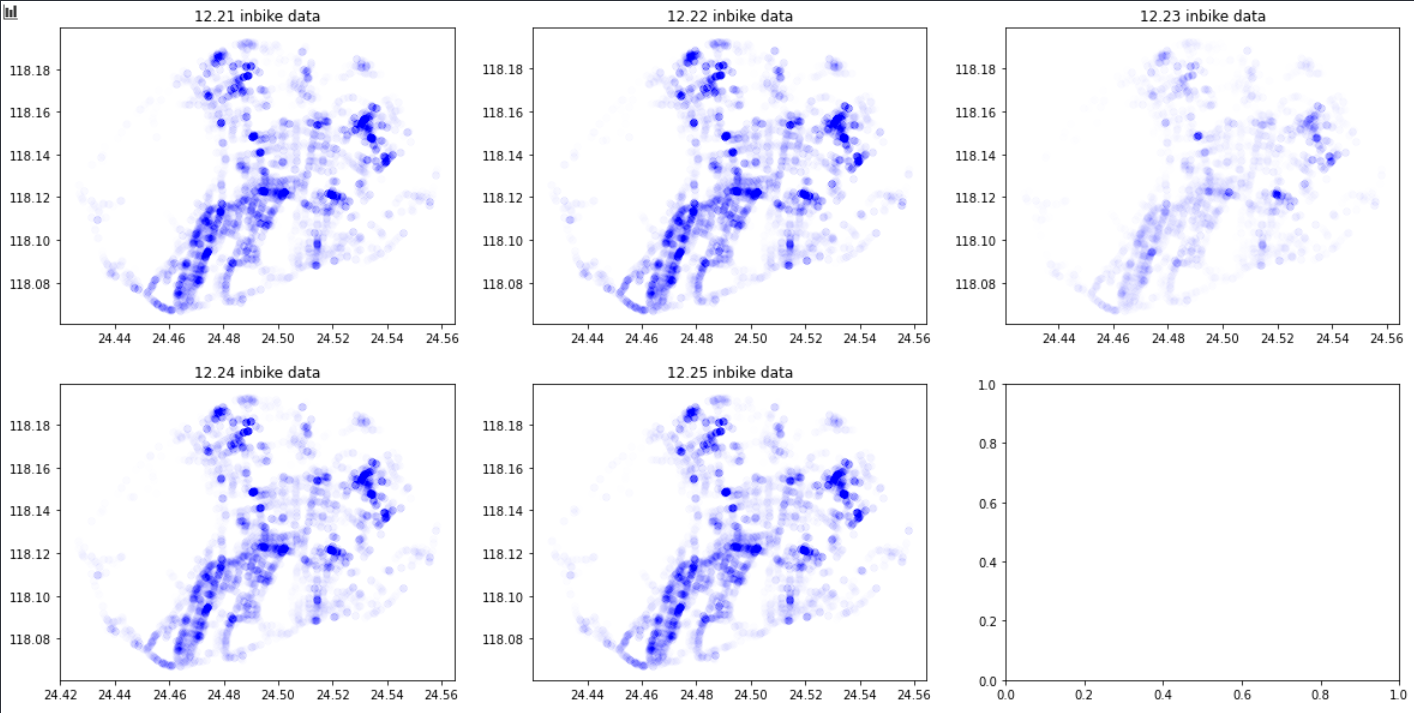


图4.1.1 12.21-12.25早高峰关锁地点分布

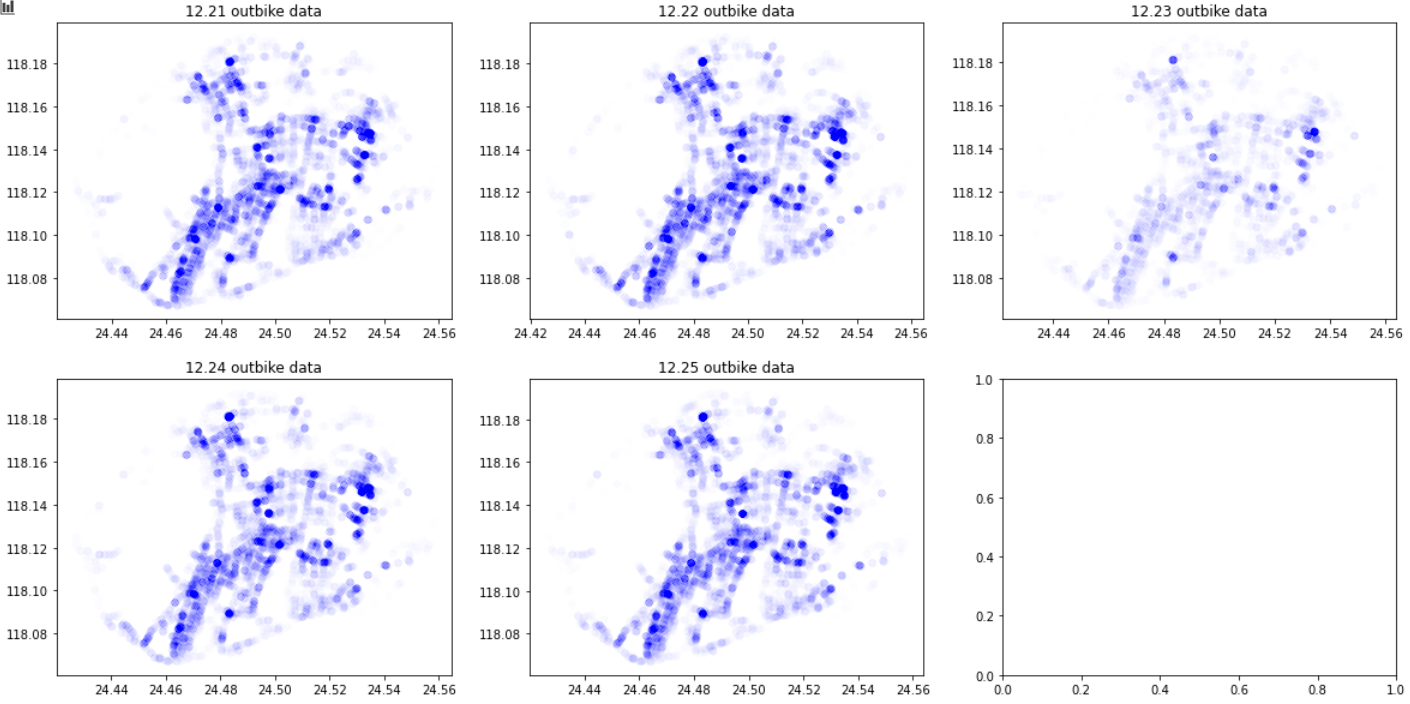


图4.1.2 12.21-12.25早高峰开锁地点分布

从图中可以看出,12.21, 12.22, 12.24, 12.25四天开/关锁数据点分布都十分类似,而12.23自行车使用情况明显较少.为了防止某一地区四天自行车滞留量为负,而剩下的一天为一个极大的数字导致的将该点误认为潮汐点的错误发生,我们在后续将停车围栏聚类后对应地加了一个ACTIVE\_DAYS(有多少天流量为正)维度观察这一个簇中的停车围栏是否存在该种错误.

### 4.2 五种潮汐点识别方案

#### 4.2.1 方案一. 基于围栏大小设置不同”乱停车”容忍度, 根据滞留车辆数量从高到低排序停车区域的方法(./tools/method1.ipynb)

在观察fence\_position.csv时, 我们发现不同围栏的大小差距非常之大. 而以大型停车点围栏边界为基础, 延伸相同长度之后覆盖的额外范围是远远大于小型停车点的.

