**车辆分配及导航**

本算法更适用于处于河南中心部分的城市，如郑州，处在边缘位置的城市也支持

**程序步骤：**

**1.根据城市之间的连通状态，构建以城市为结点、两个城市间的距离（根据两个城市经纬度计算的欧式距离）作为边权重的无向图。**

**2.根据起始点，对除了起始点之外的其他点进行聚类，将点划分成几个部分。**

**3.在每个部分中找出货车运输的最优路径，根据这些路径上的订单重量分配货车。**

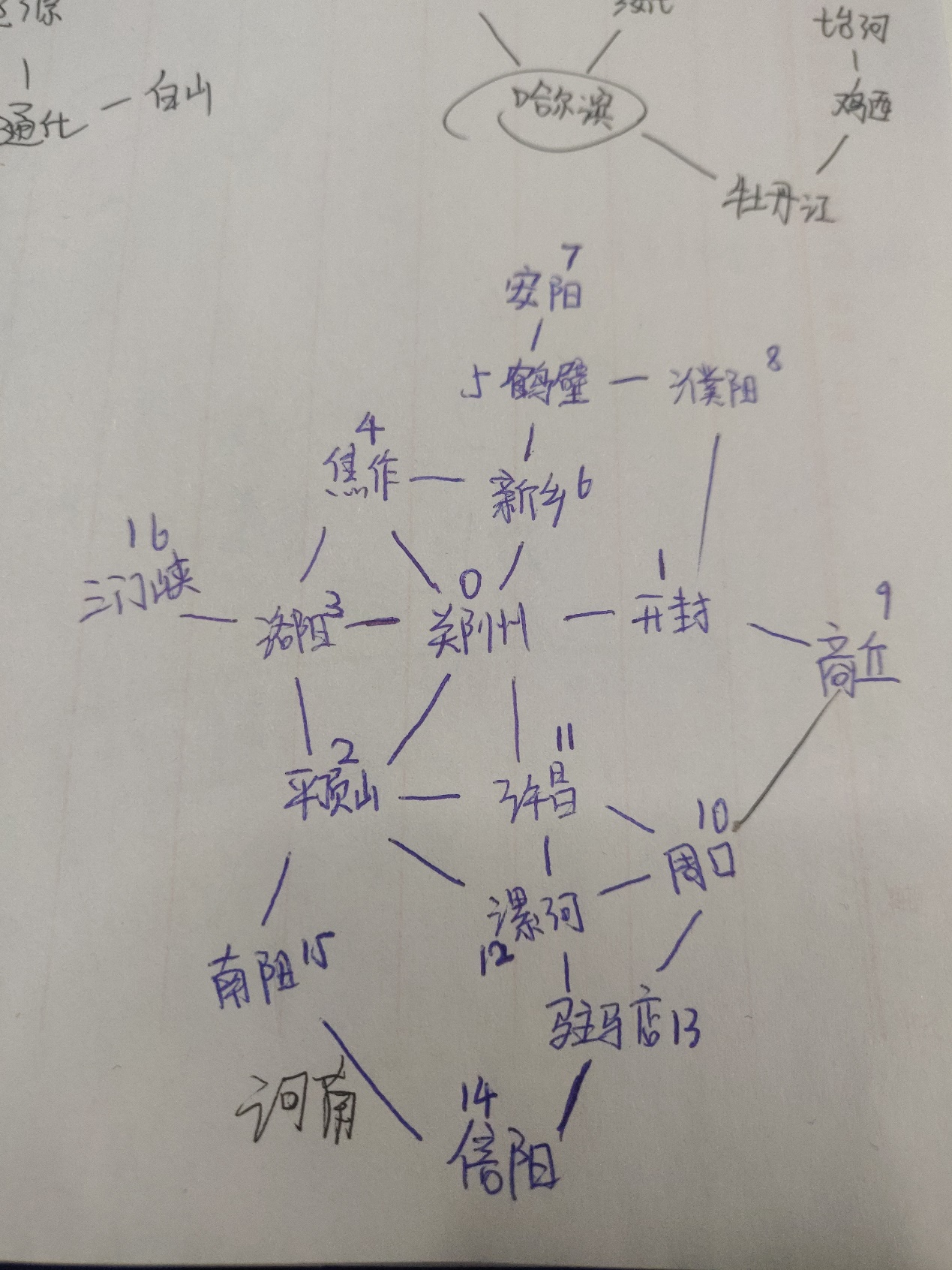
**4.配置程序的本地接口，为系统提供车辆分配和导航的服务**

**1.0构建河南省主要城市之间的连通状态**

以下为高德地图中，河南省主要城市之间的公路连通状况：



根据城市之间公路的连通情况，大致构建的无向图如下图所示。

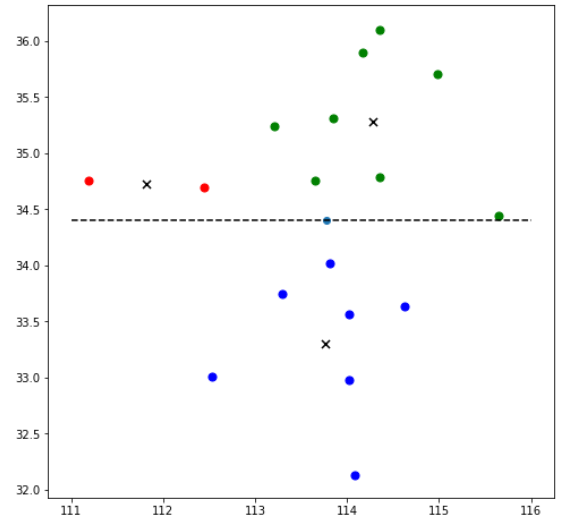


