This network represents the neural connections of the Caenorhabditis elegans nematode (1986).
D.J. Watts and S.H. Strogatz, "Collective dynamics of 'small-world' networks." Nature 393, 440-442 (1998).

Each node is a neuron.

Each edge is a synaptic link.

In [1]:

```
# import libraries
import networkx as nx
import numpy as np
from matplotlib import pyplot as plt, animation
from networkx.drawing.nx_pydot import graphviz_layout
import copy
import time
import random
import itertools
from itertools import islice

%matplotlib notebook

# Load graph using NetworkX
def load_graph_from_file(file):
    G = nx.read_gml(file)
    return G
# G = nx.complete_graph(4)
```

In [2]:

```
# Visualize network
def visualize_as_shell(graph, figure_name, figure_size):
    plt.figure(figure_name,figsize=figure_size)
    pos = nx.shell_layout(G, center=(1,1)) # other layouts: bipartite, circular*, kamada_k

    groups = set([d[1] for d in G.degree()])
    mapping = dict(zip(sorted(groups),count()))
    colors = [mapping[d[1]] for d in G.degree()]

#     nx.draw(G, pos, node_size=250, arrows=graph.is_directed()) #, node_colors=colors)
    nx.draw(G, with_labels=False, node_size=200, pos=pos, node_color=colors, arrows=graph.
    nx.draw_networkx_labels(G, pos, font_size=10)
```

In [3]:

```
# Could visualize with colored nodes for higher degree
from itertools import count
def visualize_with_colors_for_degree(graph, figure_name, figure_size):
    plt.figure(figure_name,figsize=figure_size)

    groups = set([d[1] for d in G.degree()])
    mapping = dict(zip(sorted(groups),count()))
    colors = [mapping[d[1]] for d in G.degree()]

    nx.draw(G, with_labels=False, node_size=200, pos=nx.fruchterman_reingold_layout(G), nc
```

In [4]:

```
def describe_network(graph):

    # Describe the network:
    # Get Node info
    G = graph
    nodelist = G.nodes

    # Get Edge info
    edgearray = G.edges.data()
```

```
    edges_without_weight = []
    duplicates = []
    duplicate_edges = []
    for edge in edgearray:
        if (edge[0], edge[1]) in edges_without_weight:
            duplicates.append((edge[0], edge[1]))
        edges_without_weight.append((edge[0], edge[1]))

    print('There are {} duplicate edges: {}'.format(len(duplicates), duplicates))

    for edge in edgearray:
        if (edge[0], edge[1]) in duplicates:
            duplicate_edges.append(edge)

    print('These duplicates result in a total of {} edges: {}'.format(len(duplicate_edges)

    # What is the average degree? Get degree info
    degrees = G.degree()
    if G.is_directed():
        indegrees = G.in_degree()
        outdegrees = G.out_degree()
        averageindegree = (sum([val for (node, val) in sorted(indegrees, key=lambda pair:
        averageoutdegree = (sum([val for (node, val) in sorted(outdegrees, key=lambda pair
    else:
        averageindegree = "There are no indegrees for undirected graph"
        averageoutdegree = "There are no outdegrees for undirected graph"

    averagedegree = (sum([val for (node, val) in sorted(degrees, key=lambda pair: pair[0])
    density = (len(edgearray))/((len(nodelist))*(len(nodelist)-1))
    if not G.is_multigraph() and not G.is_directed():
        avg_clustering_coeff = nx.average_clustering(G)
        betweenness_centrality = nx.betweenness_centrality(G)
    else:
        avg_clustering_coeff = "Is not defined for a multigraph or digraph"
        betweenness_centrality = "Is not defined for a multigraph or digraph"

    print('avarage degree: {}'.format(averagedegree))
    print('average indegree: {}'.format(averageindegree))
    print('average outdegree: {}'.format(averageoutdegree))
    print('density (can be above 1 for multigraphs): {}'.format(density))
    print('average clustering coefficient: {}'.format(avg_clustering_coeff))
    print('betweenness centrality: {}'.format(betweenness_centrality))


    return averagedegree, averageindegree, averageoutdegree, density, avg_clustering_coeff
    # What are some other static network measures, and
    # why are they significant to this system? Get other info
```

We can convert the MultiDiGraph to a simple DiGraph by adding the weights on the duplicated edges. There are only 14 of these, so we perform this step below.

In [5]:

```
# Converting multigraph to simple graph
def convert_multigraph_to_simple(graph):
    G = graph
    G2 = nx.DiGraph()
    duplicate_edges = []
    duplicates = []
    for u,v,data in G.edges(data=True):
        w = data['value']
        if G2.has_edge(u,v):
            G2[u][v]['weight'] += w
            if (u,v) not in duplicates:
                duplicates.append((u,v))
            duplicate_edges.append((u,v,data))
```

```
                duplicate_edges.append((u,v,{'value': G2[u][v]['weight']-w}))
            else:
                G2.add_edge(u, v, weight=w)

    # print(G.edges(data=True))
    print('duplicates found (should match above): {}'.format(duplicates))
    print('duplicate edges found (should match above): {} '.format(duplicate_edges))
    print(len(G2.edges.data()))
    return G2
```

In [6]:

```
# Are there any nodes that don't have any edges going into them?
# Are there any nodes that don't have any edges going out of them
def find_disconnected_nodes(graph):
    G = graph
    nodelist = G.nodes
    edgearray = G.edges.data()
    nodes_have_incoming_edge_list = []
    nodes_have_outgoing_edge_list = []
    for node in nodelist:
        node_has_incoming_edge = False
        node_has_outgoing_edge = False
        for edge in edgearray:
            if edge[1] == node:
                node_has_incoming_edge = True
            if edge[0] == node:
                node_has_outgoing_edge = True
        nodes_have_incoming_edge_list.append((node, node_has_incoming_edge))
        nodes_have_outgoing_edge_list.append((node, node_has_outgoing_edge))

    print("nodes_without_incoming edges: {}".format([x[0] for x in nodes_have_incoming_edg
    print("nodes_without_outgoing_edges: {}".format([x[0] for x in nodes_have_outgoing_edg
    return [x[0] for x in nodes_have_incoming_edge_list if x[1]==False], [x[0] for x in no

    # There are. This causes the voter model to fail, so we will need to convert the graph
```

In [7]:

```
def draw_voter_model(i):
    fig.clear()
    iteration = model.iteration()

    pos = nx.shell_layout(g)  # positions for all nodes

    # get nodes that vote 0:
    nodes_0 = []
    nodes_1 = []
    for node, status in model.status.items():
        if status == 0:
            nodes_0.append(node)
        else:
            nodes_1.append(node)


    # nodes
    options = {"edgecolors": "tab:gray", "node_size": 250}
    nx.draw_networkx_nodes(g, pos, nodelist=nodes_0, node_color="tab:red", **options)
    nx.draw_networkx_nodes(g, pos, nodelist=nodes_1, node_color="tab:blue", **options)

    # edges
    nx.draw_networkx_edges(g, pos, alpha=0.5)
    nx.draw_networkx_edges(g,pos)

    # labels
    nx.draw_networkx_labels(g, pos, font_size=10, font_color="whitesmoke")
```

```
    fig.suptitle("Frame %d:    "%(i+1), fontweight="bold")
#     ax.set_xticks([])
#     ax.set_yticks([])
```

In [8]:

```
import ndlib.models.ModelConfig as mc
import ndlib.models.opinions as op

class voterModel:

    def __init__(self, graph, config):
        self.model = op.VoterModel(graph)
        self.config = config
        self.model.set_initial_status(config)



def create_voter_model(graph, fraction):
    config = mc.Configuration()
    config.add_model_parameter('fraction_infected', fraction)
    return voterModel(graph, config).model

def calculate_voter_convergence(model):
    num_iters = 0
    unchanged = 0
    prev_model_status = None
    status_over_time = []
    status_count_over_time = []

#     while unchanged < 1000:
    while (len(status_count_over_time) == 0 or
        (status_count_over_time[-1] != len(model.status) and status_count_over_time[-1] !=
        unchanged < 1000)):
        num_iters += 1
        if len(status_over_time) > 0 and model.status == status_over_time[-1]:
            unchanged += 1
        else:
            unchanged = 0
        status_over_time.append(copy.deepcopy(model.status))
        status_count_over_time.append(model.status_delta(model.status)[1][1])
        iteration = model.iteration()

    if unchanged >= 1000:
        return num_iters-1000, status_count_over_time

    return num_iters, status_count_over_time
```

In [9]:

```
import scikits.bootstrap as bootstrap
import warnings
warnings.filterwarnings('ignore') # Danger, Will Robinson! (not a scalable hack, and may s

import scipy.stats

def plot_mean_and_bootstrapped_ci_multiple(input_data = None, title = 'overall', name = "c
    """

    parameters:
    input_data: (numpy array of numpy arrays of shape (max_k, num_repitions)) solution met
    name: numpy array of string names for legend
    x_label: (string) x axis label
    y_label: (string) y axis label

    returns:
    None
```

