Spread of Misinformation Online

Team members: Bochao Wei, Ge Ren, Beichen Liang, Xiaoxu Zou

Git Repository: **https://github.gatech.edu/bwei39/misinfo-CSE6730**

I. Abstract

The spread of information has been unprecedentedly fast in the past few years thanks to the popularity of social media. A random piece of user-generated content can become viral on the internet and reach people around the world within an hour. However, the internet also facilitates the dissemination of unverified rumors and misinformation. Human progress by collectively creating, filtering, and sharing what we know, and the fact that false information can spread faster than true ones is a repercussion of our collective decision process.

How is the rumor spread? It is common to see some kinds of misinformation maintain high popularity within groups that share similar interests or beliefs. Do ideological groups matter? People may become less interested in a piece of information as time goes on. Is the spread time-sensitive? Due to the widespread fake news on the Internet, it may affect the reputation of individuals, influence public opinion, and even negatively affect the entire society. We try to simulate the dissemination process of fake news by investigating how fake news spreads on the Internet. The focus of the investigation is to use an SBFC (susceptible, believer, fact-checker) model based on the variant of the SIR (susceptible, infected, recovered) model. We also introduce some realistic extensions to realistically simulate the dissemination process of false information on the Internet.

II. Project description

The focus of the investigation is to use an SBFC (susceptible, believer, fact-checker) model based on the variant of the SIR (susceptible, infected, recovered) model. We also introduce some realistic extensions to realistically simulate the dissemination process of false information on the Internet.

The structure of online social networks, the underlying topology induced by connections between users, is the key to amplifying the spreading of fake news. For different users in social networks, we distinguish their roles and characteristics in social networks by distinguishing common users, influencers, and bots. The existence of "influencers" such as celebrities or public figures greatly accelerates the dissemination of fake news. In addition to influencers, bots are autonomous agents spreading fake news and aiming to influence user behavior. For different types of agents, we set different parameters separately, according to statistical distributions in order to create heterogeneity. For the construction of the network, we also consider geographic user clusters and clusters of users with common interests, which makes the network closer to the real social network.

For the simulation of fake news dissemination, we refer to the epidemic model, which is a feasible and commonly used tool for studying the dissemination of information in online social networks. In this investigation, the susceptible, believer, fact-checker (SBFC) model we consider is a variant of the SIR model. In the dissemination of fake news, believers simulate people in social networks who believe in fake news, while fact-checkers are people in social networks who believe that the news spread is fake. We assume that an agent is easily deceived by fake news initially. When a node associates with a believer, it may become a believer; and when it associates with a fact-checker, it may also become a fact-checker.

III. Literature review

In reference [1], the availability of user-provided information aggregates people around common interests and worldviews. The authors conducted a quantitative analysis of Facebook usage modeling the propagation with an oriented tree to find out homogeneity as the primary driver for content diffusion and each of the homogeneous clusters has its own cascade dynamics. Users tend to share content according to a specific narrative and ignore the rest. A branching percolation model is later introduced to account for homogeneity and polarization. The key parameters are the threshold and the neighborhood dimension. The result shows the news tends to remain inside and is transmitted solely within the homogeneous clusters.

In reference [2] claims that lies spread faster than truth. Both true and false information spread rapidly online. An individual could start a rumor cascade by tweeting a story and another individual could not only retweet it but also independently start a new cascade of the same rumor. The authors examine the problem by investigating the differential diffusion of true, false, and mixed news using a comprehensive data set of Twitter from 2006 to 2017. They found that falsehood diffused significantly faster, farther, and deeper than truth in all categories of information, considering comprehensive aspects including the novelty of fake news, the presence of bots, and the selection bias of the data.

In reference [3], Based on the similarity between epidemics and fake news spreading, they fill this gap by enriching typical epidemic models for fake news spreading with network topologies and dynamics that are typical of realistic social networks. These aspects include the different connectedness of agents in an OSN (e.g., bots, influencers, and common agents), the presence of fact-checkers, the spreading of debunking information that reduces the engagement of fake news, and the time-varying aspects of OSN user activity. They propose an agent-based simulation model by considering two adversary spreading processes, one for fake news and a second one for debunking. They differentiate among agents' roles and characteristics, by distinguishing among common users, influencers, and bots. They propose a network construction algorithm that favors the emergence both of geographical user clusters and of clusters of users with common interests and they account for the presence of influencers and bots and for unequal trust among agents. they proposed ways to add time dynamical effects in fake news epidemic models

In reference [4], this paper introduced an agent-based framework (developed in NetLogo, one of the most relevant simulation platforms) to simulate the spread of a piece of misinformation, where fake news and its debunking compete on social networks. The tool references a known model based on the spread of epidemics, in which misinformation and fact-checking compete against a group of agents who can or cannot believe in hoaxing, verifying, or forgetting news. The parameters of the model are the spread rate, the credibility of the hoaxing, the probability of forgetting, the probability of validation, the population size, and the initial seed (the number of believers at time t = 0). They provide a tool to run the model on any desired topology to adjust the values of all parameters and the results of our implementation in NetLogo match those obtained from the platform's simulation and analytical calculations. On the other hand, NetLogo offers the possibility to run a personalized version of the model in a very intuitive framework, which opens the way for improved scene analysis and simulation dynamics.

In reference [5], they focus on analyzing how long a prank lives in the network, that is, how long users think it is real. Hoaxes are very similar to ordinary viruses: people, as nodes in a social network, may "infect" and believe fake news after coming into contact with other infected nodes, or "recover" through simple fact-checking operations. Epidemiological models can describe many other phenomena, such as social contagion, information dissemination, and computer virus attacks. They consider partitioned epidemic models such as SIR (Susceptible-Infected-Recovered) and SIS (Susceptible-Infected-Susceptible), where nodes are characterized by different behaviors represented by states, and the dynamic evolution of the system is determined by the transition rates between states. They propose a stochastic epidemic model to describe the simultaneous spread of a hoax and its associated debunking: it can be viewed as an SIS model in which the infection state is divided into two sub-compartments, believers (B) and fact-checkers (F), while the transition $I \rightarrow S$ can be interpreted as a forgetting process. Furthermore, they have a fixed probability p_verify transition $B \rightarrow F$, which represents the proportion of infected users who check the reliability of the received information, thus revealing a scam. They compare our model with SIS for the lack of prevalence thresholds in scale-free networks. Through analysis, they found a threshold for this probability, a sufficient condition to ensure that the hoax will be eliminated. It provides an idea of how many fact-checkers are enough to warrant the complete removal of fake news. The behavioral similarity of our model to traditional SIS and SIR epidemic models was confirmed.

In reference [6], the authors point out that current literature on combating misinformation focuses on individuals and neglects social newsgroups -key players in the dissemination of information online. Using benchmark variables and values from the literature, the authors simulated the process using Biolayout; a big data-modeling tool. The results show social newsgroups have a significant impact on the explosion of misinformation as well as on combating misinformation. The outcome has helped me better understand and visualize how misinformation travels in the spatial space of social media.

After reading these papers, the mainstream approach is using graph theory-based models and designing different rules and node structures to simulate the spread of misinformation. We decided to also adopt this graph theory approach because it is very realistic and flexible. **The new things** we are going to do is by changing the node structures and rules set to study:

1. the segregation of beliefs in the misinformation and its interplay with Ideological groups
2. With time decay and people are losing interest then fewer people spread this misformation
3. Geographical clustering spreading speed vs influencer spreading speed

IV. Conceptual Model of the System

We consider three types of agents: commons, influencers, and bots. The nodes that exhibit high out-degree are the influencers. The bots are automated accounts constantly echoing the fake news and unchangeable. The distribution for our agents is the normal distribution with a specific mean and standard deviation based on different characteristics.

The social network we modeled as a graph. Each node represents the agents defined above and edges convey the relationship between two endpoints. Geographical proximity, attribute proximity, and randomness are taken into consideration for the network generation. Besides that, there are three parameters for each node (the vulnerability, the sharing rate, and recovery rate)

Moreover, the weights are assigned to edges which signify the trust between each agent. This allows us to assign lower influence power to bots instead of a human.

Also, there are two-time dynamical aspects in our simulation: the connection of an agent and the engagement of an agent. To model the typical loss of interest in the news over time, we assume it follows an exponential interest decay as time goes by. In our setting, a user who believes in fake news will be likely attracted to the same piece of news and the debunking message is a 0.1 probability of changing a state of a user from believer to fact-checker with respect to the probability of transitioning from susceptible to the believer.

Regarding the spreading process, we followed the example of the SIR epidemic model with three states: susceptible, believer, and fact-checker. Three processes are in between: Fact-checking, fake news spreading, and debunking. Status is transferable during these three processes but once someone becomes a fact-checker then he/she will not be gullied anymore and status won’t change.

The rough process of our process is:
Basic information scan for whether it's a fact-checker already or a bot not. After we make sure our node is still at the changeable stage then more factors like receiving time, vulnerability, trust, and the truth of news are taken into consideration. At the same time, new information is brought into our current node with every time iteratively simulation and keep going.

V. Spread of Misinformation Model

i. Build Network

We created a node class and a network class using python to create our simulation platform. For each node, it has its id, node_type (normal people or influencers or bots), and its position. It has two-position coordinates x and y that are both uniformly distributed in [0,1]. The position can indicate the 'distance' of two people on the website and people with similar interests or beliefs are assumed to have a closer distance thus a higher chance to be connected. Each node also has a list to store all the connections it makes to other nodes, and each connection has an associated weight to represent the trust between them. We then used the python package networkx to visualize the created network. Some codes are listed below:

```
import matplotlib.pyplot as plt
import networkx as nx
from enum import IntEnum
import random
import math
from math import exp
import numpy as np



import math
from queue import PriorityQueue
from random import expovariate, shuffle, uniform, randint
import copy
from matplotlib.backends.qt_compat import QtCore, QtWidgets
from matplotlib.backends.backend_qt5agg import (FigureCanvas, NavigationToolbar2QT as NavigationToolbar)
from functools import partial

class NodeType(IntEnum):
    Common=1,
    Bot=2,
    Influencer=3
def norm_sample(avg=0.0, var=0.4, clip1=-1, clip2=1, n=None):
    norm_vals = np.random.normal(avg, var, n)  # sample from a normal distribution
    return np.clip(norm_vals, clip1, clip2)  # limit the results into [0, 1]

def euclidean_distance(a, b):
    return np.linalg.norm(a - b)

class Edge:
    def __init__(self, start, dest, weight):
        self.start = start
        self.dest = dest
        self.weight = weight
class Node:
    def __init__(self,ids,node_type):
        self.id=ids
        self.type=node_type
```

```
    self.adj = []
    #self.score = np.random.uniform(-1, 1) #####define the node's political stand left or right, or their tendency in beliveving sth
    self.score=0
    self.position_x = np.random.uniform(0, 1)  #######position of x and y define people's abstract distance
    self.position_y = np.random.uniform(0, 1)
    self.message_queue = [] ###use to store information or effect on the node
    self.vulnerability = norm_sample(avg=0.5, var=0.1, clip1=0, clip2=1)
    self.reshare_rate = norm_sample(avg=0.5, var=0.1, clip1=0, clip2=1)
    self.recover_rate = norm_sample(avg=0.2, var=0.1, clip1=0, clip2=1)
    self.recovery_time = None
    self.infection_time = None
    if node_type == NodeType.Bot:
        self.infection_time = 0
def add_adj(self,edge):
    self.adj.append(edge)
def compute_distance(self,node_b):
    a = np.array([self.position_x, self.position_y])
    b = np.array([node_b.position_x, node_b.position_y])
    return euclidean_distance(a, b)/np.sqrt(2)
def update(self):
    s = sum(self.message_queue)
    l = len(self.message_queue)
    self.message_queue = []
    if l > 0:
        avg = s / l
        self.score += avg
        if self.score > 1:
            self.score = 1
        elif self.score < - 1:
            self.score = -1
        return True
    else:
        return False
def update_sir(self, engagement_news):
    number_of_messages = len(self.message_queue)
    if self.type == NodeType.Bot:
        return

    if number_of_messages != 0 and self.score != -1:
        can_fact_check = False

        for i in range(number_of_messages):
            p = random.uniform(0, 1)
            message_type, weight = self.message_queue[i]

            if message_type == -1:
                k = 0.1
            else:
                k = 1
                can_fact_check = True

            if p < self.vulnerability * k * weight * engagement_news:
                self.score = message_type
                can_fact_check = False
```

