CENTRAL SOUTH UNIVERSITY

无线传感器网络技术课程实验报告

姓　　名：	周密
学　　院：	计算机学院
学　　号：	8210182308
邮　　箱:	zhoumicsu@csu.edu.cn
指导老师：	张金焕
日　　期：	2021 年 12 月 2 日早

摘要

无线传感器网络作为物联网的一部分，占据着极为重要的作用，本课程实验基于 Python 编程语言，并且着手于网络的拓扑结构和网络分簇协议部分，其中实验一使用均匀分布，建立了一个圆形拓扑网络结构，并且确定了网络的 Sink 节点，实验二实现了 LEACH 分簇协议，基于实验一建立的圆形拓扑网络结构，基于欧几里德距离完成了网络分簇，并且基于能量消耗模型完成了对 LEACH 分簇协议低能耗效果的仿真。

关键词： 无线传感器网络; 网络拓扑结构; Python; LEACH 协议; 网络能量;

Abstract

As a part of the Internet of things, wireless sensor networks play a very important role. The experiment of this course is based on python programming language and focuses on the network topology and network clustering protocol. Experiment 1 uses uniform distribution to establish a circular topology network structure and determine the sink nodes of the network. Experiment 2 implements leach clustering protocol, Based on the circular topology network structure established in Experiment 1, the network clustering is completed based on Euclidean distance, and the low energy consumption effect of leach clustering protocol is simulated based on energy consumption model.

Keywords: Wireless sensor network; Network topology; Python; Leach agreement; Network energy;

目录

1 实验一：生成网络拓扑结构

1.1 实验目的与要求

随机生成一圆形无线传感器网络，其中汇聚节点 Sink 位于圆心位置，半径为 R，节点个数为 N，节点发射和感知半径为 r。要求：

1. 网络中节点位置参数保存，可多次使用；
2. 根据参数，绘出网络拓扑结构图；

1.2 实验内容、结果与分析

1. 生成圆形网络拓扑结构

随机生成圆形无线传感器网络，其中汇聚节点 Sink 位于圆心位置，半径 R 设为 10，节点个数 N 为 100，节点发射和感知半径为 r 为 10。实验结果如图 1所示。

通过 python 的随机数生成函数，首先生成 -1 $+1$ 内均匀分布的两个随机数，分别用作某特定节点的横纵坐标，然后放大到 $-R$ $+R$ 范围内，最后通过筛选除横纵坐标距离离 Sink/坐标原点距离不大于 R 的坐标点作为节点的坐标，再重复成功生成 $N-1$ 个节点，即完成圆形网络拓扑结构的生成。

通过调用 python 中的散点图绘制函数 scatter、以及折线图绘制函数 plot 完成整个网络的绘制和显示。

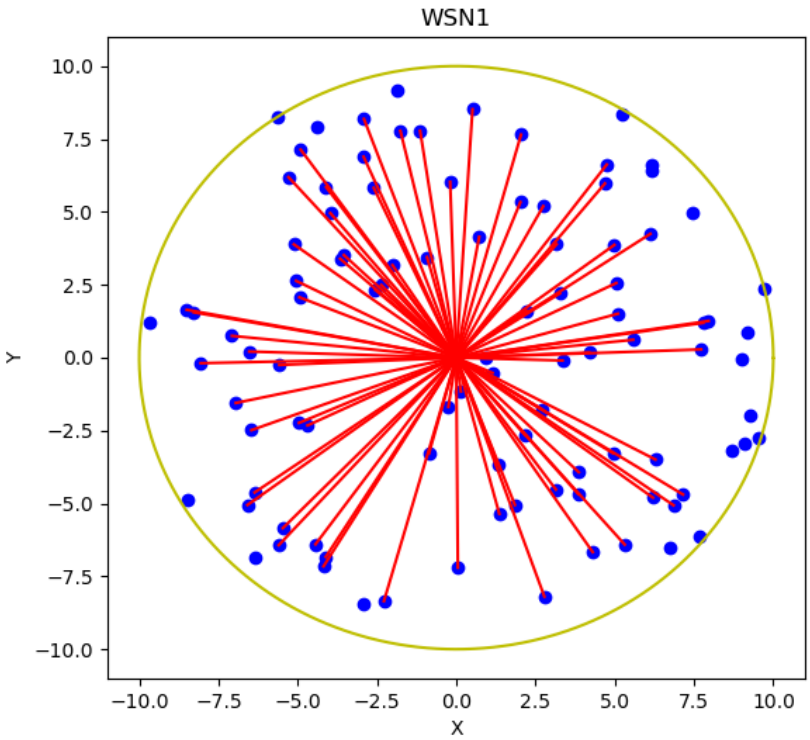

图 1: 未分簇的圆形拓扑无线传感器网络，Sink 节点位于坐标原点，网络半径为 10，通信半径为 8.7

2. 保存生成的网络拓扑结构

本文实现了网络拓扑结构节点位置的存储，存储的文件名为"top.txt"，其中一行为一个坐标点，使用"," 分隔，而不同节点坐标之间使用换行符分隔，实验结果如图 2所示。

