队伍编号	MC2203225
题号	D

基于贪心算法的站址选择问题

摘 要

随着 5G 的发展，基站和天线的种类变多，站址选择的问题变得越来越复杂。我们通过分析已给的站址与弱覆盖点的信息，基于贪心算法，Dbscan 聚类算法建立了数学模型，为解决现网的弱覆盖区域的覆盖问题提供了模型，给出了最优站址的选择与对弱覆盖点的区域聚类方案。

针对问题一，我们建立了最优站址选择模型。我们采用贪心算法，选取未被覆盖的业务量最大的点作为基站选址的参考，通过比较周围点作为站址所覆盖的总业务量大小，选取覆盖业务量最大的点，再重复考虑周围的点，最后得出局部的覆盖业务量最大值。再进行多次上述步骤，得出最优站址选择模型并得出新建基站数量为 2464。

针对问题二，我们建立了花瓣旋转模型。首先将基站覆盖图形考虑成花瓣模型，在贪心算法进行地同时，通过穷尽的算法，将该模型旋转 120 次找出每个基站覆盖业务量最大的角度，最后得出局部覆盖业务量最大值，在保证站址数与模型一得出的数目一致的情况下，求出此时覆盖的业务量占总业务量比例为 85.879%。

针对问题三，我们先比较 K-means 算法与 Dbscan 算法，最后选用 Dbscan 聚类算法，考虑 ϵ 为 20，minPts 为 1 的情况，最后包括孤立点将弱覆盖区域聚类成了 898 类。

本文所建立的模型简便易行，易于推广，经过分析验证，具有合理性和一定的现实意义。

关键词：基站选址；贪心算法；花瓣模型；Dbscan 聚类算法

目录

一、问题重述

1.1 问题背景

随着通讯技术的发展以及 5G 技术的逐步成熟，通讯带宽越来越大，但基站本身所能覆盖的范围也在逐步缩小，其相应的网络环境复杂化程度在不断加深。因此，在过去原有基站的基础之上，我们若想使新技术覆盖相同区域，便需要新建更多的基站。此外，随着网络移动设备的普及，人们对于网络环境稳定性的要求也在逐步上升，那么如何在满足当今这样复杂需求的基础之上，选择好合适的基站建设设备、规划好合适的基站建设位置便尤为关键。

1.2 问题提出

基于以上背景，我们提出了如下站址选择问题：首先根据现网天线的覆盖情况，给出现网的弱覆盖区域，并选择一定数量的点，使得在这些点上新建基站后，可以解决现网的弱覆盖区域的覆盖问题。考虑基站的建设成本等因素，有时候可能无法把所有弱覆盖区域都解决，因此我们需要考虑业务量的因素，并尽量优先解决业务量高的弱覆盖区域。

首先，题目给定了一块区域，我们需要在该区域中解决相应的问题。给定区域的大小是 2500×2500 个栅格，即 2500×2500 个点，其中横坐标范围是 0 到 2499，纵坐标范围是 0 到 2499。此外，我们同时获得了该区域中包括了每个点的坐标和相应业务量在内的所有弱覆盖点信息，以及原有基站的位置坐标。我们解决问题时可以选择的基站共有两种，分别为宏基站（覆盖范围 30，成本 10）和微基站（覆盖范围 10，成本 1），在选址的过程中应保证新建站址之以及新建站址和现有站址之间的距离的门限是 10。

在此基础之上，我们根据不同的实际需要，分步解决以下三个问题：

（1）根据给定弱覆盖点信息以及原有基站的位置信息进行站址规划，在保证以上条件的前提下，使弱覆盖点总业务量的 90% 被新建基站所覆盖。给出我们所选择的站址坐标以及每个站址所选择的基站种类。新建站址的坐标只能在给定区域内的 2500×2500 个点中进行选择。

（2）在现实状况中，每个基站的覆盖范围并不是完全的圆形，而是由 3 个扇区所构成的，其中每个扇区都指向一个方向。每个扇区在主方向上的覆盖范围最大（宏基站为 30，微基站为 10），在主方向左右 60° 的范围内可以进行覆盖，覆盖范围随着角度按线性逐渐缩小，在 60° 的时候，覆盖范围为主方向覆盖范围的一半。超过 60°，则无法被该扇区覆盖。在覆盖的过程中，我们需要保证每个基站的任意 2 个扇区的主方向之间的夹角不能小于 45°。

根据第一问中计算出的站址数量，我们为这些新建站重新确定位置以及相应扇区，判断其能否覆盖总业务量的 90%，并给出最优站址的位置以及各个主方向的角度。

（3）将距离小于等于 20 的弱覆盖点聚类，考虑聚类性质的传递性，并构建一个复杂度较低、运行速度较快的聚类算法，对所有的弱覆盖点进行聚类。

二、问题分析

2.1 问题一的分析

问题一是通过基站选取和站址规划使得弱覆盖点总业务量的 90% 被基站覆盖。同时，我们在进行覆盖的时候，除了要考虑能覆盖弱覆盖点的个数问题，还要考虑该基站所能覆盖的总业务量，因此我们在进行规划的时候，除了要考虑点的密集程度，更要考虑这个点所在位置的业务量大小。此外，我们可以选择的基站一共有两种，分别是宏基站和微基站，故我们仍需要判定在不同情况下使用不同基站所带来的经济效应。

因此，我们首先需要考虑模型的已有约束条件，并根据附件中提供的原有信息点，对原有情况进行可视化判断，从而确立我们最终使用的方法。由于问题的距离判定是基于欧式距离来判断的，在满足最大业务量的需求下，我们决定采用了贪心算法来寻找局部最优解，以尽可能覆盖住业务量大的坐标点。

首先，我们选取未被覆盖的业务量最大的点作为基站选址的参考，先将该坐标点作为圆心，计算该点新建宏基站和微基站覆盖的业务总量的比值，通过经济效应的分析来判断基站的构建类型。确定好基站类型之后，再求该点附近的局部最优解，和该点四周的八个点分别构建基站得到的总业务覆盖量比较，找出总业务覆盖量最大的情形，将基站移到该坐标点，持续往复直到基站不再移动，此时的坐标点即为局部最优的新建基站点。

对该算法进行循环，我们便可以依次在剩余弱覆盖点中找到业务量最大的点，并重复上述操作，直至弱覆盖点的 90% 的业务量被覆盖。

2.2 问题二的分析

问题二是在问题一的基础上，将新基站从以圆形覆盖，修改为使用三个 120° 的扇形“叶片”进行覆盖，因此我们需要通过调整角度的方式，使得总业务量达到最大。又由于当三片叶片不重叠时，其总覆盖面积是最大的，因此我们考虑对该模型进行简化，将三个扇形覆盖转变为不重叠的“花瓣”模型。

延续问题一判断最优站址的方法，我们考虑对每一个新建基站的局部最优解进行重新判断。即基于问题一的贪心算法，我们在其每一次的判断之前增添了旋转判别的条件，以 1° 为标准划分间断点，基于花瓣的对称性质，在 120° 的范围内旋转 120 次，再延续使用我们第一问的贪心算法。

由于第一问在计算最优站址时，确定了我们所要建造的最优站址数量。因此我们在第二问中计算最大业务量时，我们固定新建站址的总数量，计算满足“花瓣”覆盖的最大业务量，并给出重新确立的最优坐标以及最优主方向角度。

2.3 问题三的分析

根据前两问的模型，我们已经可以有效地解决弱覆盖点的基站规划问题了，但由于系统的计算数量较为庞大，将其应用在实际的生活当中显然是不现实的，因此我们需要选择一个合适的方式来优化我们前两问中提出的模型，对所有弱覆盖点进行区域聚类。

在现有的聚类方法中，我们考虑了 K-means 以及 Dbscan 聚类法。由于我们所给定的聚类方法要求中，并没有规定集群的数量，并且我们所要进行聚类的数据当中存在较多业务量过大的孤立点。因此，我们在满足聚类算法的最简化条件基础之上，选择了可以实现非线性关系聚类的 Dbscan 聚类法。

三、模型的假设

1. 假设宏基站与微基站只在建造时考虑成本，不考虑后续维修费用的成本的开支。

2. 假设各基站信号覆盖不存在衰减现象，覆盖区内接受信号强度一致。信号发射出去对于接收者来说只有两种情况，第一是接受到了信号，第二是没有接收到信息。只要接受者接收到了信号，他的信号强度就是一致的。故该弱覆盖地区被多处信号覆盖，也视为和原先一样。

3. 不考虑特殊地区的影响，认为所有地区的基础条件都一样。某些特殊地区，如军事基地、政府行政场所，因为政策原因无法大范围建立基站，或地形极端恶劣地区无法建立基站。但是由于军事政治地区占比很小，并且大部分地区于郊外，而后者存在于城市之中的概率也十分小，所以不考虑特殊地区的影响。

4. 在基站调节方向角度的时候，规定每次只能进行以整数为单位的调整。由于庞大机械的灵敏度一般较低，一般不能进行过于精确化的操作，故可默认角度调整最小精确值为 1°。

四、符号说明

符号	符号说明
(x_n, y_n)	模型一新建基站的坐标点位
$(x_n + i, y_n + j)$	(x_n, y_n) 周围的八个坐标点
(x'_n, y'_n)	模型二新建基站的坐标点位
W	弱覆盖区域的业务量总和
$W(x_i, y_i)$	该点新建坐标的业务量最大值
$W_{n.i}$	模型二中第 n 个考虑的新建基站转动 i-1 次时的覆盖业务量
M	考虑该点建宏基站所覆盖的业务量
m	考虑该点建微基站所覆盖的业务量
FL	自由空间损耗

五、模型的建立与求解

5.1 模型的准备

5.1.1 初始数据分析与可视化

在解决问题之初，我们分别提取了 2 个附件中的表格信息，将其从原有的数据转化为了图像，以帮助我们更直观地分析出我们所要解决问题，为我们的问题解决以及模型建立提供分析导向。

