2018 西安交通大学
数学建模竞赛

竞赛时间：2018 年 4 月 26 日 8 时-5 月 3 日 8 时

队伍编号 658

选　题 B

队　长　姓　名 胡金辉 班级 物理试验班 52

队员 1　姓名 孙媛媛 班级 物理试验班 52

队员 2　姓名 樊本成 班级 物理试验班 52

2018 年 4 月 26 日

无线回传拓扑规划模型研究

摘要

Relay 无线回传方案是针对光纤通信和微波通信在应用中的局限性的一种解决方案。本文研究的问题是:在给定一个地区候选站点的位置分布的情况下，根据站点见的相互位置、站点间拓扑关系等限制条件，在满足一定回传质量的前提下，建立数学模型来生成成本低廉，信号覆盖面积广的回传拓扑网络。

在 Relay 无线回传方案中，宿主站的信号覆盖范围要远远大于子站，起着直接为用户提供信号和向子站发送信号的双重作用。因此，决定宿主站的位置应当是规划过程中起始步骤。因为候选站在分布上具有明显的群聚性，所以可以采取分簇的方法进行处理。首先用 ISODATA 算法对所有的候选站进行划分，将相对集中的候选站分为一簇。为了确保簇的质量，我们引入 LOF 离群因子去评判每一个簇，当遇到异常簇时报告，提醒人工干预。经过测试后的簇，用正六边形密排法对内部站点进行进一步细分，并决定宿主站的个数和类型。

接下来，我们将采用 Dijkstra 算法来决定这些子站和宿主站之间的损耗最短的连接方式。并用简洁的矩阵表示站之间的连接。

在确定了初始的连接方式后，无线回传网络的设计已经基本完成，但此时可能存在子站信号覆盖范围与其它信号站大范围重叠的情况，我们将这种子站称为冗余子站。为了减少运营商建设网络的总成本，我们对冗余子站进行裁撤，对剩下的子站进行重新连接，以完成更加优化的网络设计。

关键词:Relay 无线回传,ISODATA 分簇,LOF 离群因子,正六边形密排法,Dijkstra 最短路径，冗余子站。

1　问题重述

1.1 问题背景

进入 21 世纪以来，人们对通信的要求有了显著的提高。为了适应这种需求，人们开发出了多种通信技术。其中比较常见的有光纤通信和微波通信两种。但是，这两种通信技术均存在明显局限性。在城市中，光纤的建设成本较高，且到站率低，很多地方由于条件限制无法连接，而微波通信严重依赖 LOS 场景，在障碍众多的城市地区难以承担通信任务。在农村地区，光纤传输建设费用对于运营商是很大的负担，而如果使用微波传输，对于相当一部分站点需要提升铁塔高度来满足微波的 LOS 场景要求，铁塔费用的增加是运营商难以承受的。在此背景下，Relay 回传方案就显示出突出的优越性。通过宿主站和子站组成的拓扑网络结构，可以显著延伸信号覆盖面积，减少距离、地形阻隔等对于信号传输的影响。

1.2 目标任务

在给定一个地区候选站位置的情况下，根据候选站的分布确定合适的 Relay 回传网络。包括判断站点安装宿主站还是子站，以及各个信号站之间的连接方式，使其满足最低回传质量要求，信号站间的距离限制，以及子站的跳数限制。

本文将尝试建立数学模型，解决如下问题：

问题一：在候选站点中，如何选出宿主站。设计算法，使得选出的宿主站的信号覆盖范围既没有大面积的重叠也没有大面积遗漏。同时尽可能使宿主站之间可以相互通信，共用卫星以减少建设卫星设备成本。

问题二：在确定宿主站之后，设计算法，确定剩余的子站的连接方式，使其连接符合基站——子站，子站——子站距离限制，以及拓扑结构限制。同时，尽可能减少信号传输过程中的回传路径损耗。

问题三：在确定了基站和子站连接方式的前提下，不同站点的信号覆盖范围可能存在冗余，对于那些有效覆盖面积很小的子站，应予以裁撤以减小成本。尝试设计算法，对冗余站点进行识别，删除后重新确定剩余站点的连接方式。

2 模型和假设与符号说明

2.1 模型与假设

假设一：所有的备选站点分布于经纬度受限的球面矩形内。

假设二：只要宿主站和子站之间距离满足限制条件，就可满足最低回传质量要求；

假设三：无限回传不存在 NLOS 影响站点之间的路径损耗采用自由空间传播模型。

假设四：不考虑信号感知面积的叠加对信号的增强作用。

2.2 变量说明

变量符号	变量含义	变量单位
α	站点的经度	度
β	站点的纬度	度
R	地球半径	千米
F	发射频率	MHz
L	卫星数量	个
M	宿主站数量	个
N	子站数量	个
C	总体成本	W USD
PL	路径损耗	1
r_s	感知半径	千米
r_c	通信半径	千米

表 1 符号说明

3 问题分析

3.1 对问题一的分析

问题一看似仅仅比受到连接问题的约束条件，但我们认为除了连接约束外，还有另外一个判断宿主站选择的因素——信号覆盖面积。根据相关资料显示，一

个站点有感知半径r_s和通信半径r_c两个半径。两者满足以下关系：

$$r_c \geq 2r_s$$

那么可以很容易联想到如果以感知半径r_s作为半径做圆划分区域，那么通信条件是一定可以满足的。圆形无法密铺，所以我们选择交叠面积较少，利用率高的六边形来操作。所谓正六边形密排法，指的是用图 1 所示的方式的平面区域进行覆盖。

所谓正六边形密排法，指的是用图 1 所示的方式的平面区域进行覆盖

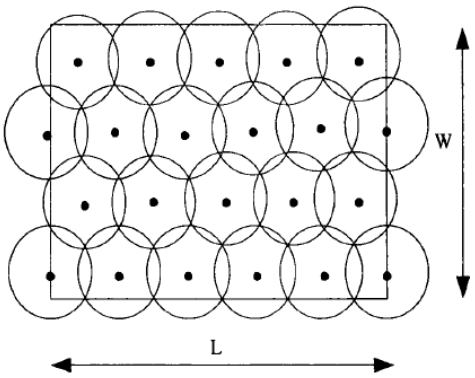


图 1[1] 正六边形密排法-1

将圆域两两相交的区域简化为直线，圆域即转化为正六边形，在空间中一如下方式铺展开来：

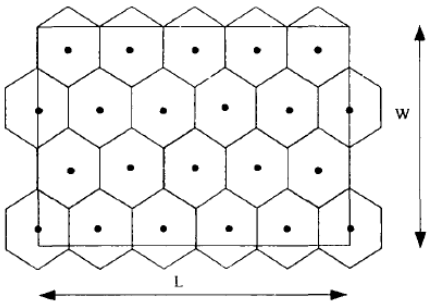


图 2[2] 正六边形密排法-2

这就是理想情况下正六边形平铺法，它可以用最少的发射站来覆盖选定区域。但是随之而来一个问题，用六边形方法需要站点分布在六边形中心，或者退而求其次要求站点分布相对均匀。但是实际情况中，因为城市还是农村，交通好坏等原因，信号站点的分布一定是不均匀的。强行使用六边形密铺效果生硬。所以我们想到应当先对所有点进行聚类分析，分成一簇簇。相对来说，一簇之内的点就相对均匀。

另外，ISO 算法的方案也并不非常完善，有可能会出现孤立点，簇内候选站分布过于零散等问题，我们为应对此情况引入 LOF 离群因子，用来判断簇的优劣。对于极度异常的簇，必须引入人工干预来修改。

3.2 对问题二的分析

在选定了宿主站的情况下，剩余的备选站点默认为子站。这些子站与宿主站之间的连接方式必须满足如下约束条件：

1） 首跳距离≤20km，之后每跳距离≤10km;

2) 宿主站的每个扇区第一级最大接入子站数为 4，最大总接入子站数 6；
3) 每个子站最多只能有 2 条无线回传连接；
4) 任意子站只能归属一个宿主站，到达所属宿主站有且只有一条通路，且该通路，且该通路包含的跳数小于等于 3；

在这写限制条件下，需要找出回传路径损耗最小的最优连接方法。

在此我们采用了 Dijkstra 算法来求得最短路径。我们采用两点欧式距离的以 10 为底的对数作为权重进行排序。我们首先根据约束条件找出可以作为起点的站点以及不能连接的站点，之后选取起点与剩余站点权重最小的作为路径，再返回重新寻找起点和不能连接点，重复此过程直至所有点都连接成功。我们便得到了路径，此路径为局部内损耗最小。

3.3 对问题三的分析

经过问题一和二后，我么已经有了宿主站和子站的选择，以及一套回传损耗最小的连接方式。但由于我们引入了感知半径的概念，使得我们对优化方案有了新的想法——冗余站的概念。根据题目可推知，宿主站的感知半径远大于子站的感知半径，如果某子站离宿主站很近，很有可能出现子站感知面积被宿主站“吞噬”的情况。虽然在现实情况中，感知面积的重叠并非坏事，它可以加强信号强度，增加信号稳定度，但我们假设四规定不考虑感知面积的重叠对信号的加强作用。因此，我们定义这种站为“冗余站”，应当被裁撤。裁撤了“冗余站”不但不会对覆盖面积有过大影响，还可以大大减小成本。

但是裁撤“冗余站”也有需要深思熟虑之处，就是连接路径的重新规划。如果对“冗余站”的定义太过苛刻，及重叠面积没有很大的站也被裁撤，很有可能增加路径连接难度，甚至造成某那些站被“孤立”，因此对“冗余站”的定量定义和裁撤要仔细斟酌。

4 数据的产生

站点分布密度与人口密度密切相关。在人口密集的地区，比如都市、城镇，站点分布非常集中，而在山区、乡村等人口相对稀少的地区，站点分布稀疏。为了适应现实当中的人口分布严重不均的情况，我们引入了如下生成方法。

我们调查了陕西省各地级市的人口分布及地级市的地理坐标，见表 2。

地级市	人口/万	坐标
西安市/渭南市/咸阳市	1852	（108.95 °E,34.27 °N）
宝鸡市	376	(107.15 °E,34.38 °N)
汉中市	348	(108.04 °E,33.07 °N)
榆林市	330	(109.77 °E,38.30 °N)
安康市	265	(109.02 °E,32.70 °N)
商洛市	238	(109.93 °E,33.87 °N)
延安市	212	(109.47 °E,36.60 °N)

表 2[3] 陕西省各地级市人口分布及地理坐标

对上述表格中的人口中心进行简化，用二维高斯函数来代表每一个简化过的人口中心，将所有高斯曲线叠加，并进行归一化，得到分别关于纬度和经度的站点分布概率密度函数，从而随机产生带有特定概率分布的候选站点分布图。

$$y1 = \frac{\psi_1}{\sqrt{2\pi\sigma_1}} \exp \sum \left(-\frac{(x1-\mu_1)^2}{2{\sigma_1}^2} \right) \qquad y2 = \sum \frac{\psi_2}{\sqrt{2\pi\sigma_2}} \exp(-\frac{(x1-\mu_2)^2}{2{\sigma_2}^2})$$

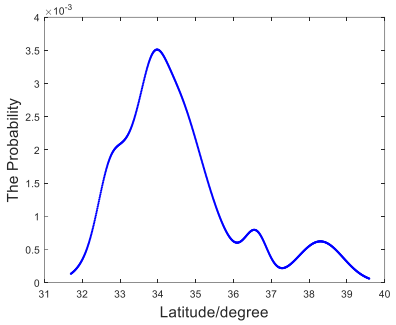


图 3 纬度分部概率密度函数

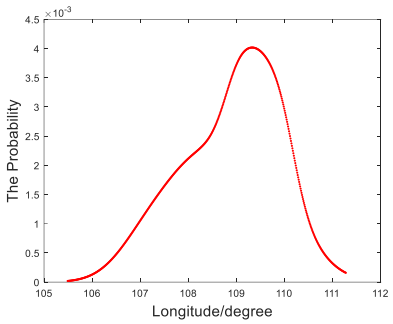


图 4 纬度分部概率密度函数

求和系数：

ψ_1	μ_1	σ_1
5	34.27	1
0.8	34.38	0.6
0.8	33.07	0.6
0.8	38.30	0.6
0.4	32.70	0.3
0.4	33.87	0.3
0.4	36.60	0.3

