# 基于时空数据分析的决策支持报告

### 概述

本人完成了出租车经营策略研究中的基础分析、原地待命策略、低速巡游策略和选作中的城市POI分析与推荐。工程代码在压缩包的DSpj文件夹内。相关数据在data文件夹内。

## 环境说明

GUN GCC Compiler

Code::Blocks 13.12

Win10 64bit

## 算法与数据结构介绍

### geohash

将经纬度hash成一个long long。

算法流程如下，比如(120,30) 范围精度是0-180 唯独是0-90

120>=(0+180)/2 故h[0]=1 30<=(0+90)/2 h[1]=0

120<=(90+180)/2 h[2]=0 30>=(0+45)/2 h[3]=1

120>=(90+135)/2 h[4]=1 30<=(22.5+45)/2 h[5]=0

以此类推二分得到每个坐标唯一的hash值

因为用了64位，所以可以保证每个点的hash是唯一的。所以用这个确实可以查到最近的点，唯一的cheat是在分割线附近，比如经度90和经度89.9999999会有很大差别。但在线上的点很少，而且这种情况依然能查询到另外一边的最近的点，所以我觉得geohash非常适合这个pj。

这个算法的优点是代码简单且容易维护，准确度高速度快。缺点是分割线上不能保证最优解。但大部分情况有最优解，极小部分有较优解。

验证发现在前四十层分割线上1e-5范围内只有437个点

空间复杂度O(nlogn)

