西安電子科技大學
XIDIAN UNIVERSITY

博弈论大作业一

多人雪堆博弈 (Snowdrift Game)

任俊杰
1702052
17170120015

智能控制 多人雪堆博弈

一、雪堆博弈简介

1.1 雪堆博弈具体化

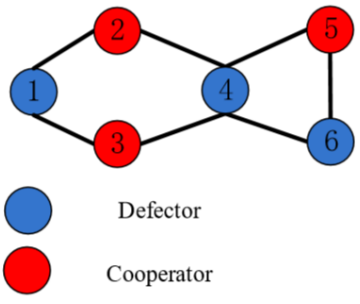
在一个风雪交加的夜晚,两人相向而来,被一个雪堆所阻,假设铲除这个雪堆使道路通畅需要的代价为 r，如果道路通畅则带给每个人的好处量化为 1。如果两人一齐动手铲雪，则他们的收益为 1，如果只有一人铲雪,虽然两个人都可以回家,但是背叛者逃避了劳动,它的收益为 1+r,而合作者的收益为 1-r;如果两人都选择不合作,两人都被雪堆挡住而无法回家,他们的收益都为 0。

雪堆博弈的收益矩阵如下（0<r<1)：

	C	D
C	1，1	1-r，1+r
D	1+r，1-r	0，0

1.2 多人参与的雪堆博弈

比如图中 6 参与人进行雪堆博弈，蓝色代表所 选策略为 D,红色代表所选策略为 C。



演化博弈：从一个随机的初始状态开始，博弈人依次调整自己的策略使自己的收益最大化，整个网络处于一个演化的过程中。

可以利用如下方法找到网络博弈的纳什均衡：

1. 随机给每个节点初试状态设置为 C 或 D

2. 依次对每个节点计算其采用 C 或者 D 的收益，改变其策略使其收益最

大

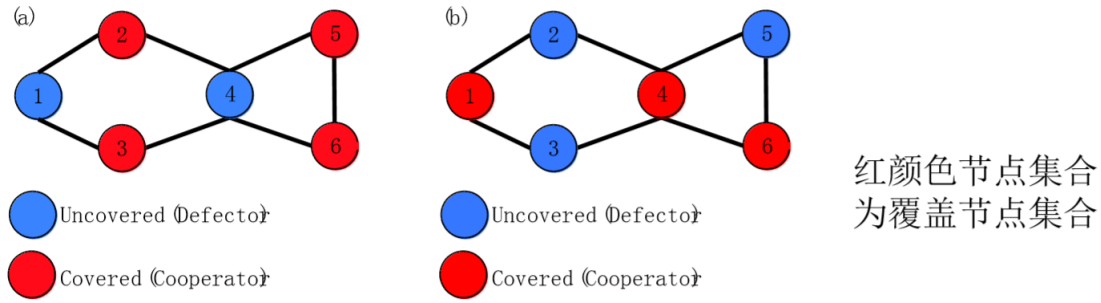
3. 重复 2 的过程到每个节点的状态不再改变

二、网络节点最小覆盖问题

网络节点最小覆盖问题：是一个著名组合优化问题，其目的在于找出给定网络的最小节点集合以覆盖所有的边。

极小节点覆盖：节点集合中去掉任何一个点，就不能覆盖网络所有边。

最小节点覆盖：极小节点覆盖中节点数目最少的



如上图所示，图（a）（b）均为极小点覆盖，但图（b）也是最小点覆盖。

三、实验问题

验证：当雪堆博弈满足 $r < \frac{1}{K_{max}}$ 时，网络博弈的纳什均衡中的采用合作策略的节点构成极小节点覆盖。

四、算法流程

1、利用 networkx 库随机生成一个网络，并对网络中的每一个节点随机选择一个策略。

2、统计当前网络的最大度，即 K_{max} 。

3、利用 K_{max}，构建雪堆博弈的收益矩阵（$r < \frac{1}{K_{max}}$）。

4、开始博弈，直到找到网络的纳什均衡，即没有人再改变自己的策略：

1）若迭代次数大于 100 次，则该网络可能没有纳什均衡，退出。

2）遍历网络中的 n 个节点，对每个节点计算选择两种策略的总收益。

3）让每个节点都选择自己收益最大的策略，并判断有无节点改变策略，无改变则退出循环，有改变则继续 1）~ 3)。

5、博弈完成时候，输出博弈前和博弈后的图像进行对比，并判断达到博弈后的网络纳什均衡中的采用合作策略的节点是否构成极小节点覆盖。

五、算法具体实现（Python）

5.0 用到的库：

其中 networkx 提供了许多网络相关的函数，借助它可以很方便的完成网络模型大搭建。注：networkx 需要搭配 matplotlib 使用。

```
import numpy as np
import random
import networkx as nx
from matplotlib import pyplot as plt
```

5.1 随机生成网络

在这里节点数量设置为 20，边的稀疏程度为 0.5（0-1 之间，越大边越多，网络越复杂）。

依次生成 20 个节点，按 0.5 的概率为每个随机选择一种策略（合作为红色、不合作为青色），每个节点都包含名称、策略、收益者三个信息。

并将这 20 个节点依次用边连接起来，并对每一个节点按一定概率（稀疏程度）与网络中随机一个节点相连。

```
node_size = 20
edge_sparse = 0.5 # 0-1
node_colors = []
# 随机生成一个网络
G = nx.Graph()
for i in range(node_size):
    # 初始化网络，随机选策略
    if random.random() > 0.5:
        choice = 0  # 选择策略为C
        node_colors.append('r')
    else:
        choice = 1  # 选择策略为D
        node_colors.append('c')

    # 添加节点
    G.add_node(i, name = i, strategy = choice, porfit = 0)

    # 随机添加边
    if i != 0:
        G.add_edge(i,i-1)
        if random.random() < edge_sparse:
            G.add_edge(i, random.randint(0,len(G.nodes)-1))
```

5.2 统计节点最大度并构建雪堆博弈收益矩阵

利用 nexwork 中的 G.degree()直接遍历每个节点的度，找出最大值。并且使得 $r = \frac{1}{K_{max}} \times 0.9$，这样就保证了 r 的范围，构建收益矩阵。

```
# 统计网络的最大度
max_degree = 0
for i in range(node_size):
    if G.degree(i) > max_degree:
        max_degree = G.degree(i)

# 显示初始网络
plt.figure("初始网络")
nx.draw_networkx(G, node_color = node_colors)
plt.show(block = False)

# 雪堆博弈收益矩阵
r = 1/max_degree * (0.9)
porfit_mat = [[(1,1),(1-r,1+r)],[(1+r,1-r),(0,0)]]
```

5.3 不断迭代博弈

迭代过程中，change 表示有无人改变策略，并对迭代次数加以限制，防止程序陷入死循环。对于每个节点，按合作合不合作两种策略，都依据收益矩阵对其的邻接节点计算收益，选择使其收益最大的策略（并同时改变节点颜色），并判断是否改变策略。

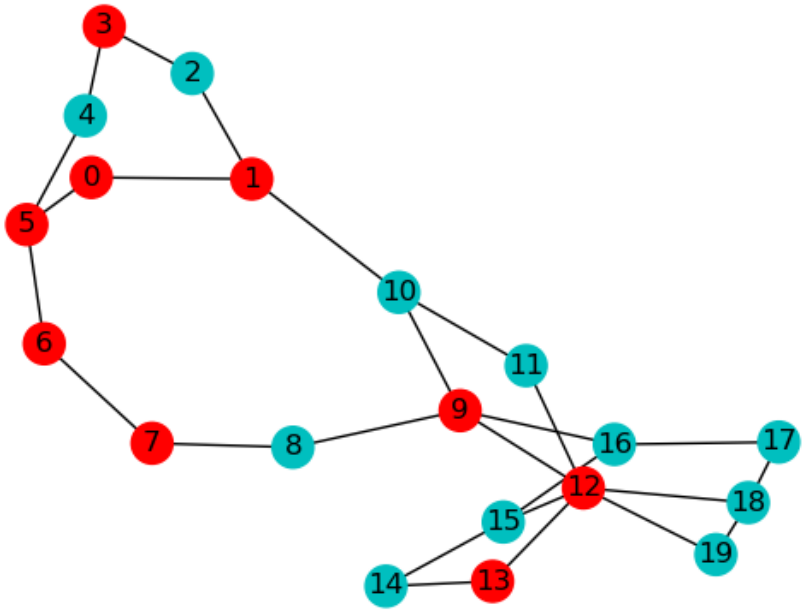
```
# 不断博弈，找到网络博弈的纳什均衡
itera = 0
change = 1
while change:
    change = 0
    itera += 1
    if itera > 100:
        print("该网络没有纳什均衡")
        exit()

    for n, nbrs in G.adjacency():   # n:遍历的第 n 个节点，nbrs：所有邻接节点
        # print("*****************",n,"*******************")
        porfit_0 = 0    # 选择 0（Cooperator）策略的收益
        porfit_1 = 0    # 选择 1（Defector）策略的收益
        for nbr,_ in nbrs.items():
            # print("n_strategy:",G.nodes[n]['strategy']," nbr_",nbr,"strategy:",G.nodes[nbr]['strategy'])
            porfit_0 += porfit_mat[0][G.nodes[nbr]['strategy']][0]
        for nbr,_ in nbrs.items():
            porfit_1 += porfit_mat[1][G.nodes[nbr]['strategy']][0]

        if porfit_0 >= porfit_1:
            if G.nodes[n]['strategy'] == 1:
                change = 1
            G.nodes[n]['strategy'] = 0
            node_colors[n] = 'r'
        else:
            if G.nodes[n]['strategy'] == 0:
                change = 1
            G.nodes[n]['strategy'] = 1
            node_colors[n] = 'c'
```

5.5 输出最终纳什均衡的图像

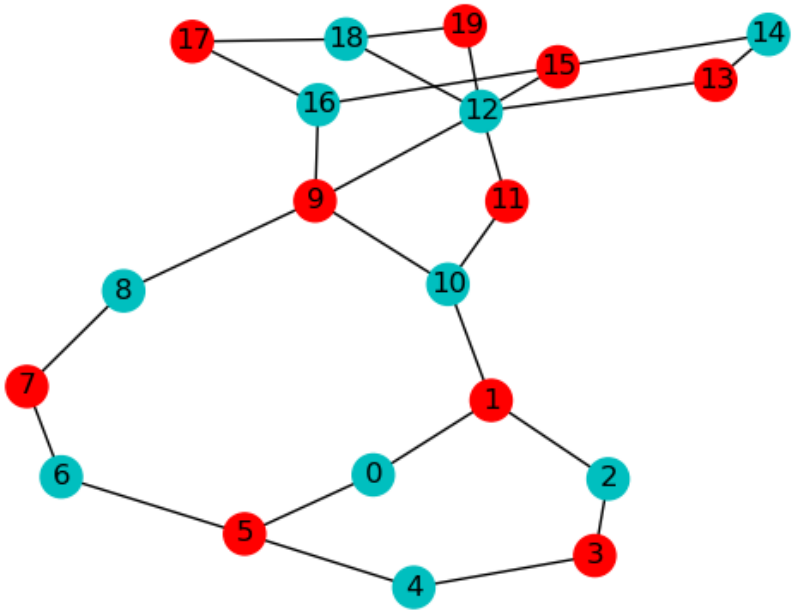
```
# 输出图像展示
print("网络的最大度：",max_degree)
plt.figure("网络博弈的纳什均衡")
nx.draw_networkx(G, node_color = node_colors)
plt.show()
```

六、结果展示与分析

初始随机生成的网络：



博弈后已经纳什均衡的网络：



从上图可以看出，网络的最大度为 6，在 12 节点上，其中选择合作的为红色节点，经过博弈后红色的节点已经构成了极小点覆盖，即去掉红色的任何一个点，红色节点就不能覆盖网络所有边。验证了结论：

当雪堆博弈满足 $r < \frac{1}{K_{max}}$ 时，网络博弈的纳什均衡中的采用合作策略的节点构成极小节点覆盖。

注：由于 networkx 库的原因，两次输出的网络形状不同，但可以验证是同一个网络。

七、心得体会

这次作业的完成，最大的收获是了解了 networkx 这个库，并收悉了其基本用法，可能对之后图的相关学习和编程实现起到很大作用。其次，又掌握了如何求解多人博弈的纳什均衡问题、极小点覆盖问题，也对雪堆博弈的这个特性有了一定了解。