表 3 纬度正态分布相关参数

ψ_2	μ_2	σ_2
5	108.95	1
1	107.15	0.6
1	108.04	0.6
1	109.77	0.6
0.8	109.02	0.3
0.8	109.93	0.3
0.8	109.47	0.3

表 4 经度正态分布相关参数

按照上述概率密度函数随机生成 1000 个候选站点。如下图所示：

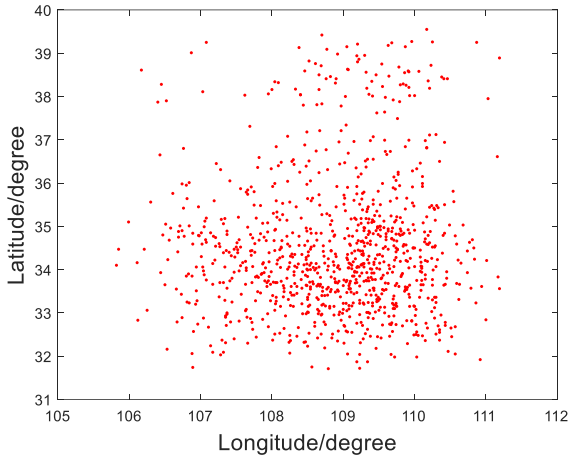
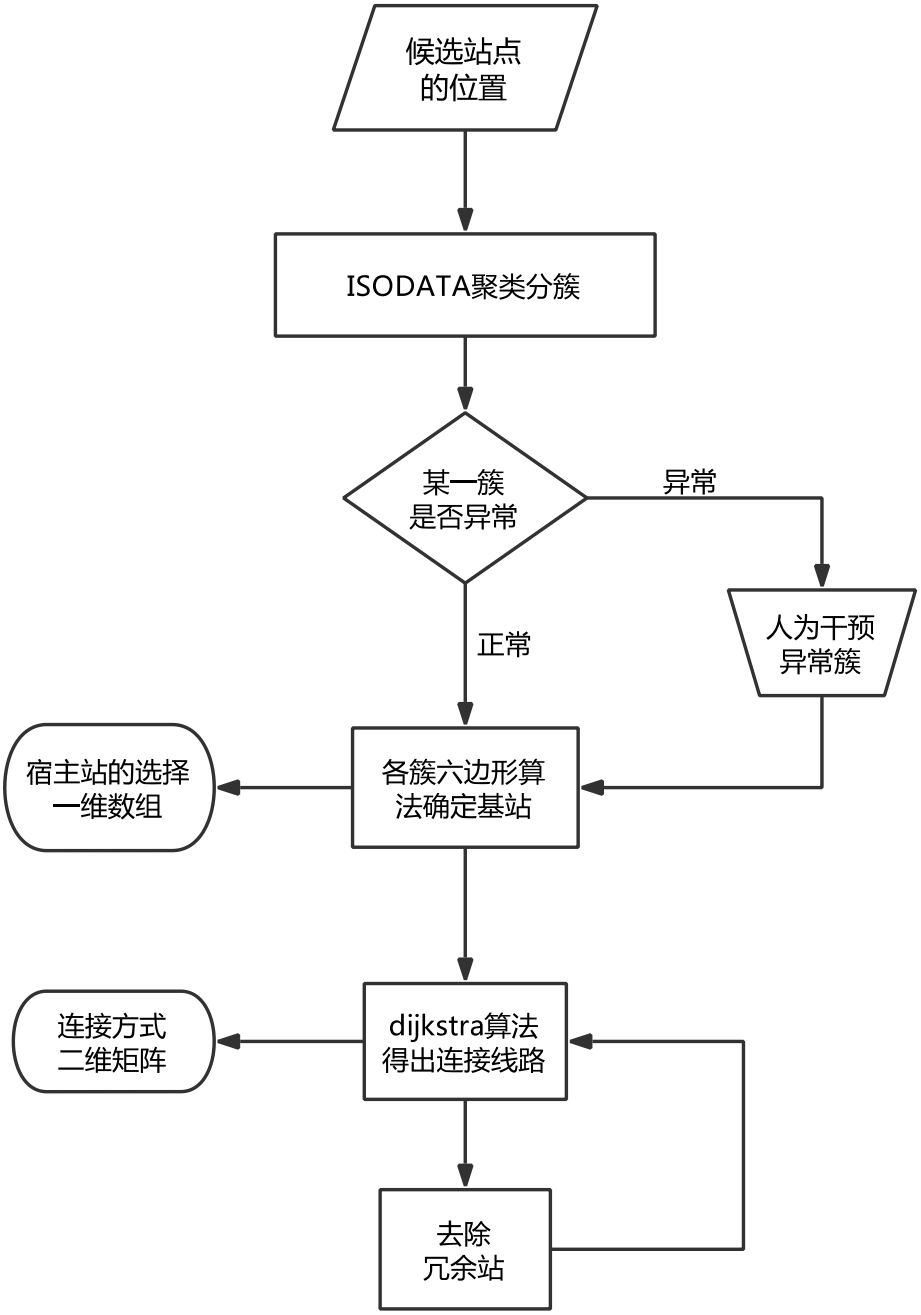


图 5 随机产生的站点图

上图显示的即是我们的初始数据，在下文中，我们将以此为基础，在候选站点中选取基站，决定子站的连接方式，并去除冗余的子站。

5　模型的建立与求解



流程图 1 问题整体流程图

5.1 问题一模型的建立和求解

之前的问题分析阐明，由于候选信号站分布本身是不均匀的，整个 1000 个候选点形成的总感知区域存在天生的感知盲区。如果我们贸然用六边形算法一味地追求全覆盖，会大大增加整套算法的工作量，并且为了尽可能满足全覆盖，该方法需要额外大量选择宿主站，也大大增加了成本。所以在这之前，根据信号站地理位置的分簇极为关键。

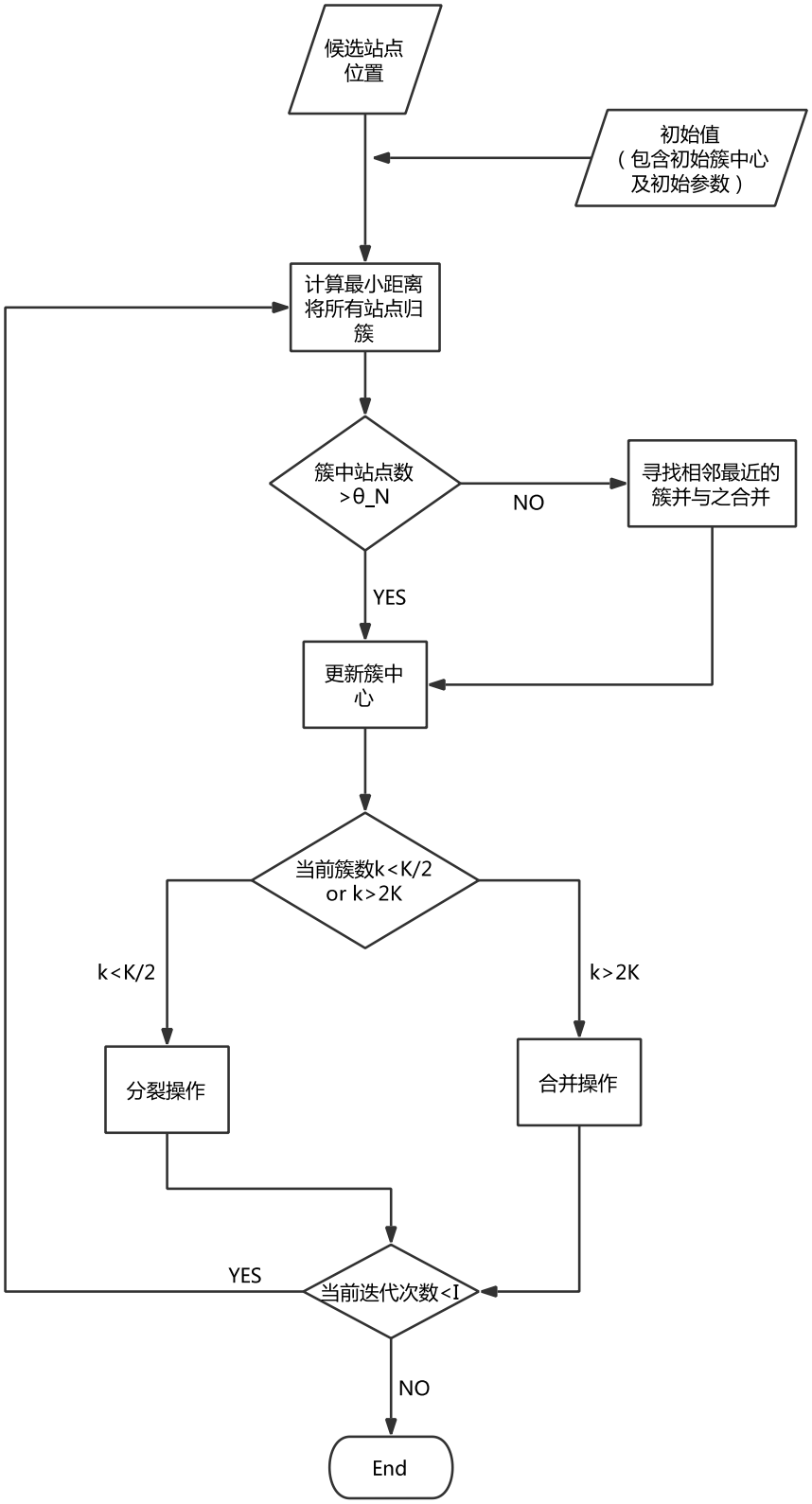
5.1.1ISODATA 聚类将候选信号站分簇

ISODATA 常用参数说明见下表。

符号	解释
K	期望的簇数
k	初始给定的簇数
θ_N	单个簇中最少的信号站数，若少于则去掉该簇
θ_S	单个簇中允许的最大标准差，若大于这个值则可能继续分裂
θ_C	两个簇中心的最小距离，若小于这个值则两簇可能合并为一
L	在一次合并操作中，可以合并的簇的对数的最大值
I	迭代运算的次数

表 5 ISODATA 符号说明

我们列出 ISODATA 算法的流程图，见下图。



流程图 2 ISODATA 算法流程

5.1.1.1 模型的建立

对于目标函数，ISODATA 的目标函数和 K-Means 的目标函数是一样的：

$$\arg\min_{S}\sum_{i=1}^{k}\sum_{x_j\in S_i}||x_j-c_i||^2$$

如流程图所示，我们先定义簇的个数（大致值），以及相应指标参数的初始值，根据距离公式计算每一个点到所有中心的距离，取最小的簇中心归于那一簇。按照一簇中最少个数指标合并数量过少的簇，然后用上述公式得出每一簇的新的簇中心。然后反复迭代，直至迭代次数。

✧ ***LOF 离群因子***[8]