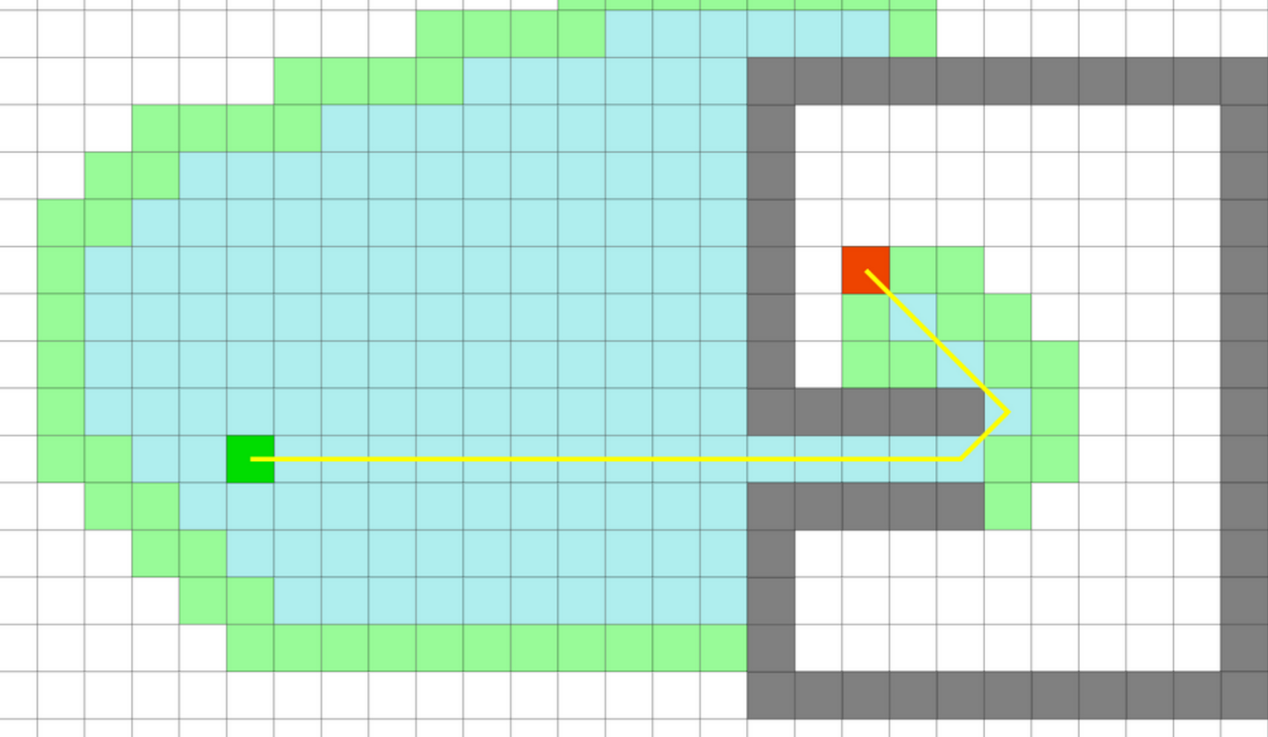
查询复杂度O(logn\*E) E为从最hash值最接近的E个点找最近点

### A\*搜索

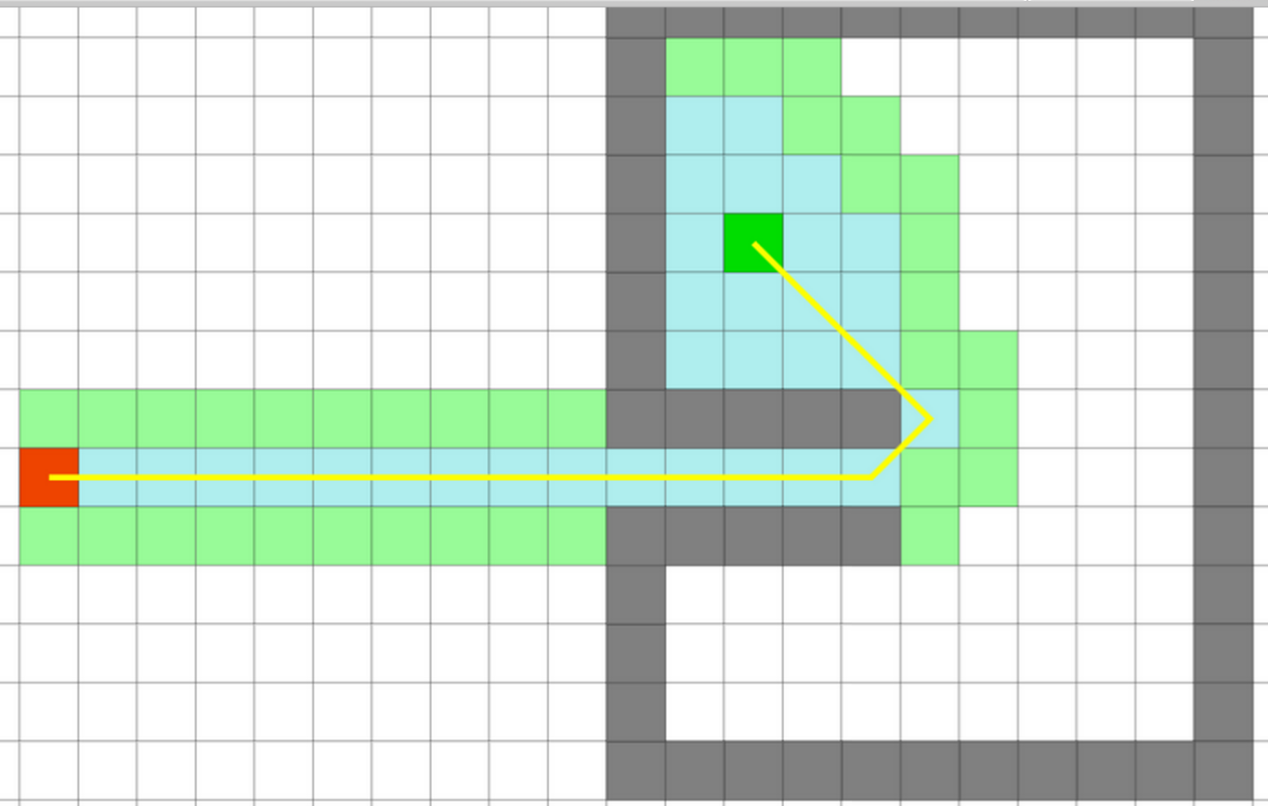
A\*搜索用于求指定起点和终点的最短路。估价函数采用当前点到终点的欧几里德距离。