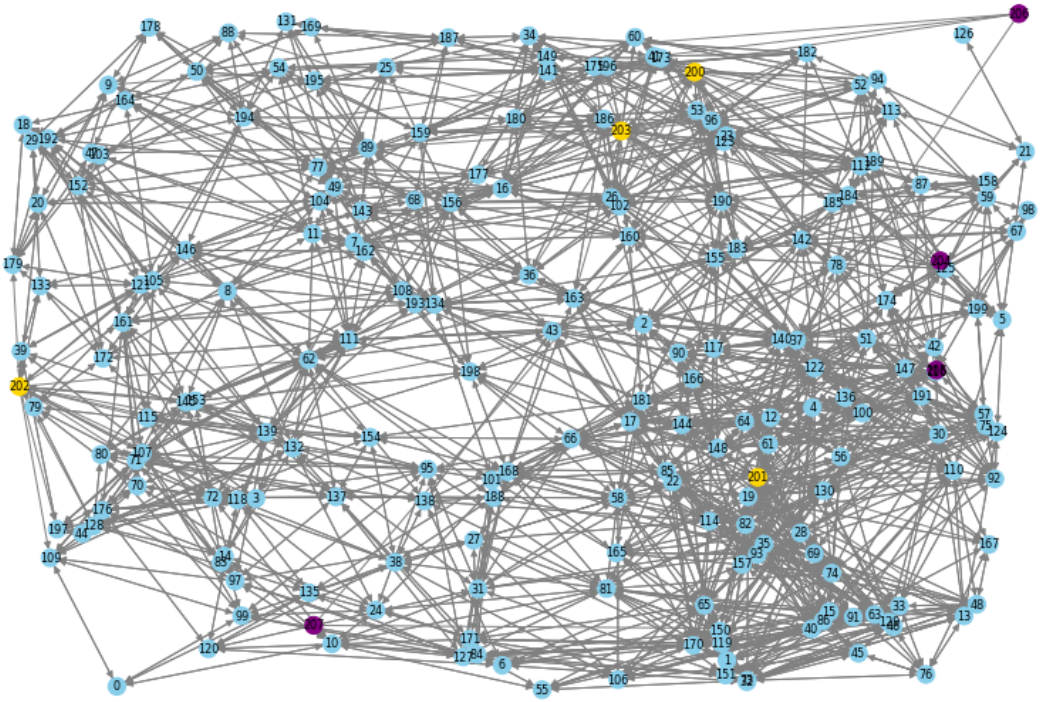
```
            if can_fact_check and self.score == 0:
                self.is_recovered()

        self.message_queue = []
    def __str__(self):
        return '{} - deg: {}\n' \
               '\tinterests: {}\n' \
               '\tscore: {}\n' \
               '\tx: {} y: {}'.format(
                self.id, len(self.adj), self.score,
                self.position_x, self.position_y)
class Network:
    def __init__(self,N_common, N_influencers, N_bots, random_const):
        self.N_common=N_common
        self.N_influencers=N_influencers
        self.N_bots=N_bots
        self.nodes={}
        self.available_id=0
        self.infected_node=None
    def gen_node(self,node_type):
        idx=self.available_id
        node=Node(idx,node_type)
        self.available_id+=1
        self.nodes[node.id]=node
        if node_type==NodeType.Influencer: ####define node property
            pass
        if node_type==NodeType.Bot: ####define node property
            node.reshare_rate = 1
        return idx
    def generate_common(self,random_const):
        def add_proximity_edge(idx_a, idx_b, dist, random_const):
            prox = (1 - dist)
            #edge = list(filter(lambda x: x.dest == idx_b, self.nodes[idx_a].adj))
            p = random.uniform(0, 1)
            if dist < random_const and p < 0.3:  ##people need to have close enough distance to be connected and a finite probability
                weight = prox
                self.nodes[idx_a].add_adj(Edge(idx_a, idx_b, weight)) ##normal people's connection are mutual
                self.nodes[idx_b].add_adj(Edge(idx_b, idx_a, weight))
        n = 0
        while n < self.N_common:
            idx = self.gen_node(NodeType.Common)
            for b in self.nodes.keys():
                if idx == b:
                    continue
                phys_dist = self.nodes[idx].compute_distance(self.nodes[b])
                add_proximity_edge(idx, b, phys_dist, random_const)
            n += 1
    def generate_influencers(self, random_const,lim=None):
        def add_proximity_edge(idx_a, idx_b, dist, random_const):
            prox = (1 - dist)
            p = random.uniform(0, 1)
            if dist < random_const and p < 0.7:  ##people need to have close enough distance to be connected, and p is higher than common
                weight = prox
```

```
            self.nodes[idx_a].add_adj(Edge(idx_a, idx_b, weight))###influencers connection is asymmetric
        if dist<random_const and p<0.2:
            self.nodes[idx_b].add_adj(Edge(idx_b, idx_a, weight*0.5)) ###fewer chance to be affected by common

    n = 0
    if lim==None:
        lim=self.N_influencers
    while n < lim:
        idx = self.gen_node(NodeType.Influencer)
        for b in self.nodes.keys():
            if idx == b:
                continue
            phys_dist = self.nodes[idx].compute_distance(self.nodes[b])
            add_proximity_edge(idx, b, phys_dist, random_const)
        n += 1

def generate_bots(self,random_const,lim=None):
    n=0
    if lim==None:
        lim=self.N_bots
    while n < lim:
        idx = self.gen_node(NodeType.Bot)
        self.nodes[idx].score = np.random.choice([-1,1])  ###now assume bots are fully biased
        for b in self.nodes.keys():
            if self.nodes[b].type == NodeType.Bot:
                continue
            p = random.uniform(0, 1)
            if p < 0.02:
                weight = 0.1
                self.nodes[idx].add_adj(Edge(idx, b, weight)) ####the bot only affects others
                #self.nodes[idx].add_adj(Edge(b, idx, weight))
        n += 1
def get_nodes_infection_time_map(self, max_time):
    nodes = {}

    for i, n in self.nodes.items():
        nodes[i] = n.get_infection_time(max_time)

    return nodes

def get_nodes_recovery_time_map(self, max_time):
    nodes = {}

    for i, n in self.nodes.items():
        nodes[i] = n.get_recovery_time(max_time)

    return nodes
def average_score(self):
    tot = 0
    for n in self.nodes:
        tot += self.nodes[n].score
    return tot / len(self.nodes)

def average_weight(self):
```

```
        tot = 0
        for n in self.nodes:
            weights = list(map(lambda edge: edge.weight, self.nodes[n].adj))
            if len(weights) == 0:
                tot += 0
            else:
                tot += sum(weights) / len(weights)
        return tot / len(self.nodes)

    def count_score_equal(self, low, up):
        return len(list(filter(lambda n: low < self.nodes[n].score <= up, self.nodes)))

    def count_node_type(self, node_type):
        return len(list(filter(lambda n: n[1].type == node_type, self.nodes.items())))
```

And a figure of an initial network with 200 common nodes, 4 influencer and 4 bots with connections are coded and plotted below. With influencers colored gold and bots colored purple.

```
node_color = {
        NodeType.Common: "skyblue",
        NodeType.Influencer: "gold",
        NodeType.Bot: "purple"
    }
def draw_initial_network(network):
        figure = plt.figure(figsize=(12, 8))
        G = nx.DiGraph()
        pos = {}
        color_map = []
        for n, node in network.nodes.items():
            G.add_node(n)
            pos[n] = [node.position_x, node.position_y]
            color_map.append(node_color[node.type])

        for a, node in network.nodes.items():
            for b in node.adj:
                G.add_edge(a, b.dest)
        nx.draw(G, pos=pos, with_labels=True, font_size=8, node_size=150, node_color=color_map, edge_color="grey")
```



While the common nodes represent the normal people online, bots nodes represent the robot online built or managed by political parties or groups who try to influence public opinion to achieve their goals. Nodes with the class of bots are assumed to be fully biased and they can only infect others. The influencer nodes, on the other hand, represent famous Youtubers, sports or music stars, famous politicians, etc. Nodes of the class of influencers have more connections to common nodes representing their influence online and their ability to spread misinformation quickly.

As people don't trust one another in the same way, common people tend to place less trust in bots and more trust in influence. Accordingly, when generating a network, we specify the connection between influencers and common nodes to be asymmetric with the influencers having a higher probability (60%) of influencing common nodes while a lower chance (10%) of being affected by them. Edges connecting bots and other nodes are defined to be one-directional with low probability meaning that bots will try to affect others but will not be affected by the state of other nodes. With four bots in plum color, four influencers in red, and a hundred common nodes in blue, the network created at one run can be visualized as the following graph.

ii. Run Simulation

For the simulation part, except for adding two special types of agent influencers and bots, a priority queue as a data structure is being used to track the event status that happened. First, the first population queue and initial infection group are initialized. Based on whether time decay is taken into consideration, the exponential function is decided to be used for time. When it comes to simulating the online social network, the example model of the spreading of a pandemic --“suspected infected recovered” model is being transferred to our “suspected, believer, fact-checker” model. The score is being used for keeping track of the status of each individual. During the propagation, the closer the distance, the more engagement of news, or the heavier the weight, the more tendency towards the status of each individual tracked. While the running process of the simulation, the engagement of agent nodes is decreasing with an exponential function. A list named n_sim_results appends also the results.

Codes are shown below:

```
class Simulator:
    def __init__(self, N_common, N_influencers, N_bots, random_const,
                 engagement_news=None):

        self.N_common = N_common
        self.N_influencers = N_influencers
        self.N_bots = N_bots

        self.random_const = random_const

        self.engagement_news = engagement_news
        self.network = Network(N_common=N_common, N_influencers=N_influencers, N_bots=N_bots,
                               random_const=random_const)

        self.network.generate_common(self.random_const)
        self.N = len(self.network.nodes)
        self.sim_network = None
    def add_influencers(self, n=None):
        if n is None:
            n = self.N_influencers
        self.network.generate_influencers(self.random_const, n)
        self.N = len(self.network.nodes)
    def add_bots(self, n=None):
        if n is None:
            n = self.N_bots
        self.network.generate_bots(n)
        self.N = len(self.network.nodes)
    def first_population_queue(self, first_infect, with_temp_dyn):
        order = []
        self.events_queue.put((0, first_infect))
```

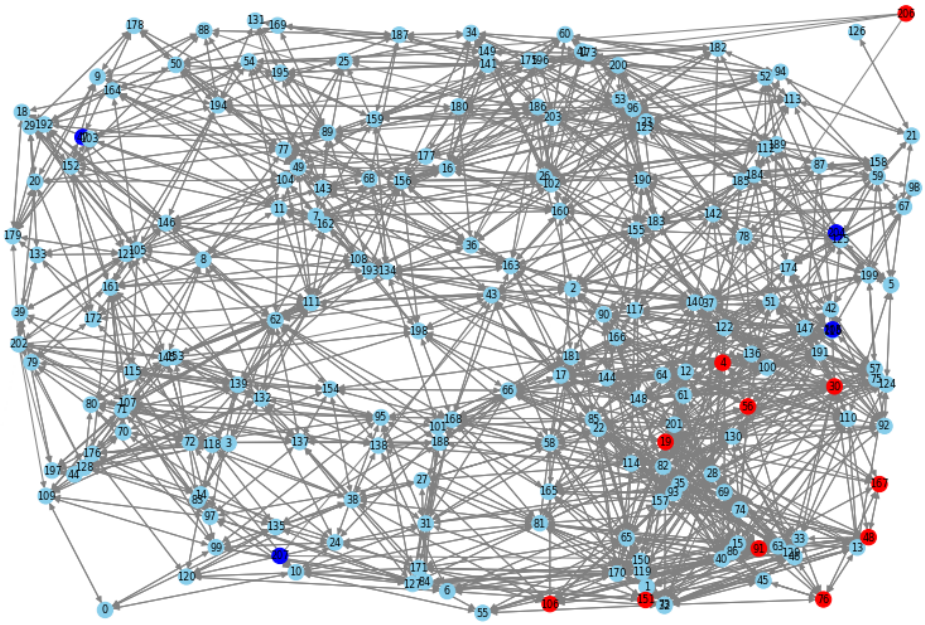
```
    for i in range(self.N):
        if i != first_infect:
            order.append(i)
    shuffle(order)
    for i in range(self.N - 1):
        next_time = expovariate(1 / 16) if with_temp_dyn else 16 ### a exponential decay next_time, for a user to check news
        self.events_queue.put((next_time, order[i]))  ######put each node with different time in the queue

def initial_infection(self, first_infect):
    if first_infect is None:
        first_infect = randint(0, self.N_common - 1)
    self.sim_network.nodes[first_infect].score = np.random.choice([-1,1])
    #self.sim_network.nodes[first_infect].reshare_rate = 1
    #self.sim_network.nodes[first_infect].recover_rate = 0
    return first_infect  ###index of the infected node
def simulate(self, max_time, recovered_debunking=False, SIR=False, first_infect=None, return_nets=True,
             weighted=True, with_temp_dyn=True):
    self.events_queue = PriorityQueue()   #######the event will happen through this
    self.sim_network = copy.deepcopy(self.network)
    first_infect = self.initial_infection(first_infect) ###create a first infection
    self.sim_network.infected_node = first_infect
    self.first_population_queue(first_infect, with_temp_dyn) ######fill the event queue with nodes

    hist_status = []
    # Add simulation checkpoint
    for i in range(0, max_time, 20):
        self.events_queue.put((i, -1))

    time = 0
    s = []
    i = []
    r = []

    cdf_i = [[0] * len(self.sim_network.nodes)]
    cdf_i[0][first_infect] = 1

    while time < max_time:
        t, node_id = self.events_queue.get()

        time = t

        if node_id == -1:  # checkpoint
            if not return_nets:
                S = 0
                I = 0
                R = 0
                cdf_i.append(copy.deepcopy(cdf_i[-1]))  ###dont know why
                for k in self.sim_network.nodes.keys():
                    if self.sim_network.nodes[k].type == NodeType.Common:
                        if self.sim_network.nodes[k].score == 1:
                            I += 1
                            cdf_i[-1][k] = 1
                        elif self.sim_network.nodes[k].score == 0:
                            S += 1
```

```
                    else:
                        R += 1
                s.append(S)
                i.append(I)
                r.append(R)
            else:
                hist_status.append((time, copy.deepcopy(self.sim_network)))
            continue

        if not SIR:
            self.sim_network.nodes[node_id].update()
        else:
            self.sim_network.nodes[node_id].update_sir(self.engagement_news(time))

        score = self.sim_network.nodes[node_id].score

        if self.sim_network.nodes[node_id].infection_time is None and score == 1:
            self.sim_network.nodes[node_id].infection_time = time
        elif self.sim_network.nodes[node_id].recovery_time is None and score == -1:
            self.sim_network.nodes[node_id].recovery_time = time

        if score == 1 or (score == -1 and recovered_debunking):
            reshare_rate = self.sim_network.nodes[node_id].reshare_rate
            for edge in self.sim_network.nodes[node_id].adj:
                p = uniform(0, 1)
                dest = edge.dest
                weight = edge.weight if weighted else 1
                if p < reshare_rate and dest != first_infect:
                    self.propagate(dest, score, weight, SIR=SIR)

        if with_temp_dyn:
            # se un nodo è un bot, allora si collega più spesso
            if self.sim_network.nodes[node_id].type == NodeType.Bot:
                self.events_queue.put((time + expovariate(1 / 4), node_id))
            else:
                self.events_queue.put((time + expovariate(1 / 16), node_id))
        else:
            self.events_queue.put((time + 16, node_id))

    if return_nets:
        return hist_status
    else:
        cdf_i = list(map(lambda x: sum(x), cdf_i))
        return (s, i, r, cdf_i), self.sim_network.get_nodes_infection_time_map(max_time), \
                self.sim_network.get_nodes_recovery_time_map(max_time)

def propagate(self, dest, score, weight, SIR):
    if SIR:
        self.sim_network.nodes[dest].message_queue.append((score, weight))
    else:
        if score > 0:
            En = self.engagement_news
        else:
            En = self.engagement_news * 0.5
```

```
            message = En * score * weight
            if message < - 1:
                message = -1
            elif message > 1:
                message = 1
            self.sim_network.nodes[dest].message_queue.append(message)
```

