CS166 Assignment 3: Network simulation

Viet Hoang Tran Duong

Spring 2020

Part 1: Modifications & Assumptions

Comparing to the existed model, I propose a new model with these modifications.

Modifications:

- Multiple opinions model: (default: 3 opinions): representing politics, education, and sports. If I keep the three topics to have the same effect on the network, the analysis would not be meaningful. Hence, I added a topic preference variable (topic_pref: $0 \leq p \leq 1$), which indicates how hard it is to change people's opinions on these topics. The new opinion change of Person i when talking to a Person j in topic k will be:

$$\Delta o_i = p_k * \alpha * w_{ij} * (o_j - o_i)$$

The change in weight still follows the same formula:

$$\Delta w_{ij} = \beta * \mathrm{w_{ij}} * \left(1 - w_{ij}\right) * (1 - \gamma|o_i - o_j|)$$

This variable represents the phenomena that there are certain topics that are more sensitive and conservative: politics: that are harder to change mind but has the same effect of breaking relationships. The effect of the new variable will be discussed in section 2.

- New relationship formation: Relationships can be formed randomly, like two people meeting in the coffee shop. The new edge is created randomly by selecting two random nodes. The difference compared to the original model is weight. Instead of initializing at 0.5 (all new relationships are the same), I instead create the weight depending on the 1st conversation they have. The conversation topic is randomly chosen. The function of the new weight is

$$1 - \frac{1}{1 + e^{-|o_i - o_j|}}$$

The second part of the formula derives from the sigmoid function to create value between 0 and 1. However, for the sigmoid function, the smaller the absolute opinion difference, the smaller the value. Therefore, I subtract such value from 1 to create the appropriate relationship and new value.

- Individual persuasiveness:
Instead of a fixed value for α, β, γ, I randomized the values using a normal distribution to represents the variability of the personalities. Each person will have an individualized α, and each relationship will have corresponding β, γ values.

Model variables:

- Network size (default = 50): positive integer: number of nodes in the system
- Alpha ($\alpha \in (0, 0.5]$): The rate at which nodes adjust their opinion to match the neighboring nodes' opinions. Real-life representation: how openminded people are.
With alpha standard deviation ($std_\alpha > 0$): The deviation within the population

$$\alpha_i \sim N(\alpha, std_\alpha) \;\&\; \alpha_i \in (0, 0.5]$$

Beta ($\beta \in [0, 1]$): The rate at which edge weights are changed in response to opinion difference. Real-life representation: the loyalty of the connection. The smaller the beta, the more loyal the relationship.
With beta standard deviation ($std_\beta > 0$): The deviation within the population

$$\beta_i \sim N(\beta, std_\beta) \ \& \ \beta_i \in [0, 1]$$

- Gamma ($\gamma > 0$): The stubbornness/pickiness regarding the opinion differences between 2 nodes. Nodes with opinions differing by more than γ^{-1} will result in an edge decreasing. with gamma standard deviation ($std_\gamma > 0$): The deviation within the population

$$\gamma_i \sim N(\gamma, std_\gamma) \ \& \ \gamma_i > 0$$

- Number of opinions (a positive integer): number of topics discussing. Each topic has a value p ($p \in [0,1]$) for how conservative the topic is. The smaller the p, the more conservative the topic/harder to change people's minds on these topics. p is somewhat similar to γ, but instead of focusing on the connections like γ, I prefer working with the people's opinions. Also, this better represents the world where people's minds are hard to change.

Assumptions:

- The topic of discussion at each time is random.
- The system is isolated (no new nodes are created)
- People mostly interact with people surrounding them, with a random chance of 1% meeting someone new.
- Each person starts with a strong opinion on every topic (either 0 or 1), representing two poles of every topic.
- The connections are undirected and have weight. Once the weight becomes too small (0.05), we terminate the connection.
- The topic conservativeness (p) are universally agreed (e.g., $p_{soccer} > p_{education} > p_{politics}$)
- The distributions of a person's open-mindedness or the loyalty or stubbornness of the connections are normal.

Modifications reasoning:

First, instead of having the same parameters for the whole population, I created individualized parameters for each node, edge. This represents the uniqueness of each person and connections in society. Second, the random edge creation is closer to the real-world scenario. New connections are created randomly, but most importantly, depending on the very first conversation between the two. This phenomenon reflects in the "halo effect," where first impression matters. Hence, the weight of the connection depends on the random topic these two nodes communicate at the beginning. Finally, I added multiple topic models. Furthermore, for each topic, I create a conservativeness variable p to represent how hard it is to change people's minds on these topics. For example, changing people in politics is much harder than soccer. All these additions create a more complicated system to the model to better mimic society.

Part 2: Local analysis:

Only two people i and j. When a person i talks to person j on topic k (p_k is the how sensitive the topic is: the more sensitive, the harder it is to change mind):

$$\Delta o_i = p_k * \alpha_i * w * (o_j - o_i)$$

$$\Delta o_j = p_k * \alpha_j * w * (o_i - o_j)$$

$$\Delta w_{ij} = \beta w * (1 - w) * (1 - \gamma |o_i - o_j|)$$

For the local analysis purpose, assume $\alpha_i = \alpha_j = \alpha, p_k = p$. Call the absolute opinion difference between i and j at time t to be

$$\Delta o_{ij_t} = |o_{i_t} - o_{j_t}|$$

Without losing generality, assuming $o_{i_t} \geq o_{j_t}$.
Then, the new $o_{i_{t+1}} = o_{i_t} + p * \alpha * w_{ij} * (o_j - o_i)$; $o_{j_{t+1}} = o_{j_t} + p * \alpha * w_{ij} * (o_i - o_j)$

$$\Delta o_{t+1} = \left|o_{i_{t+1}} - o_{i_{j+1}}\right| = \left|o_{i_t} - o_{j_t} - 2p\alpha w(o_{i_t} - o_{j_t})\right| = |\Delta o_t(1 - 2p\alpha w)|$$

We have $0 \geq -2p\alpha w_{ij}(o_{i_t} - o_{j_t})$ and $one \geq p\alpha w_{ij}$
(as $1 \geq o_{i_t} - o_{j_t}, p, \alpha, w_{ij} \geq 0$)

$\Rightarrow 0 \geq -2p\alpha w_{ij}(o_{i_t} - o_{j_t})$ and $-2\left(o_{i_t} - o_{j_t}\right) \leq -2p\alpha w_{ij}(o_{i_t} - o_{j_t})$
(as $-2\left(o_{i_t} - o_{j_t}\right) \leq 0$)

$\Rightarrow 0 \geq -2p\alpha w_{ij}(o_{i_t} - o_{j_t}) \geq -2(o_{i_t} - o_{j_t})$

$\Rightarrow o_{i_t} - o_{j_t} \geq o_{i_t} - o_{j_t} - 2p\alpha w_{ij}(o_{i_t} - o_{j_t}) \geq -(o_{i_t} - o_{j_t})$

$\Rightarrow o_{i_t} - o_{j_t} \geq o_{i_t} - o_{j_t} - 2p\alpha w_{ij}(o_{i_t} - o_{j_t}) \geq -(o_{i_t} - o_{j_t})$

$\Rightarrow \Delta o_{t+1} = \left|o_{i_t} - o_{j_t} - 2p\alpha w_{ij}(o_{i_t} - o_{j_t})\right| \leq |o_{i_t} - o_{j_t}| = \Delta o_t$

$\Rightarrow \Delta o_{t+1} \leq \Delta o_t$

Regardless of the parameters, the absolute opinion difference will reduce over time.

Hence, we see a strong connection between α and β. If the rate of change of the opinion difference cannot keep up with the rate of change of the weights, then the weights will break before the difference is small enough for the weight to increase again. Here is a vector plot to demonstrate such property:

The arrows are for the vector plot. The red and green lines are the actual path of local simulation starting from the same initial location where we plot the vector space (how the opinions difference and weight will change). The green line implies the weight will strengthen a converges to 1. The red line implies breaking the connection.

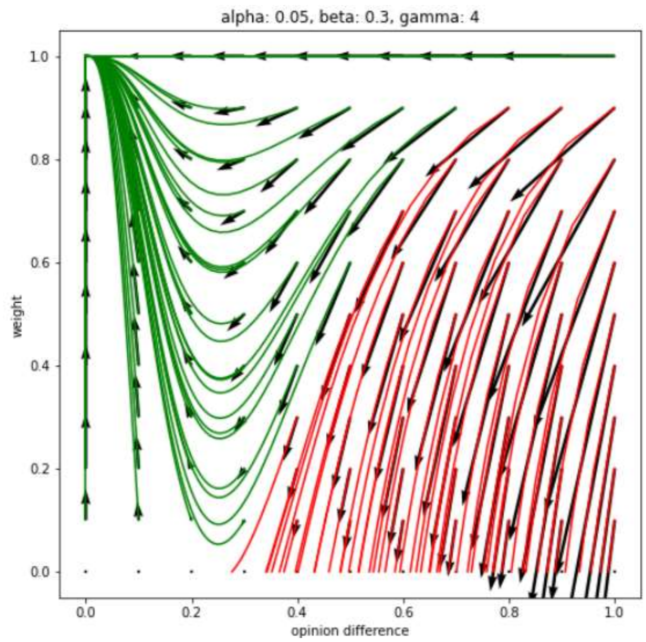


Fig 1: The vector field of the opinion and weight difference.

Each pair of parameters have a convergence rate: the percentage of paths end up with weight = 1. Here is the contour plot of the convergence rate through pairs of values. As we can see, the convergence is high when β/α is large, and when the two variables are larger as well. (1)

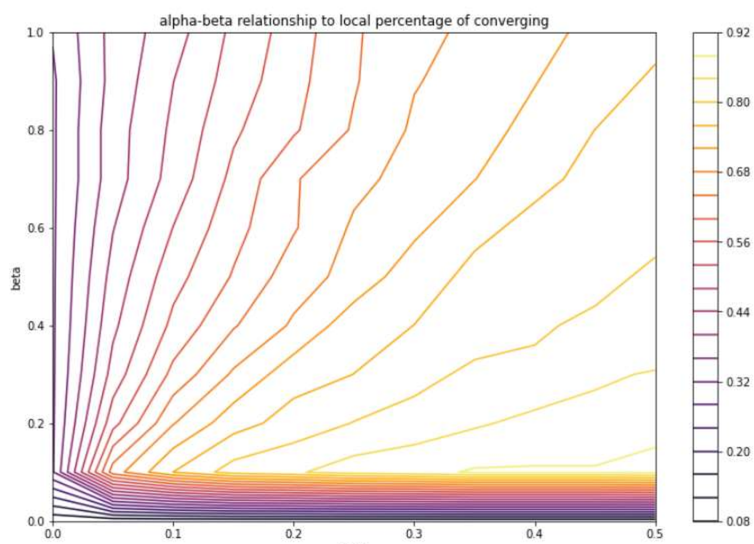


Fig 2: The contour plot of opinion convergence on the values of alpha and beta.

Another key parameter is γ

If $\gamma < 1 \Rightarrow \ \gamma|o_i - o_j| \leq 1, \Rightarrow \ 1 - \ \gamma|o_i - o_j| \geq 0 \Rightarrow \Delta w_{ij} = \beta w * (1 - w) * \left(1 - \gamma|o_i - o_j|\right) \geq 0.$

Then, the weight will always increase and leads to convergence. To understand γ better, we plot the contour for understanding the relationship between γ and α, β :

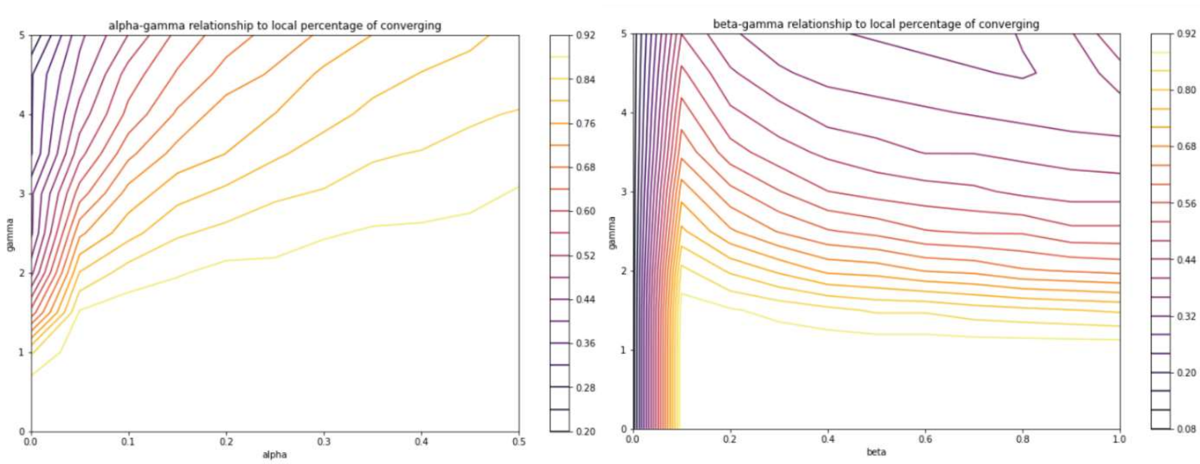


Fig 3: The contour plot of opinion convergence on the values of gamma and alpha or beta.

To find the critical values, we are looking for a location where the convergence is around 0.5.

As we can see, α is sensitive to γ: γ will force α to be smaller ($\alpha \in [0, 0.1]$) to get convergence around 0.5. For β (and with the condition (1): β/α is large enough), γ will force β to be in the range [0.2, 0.4] for convergence rate = 0.5 Finally, the value of γ that experience the widest range of variety of behaviors is at $\gamma = 4$.

Based on these plots, we choose the parameters of the model to be: $\alpha = \ 0.5, \beta = 0.3, \gamma = \ 4$

Also, I decide to use the Barabasi-Albert (BA) model for a random scale-free network with a preferential attachment mechanism. The reason is preferential attachment, which is similar to the real-world scenario, leading to the power-law degree of distributions. The purpose has asymmetry in the network: humans are not equal. Also, the purpose of this simulation is to check if the interactions will create clusters. Hence, I avoid using Watts-Strogatz because it already has the small-world property (which could be confounding for any formulation of clusters through model interaction), and Erdos-Renyi (as the random model is not representative).

Part 3: Model simulation

I draw the network on four different graphs: 3 graphs to represent the consensus within each topic, and the final graph to represent which topic each person has the strongest opinion on.