以 1 \* 1 和10 \* 10 的围栏为例:

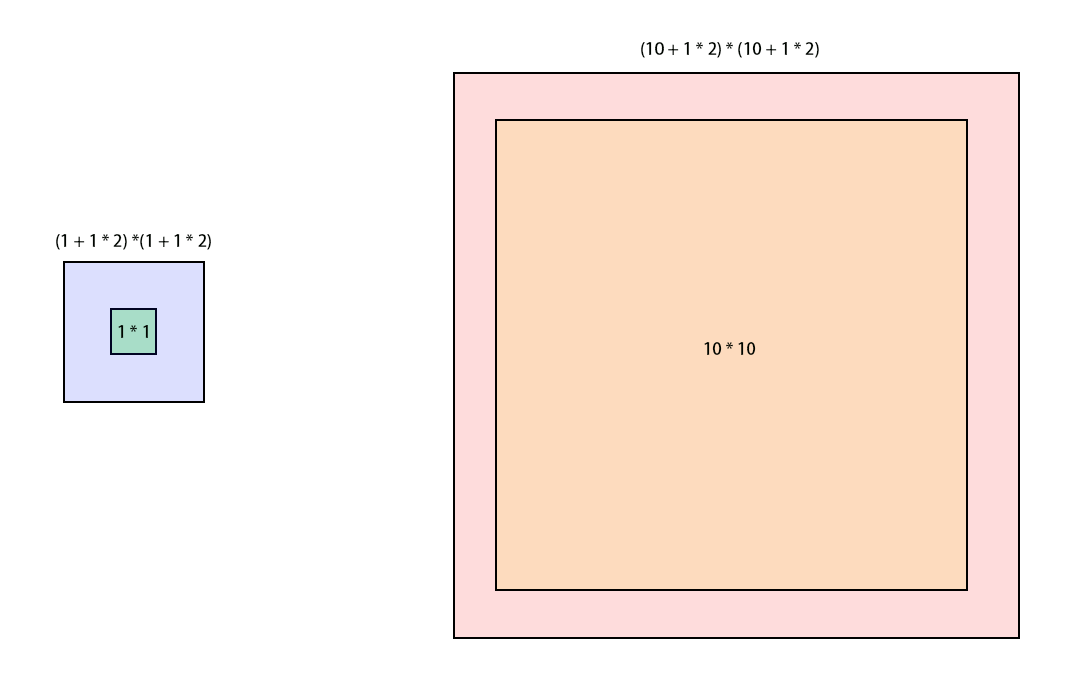


图4.1.3 1 \* 1 停车围栏和10 \* 10停车围栏分别往外扩张1米

1 \* 1围栏向外扩张1米后,额外增加的面积只有9 – 1 = 8 平方米,而 10\* 10围栏向外扩张1米后,额外增加的面积变为144-100=44平方米

所以对于较大的围栏,应减小允许的自行车停放在围栏外的距离.

基于此理论, 为了方便分类,我们加入了CENTER\_DISTANCE维度: 车辆与围栏中心点的距离

对于bikes\_data.csv中的所有非异常停车点, 我们将其拆分为可计入停车围栏流量的点和无法计入的点.

在实际的自行车调度中, 对于中小型围栏,调度人员步行搜集车辆的活动范围几乎一致,而对于大型停车点,调度人员更趋向于使用驾车+步行的方式收集该停车点附近所有的自行车.

所以,我们可将”无法计入”(也可称乱停车)点定义为: 离停车围栏中心点50米外(针对中小型停车点),或离停车边界20米外(针对大型停车点)的点

1. bikes\_valid = bikes[((bikes['CENTER\_DISTANCE'] <= 50) & (bikes['DISTANCE'] != -1)) | (bikes['DISTANCE'] <=20) & (bikes['DISTANCE'] != -1)]
2. bikes\_outliers = bikes[~bikes['BID'].isin(bikes\_valid['BID'])]

代码中DISTANCE为-1的点是无法被geohash网格覆盖到的,即距离停车点边缘最少有153米的开车/锁车点, 显然属于”乱停车”范畴.