使用 python 的常用数据处理库 numpy 即可完成二维 list 的存储功能。

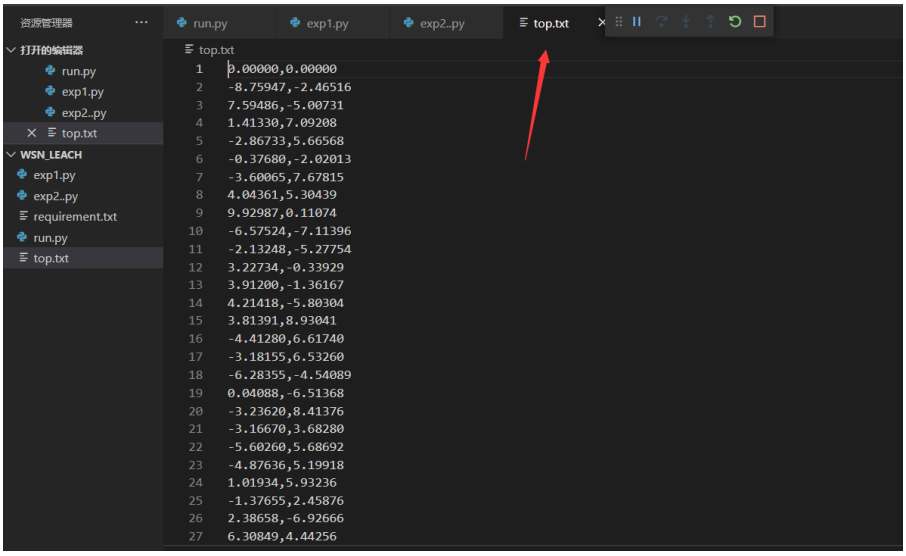

图 2: 网络拓扑结构保存为 top.txt 文件

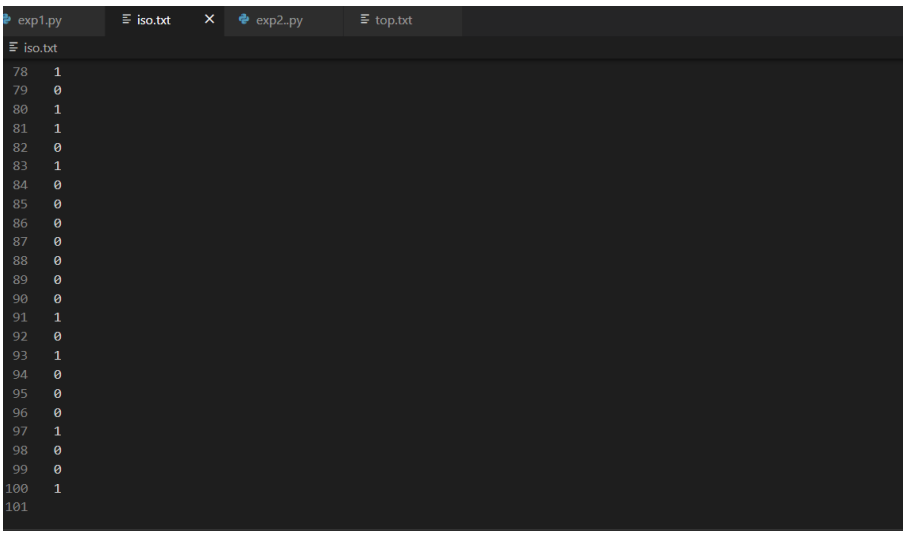

图 3: 网络中节点是否为孤立节点的标志保存为 iso.txt

1.3 实验代码

特此说明：本文实验一、实验二代码参考了 csdn 博客，但是做了必要的修改：https://blog.csdn.net/wsh596823919/article/details/79981408

```
import numpy as np
import matplotlib.pyplot as plt
import turtle

def run():
```

```
    """
    1、node_factory(N, R, r): 生成N个节点的列表， 拓扑半径为R, 通信半径r
    2、show_plt(top, iso, R): 迭代每次聚类结果, 显示连线图
    :return:
    """
      # 节点数目和圆形网络半径
      N = 100
      R = 10
      r = 8.7
      # 获取初始节点列表
      top, iso = node_factory(N, R, r)

      # 显示拓扑结构
      show_plt(top, iso, R)

      # 保存拓扑结构为txt文件
      save_top(top, iso)

    def dist(v_A, v_B):
    """
    判断两个节点之间的欧几里得距离
    :param v_A: A 二维向量
    :param v_B: B 二维向量
    :return: 一维距离
    """
      return np.sqrt(np.power((v_A[0] - v_B[0]), 2) + np.power((v_A[1] - v_B[1]), 2))

    def node_factory(N, R, r):
    """
    生成N个节点的集合
    :param N: 节点的数目
    :param R: 圆形拓扑半径
    :param r: 节点通信半径
    :param nodes: 节点的集合
    :param iso: 标志:是否为孤立节点
    :return: 节点集合nodes=[[x,y],[x,y]...] + 标志iso
    """
      nodes = []
      iso = []

      sinknode = [0, 0]
      nodes.append(sinknode)
      iso.append(0)

      i = 0
      while i < N-1:
        # 在1*1矩阵生成[x,y]坐标, 并根据离sink节点的距离做判断是否为孤立节点
        node = [np.random.uniform(-1, 1)*R, np.random.uniform(-1, 1)*R]
        if dist(node, sinknode) < R and dist(node, sinknode) > r:
          nodes.append(node)
          iso.append(1)
          i = i + 1
```

```
        elif dist(node, sinknode) < R and dist(node, sinknode) < r:
            nodes.append(node)
            iso.append(0)
            i = i + 1


    return nodes, iso


    def show_plt(top, iso, R):
    """