Topic 0: soccer, topic 1: education, topic 2: politics

In the end, here are the results

```
% likes topic 0: 24.0
% likes topic 1: 18.0
% likes topic 2: 58.0
```

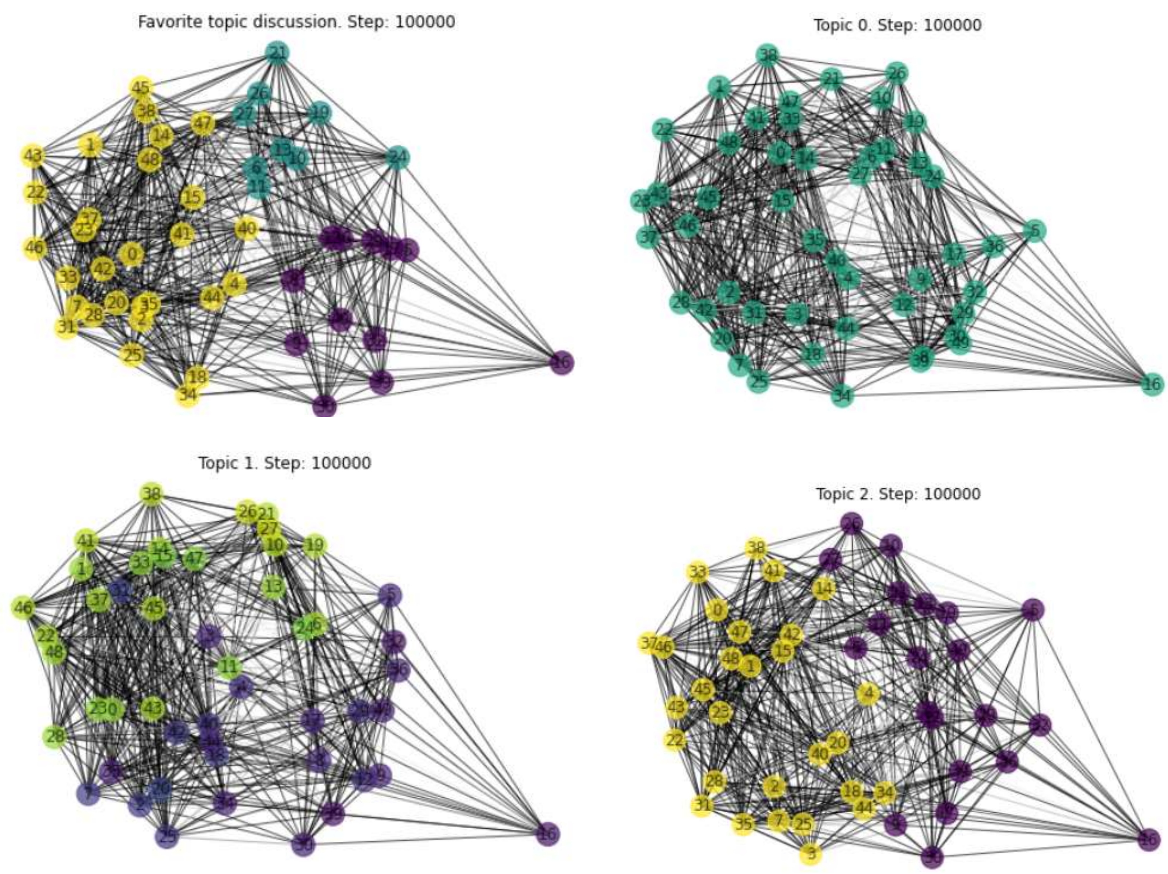


Fig 4: The final state of the network (after 100 000 steps)

As we can see, in the end, society is very connected. There are a few interesting patterns.

First, the people with the similar topic preference are closer to each other. In the "favorite topic discussion" graph has 3 clusters for three topics. This result is reasonable in real-life because it is easier talking about the same passionate topic, even if we might have differences in other topics.

Second, the soccer topic, with a high level of the open-mindedness of people and a comfortable topic to discuss, it can easily reach consensus among people. This result is interesting because we usually expect a polarized graph for a single topic simulation. This result can be explained by the strong connections people have due to politics and education that pulls people together on unimportant matters. As we can see, the strong connections in the topic 0 graphs are exactly those connecting the in-cluster nodes in topics 1 and 2. As these connections are still valid and the soccer topic is not conservative, over a long period of time, people will eventually converge. This is the problem with long-term simulations of the isolated system. To be more realistic, we can add strong preferential nodes (soccer influencers, those mindsets on soccer cannot be changed and observed the behaviors).

Third, the conservative topics (especially topic 2) separates the graph into two clear halves for the contrasting idea. However, the interesting trait is that there are still connections between the nodes from two different clusters. This result derives from topic one strong connections within clusters, and topic one and topic two each have two different clusters, and the nodes of the clusters are not the same, which helps to make the network well connected.

The well connected can also be reasoned by the connection adding mechanism. As we add more connections and all these connections built upon similarities, the majority of them will last long or die out fast. However, as these are new connections, which is highly likely to be those across clusters. Hence, after 100 000 steps, the network becomes very well connected. If we look at the numbers of edges over time, we can see that at first, the network starts eliminating weak connections (around 100 connections). After step 10000, it starts developing more connections, around 500 new connections.

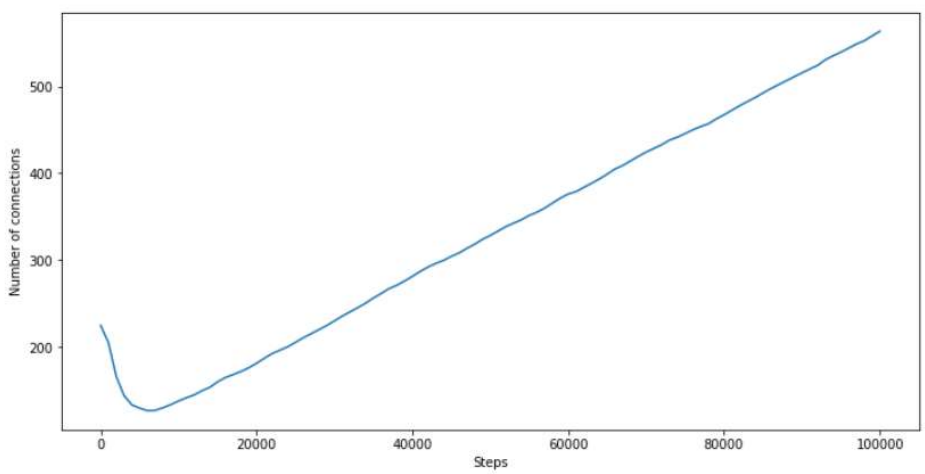


Fig 5: The number of connections with respect to the steps.

Furthermore, after a period of eliminating weak connections, the average weight of all edges becomes very high, around 0.9. This is reasonable because, based on the vector field, the connections usually decrease and either vanish or surge and converge to 1.

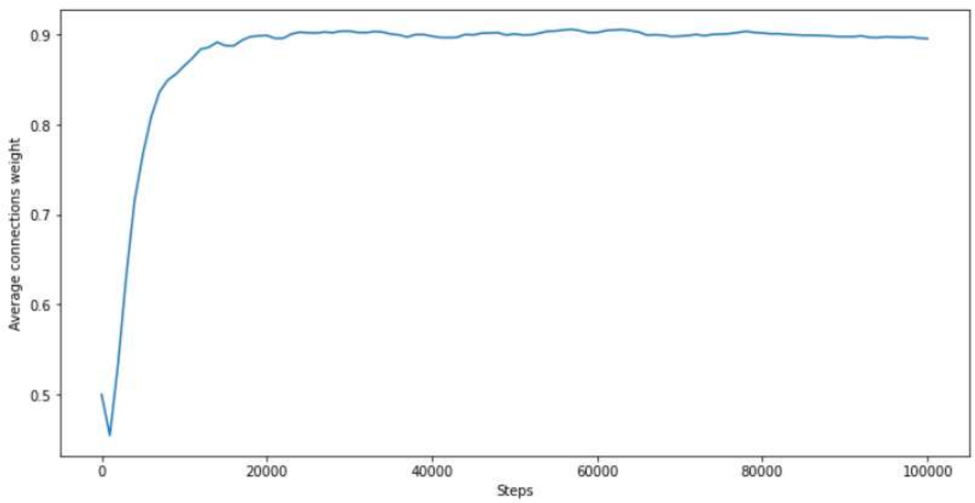


Fig 6: The average weight of connections with respect to the steps.

Also, we define a new parameter, which is topic polarization. For each topic, the polarization is the difference between the mean of people having the opinion > 0.5 and of those having an opinion < 0.5.

Based on this graph, we can see that for topic 3, the opinions polarization is flat. This is because all opinions must be either all above 0.5 or less than 0.5. Hence, we can see a consensus. The most interesting trend is for a more conservative topic: topic 1. The polarization values keep decreasing, implying the opinions of 2 different clusters are still trying to find common ground but not yet reach consensus. The opinions of nodes in topic one will have to converge at some point because the opinion difference always decreased, and some connections are too strong to be terminated (those in the same topic 2 clusters). In one simulation, after 150 000 steps, the polarization values of topic one is still decreasing and level off. On average, to predict when topic 1 reaches consensus, as topic 0 takes 60 000 steps, topic one would take approximately ten times more (because the opinion changed rate is much lower due to conservativeness of topic), and topic 2 takes 100 times more. Those are only estimation and prediction. Due to computational power, I will leave that analysis for future exploration.

Here is the result of average case for topic convergence

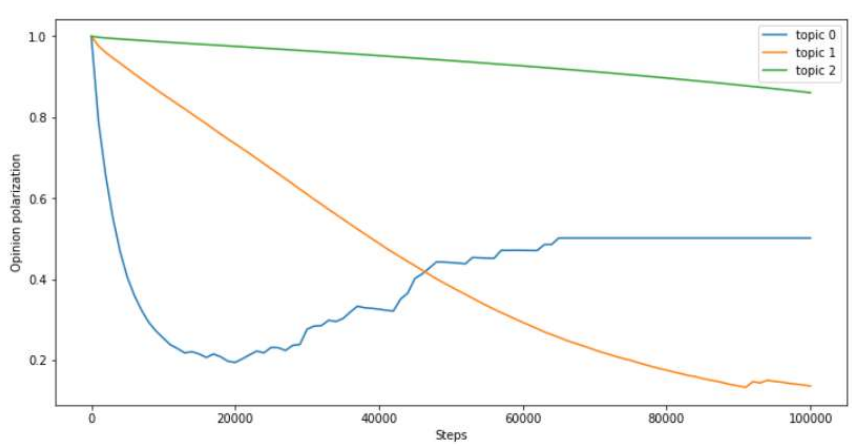


Fig 7: The number of the opinion polarization with respect to the steps for 30 trials.

Here is the result of the mentioned one simulation:

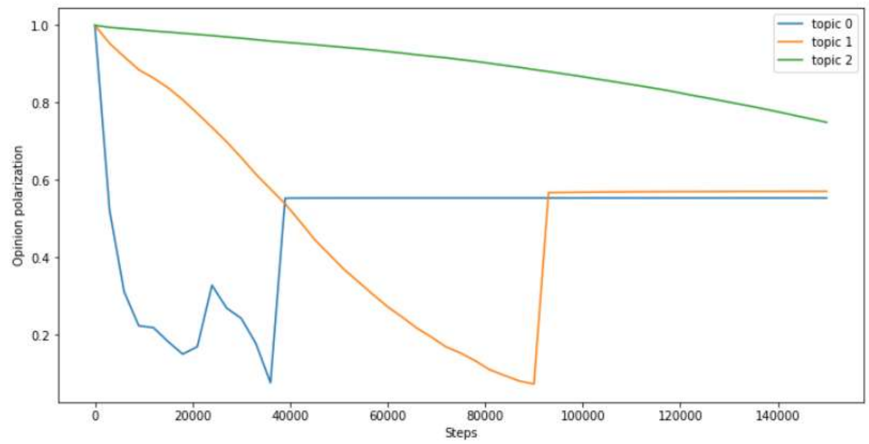


Fig 8: The number of the opinion polarization with respect to the steps for 1 simulation

Model simulation

```
from matplotlib import pyplot as plt
import networkx as nx
import random
import numpy as np
random.seed(2020)
from scipy.stats import norm
```

```


class SocialDynamicsSimulation:
    '''
    Simulate social dynamics by strengthening opinions and connection weights
    based on random interactions between nodes.
    '''

    def __init__(self, network_size=50, alpha=0.05, beta=0.3, gamma=4, alpha_std = 0.01, beta_std=0.05, gamma_std=1, num_opinion =
        '''
        Inputs:

            network_size (int) The number of nodes in the random Watts-Strogatz
              small-world network. Default: 50.

            alpha (float) The rate at which nodes adjust their opinions to
              match neighboring nodes' opinions during interactions.
              Default: 0.05. This has a standard deviation: alpha_std. Default: 0.01

            beta (float) The rate at which edge weights are changed in
              response to differing opinions. Default: 0.3.
               This has a standard deviation: abeta_std. Default: 0.05

            gamma (float) The pickiness of nodes. Nodes with opinions differing
              by more than 1/gamma will result in an edge weight decreasing.
              Default: 4.
               This has a standard deviation: alpha_std. Default: 1

            num_opinion: number of opinions (integer)
```

```
            num_opinion: number of opinions (integer)

        '''
        self.network_size = network_size
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma
        self.alpha_std = alpha_std
        self.beta_std = beta_std
        self.gamma_std = gamma_std
        self.num_opinion = num_opinion

    def initialize(self):
        '''
        Initialize the simulation with a random graph, with random 0 or 1
        opinions assigned to all nodes and initial edge weights of 0.5.
        '''
        self.graph = nx.barabasi_albert_graph(50, 5, 2020)
        for edge in self.graph.edges:
            self.graph.edges[edge]['weight'] = 0.5
            # generate random values from a normal distribution
            self.graph.edges[edge]['beta'] = norm.rvs(self.beta, self.beta_std, size = 1)[0]
            self.graph.edges[edge]['gamma'] = norm.rvs(self.gamma, self.gamma_std, size = 1)[0]
        for node in self.graph.nodes:
            # an array of opinion values
            self.graph.nodes[node]['opinion'] = [random.randint(0, 1) for _ in range(self.num_opinion)]
            # generate random values from a normal distribution
            self.graph.nodes[node]['alpha'] = norm.rvs(self.alpha, self.alpha_std, size = self.num_opinion)
        self.layout = nx.spring_layout(self.graph)  # Initial visual layout
        self.step = 0

    def observe_fav_topic(self):
        '''
        Draw the state of the network with which topic each node is most prefer
        '''
        self.layout = nx.spring_layout(self.graph, pos = self.layout, iterations=5)
        plt.clf()
        # all the nodes and get the max opinion
        master_list = np.array([list(self.graph.nodes[i]['opinion']).index(max(self.graph.nodes[i]['opinion'])) for i in self.graph
        nx.draw(
            self.graph, pos=self.layout, with_labels=True,
            node_color = master_list,
```

```
            node_color = master_list,
            edge_color=[self.graph.edges[i, j]['weight'] for i, j in self.graph.edges],
            edge_cmap=plt.cm.binary, edge_vmin=0, edge_vmax=1,
            alpha=0.7, vmin=0, vmax=self.num_opinion - 1)
        plt.title('Favorite topic discussion. Step: ' + str(self.step))

        # print the percentage preference
        for i in range(self.num_opinion):
          print("% likes topic", i, ":", np.mean(master_list == i)*100)
        plt.show()

    def observe_topic(self, topic):
        '''
        Draw the state of the network for each topic
        '''
        self.layout = nx.spring_layout(self.graph, pos = self.layout, iterations=5)
        plt.clf()
        nx.draw(
            self.graph, pos=self.layout, with_labels=True,
            # get the topic value for each node
            node_color=[self.graph.nodes[i]['opinion'][topic] for i in self.graph.nodes],
            edge_color=[self.graph.edges[i, j]['weight'] for i, j in self.graph.edges],
            edge_cmap=plt.cm.binary, edge_vmin=0, edge_vmax=1,
            alpha=0.7, vmin=0, vmax=1)
        plt.title('Topic ' + str(topic) +  '. Step: ' + str(self.step))
        plt.show()

    def update(self):
        if random.uniform(0, 1) < 0.01:
            # Create a new edge with 1-sigmoid(opinion difference in selected topic)
            nodes = list(self.graph.nodes)
            while True:
                new_edge = random.sample(nodes, 2)
                if new_edge not in self.graph.edges:
                    break
            opinions = [self.graph.nodes[n]['opinion'] for n in new_edge]

            # choose random topic
            topic = random.randint(0, self.num_opinion - 1)
            new_weight = abs(np.array(opinions[0]) - np.array(opinions[1]))[topic]
            new_weight = 1 - (1/(1 + np.exp(-new_weight)))
            self.graph.add_edge(new_edge[0], new_edge[1], weight = new_weight
```

