一　实验名称：传感器网络连通性实验

班级：物联网 1601　姓名：尹达恒　学号：1030616134

二　实验要求

1、 编写一个能够按照指定的传感器数量和传感器布置范围随机生成传感器网络邻接矩阵的算法；

2、 编写一个能够根据传感器网络邻接矩阵和传感器通信半径计算网络连通度的算法；

3、 在不同传感器数量条件下，固定随机生成传感器的数量，绘制多条传感器通信半径-传感器网络连通度曲线；

4、 在不同通信半径条件下，固定传感器通信半径，绘制多条传感器数量-传感器网络连通度曲线。

三　实验步骤

3.1　邻接矩阵随机生成算法

为简便起见，假定传感器布置在一个方形区域内，传感器网络整体为一个无向图，传感器邻接矩阵表示传感器之间的物理间距。指定传感器数量 n 和布置范围边长 d 随机生成传感器网络邻接矩阵 G 的算法可以简化为：

1. 随机生成 n 个传感器的位置 $P = \{(x_i, y_i)|0 < x_i < d, 0 < y_i < d, 0 < i \leq n\}$；
2. 计算各个传感器之间的距离，形成邻接矩阵 G。

算法如下：在实际程序中，$\{G_{ij}\}$ 的计算使用 Pytorch 中的矩阵操作，调用 GPU 进行并行计算以提升速度。

Algorithm 1 邻接矩阵生成算法

Require: 传感器数量 n、传感器布置范围边长 d
Ensure: n 为自然数、$d > 0$
function AdjacencyMatrix(n,d)
　$P \leftarrow n$ 位随机数组 $\{(x_i, y_i)|0 < x_i < d, 0 < y_i < d, 0 < i \leq n\}$
　$G \leftarrow \{G_{ij}\} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
　return G
end function

3.2　网络连通度计算方法

已知邻接矩阵 G 和传感器通信半径 r 求网络连通度的算法可以分为两步：首先由阵 G 和离 r 求出传感器网络的连通性矩阵 C，然后由 C 次方求和求得网络的总连通度。

连通性矩阵 C 的求得是对网络 G 中节点连通性进行依次判断的过程，对邻接矩阵 G 中的每个元素 G_{ij}，如果 $G_{ij} < r$，则 $C_{ij} = 1$ 表示节点间连通，否则 $C_{ij} = 0$ 表示节点间不连通。

按照网络连通度的有关定理，给定网络连通性矩阵 C 求网络连通度的方法可以用一系列矩阵次方求和实现。对于一个给定的无向图连通性矩阵 C，网络的第 k 跳连通性可以用 C 的 k 次方表示：C^k。C^k 中元素为 0 表示节点 k 跳不连通，元素不为 0 表示节点 k 跳连通。一个有 n 个节点的网络的总连通性矩阵 S 可以通过各跳的连通性矩阵求和来计算：$S = \sum_{k=1}^{n} C^k$。网络连通性可以用总连通性矩阵 S 中是否有 0 元素表示：如果总连通性矩阵 S 中没有 0 元素说明网络连通，反之说明网络不连通。为提升计算速度，当计算到某一跳的总连通性矩阵元素全部为 1 时可以直接得到矩阵 100% 连通而立即停止计算。故可得如 Algorithm2的连通度计算方法。在实际程序中，$\{C_{ij}\}$、$C' \leftarrow C' \times C$ 和 $S \leftarrow S + C'$ 的计算使

Algorithm 2 网络连通度计算方法

Require: 邻接矩阵 G、传感器数量 n、传感器通信半径 r
Ensure: n 为自然数、G 为 $n \times n$ 对称矩阵对角线元素为 0、$r \geq 0$

function CALCULATECONNECTIVITY(G,n,r)
  $C \leftarrow \{C_{ij}\} = \begin{cases} 1 & G_{ij} < r \\ 0 & G_{ij} \geq r \end{cases}$
  $C' \leftarrow C$
  $S \leftarrow C$
  for $k = 1 \rightarrow n$ **do**
    if S 中没有 0 元素 **then**
      return 1
    end if
    $C' \leftarrow C' \times C$
    $S \leftarrow S + C'$
  end for
  if S 中没有 0 元素 **then**
    return 1
  end if
  return 0
end function

用 Pytorch 中的矩阵操作，调用 GPU 进行并行计算以提升速度。

3.3 绘制传感器通信半径-传感器网络连通度曲线

绘制传感器通信半径-传感器网络连通度曲线时，传感器的数量 n 和布置范围边长 d 固定，传感器通信半径从 0 开始增加到 d，可获得一系列传感器通信半径-传感器网络连通度对应关系，以通信半径为 x 轴，网络连通度为 y 轴，绘制出的曲线即为一条传感器通信半径-传感器网络连通度曲线。改变传感器的数量 n 多次绘制，可以得到不同传感器的数量下传感器通信半径-传感器网络连通度曲线。为减小偶然性，每次求传感器网络连通度时都要多次计算取平均值。算法如 Algorithm3。