    显示分类图
    :param top: [[类1...],[类2...]....]-->[簇首,成员,成员...]
    :return:
    """
      fig = plt.figure()
      ax1 = plt.gca()

      # 设置标题
      ax1.set_title('WSN1')
      # 设置X轴标签
      plt.xlabel('X')
      # 设置Y轴标签
      plt.ylabel('Y')

      for i in range(len(top)):
        plt.scatter(top[i][0], top[i][1], color='b')
        if i != 0 and iso[i] != 1:
          plt.plot([top[0][0], top[i][0]], [top[0][1], top[i][1]], color='r')

      a, b = (top[0][0], top[0][1])
      theta = np.arange(0, 2*np.pi, 0.01)
      x = a + R * np.cos(theta)
      y = b + R * np.sin(theta)
      plt.plot(x, y, 'y')
      plt.show()

    def save_top(top, iso):
    """
    :param top: 网络拓扑结构
    :param iso: 是否为孤立节点标志
    """
      np.savetxt('top.txt', top, fmt='%.5f', delimiter=',')
      np.savetxt('iso.txt', iso, fmt='%d', delimiter=',')

    if __name__ == '__main__':
      run()
```

2 实验二：无线传感器网络 LEACH 协议

2.1 实验目的与要求

根据实验 1 中的网络拓扑及其参数，实现 LEACH 分簇协议。

要求：在一轮数据收集中

1. 标绘出网络拓扑中的簇头节点；

2. 标绘出每个簇头的簇成员。

根据实验 2 中的分簇，统计 LEACH 协议允许的结果，假设数据包的大小为 L, 能量消耗模型为：

$$\begin{cases} E_{\mathrm{Tx}}(k,\ d) = E_{\mathrm{elec}}k + E_{\mathrm{fs}}kd^2 & d < d_{\mathrm{o}} \\ E_{\mathrm{Tx}}(k,\ d) = E_{\mathrm{elec}}k + E_{mp}kd^4 & d \geq d_{\mathrm{o}}, \end{cases}$$

图 4: 网络能量消耗模型（发送方）

$$E_{\mathrm{Rx}}(k,\ d) = E_{\mathrm{elec}}k$$

图 5: 网络能量消耗模型（接收方）

Table 5 Simulation parameters

Parameter	Default value
topographical area, m	$200 \times 200 \times 20$ m
location of the base station, m	(190, 190, 19)
number of nodes	50
transmission SR of nodes (meters)	80 m
initial energy of nodes	5 J
E_{elec}	50 nJ/bit
E_{fs}	10 pJ/bit/m^2
E_{mp}	0.0013 pJ/bit/m^4
data packet size	2.4 *k*-bit
number of transmission packets	5×10^3
initial buffer size of nodes	20
simulation time	14 h