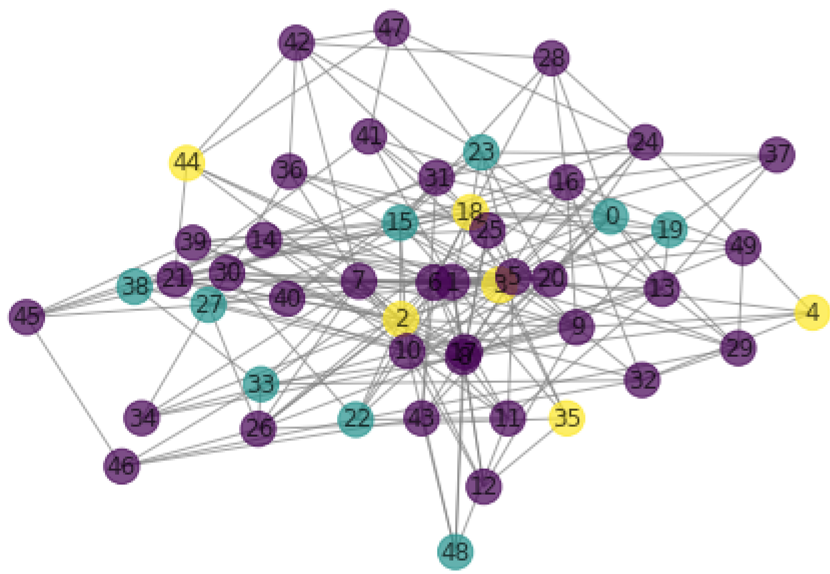
```
          self.graph.add_edge(new_edge[0], new_edge[1], weight = new_weight,
                              beta = norm.rvs(self.beta, self.beta_std, size = 1)[0],
                              gamma = norm.rvs(self.gamma, self.gamma_std, size = 1)[0])

      else:
          # Select a random edge and update node opinions and edge weight
          edge = random.choice(list(self.graph.edges))
          weight = self.graph.edges[edge]['weight']
          opinions = [self.graph.nodes[n]['opinion'] for n in edge]
          # select topic
          topic = random.randint(0, self.num_opinion - 1)
          # conservativeness of the topic
          topic_pref = 10**(-topic)
          for i in [0, 1]:
              # update the node with the topic_preference value
              self.graph.nodes[edge[i]]['opinion'][topic] = (
                  opinions[i][topic] + topic_pref * self.graph.nodes[edge[i]]['alpha'][topic] * weight * (opinions[1-i][topic] -

          self.graph.edges[edge]['weight'] = (
              weight +
              self.graph.edges[edge]['beta'] * weight * (1-weight) *
              (1 - self.graph.edges[edge]['gamma'] * abs(opinions[0][topic] - opinions[1][topic])))
          # Remove very weak connections
          if self.graph.edges[edge]['weight'] < 0.05:
              self.graph.remove_edge(*edge)
      self.step += 1

  def concensus(self):
    '''
    Calculate the polarization value:
    the polarization is the difference between the mean of people having the opinion > 0.5
    and of those having an opinion < 0.5.
    '''
    opinions = np.array([self.graph.nodes[i]['opinion'] for i in self.graph.nodes])
    opinions = opinions.T
    polarize = [np.mean(opinions[i][opinions[i] >= 0.5]) - np.mean(opinions[i][opinions[i] < 0.5]) for i in range(self.num_opinio
    if str(polarize[0]) == "nan":
      polarize[0] = np.mean(opinions[0])
    if str(polarize[1]) == "nan":
      polarize[1] = np.mean(opinions[1])
    if str(polarize[2]) == "nan":
      polarize[2] = np.mean(opinions[2])
```

```
        polarize[2] = np.mean(opinions[2])
      return polarize

    def topic_concensus(self):
      '''
      Calculate the percentage of people have strong preference on each topic
      '''
      master_list = np.array([list(self.graph.nodes[i]['opinion']).index(max(self.graph.nodes[i]['opinion'])) for i in self.graph.n
      return [np.mean(master_list == i)*100 for i in range(self.num_opinion)]

    def num_weight(self):
      '''
      Number of conenctions
      '''
      master_list = [self.graph.edges[i, j]['weight'] for i, j in self.graph.edges]
      return len(master_list)

    def avg_weight(self):
      '''
      Average connection weights
      '''
      master_list = [self.graph.edges[i, j]['weight'] for i, j in self.graph.edges]
      return np.mean(master_list)

    def get_num_opinion(self):
      '''
      get number of opinion
      '''
      return self.num_opinion


# simulate
sim = SocialDynamicsSimulation()
sim.initialize()
plt.figure()
sim.observe_fav_topic()
for j in range(sim.get_num_opinion()):
  sim.observe_topic(j)

for i in range(5):
    for k in range(20000):
        sim.update()
```

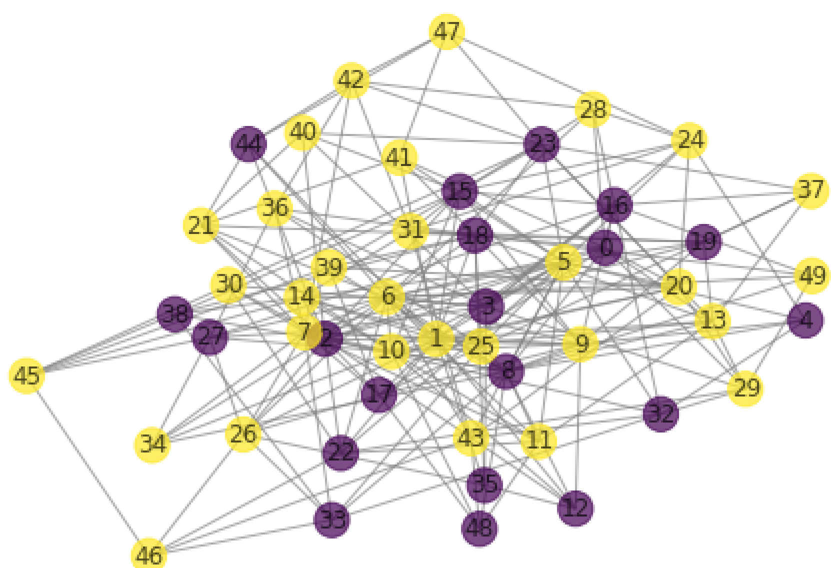
```
        sim.update()
    plt.figure()
    sim.observe_fav_topic()
    for j in range(sim.get_num_opinion()):
      sim.observe_topic(j)
```

```
% likes topic 0 : 70.0
% likes topic 1 : 18.0
% likes topic 2 : 12.0
```

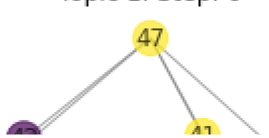
Favorite topic discussion. Step: 0

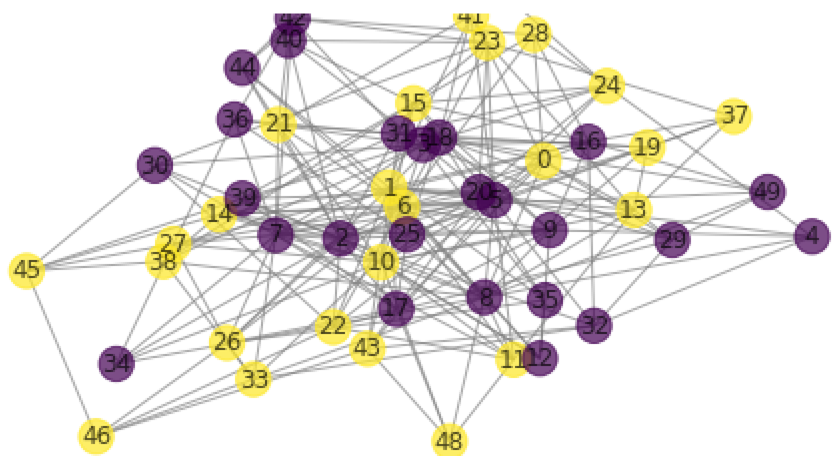


Topic 0. Step: 0

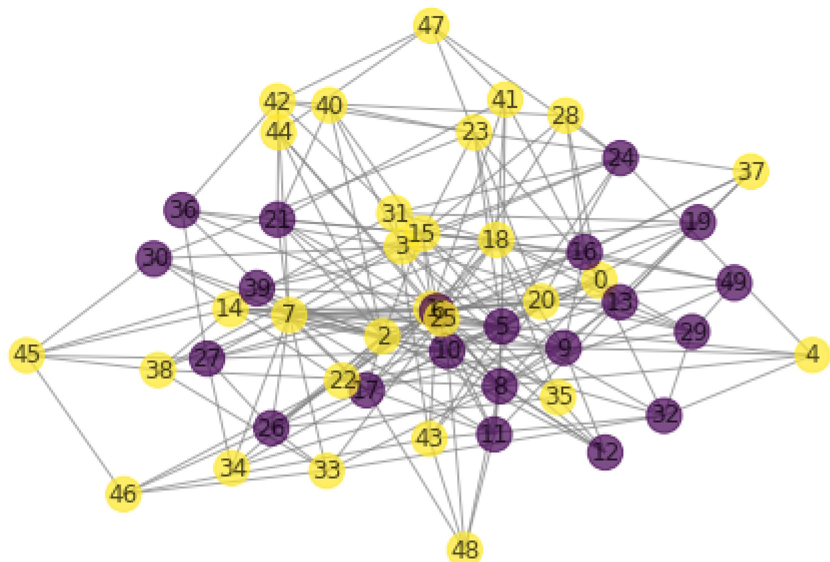


Topic 1. Step: 0



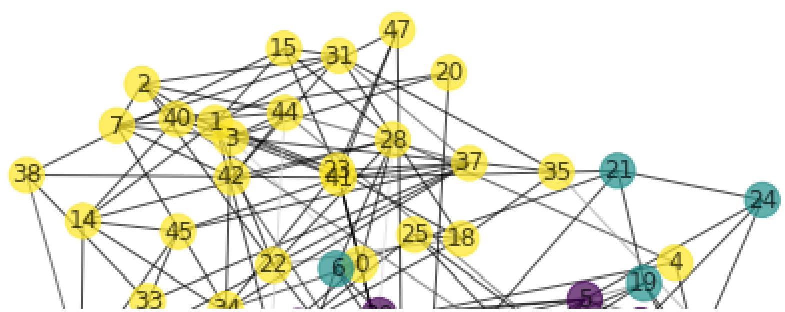


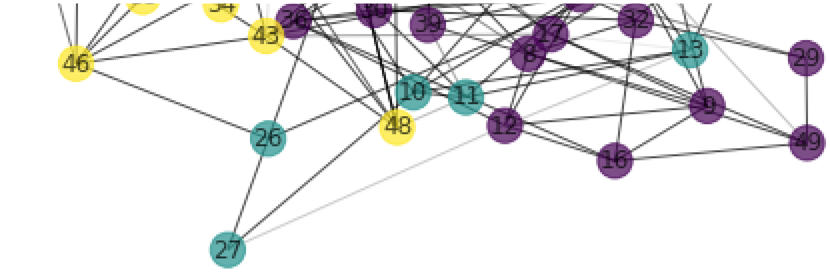
Topic 2. Step: 0



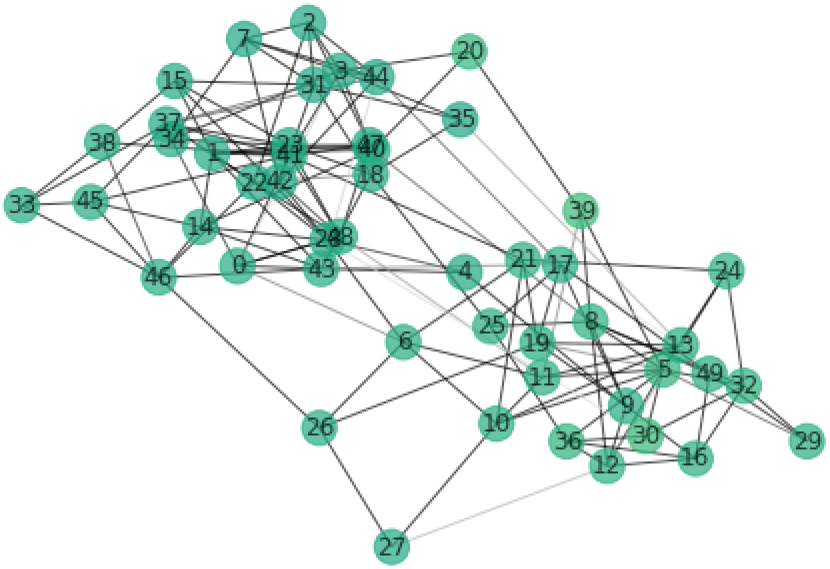
```
% likes topic 0 : 24.0
% likes topic 1 : 18.0
% likes topic 2 : 57.99999999999999
```