3.4 绘制传感器数量-传感器网络连通度曲线

绘制传感器数量-传感器网络连通度曲线时，传感器的通信半径 r 和布置范围边长 d 固定，传感器数量从 2 开始依次增加，可获得一系列传感器数量-传感器网络连通度对应关系，以传感器数量为 x 轴，网络连通度为 y 轴，绘制出的曲线即为一条传感器数量-传感器网络连通度曲线。改变传感器的通信半径 r 多次绘制，可以得到不同通信半径下传感器数量-传感器网络连通度曲线。为减小偶然性，每次求传感器网络连通度时都要多次计算取平均值。算法如 Algorithm4。

Algorithm 3 绘制传感器通信半径-传感器网络连通度曲线

Require: 传感器布置范围边长 d，重复计算次数 a，传感器数量限制 n_{max}

for $n = 2 \to n_{max}$ **do**
  $x \leftarrow$ 空数组
  $y \leftarrow$ 空数组
  for $r = 0 \to \sqrt{2}d$ **do**
    $average = 0$
    for a 次循环 **do**
      $G =$AdjacencyMatrix(n,d)
      $t =$CalculateConnectivity(G,n,r)
      $average = average + t$
    end for
    $average = average/a$
    r 值放入数组 x
    $average$ 值放入数组 y
  end for
  绘制一条 x, y 关系曲线
end for

Algorithm 4 绘制传感器通信半径-传感器网络连通度曲线

Require: 传感器布置范围边长 d，重复计算次数 a，传感器数量限制 n_{max}

for $r = 0 \to \sqrt{2}d$ **do**
  $x \leftarrow$ 空数组
  $y \leftarrow$ 空数组
  for $n = 2 \to n_{max}$ **do**
    $average = 0$
    for a 次循环 **do**
      $G =$AdjacencyMatrix(n,d)
      $t =$CalculateConnectivity(G,n,r)
      $average = average + t$
    end for
    $average = average/a$
    n 值放入数组 x
    $average$ 值放入数组 y
  end for
  绘制一条 x, y 关系曲线
end for

四　实验结果

4.1　传感器通信半径-传感器网络连通度曲线

传感器布置范围边长 $d = 10$、重复计算次数 $a = 2000$ 时，取 $n \in \{10, 20, 30, 40, 50\}$，$r$ 为 0.01～10 之间间隔 0.1 的 100 个浮点数，绘制传感器通信半径-传感器网络连通度曲线，程序运行环境为 Windows 10 + CUDA 10.0 + Python 3.6 + Pytorch 1.0，CPU 为 Intel i5-7200U，GPU 为 Nvidia GeForce 940MX，程序运行时间约 865.8s，计算结果如图 1。

4.2　传感器数量-传感器网络连通度曲线

传感器布置范围边长 $d = 10$、重复计算次数 $a = 2000$ 时，取 $r \in \{2, 4, 6, 8, 10\}$，n 为 1～25 共 25 个整数，绘制传感器数量-传感器网络连通度曲线，程序运行环境和硬件配置同上，程序运行时间约 71.86s，计算结果如图 2。

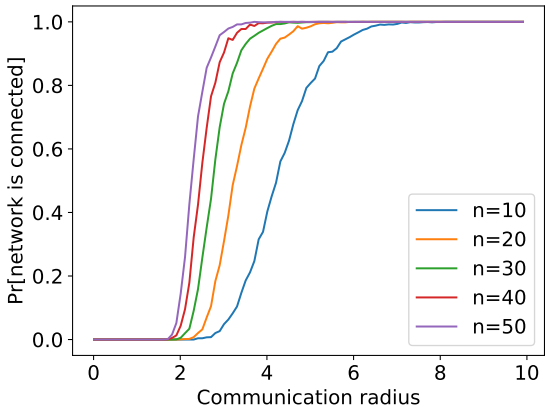

图 1. 通信半径-网络连通度曲线

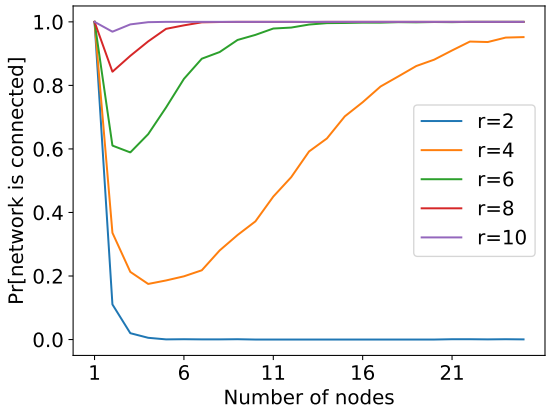

图 2. 传感器数量-网络连通度曲线

五　实验结果分析

由图 1可知，在传感器数目一定的情况下，传感器的通信半径越大，网络的连通度越高；图 2可知，在传感器通信半径一定的情况下，单位面积内的传感器的数目越多，网络的连通度越高。但是不管是增加传感器数目还是提高传感器的通信半径，网络的连通度的增加都是有极限的，传感器的通信半径或传感器数目提高到一定程度都会造成网络资源的浪费，在实际传感网系统设计过程中应有所取舍。