**2.0对城市进行聚类，划分出大致方向**

**目的：为了找出货车要配送的不同方向，此处使用聚类算法，将城市聚成几个簇**

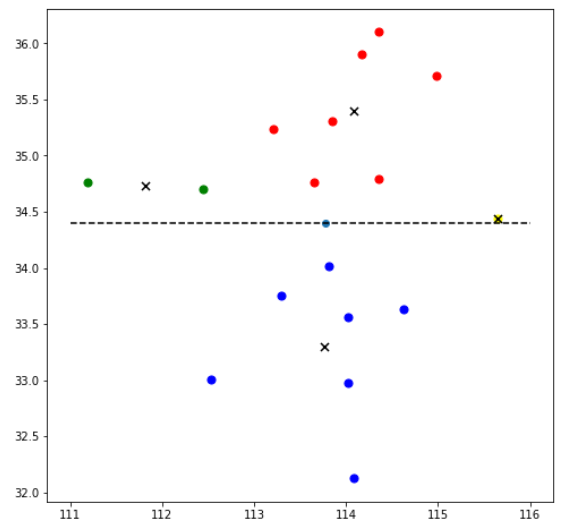
对河南省的主要城市使用KMeans算法进行聚类，结果大致如下：

当要聚成的簇为3时：



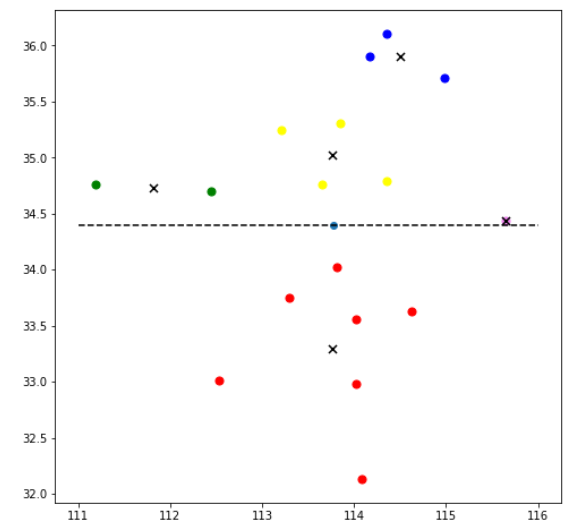
其中 x点表示聚类中心，水平线表示所有城市纬度的均值m（用于大致划分上下两个方向）。

当要聚成的簇为4时：



其中有一个点（最右边的点，商丘市）单独为1类。

当要聚成的簇为5时：



此时就会有规模更小的，更局部性的簇产生了，并不是我们想要的大致地划分出方向，我们知道，聚类算法效果的评估更多时候要取决于业务，所以综合上述三种要聚成的簇的取值，我们决定将城市划分为3个簇