下面介绍 LOF 算法的相关定义：

1) $\boldsymbol{d(p,o)}$：两点 p 和 o 之间的距离；

2) **k-distance：第 k 距离**

对于点 p 的第 k 距离$\mathrm{d}_k(\mathrm{p})$定义如下：

$dk(p)=d(p,o)$，并且满足：

a) 在集合中至少有不包括 p 在内的 k 个点$o,\in C\{x\neq p\}$,满足$d(p,o,)\leq d(p,o)$；

b) 在集合中最多有不包括 p 在内的 k−1 个点$o,\in C\{x\neq p\}$,满足$d(p,o,)<d(p,o)$；

p 的第 k 距离，也就是距离 p 第 k 远的点的距离，不包括 p，如图 6。

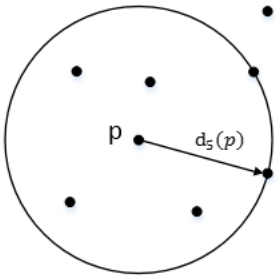


图 6 第 k 距离

3) **k-distance neighborhood of p：第 k 距离邻域**

点 p 的第 k 距离邻域$Nk(p)$，就是 p 的第 k 距离即以内的所有点，包括第 k 距离。

因此 p 的第 k 邻域点的个数 $|Nk(p)|\geq k$。

4) **reach-distance：可达距离**

点 o 到点 p 的第 k 可达距离定义为：

$$reach-distancek(p,o)=max\{k-distance(o),d(p,o)\}$$

也就是，点 o 到点 p 的第 k 可达距离，**至少**是 o 的第 k 距离，或者为 o、p 间的真实距离。

这也意味着，离点 o 最近的 k 个点，o 到它们的可达距离被认为相等，且都等于$dk(o)$。

如图 7，o1 到 p 的第 5 可达距离为$d(p,o1)$，o2 到 p 的第 5 可达距离为$d5(o2)$。

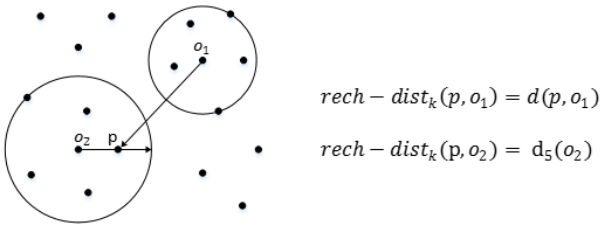


图 7 可达距离

5) local reachability density：局部可达密度

点 p 的局部可达密度表示为：

$$lrd_k(p)=1/(\frac{\sum_{o\in N_k(p)} reach-dist_k(p,o)}{|N_k(p)|})$$

表示点 p 的第 k 邻域内点到 p 的平均可达距离的倒数。

注意，是 p 的邻域点$Nk(p)$到 p 的可达距离，不是 p 到$Nk(p)$的可达距离，一定要弄清楚关系。并且，如果有重复点，那么分母的可达距离之和有可能为 0，则会导致 lrd 变为无限大，下面还会继续提到这一点。

这个值的含义可以这样理解，首先这代表一个密度，密度越高，我们认为越可能属于同一簇，密度越低，越可能是离群点。如果 p 和周围邻域点是同一簇，那么可达距离越可能为较小的$dk(o)$，导致可达距离之和较小，密度值较高；如果 p 和周围邻居点较远，那么可达距离可能都会取较大值$d(p,o)$，导致密度较小，越可能是离群点。

6) local outlier factor：局部离群因子

点 p 的局部离群因子表示为：

$$LOF_k(p)=\frac{\sum_{o\in N_k(p)}\frac{lrd_k(o)}{lrd_k(p)}}{|N_k(p)|}=\frac{\sum_{o\in N_k(p)} lrd_k(o)}{|N_k(p)|}/lrd_k(p)$$

表示点 p 的邻域点$Nk(p)$的局部可达密度与点 p 的局部可达密度之比的平均数。

如果这个比值越接近 1，说明 p 的其邻域点密度差不多，p 可能和邻域同属一簇；如果这个比值越小于 1，说明 p 的密度高于其邻域点密度，p 为密集点；如果这个比值越大于 1，说明 p 的密度小于其邻域点密度，p 越可能是异常点。

现在概念定义已经介绍完了，现在再回过头来看一下 lof 的思想，主要是通过比较每个点 p 和其邻域点的密度来判断该点是否为异常点，如果点 p 的密度越低，越可能被认定是异常点。至于密度，是通过点之间的距离来计算的，点之间距离越远，密度越低，距离越近，密度越高，完全符合我们的理解。而且，因为 lof 对密度的是通过点的第 k 邻域来计算，而不是全局计算，因此得名为“局部”异常因子。