图 6: 网络模型参数参考

要求：在一轮数据收集中，每个节点都产生一个数据包，并汇聚到 Sink 节点

1. 统计每个节点发送和接收的数据包个数，并直观显示出来；

2. 统计每个节点能量消耗，并给出三维能量消耗图。

2.2 实验内容、结果与分析

1. 绘制网络拓扑的簇头节点及其簇成员

在完成整个网络拓扑结构的建模之后，通过 LEACH 算法可以对网络中的节点进行分簇。对于 LEACH，这里有必要对它做一些必要的介绍：

LEACH 来源于 Wendi Rabiner Heinzelman, Anantha Chandrakasan, 和 Hari Balakrishnan 三人在 2000 年 Proceedings of the 33rd Hawaii International Conference on System Sciences 上的一篇文章 Energy-Efficient Communication Protocol forWireless Microsensor Networks。

LEACH 协议全称是“低功耗自适应集簇分层型协议”(Low Energy Adaptive Clustering Hierarchy)。

该算法基本思想是：以循环的方式随机选择蔟首节点，将整个网络的能量负载平均分配到每个传感器节点中，从而达到降低网络能源消耗、提高网络整体生存时间的目的。仿真表明，与一般的平面多跳路由协议和静态分层算法相比，LEACH 协议可以将网络生命周期延长 15%。

LEACH 在运行过程中不断的循环执行蔟的重构过程，每个蔟重构过程可以用回合的概念来描述。每个回合可以分成两个阶段：蔟的建立阶段和传输数据的稳定阶段。为了节省资源开销，稳定阶段的持续时间要大于建立阶段的持续时间。蔟的建立过程可分成 4 个阶段：蔟首节点的选择、蔟首节点的广播、蔟首节点的建立和调度机制的生成。

蔟首节点的选择依据网络中所需要的蔟首节点总数和迄今为止每个节点已成为蔟首节点的次数来决定。具体的选择办法是：每个传感器节点随机选择 0 1 之间的一个值。如果选定的值小于某一个阀值，那么这个节点成为蔟首节点。

选定蔟首节点后，通过广播告知整个网络。网络中的其他节点根据接收信息的信号强度决定从属的蔟，并通知相应的蔟首节点，完成蔟的建立。最后，蔟首节点采用 TDMA 方式为蔟中每个节点分配向其传递数据的时间点。

稳定阶段中，传感器节点将采集的数据传送到蔟首节点。蔟首节点对蔟中所有节点所采集的数据进行信息融合后再传送给汇聚节点，这是一种叫少通信业务量的合理工作模型。稳定阶段持续一段时间后，网络重新进入蔟的建立阶段，进行下一回合的蔟重构，不断循环，每个蔟采用不同的 CDMA 代码进行通信来减少其他蔟内节点的干扰。

LEACH 协议主要分为两个阶段：即簇建立阶段 (setup phase) 和稳定运行阶段 (ready phase)。簇建立阶段和稳定运行阶段所持续的时间总和为一轮 (round)。为减少协议开销，稳定运行阶段的持续时间要长于簇建立阶段。

在簇建立阶段，传感器节点随机生成一个 0，1 之间的随机数，并且与阈值 T(n) 做比较，如果小于该阈值，则该节点就会当选为簇头。参考 csdn 博客https://blog.csdn.net/wangh0802/article/details/78656775

在这里，通过概率选取簇头，形成簇头组成的 list，随后对剩余的每一个节点，都要对簇头 list 进行遍历，找到离该节点最近的簇头，记录下该距离，并将该节点归入到该簇头的簇下。实验结果如图 7和图 8所示。