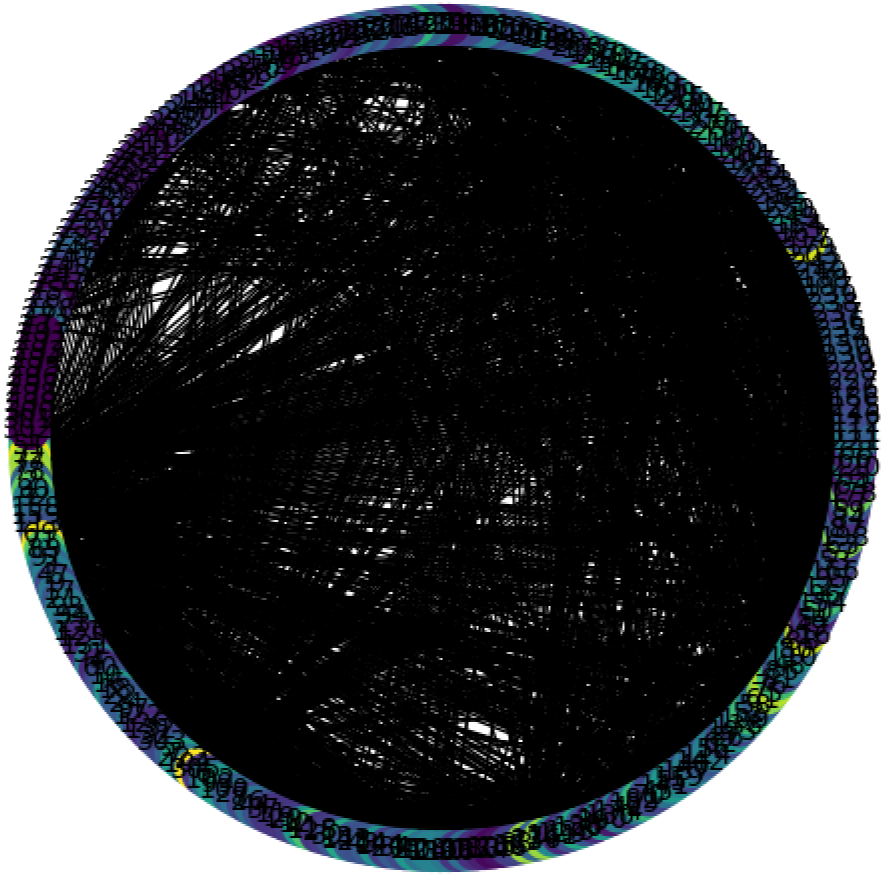
```
    """

    generations = len(input_data[0])

    fig, ax = plt.subplots()
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.set_title(title)
    for i in range(len(input_data)):
        CIs = []
        mean_values = []
        for j in range(generations):
            mean_values.append(np.mean(input_data[i][j]))
#               CIs.append(bootstrap.ci(input_data[i][j], statfunction=np.mean))
        mean_values=np.array(mean_values)

#           print(CIs)
#           high = []
#           low = []
#           for j in range(len(CIs)):
#               low.append(CIs[j][0])
#               high.append(CIs[j][1])

#           low = np.array(low)
#           high = np.array(high)

        y = range(0, generations)
        ax.plot(y, mean_values, label=name[i])
#           ax.fill_between(y, high, low, alpha=.2)
        ax.legend()
```

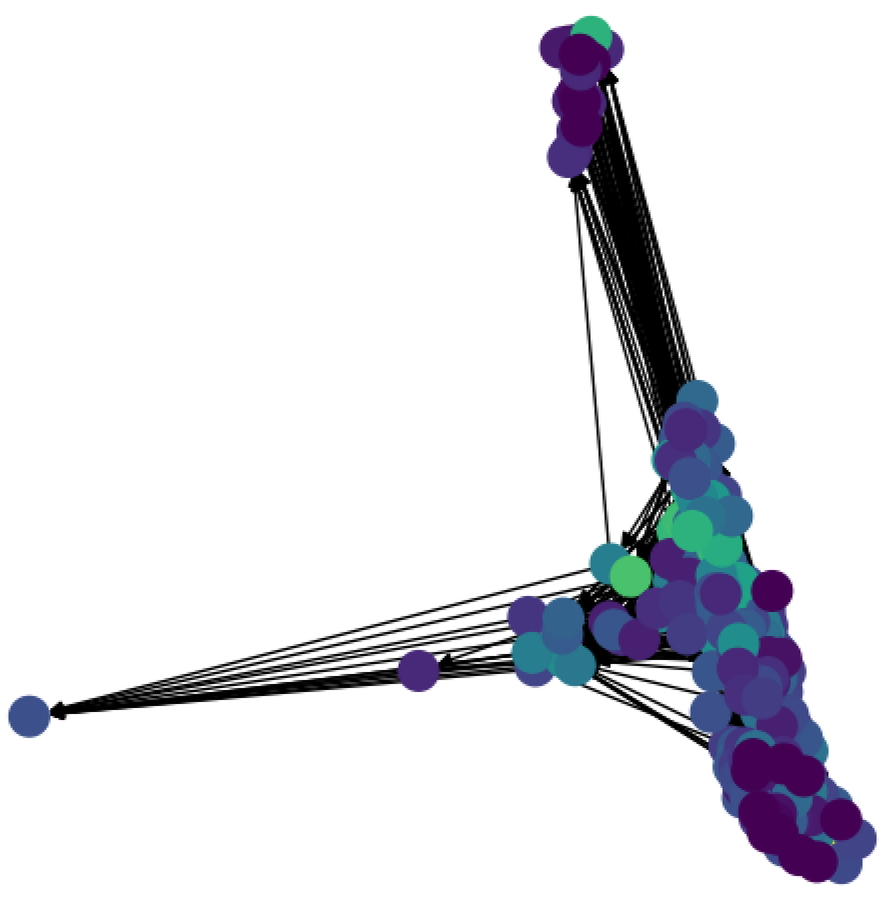
In [10]:

```
G = load_graph_from_file("celegansneural/celegansneural.gml")
```

In [11]:

```
visualize_as_shell(G, '1', (5,5))
visualize_with_colors_for_degree(G, '2', (5,5))
```