**2.1根据起始点大致划分出向北，和向南两个方向**

1. 根据对城市进行的聚类分析，结合实际情况，我们先判断起始点的纬度w是否大于所有城市纬度的平均值m。
2. 如果起始点的纬度大于m，则以w为划分线，纬度大于w的城市集合作为一个簇（位于起始点纬度以上的划分为一个簇），对纬度小于w的点（起始点以下的点）用聚类划分为2个簇；如果起始点的纬度位于m以下，则纬度小于w的城市作为一个簇，大于w的城市用聚类划分为2个簇。

**2.2对聚类之后的城市所在的簇进行修正**

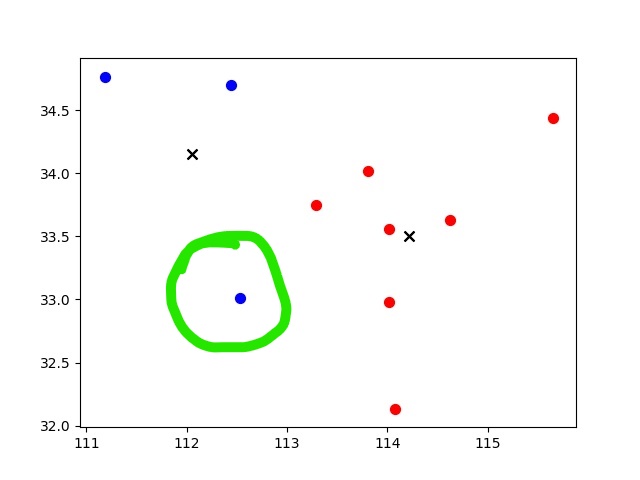
**聚类之后会出现的不合理的情况，及相应的处理如下：**

1.如果一个簇中的某个城市city 与其所在的簇中的所有城市都没有连通关系，而是与另一个簇c\_2中的城市有连通关系，则在c\_1所构成的无向图中删除city，把city放到c\_2中。

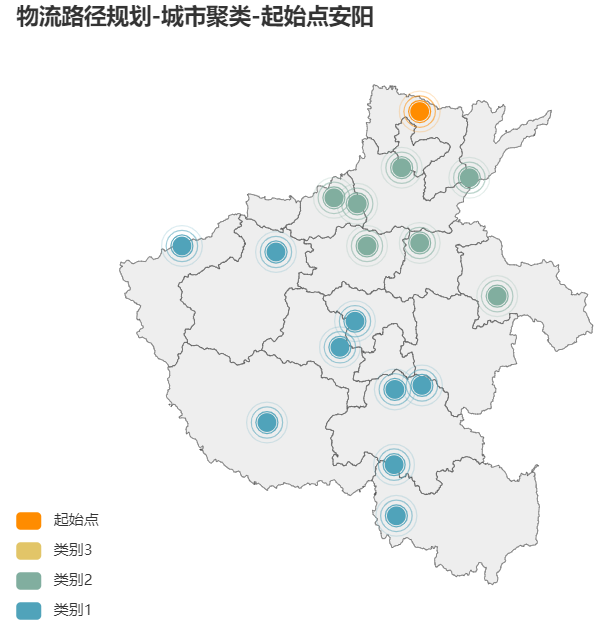
2.如果一个簇c\_1中的某个城市city与自己簇中的结点无连通，但是与位于维度的均值w的另一边的簇c\_3有联通，则将city归并到簇c\_3中。

3.对于起始点处于边缘位置的点，进行2.1的操作以后，会出现起始点与某个簇c不直接连通的情况，此时就寻找从起始点到c的最短路径，记录下中间经过的结点，将这些结点归并到簇c当中。