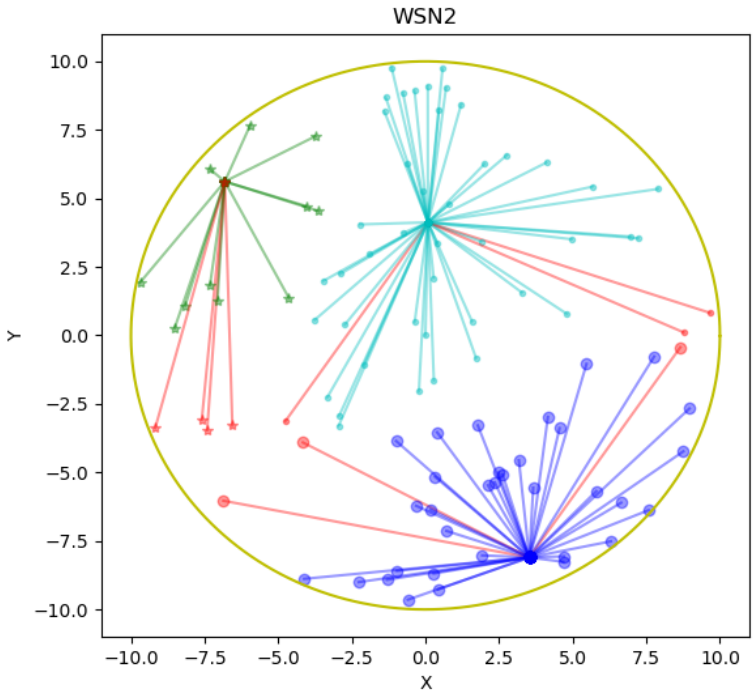

图 7: 使用 LEACH 分簇协议分簇后的网络结构（第一轮），连线表示节点隶属于哪一簇，红线连接表示超出通信半径 r=8.7，此节点为该簇中的孤立节点

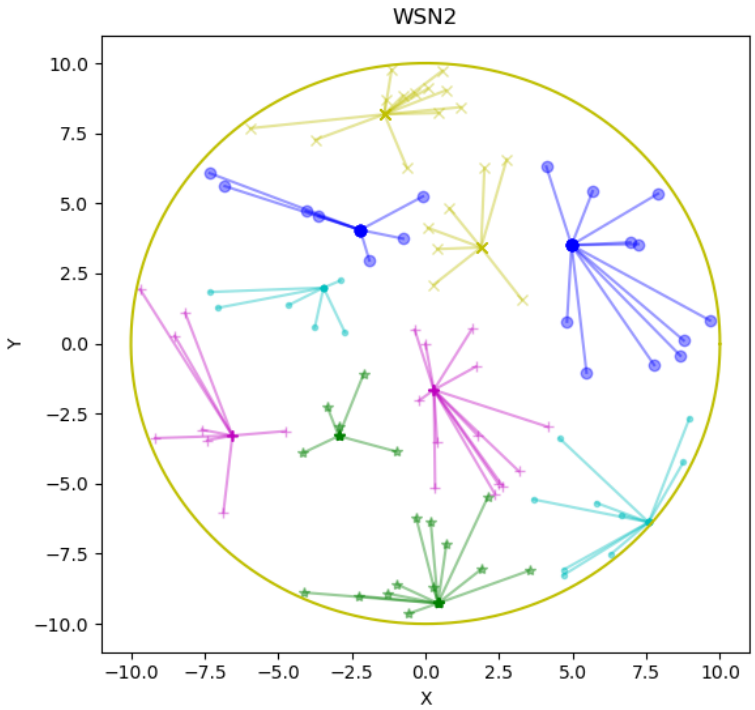

图 8: 使用 LEACH 分簇协议分簇后的网络结构（第二轮），连线表示节点隶属于哪一簇，红线连接表示超出通信半径 r=8.7，此节点为该簇中的孤立节点

2. 数据包的发送以及能量仿真

这里对实验内容做了一定的简化，只考虑了在完成分簇之后，各个簇中的节点向簇头发送数据的能量损失，而没有考虑其他的能量损失，但是这种简化是有一定道理的，在网络拓扑结构比较合理，在区域内分布比较均匀的情况下，实际上网络的绝大部分能量是由有效的数据包交换消耗的。因此我们忽略了一下能量消耗：在分簇时，各个节点在物理上实际上时彼此独立分隔的，然后要做到对于整个网络的自组织功能，实现网络按照特定算法分簇，是需要进行数据包交换的，即由能量的消耗。在每一簇内，当节点们把数据包传送给簇头时，我们只考虑单跳的情况，并且认为信道是时不变的理想的，并且我们认为所有的数据包的长度相同，数目也是稳定的。在完成从节点到簇中心的传送后，我们没有考虑到 Sink 节点的传送。所以综合以上几点，我们这里提出的仿真不是完整的仿真，而是多种近似、简化后的结果。

在实验二的 1 中，我们可以获得每一个簇中的节点，并且可以得到簇到其中各个节点的距离，在获得该距离之后，我们即可通过上述提出的能量消耗模型进行仿真，值得注意的是：收发双方的能量消耗模型是不同的，并且发送方的能量消耗与距离有关，这一点是很直观的：当距离较远时，如果要保证一定的信噪比使得接收方能够成功恢复数据，那么就需要加大信号的发送功率。并且，这里我们对于每个节点不仅仅存储其坐标节点，我们同时保存了它的剩余能量值，以便此处进行网络能量的分析。