Below shows the codes and graph after 60 times of simulation.

```
def draw_simulation_network(network):
    figure = plt.figure(figsize=(12, 8))
    G = nx.DiGraph()
    pos = {}
    color_map = []
    for n, node in network.nodes.items():
        G.add_node(n)
        pos[n] = [node.position_x, node.position_y]
        if 0.5 > node.score > -0.5:
            color_map.append("skyblue")
        elif node.score > 0.5:
            color_map.append("red")
        else:
            color_map.append("blue")
    for a, node in network.nodes.items():
        for b in node.adj:
            G.add_edge(a, b.dest)
    nx.draw(G, pos=pos, with_labels=True, font_size=8, node_size=150, node_color=color_map, edge_color="grey")
sim_results=simu.simulate(60)   ##will return the history of network at every 20 iteration
draw_simulation_network(sim_results[-1][1]) ###get the last network after simulation
```



VI. Experimental results and validation

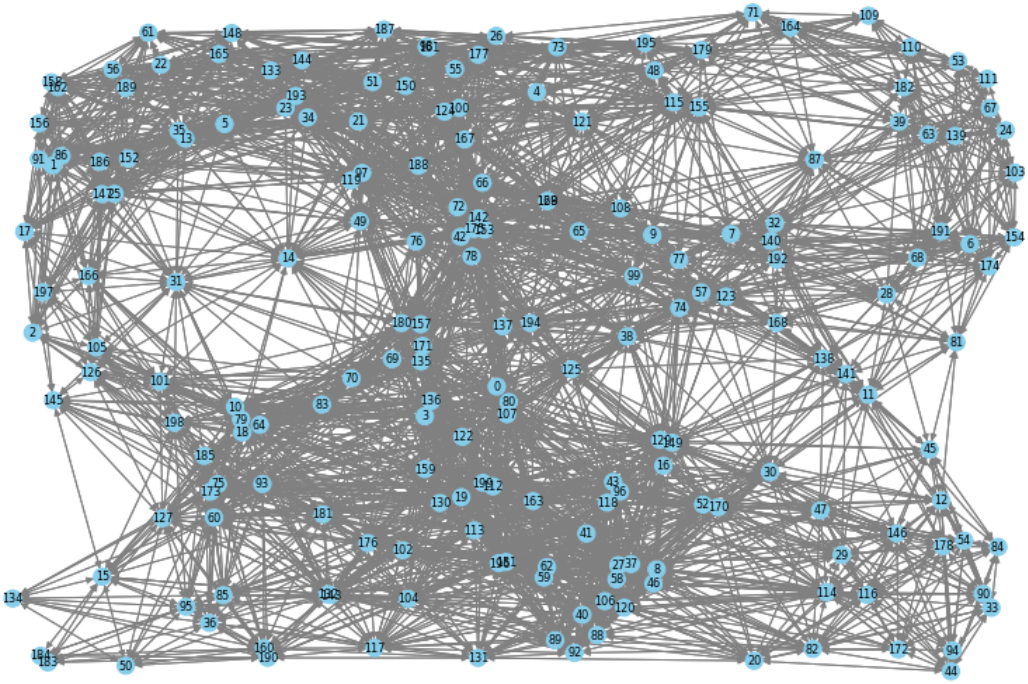
i. Idea (1): The most effective strategy to spread a piece of information: propagation with influencers?bots?hybrid? or location clusters?

Experiment:

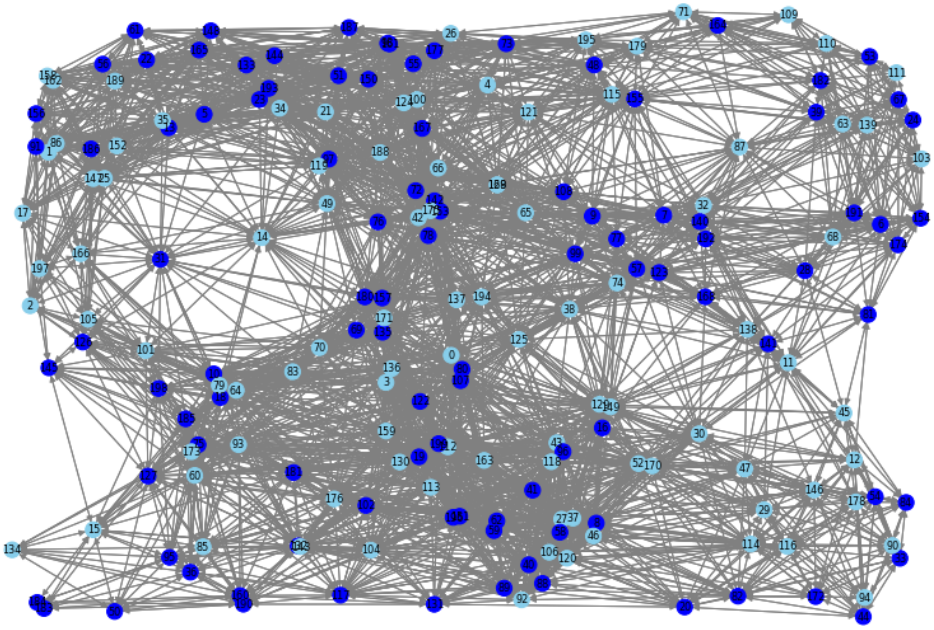
Assuming we, the evils, are trying to spread a piece of misinformation, represented by a state score of 1, among a community of 2000 people leaning towards not believing it, where the common nodes state are distributed uniformly on the other half of the axis by self.score=np.random.uniform(-1, 0). We have a limited budget and a tight deadline, and our goal is to convince as many people as possible quickly within the community. There are five viable ways to do that.

First, one of us, with a score of 1, sneaks into their social network and tries to influence others with the misinformation. Based on our literature review, we know that people tend to believe the ideas they agree with. Thus, when modeling the success rate of the influence between common people, we proposed a probability of 1 - (abs(self.nodes[idx_a].score - self.nodes[idx_b].score) * 0.5), meaning people with opposing views have a smaller possibility of affecting the other party.

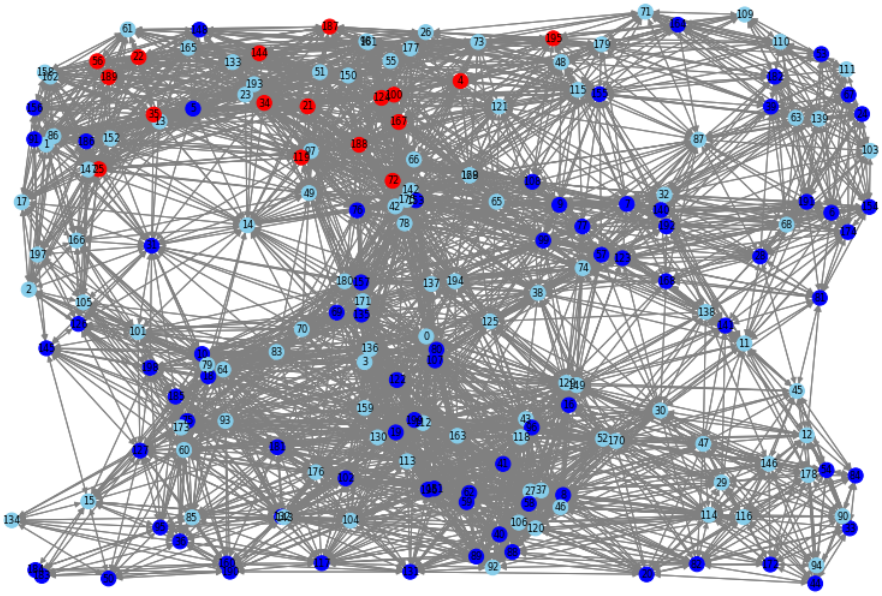
A reduced state graph of the initial network with 200 nodes is shown here. Common nodes are represented by the color of sky blue.



Initial score of the network. n.score > 0.5 is labeled red. Neutral, with -0.5 < n.score < 0.5 is labeled in light blue. Still not believing is labeled in blue.

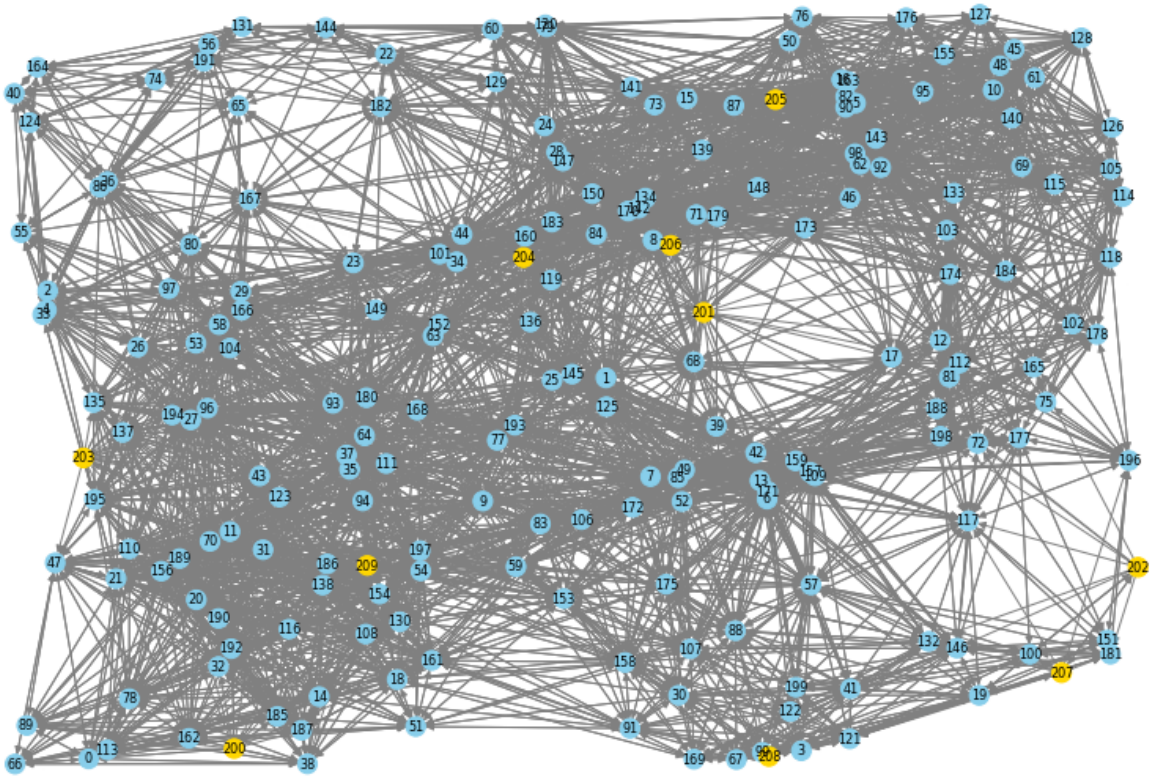


After 60 iterations, the number of people who tend to believe the information, with n.score > 0.5 is labeled red. Neutral, with -0.5 < n.score < 0.5 is labeled in light blue. Still not believing is labeled in blue.