Favorite topic discussion. Step: 20000

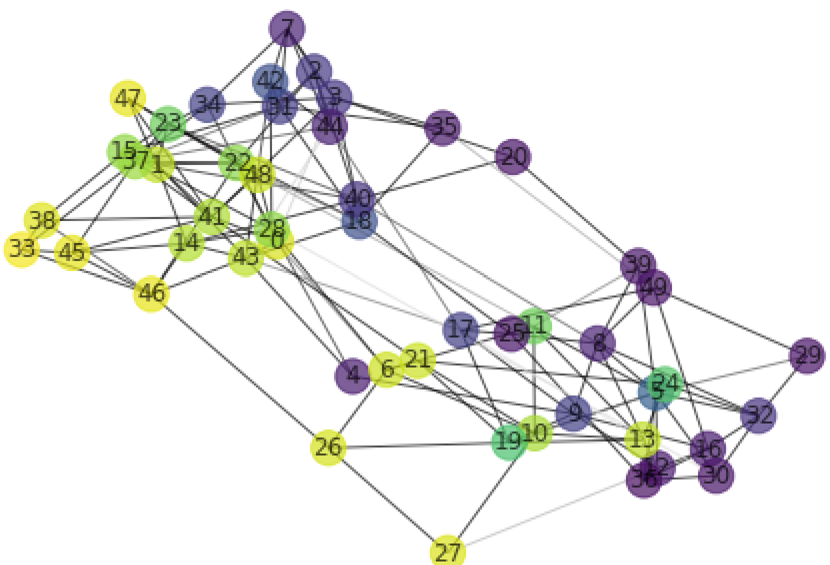




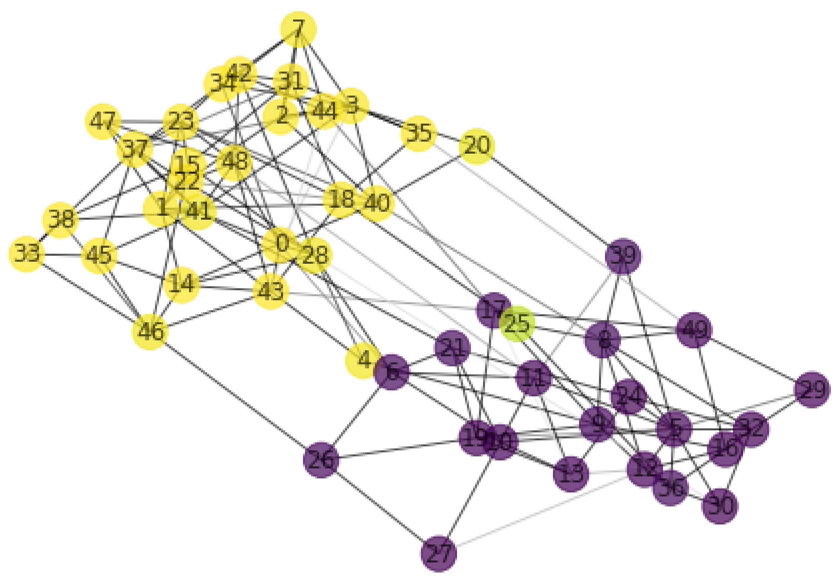
Topic 0. Step: 20000



Topic 1. Step: 20000

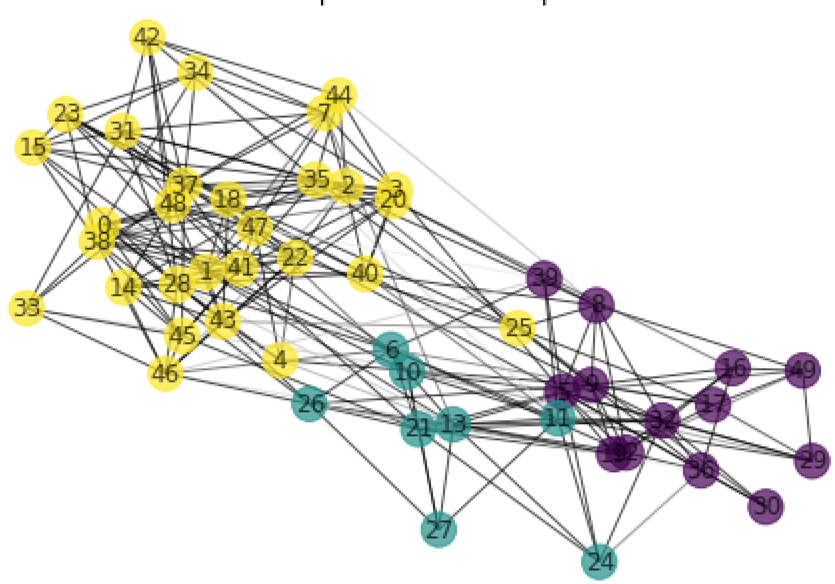


Topic 2. Step: 20000

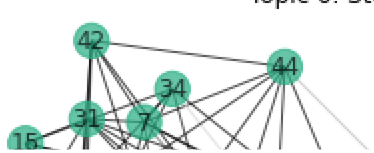


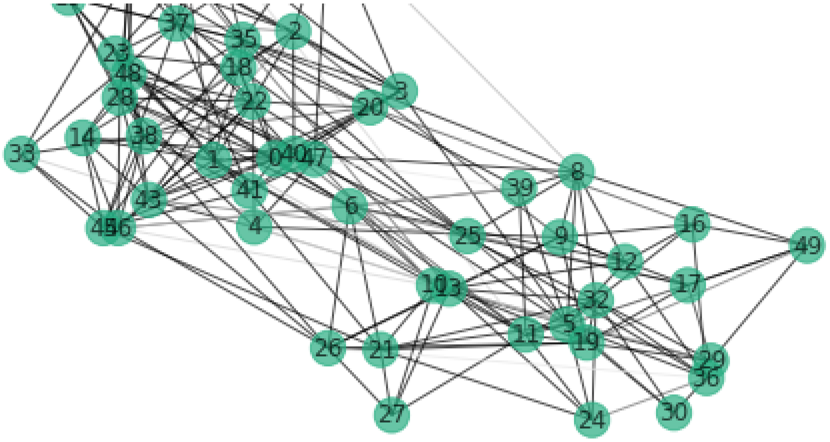
```
% likes topic 0 : 26.0
% likes topic 1 : 16.0
% likes topic 2 : 57.99999999999999
```

Favorite topic discussion. Step: 40000



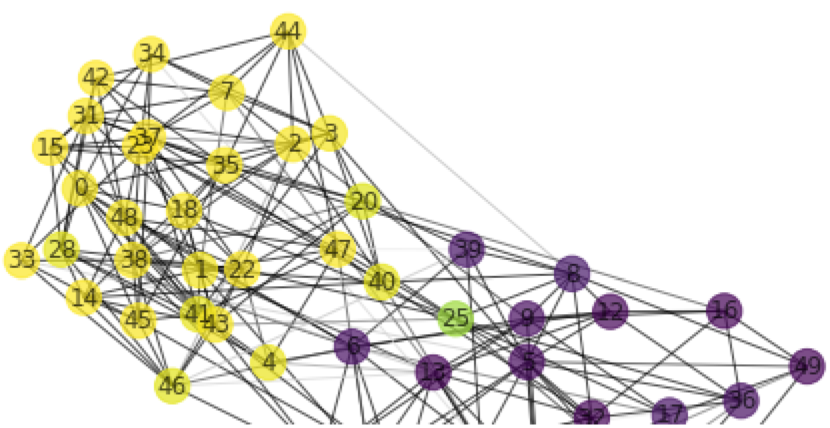
Topic 0. Step: 40000

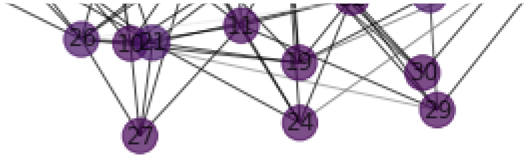




Topic 1. Step: 40000

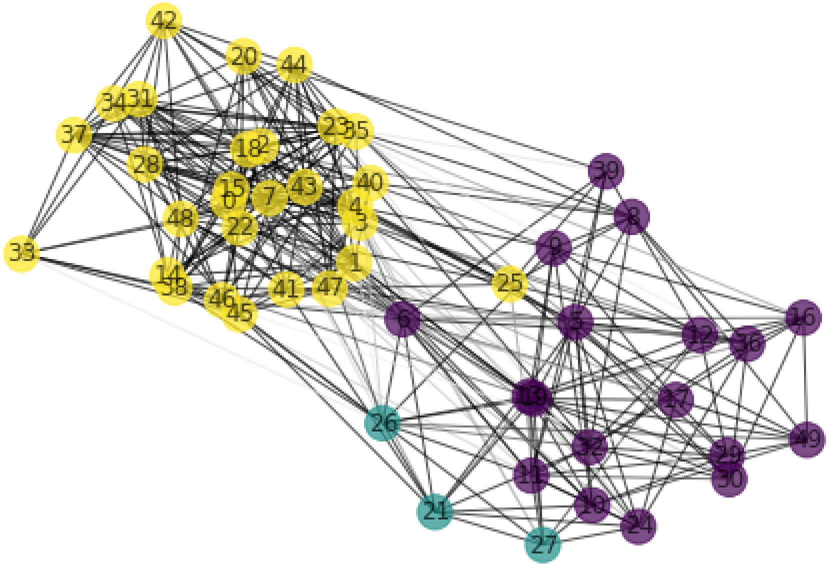
Topic 2. Step: 40000



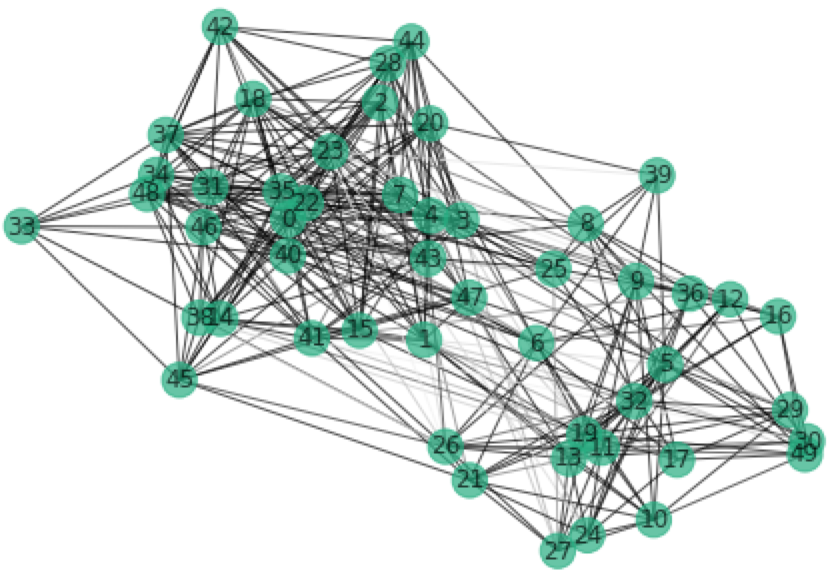


```
% likes topic 0 : 36.0
% likes topic 1 : 6.0
% likes topic 2 : 57.99999999999999
```

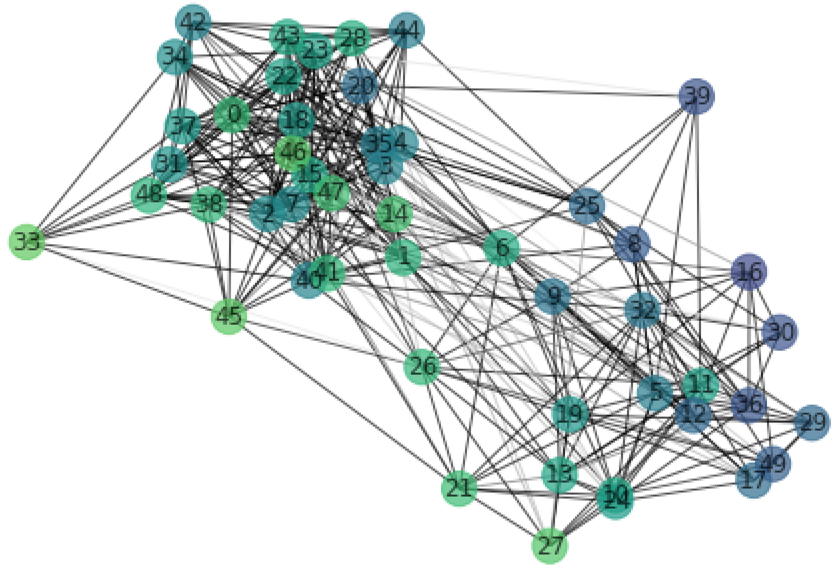
Favorite topic discussion. Step: 60000



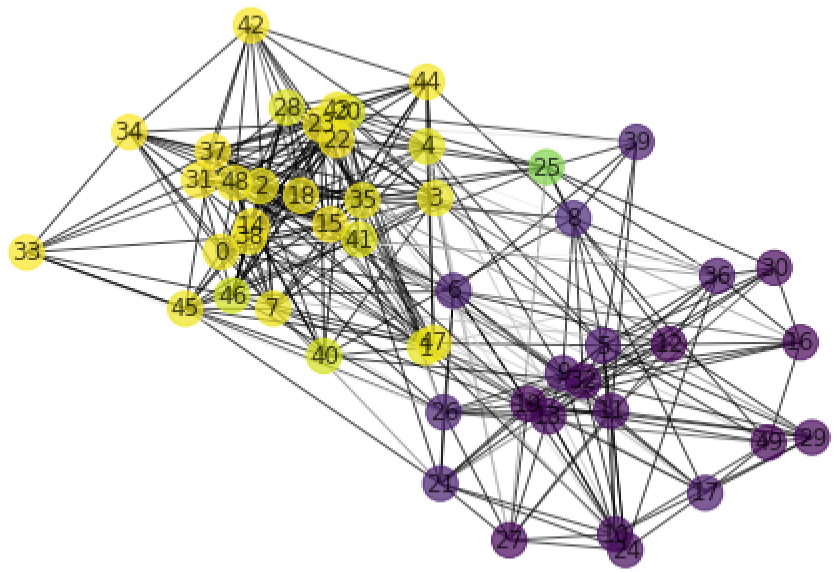
Topic 0. Step: 60000



Topic 1. Step: 60000

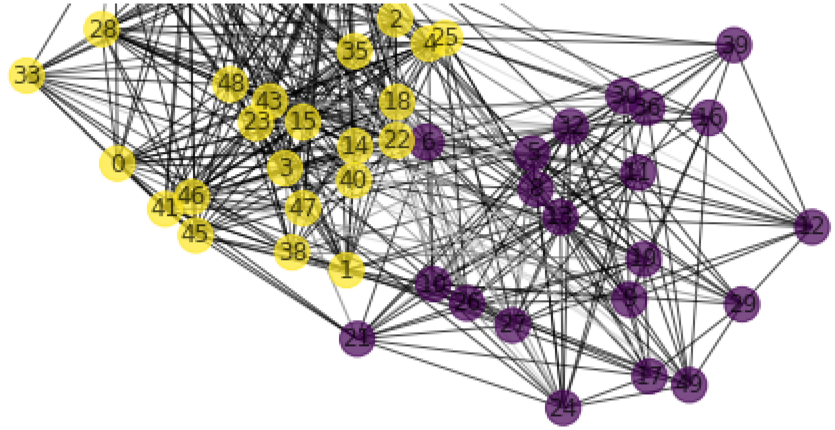


Topic 2. Step: 60000

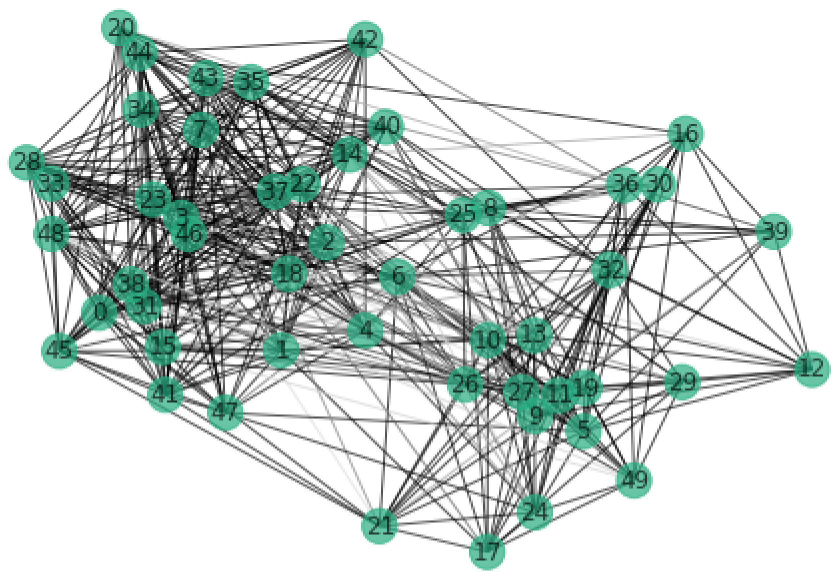


```
% likes topic 0 : 42.0
% likes topic 1 : 0.0
% likes topic 2 : 57.99999999999999
```

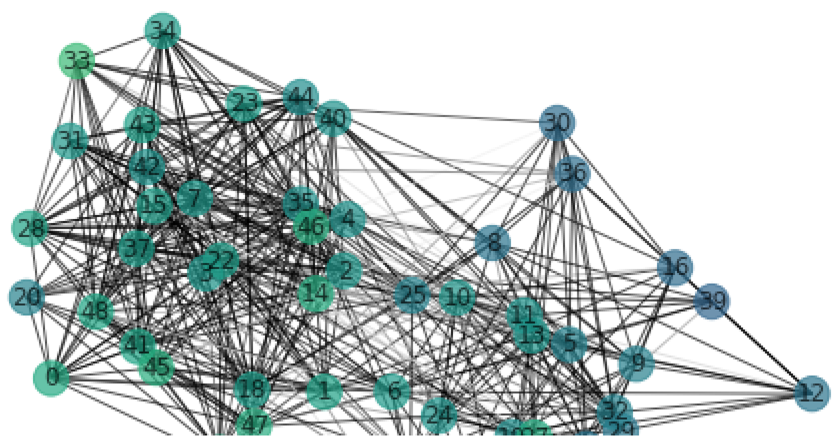
Favorite topic discussion. Step: 80000

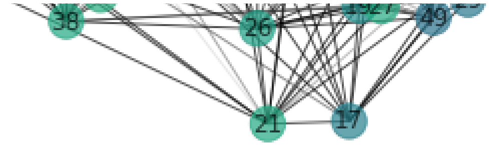


Topic 0. Step: 80000

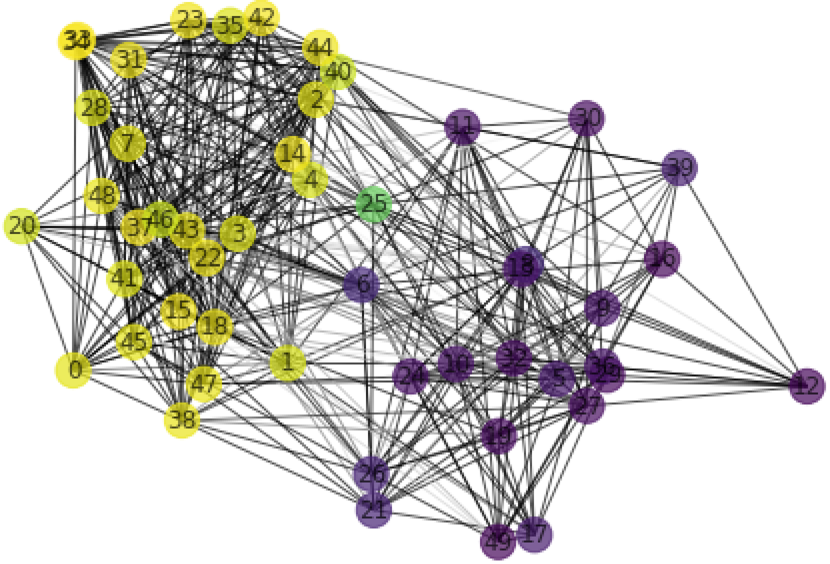


Topic 1. Step: 80000



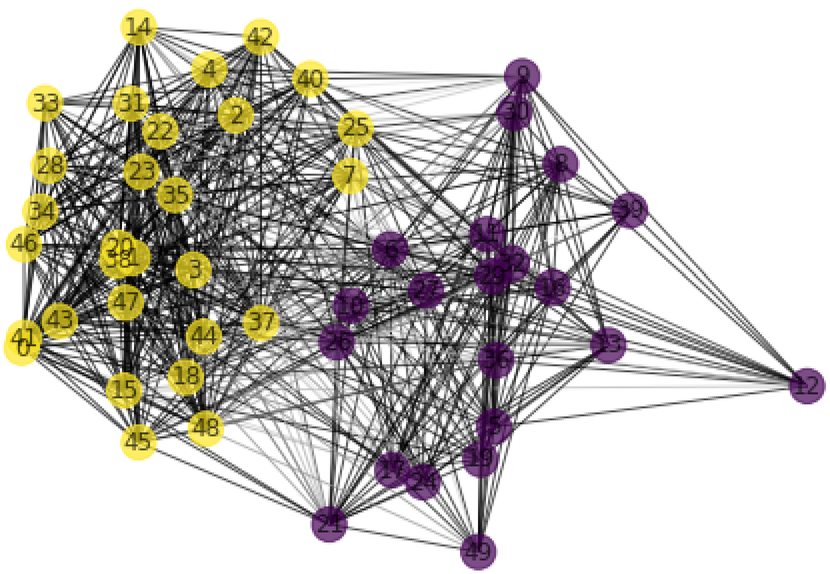


Topic 2. Step: 80000



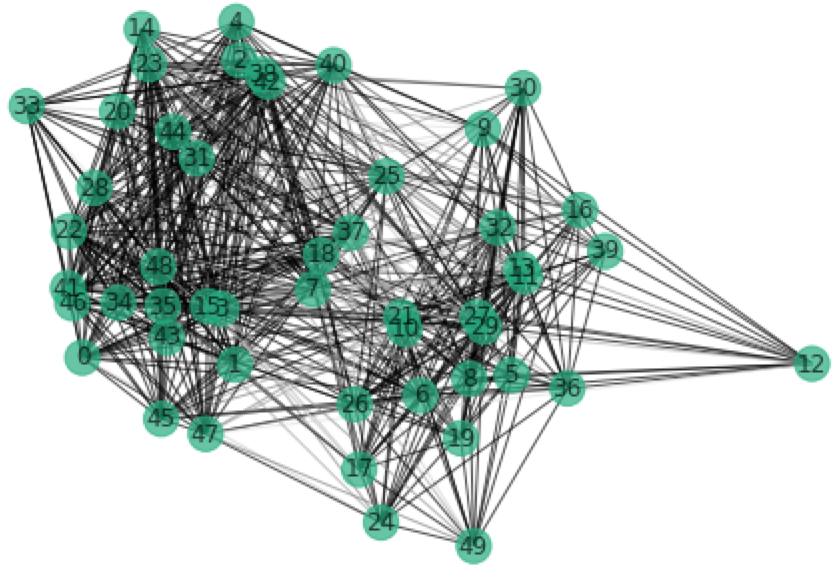
```
% likes topic 0 : 42.0
% likes topic 1 : 0.0
% likes topic 2 : 57.99999999999999
```

Favorite topic discussion. Step: 100000

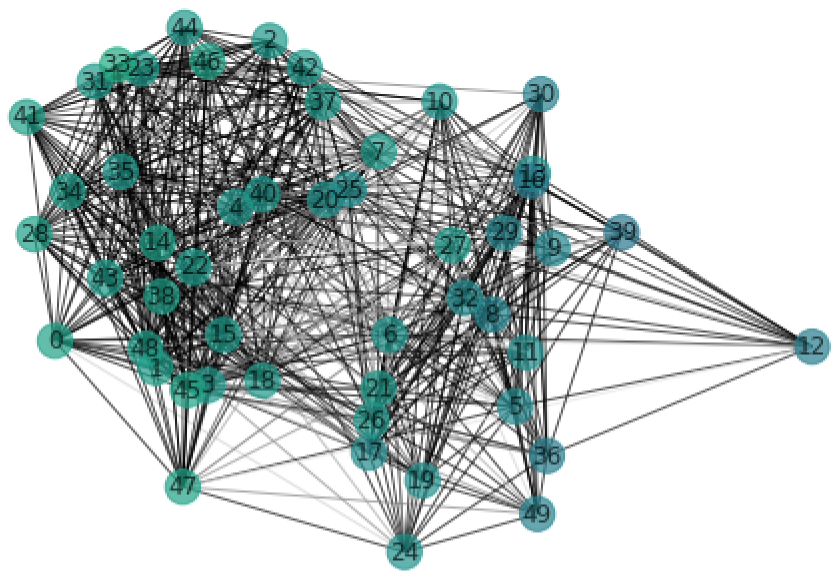


Topic 0. Step: 100000

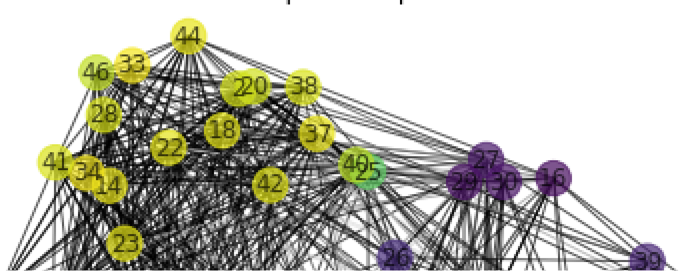
Topic 0. Step: 100000

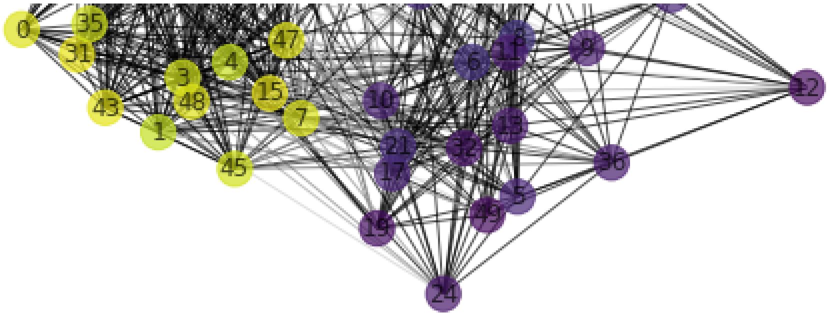


Topic 1. Step: 100000



Topic 2. Step: 100000

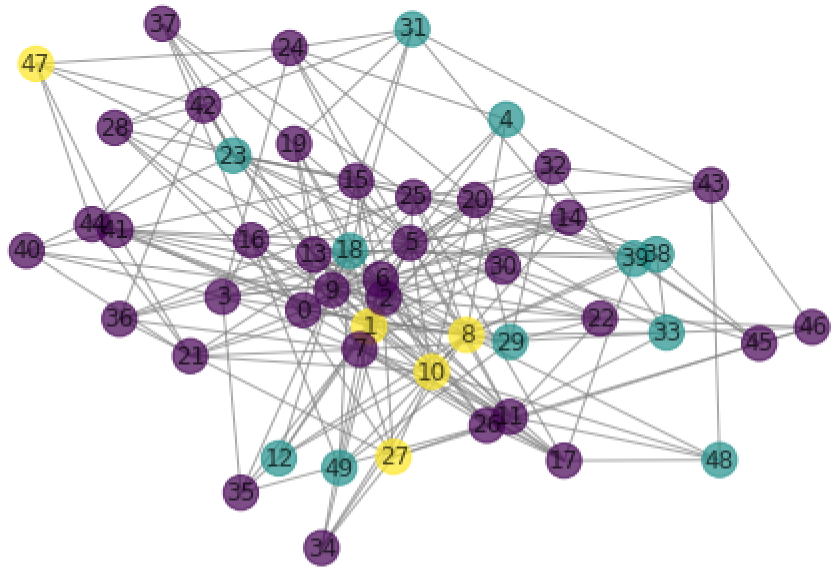




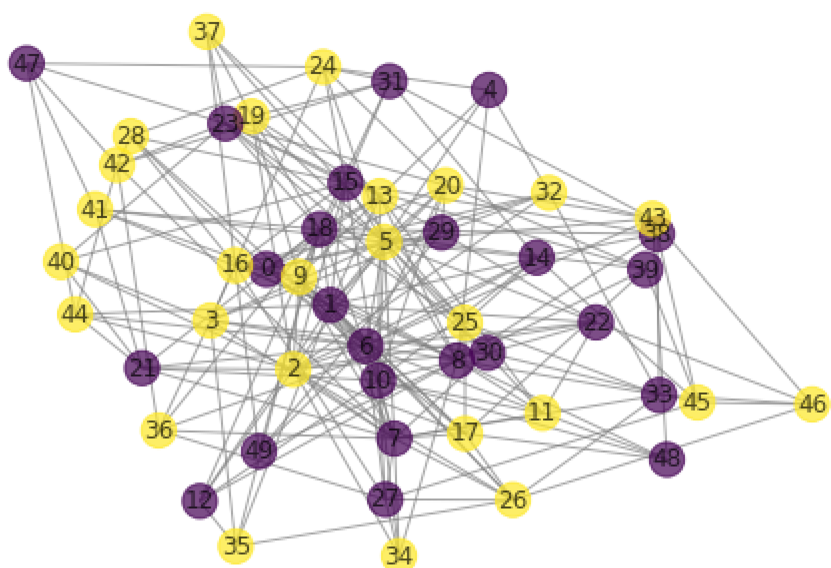

```
# simulate and track variables
sim = SocialDynamicsSimulation()
sim.initialize()
plt.figure()
sim.observe_fav_topic()
for j in range(sim.get_num_opinion()):
  sim.observe_topic(j)

# track all the metrics
track_concensus = [sim.concensus()]
track_topic = [sim.topic_concensus()]
track_num_weight = [sim.num_weight()]
track_avg_weight = [sim.avg_weight()]

for i in range(5):
    print(i)
    for k in range(30000):
        sim.update()
        if (k+1)%3000 == 0:
          track_concensus.append(sim.concensus())
          track_topic.append(sim.topic_concensus())
          track_num_weight.append(sim.num_weight())
          track_avg_weight.append(sim.avg_weight())
```

```
% likes topic 0 : 68.0
% likes topic 1 : 22.0
% likes topic 2 : 10.0
```

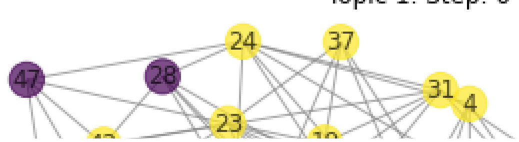
Favorite topic discussion. Step: 0

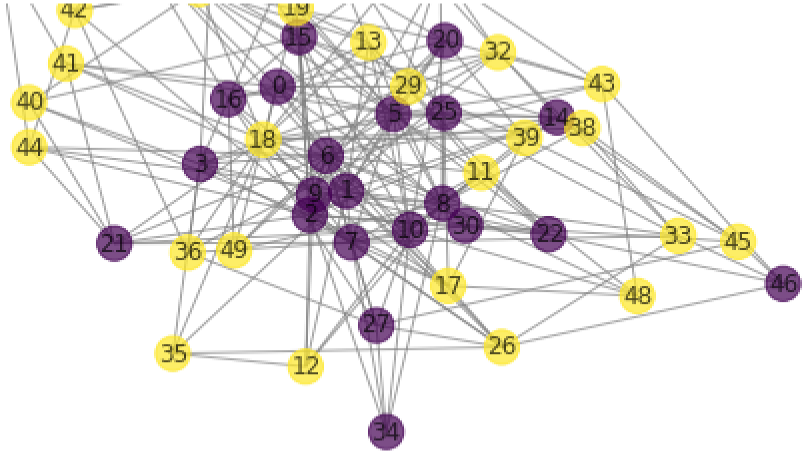


Topic 0. Step: 0

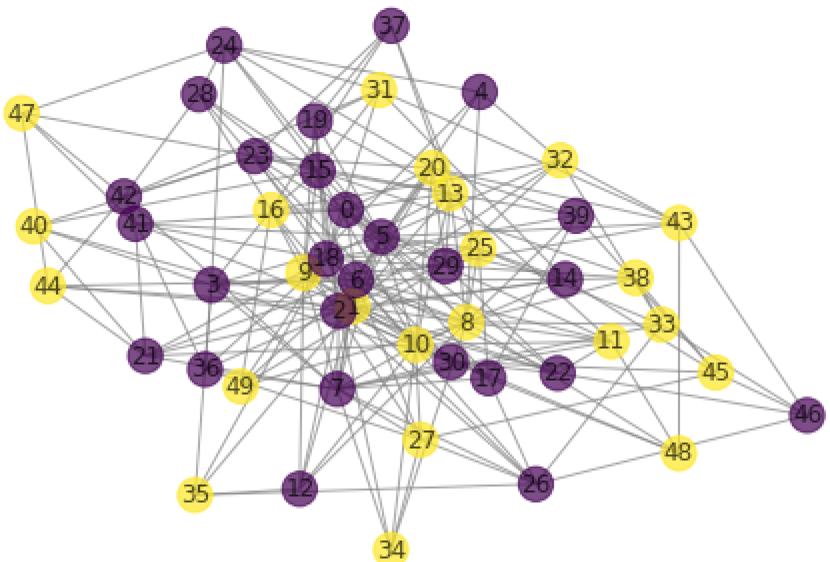


Topic 1. Step: 0





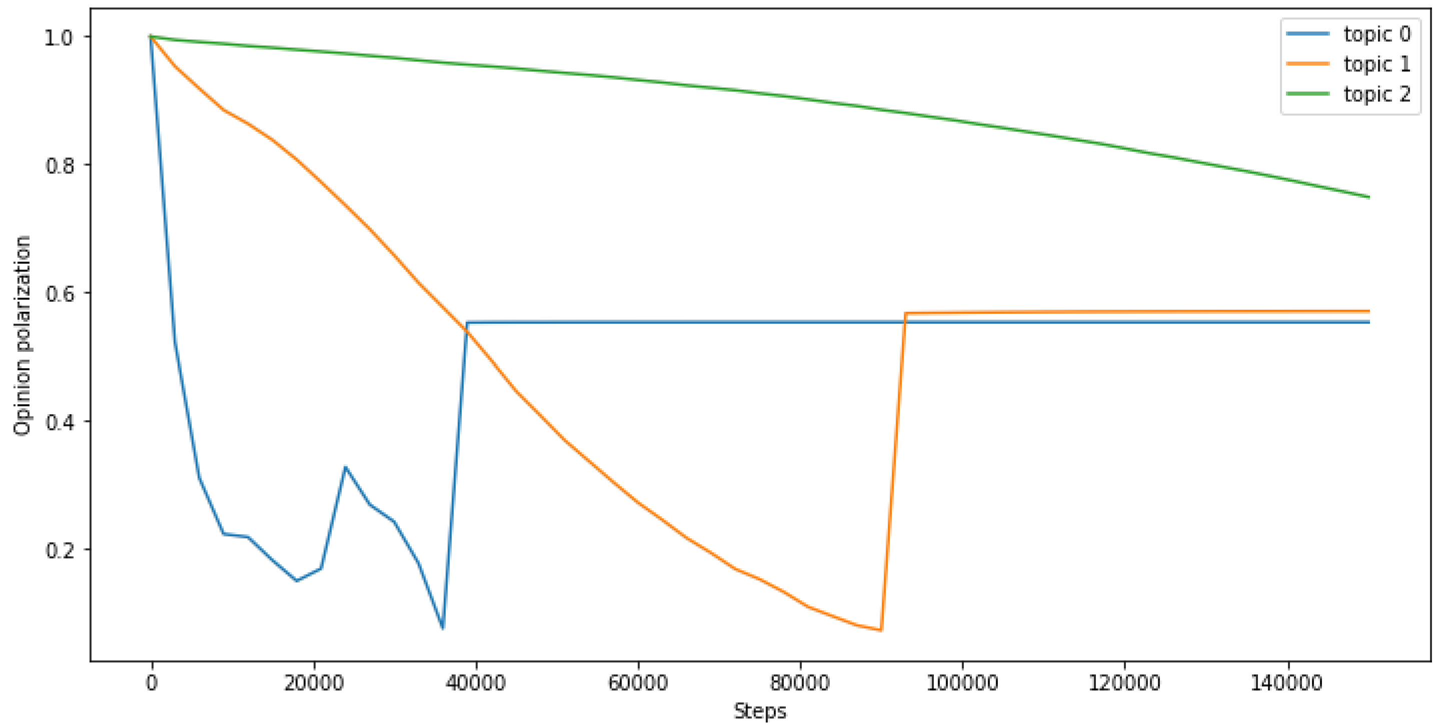
Topic 2. Step: 0



```
0
1
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:3335: RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:161: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
2
3
4
```

```
# visualize the result
```

```python

plt.figure(figsize = (12, 6))
for i in range(sim.get_num_opinion()):
  plt.plot(np.array(range(len(np.array(track_concensus).T[0])))*3000, np.array(track_concensus).T[i], label = "topic " + str(i))
plt.legend()
plt.xlabel("Steps")
plt.ylabel("Opinion polarization")
plt.show()
```



Model simulation for multiple trials

```python
# simulate the whole code
num_trial = 30
# track results over trials
total_concensus = []
total_topic = []
```

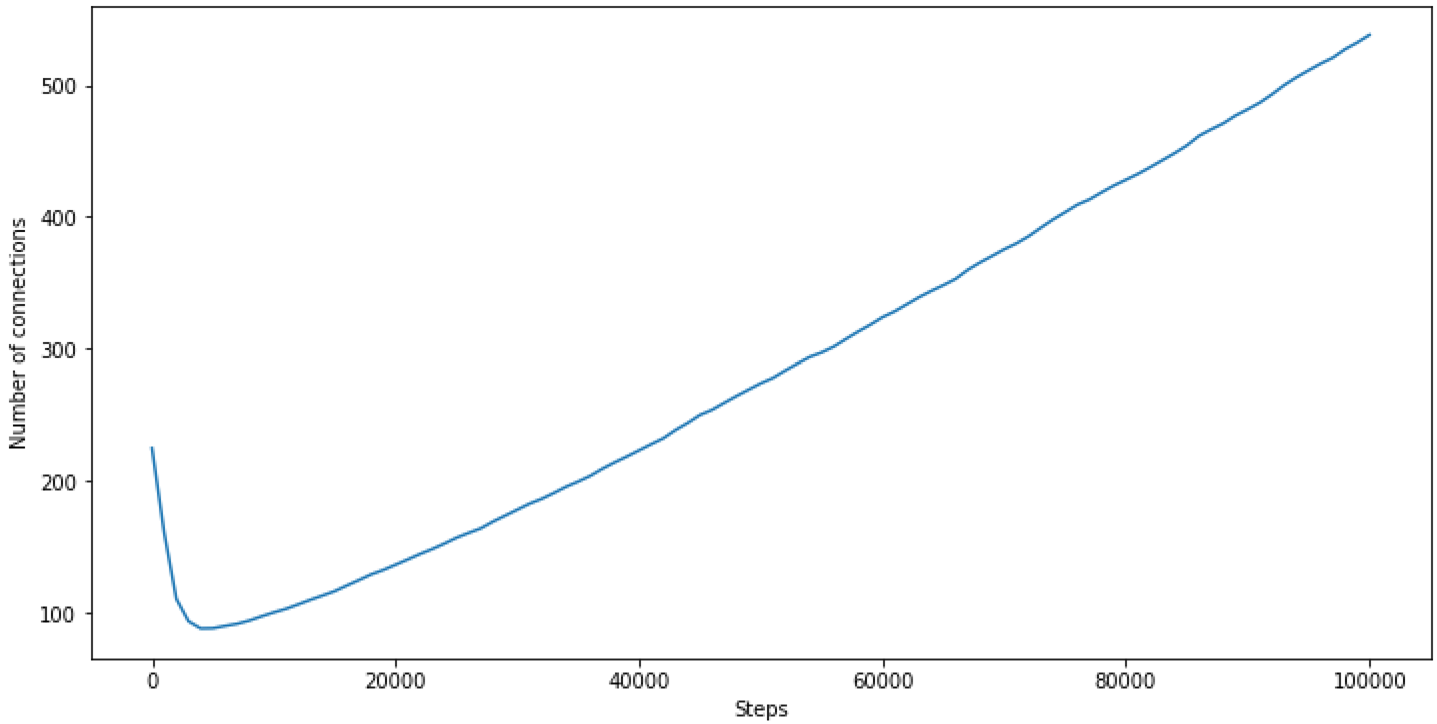
```
total_num_weight = []
total_avg_weight = []
for i in range(num_trial):
  sim = SocialDynamicsSimulation()
  sim.initialize()

  track_concensus = [sim.concensus()]
  track_topic = [sim.topic_concensus()]
  track_num_weight = [sim.num_weight()]
  track_avg_weight = [sim.avg_weight()]

  for i in range(5):
      for k in range(20000):
          sim.update()
          if (k+1)%1000 == 0:
            # track every 1000 steps
            track_concensus.append(sim.concensus())
            track_topic.append(sim.topic_concensus())
            track_num_weight.append(sim.num_weight())
            track_avg_weight.append(sim.avg_weight())
  total_concensus.append(track_concensus)
  total_topic.append(track_topic)
  total_num_weight.append(track_num_weight)
  total_avg_weight.append(track_avg_weight)
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:3335: RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:161: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

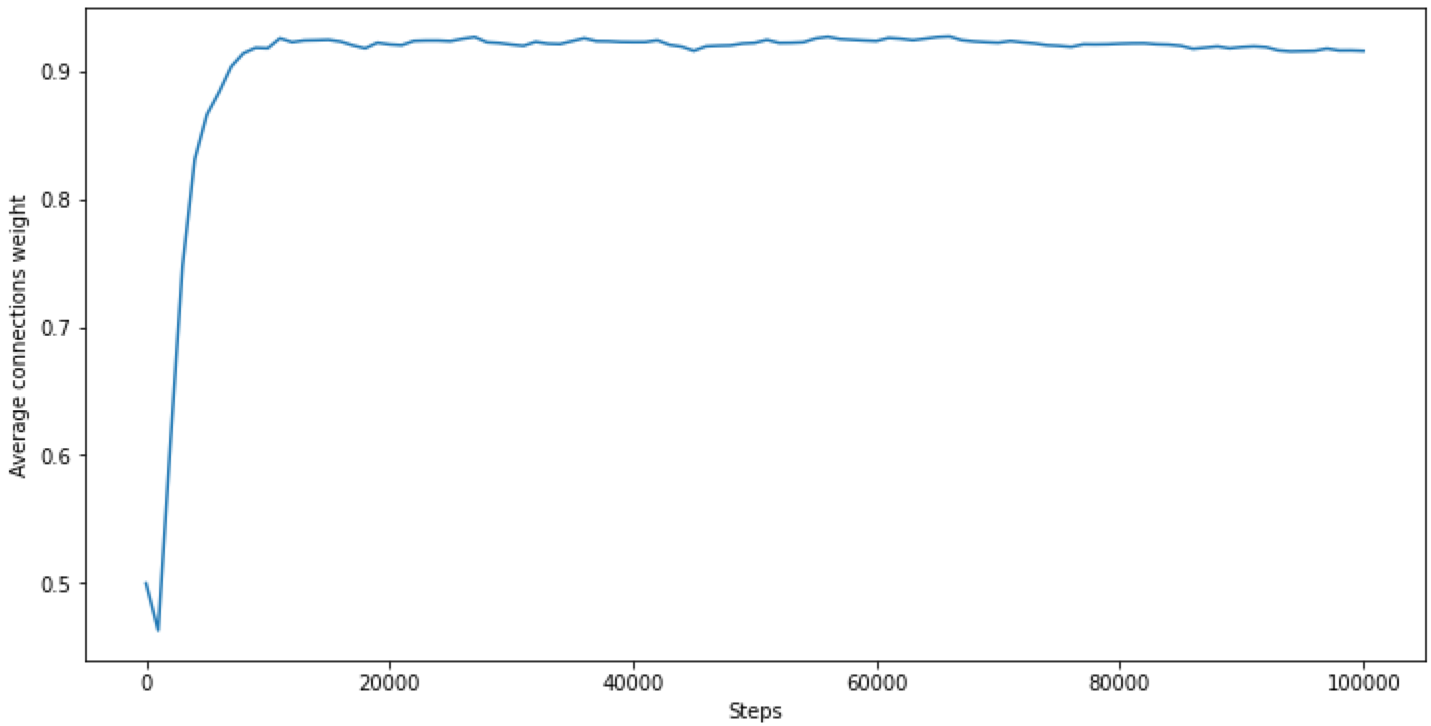
```
# get the mean of the trials
final_concensus = np.mean(np.array(total_concensus), axis = 0)
final_topic = np.mean(np.array(total_topic), axis = 0)
final_num_weight = np.mean(total_num_weight, axis = 0)
final_avg_weight = np.mean(total_avg_weight, axis = 0)
```