对于每个簇中的孤立的节点，我们做这样的处理：能够发送数据按照发送数据包消耗能量模型进行处理，但是对于该节点所在的簇头节点来讲，并不会损耗能量。

在出现第一个网络节点能量耗尽时，实验结果的三维能量分布图如图 9所示。

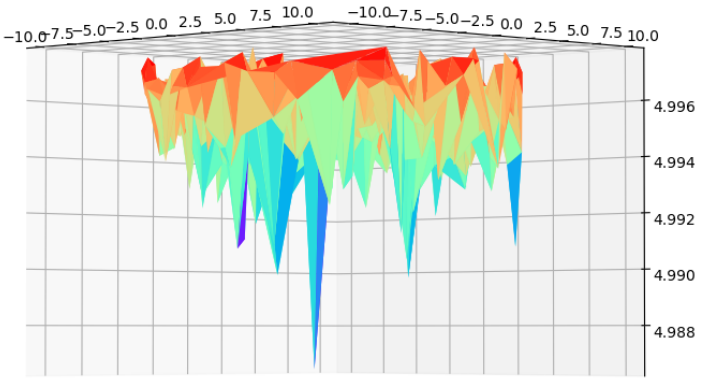

图 9: 在迭代循环 1000 次之后，各个节点的能量剩余三维图，为了效果更好，这里使用 500 个节点

从之前的分析可以看出，LEACH 算法存在的一定的缺点：在 LEACH 算法中，每一轮循环都要重新构造簇，而构造簇的能量开销比较大。其次，远离汇聚节点的簇头节点可能会由于长距离发送数据而过早耗尽自身能量，造成网络分割。另外，LEACH 算法没有考虑簇头节

点当前的能量状况，如果能量很低的节点当选为簇头节点，那么将会加速该节点的死亡，影响整个网络的生命周期。

并且 LEACH 的使用存在一定的局限性：1. 由于 LEACH 假定所有节点能够与汇聚节点直接通信，并且每个节点都具备支持不同 MAC 协议的计算能力，因此该协议不适合在大规模的无线传感器网络中应用。2. 协议没有说明簇头节点的数目怎么分布才能及于整个网络。因此，很可能出现被选的簇头节点集中在网络某一区域的现象，这样就会使得一些节点的周围没有任何簇头节点。3. 由于 LEACH 假定在最初的簇头选择回合中，所有的节点都携带相同的能量，并且每个成为簇头的节点都消耗大致相同的能量。因此，协议不适合节点能量不均衡的网络。

2.3 实验代码

特此说明：本文实验一、实验二代码参考了 csdn 博客，但是做了必要的修改：https://blog.csdn.net/wsh596823919/article/details/79981408

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


def dist(v_A, v_B):
"""
判断两个节点之间的一维距离
:param v_A: A 二维向量
:param v_B: B 二维向量
:return: 一维距离
"""
  return np.sqrt(np.power((v_A[0] - v_B[0]), 2) + np.power((v_A[1] - v_B[1]), 2))


def node_factory(N, R, r, energy=50):
"""
生成N个节点的拓扑网络
:param N: 网络中节点个数
:param R: 圆形拓扑半径
:para r: 通信半径范围，超出此范围为某簇的孤立节点
:param selected_flag: 标志:是否被选择为簇首-->初始化为0
:param energy: 能量
:return: 节点集合nodes=[[x,y,e],[x,y,e]...]
"""
  nodes = []
  selected_flag = []
  iso = []

  # 中心sink节点
  sinknode = [0, 0, energy]
  nodes.append(sinknode)
  selected_flag.append(0)

```

```
    # 随机生成圆形拓扑网络
    i = 0
    while i < N-1:
      # 在1*1矩阵生成[x,y]坐标
      node = [np.random.uniform(-1, 1)*R, np.random.uniform(-1, 1)*R, energy]
      if dist(node, sinknode) < R and dist(node, sinknode) > r:
        nodes.append(node)
        selected_flag.append(0)
        iso.append(1)
        i = i + 1
      elif dist(node, sinknode) < R and dist(node, sinknode) < r:
        nodes.append(node)
        selected_flag.append(0)
        iso.append(0)
        i = i + 1


    return nodes, selected_flag, iso


def sel_heads(r, nodes, flags):
"""
使用leach协议，选取簇头（注意这里还没有开始进入正式的分簇，这里只选了簇头）
:param r: 轮数
:param nodes: 节点列表
:param flags: 选择标志
:param P: 比例因子
:return: 簇首列表heads,簇成员列表members
"""
    # 阈值函数 Tn 使用leach计算
    P = 0.05 * (100 / len(nodes))
    Tn = P / (1 - P * (r % (1 / P)))
    heads = []  # 簇首列表
    members = []    # 簇成员列表
    n_head = 0  # 本轮簇首数
    rands = [np.random.random() for _ in range(len(nodes))] # 对每个节点生成对应的随机数，用于筛选簇头