5.1.1.2 模型的求解与结果

在我们实际操作中，我们定义迭代的次数为 30，下面是第 1，10，20，30 次迭代后的图像，可以清晰地看出 IOSDATA 的效果。

在如此分簇的基础上，我们满足了可以使用六边形算法的条件。

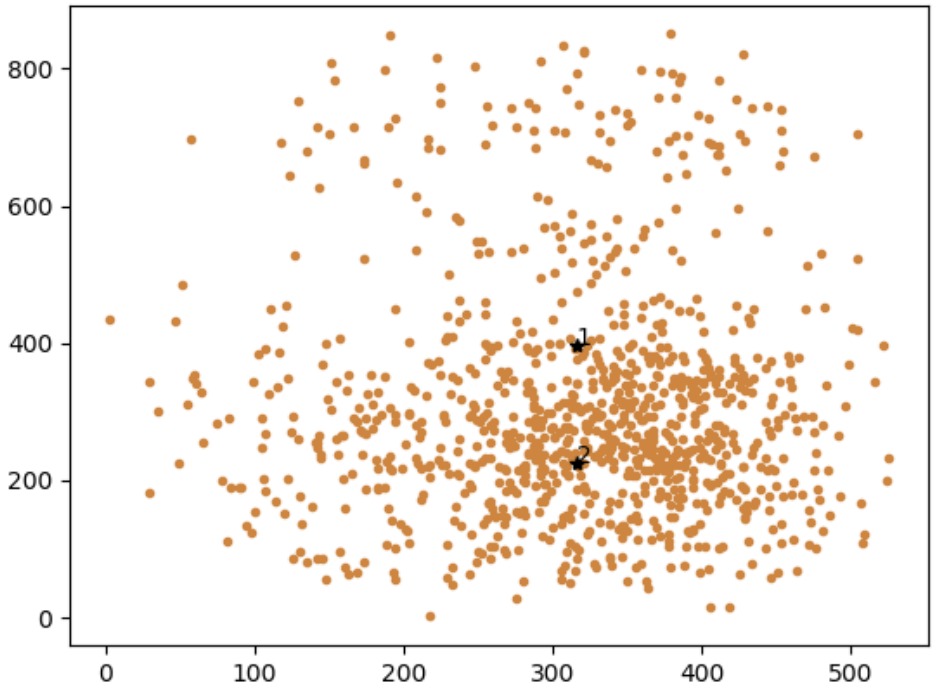


图 8 第 1 次迭代

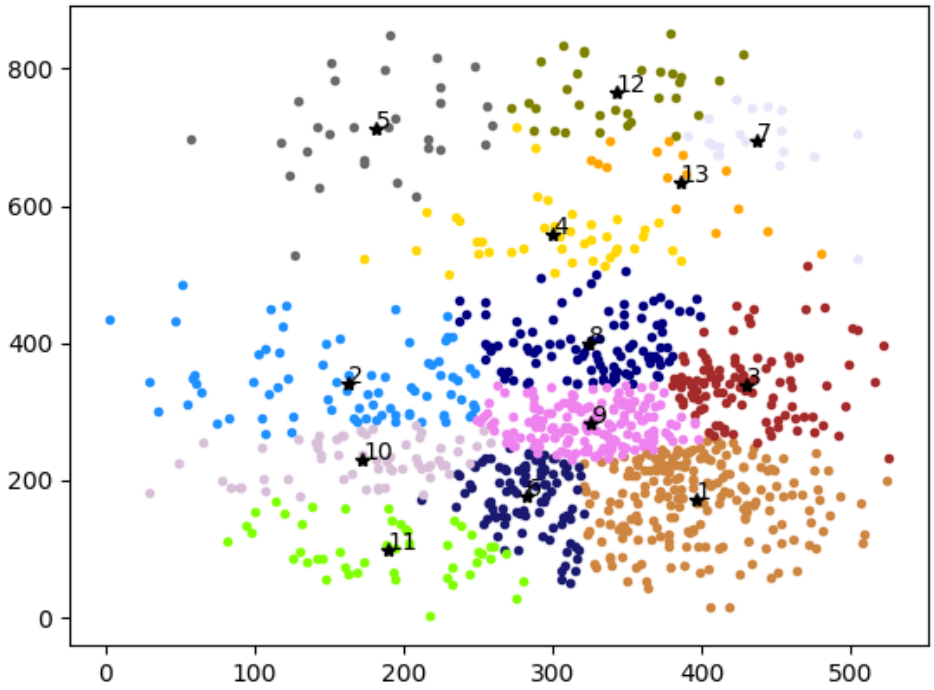


图 9 第 10 次迭代

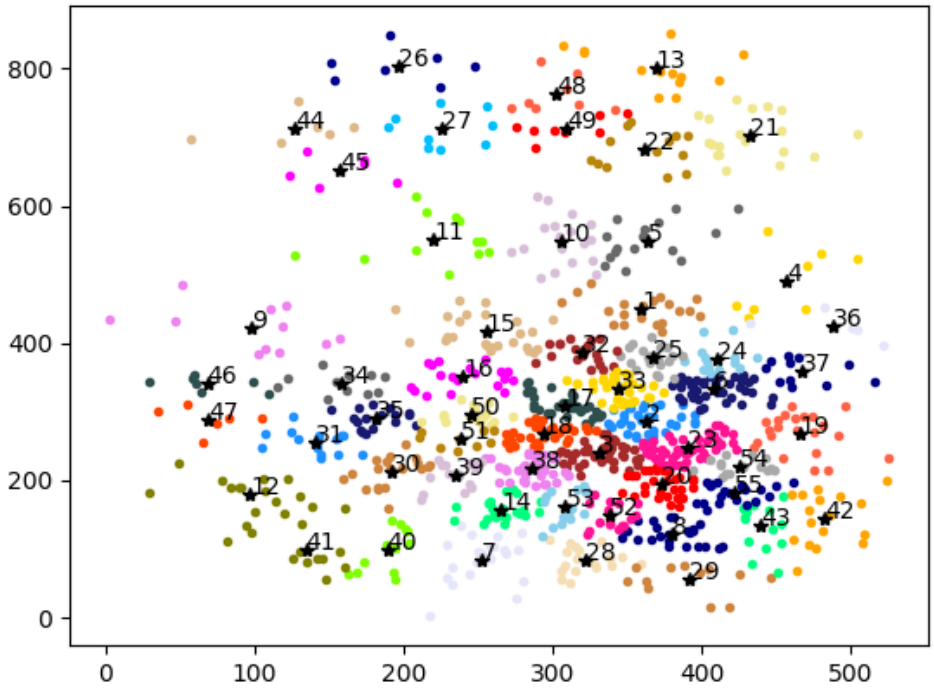


图 10 第 20 次迭代

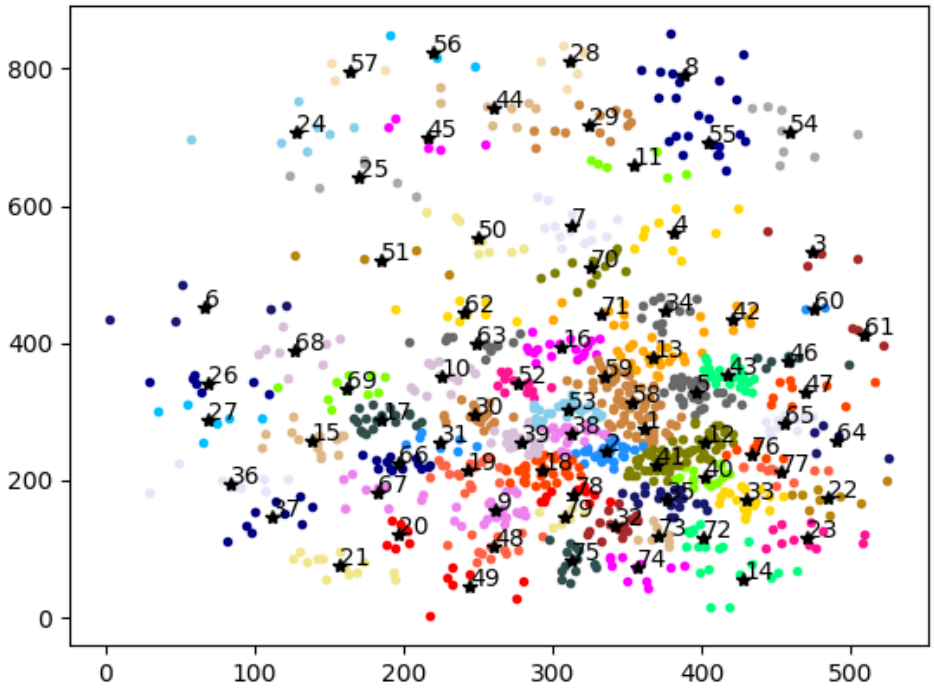
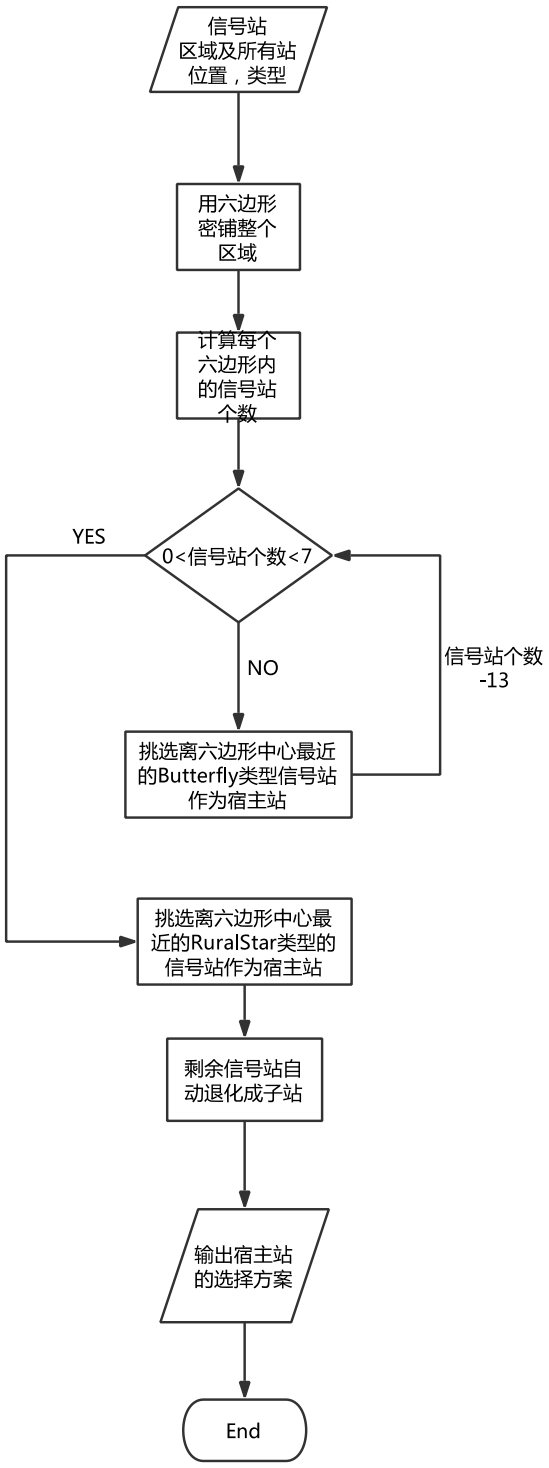


图 11 第 30 次迭代

5.1.2 六边形算法确定宿主站

下面讨论均匀的情况。按照算法，在经过 ISODATA 分簇后，每一簇都可以看作是相对均匀的情况，也就适用于六边形算法确定宿主站。下面是六边形算法的简要流程图。



流程图 3 六边形算法

5.1.2.1 模型的建立

按照流程图所示，我们在取得一定区域内所有信号点的位置后，用六边形密铺整个区域。题目中所给出的宿主站的通信半径为 50km，按照$r_c > 2r_s$的规律，我们定义宿主站的感知半径$r_{s宿主}$=20km，子站的感知半径 $r_{s子}$=5km。所以我们选择六边形边长为 20km，如果每一个六边形内都保证有一个宿主站，就大致可以保证能全覆盖。

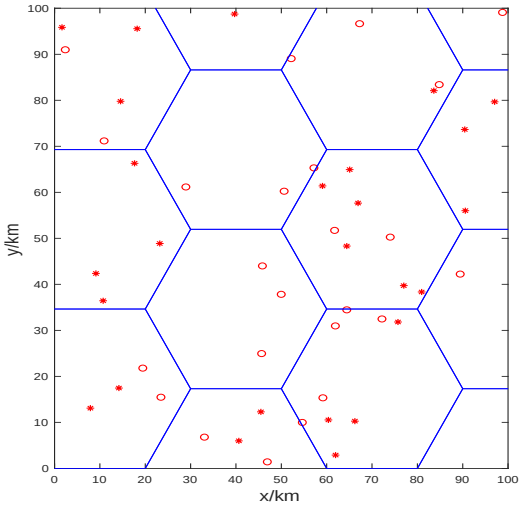


图 12 六边形密排示意图

✧ ***计算在某一六边形内的点的个数***

在六边形算法中很重要的判断依据是一个六边形内的候选信号站的个数。已知六边形的中心点坐标$O(x_0,y_0)$，边长为d，和待判断的点坐标$M(x_1,y_1)$首先我们先做一个六边形的外切矩形。判断该点是否在外切矩形内。

$$\text{在外切矩形内} \Leftrightarrow \begin{cases} x_0 - d < x_1 < x_0 + d \\ y_0 - \dfrac{\sqrt{3}}{2}d < x_1 < y_0 + \dfrac{\sqrt{3}}{2}d \end{cases}$$

当该点在六边形外切矩形中时，再用 MP 和 MN 的长度来判断。如图所示，当 MP>MN 时，该点在六边形内。其中$MN = \frac{|y_0 - y_1|}{\sqrt{3}}$。

✧ ***宿主站的选择***

按照流程图所示，我们先对六边形内的候选信号站个数进行分类。以 7 为分界的原因是，一个 RuralStar 类型的宿主站最多可以连接 6 个子站，所以理想情况下，不考虑连接的可行性问题，个数≤ 7的六边形只要有一个 RuralStar 类型的宿主站即可。（我们也暂时不考虑该六边形内有没有 Ruralstar 候选信号站的问题）如果个数超过 7，则选择一个 Buttherfly 类型的宿主站，考虑到宿主站成本是子站成本的 2 倍，理想情况下一个 Butterfly 类型的宿主站可以链接 12 个子站，所以每次选择一个 Butterfly 类型的宿主站，我们默认待分配的点个数减少（12+1）=13 个。

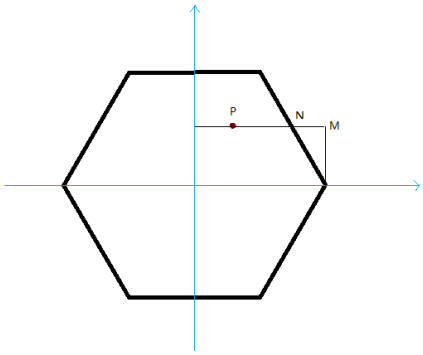


图 13 判断点与六边形关系示意图

✧ ***需要额外考虑的异常情况***

实际操作过程中我们发现，由于所有候选信号站都是用一定概率分布的随机函数随机产生的，点的分布有时极为尴尬，产生需要我们额外考虑甚至是需要人工干预的情况。

（1）**一个六边形中只有一个点。**在我们的模型中，我们假设并默认不把两个不同六边形内的点相互连接，因此这种情况只能选择这个唯一点成为宿主站。但是如果该地区不是一个需要大面积覆盖信号的允许盲区呢？那么也许这个站点的存在并不是很有必要，换成一个子站与其最近邻的站连接，或者无法实现时直接放弃这个候选点，我们认为也是完全可行的。

（2）**一个六边形中站点个数在 0～7 之间，但没有 RuralStar 类型的信号站。**对于这种情况，按照我们的流程图将会是没有宿主站生成。但是实际情况下这种情况就很容易解决，选取离六边形中心最近的 Butterfly 类型的信号站作为宿主站即可。

（3）**一个六边形中站点个数>7，但是没有 Butterfly 类型的信号站。**这种情况也是违反我们算法的异常情况，但是实际很好解决。按照将（个数 / 7）向上取整得到 RuralStar 类型的宿主站的个数。选择最接近中心的相应个数的信号站称为宿主站。但是实际操作中我们发现这种情况几乎不会发生。

虽然以上三种情况都是违反我们六边形算法的异常情况，但我们编写了相应的应对代码处理以上几种情况。另外，值得一提的是，这些异常情况是小概率事件，不应该称成为算法的核心问题。

5.1.2.2 模型的求解与结果

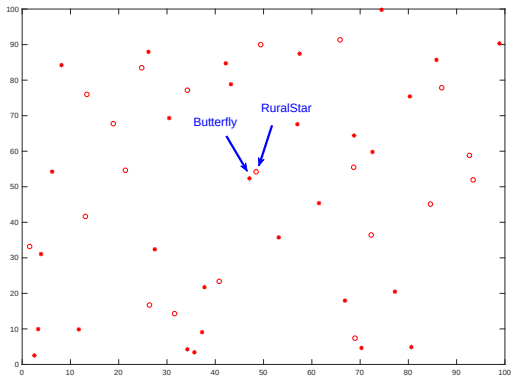


图 14 候选站不同类型分布图

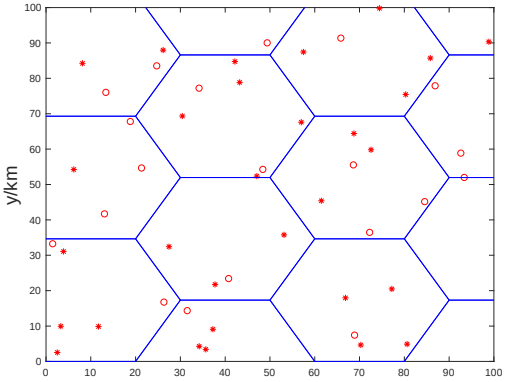


图 15 六边形密排

我们选择在 200km*200km 的区域随机生成 50 个候选站，选择宿主站。

上面图 14 是随机生成的 50 个点，分别有 1～50 的序号标记，实心红点表示 Butterfly 类型，空心表示 RuralStar 类型。图 15 是六边形划分区域后的情况。可以看到 200km*200km 选取 50 个点带来的一个六边形内的点的个数在 0～10 之间分布。这也印证上面提到的六边形内点过密是小概率事件。下面我们按照我们的算法选择宿主站。

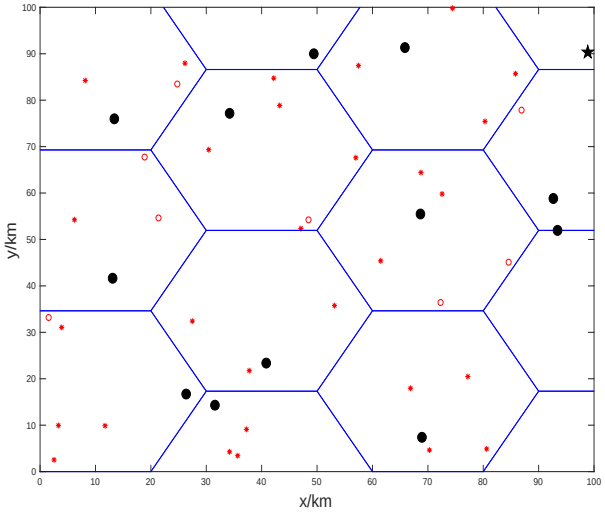


图 16 选择宿主站

其中黑色五角星表示 Butterfly 类型的宿主站，黑色实心大圆表示 RuralStar 类型的宿主站。

按照题目的格式我们给出宿主站选择的一维数组（1～50）：

序号	类型	宿主站选择	序号	类型	宿主站选择	序号	类型	宿主站选择
1	0	1	18	1	0	35	1	0
2	1	0	19	1	0	36	0	1
3	1	0	20	0	1	37	1	0
4	0	1	21	1	0	38	0	1
5	1	0	22	0	0	39	1	0
6	1	0	23	1	0	40	1	0
7	0	0	24	1	0	41	0	1
8	0	1	25	0	0	42	0	1
9	1	0	26	0	0	43	1	0
10	1	0	27	1	0	44	1	0
11	1	0	28	1	0	45	0	1
12	1	0	29	0	0	46	0	1
13	1	0	30	0	0	47	1	0
14	0	1	31	0	1	48	1	0
15	1	1	32	1	0	49	1	0
16	0	0	33	1	0	50	0	0
17	1	0	34	1	0			