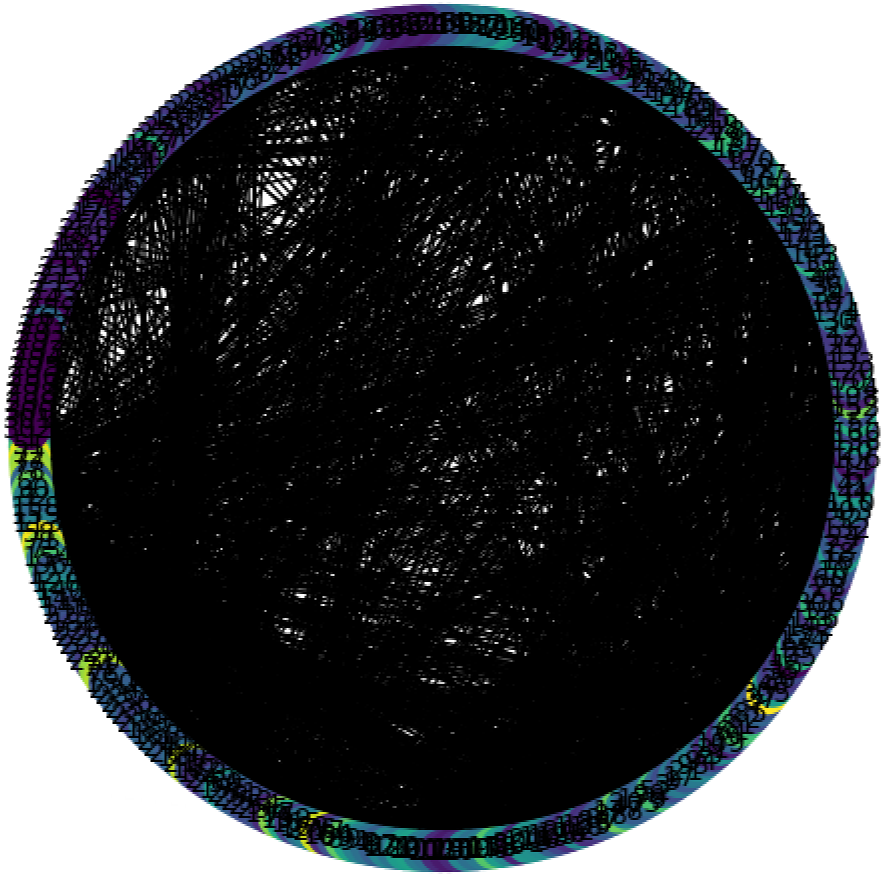
In [12]:
```
describe_network(G)
```

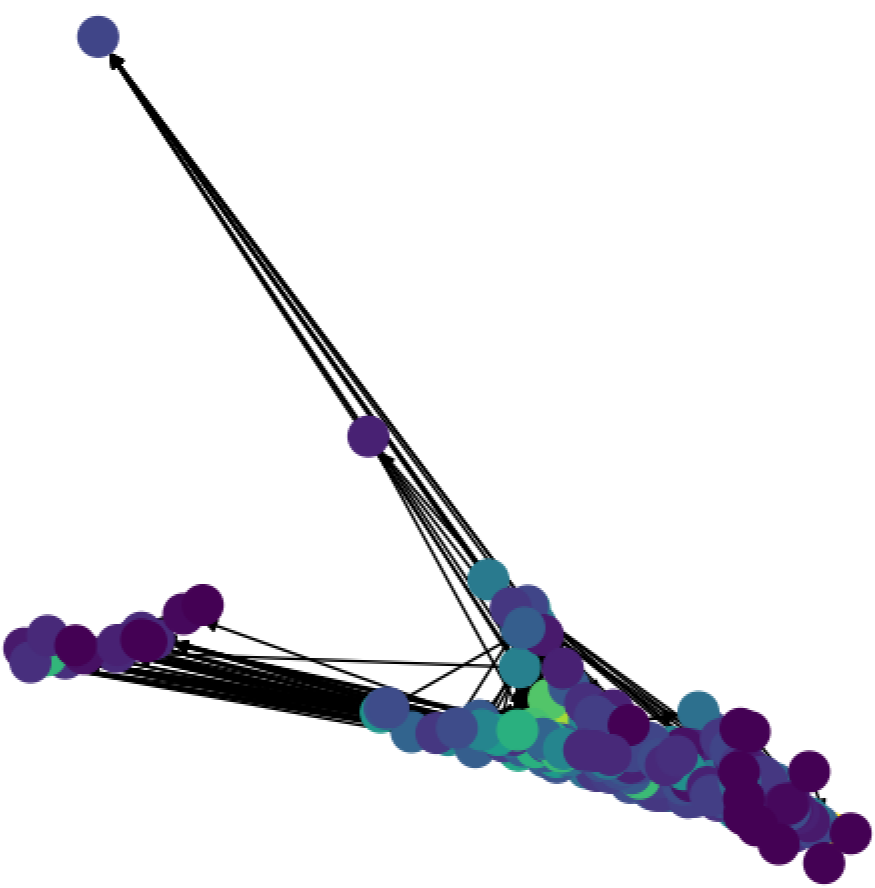
```
There are 14 duplicate edges: [('71', '240'), ('39', '303'), ('141', '23'), ('178', '30
5'), ('111', '303'), ('177', '305'), ('199', '189'), ('172', '150'), ('149', '187'), ('14
9', '250'), ('149', '306'), ('269', '305'), ('268', '305'), ('273', '305')]
These duplicates result in a total of 28 edges: [('71', '240', {'value': 1}), ('71', '24
0', {'value': 2}), ('39', '303', {'value': 1}), ('39', '303', {'value': 2}), ('141', '23',
{'value': 1}), ('141', '23', {'value': 1}), ('178', '305', {'value': 3}), ('178', '305',
{'value': 21}), ('111', '303', {'value': 1}), ('111', '303', {'value': 1}), ('177', '305',
{'value': 3}), ('177', '305', {'value': 22}), ('199', '189', {'value': 1}), ('199', '189',
{'value': 3}), ('172', '150', {'value': 1}), ('172', '150', {'value': 1}), ('149', '187',
{'value': 1}), ('149', '187', {'value': 1}), ('149', '250', {'value': 1}), ('149', '250',
{'value': 1}), ('149', '306', {'value': 1}), ('149', '306', {'value': 1}), ('269', '305',
{'value': 2}), ('269', '305', {'value': 8}), ('268', '305', {'value': 2}), ('268', '305',
{'value': 2}), ('273', '305', {'value': 1}), ('273', '305', {'value': 10})]
avarage degree: 15.885521885521886
average indegree: 7.942760942760943
average outdegree: 7.942760942760943
density (can be above 1 for multigraphs): 0.026833651833651835
average clustering coefficient: Is not defined for a multigraph or digraph
betweenness centrality: Is not defined for a multigraph or digraph
```

Out[12]:
```
(15.885521885521886,
 7.942760942760943,
 7.942760942760943,
 0.026833651833651835,
 'Is not defined for a multigraph or digraph',
 'Is not defined for a multigraph or digraph')
```

In [13]:
```
G = convert_multigraph_to_simple(G)
visualize_as_shell(G, '3', (5,5))
visualize_with_colors_for_degree(G, '4', (5,5))
describe_network(G)
```

```
duplicates found (should match above): [('71', '240'), ('39', '303'), ('141', '23'), ('17
8', '305'), ('111', '303'), ('177', '305'), ('199', '189'), ('172', '150'), ('149', '18
7'), ('149', '250'), ('149', '306'), ('269', '305'), ('268', '305'), ('273', '305')]
duplicate edges found (should match above): [('71', '240', {'value': 2}), ('71', '240',
{'value': 1}), ('39', '303', {'value': 2}), ('39', '303', {'value': 1}), ('141', '23', {'v
alue': 1}), ('141', '23', {'value': 1}), ('178', '305', {'value': 21}), ('178', '305', {'v
alue': 3}), ('111', '303', {'value': 1}), ('111', '303', {'value': 1}), ('177', '305', {'v
alue': 22}), ('177', '305', {'value': 3}), ('199', '189', {'value': 3}), ('199', '189',
{'value': 1}), ('172', '150', {'value': 1}), ('172', '150', {'value': 1}), ('149', '187',
{'value': 1}), ('149', '187', {'value': 1}), ('149', '250', {'value': 1}), ('149', '250',
{'value': 1}), ('149', '306', {'value': 1}), ('149', '306', {'value': 1}), ('269', '305',
{'value': 8}), ('269', '305', {'value': 2}), ('268', '305', {'value': 2}), ('268', '305',
{'value': 2}), ('273', '305', {'value': 10}), ('273', '305', {'value': 1})]
2345
```





There are 0 duplicate edges: []

```
These duplicates result in a total of 0 edges: []
avarage degree: 15.79124579124579
average indegree: 7.895622895622895
average outdegree: 7.895622895622895
density (can be above 1 for multigraphs): 0.026674401674401674
average clustering coefficient: Is not defined for a multigraph or digraph
betweenness centrality: Is not defined for a multigraph or digraph
```

Out[13]:

```
(15.79124579124579,
 7.895622895622895,
 7.895622895622895,
 0.026674401674401674,
 'Is not defined for a multigraph or digraph',
 'Is not defined for a multigraph or digraph')
```

In [14]:

```python
no_incoming_edge, no_outgoing_edge = find_disconnected_nodes(G)
completely_disconnected = []
if len(no_incoming_edge) > 0 and len(no_outgoing_edge) > 0:
    for node in no_incoming_edge:
        if node in no_outgoing_edge:
            completely_disconnected.append(node)
print("There are a total of {} nodes that are completely disconnected".format(len(complete
print("\nIt may benefit us to convert this graph to undirected to perform the voter model
    "experiment since the model we use here will struggle if the node has no incoming edge
```

```
nodes_without_incoming edges: ['11', '12', '53', '64', '151', '175', '176', '191', '210',
'211', '212', '243', '259', '267', '273', '291', '292', '293', '294', '295', '296', '297',
'298', '299', '300', '301', '302']
nodes_without_outgoing_edges: ['305', '306', '303']
There are a total of 0 nodes that are completely disconnected

It may benefit us to convert this graph to undirected to perform the voter model
experiment since the model we use here will struggle if the node has no incoming edges.
```

In [24]:

```python
g = G.to_undirected()
ad, aid, aod, den, acc, bc = describe_network(g)

def take(n, iterable):
    "Return first n items of the iterable as a list"
    return list(islice(iterable, n))

print(dict(sorted(bc.items(), key=lambda item: item[1])))
top_10 = take(10, dict(sorted(bc.items(), key=lambda item: item[1])).items())
bottom_10 = take(10, dict(sorted(bc.items(), key=lambda item: item[1], reverse=True)).iter
print(top_10)
print(bottom_10)
```

```
There are 0 duplicate edges: []
These duplicates result in a total of 0 edges: []
avarage degree: 14.464646464646465
average indegree: There are no indegrees for undirected graph
average outdegree: There are no outdegrees for undirected graph
density (can be above 1 for multigraphs): 0.024433524433524433
average clustering coefficient: 0.2923632978321903
betweenness centrality: {'1': 0.001213353962106609, '51': 0.01574263220410914, '72': 0.048
01549330577608, '77': 0.028535608144431866, '78': 0.031163449440266954, '2': 0.00304589218
48214096, '90': 0.018474870705423054, '92': 0.004705215633435171, '158': 0.004090854671225
415, '159': 0.002784003917069664З, '113': 0.004833755630940964, '58': 0.00466842284607132
4, '71': 0.05759006668189357, '73': 0.0245029035321954, '154': 0.0007345842863693228, '9
6': 0.00976574450109922, '127': 0.003180632172648924, '128': 0.0046953203212667, '140': 0.
005783679167607876, '144': 0.004861382220990804, '146': 0.0016177231233559023, '225': 0.00
17586032930189844, '186': 0.0008423497159532861, '226': 0.0008261379323095446, '227': 0.00
05419523439376386, '228': 0.0004685305321133337, '229': 0.0011898106039636772, '230': 0.00
07154641036217624, '74': 0.026740831528213608, '76': 0.018037489841215257, '150': 0.001232
985286878109, '234': 0.0010735498166460943, '235': 0.0007714892253235477, '236': 0.0016008
```