我们根据原有基站的距离门限 10，绘制了如下原有基站在题设区域中的可视化结果图，我们可以发现原有基站在地图上的位置呈类随机分布，且范围遍布整个地区。

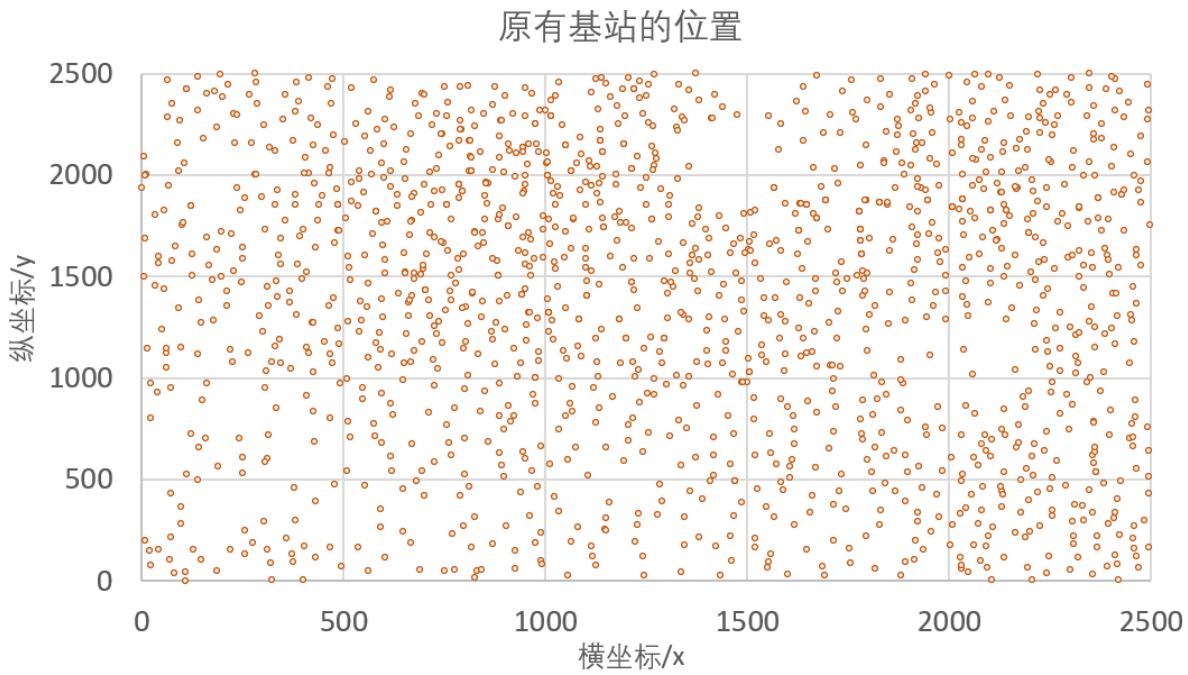

图 1 原有基站的分布图

其次，我们分析了弱覆盖区域中的坐标点及业务量。一共有 182807 个点，总业务量之和为 7056230。在所有的弱覆盖点业务量中，最小的值是 0.000192，最大的值是 47795.01。然而，其中业务量小于 1 的点个数为 56925，其相应的业务量之和为 18641.55；但业务量大于 1000 的点，却以 666 个的数量，覆盖了 2188114 的业务量。

由此我们可以分析得出，业务量小于 1 的点虽然占了 30% 以上，但是其总业务量仅有 0.26%，而业务量大于 1000 的点在个数上只占 0.36%，但总业务量却占到了惊人的 30% 以上。

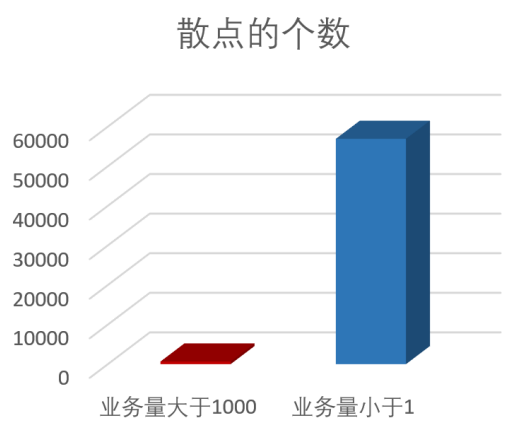

图 2 弱覆盖点的个数

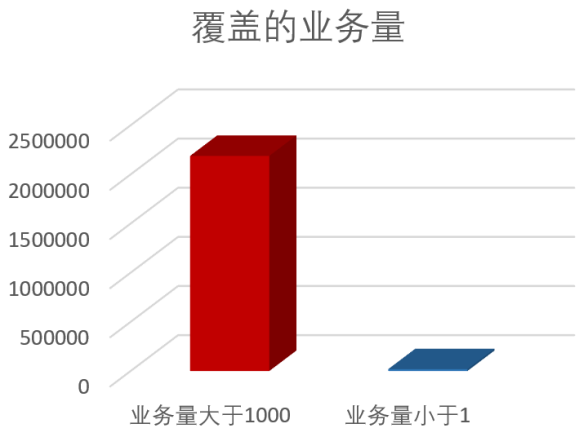

图 3 所覆盖的业务量

同时，为了满足站址规划问题，使得弱覆盖点总业务量的 90% 被规划基站覆盖，我们应该优先使业务量大的坐标点被覆盖。所以未被覆盖的业务量最大的点，便是我们建造基站时所要聚焦的核心问题。

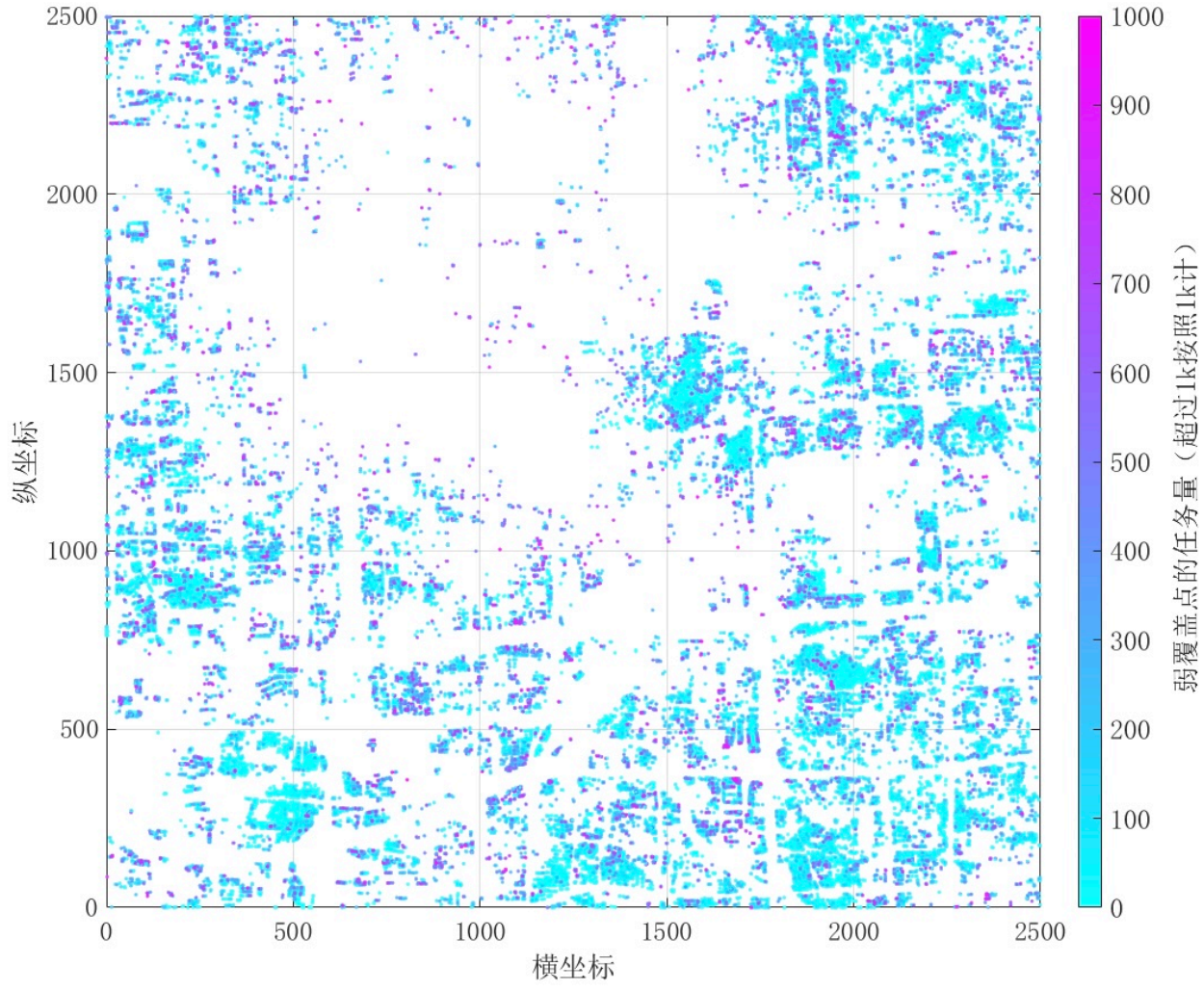

图 4 弱覆盖点的业务量热力分布图

5.1.2 模型所用术语的定义

目标要求：我们先算出弱覆盖区域总的业务量 W，以总业务量的 90% 作为目标，即当总业务量达到 0.9W 时目标完成。

选择基站类型：由于宏基站的成本是 10，微基站的成本是 1，所以选择宏基站的情况是当该地址建造宏基站后业务的覆盖量是建造微基站后业务覆盖量的十倍。若未达到则选择构建微基站。

选择构建基站地点的原理：该问题的目标是进行站址规划，使得弱覆盖点总业务量的 90% 被规划基站覆盖，所以我们每次都以弱覆盖区中未被覆盖的业务量最大的点作为基站选址参考。但是考虑到以当前业务量最大的点作为基站选址未必是包含该点的覆盖业务量的最大值，所以我们考虑范围内局部搜索最优解。

5.2 模型一的建立与求解

（1）确定可以建立基站的点

首先, 我们需要确定新建基站可以进行建立的坐标点，即判断弱覆盖区域中业务量最大的点是否与已有基站冲突。若不冲突则表示该点坐标为（x_1,y_1），若冲突则在半径范围内寻找不与已有基站冲突的点。若这样的点不存在，我们则跳过该点，若存在则选择出距离最近的点，并将该点坐标赋为（x_1,y_1）。

在选出考虑建立基站的坐标点（x_1,y_1）之后，我们判断新建基站的类型。由于宏基站的成本是 10，微基站的成本是 1，在只考虑基站成本与覆盖业务量的效果的情况下，认为当建造宏基站后的覆盖业务量大于建造微基站后的覆盖业务量的十倍，就考虑建造宏基站，否则建造微基站。

