# 共享单车的成本分析与网络分布

## 摘要

## 关键词

## Abstract

## Keyword

## 目录

## 引言

### 研究背景

共享单车是指企业在校园、地铁站、公交站、居民区、商业区、公共服务区等提供自行车单车共享服务，是一种分时租赁模式。共享单车具有操作方便、低碳环保等优点，解决了“最后一公里”出行的问题，并在一定程度上缓解了交通问题。

据不完全统计，全国目前有共享单车公司30多家，累计投放车辆超过1000万辆，注册用户超过1亿人次，累计使用人数超过10亿人次[[1]](#footnote-1)。然而在许多城市，共享单车的过量投放挤占了人行道和自行车道等公共空间，不仅影响了行人的通行，加重了城市的交通负担，而且影响了城市环境的美观。同时，过量投放的共享单车也会给公司增加运输和维护的成本。本文尝试对共享单车的各种成本进行假设和分析，并在此基础上建立数学模型，以期找到一种既能满足用户需求，又能减少公司运营成本的解决方案。

### 研究内容

本文的主要工作分为三部分。第一部分对共享单车涉及到的各类成本进行分析，并建立最简单的数学模型，研究理想情况下的车站的分布和成本。第二部分以此为基础，考虑用户密度的变化及车站自行车数量变化带来的成本变化，以建立更全面的模型。第三部分则应用第一、二部分的假设，结合实际生活中的经验，在仅有离散的用户概率密度函数的情况下，研究计算使成本最小的浮动车站的位置的算法，并使用计算机进行编程和模拟，结合实际分析算法的正确性。

### 论文组织结构

论文主要分为六个章节：

第一章简要介绍研究的背景和研究的内容。

第二章介绍共享单车中涉及到的成本。

第三章计算一个用户均匀分布的车站固定的理想平面上的共享单车成本。

第四章计算一个用户连续分布的车站固定的理想平面上的共享单车成本。

第五章计算一个用户概率分布的车站可移动的理想平面上的共享单车成本，并结合实际地图进行分析。

第六章对整篇论文进行总结，并指出未来的可研究方向。

## 共享单车与车站网络的成本分析

为了能够尽可能准确的建立数学模型，本章研究和分析共享单车与车站涉及到的所有成本，并分析哪些是主要影响因素，并进行归纳。

共享单车自身的成本应该包括购买自行车的成本。目前，我国各个共享单车公司的自行车成本均不相同，从仅包含基本功能、造价最低的200元，到安装智能锁、GPS定位等造价高昂的2000元，每家公司都有不同的策略。考虑到共享单车的造价直接的影响其投放成本，对车站的车辆数量有着直接关系，因此应该列入考虑范围。

相比于造价，共享单车的维护成本则占据了其生命周期中成本的主要部分。一辆200元的共享单车，其一年的维护成本可能高达1000元。本文考虑实际生活中，共享单车公司对于自行车的维护并不是针对每一辆车，而是针对每个车站的全部车辆，因此将其划入与车站的自行车数量相关的维护成本的一部分。

共享单车车站间的运输车站成本的一部分。在大型车站间，通常使用货车或面包车进行运输，而在小型车站间则使用三轮货车。考虑实际中，三轮货车一次可以运输十几辆自行车，数量相当于一个小型车站的自行车数量，因此本文将运输成本抽象为一个与距离相关的成本。

车站本身也存在一定的维护成本，同时车站本身占用的社会资源也应该考虑在内。目前很多城市已有计划限制或打算限制共享单车的投放，可以预计未来可能存在针对车站占用公共资源进行收费。本文考虑对每个车站都存在一个固定的成本，这部分成本包含了对于车站的维护，包括车辆摆放、人员雇佣等成本，也包含占用的社会成本。

本文将用户的需求也作为成本的一部分来考量。当用户周围没有车辆可以使用，或用户需要移动很长的距离才能获得一辆可以使用的车辆时，用户对于公司必然有一个负面印象。因此研究车站的分布时，用户的移动距离必须要作为一个成本进行考虑。

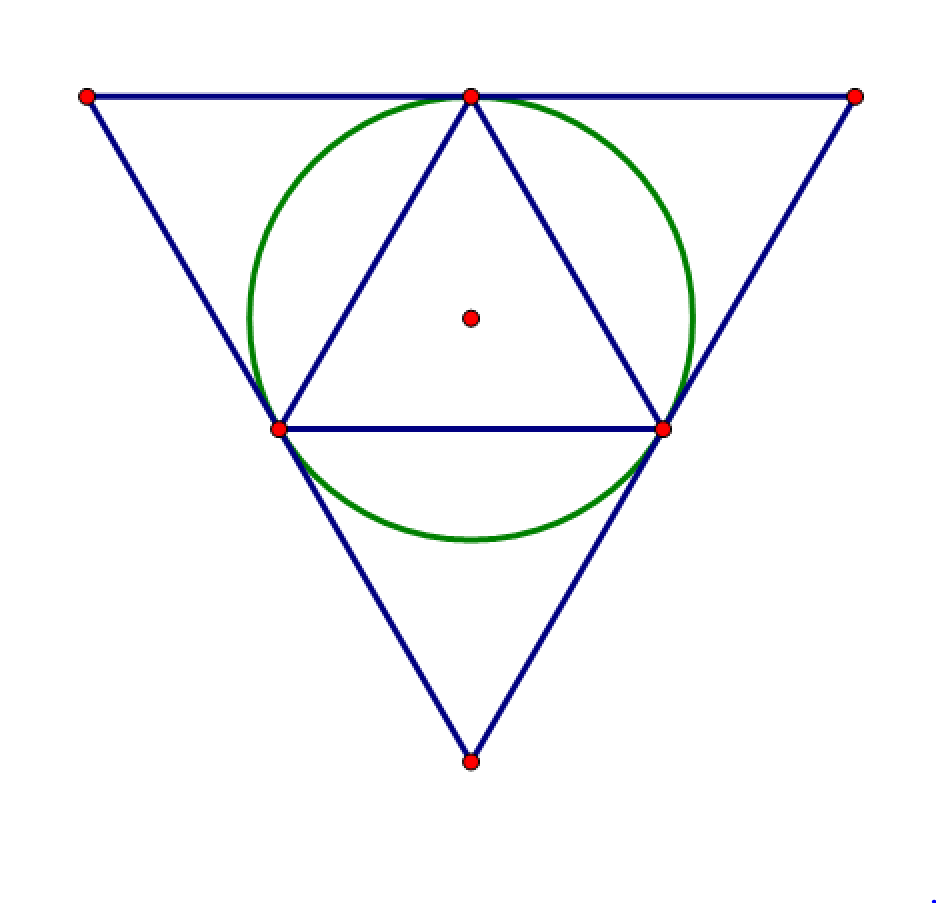
## 车站网络覆盖的近似图形选择

在开始分析简单的车站网络前，首先要选择每个车站覆盖的近似图形。假设每个小型车站的覆盖图形为以R为半径的圆形。但使用圆形覆盖时，两个圆形相交部分的用户必然按两交点连线分割，因此圆形的车站覆盖必然转化为某种凸多边形覆盖。

已知可以密铺平面的正多边形有三角形、正方形和六边形。定义车站的使用面积为用户会选择该车站的点组成的多边形的面积。定义车站利用率为车站的使用面积除以车站的覆盖面积，则选择高利用率的图形对降低成本至关重要。因此在接下来的小节中，本文将分别计算三角形，正方形和六边形的车站利用率。

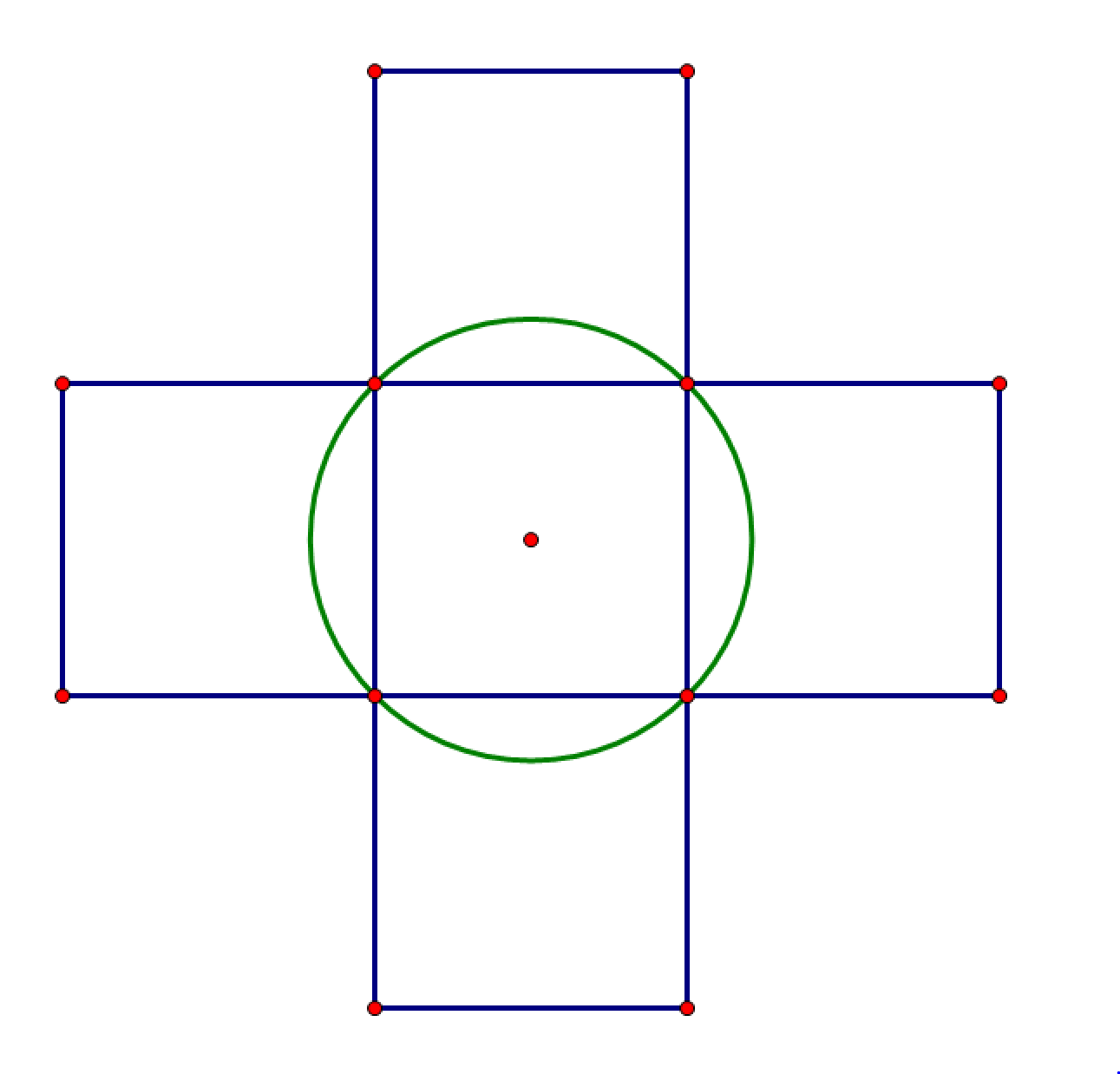
### 三角形

当车站覆盖形状为正三角形时，如右图所示，此时车站的覆盖面积为，而实际使用面积为



根据定义，三角形的车站利用率为：

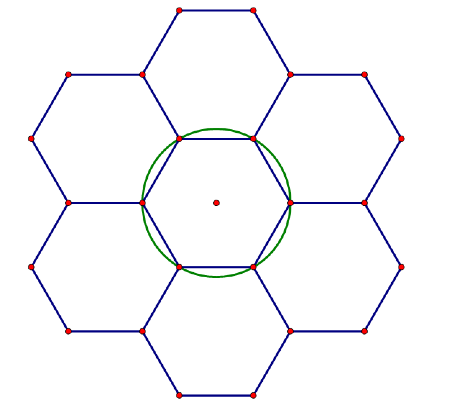
### 正方形



当车站覆盖形状为正方形时，如右图所示，车站的覆盖面积仍然为，而实际使用面积为

则根据定义，正方形的车站利用率为：

### 六边形

当车站覆盖形状为六边形时，如右图所示，车站的覆盖面积依然为，而实际使用面积为

则根据定义，六边形的车站利用率为：

根据以上计算，六边形的车站利用率最高，故在下文中，车站使用六边形进行近似计算。

## 理想环境下的车站网络成本

在本章中，本文假设所有用户是均匀分布在一个平面上的，且用户会选择距离自己最近的一个车站出行，目的地也是一个车站。同时假设，对于一个大型车站的影响范围之内的所有小型车站，其所有运输成本均视为从大型车站发出。基于这些假设，本文开始推导理想环境下的车站网络成本。

### 均匀密度下的用户距离成本

用户距离成本即用户从出发地步行至自行车车站的成本。由于小型车站覆盖的面积很小，因此这里假设用户距离成本与距离成正比关系，即：

其中i为用户编号，j为车站编号，指用户i和车站j的距离，c为参数，为用户距离成本。

由于假设用户是均匀分布在平面上的，设用户的分布为

其中M为用户总数，p为用户概率密度。

 对于六边形车站，假设其中心为O，各顶点为A到F，如右图所示。则车站总的用户距离成本为三角形ABO的成本的六倍。对于正三角形ABO，设AO、BO上的点C、D，且CD平行于AB，则CD上所有点到O点的距离之和等于正三角形CDO的面积。设正三角型CDO的边长为x，则三角形CDO的面积为：

三角形ABO内所有点到O的距离之和等于三角形OAB内所有平行于AB的线段上的点到O的距离之和，即：

三角形ABO内所有点的用户距离成本为：

则车站O的用户距离成本为：

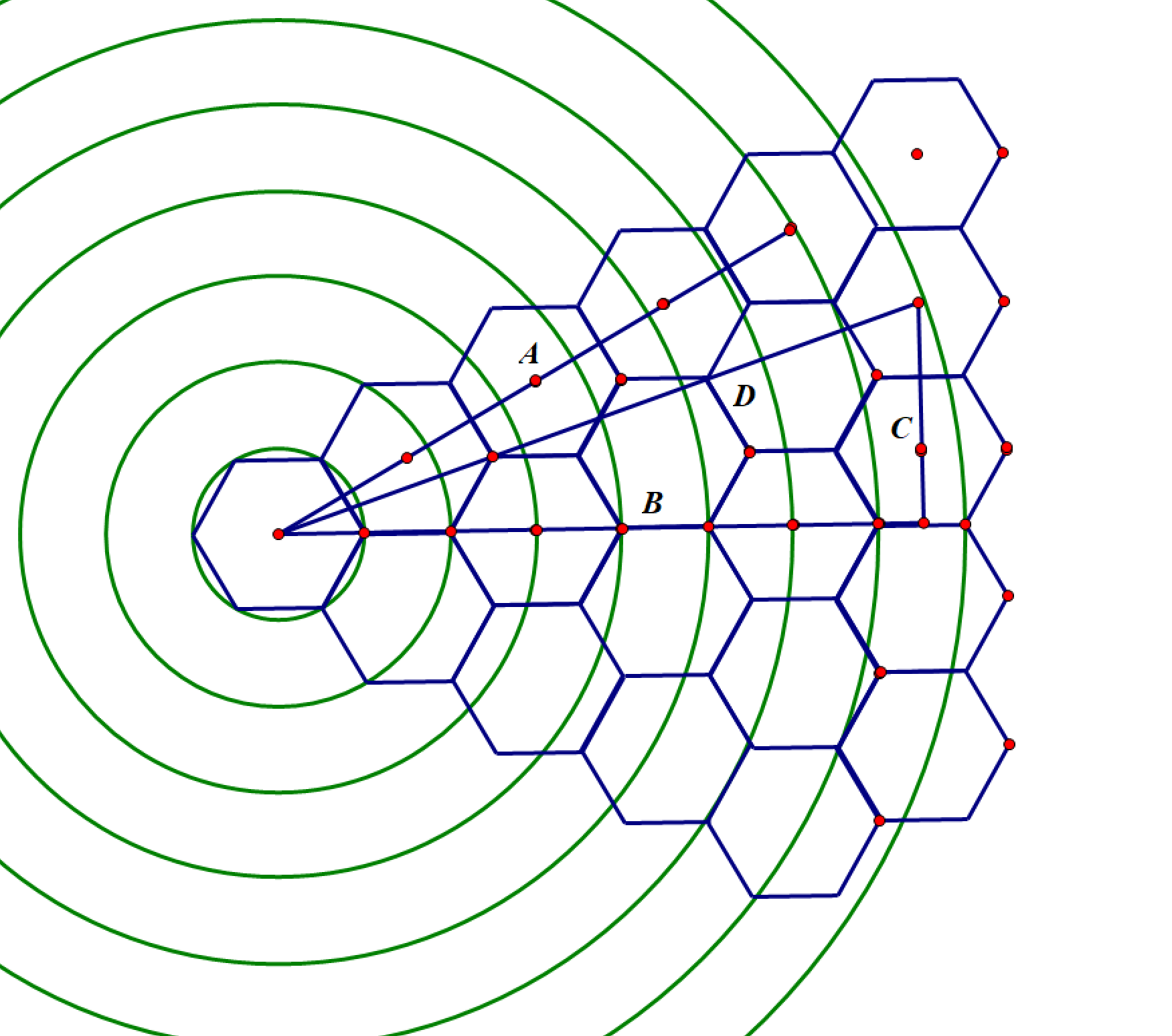
### 运输成本

由于共享单车会在不同时段出现“潮汐”现象，即不同地区，在不同时间段车辆会向不同车站集中。如住宅区附近，早上共享单车会向地铁站集中，而晚上则从地铁站分散。基于第二章的假设，即小型车站的运输成本只与小型车站到大型车站的距离有关，则小型车站i的运输成本为：