289652584206, '237': 0.0023654311563487706, '238': 0.0016273760419995243, '239': 0.0008384
090863910738, '187': 0.0018834078767990192, '188': 0.002300532429948423, '240': 0.00253715
43374374896, '242': 0.005205525473025064, '204': 0.0005114960471872843, '180': 0.001964852
3576482712, '216': 0.025029687942043584, '162': 0.0005423006318800255, '163': 0.0008176011
673736431, '249': 0.001236274886268793, '250': 0.001128729274751487, '195': 0.006534420931
859534, '251': 0.0018763677965927563, '252': 0.0008643713112940953, '253': 0.0007029679276
888326, '254': 0.0008597462970077472, '255': 0.001283686180970819, '197': 0.00034845538343
82239, '75': 0.013368222572963674, '199': 0.01556946793128174, '165': 0.000235326973359802
65, '169': 0.0009469414754769799, '276': 0.001766127404282391, '217': 0.02569794399335237
7, '164': 0.0003731514077655832, '69': 0.0019306523503138408, '89': 0.014008952611241717,
'91': 0.011941857275945043, '4': 0.006263090103729547, '8': 0.003084340439595756, '97': 0.
0035236697675283097, '121': 0.005667072780302987, '122': 0.005958727119781775, '99': 0.003
8678835162178344, '100': 0.006050848776730488, '101': 0.004089431748280014, '102': 0.00528
9978468636177, '129': 0.001210293960394519, '130': 0.0005033745670758428, '131': 0.0009597
66890279008, '139': 0.005201786424722332, '107': 0.004207295027700345, '108': 0.0026879695
498767293, '118': 0.016497160701384392, '55': 0.00896471180160393, '56': 0.006293245268302
855, '201': 0.00014389558015683496, '115': 0.005852501143910876, '200': 0.0037480954913220
554, '119': 0.00461493202872725, '123': 0.0025825794421908727, '126': 0.00338225190122207
3, '202': 0.0003138469884399292, '116': 0.005650901377818875, '125': 0.00168621349254392
2, '83': 0.0008437944658185567, '87': 0.003191186566593027, '88': 0.0007161422589949475,
'178': 0.02364901076923563, '47': 0.007074815185759396, '203': 0.00035395099195502686, '17
9': 0.00164530473934949, '98': 0.003042361195645992, '52': 0.017991376639638578, '3': 0.0
02800549396308381, '9': 0.003084581731772860 5, '17': 0.001833160692987866 2, '21': 0.003056
1115130606767, '93': 0.004774541304956841, '94': 0.007972641101016151, '23': 0.00334916222
70683, '31': 0.0012156284331879459, '109': 0.0028157649616493445, '39': 0.0003322796546873
5684, '61': 0.0032401872696085562, '81': 0.005282109528878778 6, '82': 0.00646522108801505
8, '85': 0.0015613468598052855, '7': 0.002278188391232718, '37': 0.001164117892909683, '30
5': 0.3033972016765733, '40': 0.005053579477107628 5, '26': 0.0029185587448964964, '35': 0.
00016442639194820427, '15': 0.001117632769457102, '105': 0.0002885448825952639, '95': 0.00
7715967747035695, '124': 0.001393747282840344, '135': 0.00474697675604031 7, '142': 0.00199
946594647305, '136': 0.005957486961947 88, '19': 0.00070588728396239 96, '306': 0.0444894369
55661085, '27': 0.004722008832615756, '28': 0.0039037694766318825, '60': 0.000626572110852
4488, '10': 0.0026521205692062 1, '16': 0.0008455623771243857, '18': 0.003813898570972587 4,
'22': 0.001313827902174071, '24': 0.0025474231786939395, '132': 0.0008392202229555646, '3
2': 0.0016706212112268103, '303': 0.0032322734050416655, '110': 0.0024457830217336648, '4
4': 0.00405295379550071, '41': 0.005295874267688393, '6': 0.0014879758904321702, '14': 0.0
008930836730002422, '106': 0.0004509412049010065, '20': 0.0007816348078904871, '138': 0.00
6507840412449884, '160': 0.014605025308529881, '5': 0.0014988231938799065, '222': 0.009117
301195794866, '152': 0.0010791941674675697, '198': 0.031147669688527554, '134': 0.00108354
08047650428, '70': 0.0021856073833454183, '120': 0.004817814099006116, '133': 0.0015156814
49572904, '11': 0.0007076905500723569, '29': 0.0016683518638346564, '12': 0.00053673752226
42717, '25': 0.0017666190874517955, '30': 0.0018651585704778887, '174': 0.0034567818008106
543, '161': 0.008167909665937757, '34': 0.0014697882590276766, '36': 0.000460823078485474
7, '46': 0.003123668828085282 2, '38': 0.002463782700527809, '13': 0.001318114734423269, '1
43': 0.011739016104695587, '43': 0.0010340855660417808, '141': 0.0056611435486221195, '17
3': 0.0030321506193536703, '156': 0.0009450466179770482, '33': 0.0007049375976583876, '15
5': 0.001959842071918, '104': 0.0005305267737023904, '117': 0.007102522576188044, '256':
0.0010596764371841967, '103': 0.0007729714566463541, '177': 0.026384293665241663, '62': 0.
005817093032562115, '170': 0.0017745955061803738, '67': 0.0027861190941301 1, '137': 0.0076
04590021070313, '171': 0.0007933992061300549, '79': 0.003446467367394885, '218': 0.0111961
37461915974, '219': 0.011570675084147289, '185': 0.0012312188438437664, '42': 0.0035035501
645968355, '189': 0.0021478600572206203, '275': 0.0003009746444840407, '196': 0.0001836259
843016367, '147': 0.005047746651562225, '172': 0.00138112694137208 8, '220': 0.011023444910
263252, '114': 0.003933031864566714, '45': 0.0007558200197061483, '57': 0.0029500618944217
6, '63': 0.0001910478602966271, '145': 0.0022882216368667 08, '214': 2.1454145753367575e-0
5, '86': 0.0014470019391378455, '193': 0.00537877600721831, '84': 0.0020111428523532726,
'245': 0.0018058930200732066, '213': 3.540282273049489e-05, '111': 0.005091798971083266,
'59': 0.0002660920145754785, '166': 0.0031560912527797 46, '49': 5.899806333869996e-05, '5
0': 6.73223537314977e-05, '241': 0.0018740118892108266, '215': 8.275999072157765e-05, '14
8': 0.004473657285615926 6, '80': 0.00403286819332357, '221': 0.0053755295264915 87, '157':
0.00582982030775288 3, '48': 0.005807230550575694, '192': 0.0030434450844223147, '168': 0.0
024271514480575427, '244': 0.0016739246021723541, '223': 0.00067076027132963 4, '260': 0.00
028059936931253 6, '261': 0.00012791463376251908, '262': 0.00020506799225748 58, '263': 0.00
016492951784084265, '264': 0.00025082386305599636, '265': 0.00017517606476764202, '266':
0.00033758416228576126, '54': 0.0029550333948783663, '153': 0.002464474266388877 5, '53':
0.0016282512623438106, '65': 0.0, '269': 0.0007482608038001508, '270': 0.0043061656577433 1