根据以上的步骤，我们考虑使用贪心算法来完成问题一模型中接下来的基站选择问题。我们最终目的是使得新建基站使得总的覆盖弱覆盖区的总业务量达到最大，故根据贪心算法，我们可以将该目的分为若干个子问题，即每一个新建基站的覆盖的业务量达到我们所考虑的坐标点区域附近的最大值。

（2）使用贪心算法探索附近区域建造的可行性

下面我们开始对（x_1,y_1）附近的区域进行考虑。我们如下图表示出（x_1,y_1）以及用（x_1+i,y_1+i）（$i\cdot j\in\{-1,0,1\}$,i,j 不同时等于 0）表示的附近八个点。

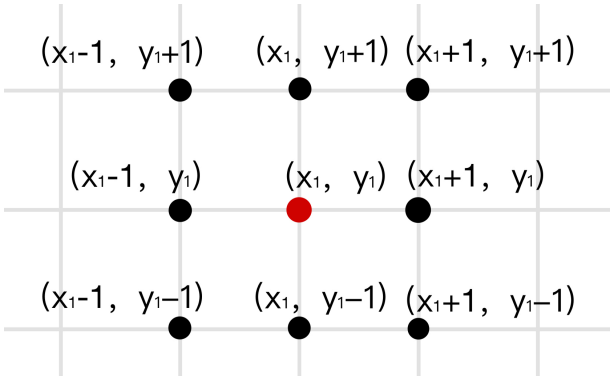

图 5 点的坐标表示图

然后开始考虑以这九个点各为基站建造点，分别求出九个点作为基站点时覆盖的业务量 W（x_1,y_1）和 W（x_1+i,y_1+i），并找出其中的最大值。在计算覆盖业务量时，我们采用欧氏距离进行计算。以（x_1,y_1）为例，比较点（x_2,y_2）与（x_1,y_1）的距离 d，若 d<r，则该点在被该基站覆盖的范围内。

$$d=\sqrt{(x_1-x_2)^2+(y_1\cdot y_2)^2} \tag{1}$$

此后，便计算 W（x_1,y_1），若其为最大值，则（x_1,y_1）是局部最优解；若最大值为 W（x_1+i,y_1+i）则该点为当前的局部最优解。那么我们只需将当前所计算出的局部最优解设定为新建基站的点，并将该点的坐标定为（x_1,y_1）。重复上述算法，直到 W（x_1,y_1）为最大值。则新建基站于（x_1,y_1）是（x_1,y_1）附近的局部最优解。同时，将（x_1,y_1）上基站所覆盖范围内的点的业务量记为 0，以防止之后新基站建立重复计算该范围的业务量。

在（x_1,y_1）建立基站之后，继续寻找剩余未被基站覆盖弱覆盖区域中找出业务量最大的点，并通过以上步骤确定出考虑建立基站的坐标点（x_2,y_2）。重复 n-1 次，建立完 n-1 个基站，当建立第 n 个基站于（x_2,y_2）后，满足所覆盖点业务量之和大于总业务量的 90% 之后，算法停止

$$\sum W(x_i,y_i)\ (i=1.2,\cdots,n)\geqslant 0.9W \tag{2}$$

以下为本算法的流程图：

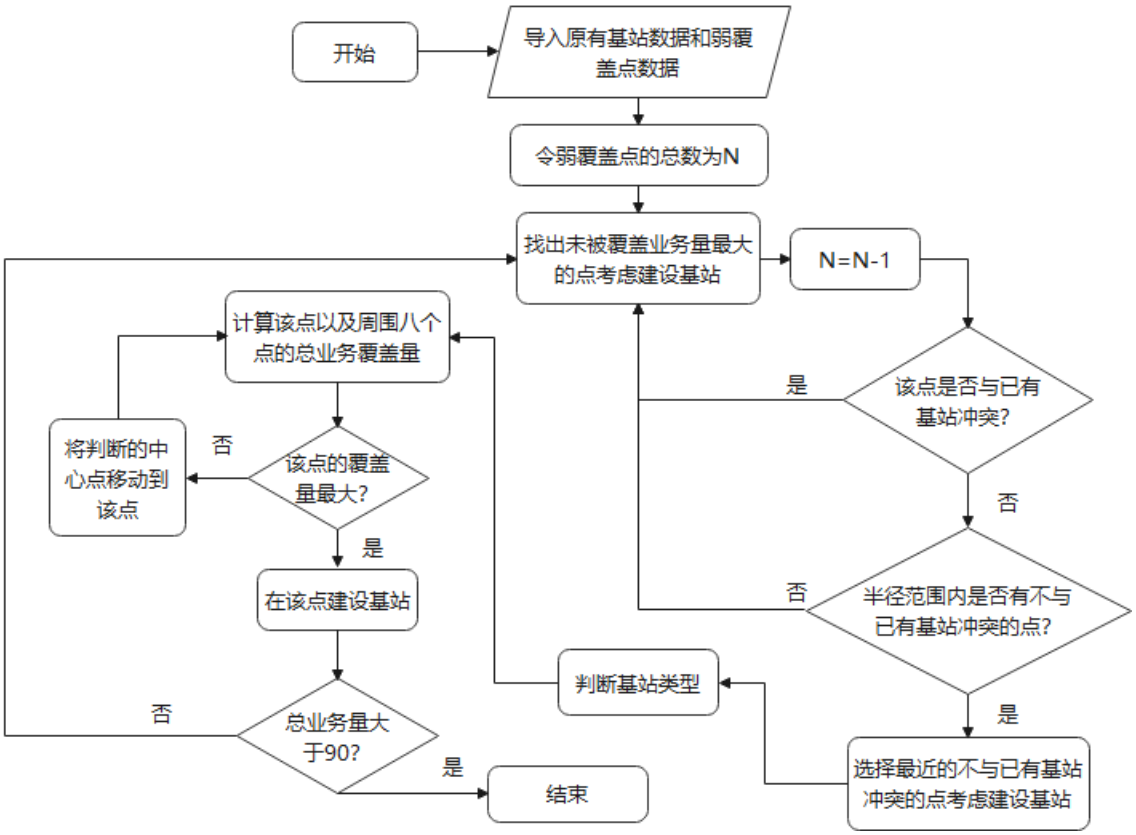

图 6 贪心算法流程图

经过贪心算法的运行，我们最终输出了将要建设基站的点的坐标，并发现我们所选择的基站均为 d=10 的基站。我们将新建的 2464 个基站以及原有基站绘制成了如下的基站分布图。

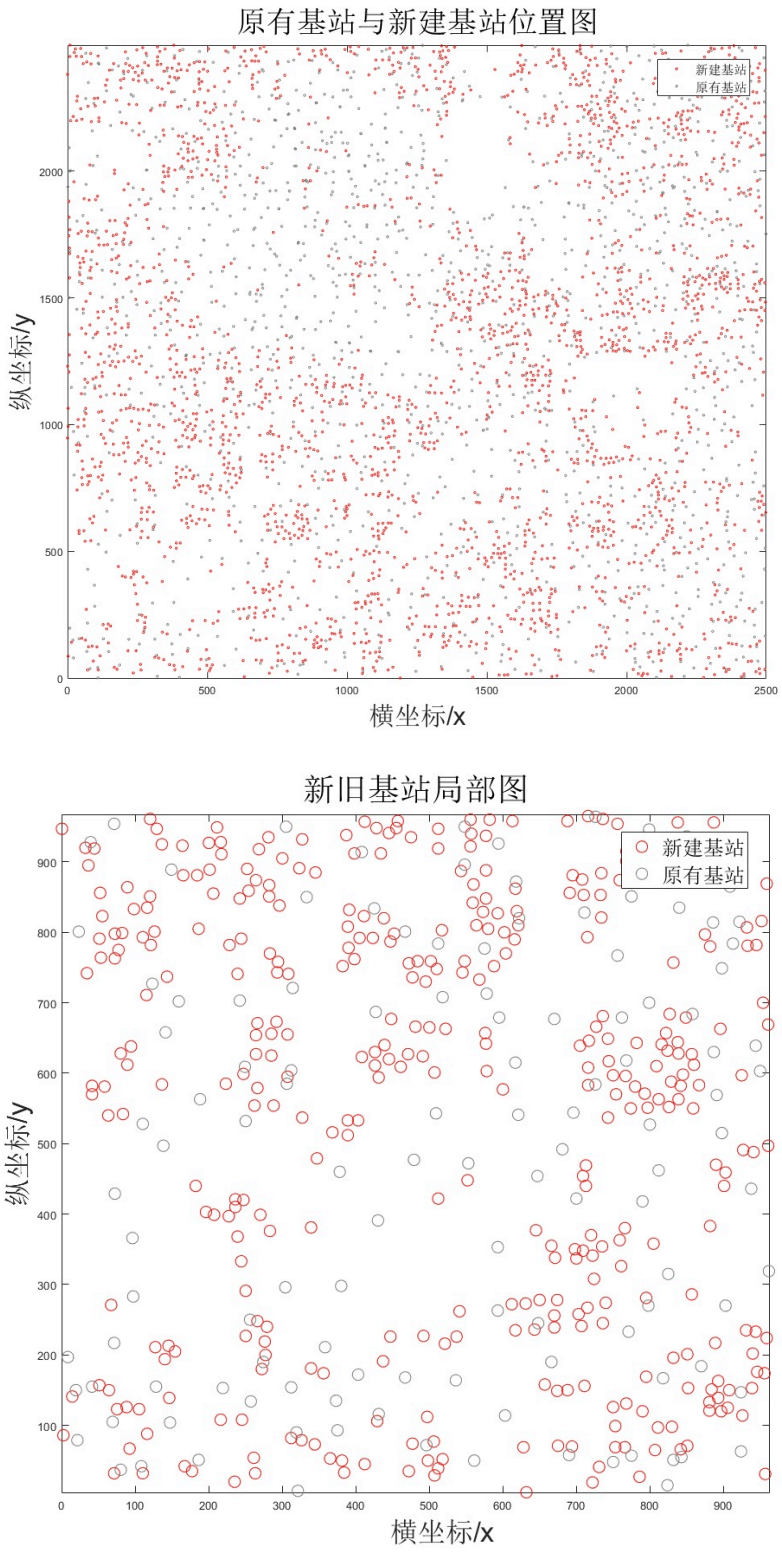

图 7 新建基站位置覆盖图

5.3 模型二的建立与求解

5.3.1 模型的分析

（1）最大覆盖面积

题目给出的模型是每个基站有 3 个 120° 的扇形覆盖，其中任意 2 个扇区的主方向之间夹角不能小于 45°。首先我们发现当一个“叶片”的主方向与另一个“叶片”的主方向夹角为 120° 时，该图形的总覆盖面积最大。