实现上用优先队列来弹出 到起点距离+估价值 最小的点，之后把所有相邻点插入优先队列。

这里给大家介绍一种**双向A\***的算法。



对于这张图，从起点到终点要跑大量的无关格子，但如果从终点到起点呢？



可以看出这种数据终点到起点跑得非常快。

这种数据大概属于起点附近的图比较稠密，终点附近的图比较稀疏。

这里也可以看出如果两个点在河的两端，桥在较远的地方，A\*算法会退化成类似BFS。这也提醒我们如果追求速度的话可以基于图的类型设计不同的算法。

还有可以用动态调整的策略。

我们发现离目标点远的时候算短距离很重要，近的时候就没那么在意距离了。

f(s)=real(s)+predict(s)\*w(s) real是当前距离，predict是估计距离，w是权重，离目标越近权重越小。

另外我们发现对于距离近的点对dijkstra很快 远的很慢。考虑双向搜索中一边用A\*，另外一边在终点附近用dijkstra?

还有就是可以从大量经过起点和终点的订单轨迹中提取出关键点，从关键点往两个方向搜索最短路。

### 计算几何

用叉积和面积等基础知识求点到线段的距离。但经度和纬度并不是严格意义上的二维平面。需要经过转换。

## 平衡树

大量的数据使用平衡树维护，比如根据hash值来查找点。实现上直接使用了stl的map。

## 排序算法

快速排序。

## 一些功能的实现

### GPS偏移

pair**<**int**,**int**>** findNearestRoad**(**pair**<**double**,**double**>** p**);**//找到离p点最近的边(a,b)

根据p点的hash值在map中查询hash值最接近的20个点。遍历这20个点的所有边，找到离p点最近的边。返回边两头的点编号。

### 坐标转换

理论上应该把经纬度坐标映射到x,y坐标。以确保点到线段距离的正确性。实践中发现经纬度乘上某些常数映射后与直接把经纬度当作x,y坐标影响肉眼不可见。

### 司机找最近订单

int getNearestorder(pair<double,double> pos,Road &road,Order &order);

给出位置，找最近的订单。实现上先将司机的位置哈希得到\_hash。在所有当前订单起点哈希值构成的平衡树中查找\_hash.将\_hash附近的40个订单取出。按照他们与司机的距离排序。返回最近的订单。

### 维护当前订单

把订单按时间排序，每次单调的把当前时刻前的订单插入map。（key为起点位置，value为订单编号）

### 模拟等待策略

void stayStrategy**(**Order **&**order**,**Road **&**road**);**

先得到司机的初始位置，初始时间。然后查找最近的订单，如果在三公里内的话，接上乘客并驶向目的地。否则等待300秒，再查询最近的订单。直到时间超过这个时段。

### 模拟低速巡游策略

void lowStrategy**(**Order **&**order**,**Road **&**road**);**

先得到司机的初始位置，初始时间。然后查找最近的订单，如果在三公里内的话，接上乘客并驶向目的地。否则找到司机位置和最近订单的最短路。每走过一条路并且距离上次查询时间大于300秒，则查询最近订单，若小于三公里，则接单，否则继续跑。