3, '272': 0.0004864510104736788, '64': 4.224176091119609e-05, '66': 0.0002231505821353254,
'68': 0.0002940634427080991, '112': 0.0036172514448994746, '274': 0.00010379298527111198,
'282': 0.0002481751169057299, '277': 0.0006366047412652038, '278': 0.0001987653534932651,
'279': 0.0004157050001908774, '246': 0.0004712308998397556, '247': 0.0008010729133770276,
'280': 0.0003341633733351251, '281': 0.000313498748770615, '231': 2.8630325240494733e-06,
'232': 1.2815478917173832e-05, '248': 0.0006003334083265434, '207': 0.0004315868245092458
4, '183': 0.0004529311607390043, '258': 0.0013345394863053268, '224': 0.000458656082076304
13, '257': 2.5099251794167047e-05, '167': 4.986519911290093e-05, '149': 0.0052787325864575
93, '190': 4.1961812448275783e-05, '194': 0.0005194298183860499, '184': 0.0004516143049483
911, '205': 0.0006584865786257114, '206': 0.0002335570373671141, '208': 0.0005316424557992
954, '151': 9.994194574033155e-05, '175': 0.0, '176': 0.0, '271': 0.00038323536063250004,
'268': 4.475351352691718e-05, '181': 0.0003364128702946777, '182': 0.0003317999332898925,
'191': 0.0, '209': 9.66814519023869e-05, '210': 0.00022589609831979247, '211': 0.0, '212':
0.00011043419508204959, '233': 3.272037170342255e-06, '243': 0.00014050672948978034, '25
9': 5.7260650480989465e-06, '267': 0.0, '273': 0.0027873161979715547, '291': 0.0, '292':
0.0, '293': 0.0, '294': 0.0, '295': 0.0, '296': 0.0, '297': 0.0, '298': 0.0, '299': 0.0,
'300': 0.0, '301': 0.0, '302': 0.0}
{'65': 0.0, '175': 0.0, '176': 0.0, '191': 0.0, '211': 0.0, '267': 0.0, '291': 0.0, '292':
0.0, '293': 0.0, '294': 0.0, '295': 0.0, '296': 0.0, '297': 0.0, '298': 0.0, '299': 0.0,
'300': 0.0, '301': 0.0, '302': 0.0, '231': 2.8630325240494733e-06, '233': 3.27203717034225
5e-06, '259': 5.7260650480989465e-06, '232': 1.2815478917173832e-05, '214': 2.145414575336
7575e-05, '257': 2.5099251794167047e-05, '213': 3.540282273049489e-05, '190': 4.1961812448
275783e-05, '64': 4.224176091119609e-05, '268': 4.475351352691718e-05, '167': 4.9865199112
90093e-05, '49': 5.899806333869996e-05, '50': 6.73223537314977e-05, '215': 8.2759990721577
65e-05, '209': 9.66814519023869e-05, '151': 9.994194574033155e-05, '274': 0.00010379298527
111198, '212': 0.00011043419508204959, '261': 0.00012791463376251908, '243': 0.00014050672
948978034, '201': 0.00014389558015683496, '35': 0.00016442639194820427, '263': 0.000164929
51784084265, '265': 0.00017517606476764202, '196': 0.0001836259843016367, '63': 0.00019104
78602966271, '278': 0.0001987653534932651, '262': 0.0002050679922574858, '66': 0.000223150
5821353254, '210': 0.00022589609831979247, '206': 0.0002335570373671141, '165': 0.00023532
697335980265, '282': 0.0002481751169057299, '264': 0.00025082386305599636, '59': 0.0002660
920145754785, '260': 0.000280599369312536, '105': 0.0002885448825952639, '68': 0.000294063
4427080991, '275': 0.0003009746444840407, '281': 0.000313498748770615, '202': 0.0003138469
884399292, '182': 0.0003317999332898925, '39': 0.00033227965468735684, '280': 0.0003341633
733351251, '181': 0.0003364128702946777, '266': 0.00033758416228576126, '197': 0.000348455
3834382239, '203': 0.00035395099195502686, '164': 0.0003731514077655832, '271': 0.00038323
536063250004, '279': 0.0004157050001908774, '207': 0.00043158682450924584, '106': 0.00045
09412049010065, '184': 0.0004516143049483911, '183': 0.0004529311607390043, '224': 0.00045
865608207630413, '36': 0.0004608230784854747, '228': 0.0004685305321133337, '246': 0.00047
12308998397556, '272': 0.0004864510104736788, '130': 0.0005033745670758428, '204': 0.00051
14960471872843, '194': 0.0005194298183860499, '104': 0.0005305267737023904, '208': 0.00053
16424557992954, '12': 0.0005367375222642717, '227': 0.0005419523439376386, '162': 0.000542
3006318800255, '248': 0.0006003334083265434, '60': 0.0006265721108524488, '277': 0.0006366
047412652038, '205': 0.0006584865786257114, '223': 0.000670760271329634, '253': 0.00070296
79276888326, '33': 0.0007049375976583876, '19': 0.0007058872839623996, '11': 0.00070769055
00723569, '230': 0.0007154641036217624, '88': 0.0007161422589949475, '154': 0.000734584286
3693228, '269': 0.0007482608038001508, '45': 0.0007558200197061483, '235': 0.0007714892253
235477, '103': 0.0007729714566463541, '20': 0.0007816348078904871, '171': 0.00079339920613
00549, '247': 0.0008010729133770276, '163': 0.0008176011673736431, '226': 0.00082613793230
95446, '239': 0.0008384090863910738, '132': 0.0008392202229555646, '186': 0.00084234971595
32861, '83': 0.0008437944658185567, '16': 0.0008455623771243857, '254': 0.0008597462970077
472, '252': 0.0008643713112940953, '14': 0.0008930836730002422, '156': 0.00094504661797704
82, '169': 0.0009469414754769799, '131': 0.000959766890279008, '43': 0.001034085566041780
8, '256': 0.0010596764371841967, '234': 0.0010735498166460943, '152': 0.001079194167467569
7, '134': 0.0010835408047650428, '15': 0.001117632769457102, '250': 0.001128729274751487,
'37': 0.001164117892909683, '229': 0.0011898106039636772, '129': 0.00121029396039451 9,
'1': 0.0012133539621066 09, '31': 0.0012156284331879459, '185': 0.0012312188438437664, '15
0': 0.00123298528687810 9, '249': 0.00123627488626879 3, '255': 0.00128368618097081 9, '22':
0.0013138279021740 71, '13': 0.0013181147344232 69, '258': 0.0013345394863053268, '172': 0.0
01381126941372088, '124': 0.001393747282840344, '86': 0.0014470019391378455, '34': 0.00146
97882590276766, '6': 0.0014879758904321702, '5': 0.0014988231938799065, '133': 0.001515681
449572904, '85': 0.0015613468598052855, '236': 0.0016008289652584206, '146': 0.00161772312
33559023, '238': 0.0016273760419995243, '53': 0.0016282512623438106, '179': 0.001645304739
34949, '29': 0.0016683518638346564, '32': 0.0016706212112268103, '244': 0.0016739246021723
541, '125': 0.0016862134925439232, '225': 0.0017586032930189844, '276': 0.0017661274042823
91, '25': 0.0017666190874517955, '170': 0.0017745955061803738, '245': 0.0018058930200732 06

```
6, '17': 0.0018331606929878662, '30': 0.0018651585704778887, '241': 0.0018740118892108266,
'251': 0.0018763677965927563, '187': 0.0018834078767990192, '69': 0.0019306523503138408,
'155': 0.001959842071918, '180': 0.0019648523576482712, '142': 0.00199946594647305, '84':
0.0020111428523532726, '189': 0.0021478600572206203, '70': 0.0021856073833454183, '7': 0.0
02278188391232718, '145': 0.002288221636866708, '188': 0.002300532429948423, '237': 0.0023
654311563487706, '168': 0.0024271514480575427, '110': 0.0024457830217336648, '38': 0.00246
3782700527809, '153': 0.0024644742663888775, '240': 0.0025371543374374896, '24': 0.0025474
231786939395, '123': 0.0025825794421908727, '10': 0.00265212056920621, '108': 0.0026879695
498767293, '159': 0.0027840039170696643, '67': 0.00278611909413011, '273': 0.0027873161979
715547, '3': 0.002800549396308381, '109': 0.0028157649616493445, '26': 0.00291855874489649
64, '57': 0.00295006189442176, '54': 0.0029550333948783663, '173': 0.0030321506193536703,
'98': 0.0030423611956459924, '192': 0.0030434450844223147, '2': 0.0030458921848214096, '2
1': 0.0030561115130606767, '8': 0.003084340439595756, '9': 0.0030845817317728605, '46': 0.
0031236688280852822, '166': 0.003156091252779746, '127': 0.003180632172648924, '87': 0.003
1911865665930276, '303': 0.0032322734050416655, '61': 0.0032401872696085562, '23': 0.00334
91622270683, '126': 0.0033822519012222073, '79': 0.003446467367394885, '174': 0.0034567818
008106543, '42': 0.0035035501645968355, '97': 0.0035236697675283097, '112': 0.003617251444
8994746, '200': 0.0037480954913220554, '18': 0.0038138985709725874, '99': 0.00386788351621
78344, '28': 0.0039037694766318825, '114': 0.003933031864566714, '80': 0.0040328681933235
7, '44': 0.00405295379550071, '101': 0.004089431748280014, '158': 0.004090854671225415, '1
07': 0.004207295027700345, '270': 0.004306165657743313, '148': 0.0044736572856159266, '11
9': 0.00461493202872725, '58': 0.004668422846071324, '128': 0.0046953203212667, '92': 0.00
4705215633435171, '27': 0.004722008832615756, '135': 0.004746976756040317, '93': 0.0047745
41304956841, '120': 0.004817814099006116, '113': 0.004833755630940964, '144': 0.0048613822
20990804, '147': 0.005047746651562225, '40': 0.0050535794771076285, '111': 0.0050917989710
83266, '139': 0.005201786424722332, '242': 0.005205525473025064, '149': 0.0052787325864575
93, '81': 0.0052821095288787786, '102': 0.005289978468636177, '41': 0.005295874267688393,
'221': 0.005375529526491587, '193': 0.00537877600721831, '116': 0.005650901377818875, '14
1': 0.0056611435486221195, '121': 0.005667072780302987, '140': 0.005783679167607876, '48':
0.005807230550575694, '62': 0.005817093032562115, '157': 0.005829820307752883, '115': 0.00
5852501143910876, '136': 0.00595748696194788, '122': 0.005958727119781775, '100': 0.006050
848776730488, '4': 0.006263090103729547, '56': 0.006293245268302855, '82': 0.0064652210880
15058, '138': 0.006507840412449884, '195': 0.006534420931859534, '47': 0.00707481518575939
6, '117': 0.007102522576188044, '137': 0.007604590021070313, '95': 0.007715967747035695,
'94': 0.007972641101016151, '161': 0.008167909665937757, '55': 0.00896471180160393, '222':
0.009117301195794866, '96': 0.00976574450109922, '220': 0.011023444910263252, '218': 0.011
196137461915974, '219': 0.011570675084147289, '143': 0.011739016104695587, '91': 0.0119418
57275945043, '75': 0.013368222572963674, '89': 0.014008952611241717, '160': 0.014605025308
529881, '199': 0.01556946793128174, '51': 0.01574263220410914, '118': 0.01649716070138439
2, '52': 0.017991376639638578, '76': 0.018037489841215257, '90': 0.018474870705423054, '17
8': 0.02364901076923563, '73': 0.0245029035321954, '216': 0.025029687942043584, '217': 0.0
25697943993352377, '177': 0.026384293665241663, '74': 0.026740831528213608, '77': 0.028535
608144431866, '198': 0.031147669688527554, '78': 0.031163449440266954, '306': 0.0444894369
55661085, '72': 0.04801549330577608, '71': 0.05759006668189357, '305': 0.3033972016765733}
[('65', 0.0), ('175', 0.0), ('176', 0.0), ('191', 0.0), ('211', 0.0), ('267', 0.0), ('29
1', 0.0), ('292', 0.0), ('293', 0.0), ('294', 0.0)]
[('305', 0.3033972016765733), ('71', 0.05759006668189357), ('72', 0.04801549330577608),
('306', 0.044489436955661085), ('78', 0.031163449440266954), ('198', 0.03114766968852755
4), ('77', 0.028535608144431866), ('74', 0.026740831528213608), ('177', 0.0263842936652416
63), ('217', 0.025697943993352377)]
```


In [193…

```
# Make animation of voter model
# model = create_voter_model(g, 0.4)

# fig = plt.figure(figsize=(50,50))
# n_iter = 2000
# ani = animation.FuncAnimation(fig, draw_voter_model, frames=n_iter, interval=100, repeat

# ani.save('dynamic_images.mp4')
```

In [194…

```
# Create and run voter model
num_runs = 10
num_iters_list = []
status_count_over_time_list = np.zeros((num_runs, 200000))
```

```
fraction_infected = 0.6

for i in range(num_runs):
    start_time = time.time()
    model = create_voter_model(g, fraction_infected)
    num_iters, status_count_over_time = calculate_voter_convergence(model)
    status_count_over_time = np.array([status_count_over_time[j] if j < len(status_count_c
    num_iters_list.append(num_iters)
    status_count_over_time_list[i] = status_count_over_time
    print('convergence of voter model', i, time.time()-start_time, num_iters)

status_count_over_time_list_truncated = np.zeros((num_runs, max(num_iters_list)))
for i in range(num_runs):
    status_count_over_time_list_truncated[i] = status_count_over_time_list[i][:max(num_ite
status_count_over_time_list = np.zeros((num_runs, max(num_iters_list)))
status_count_over_time_list = status_count_over_time_list_truncated
print(status_count_over_time_list)
```

```
convergence of voter model 0 33.49984407424927 108256
convergence of voter model 1 12.829546451568604 41520
convergence of voter model 2 1.796046257019043 5718
convergence of voter model 3 14.479466915130615 47292
convergence of voter model 4 6.871917009353638 22261
convergence of voter model 5 10.637659072875977 34565
convergence of voter model 6 10.942421913146973 35687
convergence of voter model 7 28.095690965652466 91039
convergence of voter model 8 5.05385160446167 16516
convergence of voter model 9 4.198114395141602 13445
[[178. 178. 178. ... 296. 296. 297.]
 [178. 178. 179. ... 297. 297. 297.]
 [178. 178. 179. ... 297. 297. 297.]
 ...
 [178. 178. 179. ...   0.   0.   0.]
 [178. 178. 178. ...   0.   0.   0.]
 [178. 178. 178. ... 297. 297. 297.]]
```