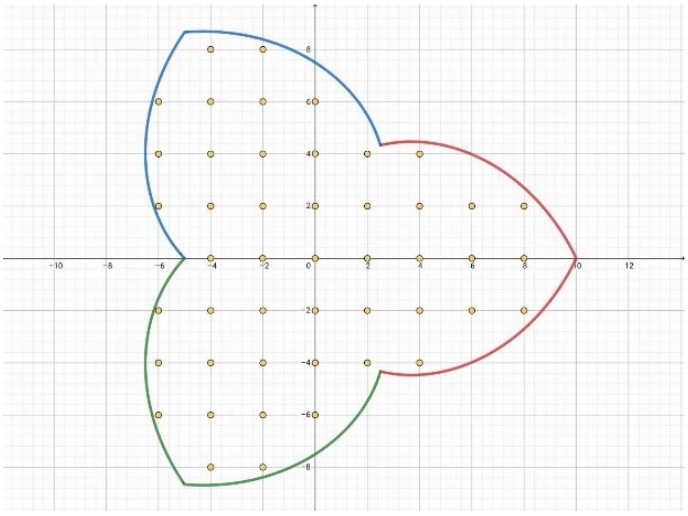

图 8 最大覆盖面积图

（2）最大重合情况

当各扇形之间有夹角小于 120° 时，扇形面积会存在重合，会有部分弱覆盖区被重复覆盖，且随着重合夹角的增大，被重复覆盖的面积会逐渐增大，总的弱覆盖去覆盖面积会减小，所以我们认为扇形之间夹角的最小值应该尽可能大，不应存在较多覆盖区域重合的部分。

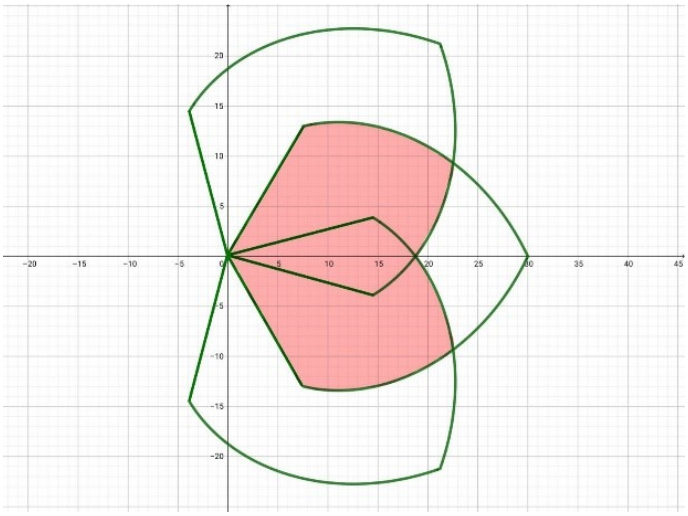

图 9 最大重合角度图

（3）结果简化

通过数据分析部分，我们发现弱覆盖区分布与业务量大小分布整体呈无明显规律，可能存在少数部分区域，花瓣重合度较大的情况覆盖总业务量更大，但认为该部分较少，可以忽略不计。综上分析考虑我们考虑模型，故我们将使用图 8 的情况，即面积不重合的情况进行模型的简化分析。

5.3.2 模型的建立

（1）函数的计算

为对我们的模型进行求解，我们需要求出图 8 的函数表达式。因为扇形从主方向左右 60° 覆盖范围是按照线性逐渐减小，在 60° 的时候，覆盖范围为主方向覆盖范围的一半，所以我们考虑使用极坐标的方式，计算出曲线的函数初始表达式：

$$f_0(x)=\begin{cases}\rho=r\cdot\left(1-\frac{3}{2\pi}\left|\theta+\frac{2\pi}{3}\right|\right), -\pi<\theta\leqslant-\frac{\pi}{3}\\ \rho=r\cdot\left(1-\frac{3}{2\pi}|\theta|\right), -\frac{\pi}{3}<\theta\leqslant\frac{\pi}{3}\\ \rho=r\cdot\left(1-\frac{3}{2\pi}\mid\theta-\frac{2\pi}{3}|\right), \frac{\pi}{3}<\theta\leqslant\pi\end{cases} \tag{3}$$

这是初始位置下的极坐标函数表达式，接下来得到的便是考虑转动角度 t 后的函数表达式：

$$f(x)=\begin{cases}\rho=r\cdot\left(1-\frac{3}{2\pi}\left|\theta+\frac{2\pi}{3}-t\right|\right), -\pi+t<\theta\leqslant-\frac{\pi}{3}+t\\ \rho=r\cdot\left(1-\frac{3}{2\pi}|\theta-t|\right), -\frac{\pi}{3}+t<\theta\leqslant\frac{\pi}{3}+t\\ \rho=r\cdot\left(1-\frac{3}{2\pi}\left|\theta-\frac{2\pi}{3}-t\right|\right), \frac{\pi}{3}+t<\theta\leqslant\pi+t\end{cases} \tag{4}$$

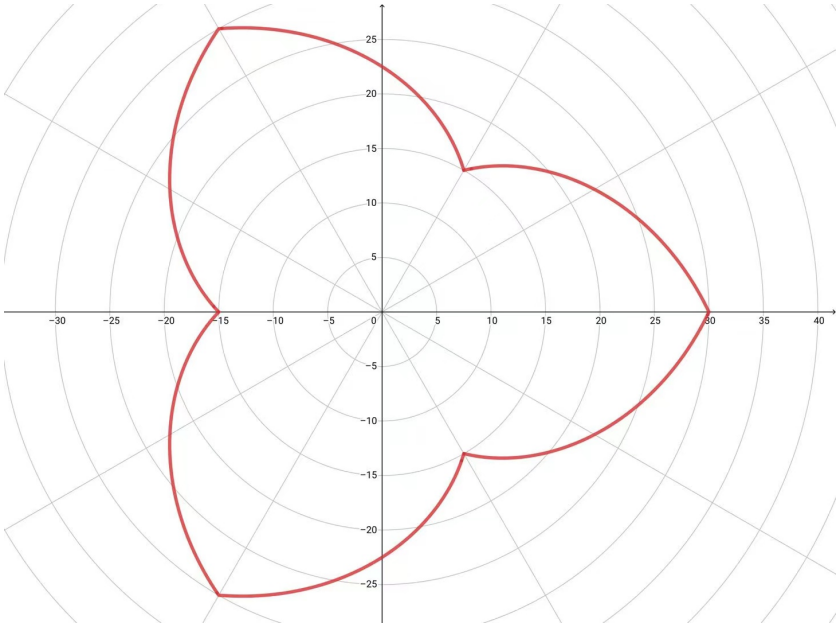

图 10 极坐标表示形式图

（2）距离判定

由于我们的图形“花瓣”，是由分段函数所表示出来的，因此我们在判断一个点是否在里面时，也应该根据本分段函数的公式，进行一个选择。在判别时，我们首先将该点的坐标转化为极坐标的形式，并带入角度 θ 以得到该角度下的长度 ρ。通过对长度和两点间距离公式算出的距离进行比较，我们便可以判定该点是否在我们的“花瓣”中。

$$\sqrt{(x_i - x_1)^2 + (y_i - y_1)^2} \leqslant f\left(\arctan\frac{y_i - y_1}{x_i - x_1}\right) \tag{5}$$

（3）算法求解

在模型一的基础上，我们加一步判断，即当考虑一个点建立基站时，旋转遍历每一种图形，找出覆盖业务量最大的情况。由于该图形是轴对称图形且有三条对称轴，所以每旋转 120° 会与初始形状重合，所以我们只需考虑旋转 120° 的情况。

首先选出未被覆盖的业务量最大的点考虑建立基站，初始位置认为 θ=0。与模型一一致，首先判断该点是否与现有基站冲突，确定出考虑建立基站的坐标点（x_1’,y_1’），再在该地以 θ=0 的情况分别建立宏基站与微基站的覆盖业务量，选出基站类型。确定好基站类型后，计算该基站覆盖的总业务量为 $W_{1.1}$。然后旋转 1 次，使 θ=1°，求出该基站覆盖的总业务量。

以下便是将该图形旋转 45° 的示意图，我们从中可以看到，在图形旋转的过程之中，其内部所覆盖的数据点个数会不断地发生改变。

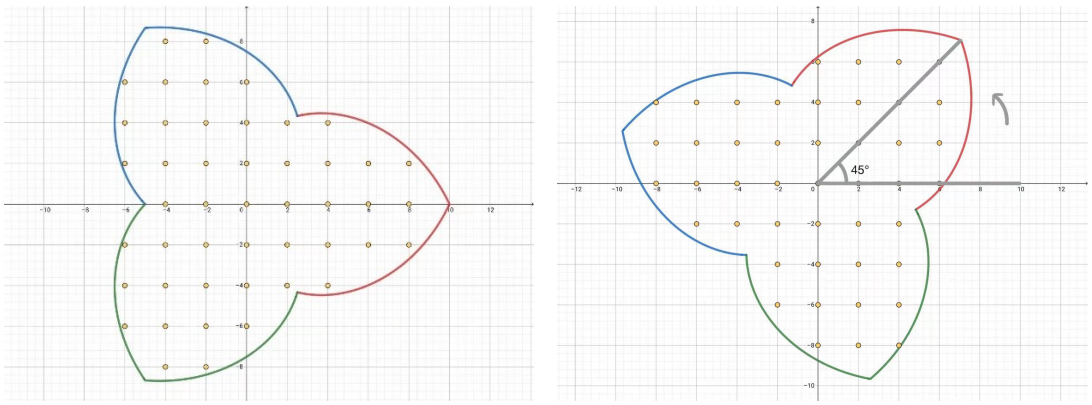

图 11 45° 旋转示意图

在旋转了 120° 之后，我们便可以得到 $W_{1.i}$(i=1,2,⋯,120), 并在其中找出最大的值作为在（x_1’,y_1’）点建立基站得到的覆盖业务量的最大值。

接下来再重复模型一的算法，考虑坐标（x_1’,y_1’）周围八个点建立基站，用同样的旋转法找出周围八个点建立基站带来的覆盖业务量的最大值，最后找出这九个点建立基站的覆盖业务量的最大值。若最大值为 $W_{1.k}$,($k \in i$) 则确定好（x_1’,y_1’）建立基站，若不是，则将最大值的该点设为（x_1’,y_1’），重新与该点周围八个点进行比较，如此重复直到（x_1’,y_1’）新建基站的业务量大于周围八个点新建基站的业务量后，定下该点新建基站。

由于在问题一中我们计算得出所要建造的最优站址的数量 2464，即我们的总新建站址数量为 2464，那么当我们重复 2464 次后，便可定下（x_n’,y_n’），以及相应点的角度。

此时，我们的算法便会停止，我们会根据这些点重新确立最优坐标以及最优的主方向角度。

根据我们问题二最终跑出的坐标结果，我们将问题二的新建基站、问题一的新建基站以及原有基站，绘制出了如下的基站分布叠加图。

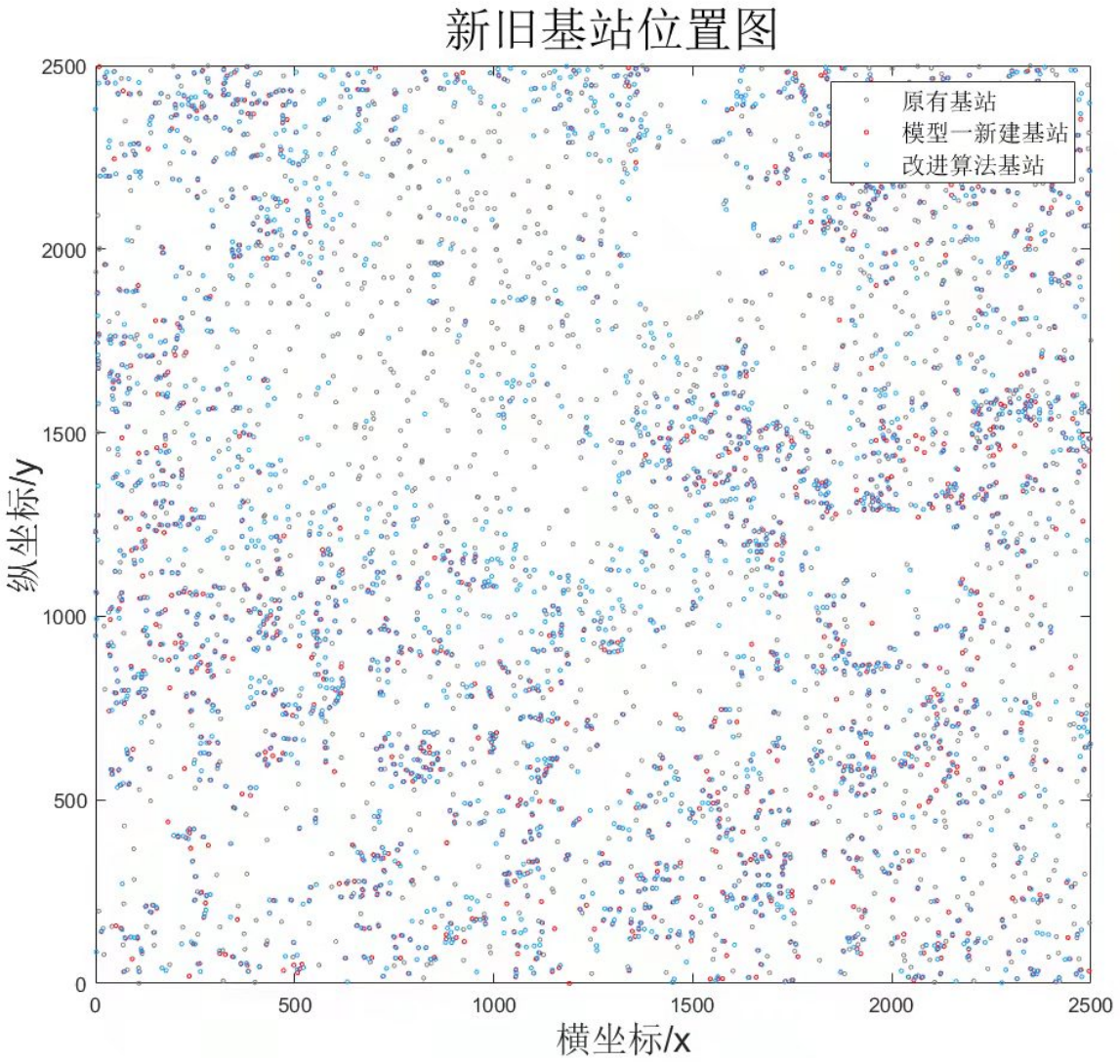

图 12 “花瓣”形基站分布图

通过计算，我们得到在问题二中改进算法的基站所能覆盖的值为 6059824，它占总业务量的比值为 85.879%，并且得出每个基站对应 θ 的信息于附件中。

5.4 模型三的建立与求解

5.4.1 模型分析

问题三是对弱覆盖点进行区域聚类。而聚类的要求是对于距离不大于 20 的 2 个弱覆盖点聚为一类，并且聚类性质具有传递性，最后还考虑聚类所用方法的总时间复杂度尽量低。

而当今较为常用的对数据进行数据处理的方式为：Dbscan 聚类法和 K-Means 聚类法。K-Means 聚类方法是需要用户指定创建的簇数 k，而这个 k 值的选定是非常难以估计的，具有较大的主观性，并且需要根据初始聚类中心来确定一个划分，而这个初始聚类中心的选择对聚类效果有较大的影响，一旦初始值选择不好，可能不乏得到有效的聚类结果。当数据量非常大时，k 均值算法在大数据集上收敛较慢，算法时间开销非常大。

相反，Dbscan 聚类方法不需要像 K-Means 那样预先确定集群的数量，也对异常值孤立点不敏感，并能将高密度数据分离成小集群。此外，它还可以将数据中特有结构的点聚类在一起，使得聚类具有结构性差异，而 K-means 聚类法会将同一结构的点破坏开，单纯按照机械划分进行聚类。

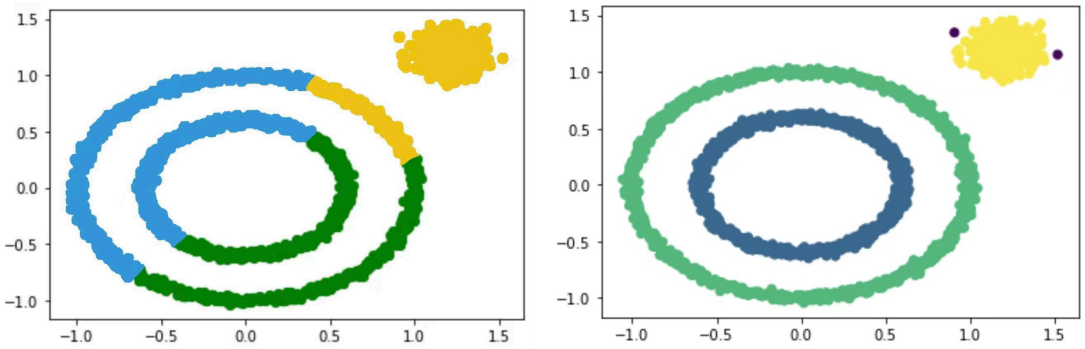

图 13 K-Means 算法和 Dbscan 算法的结果演示图

再分析题目所给的数据，有如下几个特征：

1. 处理的数据体积庞大。
2. 存在较多的孤立点。
3. 难以判断究竟要将地区具体分类为几类。
4. 需要将距离不大于 20 的点聚为一类，且聚类性质具有传递性。

综上所述，选择 Dbscan 聚类方法更为合适。

5.4.2 模型求解

首先我们将所有的弱覆盖点地区看成是带有坐标位置的数据点，同时确定以下两个参数：ϵ 和 minPts。ϵ 为在一个点周围邻近区域的半径，minPts 为邻近区域内至少包含点的个数。根据 Dbscan 算法我们将数据点分为三类：

1. 核心点：在半径 ϵ 内含有不少于 MinPts 数目的点

2. 边界点：在半径 ϵ 内点的数量小于 MinPts，但是落在核心点的领域内

同时，我们定义以下术语：

1. 邻域：指定数据点半径为 20 内的区域称为该对象的邻域。

2. 直接密度可达：对于该弱覆盖区内所有带有坐标位置的数据点的集合，如果其中点 p 在 q 的邻域内，并且 q 为核心点，则 p 从 q 直接密度可达。

3. 密度可达：对于该弱覆盖区内所有带有坐标位置的数据点的集合，给定一串点 p_1，$p_2 \cdots p_n$，p=p_1，q=p_n，假如对象 p_i 从 p_{i-1} 直接密度可达，那么点 q 从点 p 密度可达。

4. 密度相连：存在该弱覆盖区内所有带有坐标位置的数据点的集合中一点 o，如果点 o 到点 p 和点 q 都是密度可达的，那么 p 和 q 密度相连。

现在我们便可以发现，密度可达是直接密度可达的闭包，并且这种关系是非对称的，密度相连是对称关系，所以在该模型中我们采用 Dbscan 算法目的是找到密度相连的最大集合，即满足若 2 个弱覆盖点的距离不大于 20，则这 2 个弱覆盖点应聚为一类，并且聚类性质具有传递性。

接下来我们开始 Dbscan 算法将所有弱覆盖区的点进行分类，算法流程图如下所示：

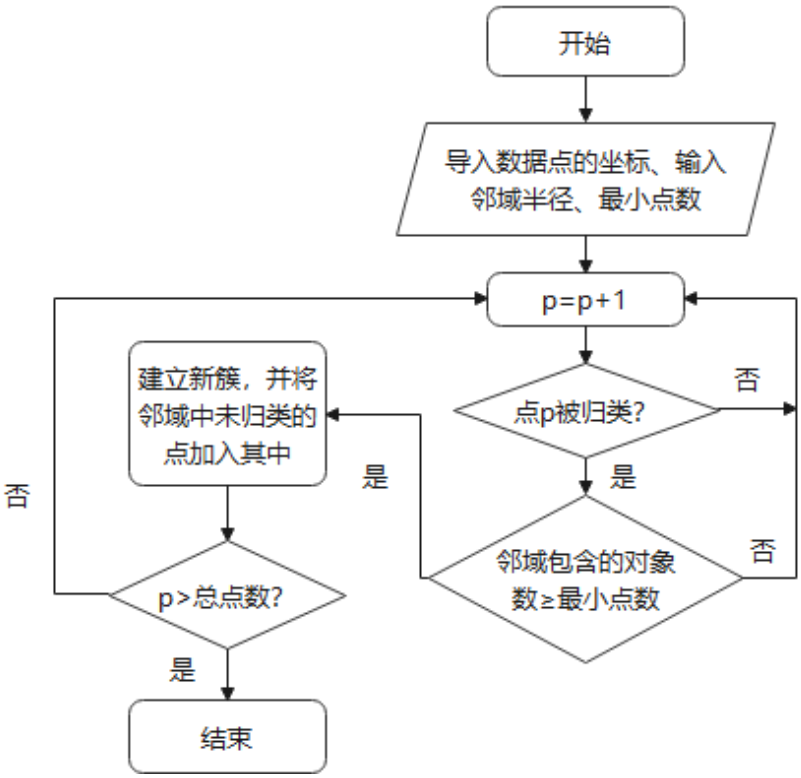

图 14 Dbscan 算法流程图

经过 Dbscan 算法的运行，我们将所有的弱覆盖点共分成了 898 个类，我们使用不同的颜色，对其进行了标记，最终的聚类结果如下所示：

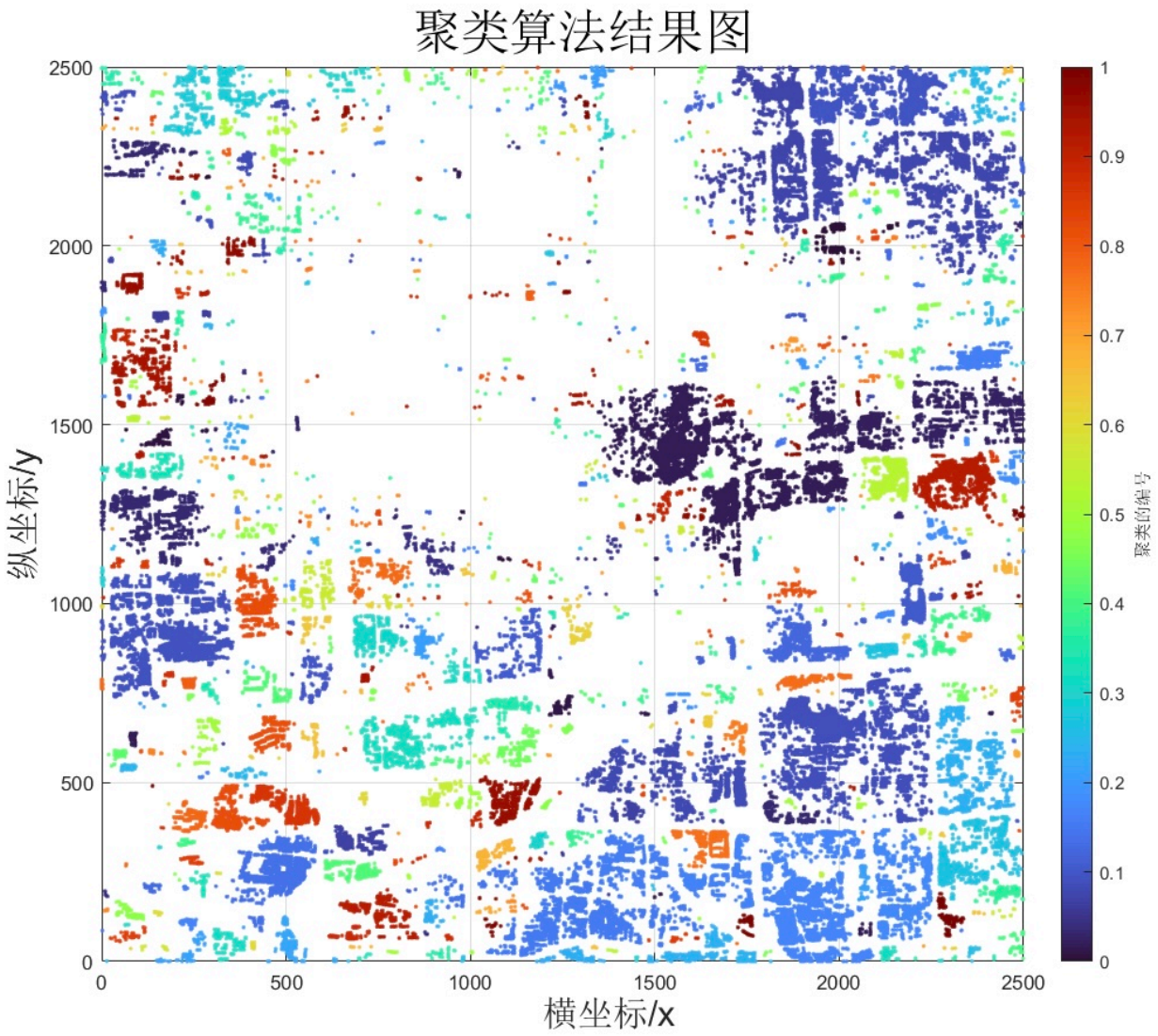

图 15 Dbscan 聚类结果图

六、灵敏度分析

由于我们考虑建造基站的类型只参考题目给出的成本与覆盖量的关系，认为当该点建造宏基站覆盖的业务量是建造微基站的十倍以上，才使用宏基站。

但通过查询数据表明，基站后续的维修，租金与人工成本占基站初始成本的三分之一以上，这些相关数据表明实际也需要考虑尽可能减少新建基站数量，从而影响判断建立基站类型的比值。所以我们考虑改变判断建造基站类型的覆盖业务量比值，并设置为M/m。

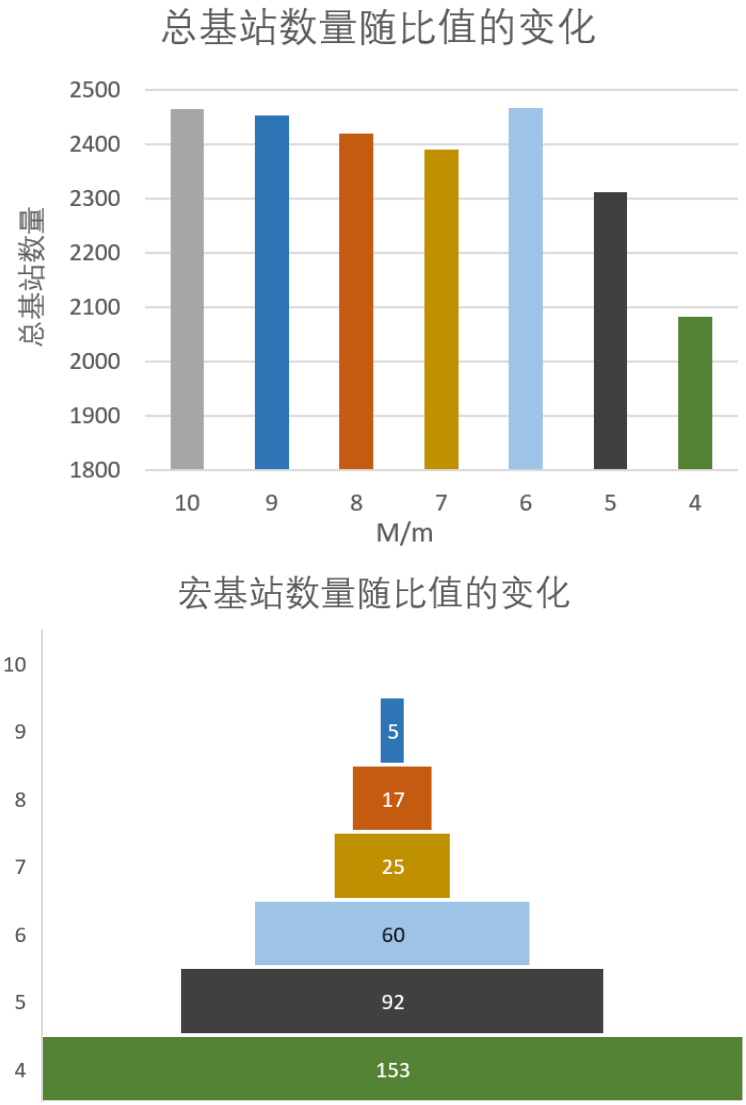

图 16 总基站数和微基站数随 M/m 的变化图

通过上图，我们可以得出，当比值 M/m 从 10 等差减小到 4 时，新建宏基站数目在增加，并且 M/m 越小，新建宏基站数增长得越快；而基站总数在 M/m 从 10 到 6 时，没有明显变化，但从 6 到 4 建立基站总数下降得越来越快。

综上所述，我们可以得到新建宏基站数随比值 M/N 减小越来越敏感，而基站总数在比值 M/m 从 10 到 6 时不敏感，而从 6 到 4 时越来越敏感。

七、模型的评价与推广

7.1 模型的优点

1. 该模型把求整体最优解巧妙地转化为求局部最优解，对问题进行了相当的简化，易于理解与操作。

2. 在模型一中，我们有选择性的对数据进行了结构性优化处理，使得新数据与题目关联度更高。

3. 在模型二中，我们巧妙的将扇区覆盖区域简化成“花瓣”模型，并用严谨的数学推导论证了其可行性，从而降低了题目的复杂度。

4. 文章对模型所得结果进行了直观清晰的可视化处理。

5. 模型原创性很高，大部分内容都是经过自行推导所建立的。

7.2 模型的缺点

1. 采用的贪心策略具有后效性，只能保证前段局部最优解。

2. 模型二采用花瓣模型，失去了每个扇形地灵活性。

7.3 模型的推广

在模型的推广部分中，我们首要考虑了以下 2 个问题：

（1）实际成本

模型只考虑了购买不同基站的成本，但没有考虑如维修成本、人工成本等其他方面的成本。实际上，一个 5G 基站的基础建设成本约为 30 万，但如果我们将租金和人工等其他成本算上，平均下来约为 45 万。在这种情况下，建造过多的微基站会有较高的后期维护成本，因此宏基站和微基站的实际成本比并不是理想的 10：1。

所以在解决实际基站建设问题的时候，我们应该考虑到不同地区租金和人工成本的差异性，并结合当地具体的状况修正比例系数，重新对我们的贪吃算法进行参数修正。

（2）信号覆盖

模型中只考虑了一个基站的覆盖面积，并没有考虑基站本身产生的信号强度会随着距离的增大而不断衰减。因此我们需要引入信号有效覆盖理论中的信号运输损耗表达式。

$$FL(dB) = 20\lg R(\text{ km}) + 20\lg f(GHz) + 92.44 \tag{6}$$

因此，随着 R 增大，信号强度会逐渐减弱，基站边缘本身的弱覆盖点，可能受到墙体等因素的干扰，在基站建立后仍为弱覆盖点。

参考文献

[1] 吴淑花, 蒋成煜. 结合城市发展规划的移动通信基站选址 [J]. 电信工程技术与标准化, 2006, 19(9):3.

[2] 何家爱. 5G 基站规划建设的难点探讨 [J]. 信息通信, 2018(11):2.

[3] 高炜欣, 罗先觉, 朱颖. 贪心算法结合 Hopfield 神经网络优化配电变电站规划 [J]. 电网技术, 2004, 28(7):4.

[4] 刘洋, 陈英武, 谭跃进. 基于贪婪算法的卫星地面站任务规划方法 [J]. 系统工程与电子技术, 2003, 25(10):3.

[5] 周世兵, 徐振源, 唐旭清. K-means 算法最佳聚类数确定方法 [J]. 计算机应用, 2010, 30(8):1995-1998.

[6] 周水庚, 周傲英, 曹晶. 基于数据分区的 DBSCAN 算法 [J]. 计算机研究与发展, 2000, 37(10):7.

[7] 谢飞, 熊立翔. 低压电力网载波通信信号衰减特性的研究 [J]. 电子技术应用, 1998, 24(1):2.

[8] 魏鹏涛, 曾宇, 王海宁, 等. 基于大数据的 5G 基站退服成本估算 [J]. 电子技术应用, 2019, 45(10):5.

附　录

附录 1：问题一的 python 代码

```python
from itertools import product
from cv2 import COLOR_RGB2BGR, cvtColor, imwrite
from matplotlib import pyplot as plt
from numpy import ndarray, zeros
from tqdm import tqdm


iterNCir = lambda r: tuple(
    (i, j)
    for i, j in product(range(-r, r + 1), range(-r, r + 1))
    if i * i + j * j <= r ** 2
)
iter10 = sorted(iterNCir(10), key=lambda x: x[0] ** 2 + x[1] ** 2)
iter30 = sorted(iterNCir(30), key=lambda x: x[0] ** 2 + x[1] ** 2)
# 以 (0, 0) 为中心，10 或 30 为半径的所有整数点，以到原点的距离排序


lostTable = [] # 一些需要特殊处理的点就放在这里，下文会讲


def colorize(val: float) -> "tuple[int]":
    """按数量级着色，可视化的时候调用"""
    if val < 100:
        return 0, int(255 * (val / 100) ** 0.5), 0
    elif val < 1000:
        return 0, 0, 255
    elif val < 10000:
        return 255, 0, 0
    else:
        return 255, 255, 255


def inRect(x: int, y: int) -> bool:
    """判断是否在正方形平面内"""
    return (0 <= x < 2500) and (0 <= y < 2500)


def circleVal(pic: ndarray, x: int, y: int, r) -> float:
    """返回以 (x, y) 为中心，r 为半径的圆内的所有点值的和"""
    return sum(
```

```
            pic[x + i, y + j]
            for i, j in (iter30 if r == 30 else iter10)
            if i * i + j * j <= r ** 2 and inRect(x + i, y + j)
        ) + sum(k for i, j, k in lostTable if (i - x) ** 2 + (j - y) ** 2 < r ** 2)


def tryMove(pic: ndarray, mask: ndarray, x: int, y: int, r: int) -> "tuple[int]":
    """以 (x, y) 为起点，看往哪边移动可以找到极大值"""
    neighbor = lambda x, y: (
        (x + 1, y),
        (x - 1, y),
        (x, y + 1),
        (x, y - 1),
        (x + 1, y + 1),
        (x + 1, y - 1),
        (x - 1, y + 1),
        (x - 1, y - 1),
    ) # 上下左右，还有四个角都要考虑

    # 如果非常不幸，这个坐标一上来就和别的基站重合的
    # 就看看半径范围内有没有不重合的点，有的话就移过去，优先选距离近的
    # 如果不存在，按理来说应该跳过这个点，选第二大的值，但第二大的值不太好求
    # 这里的做法是，先把它加入特殊点，再把这个点的值置为零
    # 这样求最大点的时候就会跳过它，求和 (circleVal) 的时候特殊点单独处理
    if mask[x, y]:
        for i, j in iter30 if r == 30 else iter10:
            if inRect(x + i, y + j) and not mask[x + i, y + j]:
                return tryMove(pic, mask, x + i, y + j, r)
        lostTable.append((x, y, pic[x, y]))
        pic[x, y] = 0
        return None
    # 如果八个方向某个点不和其他基站冲突且可以覆盖更大的值，就移过去
    cont = True
    while cont:
        here = circleVal(pic, x, y, r)
        cont = False
        for i, j in neighbor(x, y):
            if inRect(i, j) and not mask[i, j] and circleVal(pic, i, j, r) > here:
                x, y = i, j
                cont = True
                break
    return x, y
```

```python
def useR(pic: ndarray, x: int, y: int) -> int:
    """
    判断这个坐标比较适合什么半径，哪种性价比更高一点，考虑大小基站的时候用得上
    如果在此画一个大圆比画小圆圈住的业务量多了十倍以上，当然大圆划算
    """
    return 30 if circleVal(pic, x, y, 30) > circleVal(pic, x, y, 10) * 10 else 10


if __name__ == "__main__":
    pic = zeros((2500, 2500), dtype="float64") # 业务量二维图
    img = zeros((2500, 2500, 3), dtype="uint8") # 可视化用的图，不用管
    xs = zeros(182807, dtype="uint16") # 业务量的 x 坐标
    ys = zeros(182807, dtype="uint16") # 业务量的 y 坐标
    zs = zeros(182807, dtype="float64") # 业务量的z坐标（业务量大小理解为高度）
    # 解析csv文件
    with open("data1.csv") as f: # 附件1
        next(f) # next 去掉第一行
        for i, line in tqdm(enumerate(f), total=182807):
            x, y, z = line.split(',')
            xs[i], ys[i], zs[i] = int(x), int(y), float(z)
            pic[xs[i], ys[i]] = zs[i]
            img[xs[i], ys[i]] = colorize(zs[i])

    xs1 = zeros(1474, dtype="uint16") # 现有基站 x 坐标
    ys1 = zeros(1474, dtype="uint16") # 现有基站 y 坐标
    pic1 = zeros((2500, 2500), dtype=bool) # 二维的标记，为真的话说明这里建基站会和
        已有的冲突
    # 解析csv文件
    with open("data2.csv") as f: # 附件2
        next(f)
        for i, line in tqdm(enumerate(f), total=1474):
            _, x, y = line.split(',')
            xs1[i], ys1[i] = int(x), int(y)
            for j, k in iter10:
                if inRect(xs1[i] + j, ys1[i] + k):
                    pic1[xs1[i] + j, ys1[i] + k] = True

    totalSum = 0 #      总共覆盖的值之和
    cost = 0 #          总花费
    points = [] #       基站选址
    pic0 = pic.copy() # 每个点的值
    mask = pic1.copy() # 掩码，用于快速判断两个基站相交
    with tqdm(total=6350607) as bar: # 进度条
        r = 10
```

```
    while totalSum < 6350607: # 6350607 = 0.9 * 总业务量
        xi, yi = divmod(pic0.argmax(), 2500) # 首先取得当前二维业务图中最大值的横纵坐标
        pos = tryMove(pic0, mask, xi, yi, 10) # 求得最终取极大值的坐标
        if pos is None: # 如果为 None 说明一头栽进了基站堆里面，怎么走都会和某个已经建好了的基站冲突，只能跳过这个选址
            continue
        xj, yj = pos

        r = 10
        while (r1 := useR(pic0, xj, yj)) != r:
            r = r1
            xj, yj = tryMove(pic0, mask, xj, yj, r)
        # 具体思路就是，一直循环到上下左右移动和改变半径都不能增大覆盖的业务量为止

        cir = circleVal(pic0, xj, yj, r) # 求出以该点为圆心画圆，圈住的业务量
        totalSum += cir # 加入总共覆盖的业务量
        points.append((xj, yj, r, cir)) # 把该点加入基站选址
        cost += 10 if r == 30 else 1 # 花费也要增加
        bar.update(int(cir)) # 手动推进进度条
        for i, j in iter30 if r == 30 else iter10:
            if inRect(xj + i, yj + j):
                pic0[xj + i, yj + j] = 0 # 把圆圈圈住的点的业务量都置为零，防止多次计算
        for i, j in iter10:
            if inRect(xj + i, yj + j):
                mask[xj + i, yj + j] = True # 10 米范围内的 mask 都置为真，表明放在此处会和已有的基站冲突

# 把新建基站的信息写入文件
with open("选址.txt", 'w') as out:
    out.write(f"Totally cost: {cost}\n\n")
    out.write("x坐标 | y坐标 | 半径 | 覆盖的业务量\n")
    for x, y, r, val in points:
        out.write(f"{x:4} | {y:4} | {r:2} | {val:10.4f}\n")

# 绘图:
outPic1 = zeros((2500, 2500, 3), dtype="uint8")
outPic2 = zeros((2500, 2500, 3), dtype="uint8")
outPic1[pic0 > 0] = 0, 255, 0
outPic1[mask] = 255, 0, 0
outPic1[pic1] = (255,) * 3
outPic2[pic1] = (255,) * 3
imwrite("keep and selected site.png", cvtColor(outPic1, COLOR_RGB2BGR))
```

```
    imwrite("keep site.png", cvtColor(outPic2, COLOR_RGB2BGR))
    imwrite("visualization.png", cvtColor(img, COLOR_RGB2BGR))
    plt.imshow(outPic1, interpolation="bilinear")
    plt.show()

    barX = ["0-10", "10-100", "100-1000", "1000-10000", "10000+"]
    barY = [
        len(zs[zs < 10]),
        len(zs[(zs > 10) & (zs < 100)]),
        len(zs[(zs > 100) & (zs < 1000)]),
        len(zs[(zs > 1000) & (zs < 10000)]),
        len(zs[zs > 10000]),
    ]
    plt.bar(barX, barY)
    plt.savefig("数量级统计.png")
```

附录 2：问题二的 python 代码

```
from itertools import product
from matplotlib import pyplot as plt
from numpy import ndarray, zeros
from tqdm import tqdm
from cv2 import COLOR_RGB2BGR, cvtColor, imwrite
import math



iterNCir = lambda r: tuple(
    (i, j)
    for i, j in product(range(-r, r + 1), range(-r, r + 1))
    if i * i + j * j <= r ** 2
)
iter10 = sorted(iterNCir(10), key=lambda x: x[0] ** 2 + x[1] ** 2)
iter30 = sorted(iterNCir(30), key=lambda x: x[0] ** 2 + x[1] ** 2)
# 以 (0, 0) 为中心，10 或 30 为半径的所有整数点，以到原点的距离排序


lostTable = [] # 一些需要特殊处理的点就放在这里，下文会讲


def colorize(val: float) -> "tuple[int]":
    """按数量级着色，可视化的时候调用"""
    if val < 100:
        return 0, int(255 * (val / 100) ** 0.5), 0
```

```
    elif val < 1000:
        return 0, 0, 255
    elif val < 10000:
        return 255, 0, 0
    else:
        return 255, 255, 255


def inRect(x: int, y: int) -> bool:
    """判断是否在正方形平面内"""
    return (0 <= x < 2500) and (0 <= y < 2500)


Π = math.pi
r = 10
  = 0
n = 0
  = 0
area_all_base_stations = []
area_all_base_stations_cover = []
area_one__base_station_new_cover = []
total_traffic = []


def clover(self):

    if   > -Π and   <= -(Π/3):
         = r*(1-(3/(2*Π))*abs( +(2*Π/3)-(n/360)*(2*Π)))
    elif   > -(Π/3) and   <= (Π/3):
         = r*(1-(3/(2*Π))*abs( -(n/360)*(2*Π)))
    elif   > (Π/3) and   <= Π:
         = r*(1-(3/(2*Π))*abs( -((2*Π)/3)-(n/360)*(2*Π)))


def tryMove(pic: ndarray, mask: ndarray, x: int, y: int, r: int) -> "tuple[int]":
    """以 (x, y) 为起点，看往哪边移动可以找到极大值"""

    neighbor = lambda x, y: (
        (x + 1, y),
        (x - 1, y),
        (x, y + 1),
        (x, y - 1),
        (x + 1, y + 1),
        (x + 1, y - 1),
```

```
    (x - 1, y + 1),
    (x - 1, y - 1),
) # 上下左右，还有四个角都要考虑

# 如果非常不幸，这个坐标一上来就和别的基站重合的
# 就看看半径范围内有没有不重合的点，有的话就移过去，优先选距离近的
# 如果不存在，按理来说应该跳过这个点，选第二大的值，但第二大的值不太好求
# 这里的做法是，先把它加入特殊点，再把这个点的值置为零
# 这样求最大点的时候就会跳过它，求和（circleVal）的时候特殊点单独处理
if mask[x, y]:
    for i, j in iter30 if r == 30 else iter10:
        if inRect(x + i, y + j) and not mask[x + i, y + j]:
            return tryMove(pic, mask, x + i, y + j, r)
    lostTable.append((x, y, pic[x, y]))
    pic[x, y] = 0
    return None
# 如果八个方向某个点不和其他基站冲突且可以覆盖更大的值，就移过去
cont = True
while cont:
    here = clover(pic, x, y, r)
    cont = False
    for i, j in neighbor(x, y):
        if inRect(i, j) and not mask[i, j] and clover(pic, i, j, r) > here:
            x, y = i, j
            cont = True
            break
return x, y

for p in area_all_base_stations:
    for n, index in enumerate(range(120)):
        total_traffic_max = 0
        for q in area_all_base_stations_cover[index]:
            #该弱覆盖点还在新覆盖区域中
             0 = ((p[0],q[0])**2 + (p[0],q[0])**2)*.5
              = math.atan((q[1]-p[1])/(q[0]-p[0]))
            clover( )
            #进行比较
            if  0 <=  :
                area_one__base_station_new_cover.append(q)
               #记录覆盖业务量总值
                total_traffic.append(q[2])

        #将最大t储存在max里面
        if total_traffic > total_traffic_max:
```

```
                total_traffic_max = total_traffic
                index_max = index

        return  [index_max]
        return total_traffic


def useR(pic: ndarray, x: int, y: int) -> int:
    """
    判断这个坐标比较适合什么半径，哪种性价比更高一点，考虑大小基站的时候用得上
    如果在此画一个大圆比画小圆圈住的业务量多了十倍以上，当然大圆划算
    """
    return 30 if clover(pic, x, y, 30) > clover(pic, x, y, 10) * 10 else 10


if __name__ == "__main__":
    pic = zeros((2500, 2500), dtype="float64") # 业务量二维图
    img = zeros((2500, 2500, 3), dtype="uint8") # 可视化用的图，不用管
    xs = zeros(182807, dtype="uint16") # 业务量的 x 坐标
    ys = zeros(182807, dtype="uint16") # 业务量的 y 坐标
    zs = zeros(182807, dtype="float64") # 业务量的z坐标（业务量大小理解为高度）
    # 解析csv文件
    with open("data1.csv") as f: # 附件1
        next(f) # next 去掉第一行
        for i, line in tqdm(enumerate(f), total=182807):
            x, y, z = line.split(',')
            xs[i], ys[i], zs[i] = int(x), int(y), float(z)
            pic[xs[i], ys[i]] = zs[i]
            img[xs[i], ys[i]] = colorize(zs[i])

    xs1 = zeros(1474, dtype="uint16") # 现有基站 x 坐标
    ys1 = zeros(1474, dtype="uint16") # 现有基站 y 坐标
    pic1 = zeros((2500, 2500), dtype=bool) # 二维的标记，为真的话说明这里建基站会和
        已有的冲突
    # 解析csv文件
    with open("data2.csv") as f: # 附件2
        next(f)
        for i, line in tqdm(enumerate(f), total=1474):
            _, x, y = line.split(',')
            xs1[i], ys1[i] = int(x), int(y)
            for j, k in iter10:
                if inRect(xs1[i] + j, ys1[i] + k):
                    pic1[xs1[i] + j, ys1[i] + k] = True
```

```
totalSum = 0 #      总共覆盖的值之和
cost = 0 #          总花费
points = [] #       基站选址
pic0 = pic.copy() # 每个点的值
mask = pic1.copy() # 掩码，用于快速判断两个基站相交
with tqdm(total=6350607) as bar: # 进度条
    r = 10
    while totalSum < 6350607: # 6350607 = 0.9 * 总业务量
        xi, yi = divmod(pic0.argmax(), 2500) # 首先取得当前二维业务图中最大值的横
            纵坐标
        pos = tryMove(pic0, mask, xi, yi, 10) # 求得最终取极大值的坐标
        if pos is None: # 如果为 None 说明一头栽进了基站堆里面，怎么走都会和某个已
            经建好了的基站冲突，只能跳过这个选址
            continue
        xj, yj = pos
        r=10
        while (r1 := useR(pic0, xj, yj)) != r:
            r = r1
            xj, yj = tryMove(pic0, mask, xj, yj, r)
        # 具体思路就是，一直循环到上下左右移动和改变半径都不能增大覆盖的业务量为止

        cir = clover(pic0, xj, yj, r) # 求出以该点为圆心画圆，圈住的业务量
        totalSum += cir # 加入总共覆盖的业务量
        points.append((xj, yj, r, cir)) # 把该点加入基站选址
        cost += 10 if r == 30 else 1 # 花费也要增加
        bar.update(int(cir)) # 手动推进进度条
        for i, j in iter30 if r == 30 else iter10:
            if inRect(xj + i, yj + j):
                pic0[xj + i, yj + j] = 0 # 把圆圈圈住的点的业务量都置为零，防止多次
                    计算
        for i, j in iter10:
            if inRect(xj + i, yj + j):
                mask[xj + i, yj + j] = True # 10 米范围内的 mask 都置为真，表明放
                    在此处会和已有的基站冲突

# 把新建基站的信息写入文件
with open("选址9.txt", 'w') as out:
    out.write(f"Totally cost: {cost}\n\n")
    out.write("x坐标 | y坐标 | 半径 | 覆盖的业务量\n")
    for x, y, r, val in points:
        out.write(f"{x:4} | {y:4} | {r:2} | {val:10.4f}\n")

# 绘图:
outPic1 = zeros((2500, 2500, 3), dtype="uint8")
```

```
    outPic2 = zeros((2500, 2500, 3), dtype="uint8")
    outPic1[pic0 > 0] = 0, 255, 0
    outPic1[mask] = 255, 0, 0
    outPic1[pic1] = (255,) * 3
    outPic2[pic1] = (255,) * 3
    imwrite("keep and selected site.png", cvtColor(outPic1, COLOR_RGB2BGR))
    imwrite("keep site.png", cvtColor(outPic2, COLOR_RGB2BGR))
    imwrite("visualization.png", cvtColor(img, COLOR_RGB2BGR))
    plt.imshow(outPic1, interpolation="bilinear")
    plt.show()

    barX = ["0-10", "10-100", "100-1000", "1000-10000", "10000+"]
    barY = [
        len(zs[zs < 10]),
        len(zs[(zs > 10) & (zs < 100)]),
        len(zs[(zs > 100) & (zs < 1000)]),
        len(zs[(zs > 1000) & (zs < 10000)]),
        len(zs[zs > 10000]),
    ]
    plt.bar(barX, barY)
    plt.savefig("数量级统计.png")
```

附录 3：问题三的 matlab 代码

```
function mathor
    tic
    fileID = fopen('data1.csv', 'r');
    xsys = textscan(fileID, "\%f,\%f,\%f", 182807);
    fclose(fileID);
    xsys = [xsys\{1\} xsys\{2\}];
    idx = sortrows([xsys dbscan(xsys, 20, 1)], 3);
    \% disp(size(idx))
    fileID = fopen('dataOut.csv', 'w');
    for i = 1:182807
        fprintf(fileID, "\%4d, \%4d, \%3d\n", idx(i, :));
    end
    fclose(fileID);
    toc
    % 历时 53.460746 秒
end
```