表 6 宿主站选择情况

5.2 问题二的求解

5.2.1Dijkstra 算法模型建立

Dijkstra 算法采用的是一种贪心的策略，声明一个数组 D_ij 来保存源点到各个顶点的最短距离和一个作为七点的顶点的集合：start，初始时，对于顶点 s 存在能直接到达的边（s,m），则把 D_ij[m]设为 w（s,m）,同时把所有其他（s 不能直接到达的）顶点的路径长度设为无穷大。初始时，集合 start 只有顶点 s。

然后，从 D_ij 数组选择最小值，则该值就是源点 s 到该值对应的顶点的最短路径，并且重新判断可以作为起点的顶点，把该点加入到 start 中，此时完成一次循环，然后，我们需要看看新的起点到其他需要到达的点的路径权重，找到最小值并完成循环。

重复上述动作，完成路径。

5.2.1.1 模型的求解与结果

由于 1000 个点数量庞大，我们在这里用 10 个点来展示该算法的实际操作结果。**第 1 个点为基站，其余为子站。**

经过计算，我们得到 10*10 的矩阵，其中 0 表示无连接，1 表示连接。

0 0 1 0 1 0 1 0 1 1
0 0 0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0 0 0
1 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
1 0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0

各点的坐标为：

60.9 59.0
22.8 85.7
80.0 55.7
62.7 106.2
86.6 88.9
52.6 21.4
57.9 26.6
88.1 81.7
31.8 64.4
77.1 20.0

5.3 冗余站点裁撤模型的建立

关于冗余信号站的裁撤是本文的创新之处。除了考虑连接方式是否符合约束外，我们还追求各信号站感知范围的少重叠，大覆盖。但是由于候选信号站数据的缺陷，将会有很多信号站（尤其是子站），其感知面积被其他信号站的感知面积“吞噬”，造成该信号站称为“冗余信号站”。对“冗余信号站”的裁撤会大大降低总体成本而不会对该地区信号覆盖面积产生不良影响。

5.3.1 冗余率模型的建立

每一个信号站的感知范围都可以用一个以信号站为圆心，感知半径r_s为半径的圆域C_i来表示。各信号站的冗余率可用其圆域与其它信号站圆域重叠面积占其感知面积的比值来度量。

实际情况中，一个信号站的感知面积很可能与多个信号站重叠，但是 3 个及以上半径待定的圆相互有重叠的情况太过复杂，大部分情况下都选择求出近似的数值解，解析方法计算量非常庞大。为了简化模型，我们采用两两比较的方法计算冗余率，并且给出相应的判断是否应该被裁撤的依据。

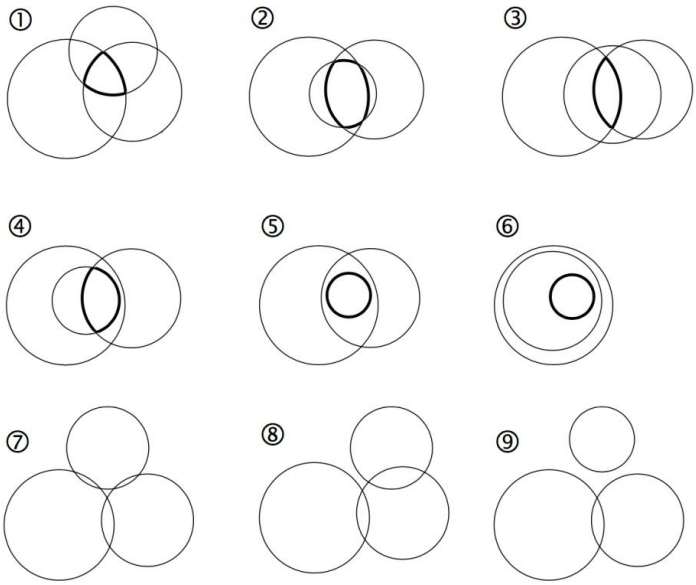


图 17 圆的关系

对于二圆的情况，圆域之间只有五种几何关系：内含、内切、相交、外切、相离。其中内含和内切属于完全重叠，相交属于部分重叠，外切和相离属于完全不重叠。

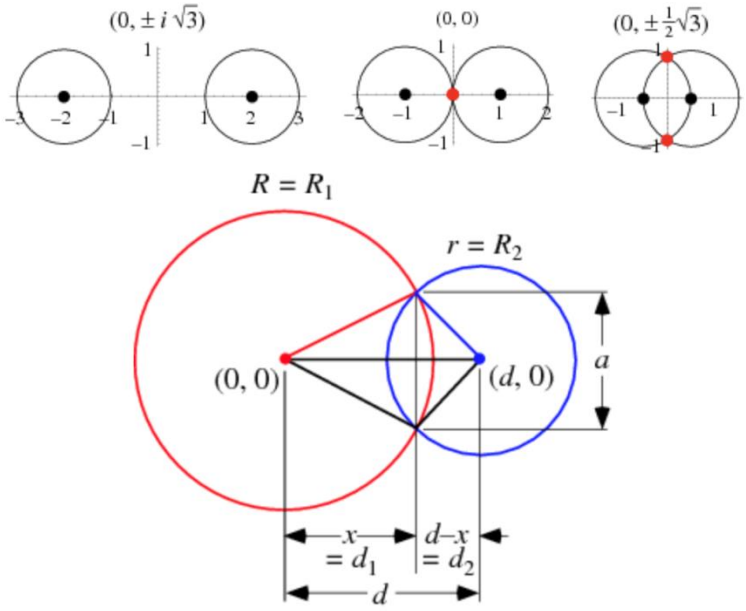


图 18 计算方法

定义 Y 为冗余率

（1）当$d \geq \boldsymbol{R_1 + R_2}$时　Y=0；

（2）当｜$d \leq |\boldsymbol{R_1 - R_2}|$｜时　Y=1；

（3）其他情况：

$$\boldsymbol{\theta_1} = \mathbf{arccos}(\frac{\boldsymbol{R_1^2 + d^2 - R_2^2}}{\boldsymbol{2R_1 d}})$$

$$\boldsymbol{\theta_2} = \mathbf{arccos}(\frac{\boldsymbol{R_2^2 + d^2 - R_1^2}}{\boldsymbol{2R_2 d}})$$

$$\boldsymbol{h_1} = (\frac{\boldsymbol{R_1^2 + d^2 - R_2^2}}{\boldsymbol{2d}})$$

$$\boldsymbol{h_2} = (\frac{\boldsymbol{R_2^2 + d^2 - R_1^2}}{\boldsymbol{2d}})$$

$$\mathbf{m} = \boldsymbol{R_1}\sqrt{\mathbf{1} - \frac{\boldsymbol{cos\theta_1}}{\mathbf{2}}};$$

$$\mathbf{Y} = \frac{\boldsymbol{\theta_1 R_1^2 + \theta_2 R_2^2 - m(h_1 + h_2)}}{\boldsymbol{\pi R_1^2}}$$

5.3.2 模型的求解与结果

见图 19。

以同上 50 个点为例，我们建立一个 50*50 的二维矩阵，元素（i，j）表示第 i 个站和第 j 个站交叠感知面积占第 i 个站感知总面积（以感知半径为半径的圆）的比例。因此这个矩阵并不是对称矩阵，因为 i 和 j 与 j 和 i 交叠面积虽然一样，但作为分母的面积不一样，分别是 i 和 j 的感知面积。

另外根据我们的算法，自己与自己的交叠面积比例为 1，这也可以以下的 50*50 的矩阵看出，对角元全是 1。

如何判断冗余点呢？我们给定一个 0 到 1 之间的数，作为判断冗余的指标，除对角元外，如果某一点对应的列向量有超过该指标的元素则说明该点是冗余站。这个指标可以在一定程度浮动，但一般选择 1。

而且通过模型我们发现，分布密度不太大时，子站与子站之间有较大交叠面积的可能性比较小低，大多是子站被基站“吞噬”的情况。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU	AV	AW	AX
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0.8	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0.1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0.9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.9	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0.8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	1	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0	0.3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0.1	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0.3	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0.3	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0.4	0	0	0.6	0	0	0	1	0	0	0	0	0	0	0
44	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0.9
45	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
46	0.3	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0	0	0.8	0.7	0	0	0	0	0	0.7	0	0	0	0	0	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0.1	0
48	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0	0	1	0	0	0	0.2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
49	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8	0	1	0
50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0	1

图 19 冗余点判断

6　模型结果的评价与分析

6.1 总成本

总成本=6420W USD；

总损耗经过 Dijkstra 算法已经降到局部最低。

总计算时间：3 分钟左右处理 1000 个站点。

6.2 模型的优点

（1）产生的数据以实际人口分布情况为依据，使得模型在现实中具有更好的通用性和推广性；

（2）用聚簇的思想对候选的站点进行分割处理，很好地解决了站点分布不均带来的问题，大大简化了问题；

(3)选取基站时，最大限度保证了基站之间的距离少于 50km，使其能够以微波通信的方式相互连接，共用卫星信号，减少卫星建设成本；

(4)在解决子站连接问题时，充分考虑了损耗问题，采取了平均损耗相对较小的连接方式，提高了信号质量；

(5)对于信号覆盖范围重叠范围过大的子站进行了裁撤，兼顾了信号质量和建站成本。

6.3 模型的缺点

(1)在分簇这一步操作中，存在部分异常的近均匀分布簇和离散点，这类异常无法用算法解决，必须引入人为干涉来解决问题；

(2)连接子站时，对于一些基站过远的子站因距离和跳数的限制无法连接，只能放弃；

(3)在判断冗余子站的过程中，未能辨别哪些子站的信号范围被其它发射站完全覆盖。在删除的过程中可能造成信号覆盖范围减小和通信质量变差。

(4)在删除冗余子站后，必须重新决定连接方法，并用程序对冗余子站进行判断，这将导致程序收敛速度慢，不能很快得出最终结果。

7　参考文献

[1]赵国炳 陈国定 《无线传感器网络覆盖问题研究》 www.cnki.com 2018 年 4 月 30 日；

[2] https://blog.csdn.net/cuixiping/article/details/11716805, 高效判断点是否在正六边形蜂窝内的方法 - CSDN 博客

[3] 陕西省 百度百科 https://baike.baidu.com/item/陕西/193811?fromtitle=陕西省&fromid=19657132&fr=aladdin；

[4]Peter Harrington 《Machine Learning in Action》 北京 人民邮电出版社 2013 年 6 月第 1 版；

[5] https://blog.csdn.net/wangyibo0201/article/details/51705966 2018 年 5 月 1 日；

[6]崔逊学 黎明曦 《无线传感器网络的覆盖优化机制研究》 《中兴通讯技术》第 11 卷 第 4 期 页 62 至页 65；

[7]王芳 《无线传感器网络覆盖的数学模型问题研究》www.cnki.com 2018 年 4 月 30 日；

[8] https://blog.csdn.net/wangyibo0201/article/details/51705966, 异常点/离群点检测算法——LOF - CSDN 博客
[9] https://blog.csdn.net/liangdas/article/details/39805815, ISODATA 分类法 - CSDN 博客
[10]http://mathworld.wolfram.com/Circle-CircleIntersection.html , Circle-Circle Intersection -- from Wolfram MathWorld
[11] Fewell M P, Division M O. Area of common overlap of three circles[C]// Technical Note DSTO-TN-0722, Defence Science and Technology Organization. 2006.
[12] https://blog.csdn.net/sinat_33829806/article/details/78387993?locationNum=1&fps=1, Machine Learning 之 LOF 离群点检验 - CSDN 博客
[13] https://github.com/molivia/isodata, molivia/isodata