其中为小型车站到大型车站的距离，b为参数。

### 大型车站覆盖下的小型车站数量

假设一个大型车站的覆盖半径是L。根据作图，列出L与车站数的对应关系如下表。



|  |  |
| --- | --- |
| L | 车站数 |
| 1R | 1 |
| 2R | 7 |
| 3R | 13 |
| 4R | 19 |
| 5R | 31 |
| 6R | 43 |

再往下计算几位后并没有发现什么规律，因此作图，分析规律，进行计算。

由于六边形的对称性，因此只需要分析60度方向能够覆盖的六边形，在经过一定的去重处理，即可得到最终结果。而对于这60度方向，可以发现六边形的构成了一个新的大等边三角形。对于第x竖排的六边形，这一竖排与中心所构成的正三角形边长和高分别是

因此当时，整个正三角形所包含的的六边形全部被圆所覆盖。

对于不能被圆完整覆盖的列，首先计算其是否在六边形内。当时，该列处于圆内，即

设此时以L为半径的圆能够覆盖的最大列为第列，则

假设，由于均为正整数，所以

对于满足条件的列，计算其高B，利用勾股定理，使用L和B计算其最高的六边形中心点距离C，进而计算出满足的六边形数量。

由于其计算过于复杂，固不在此过多计算，仅使用小型车站到大型车站的距离 表达，而不对其和进行求解化简。

### 均匀密度下的车站网络成本

根据第二章的假设，每个小型车站还有固定成本F，用于维护车站。因此小型车站的总成本为:

而一个大型车站覆盖的总成本为：

其中N为大型车站覆盖下的小车站数量，其求解方法已在3.2.3中详述。

### 非均匀密度下的车站网络成本

在以上几节中，本文假设用户是均匀分布的。在实际中，用户显然是按某种概率密度分布。本章将讨论在用户密度以某种概率密度函数分布的情况下，单个大型站点的成本分析。

假设用户的分布为：

其中M为用户总数，为平面上的用户概率密度函数。

类似于3.2.1，对于六边形车站O，车站总的用户距离成本为三角形ABO的成本的六倍。对于正三角形ABO内的任意一点，其用户距离成本为：

则三角形ABO的用户距离成本为：

则车站O的用户距离成本为：

小型车站O的总成本为：

则一个大型车站覆盖下的所有小型车站成本为：

## 浮动车站网络的算法与模拟

在第四章中，本文讨论了理想环境下的车站网络成本。但在现实环境中，统计结果很难用一个或多个函数近似，但可以得到用户的分布，并用这个分布计算一个小分区的用户概率密度。同时，现实中共享单车的车站并非是按理想六边形分布的：对于用户密度大的地区，会有更多的车站提供服务，一般的大车站也会在这个地区；而对于用户密度小的偏远地区或小街小巷，车站则非常少。本章讨论的内容是如何根据用户密度概率计算浮动的车站，并使用计算机进行模拟分析。

### 公式总结

根据第四章的结论，对于一个大型车站下覆盖的某一小型车站i，其成本包含该车站的全部用户的距离成本，运输成本，及固定成本F。即：

同时考虑到车站影响范围不会太大，假设每个用户的距离成本正比于用户最近的车站的距离，即：

其中i为用户编号，j为车站编号，指用户i和车站j的距离，c为参数，为用户距离成本。

该车站总的用户距离成本为：

其中m为该车站的用户编号，n为该车站的用户总数。

小型车站的运输成本与该车站和大型车站的距离成正比，即：

其中为小型车站到大型车站的距离，b为参数。

代入小型车站i的总成本：

则一个大型车站覆盖的全部小型车站成本为：

其中N为小型车站总数。

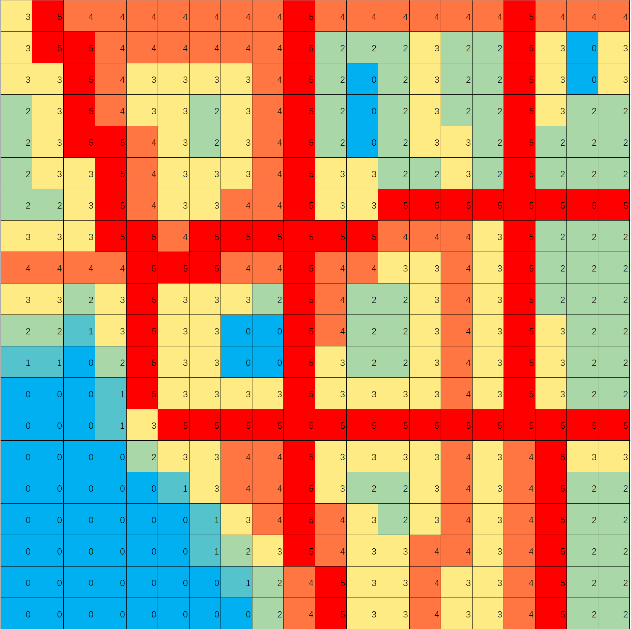
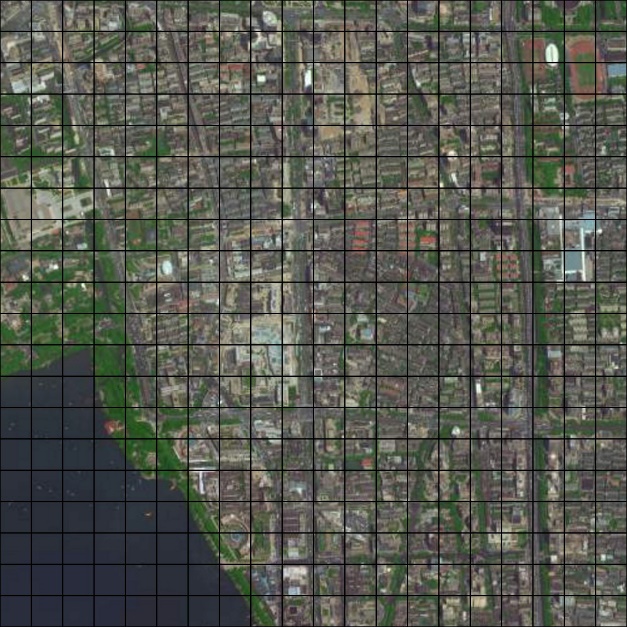
### 数据生成

#### 用户概率密度

由于无法从网络上的公开资料中获得详细的人口密度或用户密度数据，这里本文根据作者实际生活经验结合地图假设用户概率密度。

首先从网上截取2km\*2km的城区地图，大小为1000像素\*1000像素，即每个像素点为2m\*2m。然后在图片上绘制20\*20的网格，并根据人口密度为网格划分权重。这里假设了用户密度和人口密度是线性相关的。将人口密度按格子划分权重，根据调查为每个格子设置权重wi。最后求出所有权重值的总和W，用每个格子的权重除以权重的总和，就可以看做每格近似的用户概率pi，除以每格的面积就是用户概率密度，即每格的用户概率：

由于本文无法获得真实的人口密度数据或用户密度数据，采用人为对格子设置权重的方式，根据实际生活经验对每个格子按0-5进行打分，结果如图。为了有较好的的聚集效果，在计算实际权重时，按权重的平方进行计算。



左图为某市城区地图2km\*2km，并分为20\*20格。右图为按人口权重进行打分后的结果。

#### 用户位置生成

得到每个格子的用户概率数据后，需要用3个0~1的随机数来决定用户的位置。第一个随机数r1用于计算用户所处的格子i，这里采用累计概率的方法，即：

1. 第三个随机数r2、r3则用于计算用户的坐标，简单以格子左下角为原点，右上角为(1, 1)点，以(r2, r3)作为用户在格子内的坐标点，再加上格子本身的坐标即可：

#### 生成结果

根据查询资料，本文假设该大型站点一小时内有1000名用户。使用上文所述方法生成的用户点如右图所示。

在接下来的部分，本文将使用这份数据进行模拟计算。

### 模拟算法

在5.2中，本文讨论了如何在x-y平面上生成符合实际的用户点用以模拟。本节的主要内容是以5.2生成的用户点为输入，设计算法以计算成本最小的车站分布。车站位置分布主要包括两个部分：车站的数量和车站的位置。这里本文介绍K-means算法及其改进算法ISODATA，并结合上文公式对算法进行改造，计算出较优车站分布。

#### K-means算法

K-means算法是一种简单的聚类算法，常用于机器学习，对数据集进行分类。K-means算法会将映射到空间中的点按距离分成K类，具体算法[[2]](#footnote-2)是：

1. 从数据集中随机选取*k*个数据点作为聚类质心。
2. 重复一下过程直至结果收敛：
   1. ，计算其所属的类：
   2. 对于每个类*j*，重新计算其质心:

可以看到K-means算法和上文中需求非常相似。通过将距离与成本进行映射，分类点映射车站，可修改为本文所需算法。不过K-means算法的缺点在于分类数K是人为规定的，同时最终分类K点的位置受到初始K点影响。在K-means改进算法K-means++中，通过提高各类初始质心点间的距离，减少初始选择点对结果的干扰。但这种算法的分类数仍是固定的，不符合本文的需求，因此需要进一步的改进。

#### ISODATA算法

ISODATA是一种对K-means的优化算法。ISODATA算法相比于K-means算法，添加了分裂操作和合并操作，并通过这两个操作增加或减少类的数量。

相比于K-means算法，ISODATA算法在进行一轮计算后，对每一类会计算其标准差，如果某一维度的标准差大于一个设定值S，则对这个类进行分裂操作。分裂操作的具体算法为：

1. 计算类的标准差向量：

每个分量为：

其中为类的样本个数，为向量的维数。

1. 求出每类的最大标准差分量,
2. 如果，则把该类分为两个新的类，类的质心分别为：

如果两个类的质心小于某个设定值D，则对着两个类进行合并操作，其具体算法是：

1. 计算类的距离,
2. 如果，则把两个类合并为一个新的类，类的质心为：

其中 分别为两个类的样本各数。

在ISODATA中对类仍有一些其他的限制，但与本文需求并不相关，故不在此多做介绍。

有了分裂操作和合并操作，算法就可以对车站的数量进行修改。将成本映射为标准差，即可以将ISODATA算法应用于增加或减少车站的数量。结合基本的K-means算法，以5.2中生成的用户点作为输入，进行编程计算最佳的车站位置。程序代码部分请参见附录。下面将展示计算的结果。

#### 模拟算法

本节中，根据5.1中所述公式，结合K-means和ISODATA算法，本文构建实际的模拟算法。

1. 以所有用户点为初始用户类。
2. 计算当前所有类的车站位置，

其中i为当前类编号，k为当前类内的用户个数，为用户i的坐标，c为用户距离成本的系数，b为车站运输成本的系数。

此处将K-means算法中的所有点的算数平均，转化为根据成本的加权平均，且将运输成本考虑为一点进行计算。每个用户的权重均为c，是由用户距离成本公式所定义的。由于运输起点恒定是原点，因此只在分母中体现运输成本的权重，即b。

1. 计算距离原点最近的站点，并强制其为原点。这是因为原点处的大车站本身又是一个小车站。K-means算法并不能固定一个类点，因此此处必须强制定义。
2. 对所有类计算是否满足分裂操作的条件，如果满足则分裂。

检查各用户到车站的距离的标准差（即该车站用户距离成本）是否大于一输入值S，如果大于先进行分裂操作，具体算法与5.3.2中相同。特殊的，如果该站点为原点，即大型车站，如果需要分裂，则分裂结果为原点和其真实质心。分裂完成后，计算两车站成本之和是否大于之前单车站的情况，如果大于，则回退，否则完成分裂。

1. 对任意两类计算是否满足合并操作的条件，如果满足则合并。

如果两车站的距离小于一输入值R，则进行合并操作，具体算法与5.3.2中相同。同样，如果有一个点为原点，则合并结果为原点。合并完成后，检查合并的车站成本是否小于之前成本，如果大于则回退，否则完成合并。

1. 重新计算各个用户j所属的类i。

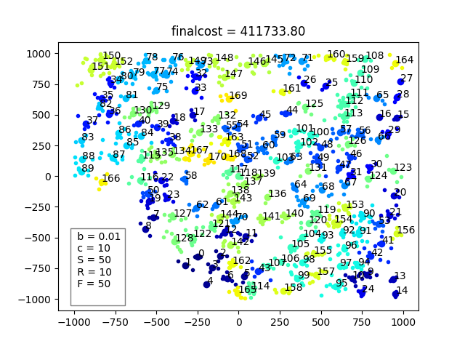
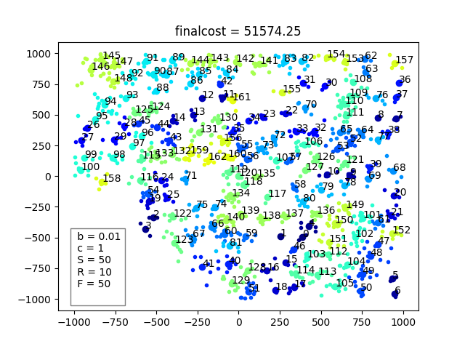
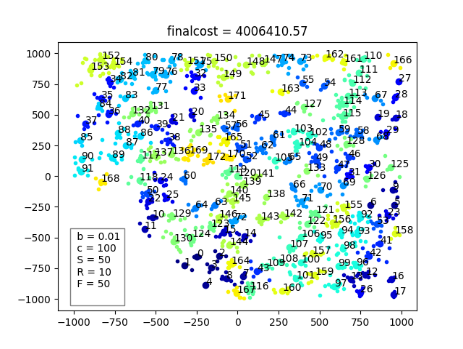
对于空类，则清除。

1. 重复2到5步骤，直至结果收敛或次数到达一极大值。
2. 计算最终成本，并输出最终成本与车站位置。

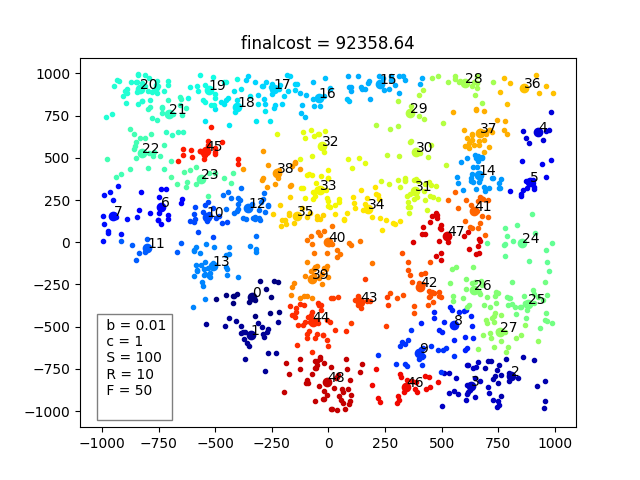
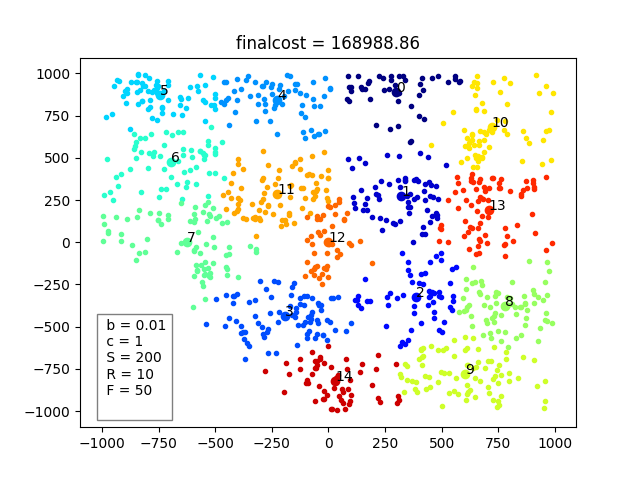
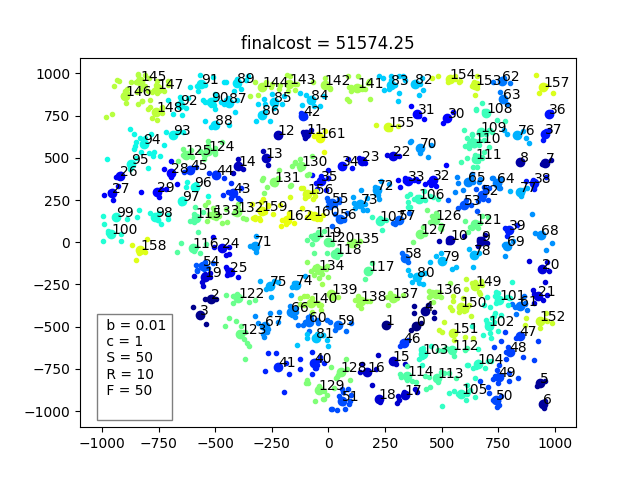
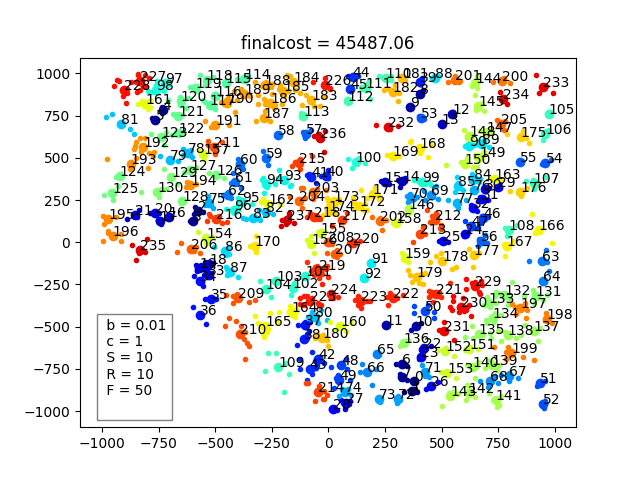
### 模拟结果

本文作者进行了多组试验，对不同参数的用途进行了试验和分析，结果如下。图中有数字标注的大圆点即为最终车站的位置。

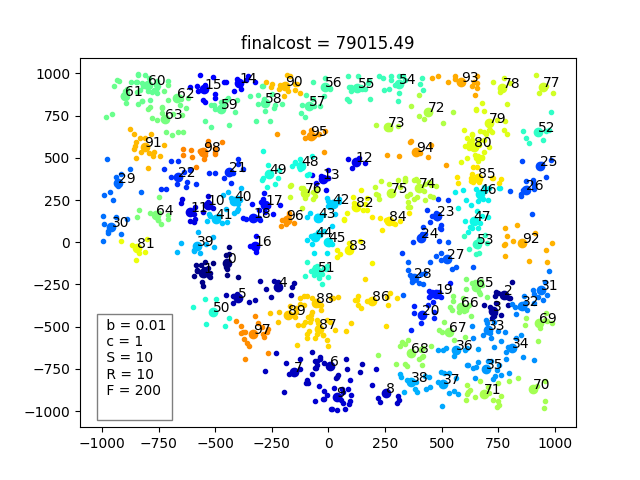
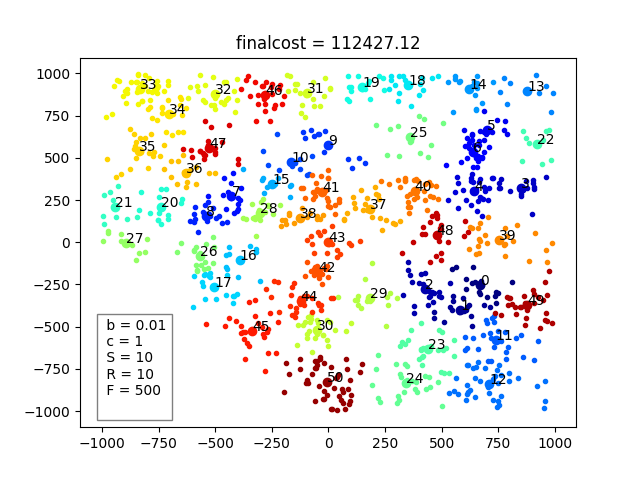
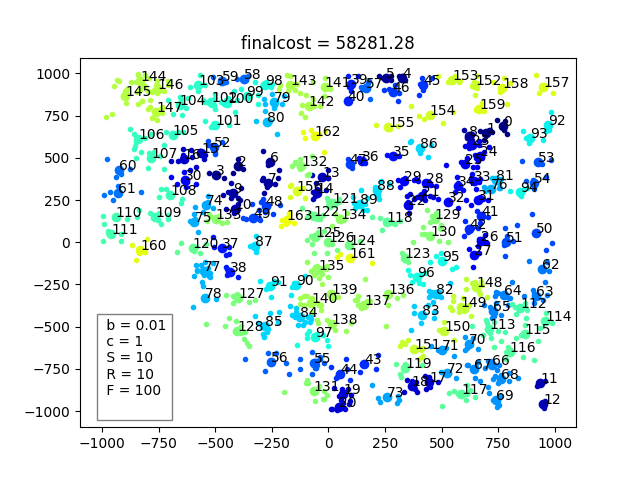
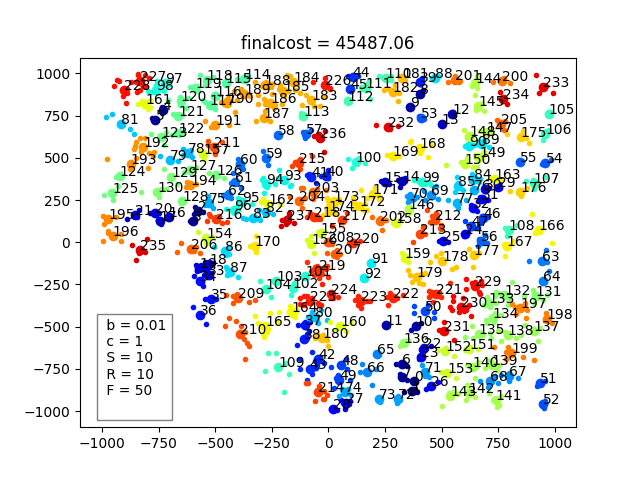
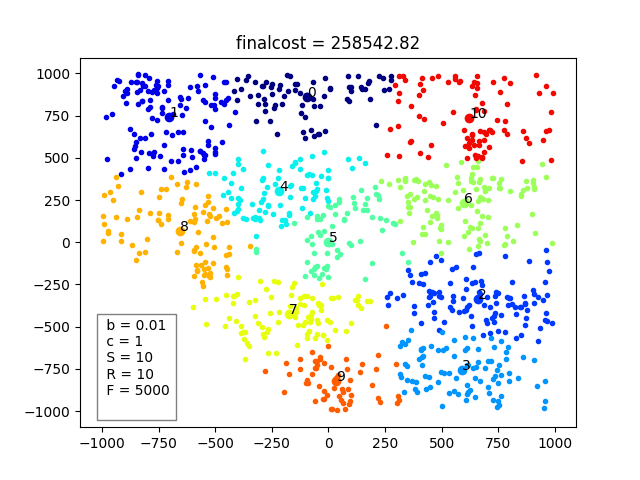
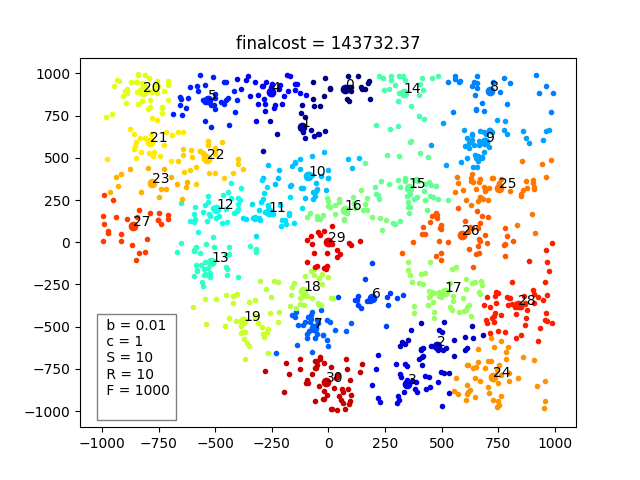
1. 只改变用户距离成本参数c：可以发现车站数量并没有变化，成本随c线性上升。因此c对车站数量并没有显著影响。



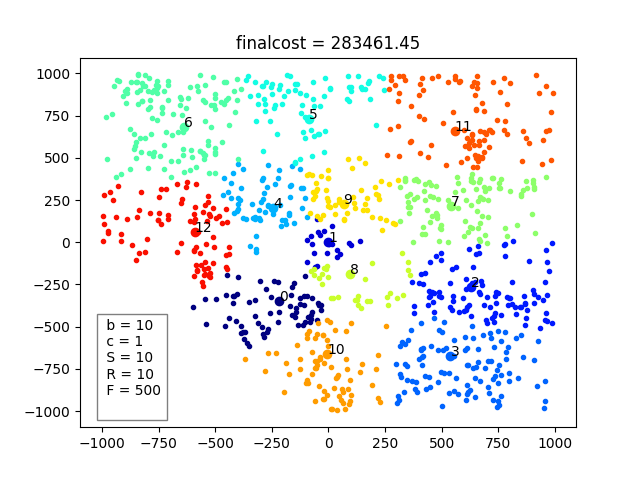
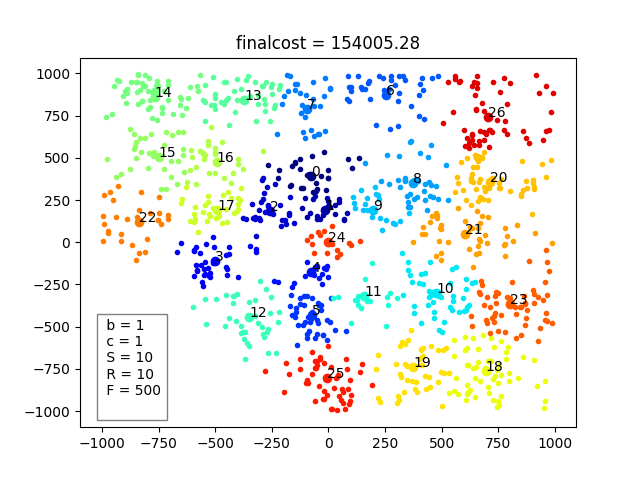
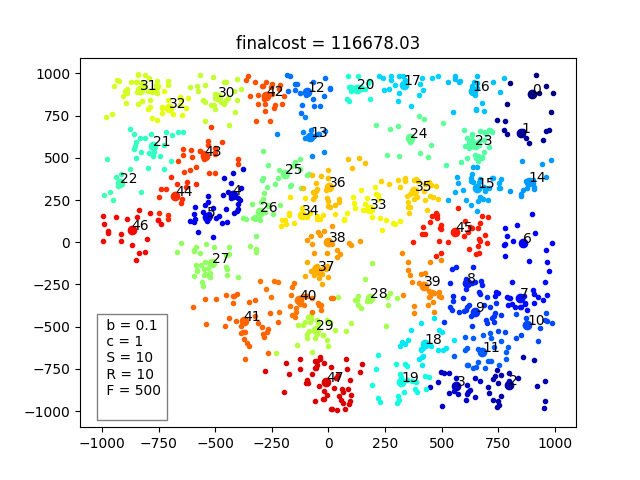
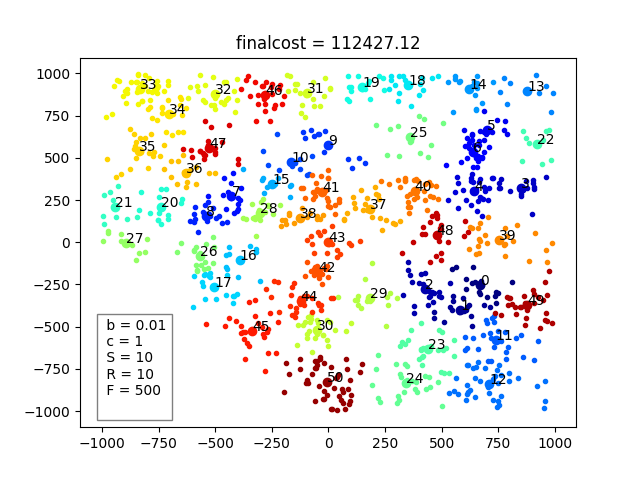
1. 只改变分裂操作阈值S：可以发现车站数量随着阈值上升显著减少，同时成本也随之增加。因此用户成本对总成本的影响远远大于车站固定成本及运输成本。阈值直接影响一个小型车站的覆盖范围，且随着车站数的增加总成本反而减少。为了降低总成本，共享单车公司会通过增加车站数量，人为的降低每个车站的阈值S。可以假设存在一种情况：当S值调整到非常小时，如S=10图所示，车站将会密布整条马路！



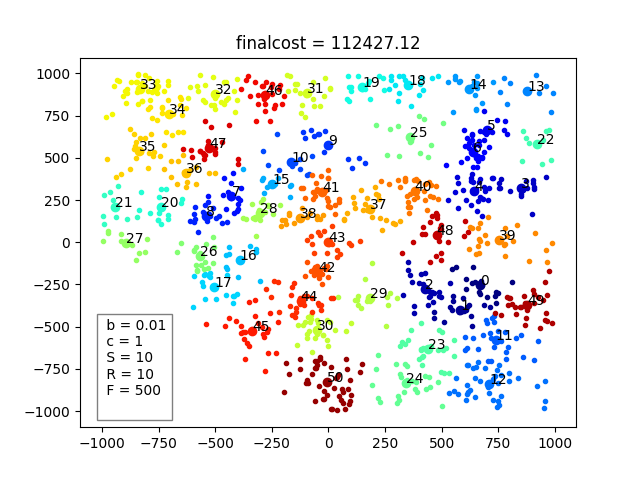
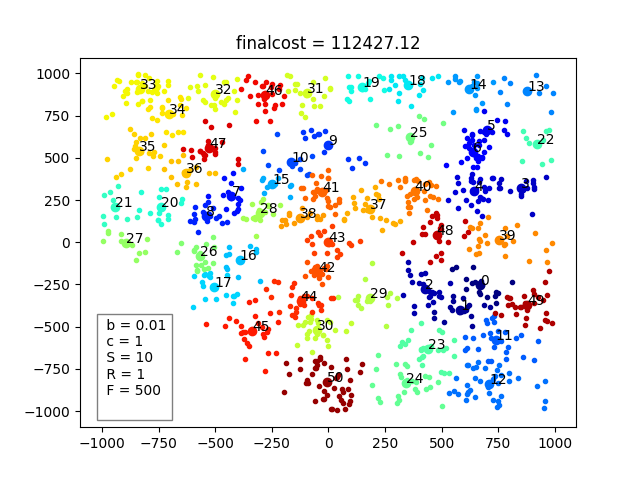
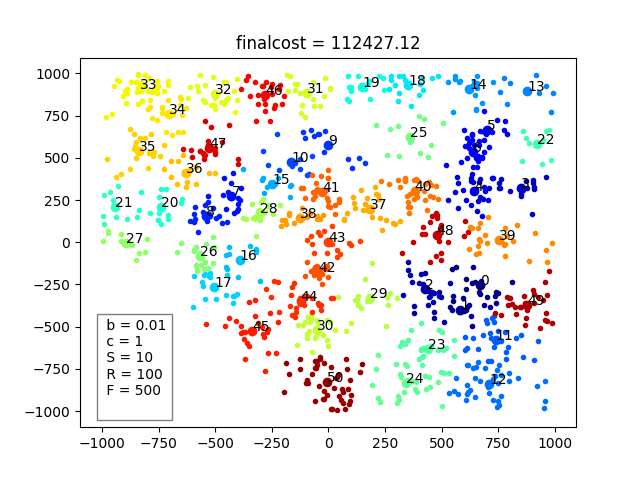
1. 只改变车站固定成本F时：可以看到随着固定成本的增加，车站数在减少，总成本也在增加。在F为200~1000间，均有不错的效果。



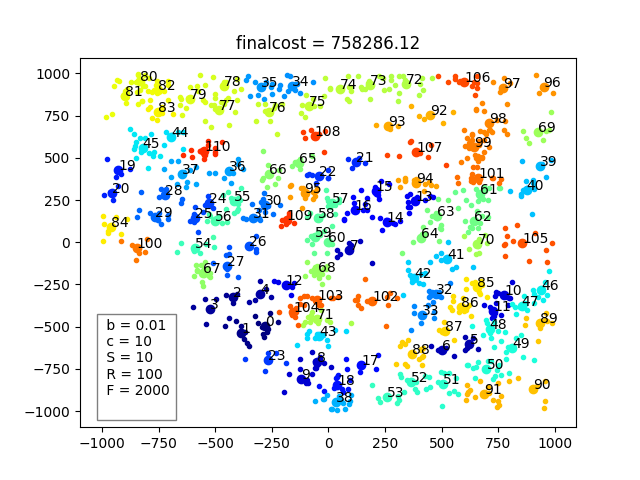
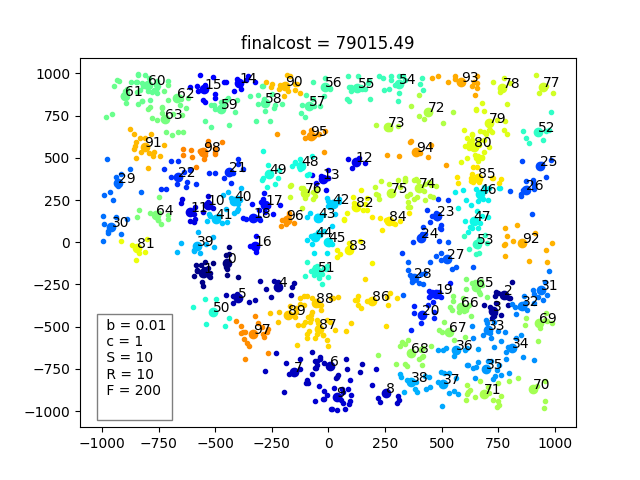
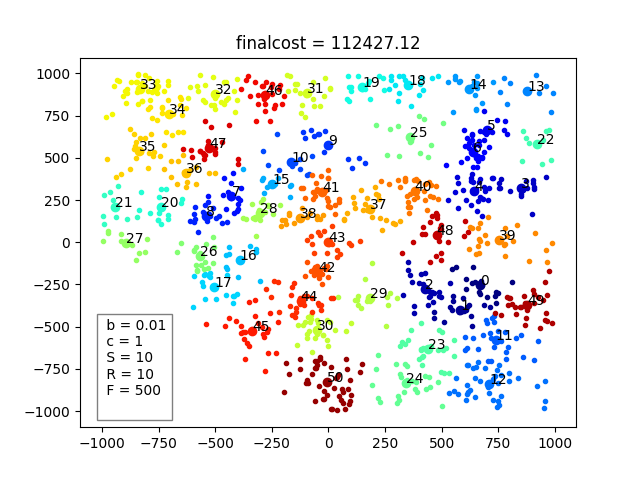
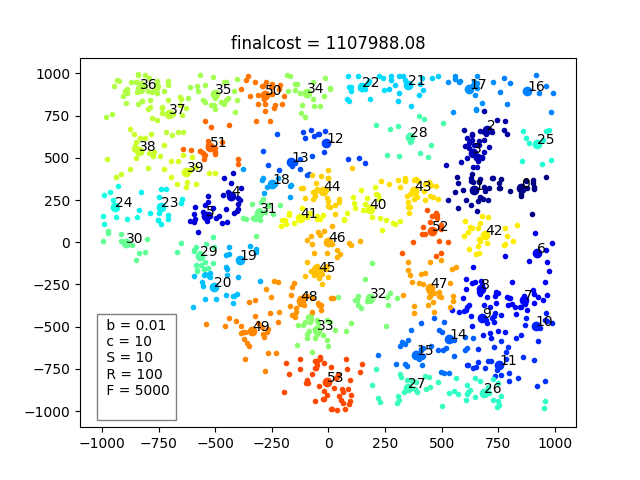
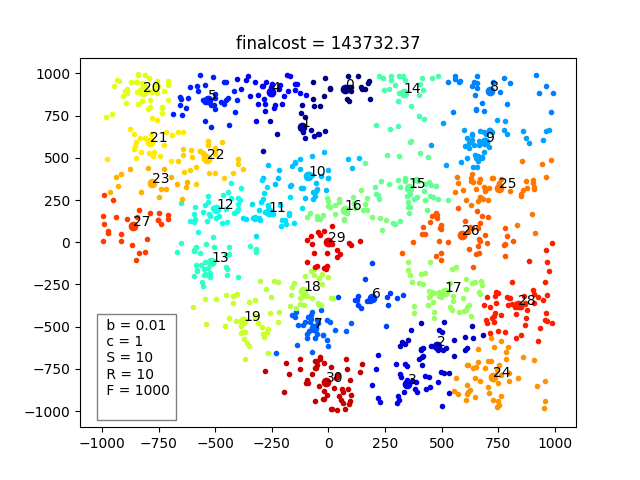
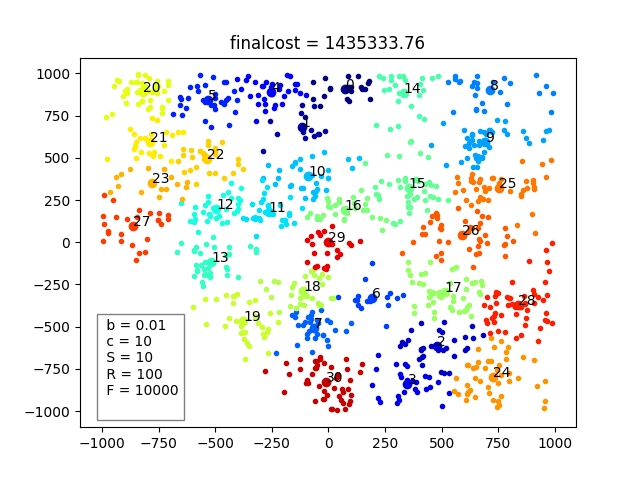
1. 只改变运输成本系数b时：可以看到随着运输成本的增加，车站数量在减少。因此运输成本也可以控制车站数量。



1. 只改变合并操作阈值R时：可以发现，车站数和成本完全没有变化。该阈值只在车站数量较多，距离比较近时有效。



1. 改变用户距离成本与车站固定成本的比值：由于1中用户距离成本并没有影响，而由3中固定成本对车站数量有影响。从实际经验出发，用户距离成本显然对车站数量有影响，因此作者假设，1中用户距离成本远大于车站固定成本，因此增加用户距离成本对车站数量无影响。为了验证这一结论，作者进行了多组比较。可以从下面的结果看出，在比例相同时，车站的分布并没有太大变化；比例不同时，车站的分布显著不同。因此，车站的分布与用户距离成本和车站固定成本的比值有关。



综合以上几点，可以发现可以影响共享单车小型车站网络的因素有：运输成本，车站分裂阈值，及用户距离成本和车站固定成本的比值有关。在实际中，公司和监管部门可以通过控制这几个参数，来控制小型车站的分布。例如，为了解决共享单车在部分地区供远大于求，造成车辆淤积的情况，监管部门可以通过收取管理和清理费用，直接增加车站的固定成本，间接减少了车站的数量。

## 总结和未来可行研究方向

本文对共享单车的成本和车站网络进行了研究。首先，本文从理想化的条件开始，构建最基础的数学模型，对共享单车车站分布网络和成本进行分析。之后，本文逐步增加影响参数，构建了一个较为复杂但更贴合实际的模型。本文又进一步利用计算机对构建的数学模型进行了模拟，并对结果进行分析。最后基于分析结果，本文提出了可行的控制共享单车车站分布网络的建议，达成了研究的目的。

由于时间上的原因，本文并没有对存在大小车站的复杂网络进行构建。同时，在分析共享单车网络时，也并未考虑其他交通工具、周围设施、交通拥堵等情况对网络的影响。对于未来可行的研究方向，除补充上述两点外，作者认为对于用户出行的假设是过于理想化的，可以研究随用户使用而自发生成的新的小型车站的算法，也可以研究如何从实际的数据出发，准确的计算各类参数，为准确的网络建立提供基础。

## 附录

### 参考文献

有机蔬菜配送网点模型分析与建立，作者：王亚哲，郑州市第一中学

基于正六边形网格的覆盖问题的研究，作者：宋燕妮，华东理工大学

K-means聚类算法的三种改进介绍与对比，<http://www.cnblogs.com/yixuan-xu/p/6272208.html>

### 代码

**Generation.py 用于生成用户点。**

import numpy.random as random

import matplotlib.pyplot as plt

weightMap = [ \

3,5,4,4,4,4,4,4,4,5,4,4,4,4,4,4,5,4,4,4,\

3,5,5,4,4,4,4,4,4,5,2,2,2,3,2,2,5,3,0,3,\

3,3,5,4,3,3,3,3,4,5,2,0,2,3,2,2,5,3,0,3,\

2,3,5,4,3,3,2,3,4,5,2,0,2,3,2,2,5,3,2,2,\

2,3,5,5,4,3,2,3,4,5,2,0,2,3,3,2,5,2,2,2,\

2,3,3,5,4,3,3,3,4,5,3,3,2,2,3,2,5,2,2,2,\

2,2,3,5,4,3,3,4,4,5,3,3,5,5,5,5,5,5,5,5,\

3,3,3,5,5,4,5,5,5,5,5,5,4,4,4,3,5,2,2,2,\

4,4,4,4,5,5,5,4,4,5,4,4,3,3,4,3,5,2,2,2,\

3,3,2,3,5,3,3,3,2,5,4,2,2,3,4,3,5,2,2,2,\

2,2,1,3,5,3,3,0,0,5,4,2,2,3,4,3,5,3,2,2,\

1,1,0,2,5,3,3,0,0,5,3,2,2,3,4,3,5,3,2,2,\

0,0,0,1,5,3,3,3,3,5,3,3,3,3,4,3,5,3,2,2,\

0,0,0,1,3,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,\

0,0,0,0,2,3,3,4,4,5,3,3,3,3,4,3,4,5,3,3,\

0,0,0,0,0,1,3,4,4,5,3,2,2,3,4,3,4,5,2,2,\

0,0,0,0,0,0,1,3,4,5,4,3,2,3,4,3,4,5,2,2,\

0,0,0,0,0,0,1,2,3,5,4,3,3,4,4,3,4,5,2,2,\

0,0,0,0,0,0,0,1,2,4,5,3,3,4,3,3,4,5,2,2,\

0,0,0,0,0,0,0,0,2,4,5,3,3,4,3,3,4,5,2,2 \

]

row = 20

# sumWeight = sum(weightMap)

# probabilityMap = [i/sumWeight for i in weightMap]

sumWeight = sum(i\*\*2 for i in weightMap)

probabilityMap = [i\*\*2 /sumWeight for i in weightMap]

userNumber = 1000

userList = []

for i in range(userNumber):

# if i > 0 and i \* 100 % userNumber == 0:

# print("Processing..{}%".format(i/100))

r = random.ranf()

x = random.ranf()

y = random.ranf()

for i in range(len(probabilityMap)):

if r >= probabilityMap[i]:

r -= probabilityMap[i]

else:

#-1000~1000 total, 100 per grid

x = (i % row) \* 100 + 100 \* x - 1000

y = (20 - 1 - int(i / row)) \* 100 + 100 \* y - 1000

userList.append([x, y])

plt.plot(x, y, 'r.')

break

plt.savefig("userpoint-square.png")

plt.show()

file = open("user.txt", "w")

for u in userList:

file.write(str(u))

file.write('\n')

file.close()

# if \_\_name\_\_ == '\_\_main\_\_':

# r1 = random.ranf()

# print(r1)

**Simulation.py 用于建立模型并进行模拟。**

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

#运输成本

b = 0.01

#用户距离

c = 10

#分裂

S = 10

#合并

R = 100

#固定成本

F = 2000

userPosition = [[0, 0]]

def loadUserPosition():

file = open("user.txt", "r").readlines()

user = []

for line in file:

tmp = line.strip().split(',')

x = tmp[0].strip('[]')

y = tmp[1].strip('[]')

# print(x, y)

user.append([float(x), float(y)])

return user

def massCenter(userList):

if not userList:

return None;

x, y = zip(\*(userList))

sumx = sum(x) \* c

sumy = sum(y) \* c

x = sumx / (len(x) \* c + b)

y = sumy / (len(y) \* c + b)

return [x, y]

def \_utestMassCenter():

a = [[1, 1], [2, 2], [3, 3], [-1, -1], [-2, -2], [-3, -3]]

print(massCenter(a))

def distance(p1, p2):

if not p1 or not p2:

return None

x = p1[0] - p2[0]

y = p1[1] - p2[1]

return np.sqrt(x \* x + y \* y)

def \_utestDistance():

print(distance([-1, -1], [2, 2]))

def forceBaseStation(muList):

if [0, 0] in muList:

return

minDis = 10000

minNum = 0

for i in range(len(muList)):

tmpDis = distance(muList[i], (0, 0))

if tmpDis < minDis:

minDis = tmpDis

minNum = i

muList[minNum] = [0, 0]

def \_utestForceBaseStation():

g = [[-1, -2], [1, 1], [2, 2], [3, 3], [4, 4]]

forceBaseStation(g)

print(g)

def stationCostCalculation(group, massCenter):

if not group:

return None

d = sum(c \* distance(i, massCenter) for i in group)

t = b \* distance(massCenter, (0, 0))

return d + t + F

def \_utestStationCostCalculation():

g = [[0, 0], [1, 1], [2, 2], [3, 3], [4, 4]]

m = massCenter(g)

a = stationCostCalculation(g, m)

print(a)

def splitGroup(group, mc):

if not group:

return None

stdDeviation = [np.sqrt(1/len(group) \* sum((i[j] - mc[j])\*\*2 for i in group)) for j in range(len(group[0]))]

# print(stdDeviation)

if not [i for i in stdDeviation if i > S]:

return None

if mc == [0, 0]:

mcplus = massCenter(group)

mcminus = [0, 0]

else:

mcplus = [mc[i] + stdDeviation[i] for i in range(len(mc))]

mcminus = [mc[i] - stdDeviation[i] for i in range(len(mc))]

mpgroup, mmgroup = buildGroup(group, [mcplus, mcminus])

mpcost = stationCostCalculation(mpgroup, mcplus)

mmcost = stationCostCalculation(mmgroup, mcminus)

precost = stationCostCalculation(group, mc)

# print(mcplus, mcminus)

if mpcost and mmcost and precost > mpcost + mmcost:

return [mcplus, mcminus]

else:

return None

def \_utestSplitGroup():

g = [[0, 0], [1, 1], [2, 2], [3, 3], [4, 4]]

m = massCenter(g)

a = splitGroup(g, m)

print(a)

def mergeGroup(group1, massCenter1, group2, massCenter2):

if not group1 or not group2:

return None

dis = distance(massCenter1, massCenter2)

if dis > R:

return None

weight1 = len(group1)

weight2 = len(group2)

#std deviation

if massCenter1 == [0, 0] or massCenter2 == [0, 0]:

mu = [0, 0]

else:

mu = [(massCenter1[i] \* weight1 + massCenter2[i] \* weight2)/(weight1 + weight2) for i in range(len(massCenter1))]

m1cost = stationCostCalculation(group1, massCenter1)

m2cost = stationCostCalculation(group2, massCenter2)

precost = stationCostCalculation(group1 + group2, mu)

if precost > m1cost + m2cost:

return None

else:

return [mu]

def \_utestMergeGroup():

g1 = [[0, 0], [1, 1], [2, 2], [3, 3], [4, 4]]

m1 = massCenter(g1)

print(m1)

g2 = [[0, 4], [1, 3], [2, 2], [3, 1], [4, 0]]

m2 = massCenter(g2)

print(m2)

a = mergeGroup(g1, m1, g2, m2)

print(a)

def buildGroup(userList, muList):

group = [[] for i in range(len(muList))]

for user in userList:

minGroup = 0

minDis = 10000000

for i in range(len(muList)):

tmpDis = distance(user, muList[i])

if tmpDis < minDis:

minDis = tmpDis

minGroup = i

group[minGroup].append(user)

# print(group)

return group

def removeEmptyGroup(group, muList):

i = 0

while i < len(group):

#empty group

if not group[i]:

muList.pop(i)

group.pop(i)

i -= 1

i += 1

def \_utestBuildGroup():

g = [[0, 0], [1, 1], [2, 2], [3, 3], [4, 4]]

m = [[1, 1], [3, 3]]

a = buildGroup(g, m)

print(a)

def StationCalculation(userList):

#step 1

group = [userList]

lastMuList = []

maxStep = 1000

maxError = 1

error = 100

step = 0

while step < maxStep and error > maxError:

print("Step ", step)

step += 1

#step 2

massCenterList = [massCenter(g) for g in group]

#step 3

forceBaseStation(massCenterList)

#step 4

muList = []

i = 0

while i < len(group):

sg = splitGroup(group[i], massCenterList[i])

#if split, then do not check for merge

if sg:

group.pop(i)

massCenterList.pop(i)

i -= 1

muList += sg

i += 1

#step 5

i = 0

while i < len(group):

j = 0

while j < len(group):

if i is not j:

mg = mergeGroup(group[i], massCenterList[i], group[j], massCenterList[j])

if mg:

group.pop(i)

massCenterList.pop(i)

if i > j:

group.pop(j)

massCenterList.pop(j)

else:

group.pop(j - 1)

massCenterList.pop(j - 1)

i -= 1

j -= 1

muList += mg

break

j += 1

i += 1

muList += massCenterList

#step 6

group = buildGroup(userList, muList)

removeEmptyGroup(group, muList)

#step 7

if lastMuList and muList and len(lastMuList) == len(muList):

error = [np.sqrt(1/len(lastMuList) \* sum((lastMuList[i][j] - muList[i][j])\*\*2 for i in range(len(muList)))) for j in range(len(muList[0]))]

error = sum(i\*\*2 for i in error)

print("Error: ", error)

lastMuList = muList

return group, muList

def \_utestStationCalculation():

group, muList = StationCalculation([[0, 0], [1, 1], [2, 2], [3, 3], [4, 4]])

print(group, muList)

print(stationCostCalculation([[0, 0], [1, 1], [2, 2], [3, 3], [4, 4]], [0, 0]))

finalProcess(group, muList)

def finalProcess(group, muList):

totalCost = sum(stationCostCalculation(group[i], muList[i]) for i in range(len(group)))

print(totalCost)

colormap = cm.jet(range(256))

# colormap = colormap[10:245]

step = int(len(colormap)/len(group))

step = 1 if step == 0 else step

tmp = 0

for g in group:

for p in g:

plt.plot(p[0], p[1], color = colormap[tmp], linestyle = 'None', marker = '.')

tmp += step

tmp = 0

count = 0

for m in muList:

plt.plot(m[0], m[1], color = colormap[tmp], linestyle = 'None', marker = 'o')

plt.annotate(str(count), m)

count += 1

tmp += step

s = " b = {}\n c = {}\n S = {}\n R = {}\n F = {}\n".format(b, c, S, R, F)

plt.text(-1000, -1000, s, bbox=dict(facecolor='white', alpha=0.5))

plt.title("finalcost = {:.2f}".format(totalCost))

plt.savefig("final\_{}\_{}\_{}\_{}\_{}.png".format(b, c, S, R, F))

plt.show()

return totalCost

if \_\_name\_\_ == '\_\_main\_\_':

# \_utestMassCenter()

# \_utestDistance()

# \_utestSplitGroup()

# \_utestMergeGroup()

# \_utestForceBaseStation()

# \_utestBuildGroup()

# \_utestStationCostCalculation()

# \_utestStationCalculation()

user = loadUserPosition()

g, s = StationCalculation(user)

finalProcess(g, s)

1. 交通部：企业累计在全国范围内投放共享单车超过1000万辆，<http://cn.chinadaily.com.cn/2017-05/23/content_29459777.htm> [↑](#footnote-ref-1)
2. 具体算法的数学表达参考了《K-means聚类算法》一文，http://www.cnblogs.com/jerrylead/archive/2011/04/06/2006910.html [↑](#footnote-ref-2)