```
# visualize the number of connections
plt.figure(figsize = (12, 6))
plt.plot(np.array(range(len(final_num_weight)))*1000, final_num_weight)
plt.xlabel("Steps")
plt.ylabel("Number of connections")
```

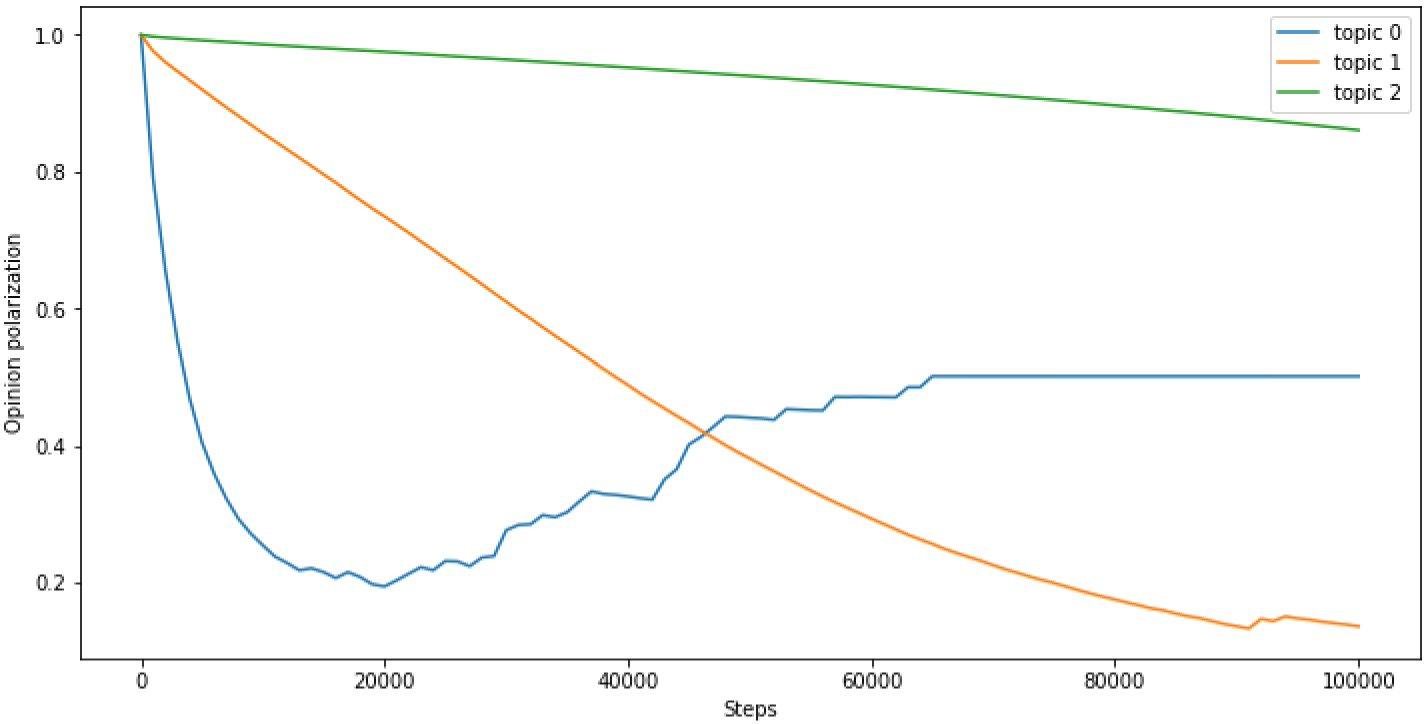
```
plt.ylabel("Number of connections")
plt.show()
```



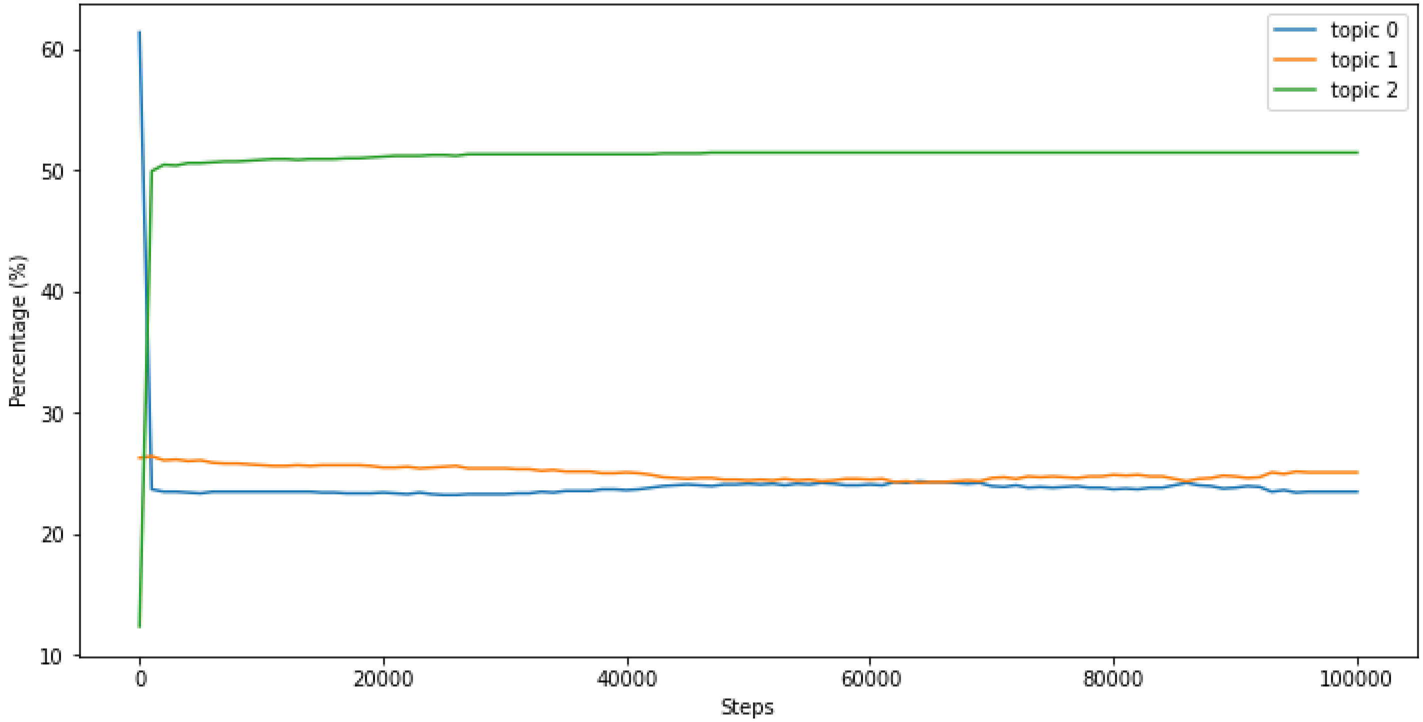

```
# visulize the average weight
plt.figure(figsize = (12, 6))
plt.plot(np.array(range(len(final_avg_weight)))*1000, final_avg_weight)
plt.xlabel("Steps")
plt.ylabel("Average connections weight")
plt.show()
```




```
# visulize the polarization values
plt.figure(figsize = (12, 6))
for i in range(sim.get_num_opinion()):
  plt.plot(np.array(range(len(final_concensus)))*1000, np.array(final_concensus).T[i], label = "topic " + str(i))
plt.legend()
plt.xlabel("Steps")
plt.ylabel("Opinion polarization")
plt.show()
```




```
# visulize the preferred topic
plt.figure(figsize = (12, 6))
for i in range(sim.get_num_opinion()):
  plt.plot(np.array(range(len(final_concensus)))*1000, np.array(final_topic).T[i], label = "topic " + str(i))
plt.legend()
plt.xlabel("Steps")
plt.ylabel("Percentage (%)")
plt.show()
```



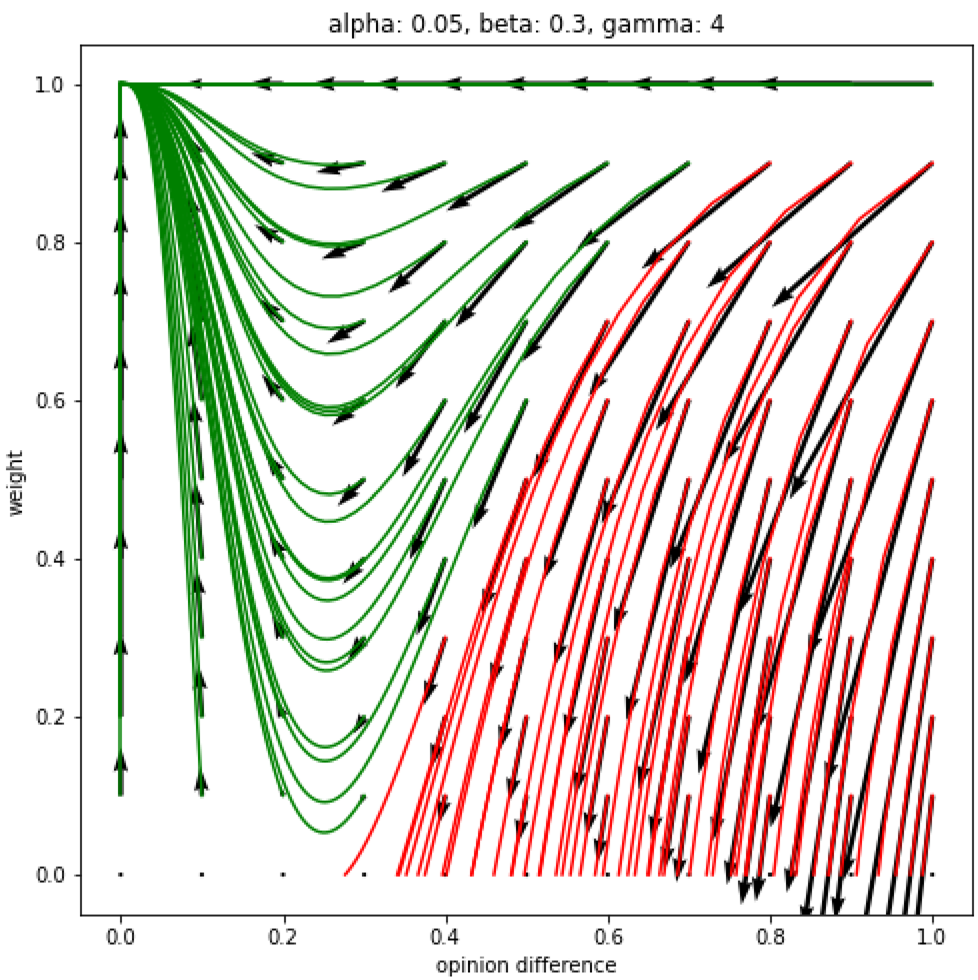
Local Analysis for 2 nodes

```
# local simulation function
def LocalSimulation(opinion_diff, weight = 1, num_opinion = 3, alpha = 0.07, beta = 0.3, gamma = 4):
  track_opinion = [opinion_diff]
  track_weight = [weight]
  for i in range(1000):
    # these formula are in the analysis pdf: measuring the changes in the next opinion and weight basd on current values
    track_opinion.append(abs(track_opinion[-1] * (1-2*alpha*track_weight[-1])))
    track_weight.append(track_weight[-1] + beta*track_weight[-1]*(1-track_weight[-1])*(1-gamma*track_opinion[-1]))
  return track_opinion, track_weight

```

```
def vector_field(alpha=0.03, beta=0.3, gamma=4, plot_option = True, multi_plot = False):
```

```
#2D vector field plots
# Create grid coordinates
opp_diff = np.linspace(0, 1, 11)
edge_weight = np.linspace(0, 1, 11)
opp_diff_grid, edge_weight_grid = np.meshgrid(opp_diff, edge_weight)
if plot_option and not multi_plot:
  plt.figure(figsize=(8, 8))
# Compute vector field
vector_x = abs(1-2*alpha*edge_weight_grid)*opp_diff_grid - opp_diff_grid
vector_y = beta*edge_weight_grid*(1-edge_weight_grid)*(1-gamma*opp_diff_grid)
converge, diverge, other = 0, 0, 0
for a in opp_diff:
  for b in edge_weight:
    x_axis, y_axis = LocalSimulation(a, b, alpha = alpha, beta = beta, gamma = gamma)

    if y_axis[-1] < 0.05:
      if plot_option:
        plt.plot(x_axis, y_axis, color = "red")
      diverge += 1
    elif y_axis[-1] > 0.95:
      if plot_option:
        plt.plot(x_axis, y_axis, color = "green")
      converge += 1
    else:
      if plot_option:
        plt.plot(x_axis, y_axis, color = "blue")
      other += 1
# Plot vector field
if plot_option:
  plt.quiver(opp_diff_grid, edge_weight_grid, vector_x, vector_y, scale=0.5)
  plt.title('alpha: {}, beta: {}, gamma: {}'.format(alpha, beta, gamma))
  plt.xlabel('opinion difference')
  plt.ylabel('weight')
  plt.show()
else:
  return converge/np.sum([converge, diverge, other]), diverge/np.sum([converge, diverge, other])
```

```
vector_field(alpha = 0.05, beta = 0.3, gamma = 4, plot_option=1)
```




```

```

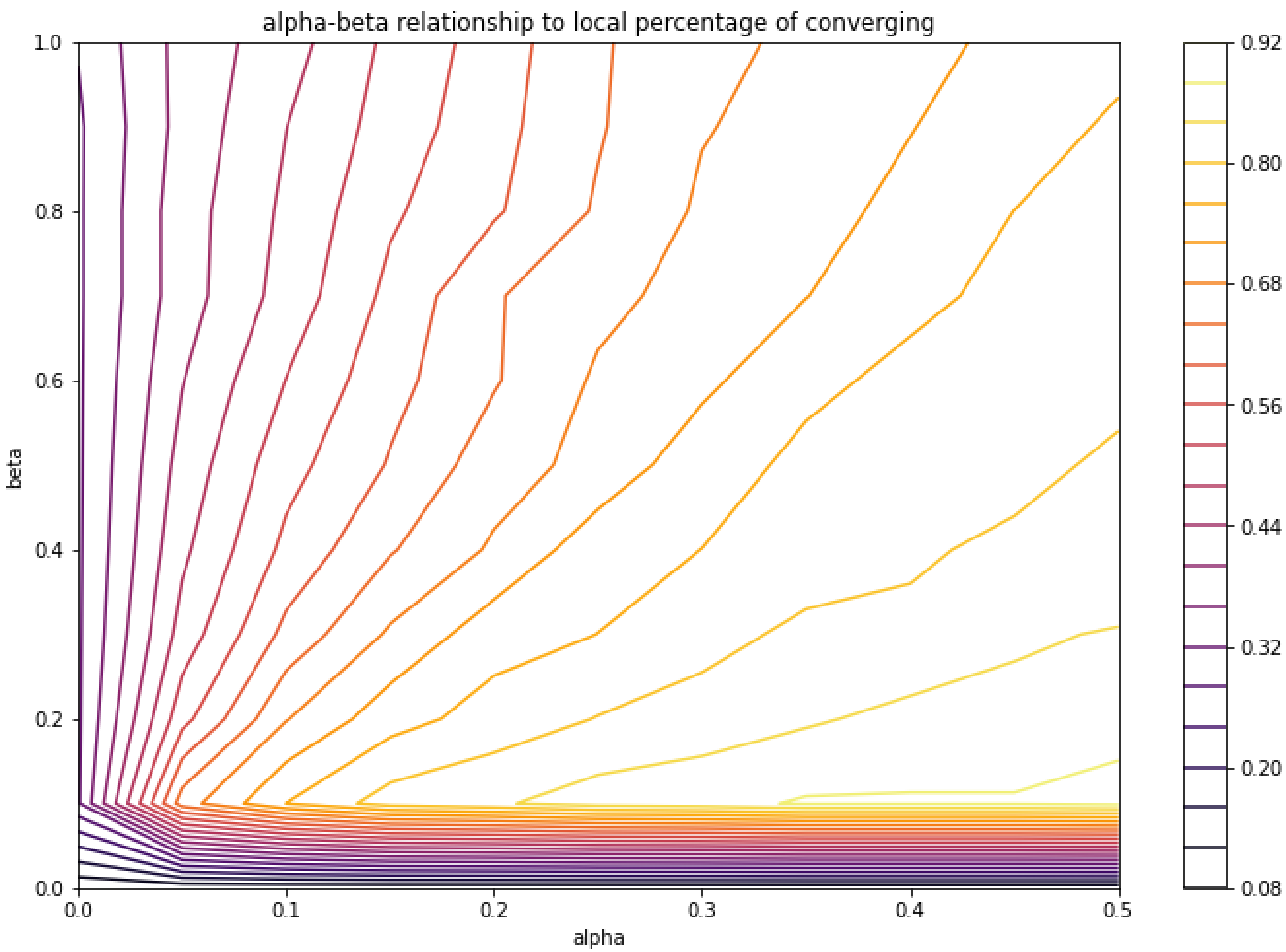
Contour plot for local analysis

```
# initlize some values for alpha, beta, gamma
points = 11
a = np.linspace(0, 0.5, points)
b = np.linspace(0, 1, points)
g = np.linspace(0, 5, points)
```

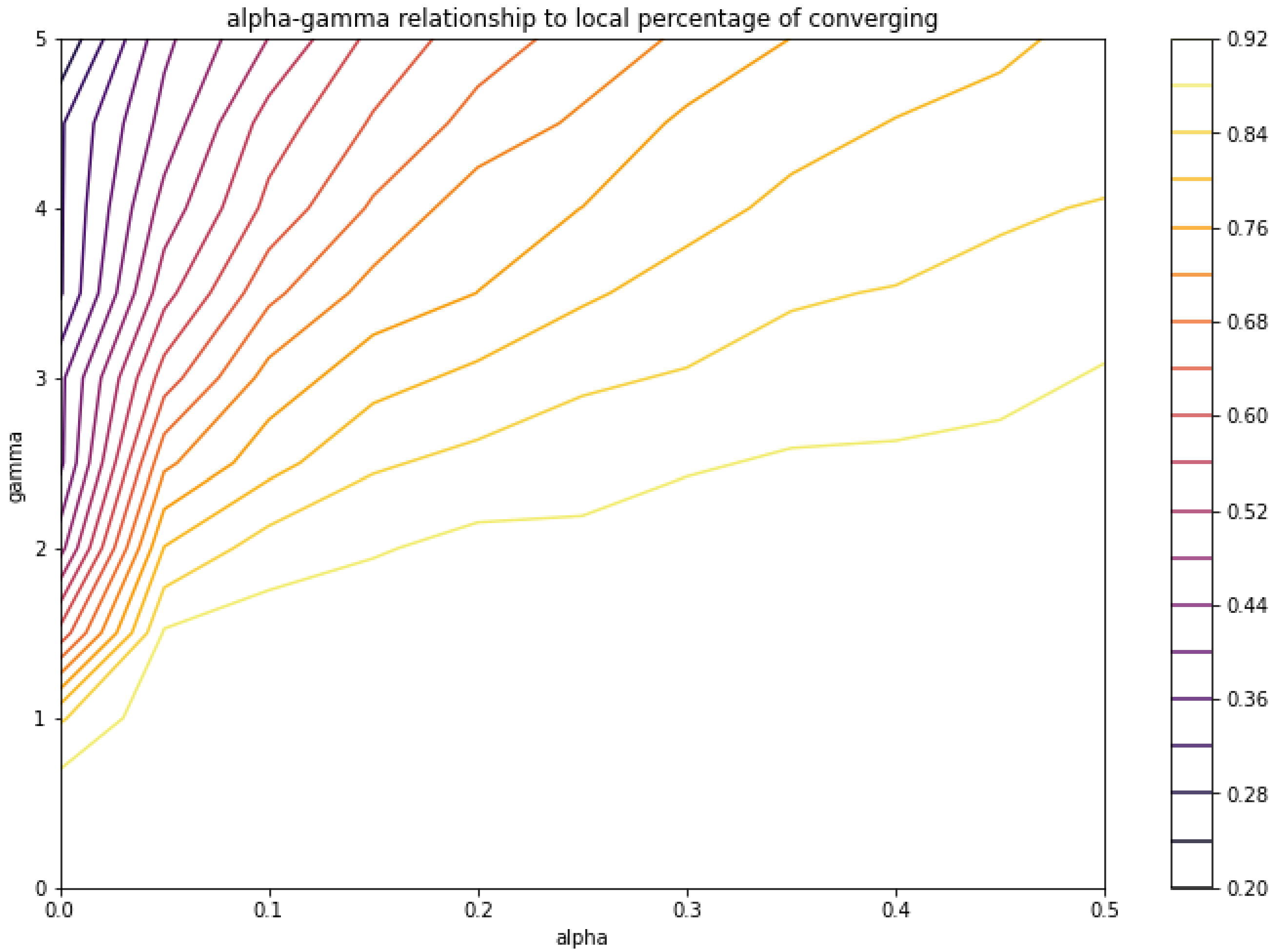
```
#visulize the contour plot of these parameters with the function value is the convergence rate
def contour(points, alpha = [0], beta = [0], gamma = [0]):
  a, b, g = alpha, beta, gamma
  if sum(g) == 0:
    a, b = a, b
  elif sum(b) == 0:
    a, b = a, g
  elif sum(a) == 0:
    a, b = b, g

  # shape for contour plot
  a, b = np.meshgrid(a, b)
  a, b = a.flatten(), b.flatten()
  track_converge = []
  track_diverge = []
  for i in range(points**2):
    val_1, val_2 = a[i], b[i]
    # simulate LocalAnalysis to find the convergence rate
    if sum(gamma) == 0:
      c1, d1 = vector_field(alpha = val_1, beta = val_2, plot_option=0)
    elif sum(beta) == 0:
      c1, d1 = vector_field(alpha = val_1, gamma = val_2, plot_option=0)
    elif sum(alpha) == 0:
      c1, d1 = vector_field(beta = val_1, gamma = val_2, plot_option=0)
    track_converge.append(c1)
    track_diverge.append(d1)

  # outcome
  Z = np.array(track_converge).reshape(points, points)
  plt.figure(figsize = (12, 8))
  plt.contour(a.reshape(points, points), b.reshape(points, points), np.array(track_converge).reshape(points, points), 20, cmap='inf
  plt.colorbar()

  # labels
  if sum(gamma) == 0:
    plt.xlabel("alpha")
    plt.ylabel("beta")
    plt.title("alpha-beta relationship to local percentage of converging")
  elif sum(beta) == 0:
    plt.xlabel("alpha")
    plt.ylabel("gamma")
```

```
    plt.title("alpha-gamma relationship to local percentage of converging")
  elif sum(alpha) == 0:
    plt.xlabel("beta")
    plt.ylabel("gamma")
    plt.title("beta-gamma relationship to local percentage of converging")
```

```
# visualize the plot with alpha-beta parameters
contour(points, alpha = a, beta = b)
```

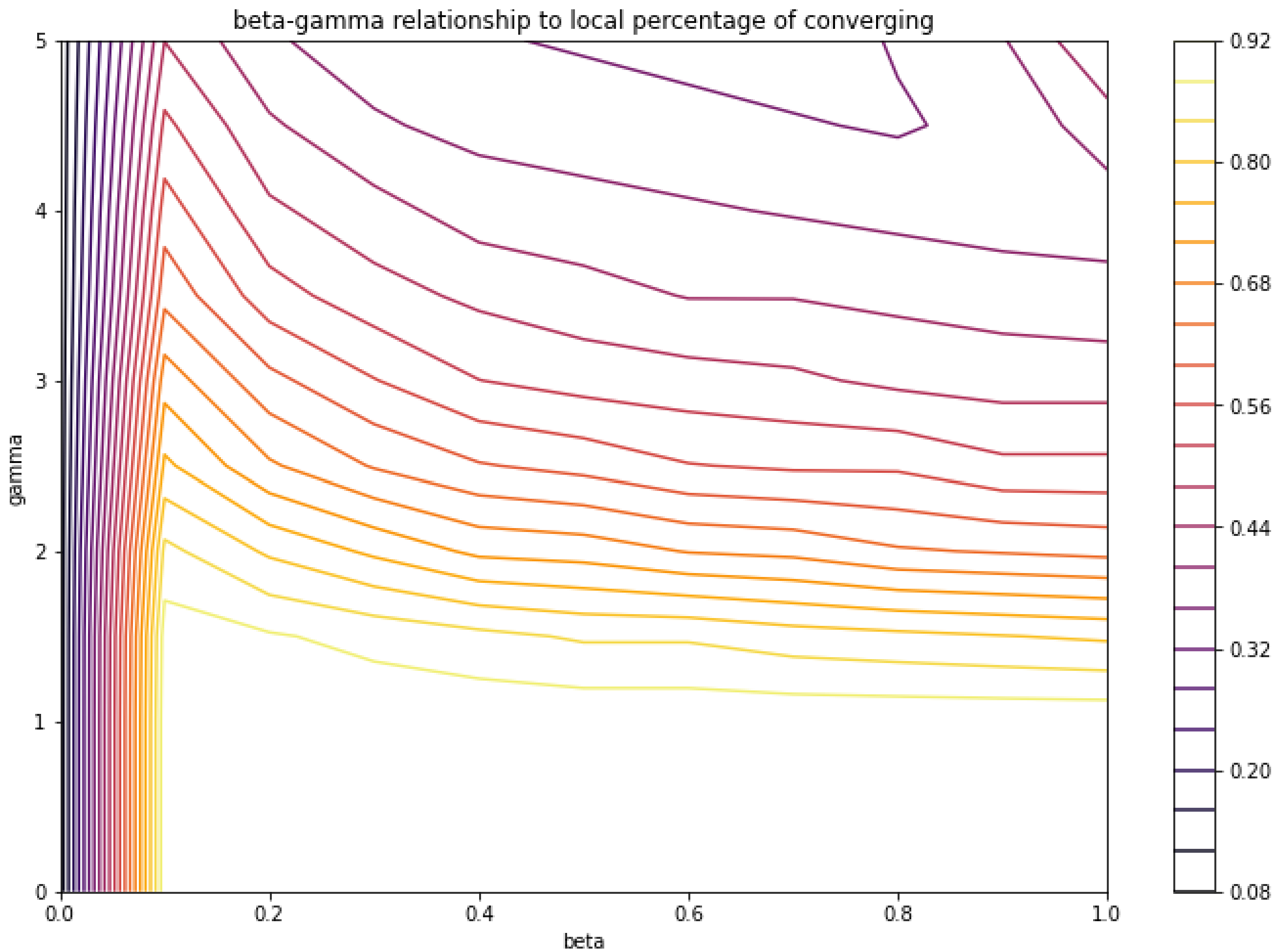



```
# visualize the plot with alpha-gamma parameters
contour(points, alpha = a, gamma = g)
```




```
# visualize the plot with beta-gamma parameters
contour(points, beta = b, gamma = g)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: RuntimeWarning: overflow encountered in double_scalars
  import sys
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: RuntimeWarning: overflow encountered in double_scalars
```




```
1
```