2.2的第一点所述的情况

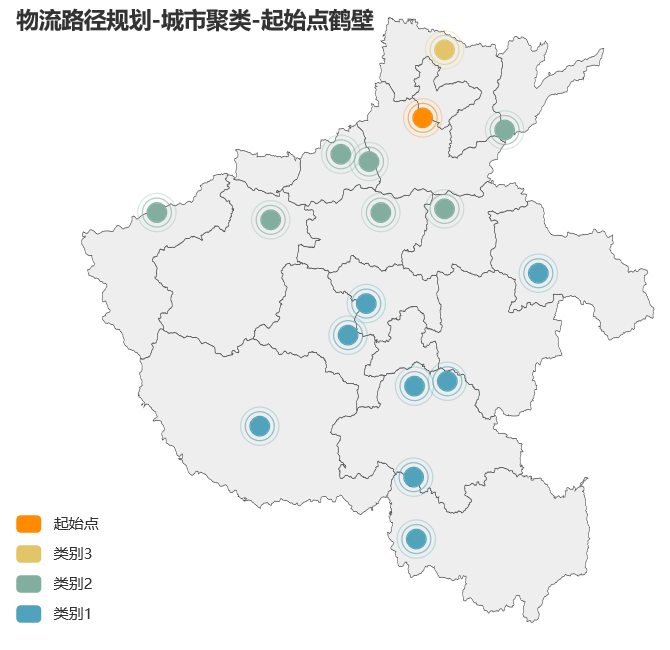


2.2的第三点所述的情况



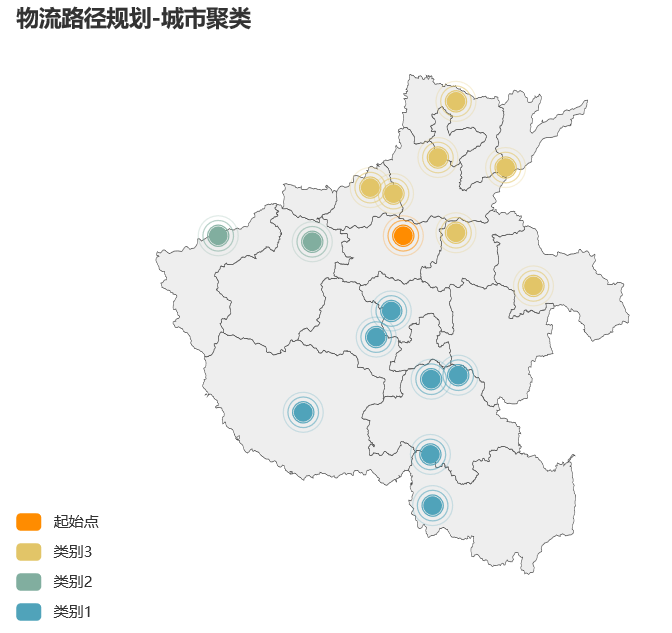
* 1. **对于每个簇c，切断c与其他簇的连通状态，但保持与起始点的连通状态**

对于处于河南省边缘的城市的聚类，在执行了这一步操作之后，在不同的簇构建的无向图中，会出现与起始点不连通的情况，如下图所示：



对于这种情况，程序会寻找从起始点到与其不连通的簇c\_n的最短路径，以这条路径作为媒介，连通起始点与c\_n，并对这条路径经过的别的簇中的城市做记录，在后面分配重量的时候防止重复分配。

以起始点是郑州为例，修正后的聚类结果可视化如下图所示



**3.0 在步骤2划分好的每一个簇中，找出配送的最优路径**

**3.1 构建每一个簇对应的无向图c\_i**

根据每一个簇中城市之间的连通状态构建对应的无向图c\_i。

**3.2对于每一个簇c\_i，计算起始点到c内每一个城市的最短路径**

对于每一个簇c\_i，对c里面的每一个结点node，使用深度优先搜索遍历无向图c\_i，找到以起始点为源点，node为终点的最短路径所经过的结点。

以郑州市作为起点为例，以下为步骤3.2所找到的簇c\_1，c\_2，c\_3中的每一条最短路径，数字对应的是城市的编号：

**c\_1: [0, 2], [0, 11, 10], [0, 11], [0, 2, 12], [0, 2, 12, 13], [0, 2, 12, 13, 14], [0, 2, 15]**

**c\_2: [0, 1], [0, 4], [0, 6, 5], [0, 6], [0, 6, 5, 7], [0, 1, 8], [0, 1, 9]**

**c\_3: [0, 3], [0, 3, 16]**

可以发现，有一些路径其实是另一条路径的**子路径**（最优子路径存在原理），于是我们判断，如果一条路径p1是另一条路径p2的子路径，就删除路径p1，最后便得到如下的路径：

c\_1: [0, 11, 10], [0, 2, 12, 13, 14], [0, 2, 15]

c\_2: [0, 4], [0, 6, 5, 7], [0, 1, 8], [0, 1, 9]

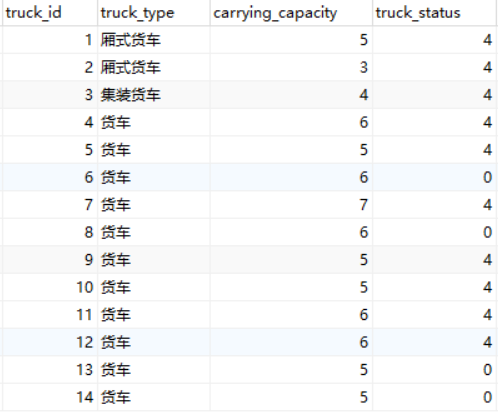
c\_3: [0, 3, 16]

c\_1，c\_2，c\_3里面的每一条路径，就是最后货车行驶时，从起始点出发到这个簇中进行配送的每一条路径。

**3.3根据这些路径上的订单重量，将订单分配给货车**

根据站点注册的货车的核载量，和货车是否空闲，给货车分配订单

我们这里有一些已注册的货车的基本信息，如下：



carrying\_capacity表示货车的核载量，以吨 (t) 为单位

truck\_status 表示货车的状态，0表示货车空闲，可以用于运输

本系统的货车分配的算法：使用让货车尽可能装满（对于相同的订单重量，分配能运完的，车的载重量之和尽可能小的车）的分配算法

对于每一条已经确定的行驶路径，计算路径上订单的总重量，分配空闲的货车去运输。

例：当前空闲的货车的载重量有7 t、6 t、5 t、5 t、3 t的，路径path上订单的总重量为13.4 t，则会分配 5 t、6 t、3 t的车去运输

本系统提供两种寻找货车分配最优解的算法：**回溯法**，**动态规划**

**3.3.1回溯法**

使用深度优先搜索，穷举所有的组合可能，对每种符合要求的可能计算一下货车重量和，找出货车重量和最小的那一组组合

*适用情况*：适用于物流分配站拥有的货车数少，订单总数少的情况

**3.3.2动态规划**：

此处设有辆待分配的货车，所有的货车载重量之和为 ，货车编号为 ，表示编号为的货车的载重量，当前路径上订单总重量为 ，函数 表示从 中选择货车，未分配的货车载重量之和剩下多少

此时要解决的问题为：在 的区间内，要找到怎么样的货车组合，使得货车组合的载重量和最小且大于

可以将上面所述的问题分解为多个子问题 ，子问题为：当面临第i辆车的选择时，如果选取了第i辆车之后，未分配的货车载重量之和小于 ，不符合条件，则 ，否则

即状态转移方程为

对每辆货车，对 中的每一个数进行遍历，最后 f(n, w) 就是最优的解

回溯找出最优解的组成：

根据子问题的条件及对应的状态转移操作，回溯出最优解的组成，找到货车对应的编号

*适用情况*：所有情况，当物流分配站拥有货车数多，要运送的订单多时，此算法事件复杂度会比回溯法小很多

**4.0 在本地配置接口，为系统提供车辆分配和导航的服务**

由于系统是使用Java语言编写的，而车辆分配和导航的算法是使用Python语言编写的，所以此算法为了提供对不同语言编写的系统的兼容，使用Django框架配置一个接口，使用Ajax技术，返回一个json格式的数据，里面的内容即车辆分配和导航的结果

以郑州市为起始点为例，展示其中的一部分

1. {
2. "0": {
3. "error": 0,
4. "truck\_number\_list": [
5. 1,
6. 3
7. ],
8. "path": [
9. "许昌",
10. "周口",
11. "商丘"
12. ],
13. "orders\_weight": [
14. 4998.5,
15. 2932.0
16. ],
17. "order\_id\_list": [
18. [
19. 5,
20. 14,
21. 24,
22. 35,
23. …
24. ],
25. [
26. 505,
27. 513,
28. 518,
29. …
30. ]
31. ]
32. }

Json里包含了字符串从 0到n的键，分别包含每一条路径的信息

“error” 为0表示分配过程中没有产生异常（当前拥有的货车能够装得下此路径上的订单），为1表示产生异常

“truck\_number\_list” 表示这条路径上分配的货车的id

“path” 表示在这条路径上运输经过的城市

“orders\_weight” 分配的不同货车对应运输的订单总重量

“order\_id\_list” 分配的不同货车对应运输的订单的id