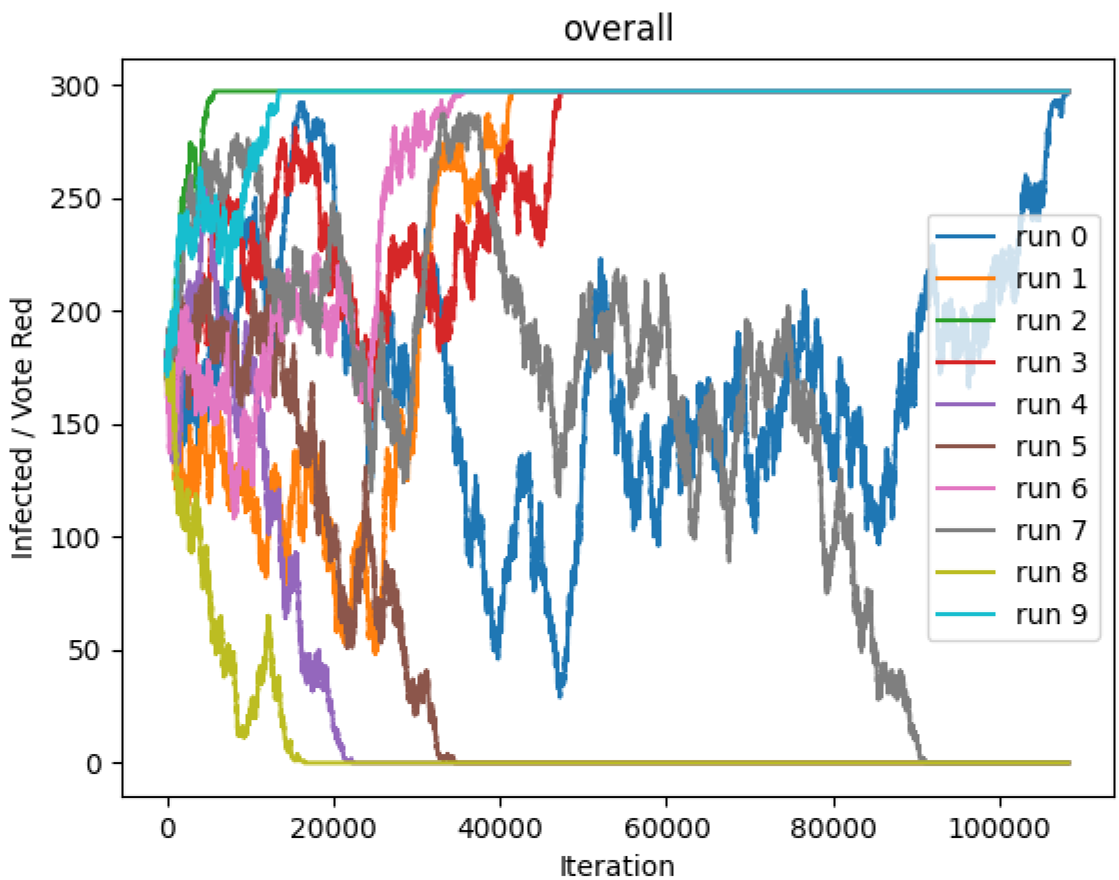
In [195…

```
# Seperate the runs that converge to red vs blue
status_count_over_time_0 = []
status_count_over_time_1 = []
status_count_over_time_other = []
for i in range(len(status_count_over_time_list)):
    if status_count_over_time_list[i][-1] == len(g.nodes):
        status_count_over_time_1.append(status_count_over_time_list[i])
    elif status_count_over_time_list[i][-1] == 0:
        status_count_over_time_0.append(status_count_over_time_list[i])
    else:
        status_count_over_time_other.append(status_count_over_time_list[i])

status_count_over_time_0 = np.array(status_count_over_time_0)
status_count_over_time_1 = np.array(status_count_over_time_1)
print(status_count_over_time_other)
```

```
[]
```

In [196…

```
# plot_mean_and_bootstrapped_ci_over_time(input_data = status_count_over_time_list, name =
# plot_mean_and_bootstrapped_ci_multiple(input_data=[np.transpose(status_count_over_time_(
plot_mean_and_bootstrapped_ci_multiple(input_data=[np.transpose(x) for x in status_count_(
print("Number of runs: {}".format(num_runs))
print("Number of runs that converge to red: {}".format(len(status_count_over_time_1)))
print("Number of runs that converge to blue: {}".format(len(status_count_over_time_0)))
```




```
Number of runs: 10
Number of runs that converge to red: 6
Number of runs that converge to blue: 4
```

In [25]:

```
# Create random graph with configuration model
sequence = [d[1] for d in g.degree]
print(sum(sequence)/len(sequence))

g = nx.configuration_model(sequence)

actual_degrees = [d for v, d in g.degree()]

actual_degrees == sequence

# Should visualize somehow
# Could visualize with larger nodes for higher degree
```