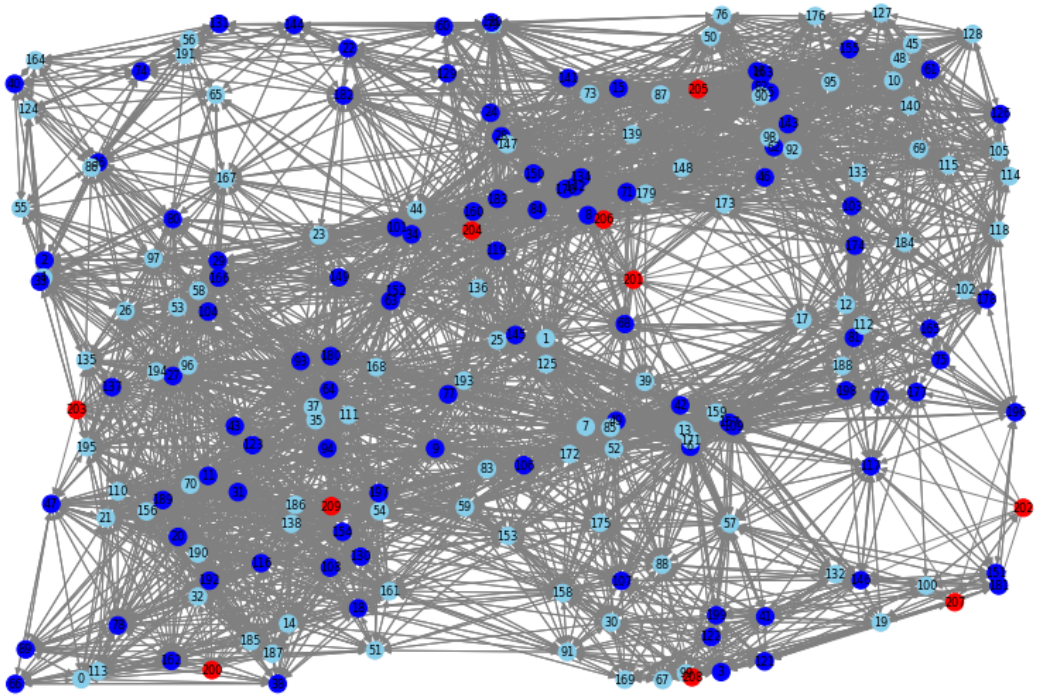


The second approach is to hire influencers to help us spread the information. The max number of influencers we can hire is 10. Assuming the influencers we hire are fully loyal to us, thus having a score of 1. Further, based on the previous theory, we set the influencer's probability of influencing common people to be 1 - (abs(self.nodes[idx_a].score - self.nodes[idx_b].score) * 0.2) and the follower's probability of reverse influencing to be 1 - (abs(self.nodes[idx_a].score - self.nodes[idx_b].score) * 0.7) with smaller weight.

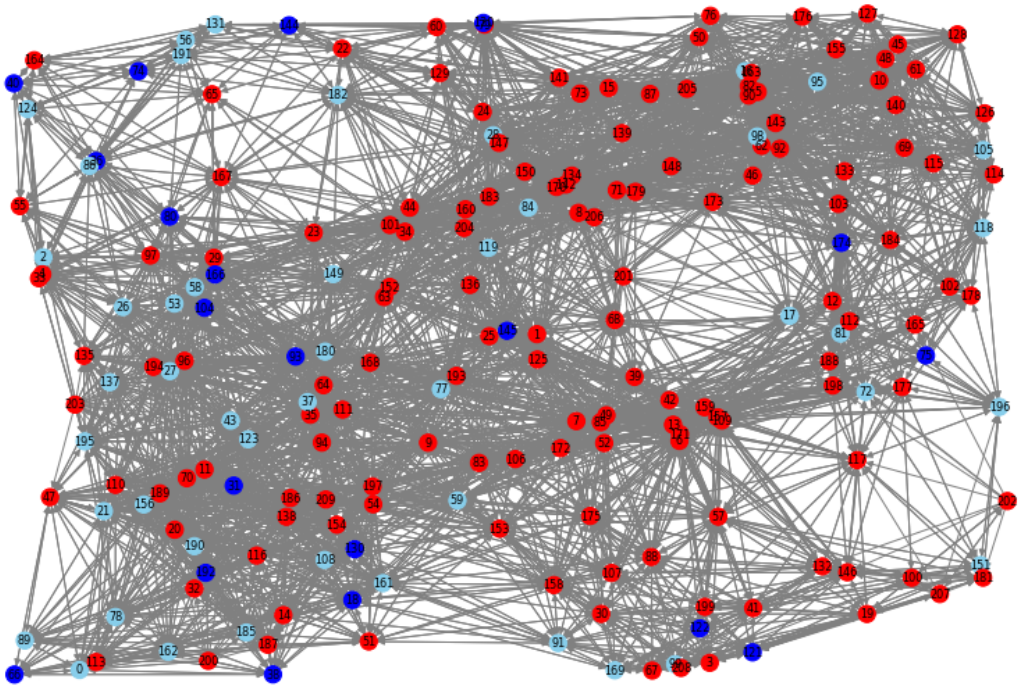
A reduced state graph of the initial network with 200 nodes is shown here. Common nodes are represented by the color of sky blue and influencers are in gold.



Initial score of the network. n.score > 0.5 is labeled red. Neutral, with -0.5 < n.score < 0.5 is labeled in light blue. Still not believing is labeled in blue.

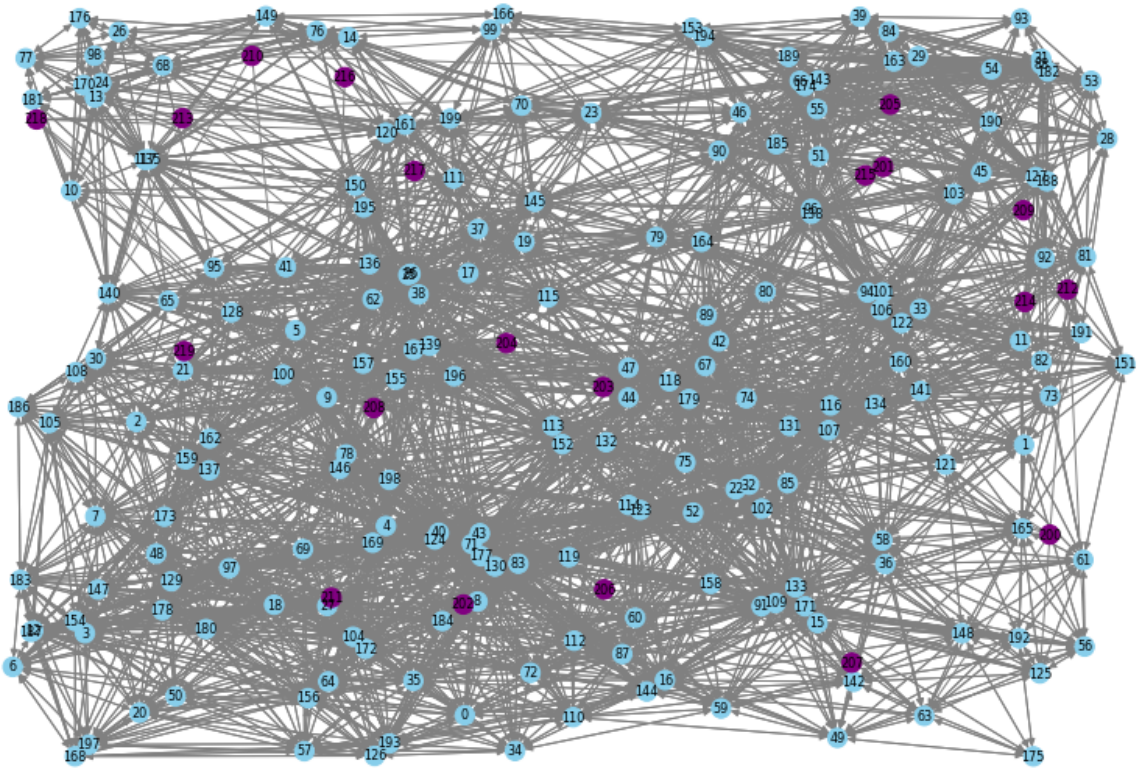


After 60 iterations, the number of people who tend to believe the information, with n.score > 0.5 is labeled red. Neutral, with -0.5 < n.score < 0.5 is labeled in light blue. Still not believing is labeled in blue.

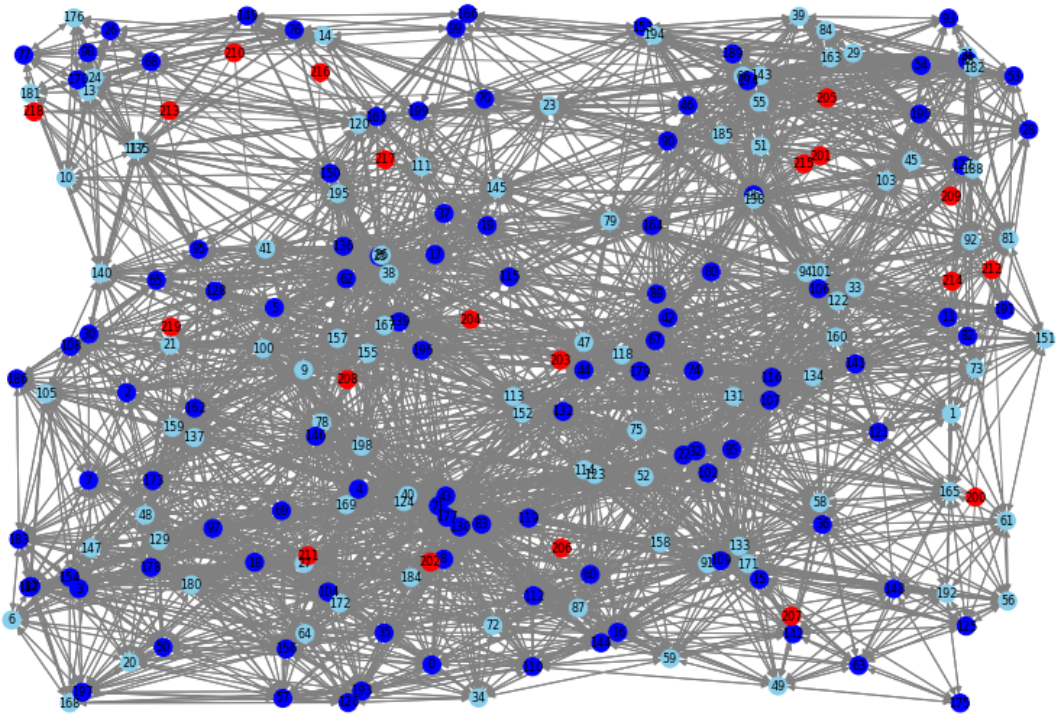


Another common technique is to use bots to influence others' opinions. The max number of bots we can put into use is 200. Assuming the bots are infected immediately and fully biased, having a score of 1 and resharing time of 0. Bots cannot be infected by other nodes and have a smaller rate of connection and smaller weight of influence.

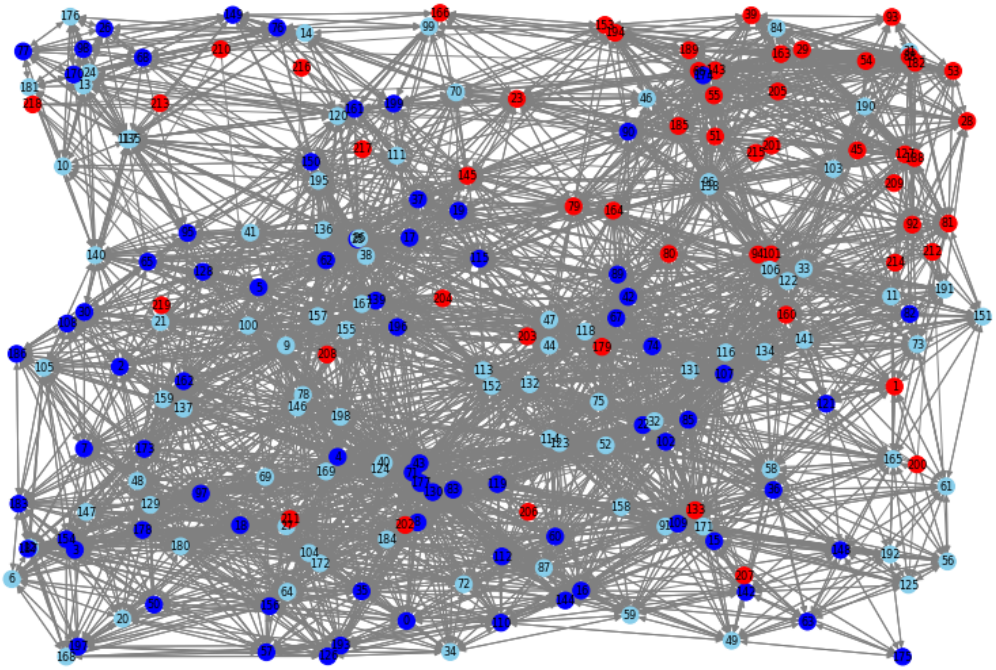
A reduced state graph of the initial network with 200 nodes is shown here. Common nodes are represented by the color of sky blue and bots are in purple.



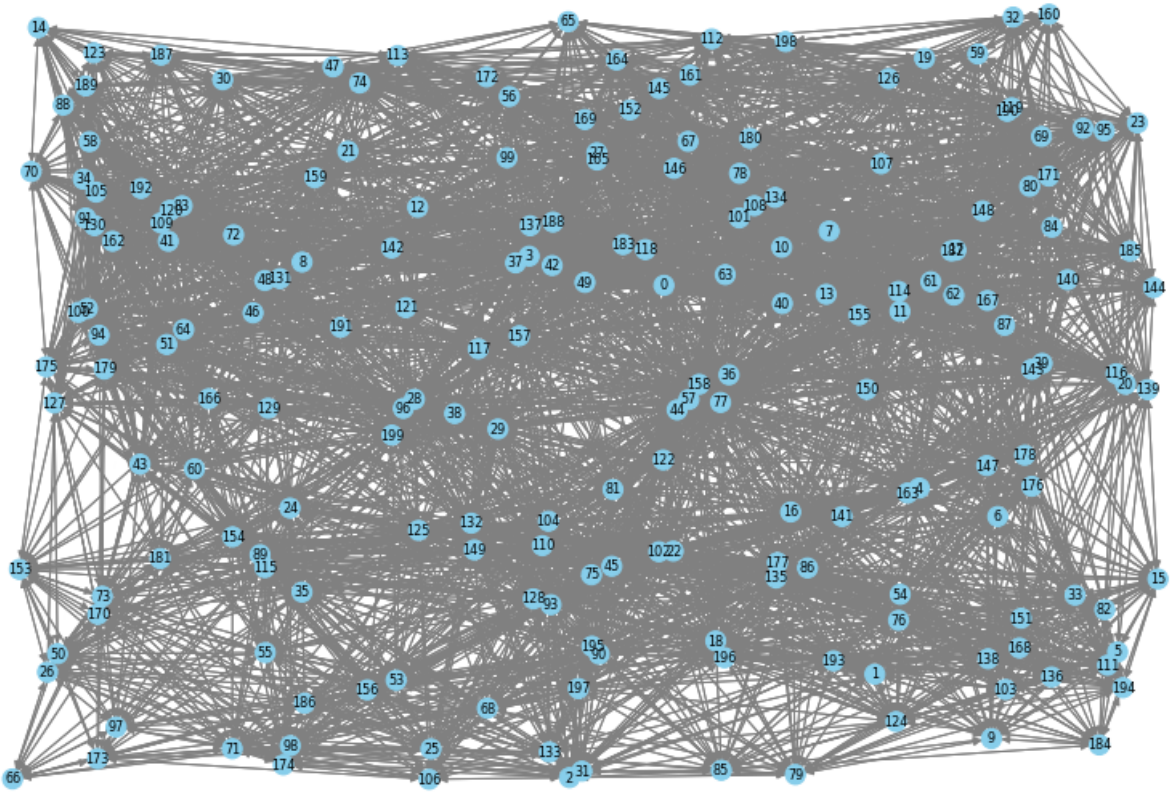
Initial score of the network. n.score > 0.5 is labeled red. Neutral, with -0.5 < n.score < 0.5 is labeled in light blue. Still not believing is labeled in blue.



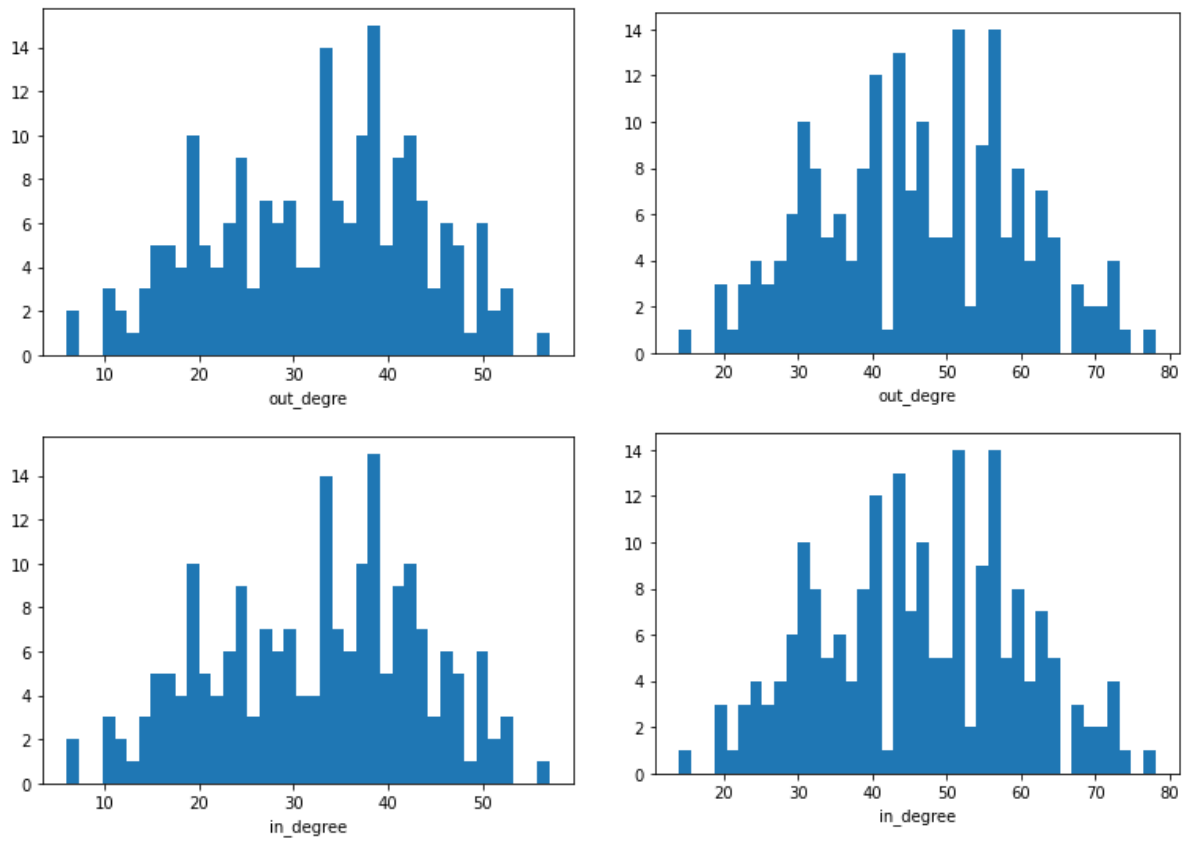
After 60 iterations, the number of people who tend to believe the information, with n.score > 0.5 is labeled red. Neutral, with -0.5 < n.score < 0.5 is labeled in light blue. Still not believing is labeled in blue.



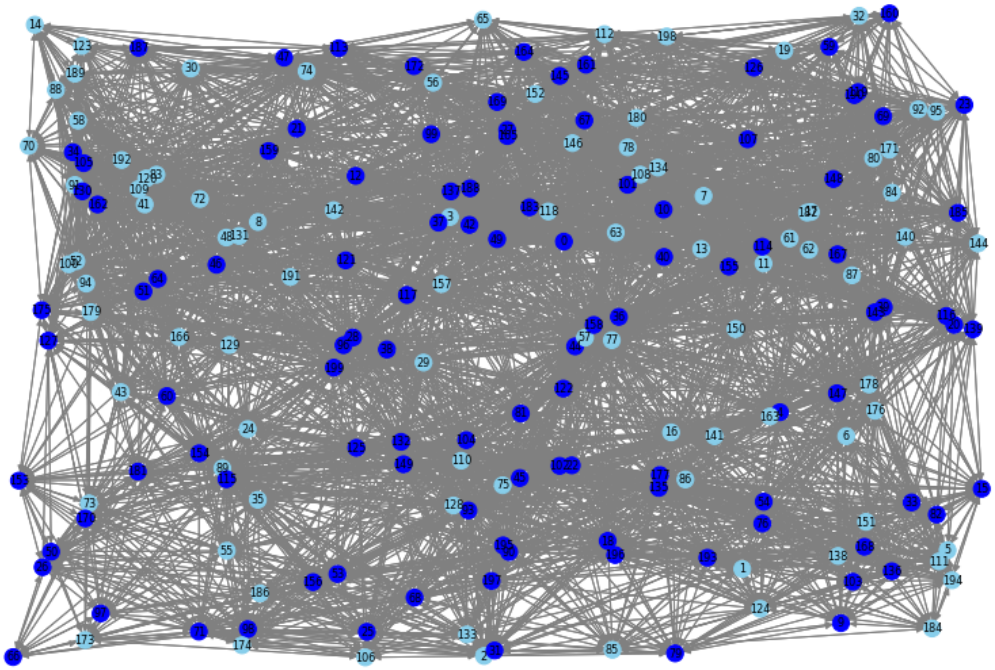
If the community is closely knitted, for example, a religious group, this form of proximity could generate information clusters that make information propagation easier. In this case, we set the random_cons = 0.25 compared to 0.2 in the previous case. The nodes within this distance have a bigger probability of connecting, thus a bigger probability of infecting others.
A reduced state graph of the initial network with 200 nodes is shown here. Common nodes are represented by the color of sky blue.



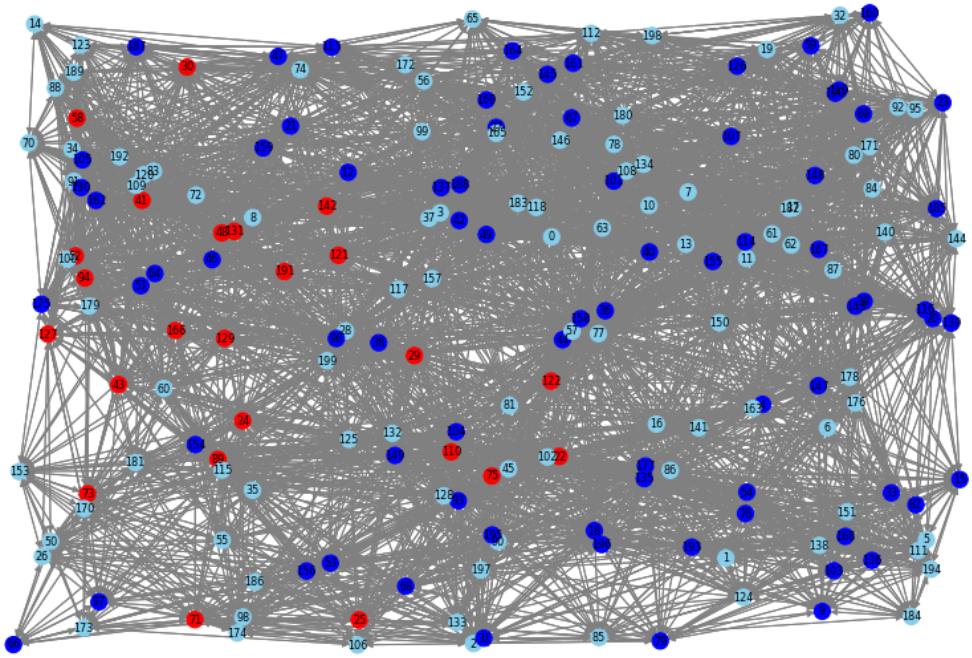
Observing the graph and comparing the degree information with the base simulation, we can see the number of connections between these nodes increased significantly. Left is base simulation, right is cluster simulation.



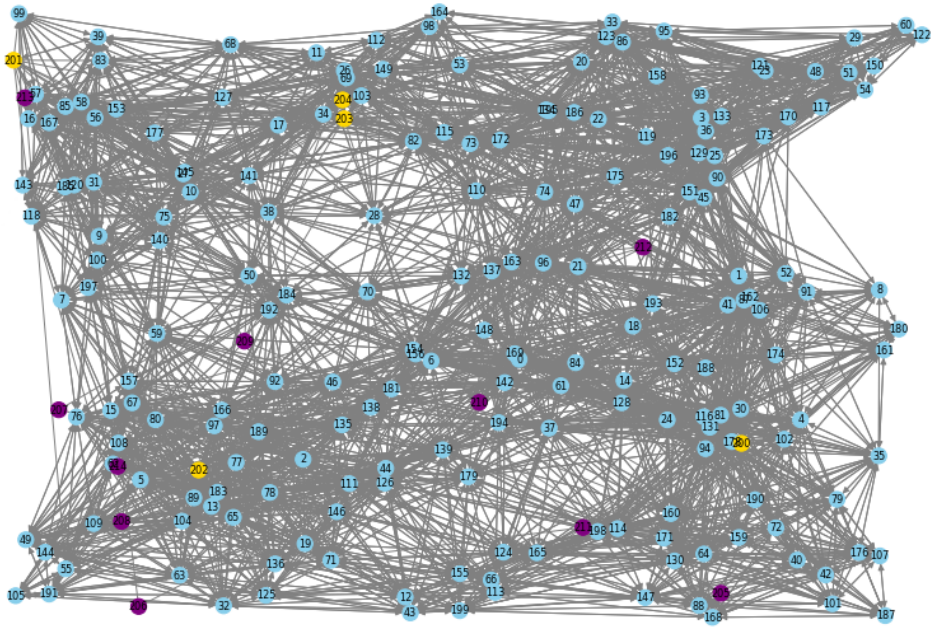
Initial score of the network with n.score > 0.5 is labeled red. Neutral, with -0.5 < n.score < 0.5 is labeled in light blue. Still not believing is labeled in blue.



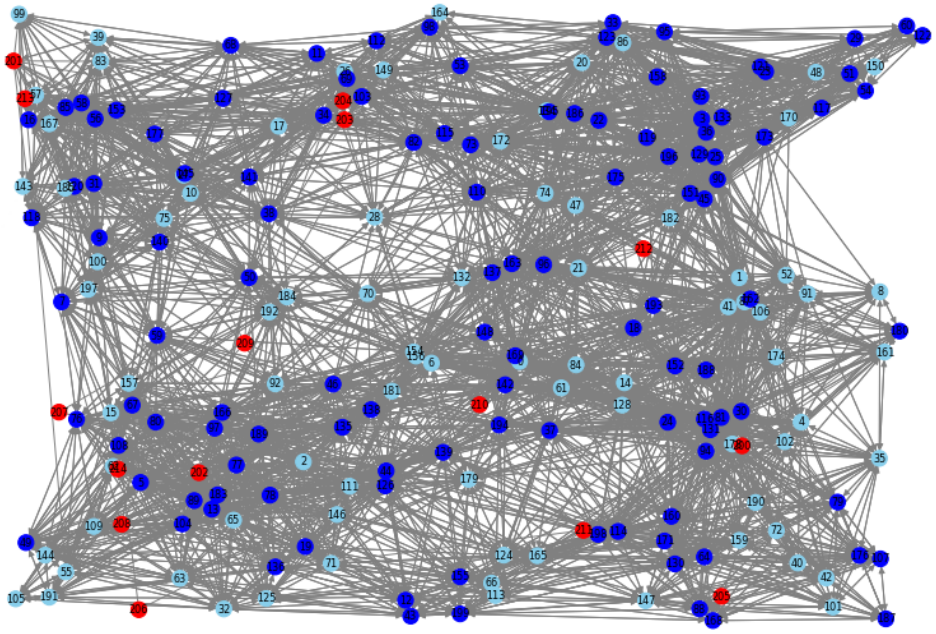
After 60 iterations, the number of people who tend to believe the information, with n.score > 0.5 is labeled red. Neutral, with -0.5 < n.score < 0.5 is labeled in light blue. Still not believing is labeled in blue.



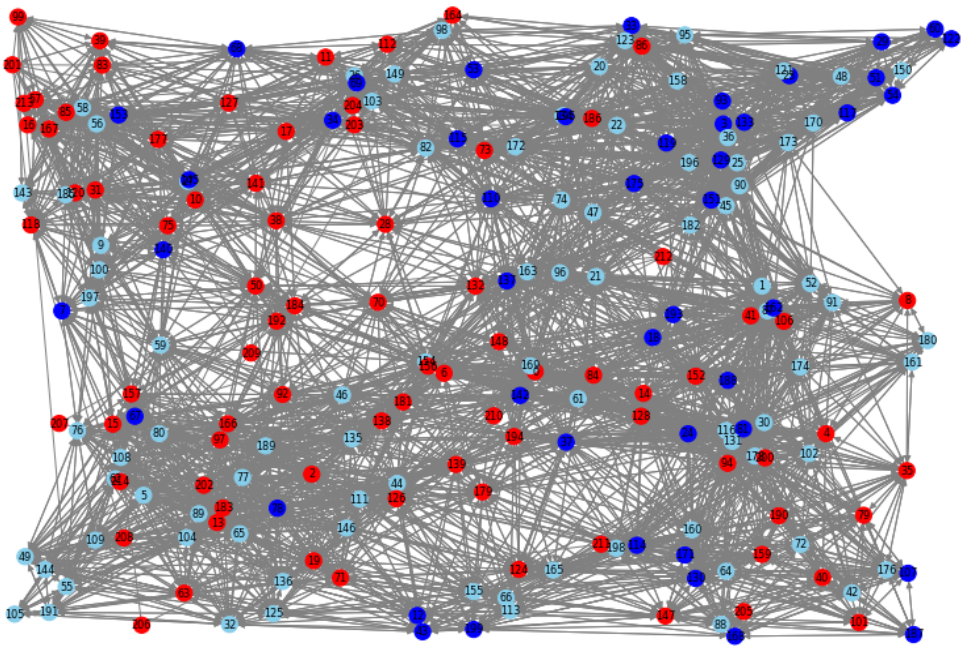
Finally we can take a hybrid method and proceed with 5 influencers and 100 bots. A reduced state graph of the initial network with 200 nodes is shown here. Common nodes are represented by the color of sky blue, influencers are in gold and bots are in purple.



Initial score of the network with n.score > 0.5 is labeled red. Neutral, with -0.5 < n.score < 0.5 is labeled in light blue. Still not believing is labeled in blue.

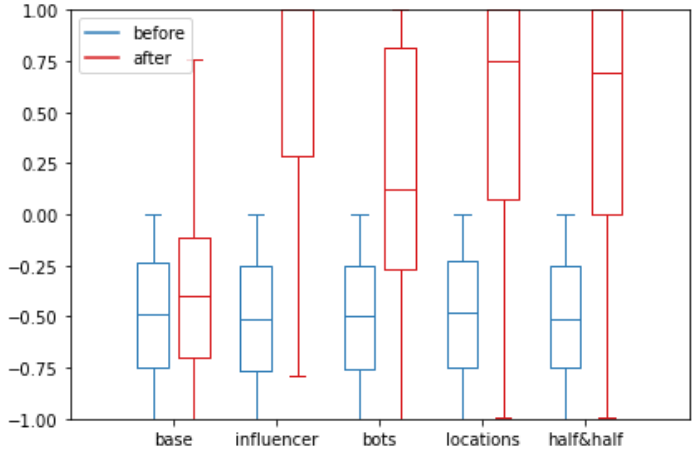


After 60 iterations, the number of people who tend to believe the information, with n.score > 0.5 is labeled red. Neutral, with -0.5 < n.score < 0.5 is labeled in light blue. Still not believing is labeled in blue.



Results:

Running simulations with the above logic and 2000 common nodes, we generate a box plot of the node scores.

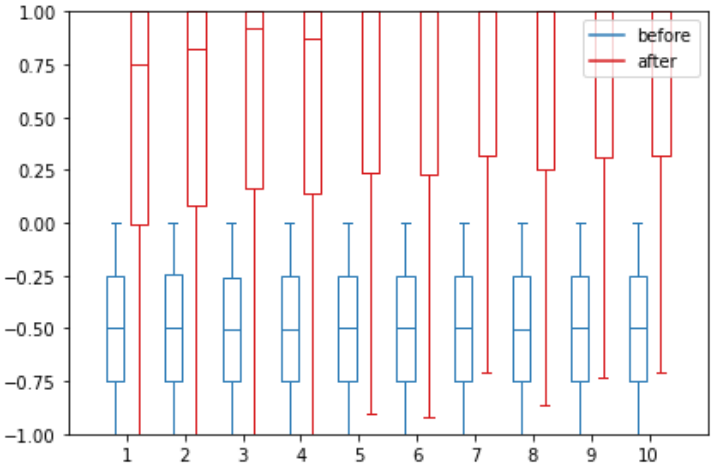


From the box plot of common node scores, it looks like employing influencers is the most effective way of spreading misinformation with a median score change of 1.5. The final score of common nodes under the influencer approach has a median of 1 as well as a highest first and third quantile. Group clustering and the half and half method also made a solid effect.

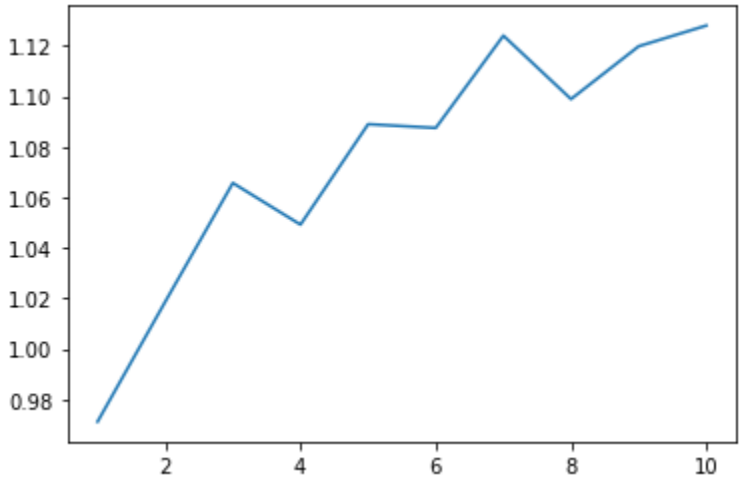
Looking for the threshold of influencer number:

A median score of fully biased 1 seems to be an overkill. Since we used 10 influencers with 2000 common nodes, we are interested in determining a threshold that is good enough to achieve the best effect.

Thus, we ran simulations with 2000 common nodes and number of influencers from 1 to 10. Each variation is runned 10 times. A rough result is as follows.



Starting from 5 influencers, the median tends to be at 1. Furthermore, starting from 7 influencers, even the most stubborn ones start to get deceived by our misinformation.
The change in mean value of the score during the simulation can be visualized by the following graph. X-axis is the number of influencers and Y-axis is the change in score.



For the median, 5 influencers can achieve a score of 1. However, the mean value can still increase after 5 influencers. At 7 influencers, the mean value change roughly parallels that of 10 influencers. Thus, we determined 7 influencers to be our threshold for the best results.
The threshold of 7 is determined based on 2000 common nodes and we have not investigated the relationship between number of influencers and number of common people. The result is only valid with 2000 common nodes and may not be proportional in the other cases.

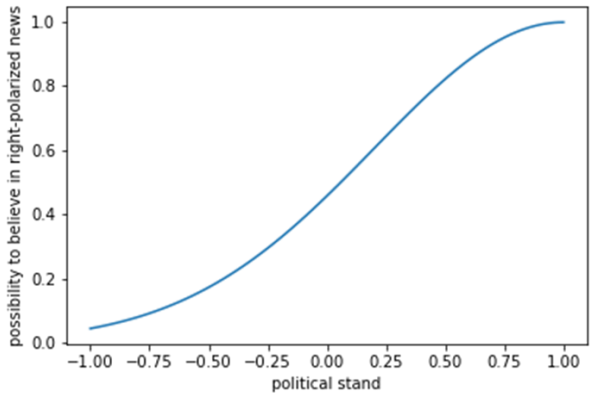
ii. Idea (2): Political segregation by the effect of biased news

Recent years, especially during the pandemic, people have noticed the enormous idea segregation between democrats and republicans. The left and the right are becoming harder and harder to agree with each other and they believe in totally different things and news source. The Covid 19 can be a hoax or a diaster. The mask and vaccination can save or control people. Bill gates can be a great entrepreneur and philanthropist or a evil demon who invented the virus. The sociaty is so divided now and we want to study the reason behind it. And we believe that the situation that our news are divided and biased is a major reason behind the political segregation. Fox vs BBC, CNN will convey totally different message toward the same event and have a huge effect on people because of on their influence.
For this idea, We want to study how politically biased news will affect people's political stand and how the society evolves under biased news.

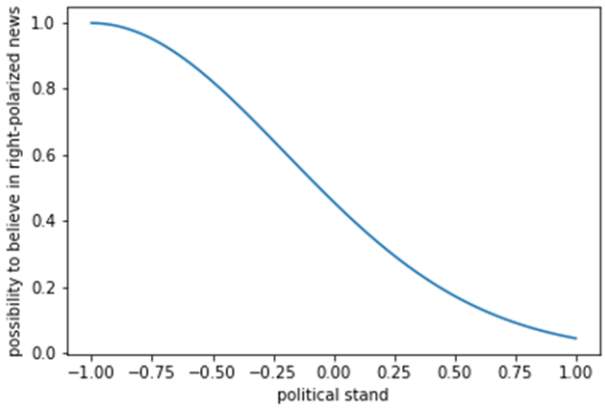
To achieve our goal, we first modified the code. Below are the changes

1, to simplify the process. I ignored the vulnerability, reshare rate and debunk rate. The node only have a score range from -1 to 1 representing the political stance. -1 means extreme left and +1 means extreme right. And the distribution of the scores follow a gaussian distribution centered at 0 with sigma=0.3 and clipped within [-1,1]

2, the distance between nodes, besides the nodes position x and y which represent their interest and geographical location, will also be affected by the initial political stance of the node. Thus people with the same political stance have a higher chance to be connected together.
3, the simulation process starts from randomly choosing one node to start the news. And half of the time a left biased news is generated and half of the time a right biased news is generated. Then it will propagate to its neighbors connected to it.
The chance for a node to believe in the right biased news is a gaussian function centered at 1 with a sigma=0.8:

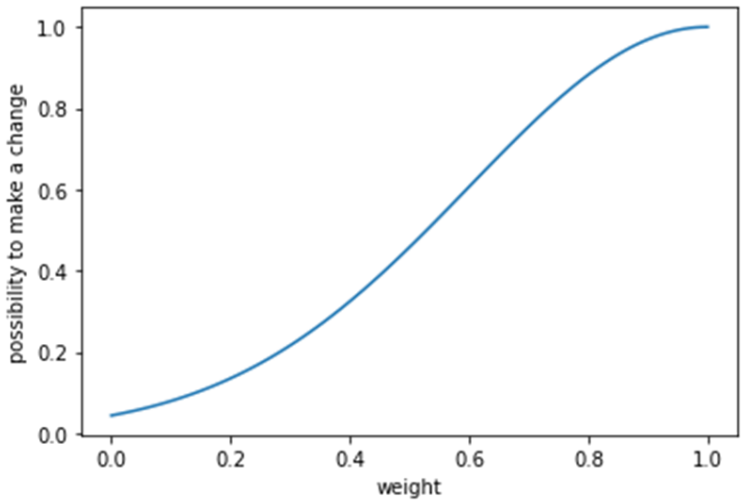


Thus the closer to the right the easier the people will believe in this right biased news.
Similar for a left biased news:



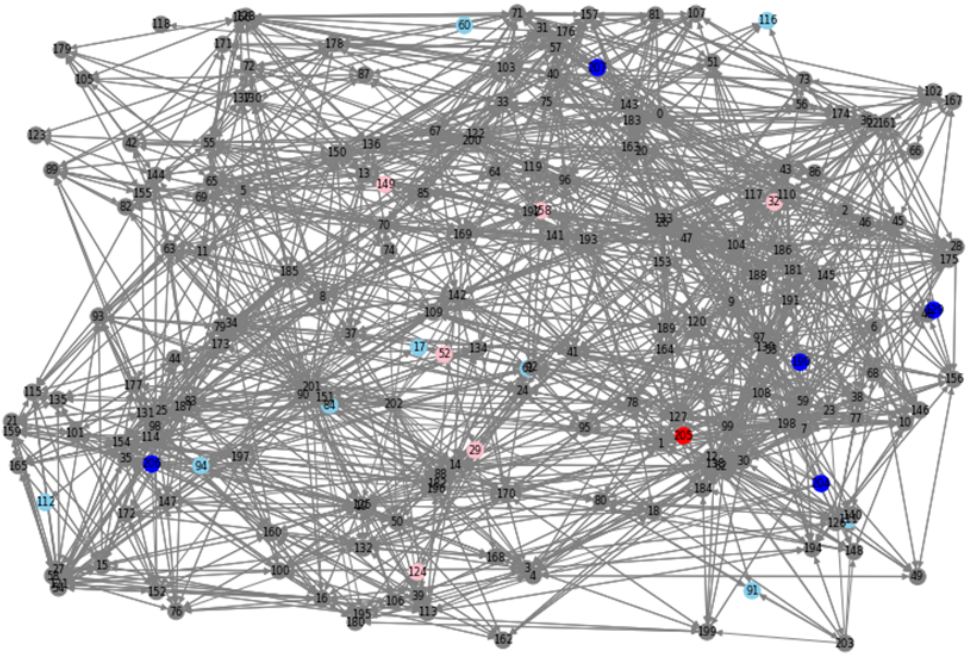
And once a person believe the news, his/her score (political stance) will be affeted. If it is a left biased news, the score will decrease 0.01. If it is a right biased news, the score will increase by 0.01. This means believing a biased news will bring people's political belief towards one direction.

After a person believe the news, they will share it to its connected neighbors. And whether the neightbor will be affected or not depends on the weight of the connection, which means how strong their relationship is, how they trust each other. The probability for the person to be affected by the sharing will also be a gaussian distribution centered at 1 with sigma=0.4:



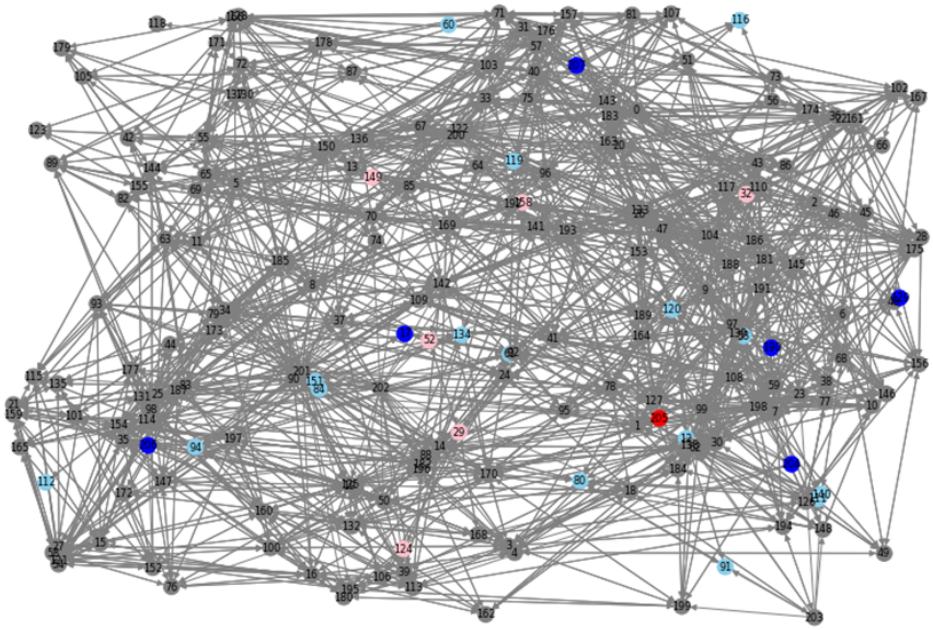
And if a person is affected by the sharing, it will also go through the same believe or not process and change their political stance accordingly as we mentioned above.
First, I use color grey for score between -0.5 to 0.5. And pink color for score 0.5 to 0.8, red for 0.8 to 1. Thus the reder the color the more 'right' the person is. I use skyblue for score -0.5 to -0.8 and blue for score -0.8 to -1. So the bluer the color is the more 'left' the person is.

First, with the gaussian distribution of the initial scores, we have most nodes in grey but only the robots are -1 or 1. And some pink and skyblue from the distribution.

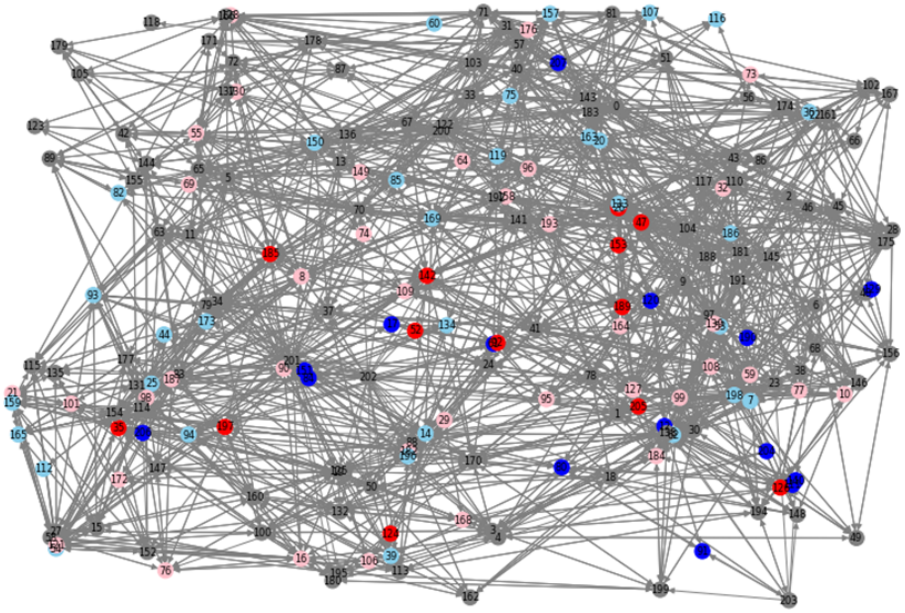


After some news belong released:

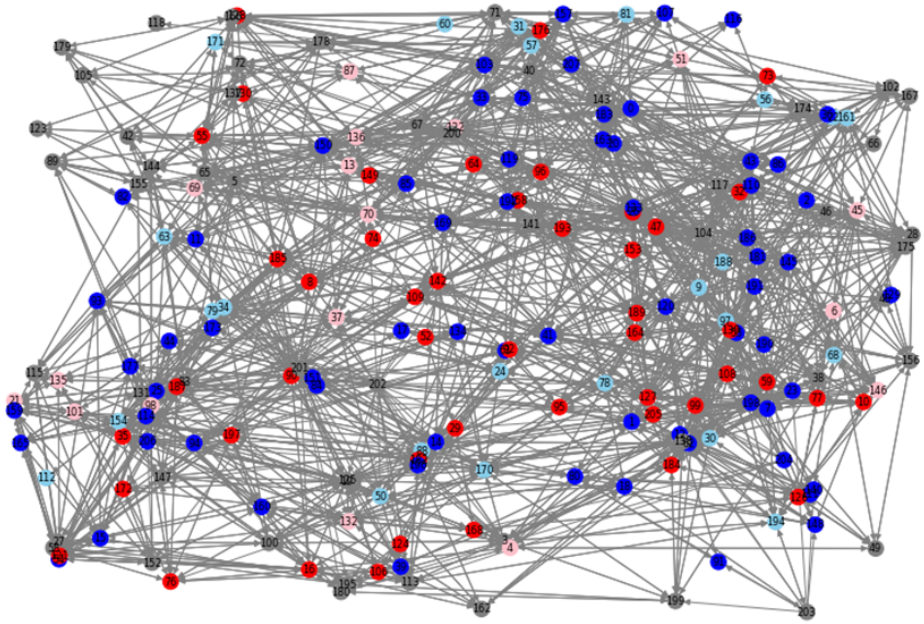
T=5



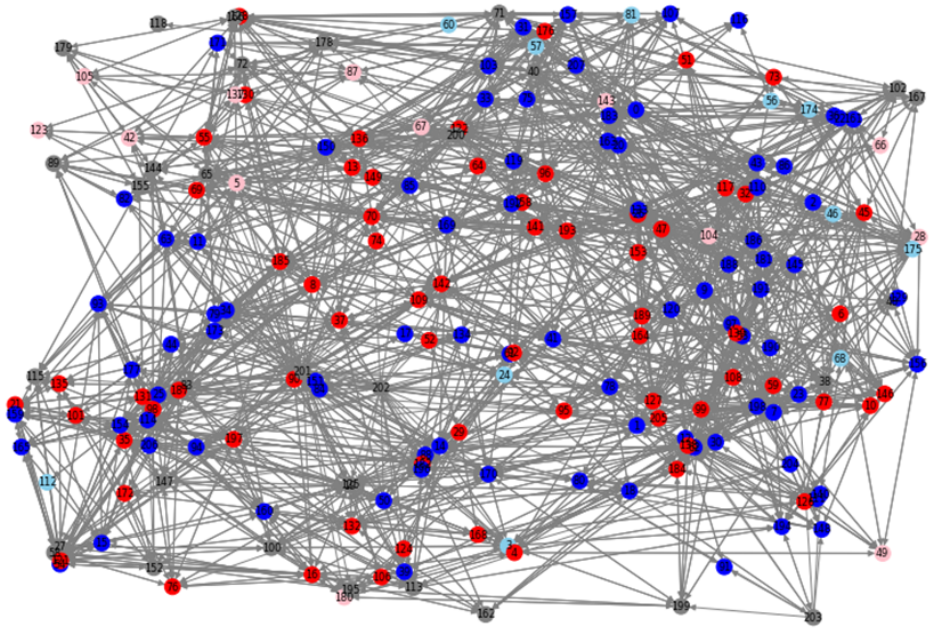
T=10:



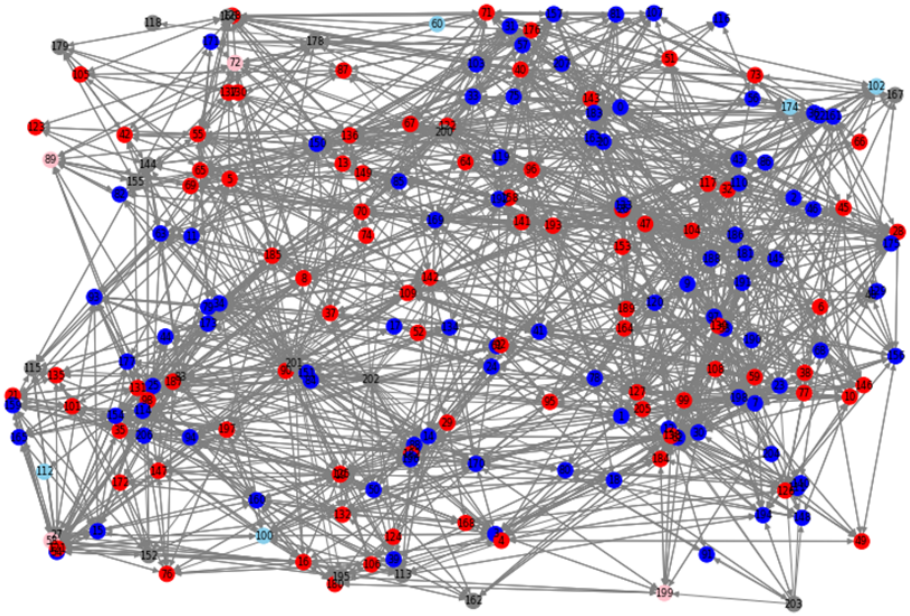
T=15:



T=20



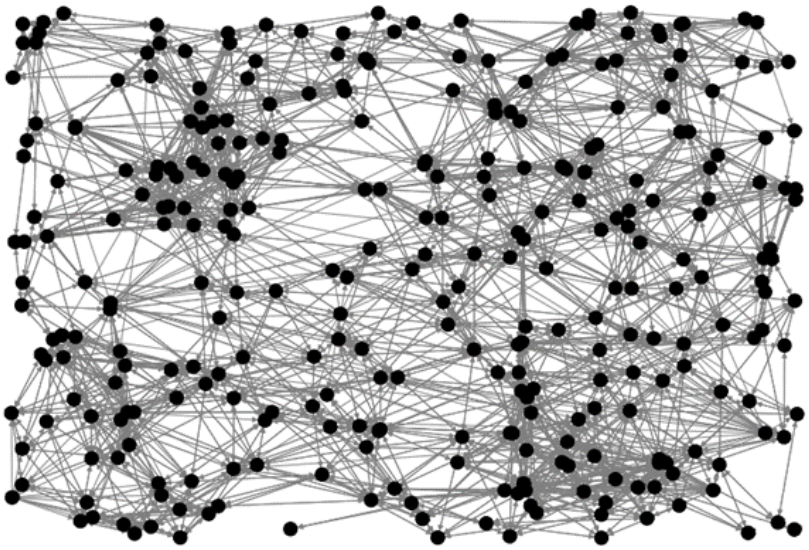
T=30



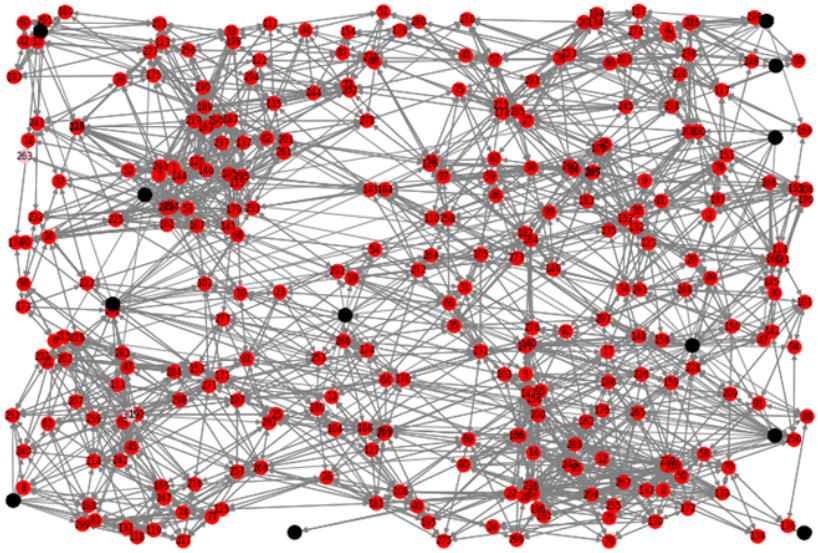
A clear political segregation has formed. So almost everyone becomes extreme left and extreme right.

Scenario two:

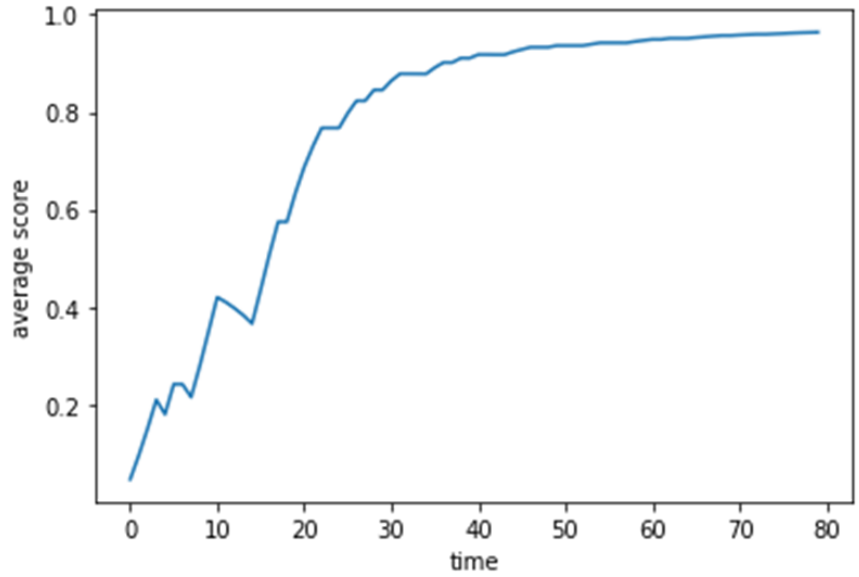
The node number is increased to 300 and node color is black if score between -0.5 to 0.5. Then I change the initial score distribution, I make everyone to start with zero score. Meaning everyone starts as absolute neural, even the bots. And we want to see how the system will evolve from a pure faultless state. So at the beginning the graph is all black:



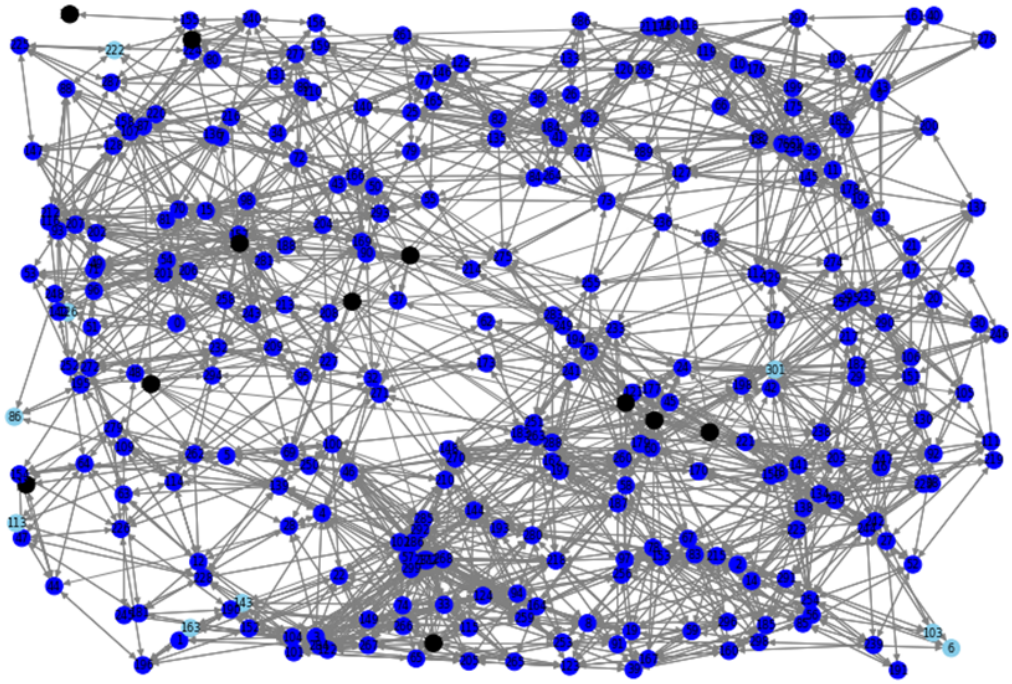
But I found the whole group will be tilted to extreme right or extreme left eventually.
T=80



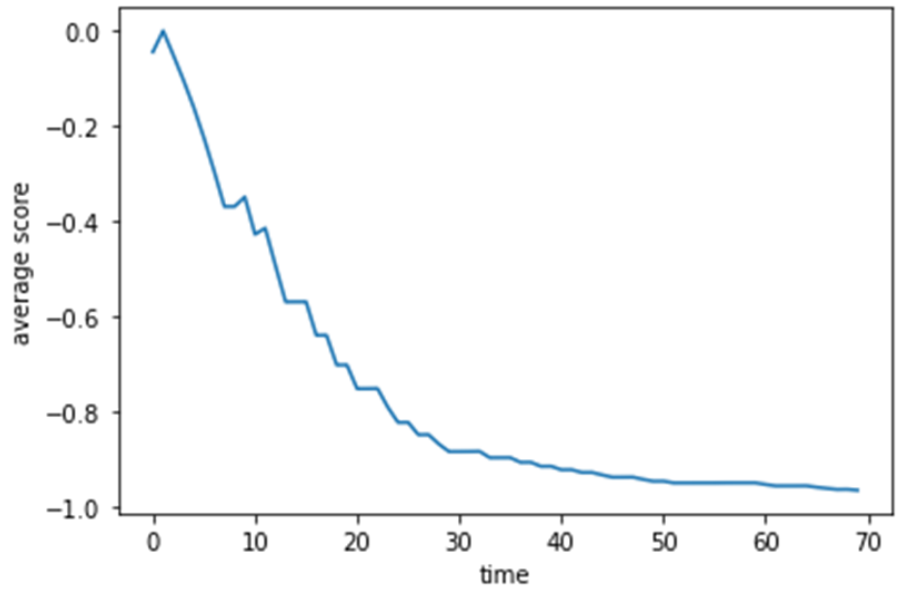
The average score vs time:



Another case after T=80:



The score vs time is



Thus the system is not stable at average score 0, with the influence of biased news. Even there are equal number of left biased and right biased news, the system will inevitably evolve to all extreme right or all extreme left. As for our society, the initial gaussian distribution of scores(political stance) is actually amazing as it can create a segregated society where at least two opinions can exist.

VII. Conclusion

In this project, we try to simulate the spread of fake news by investigating how fake news spreads on the Internet. The investigation focused on the use of the SBFC (susceptible, believing, fact-checking) model based on a variant of the SIR (susceptible, infected, recovered) model. We also introduce some realistic extensions to realistically simulate the spread of disinformation on the Internet.

To verify the validity of the model, we performed two simulations. One of them is The most effective strategy to spread a piece of information. We try to discover the most effective ways for misinformation to spread. In this process, we define two thousand nodes to simulate the process of information transfer and add influencers and bots in different experiments. Obviously, the number of influencers is very effective for information dissemination, but we finally found that in 2000 nodes, 7 influencers allow us to get the most efficient results. Another experiment is political segregation by the effect of biased news. Due to the current global covid19 environment and the decision-making and influence of different people, we wanted to study how politically biased news will affect people's political positions, and how society evolves under biased news. We performed two scenarios including three different experiments, one with a few extreme left and extreme right nodes, and two without any biased nodes. In the experiment with a few extreme left and extreme right nodes after 30 rounds, clear political segregation has formed. So almost everyone becomes extreme left and extreme right. In the scene without any biased nodes after 80 iterations, the system will inevitably evolve to all extreme right or all extreme left. We, therefore, believe that the initial Gaussian distribution of scores(political stance) is actually amazing as it can create a segregated society where at least two opinions can exist.

As for future extension work, we think time decay can be taken into consideration, the interest of the population in a fake news item decays over time. Another aspect can be looked at could be including more nodes and includes agents with multiple popularity levels in addition to regular users and influencers.

References:

[1] Del Vicario, Michela, et al. "The spreading of misinformation online." *Proceedings of the National Academy of Sciences* 113.3 (2016): 554-559.

[2] Vosoughi, Soroush, Deb Roy, and Sinan Aral. "The spread of true and false news online." *Science* 359.6380 (2018): 1146-1151.

[3] Lotito, Quintino Francesco, Davide Zanella, and Paolo Casari. "Realistic aspects of simulation models for fake news epidemics over social networks." *Future Internet* 13.3 (2021): 76.

[4] Sulis, Emilio, and Marcella Tambuscio. "Simulation of misinformation spreading processes in social networks: an application with NetLogo." *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2020.

[5] Tambuscio, Marcella, et al. "Fact-checking effect on viral hoaxes: A model of misinformation spread in social networks." *Proceedings of the 24th international conference on World Wide Web*. 2015.

[6] Safieddine, Fadi, Milan Dordevic, and Pardis Pourghomi. "Spread of misinformation online: Simulation impact of social media newsgroups." *2017 Computing Conference*. IEEE, 2017.

Appendix: Division of labor

Coding: Bochao, Ge Ren, Beichen Liang, Xiaoxu Zou

Report and analysis: Bochao, Ge Ren, Beichen Liang, Xiaoxu Zou