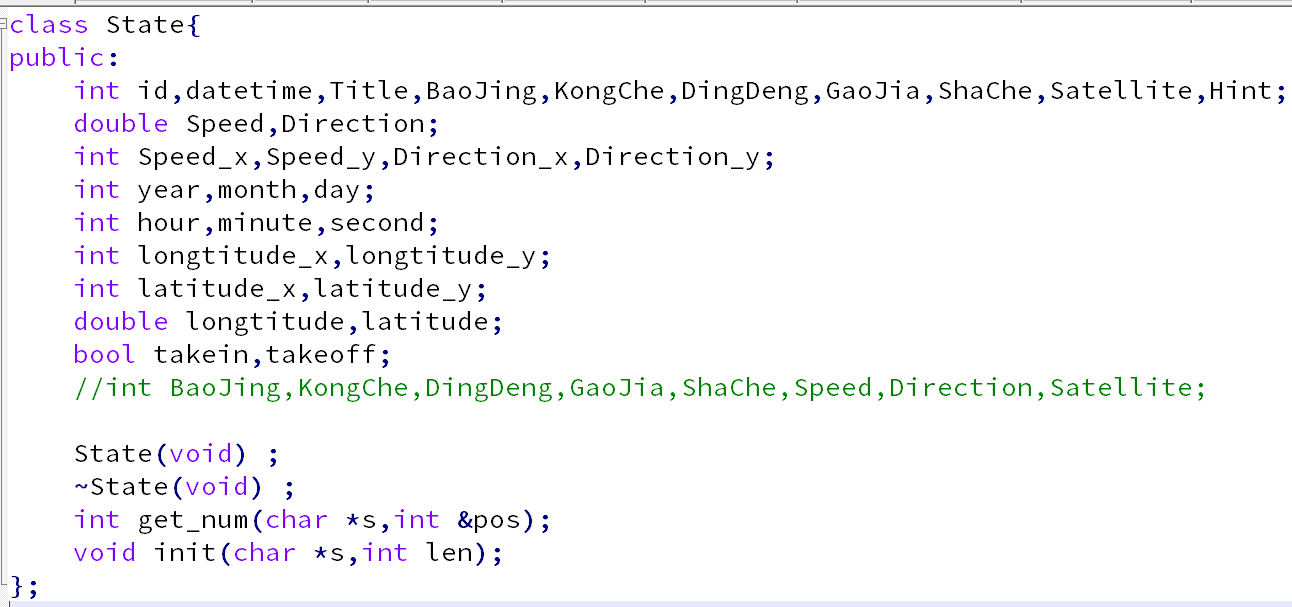
### 路口对接

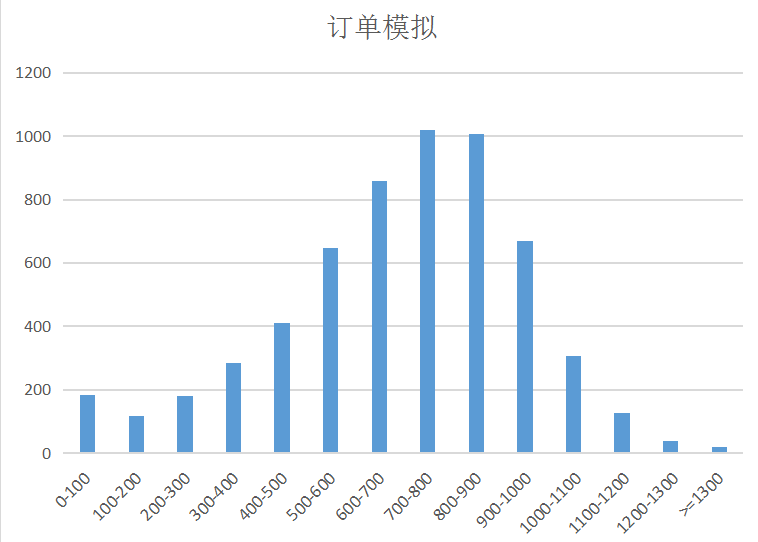
路网的构建中有些路是同一个端点，但经纬度有少许偏移。做法是在建立路网的过程中把点的坐标\*10000去整看待来辨别是否相同。

## 必做部分

### 模拟司机订单

这个任务涉及到大规模的数据处理，我们用遍历字符串的方式把所有提供的数据都存在了数据类型State中。对于每个司机，他的每个状态之间的距离就是他的行驶距离，每次顶灯亮起的状态则是载客状态，可以求得载客距离。之后输出每个司机的收入，支出，利润即可。





x坐标是收入范围，y坐标是司机数目，可以看出收入在600-700

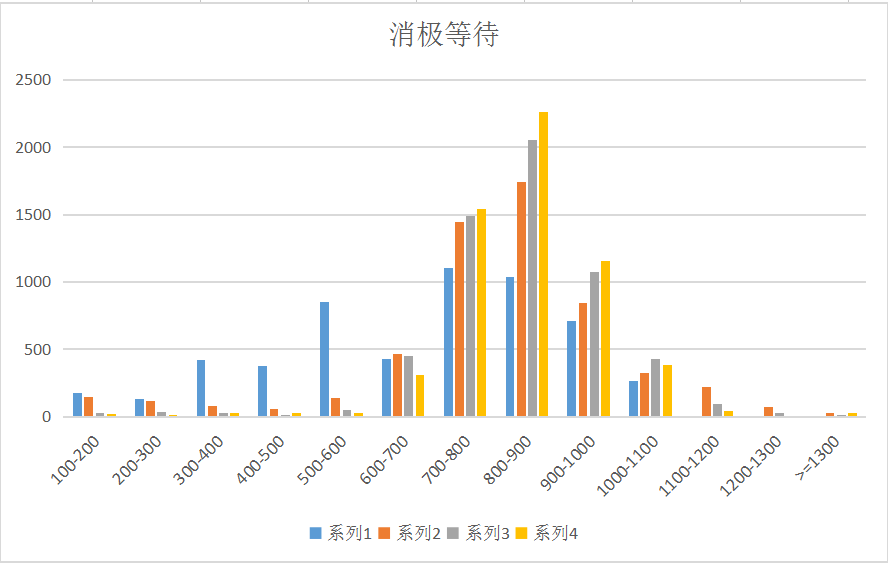
### 模拟原地待命策略

先把所有司机在每个时段的第一个时间点的位置提取出来。

如上文所提到的实现即可。

注意要删除已经接到的单避免重复刷单。（如果点对很近的话可以刷起步价）

注意对于不连通的点对，直接直线到达。

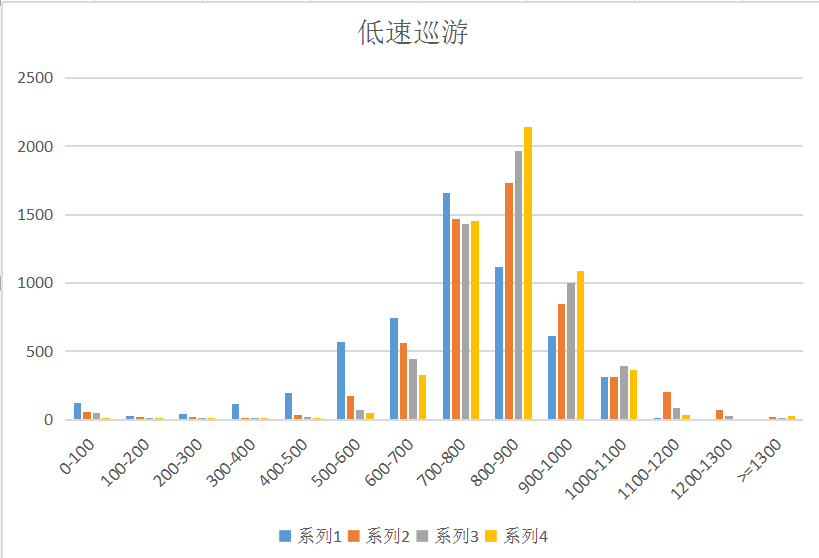


这是四个时段的收入分布

### 模拟低速巡游策略

如上文所提到的实现即可。

注意如果和最近订单之间没有路的话，删除该订单并换一个订单。



## 五、选作部分（城市POI分析与推荐）

### POI估价

通过geohash用类似找最近订单（找最近的点）的方法找到poi附近的所有上车点和下车点，如果在两公里以内则估价+1，在500以内估价+3。

实践中发现如人民广场这样的餐厅密集的地方无法把这些餐厅做一个区分。考虑把人流量除以餐厅数量，发现效果一般。因为在人民广场附近你还是要推人民广场的餐厅的吧。

### 推荐估价

void POI**::**Introduce**(**pair**<**double**,**double**>** p**,**Road **&**road**)**

通过geohash找到离查询点p最近的100个poi。将距离和他们的估价作为参数，得到一个score。把score最高的20个poi推送出来。

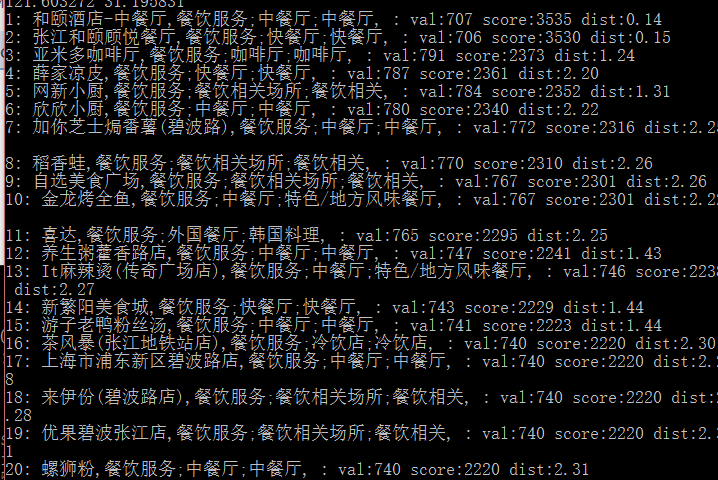
参数我调了一会儿大概就是500米内和两公里内有小幅加成。因为这个查询数量较少可以把poi数扩展到10000个甚至100000个。



### 实例分析

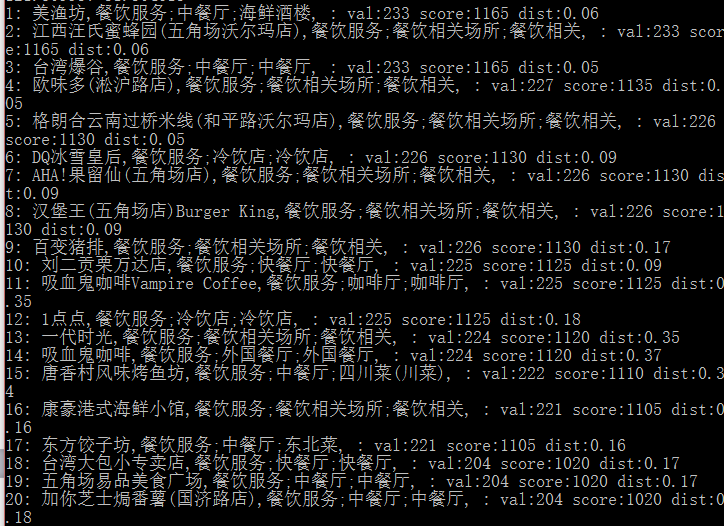
我们来选取几个例子来推荐餐厅。





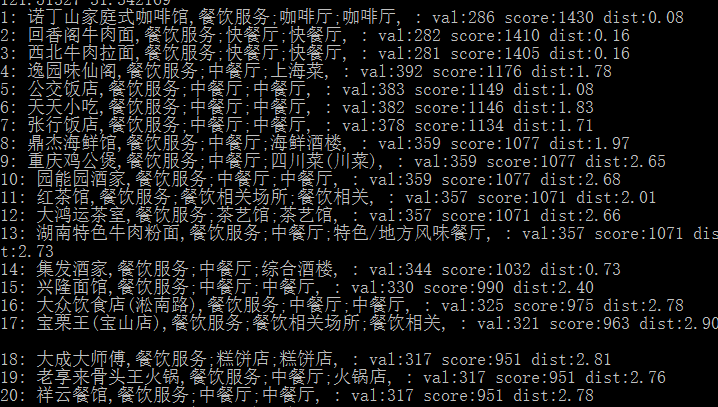
嗯这些都是张江附近人气比较高的餐厅.





可以看出邯郸不愧是市中心。





江湾近距离的餐馆还是比较少的。