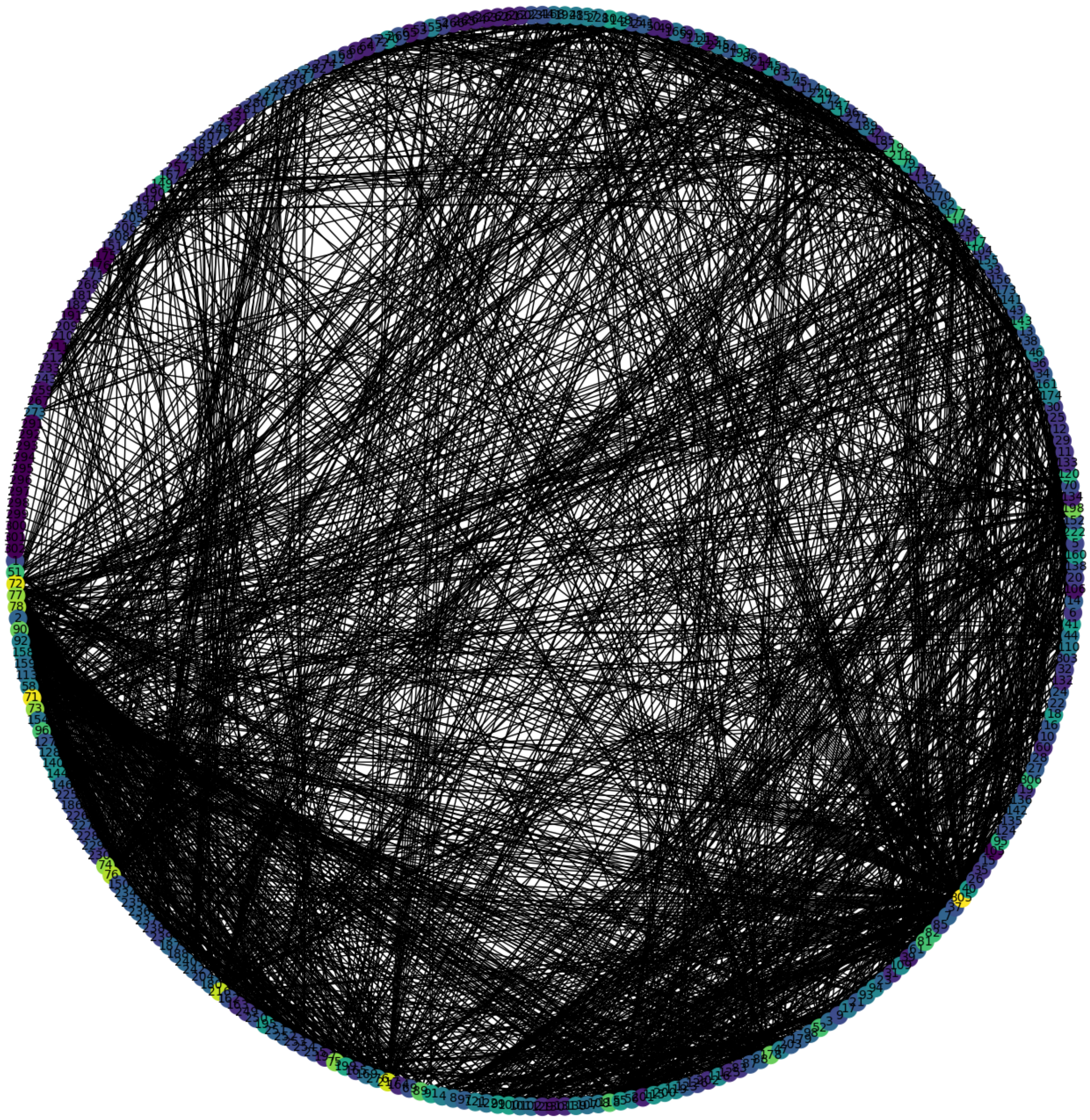
```
14.464646464646465
```

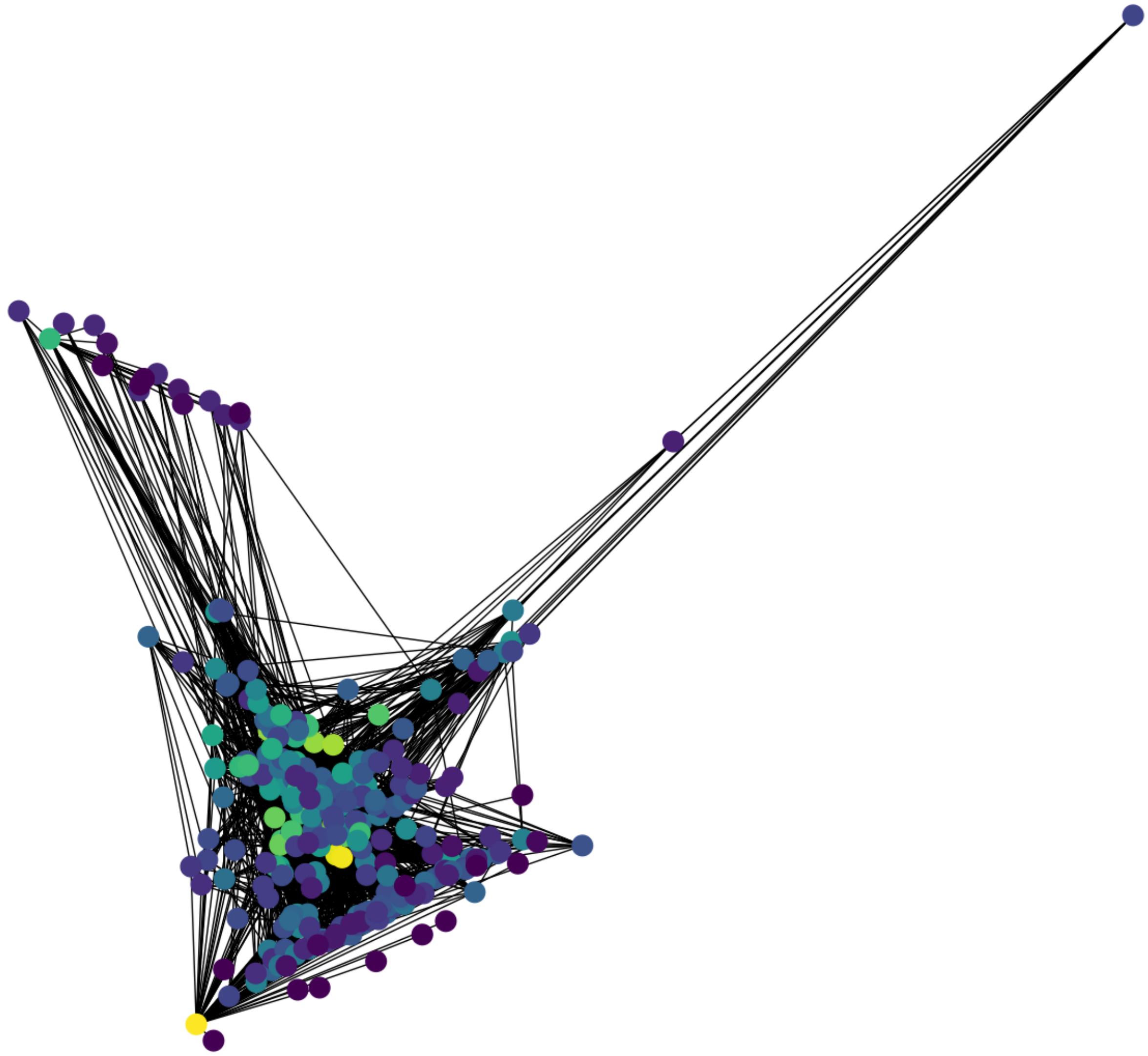
Out[25]:

```
True
```

In [26]:

```
visualize_as_shell(g, '5', (5,5))
visualize_with_colors_for_degree(g, '6', (5,5))
describe_network(g)
```





```
There are 150 duplicate edges: [(2, 86), (2, 117), (2, 117), (2, 189), (2, 123), (2, 2),
(2, 2), (2, 12), (2, 55), (2, 217), (2, 58), (2, 28), (3, 6), (3, 49), (3, 100), (3, 13),
(3, 147), (3, 181), (3, 117), (3, 43), (4, 60), (4, 260), (4, 176), (4, 84), (4, 70), (5,
144), (5, 187), (9, 85), (9, 105), (12, 12), (12, 12), (12, 117), (12, 117), (12, 117), (1
2, 28), (12, 28), (12, 28), (12, 77), (12, 23), (12, 13), (12, 64), (12, 178), (12, 178),
(12, 129), (12, 151), (12, 113), (12, 261), (13, 238), (13, 219), (13, 100), (13, 188), (1
3, 195), (15, 156), (15, 165), (15, 129), (15, 80), (15, 153), (16, 153), (18, 117), (18,
56), (19, 50), (19, 117), (19, 117), (20, 81), (25, 143), (28, 117), (28, 164), (28, 43),
(28, 279), (29, 189), (29, 153), (29, 153), (29, 181), (29, 224), (29, 117), (32, 164), (3
2, 131), (33, 151), (33, 245), (43, 63), (43, 112), (48, 143), (51, 129), (55, 130), (55,
117), (55, 117), (55, 176), (56, 171), (60, 60), (60, 80), (60, 117), (60, 129), (60, 19
7), (61, 84), (63, 71), (69, 216), (77, 217), (77, 117), (84, 202), (84, 117), (85, 182),
(90, 117), (90, 95), (93, 205), (95, 152), (95, 118), (99, 205), (100, 202), (100, 141),
(100, 182), (101, 117), (107, 143), (112, 129), (113, 117), (113, 117), (113, 151), (117,
170), (117, 170), (117, 170), (117, 170), (117, 170), (117, 239), (117, 268), (117, 123),
(117, 118), (117, 204), (117, 233), (117, 233), (117, 153), (117, 153), (117, 172), (117,
280), (119, 204), (120, 129), (120, 178), (125, 197), (142, 143), (148, 181), (153, 258),
(156, 209), (159, 169), (163, 273), (166, 251), (178, 181), (178, 264), (189, 222), (205,
205), (206, 267), (217, 254), (223, 232)]
These duplicates result in a total of 282 edges: [(2, 86, {}), (2, 86, {}), (2, 117, {}),
(2, 117, {}), (2, 117, {}), (2, 189, {}), (2, 189, {}), (2, 123, {}), (2, 123, {}), (2, 2,
{}), (2, 2, {}), (2, 2, {}), (2, 12, {}), (2, 12, {}), (2, 55, {}), (2, 55, {}), (2, 217,
{}), (2, 217, {}), (2, 58, {}), (2, 58, {}), (2, 28, {}), (2, 28, {}), (3, 6, {}), (3, 6,
{}), (3, 49, {}), (3, 49, {}), (3, 100, {}), (3, 100, {}), (3, 13, {}), (3, 13, {}), (3, 1
47, {}), (3, 147, {}), (3, 181, {}), (3, 181, {}), (3, 117, {}), (3, 117, {}), (3, 43,
```

```
{}), (3, 43, {}), (4, 60, {}), (4, 60, {}), (4, 260, {}), (4, 260, {}), (4, 176, {}), (4,
176, {}), (4, 84, {}), (4, 84, {}), (4, 70, {}), (4, 70, {}), (5, 144, {}), (5, 144, {}),
(5, 187, {}), (5, 187, {}), (9, 85, {}), (9, 85, {}), (9, 105, {}), (9, 105, {}), (12, 12,
{}), (12, 12, {}), (12, 12, {}), (12, 117, {}), (12, 117, {}), (12, 117, {}), (12, 117,
{}), (12, 28, {}), (12, 28, {}), (12, 28, {}), (12, 28, {}), (12, 77, {}), (12, 77, {}),
(12, 23, {}), (12, 23, {}), (12, 13, {}), (12, 13, {}), (12, 64, {}), (12, 64, {}), (12, 1
78, {}), (12, 178, {}), (12, 178, {}), (12, 129, {}), (12, 129, {}), (12, 151, {}), (12, 1
51, {}), (12, 113, {}), (12, 113, {}), (12, 261, {}), (12, 261, {}), (13, 238, {}), (13, 2
38, {}), (13, 219, {}), (13, 219, {}), (13, 100, {}), (13, 100, {}), (13, 188, {}), (13, 1
88, {}), (13, 195, {}), (13, 195, {}), (15, 156, {}), (15, 156, {}), (15, 165, {}), (15, 1
65, {}), (15, 129, {}), (15, 129, {}), (15, 80, {}), (15, 80, {}), (15, 153, {}), (15, 15
3, {}), (16, 153, {}), (16, 153, {}), (18, 117, {}), (18, 117, {}), (18, 56, {}), (18, 56,
{}), (19, 50, {}), (19, 50, {}), (19, 117, {}), (19, 117, {}), (19, 117, {}), (20, 81,
{}), (20, 81, {}), (25, 143, {}), (25, 143, {}), (28, 117, {}), (28, 117, {}), (28, 164,
{}), (28, 164, {}), (28, 43, {}), (28, 43, {}), (28, 279, {}), (28, 279, {}), (29, 189,
{}), (29, 189, {}), (29, 153, {}), (29, 153, {}), (29, 153, {}), (29, 181, {}), (29, 181,
{}), (29, 224, {}), (29, 224, {}), (29, 117, {}), (29, 117, {}), (32, 164, {}), (32, 164,
{}), (32, 131, {}), (32, 131, {}), (33, 151, {}), (33, 151, {}), (33, 245, {}), (33, 245,
{}), (43, 63, {}), (43, 63, {}), (43, 112, {}), (43, 112, {}), (48, 143, {}), (48, 143,
{}), (51, 129, {}), (51, 129, {}), (55, 130, {}), (55, 130, {}), (55, 117, {}), (55, 117,
{}), (55, 117, {}), (55, 176, {}), (55, 176, {}), (56, 171, {}), (56, 171, {}), (60, 60,
{}), (60, 60, {}), (60, 80, {}), (60, 80, {}), (60, 117, {}), (60, 117, {}), (60, 129,
{}), (60, 129, {}), (60, 197, {}), (60, 197, {}), (61, 84, {}), (61, 84, {}), (63, 71,
{}), (63, 71, {}), (69, 216, {}), (69, 216, {}), (77, 217, {}), (77, 217, {}), (77, 117,
{}), (77, 117, {}), (84, 202, {}), (84, 202, {}), (84, 117, {}), (84, 117, {}), (85, 182,
{}), (85, 182, {}), (90, 117, {}), (90, 117, {}), (90, 95, {}), (90, 95, {}), (93, 205,
{}), (93, 205, {}), (95, 152, {}), (95, 152, {}), (95, 118, {}), (95, 118, {}), (99, 205,
{}), (99, 205, {}), (100, 202, {}), (100, 202, {}), (100, 141, {}), (100, 141, {}), (100,
182, {}), (100, 182, {}), (101, 117, {}), (101, 117, {}), (107, 143, {}), (107, 143, {}),
(112, 129, {}), (112, 129, {}), (113, 117, {}), (113, 117, {}), (113, 117, {}), (113, 151,
{}), (113, 151, {}), (117, 170, {}), (117, 170, {}), (117, 170, {}), (117, 170, {}), (117,
170, {}), (117, 170, {}), (117, 239, {}), (117, 239, {}), (117, 268, {}), (117, 268, {}),
(117, 123, {}), (117, 123, {}), (117, 118, {}), (117, 118, {}), (117, 204, {}), (117, 204,
{}), (117, 233, {}), (117, 233, {}), (117, 233, {}), (117, 153, {}), (117, 153, {}), (117,
153, {}), (117, 172, {}), (117, 172, {}), (117, 280, {}), (117, 280, {}), (119, 204, {}),
(119, 204, {}), (120, 129, {}), (120, 129, {}), (120, 178, {}), (120, 178, {}), (125, 197,
{}), (125, 197, {}), (142, 143, {}), (142, 143, {}), (148, 181, {}), (148, 181, {}), (153,
258, {}), (153, 258, {}), (156, 209, {}), (156, 209, {}), (159, 169, {}), (159, 169, {}),
(163, 273, {}), (163, 273, {}), (166, 251, {}), (166, 251, {}), (178, 181, {}), (178, 181,
{}), (178, 264, {}), (178, 264, {}), (189, 222, {}), (189, 222, {}), (205, 205, {}), (205,
205, {}), (206, 267, {}), (206, 267, {}), (217, 254, {}), (217, 254, {}), (223, 232, {}),
(223, 232, {})]
avarage degree: 14.464646464646465
average indegree: There are no indegrees for undirected graph
average outdegree: There are no outdegrees for undirected graph
density (can be above 1 for multigraphs): 0.024433524433524433
average clustering coefficient: Is not defined for a multigraph or digraph
betweenness centrality: Is not defined for a multigraph or digraph
```