    # 遍历随机数列表，选取簇首
    for i in range(len(nodes)):
      # 随机数低于阈值-->选为簇首
      if rands[i] <= Tn:
        flags[i] = 1
        heads.append(nodes[i])
        n_head += 1
      # 随机数高于阈值
      else:
        members.append(nodes[i])

    return heads, members
```

```
86      def classify(nodes, flag, r, mode=1, k=20):
87      """
88      对网络进行簇分类
89      :param nodes: 节点列表
90      :param flag: 节点标记
91      :param mode: 0-->显示图片(死亡节点不显示)  1-->显示结束轮数
92      :param k: 轮数
93      :return: 簇分类结果列表 classes[[类1..],[类2...],.....]  [类1...簇首...簇成员]
94      """
95        # 能量损耗模型的参数
96        b = 2400
97        e_elec = 5*np.power(10., -9)
98        e_fs = 10*np.power(10., -12)
99        e_mp = 0.0013*np.power(10., -12)
100       d_0 = 80
101
102       # k轮的有效集合: 无死亡节点
103       iter_classes = []
104       # 是否已有节点能量为0
105       e_is_empty = 0
106
107       # 迭代r轮
108       for r in range(k):
109         # mode1: 若无死亡节点 继续迭代
110         if e_is_empty == 0:
111           # 获取簇首列表, 簇成员列表
112           heads, members = sel_heads(r,nodes,flag)
113
114           # 建立簇类的列表
115           if len(heads) == 0:
116             break
117           classes = [[] for _ in range(len(heads))]
118
119           # 将簇首作为首节点添加到聚类列表中
120           for i in range(len(heads)):
121             classes[i].append(heads[i])
122
123           # 簇分类:遍历节点node
124           for member in members:
125
126             # 选取距离最小的节点
127             dist_min = 100000
128
129             # 判断和每个簇首的距离
130             for i in range(len(heads)):
131
132               dist_heads = dist(member, heads[i])
133
134               # 找到距离最小的簇头对应的heads下标i
135               if dist_heads < dist_min:
136               dist_min = dist_heads
137               head_cla = i
```

```
            if dist_min==1:
              print("本轮没有簇首!")
              break
              # 添加到距离最小的簇首对应的聚类列表中

            classes[head_cla].append(member)

            # 正式的数据传输过程, 使用能量消耗模型
            if int(member[2]) > 0 and int(heads[head_cla][2]) > 0:
              if dist_min < d_0:
                member[2] -= e_elec*b+e_fs*b*dist_min
              else:
                member[2] -= e_elec*b+e_mp*b*dist_min

              if dist([member[0], member[1]], [heads[head_cla][0], heads[head_cla][1]])
    <= r:
                heads[head_cla][2] -= e_elec*b
              else:
                pass


              # heads[head_cla][2] -= e_elec*b
            else:
            e_is_empty = mode
            break
          iter_classes.append(classes)

        else:
          print("第", r, "轮能量耗尽")
          break

      return iter_classes


    def show_plt(classes, R, r):
    """
    显示分类图
    :param classes: [[类1...],[类2...]....]-->[簇首,成员,成员...]
    :param R: 圆形拓扑半径
    :para r: 通信半径范围, 超出此范围为某簇的孤立节点
    :return:
    """
      fig = plt.figure()
      ax1 = plt.gca()

      # 设置标题
      ax1.set_title('WSN2')
      # 设置X轴标签
      plt.xlabel('X')
      # 设置Y轴标签
      plt.ylabel('Y')

```

```
    # 簇内的显示点图标及连线颜色，以得到较好的显示结果
    icon = ['o', '*', '.', 'x', '+', 's']
    color = ['b', 'g', 'c', 'y', 'm']

    x, y, e = [], [], []

    # 对不同的簇进行不同的显示，以得到较好的显示结果
    for i in range(len(classes)):
      centor = classes[i][0]
      x.append(centor[0])
      y.append(centor[1])
      e.append(centor[2])
      for point in classes[i]:
        if point[2] > 0 and dist(centor, point) < r:
          ax1.plot([centor[0], point[0]], [centor[1], point[1]], c=color[i % 5], marker
=icon[i % 5], alpha=0.4)
        elif point[2] > 0 and dist(centor, point) > r:
          ax1.plot([centor[0], point[0]], [centor[1], point[1]], c='r', marker=icon[i %
 5], alpha=0.4)
        else:
          pass


    a, b = (0., 0.)
    theta = np.arange(0, 2*np.pi, 0.01)
    x = a + R * np.cos(theta)
    y = b + R * np.sin(theta)
    plt.plot(x, y, 'y')

    # 显示
    plt.show()

  def show_eninfo(iter_classes):
    fig = plt.figure()
    ax1 = Axes3D(fig)
    lastclass = iter_classes[-1]

    x, y, e = [], [], []

    # 将所有节点的剩余能量统计起来，用于后续能量三维图的显示
    for i in range(len(lastclass)):
      centor = lastclass[i][0]
      x.append(centor[0])
      y.append(centor[1])
      e.append(centor[2])
      for point in lastclass[i]:
        if point[2] > 0:
          x.append(point[0])
          y.append(point[1])
          e.append(point[2])
        else:
          pass
```

```
    # 需要进行数据类型转换list->ndarray，才能进行三维图像的显示
    x = np.array(x)
    y = np.array(y)
    e = np.array(e)

    # 显示三维图像
    ax1.plot_trisurf(x, y, e, cmap='rainbow')
    plt.show()

def run():
"""
"""
    # N = int(input("请输入节点个数:"))
    N = 500
    R = 10
    r = 8.7

    # 获取初始节点列表
    nodes, flag, iso = node_factory(N, R, r, energy=5)
    # 对节点列表进行簇分类,mode为模式 2种
    iter_classes = classify(nodes,flag, r, mode=1, k=200)
    # 迭代每次聚类结果，显示连线图
    for classes in iter_classes:
        # 显示分类结果
        show_plt(classes, R, r)

    show_eninfo(iter_classes)
if __name__ == '__main__':
    run()
```

3 实验总结与体会

3.1 感悟

无线传感器网络的仿真工具有很多，之前老师推荐的时 omnet++，omnet++ 作为专业的网络仿真开源工具，具有十分强大的功能。本文使用的是 python 语言直接作为开发语言，python 作为一种解释性语言，十分容易上手，能够让开发者多关注实验业务方面，而不是纠结于语法。同时，python 语言现在越来越流行，基于以上考虑，本文使用 python 作为开发语言。

无线传感器网络的仿真中，我总结主要有几个重点内容：网络拓扑结构的建立，网络分簇协议的实现，网络转化路由协议的实现等，在本实验中，出于时间有限，所以主要内容是前两个相对容易的部分。

1. 网络拓扑结构的建立

需要用到 python 中的随机数生成的点：np.random.funciton(random()/uniform/.....)，用

到欧几里德距离的计算，用到结构体、类等面向对象的概念，字典、列表等数据结构。

2. 网络分簇协议的实现

分簇协议类似于机器学习中属于无监督学习的聚类的方法，不过这里的分簇协议需要考虑数据包转发、能量效率等问题，而在聚类中只需要考虑距离。

其中 LEACH 协议存在一定的缺点和局限性，在 LEACH 算法中，每一轮循环都要重新构造簇，而构造簇的能量开销比较大。其次，远离汇聚节点的簇头节点可能会由于长距离发送数据而过早耗尽自身能量，造成网络分割。另外，LEACH 算法没有考虑簇头节点当前的能量状况，如果能量很低的节点当选为簇头节点，那么将会加速该节点的死亡，影响整个网络的生命周期。

并且 LEACH 的使用存在一定的局限性：1. 由于 LEACH 假定所有节点能够与汇聚节点直接通信，并且每个节点都具备支持不同 MAC 协议的计算能力，因此该协议不适合在大规模的无线传感器网络中应用。2. 协议没有说明簇头节点的数目怎么分布才能及于整个网络。因此，很可能出现被选的簇头节点集中在网络某一区域的现象，这样就会使得一些节点的周围没有任何簇头节点。3. 由于 LEACH 假定在最初的簇头选择回合中，所有的节点都携带相同的能量，并且每个成为簇头的节点都消耗大致相同的能量。因此，协议不适合节点能量不均衡的网络。

3.2 收获

收获了 python 的编程技术的提升，收获了 debug 的能力，练习了 latex 写报告、论文的一些技巧，巩固了无线传感器网络课程的基础知识（如 LEACH 分簇协议等），了解和熟悉了无线传感器网络的仿真过程，以及使用编程语言实现它的方法。