8 附录

附录 1 产生一定概率分布的候选站点数据的代码

```
%% superimposed normal distribution of probabality(using Longitude & Latitude)
% Shaanxi (105.4864,111.2661)E (31.7030,39.5864)N
% CITY LONGITUDE LATITUDE------COEFFICIENT

% Xi'an 108.95E 34.27N-----6

% Baoji 107.15E 34.38N-----1
% Hanzhong 108.04E 33.07N------1
% Yulin 109.77E 38.30N-----1

% Ankang 109.02E 32.70N-----0.8
% Shangyu 109.93E 33.87N-----0.8
% Yaan 109.47E 36.60N-----0.8
format long
x1=105.49:0.01:111.27;
x2=31.70:0.01:39.59;
y1=5*normpdf(x1,108.95,1)+normpdf(x1,107.15,0.6)+normpdf(x1,108.04,0.6)+normpdf(x1,109.77,0.6)+0.8*normpdf(x1,109.02,0.3)+0.8*normpdf(x1,109.93,0.3)+0.8*normpdf(x1,109.47,0.3);
y2=5*normpdf(x2,34.27,1)+0.8*normpdf(x2,34.38,0.6)+0.8*normpdf(x2,33.07,0.6)+0.8*normpdf(x2,38.30,0.6)+0.4*normpdf(x2,32.70,0.3)+0.4*normpdf(x2,33.87,0.3)+0.4*normpdf(x2,36.60,0.3);
y1=y1/sum(y1);
y2=y2/sum(y2);
figure
plot(x1,y1,'r.')
```

```
23.
24. xlabel('Longitude/degree','FontSize',15);ylabel('The Probability','FontSize'
    ,15);
25. figure
26. plot(x2,y2,'b.')
27. xlabel('Latitude/degree','FontSize',15);ylabel('The Probability','FontSize',
    15);
28.
29. %% create the longitude and latitude of dots
30. alphabet1=105.49:0.01:111.27;
31. alphabet2=31.70:0.01:39.59;
32. x=randsrc(1000,1,[alphabet1;y1]);
33. y=randsrc(1000,1,[alphabet2;y2]);
34. figure
35. plot(x,y,'r.')
36. xlabel('Longitude/degree','FontSize',15);
37. ylabel('Latitude/degree','FontSize',15);
38.
39.
40. %% superimposed normal distribution of probabality (using Euclidean space)
41. % x range(0,528)km,yrange(0,877)km
42. % Xi'an 108.95E 34.27N-----6 (316.8,285.7)
43.
44. % Baoji 107.15E 34.38N-----1 (152.9,297.9)
45. % Hanzhong 108.04E 33.07N------1 (266.7,152.3)
46. % Yulin 109.77E 38.30N-----1 (391.4,733.6)
47.
48. % Ankang 109.02E 32.70N-----0.8 (323.2,111.2)
49. % Shangyu 109.93E 33.87N-----0.8 (406.0,241.2)
50. % Yaan 109.47E 36.60N-----0.8 (364.1,544.7)
51. format long
52. x1=0:0.1:528;
53. x2=0:0.1:877;
54. y1=5*normpdf(x1,316.8,100)+normpdf(x1,152.9,60)+normpdf(x1,266.7,60)+normpdf
    (x1,391.4,60)+0.8*normpdf(x1,323.2,30)+0.8*normpdf(x1,406.0,30)+0.8*normpdf(
    x1,364.1,30);
55. y2=5*normpdf(x2,285.7,100)+0.8*normpdf(x2,297.9,60)+0.8*normpdf(x2,152.3,60)
    +0.8*normpdf(x2,733.6,60)+0.4*normpdf(x2,111.2,30)+0.4*normpdf(x2,241.2,30)+
    0.4*normpdf(x2,544.7,30);
56. y1=y1/sum(y1);
57. y2=y2/sum(y2);
58. figure
59. plot(x1,y1,'r.')
60. xlabel('x/km','FontSize',15);ylabel('The Probability','FontSize',15);
```

```
figure
plot(x2,y2,'b.')
xlabel('y/km','FontSize',15);ylabel('The Probability','FontSize',15);

%% create the longitude and latitude of dots
alphabet1=0:0.1:528;
alphabet2=0:0.1:877;
x=randsrc(1000,1,[alphabet1;y1]);
y=randsrc(1000,1,[alphabet2;y2]);
figure
plot(x,y,'r.')
xlabel('x/km','FontSize',15);
ylabel('y/km','FontSize',15);
```

附录 2 六边形算法（用到的自编的 function 随后）的代码

```
%% rand the latitude and longitude of the dots
% use randi([0,1]) to create the style of the dot
% 1 represent betterfly style and 0 ruralstar
figure
side=100;          % define the size of the area
data=rand(50,2)*side;
style=randi([0,1],50,1);
%data3=[data;style];
%plot(x,y,'r.')
axis([0,side,0,side])


%% Hypordistribution
% Shaanxi (105.4864,111.2661)E (31.7030,39.5864)N
Longitude=105.49+rand(1000,1)*(111.27-105.49);
Latitude=31.70+rand(1000,1)*(39.59-31.70);
figure
plot(Longitude,Latitude,'r.')
xlabel('Logitude/degree','FontSize',15);
ylabel('Latitude/degree','FontSize',15)

%% show the style of all the dots
% use * to show butterfly style and o ruralstar.
figure
for x=1:50
    if style(x)==1
        plot(data(x,1),data(x,2),'r*')
```

```
        hold on
    else
        plot(data(x,1),data(x,2),'ro')
        hold on
    end
end
axis([0,side,0,side])
%% hexagon with the d=20
d=20;
Nx=ceil(side/((1+sqrt(3))/2*d));
Ny=ceil(side/(sqrt(3)*d))*2+1;
cent=zeros(Nx,Ny,2);
for m =1:2:Ny
    for n=2:2:Nx
        cent(m,n,:)=[(3)*d/2*(n-1)+d./2,(m-1)*sqrt(3)*d/2];
    end
end
for m=2:2:Ny
   for n=1:2:Nx
       cent(m,n,:)=[(3)*d/2*(n-1)+d./2,(m-1)*sqrt(3)*d/2];
   end
end

for m=1:Ny
    for n=1:Nx
        if cent(m,n,1)~=0 || cent(m,n,2)~=0
            hexagon(cent(m,n,1),cent(m,n,2),d,'b');
            hold on
        end
    end
end
axis([0,side,0,side])
%%
xlabel('x/km','FontSize',18)
ylabel('y/km','FontSize',18)
%%
hold on

%% hexagon with the dd=5
dd=5;
for m=1:Ny
    for n=1:Nx
        if cent(m,n,1)~=0 || cent(m,n,2)~=0
            hexagon(cent(m,n,1),cent(m,n,2),dd,'g');
```

```
            hold on
        end
    end
end
axis([0,side,0,side])

%%
hold on

%% Count the number of useful cent
% usecentnum
% usecent
usecentnum=0;
for m=1:Ny
    for n=1:Nx
        if cent(m,n,1)~=0||cent(m,n,2)~=0
            usecentnum=usecentnum+1;
        end
    end
end
usecent=zeros(usecentnum,2);
x=1;
for m=1:Ny
    for n=1:Nx
        if cent(m,n,1)~=0 ||cent(m,n,2)~=0
            usecent(x,:)=cent(m,n,:);
            x=x+1;
        end
    end
end

%% Among all the dots, which is in the specific hexagon
% 1. first decide whether it's in the according rectangle (among all the do
ts)
% create a rectangle(50,2) with 1 represent in and 0 out
% 2. Then decide whether the dots in the rectangle is in the hexagon or not

% create a hexagon(50,2) to record the qulified dots
rectangle=zeros(usecentnum,50);
for cent=1:usecentnum
    for dot=1:50
        xcent=usecent(cent,1);
        ycent=usecent(cent,2);
        xdot=data(dot,1);
```

```
        ydot=data(dot,2);
        if inornotRectangle(xcent,ycent,xdot,ydot,d)
            rectangle(cent,dot)=1;
        end

    end
end

%% in or out of the hexagon
InorOut=zeros(usecentnum,50);
InorOutnum=zeros(usecentnum,1);
for m=1:usecentnum
   for n=1:50
       if rectangle(m,n)~=0
           xcent=usecent(m,1);
           ycent=usecent(m,2);
           xdot=data(n,1);
           ydot=data(n,2);
           if inornotHexagon(xcent,ycent,xdot,ydot,d)
               InorOut(m,n)=1;
               InorOutnum(m)=InorOutnum(m)+1;
           end
       end

   end
end

%% in or out of the samaller hexagon
Smaller=zeros(usecentnum,50);
Smallernum=zeros(usecentnum,1);
for m=1:usecentnum
   for n=1:50
      if InorOut(m,n)~=0
         if inornotRectangle(usecent(m,1),usecent(m,2),data(n,1),data(n,2),
dd)
            if inornotHexagon(usecent(m,1),usecent(m,2),data(n,1),data(n,2
),dd)
                Smaller(m,n)=1;
                Smallernum(m)=Smaller(m)+1;
            end
         end
      end
   end
end
```

```
156.
157. %%
158. MotherButterfly=zeros(usecentnum,2);
159. MotherButterflynum=zeros(usecentnum,1);
160. MotherRuralstar=zeros(usecentnum,2);
161. MotherRuralstarnum=zeros(usecentnum,1);
162. %% The first Alarm situation (Only one dot in the hexagon)
163. for m=1:usecentnum
164.    if InorOutnum(m)==1
165.        n=find(InorOut(m,:));
166.        if style(n)==1
167.            MotherButterfly(m,:)=data(n,:);
168.            MotherButterflynum(m)=n;
169.        else
170.            MotherRuralstar(m,:)=data(n,:);
171.            MotherRuralstarnum(m)=n;
172.        end
173.    end
174. end
175.
176. %% The second Alarm situation(<=7dots but only have Butterfly style)
177. for m=1:usecentnum
178.    if InorOutnum(m)<=7
179.        nlist=find(InorOut(m,:));
180.        ntest=nlist;
181.       for  n=1:length(nlist)
182.          if style(nlist(n))==1
183.              ntest=ntest-1;
184.          end
185.       end
186.       if ntest==0    % just the situation
187.           InorOutnum(m)=13;
188.       end
189.    end
190. end
191. %% devide upon the number of dots in one hexagon
192.
193. for m=1:usecentnum
194.     distance=10^10;
195.     while InorOutnum(m)>7
196.         for n=1:50
197.             if (InorOut(m,n)==1) && (style(n)==1)
198.                 newdis=sqrt((usecent(m,1)-data(n,1))^2+(usecent(m,2)-data(n
    ,2))^2);
```

```
                if newdis<=distance
                    MotherButterfly(m,:)=data(n,:);
                    MotherButterflynum(m)=n;
                    distance=newdis;
                end
            end
        end
% plot(MotherButterfly(1),MotherButterfly(2),'kp','MarkerSize',12,'MarkerFaceColor','k')
% hold on
% drawcircle(MotherButterfly(1),MotherButterfly(2),15);
% hold on
        InorOutnum(m)=InorOutnum(m)-13;
    end
end
%% Rural Star

for m=1:usecentnum
    if InorOutnum(m)>=1 && InorOutnum(m)<=7
        distance=10^10;
        for n=1:50
            if InorOut(m,n)==1 && style(n)==0
                newdis=sqrt((usecent(m,1)-data(n,1))^2+(usecent(m,2)-data(n,2))^2);
                if newdis<=distance
                    MotherRuralstar(m,:)=data(n,:);
                    MotherRuralstarnum(m)=n;
                    distance=newdis;
                end
            end
        end
% plot(MotherRuralstar(m,1),MotherRuralstar(2),'ko','MarkerSize',10,'MarkerFaceColor','k')
% hold on
% drawcircle(MotherRuralstar(1),MotherRuralstar(2),15);
% hold on
    end
end
%% show the Mother stations
for m=1:usecentnum
    if MotherButterfly(m,:)~=[0,0]
        plot(MotherButterfly(m,1),MotherButterfly(m,2),'kp','MarkerSize',14,'MarkerFacecolor','k')
        hold on
```

```
    end
    if MotherRuralstar(m,:)~=[0,0]
        plot(MotherRuralstar(m,1),MotherRuralstar(m,2),'ko','MarkerSize',10
    ,'MarkerFaceColor','k')
        hold on
    end

end
```

以下为使用的 function

```
function hexagon(xcent,ycent,d,colour)
    z=xcent+1i*ycent;
    plot(z+d*exp(1i*linspace(0,2*pi,7)),colour);

end
function y=inornotRectangle(x0,y0,x1,y1,d)
    if x1>=x0-d&&x1<=x0+d&&(y1>=y0-(sqrt(3)*d)/2 )&&y1<=y0+sqrt(3)*d/2
        y=true;
    else
        y=false;
    end
end


%% judging whether P[x1,y1] is in the hexagon(x0,y0,d)
% suppose P[x1,y1] is in the rectangle
% how to judge:
% if |M1P|>|MN| && |M2P|>|MN|,that means the dot is in the hexagon

function y=inornotHexagon(x0,y0,x1,y1,d)
    MN=abs(y0-y1)/sqrt(3);
    M1P=abs(x0+d-x1);
    M2P=abs(x0-d-x1);
    if M1P>MN && M2P>MN
        y=true;
    else
        y=false;
    end

end
```

附录 3 计算冗余站点算法代码

```
function y=overlapping(x1,y1,R1,x2,y2,R2)
    d=sqrt((x1-x2)^2+(y1-y2)^2);
    if d>=R1+R2
        y=0;
    elseif d<=abs(R1-R2)
        y=1;
    else
        theta1=acos((R1^2+d^2-R2^2)/(2*R1*d));
        theta2=acos((R2^2+d^2-R1^2)/(2*R2*d));
        h1=(R1^2+d^2-R2^2)/(2*d);
        h2=(R2^2+d^2-R1^2)/(2*d);
        m=R1*sqrt(1-cos(theta1)^2);
        y=(theta1*R1^2+theta2*R2^2)-m*(h1+h2);
        y=y/(pi*R1*R1);
    end
end

%% calculate the overlapping area and the effciency
% using the function overlapping(x1,y1,r1,x2,y2,r2)
% R=15 r=5
Overlapping=zeros(50,50);
for target=1:50
    x1=data(target,1);
    y1=data(target,2);
    if style(target)==1 && ismember(target,MotherButterflynum)
        R1=20;
    elseif style(target)==0 && ismember(target,MotherRuralstarnum)
        R1=20;
    else
        R1=5;
    end
   m=find(InorOut(:,target));
   n=find(InorOut(m,:));  % n contains all the dots
   l=length(n);           %in the same Hexagon of the target
   for x=1:l
       if style(n(x))==1&& ismember(target,MotherButterflynum)
           R2=20;
       elseif style(target)==0 && ismember(target,MotherRuralstarnum)
           R2=20;
       else
           R2=5;
       end
       x2=data(n(x),1);
```

```
        y2=data(n(x),2);
        Overlapping(target,n(x))=overlapping(x1,y1,R1,x2,y2,R2);
    end
end
```

附录 4 计算 Lof 离群因子代码[12]

```
import math
import matplotlib.pyplot as plt

class LOF:
    #初始化
    def __init__(self,data,k,threshold):
        self.data=data
        self.k=k
        self.threshold=threshold
        self.outliners=[]

    #计算欧式距离
    def __calDistance(self,a,b):
        sum1=0
        for k in range(len(a)):
            sum1+=((a[k]-b[k])**2)
        sum1=math.sqrt(sum1)
        if sum1==0:
            sum1=0.00000001
        return sum1

    #计算对象 point 的第 k 距离领域
    def __calNk(self,point):
        Nk=[]
        k=self.k
        distList=[]
        for j in range(len(self.data)):
            dis=self.__calDistance(point,self.data[j])
            #print(disList)
            distList.append([dis,data[j]])
        distList=sorted(distList)
        distance=distList[k-1][0]
        distList=distList[0:k-1]
        disList2=[]
        for di in distList[k-1:]:
            if di[0]==distList[k-1][0]:
                disList2.append(di)
        Nk=distList+disList2
```

```
39.         Nk=[nk[1] for nk in Nk]
40.         return Nk,distance
41.
42.     #计算可达距离
43.     def __getReachDis(self,point,Nk):
44.         reachDis=[]
45.         for nk in Nk:
46.             Nk1,dis1=self.__calNk(nk)
47.             dis2=self.__calDistance(point,nk)
48.             reachDis.append(max(dis1,dis2))
49.         return reachDis
50.
51.     #计算局部可达密度
52.     def __getLrd(self,point):
53.         Nk,distance=self.__calNk(point)
54.         reachDis=self.__getReachDis(point,Nk)
55.         lrdPoint=1.0*sum(reachDis)/len(reachDis)
56.         return lrdPoint,Nk
57.
58.     def __getLrdList(self,num):
59.         lrdList=[]
60.         lrdPoint,Nk=self.__getLrd(self.data[num])
61.         for l in range(len(Nk)):
62.             lr,nk=self.__getLrd(Nk[l])
63.             lrdList.append(lr)
64.         return lrdPoint,lrdList
65.
66.     def __getLOF(self,num):
67.         lrdPoint,lrdList=self.__getLrdList(num)
68.         lofValue=0
69.         for lrd in lrdList:
70.             lofValue+=1.0*(lrd/lrdPoint)
71.         lofValue=lofValue/len(lrdList)
72.         return self.data[num],lofValue
73.
74.     def run(self):
75.         lis=[]
76.         for i in range(len(data)):
77.             lofP,lofV=self.__getLOF(i)
78.             lis.append([lofP,lofV])
79.             if lofV>self.threshold:
80.                 self.outliners.append(lofP)
81.         #self.outliners=list[(self.outliners)]
82.         return self.outliners,lis
```

```
83.
84. if __name__=="__main__":
85.     f=open('0_10.txt')
86.     d=f.readlines()
87.     data=[]
88.     for item in d:
89.         item=str(item).strip().split(' ')
90.         lis=[float(j) for j in item]
91.         data.append(lis)
92.     points=LOF(data,5,1)#k=5 阈值=1
93.     outlines,lis=points.run()
94.
95.     plt.show()
96.     print(outlines)
97.     plt.close("all")
98.     for point in data:
99.         plt.plot(point[0],point[1],'.',color='black')
100.     for point in outlines:
101.         plt.plot(point[0],point[1],'.',color='red')
102.     plt.show()
```

附录 5 ISODATA 算法代码[13]

```
1. from math import sqrt,radians
2. import math
3. import matplotlib as mpl
4. import matplotlib.pyplot as plt
5. import time
6. import numpy
7. import os
8.
9. start_time=time.time()
10. start_time_s=time.localtime(start_time);
11. print('开始时间: '+time.asctime(start_time_s))
12. BASEDIR = os.path.dirname(__file__)
13. time_struct=time.localtime()
14. time_str=[]
15. for i in range(6):
16.     time_str.append(str(time_struct[i]))
17. time_name='_'.join(time_str)
18. BASEDIR=os.path.join(BASEDIR,time_name)
19. FIG=os.path.join(BASEDIR,'FIG')
20. os.makedirs(FIG)
21. DATA=os.path.join(BASEDIR,'DATA')
22. os.makedirs(DATA)
```

```
# cluster points
points=[]
#f1=open('x.txt','r')
f1=open('经纬 x.txt','r')
x=f1.readlines()
# f2=open('latitude.txt','r')
f2=open('经纬 y.txt','r')
y=f2.readlines()
points=[[float(x[i].strip()),float(y[i].strip())] for i in range(len(x))]
f1.close()
f2.close()
# 计算两点距离
# x 为经度，y 为纬度
def distance_2point(x1, y1, x2, y2):
    return sqrt((x2-x1)**2 + (y2-y1)**2)
    #[x1,x2,y1,y2]=map(radians,[x1,x2,y1,y2])
    #return 6378*math.acos(math.cos(y2)*math.cos(y1)*math.cos(x1-x2)+math.si
n(y1)*math.sin(y2))


# 计算平均距离
def volume_estimation(cluster, center):
    num_of_points = len(cluster)
    distance = []
    for i in range(num_of_points):
        distance.append(distance_2point(center[0], center[1], cluster[i][0],
 cluster[i][1]))

    return sum(distance)/num_of_points


# 定义新的簇中心
def new_cluster_centers(cluster):
    x=0
    y=0
    for i in cluster:
        x+=i[0]
        y+=i[1]
    length = len(cluster)
    return (x/length, y/length)


# 各中心的距离
def center_distance(centers):
```

```
    D_ij = {}
    # offset coeficient
    k = 0
    for i in range(len(centers)):
        for j in range(k, len(centers)):
            if i == j:
                pass
            else:
                D_ij[(i,j)] = distance_2point(centers[i][0], centers[i][1], centers[j][0], centers[j][1])
        k +=1
    return D_ij


# MSE
def standart_deviation(values, center):
    n = len(values)
    x_coord = []
    y_coord = []
    for i in range(n):
        x_coord.append((values[i][0]-center[0])**2)
        y_coord.append((values[i][1]-center[1])**2)

    x = sqrt(sum(x_coord)/n)
    y = sqrt(sum(y_coord)/n)

    return (x,y)

#重新归类
def cluster_points_distribution(centers, points):
    centers_len = len(centers)
    points_len = len(points)
    distances = []
    distance = []

    # define array for clusters
     clusters = [[] for i in range(centers_len)]

     # iteration throught all points
     for i in range(points_len):
         # iteration throught all centers
         for j in range(centers_len):
             distance.append(distance_2point(centers[j][0], centers[j][1], points[i][0], points[i][1]))
```

```
107.         distances.append(distance)
108.         distance = []
109.
110.     # distribution
111.     for i in range(points_len):
112.         ind = distances[i].index(min(distances[i]))
113.         clusters[ind].append(points[i])
114.
115.     return clusters
116.
117.
118. def cluster_division(cluster, center, dev_vector):
119.     #divide only center of clusters
120.
121.     # coeficient
122.     k = 0.5
123.
124.     max_deviation = max(dev_vector)
125.     index = dev_vector.index(max(dev_vector))
126.     g = k*max_deviation
127.
128.     # defining new centers
129.     center1 = list(center)
130.     center2 = list(center)
131.     center1[index] += g
132.     center2[index] -= g
133.
134.     cluster1 = []
135.     cluster2 = []
136.
137.     return tuple(center1), tuple(center2)
138.
139.
140. def cluster_union(cluster1, cluster2, center1, center2):
141.     x1 = center1[0]
142.     x2 = center2[0]
143.     y1 = center1[1]
144.     y2 = center2[1]
145.     n1 = len(cluster1)
146.     n2 = len(cluster2)
147.
148.     x = (n1*x1 + n2*x2)/(n1+n2)
149.     y = (n1*y1 + n2*y2)/(n1+n2)
150.     center = (x,y)
```

```
151.     cluster = cluster1 + cluster2
152. 
153.     return center, cluster
154. 
155. def test(clusters,centers):
156.     for i in clusters:
157.         if len(i)<1:
158.             num=clusters.index(i)
159.             clusters.remove(i)
160.             centers.remove(centers[num])
161.     return
162. 
163. def clusterize():
164. 
165.     # initial values
166.     K = 20 # max cluster number
167.     THETA_N = 7 # for cluster elimination
168.     THETA_S = 8 # for cluster division
169.     THETA_C = 3 # for cluster union
170.     L = 3 #
171.     I = 30 # max number of iterations
172.     N_c = 10 # number of primary cluster centers
173. 
174.     distance = [] # distances array
175.     centers = [] # clusters centers
176.     clusters = [] # array for clusters points
177.     iteration = 1 # number of current iteration
178. 
179.     centers.append(points[0]) # first cluster center
180. 
181.     while iteration <= I:
182. 
183.         if(iteration==1):
184.             time_old=start_time
185.         else:
186.             time_old=time_now
187.         #重新分簇
188.         if len(centers) <= 1:
189.             clusters.append(points)
190.         else:
191.             clusters = cluster_points_distribution(centers, points)
192. 
193.         flag=True
194.         # 去掉样本数过小的簇（与距离最近的簇融合）
```

```
        for i in range(len(clusters)):
            if len(clusters[i]) <= THETA_N:
                flag=False
                print('样本过小！\n')
                D_ij=center_distance(centers)
                dis=[]
                for j in range(len(clusters)):
                    dis.append(D_ij[(i,j)])
                key=dis.index(min(D_ij))
                clusters[key].append(item for item in clusters[i])
                del clusters[i]
                break
            else:
                pass
            break
            if not flag:
                flag=True
                if len(centers) <= 1:
                    clusters.append(points)
                else:
                    clusters = cluster_points_distribution(centers, points)

        test(clusters,centers)
        # 寻找新的聚类中心
        centers = []
        for i in range(len(clusters)):
            centers.append(new_cluster_centers(clusters[i]))

        # 计算每个类的类内平均距离
        D_vol = []
        for i in range(len(centers)):
            D_vol.append(volume_estimation(clusters[i], centers[i]))

        # 计算所有类的总体平均距离
        if len(clusters) <= 1:
            D = 0
        else:
            cluster_length = []
            vol_sum = []
            for i in range(len(centers)):
                cluster_length.append(len(clusters[i]))
                vol_sum.append(cluster_length[i]*D_vol[i])

```

```
            D = sum(vol_sum)/len(points)


        # 判断停止、分裂或合并
        if iteration >= I:
            THETA_C = 0#算法结束

        # elif (N_c >= 2*K) or (iteration % 2 == 0):
            # pass#跳过分裂处理

        elif (K/2<K<2*K) and (iteration%2==1):
            #进行分裂处理
            vectors = []
            for i in range(len(centers)):
                vectors.append(standart_deviation(clusters[i], centers[i]))


            max_s = []
            for v in vectors:
                max_s.append(max(v[0], v[1]))

            for i in range(len(max_s)):
                length = len(clusters[i])
                coef = 2*(THETA_N+1)

                if (max_s[i] > THETA_S) and ((D_vol[i]>D and length>coef) or N_c<float(K)/2):
                    center1, center2 = cluster_division(clusters[i], centers[i], vectors[i])
                    del centers[i]
                    centers.append(center1)
                    centers.append(center2)
                    N_c += 1

                else:
                    pass

        elif (K/2<K<2*K) and (iteration%2==0):
            #进行合并处理
            D_ij = center_distance(centers)
            rang = {}
            for coord in D_ij:
                if D_ij[coord] < THETA_C:
                    rang[coord] = (D_ij[coord])
```

```
                else:
                    pass

            for key in rang.keys():
                cluster_union(clusters[key], clusters[key.next()], centers[key], centers[key.next()])
                N_c -= 1

        plotfig(clusters,centers,iteration)
        time_now=time.time()
        print('迭代'+str(iteration)+'用时：'+str(time_now-time_old)+'s')
        iteration += 1

    return clusters,centers,max_s

def plotfig(clusters,centers,I):
    color=['peru','dodgerblue','brown','gold','dimgrey','midnightblue','lavender','navy','violet','thistle','chartreuse','olive',
           'orange','springgreen','burlywood','fuchsia','darkslategrey','orangered','tomato','red','khaki','darkgoldenrod','deeppink','skyblue',
           'darkgray','darkblue','deepskyblue','wheat',
           'peru','dodgerblue','brown','gold','dimgrey','midnightblue','lavender','navy','violet','thistle','chartreuse','olive',
           'orange','springgreen','burlywood','fuchsia','darkslategrey','orangered','tomato','red','khaki','darkgoldenrod','deeppink','skyblue',
           'darkgray','darkblue','deepskyblue','wheat']
    key=0
    for i in clusters:
        for j in i:
            plt.plot(j[0],j[1],'.',color=color[key%29])
        key+=1
    key=0
    for i in centers:
        plt.plot(i[0],i[1],'*',color='black')
        plt.annotate(s=(str(key+1)),xy=i)
        key+=1
    name='iteration_'+str(I)+'.png'
    path=os.path.join(FIG,name)
    plt.savefig(path)
    plt.clf()
    plt.close('all')

if __name__ == '__main__':
    cl,cen,max_s= clusterize()
```

```
    # color=['r','b','g','gold','dimgrey','midnightblue','lavender','navy','violet','thistle','chartreuse','olive','r','b','g','gold','dimgrey','midnightblue','lavender','navy','violet','thistle','chartreuse','olive','r','b','g','gold','dimgrey','midnightblue','lavender','navy','violet','thistle','chartreuse','olive']
    # key=0
    # print(len(cl))
    # print(len(cen))
    # print(len(max_s))
    # for i in cl:
        # for j in i:
            # plt.plot(j[0],j[1],'.',color=color[key])
        #plt.show()
        # key+=1
    # key=0
    # for i in cen:
        # plt.plot(i[0],i[1],'*',color='black')
        # plt.annotate(s=(str(key)),xy=i)
        # key+=1
        # print(key)
    # plt.show()
    #print(max_s)
    test(cl,cen)
    for i in range(len(cl)):
        name=str(i)+'_'+str(len(cl[i]))+'.txt'
        filepath=os.path.join(DATA,name)
        f=open(filepath,'w')
        for j in cl[i]:
            f.write(str(j[0])+' '+str(j[1])+'\n')
        f.close()
    end_time=time.time()
    end_time_s=time.localtime(end_time)
    print('结束时间: '+time.asctime(end_time_s))
    time=end_time-start_time;
    print('进程用时: '+str(time)+'s')

```

附录 6 Dijkstra 算法代码

```
#Dijkstra

import math

def cal_distance(a,b):
    return math.sqrt((a[0]-b[0])**2+(a[1]-b[1])**2)
```

```
7.
8.  def remo(list,obj):
9.      try:
10.         list.remove(obj)
11.     except:
12.         pass
13.
14. def judge(OnOffSet,D_ij,num,style):
15.     flag=True
16.     #找起点
17.     start=[]
18.     full=[]
19.     for i in range(len(OnOffSet)):
20.         if(i<num)and(sum(OnOffSet[i])<4*style[i]):
21.             start.append(i)
22.         elif(i<num)and(sum(OnOffSet[i])>=4*style[i]):
23.             full.append(i)
24.         elif(i>=num)and(sum(OnOffSet[i])==1):
25.             start.append(i)
26.     for i in range(len(OnOffSet)):
27.         n=sum(OnOffSet[i])
28.         if(n==0):
29.             flag=False
30.     #print(start)
31.
32.     for i in range(num):
33.         xuhao=[]
34.         num_p=sum(OnOffSet[i])
35.         for j in range(len(OnOffSet[i])):
36.             if OnOffSet[i][j]==1:
37.                 tiao=1
38.                 num_p+=1
39.                 xuhao.append(j)
40.                 if(sum(OnOffSet[j])==2):
41.                     tiao=2
42.                     num_p+=1
43.                     for k in range(len(OnOffSet[j])):
44.                         if (OnOffSet[j][k]==1)and(k!=i):
45.                             son2=k
46.                     xuhao.append(son2)
47.                     if(sum(OnOffSet[son2])==1):
48.                         num_p+=1
49.                         remo(start,j)
50.                         remo(start,son2)
```

```
                        remo(start,OnOffSet[son2].index(1))
                        full.append(j)
                        full.append(son2)
                        full.append(OnOffSet[son2].index(1))
        #print(style,num)
        #print(num_p)
        if(num_p/2>6*style[i]):
            full.append(i)
            remo(start,i)
            for k in xuhao:
                remo(start,k)
                full.append(k)
    #print(start)
    #print(full)
    #print(D_ij)
    for ii in range(len(D_ij)):
        for jj in range(len(D_ij)):
            #print(ii,jj)
            if (ii in start) and (not(jj in start))and(not(jj in full)):
                #print('T')
                pass
            else:
                D_ij[ii][jj]=1e10
    return D_ij,flag

def find_min(list):
    len_a=len(list)
    len_b=len(list)
    mini=[]
    label=[]
    for i in range(len_a):
        mini.append(min(list[i]))
        label.append(list[i].index(min(list[i])))
    a=mini.index(min(mini))
    b=label[a]
    return a,b

def print_list(list):
    len_a=len(list)
    len_b=len(list)
    f=open('OnOffSet.txt','w')
    for i in range(len_a):
        for j in range(len_b):
            print(str(list[i][j])+' ')
```

```
            f.write(str(list[i][j])+' ')
        print('\n')
        f.write('\n')
    f.close()
    return

def Dijkstra(centers,points,style):
    #style 表示基站的类型，1 为 ruralstar，2 为蝴蝶型
    points=centers+points
    #print(points)
    num=len(centers)
    #print(num)
    num_points=len(points)-num
    suppot=sum([6*style[i] for i in range(len(style))])
    if(num_points>=suppot):
        print("ERROR3!")
        return
    D_ij=[]
    OnOffSet=[]
    for i in range(len(points)):
        D_ij.append([])
        OnOffSet.append([])
        for j in range(len(points)):
            D_ij[i].append(1e10)
            OnOffSet[i].append(0)
    # inn=[1e10 for i in range(len(points))]
    # D_ij=[inn for i in range(len(points))]
    # innn=[0 for i in range(len(points))]
    # OnOffSet=[innn for i in range(len(points))]#0 断 1 通
    for k in range(len(points)):
        for n in range(len(points)):
            if(k!=n):
                dis=cal_distance(points[k],points[n])
                D_ij[k][n]=math.log(dis,10)
            elif(k==n):
                D_ij[k][n]=1e10
            OnOffSet[k][n]=0
    while(True):
        D_ij_1=[]
        for k in range(len(points)):
            for n in range(len(points)):
                if(k!=n):
                    dis=cal_distance(points[k],points[n])
                    D_ij[k][n]=math.log(dis,10)
```

```
                elif(k==n):
                    D_ij[k][n]=1e10
        D_ij_1,flag=judge(OnOffSet,D_ij,num,style)
        if flag:
            break;
        #print(D_ij_1)
        a,b=find_min(D_ij_1)
        #print(D_ij[a][b])
        if(D_ij_1[a][b]>10)and(a,b>num-1):
            print("ERROR1!")
            print(D_ij_1[a][b])
            print(OnOffSet)
            return
        elif(D_ij_1[a][b]>20):
            print("ERROR2!")
            return
        OnOffSet[a][b]=1
        OnOffSet[b][a]=1
        #print(OnOffSet)
    for i in range(len(points)):
        for j in range(len(points)):
            if(i<num)and(j<num)and(i!=j)and(D_ij[i][j]<=50):
                OnOffSet[i][j]=2
    print_list(OnOffSet)
    return OnOffSet

if __name__=="__main__":
    f=open('0_10.txt')
    d=f.readlines()
    data=[]
    for item in d:
        item=str(item).strip().split(' ')
        lis=[float(j) for j in item]
        data.append(lis)
    centers=[data[0]]
    #print(centers)
    points=data[1:]
    #print(points)
    style=[2]
    Dijkstra(centers,points,style)
```