Out[26]:

```
(14.464646464646465,
 'There are no indegrees for undirected graph',
 'There are no outdegrees for undirected graph',
 0.024433524433524433,
 'Is not defined for a multigraph or digraph',
 'Is not defined for a multigraph or digraph')
```

In [30]:

```python
# Create and run voter model again on it
num_runs = 10
num_iters_list = []
status_count_over_time_list = np.zeros((num_runs, 200000))
fraction_infected = 0.4

for i in range(num_runs):
    start_time = time.time()
    model = create_voter_model(g, fraction_infected)
    num_iters, status_count_over_time = calculate_voter_convergence(model)
```

```
    status_count_over_time = np.array([status_count_over_time[j] if j < len(status_count_c
    num_iters_list.append(num_iters)
    status_count_over_time_list[i] = status_count_over_time
    print('convergence of voter model', i, time.time()-start_time, num_iters)

status_count_over_time_list_truncated = np.zeros((num_runs, max(num_iters_list)))
for i in range(num_runs):
    status_count_over_time_list_truncated[i] = status_count_over_time_list[i][:max(num_ite
status_count_over_time_list = np.zeros((num_runs, max(num_iters_list)))
status_count_over_time_list = status_count_over_time_list_truncated
print(status_count_over_time_list)
```

```
convergence of voter model 0 6.199850797653198 20685
convergence of voter model 1 15.39078402519226 51192
convergence of voter model 2 5.17496657371521 16985
convergence of voter model 3 17.389008045196533 57904
convergence of voter model 4 3.6891841888427734 12246
convergence of voter model 5 5.120919704437256 16970
convergence of voter model 6 12.67243766784668 42049
convergence of voter model 7 1.9656965732574463 6494
convergence of voter model 8 11.796181440353394 39361
convergence of voter model 9 3.620124340057373 12230
[[118. 118. 117. ...   0.   0.   0.]
 [118. 118. 119. ...   0.   0.   0.]
 [118. 118. 118. ...   0.   0.   0.]
 ...
 [118. 118. 118. ...   0.   0.   0.]
 [118. 118. 118. ... 297. 297. 297.]
 [118. 118. 119. ...   0.   0.   0.]]
```

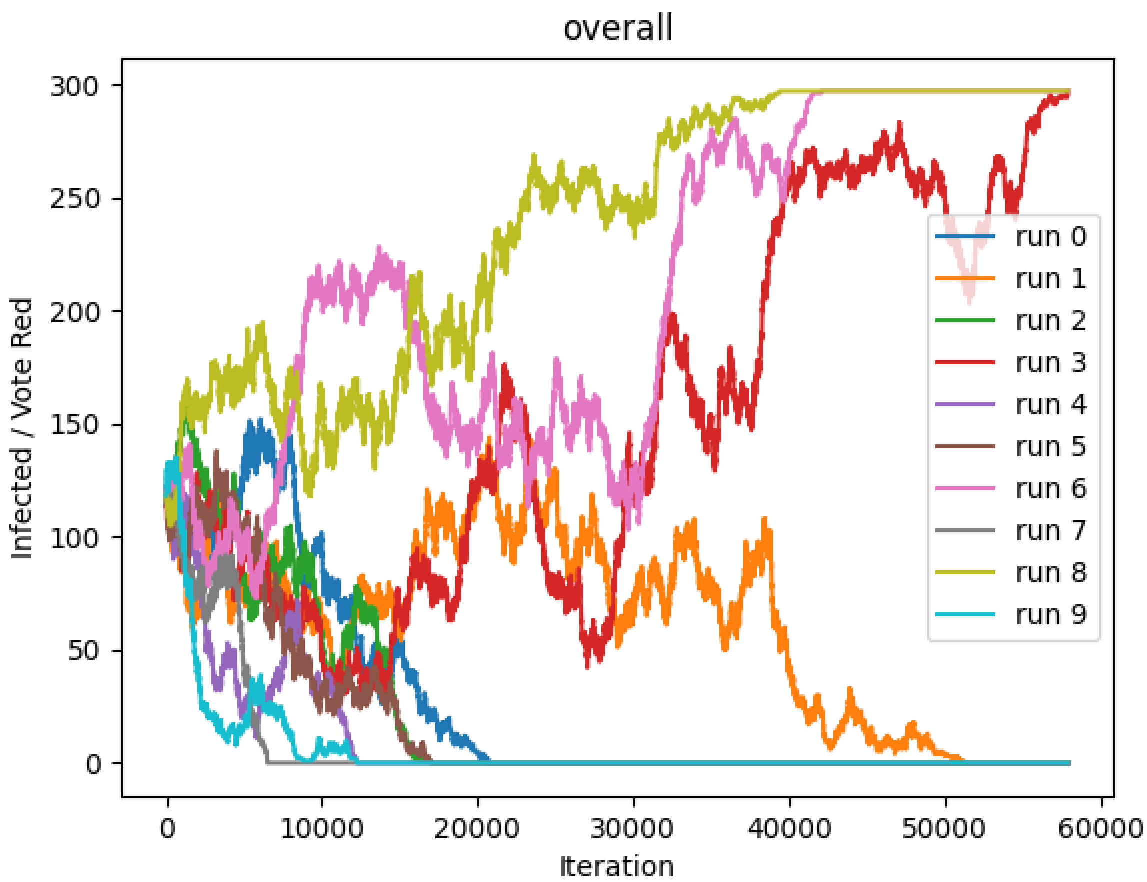
In [31]:

```
status_count_over_time_0 = []
status_count_over_time_1 = []
status_count_over_time_other = []
for i in range(len(status_count_over_time_list)):
    if status_count_over_time_list[i][-1] == len(g.nodes):
        status_count_over_time_1.append(status_count_over_time_list[i])
    elif status_count_over_time_list[i][-1] == 0:
        status_count_over_time_0.append(status_count_over_time_list[i])
    else:
        status_count_over_time_other.append(status_count_over_time_list[i])

status_count_over_time_0 = np.array(status_count_over_time_0)
status_count_over_time_1 = np.array(status_count_over_time_1)
print(status_count_over_time_other)
```

```
[]
```

In [32]:

```
plot_mean_and_bootstrapped_ci_multiple(input_data=[np.transpose(x) for x in status_count_c
print("Number of runs: {}".format(num_runs))
print("Number of runs that converge to red: {}".format(len(status_count_over_time_1)))
print("Number of runs that converge to blue: {}".format(len(status_count_over_time_0)))
```




```
Number of runs: 10
Number of runs that converge to red: 3
Number of runs that converge to blue: 7
```

In [ ]: