Opinion Dynamics on Static Network Model

June 21, 2018

0.1 Selcan Mutgan and Mike Thompson-Brusstar

selcan.mutgan@liu.se ; mrthomp@umich.edu

0.1.1 Santa Fe Institute GWCSS 2018

> Consider the following situation: Agents, connected to one another via a network, want to exchange information. Each agent has a choice of whether to send high or low quality information, and it takes some time to verify the quality. Model how information and trust flows on the network (with and without the possibility of collusion). Model, using whatever techniques you wish, the above scenario. Explicitly state your model and key assumptions. Summarize key results. Suggest some potentially interesting future directions and questions for the model. Suggest some standard social science scenarios that could be usefully modeled using such a process.

0.2 Introduction

In this study, we explore the dynamics of information diffusion on static networks where trust is at the center of information exchange. Senders of information in our model differentiate between recipients who they *trust* and recipients who they do *not* trust when they share their 'opinion.' Recipients likewise distinguish between *trustworthy* and *not trustworthy* senders. We model different decision processes for both senders and recipients that are dependant on these trust evaluations. Further, agents take time to process information that came from untrustworthy sources, and are able to update trust values as the model proceeds.

These modelling choices were motivated by our assumption that the way actors incorporate information does not only depend on the information itself but also on the structure of the network and people's trust of their 'neighbors.'

We find that the extent of connectedness in the network and the exogenously set threshold for determining the trustworthiness of signal recipients are key to the dynamics of our model.

0.3 Methodology

We use an agent-based model to explore the dynamics laid out in the homework problem; in particular, we use the python package 'mesa' as a framework for our agents and the model dynamics.

We initialize an Erdös-Rényi type random network with 200 nodes in all specifications. Agents are activated in random order, and the randomness in each of the model runs is seeded with a different value. Agents only observe their local network (i.e. their immediate neighbors), each of which is randomly assigned a trust value from a uniform distribution between zero and one.

Furthermore, we endow all agents with an 'opinion' represented as a random bit string of length 3 (e.g. [011]).

In each step of the model, agents iterate across their immediate neighbors. For each immediate neighbor, a sender(EGO) checks to see if her trust value for that neighbor is above or below the exogenously assigned trust threshold level. If the trust value for that neighbor is higher than the threshold, EGO sends the chosen ALTER their whole and unaltered 'opinion.' If EGO does not trust ALTER, EGO flips a coin. If heads, EGO sends her 'true' opinion (a 'high' quality signal). Otherwise, she sends a randomly generated bit string (a 'low quality' signal). As explained above, ALTER then decides whether to blindly accept and incorporate EGO's opinion or to 'verify' it in her next move.

ALTER takes in the signal and evaluates if her trust value for the sender is above or below the threshold. If ALTER trusts the sender, she updates her 'opinion' in the following manner: each bit in ALTER's bit string is changed to the value held by EGO with probability proportional to her trust value.

If, on the other hand, ALTER does not trust the sender, she steps out of the model into a 'verifying' state for one round. During this round, ALTER will then observe the true 'quality' of the information sent by EGO. If it is a 'true' or 'high quality' information, she updates her 'opinion' as above and revises her trust value for the sender up by an exogenously set value. Throughout our model runs, this value is set to 0.2. If the information sent is of 'low quality' (as below), ALTER discards the opinion and revises the trust value for EGO down by twice the exogenously set value. Trust values are prevented from exceeding 1 or going below zero.

These decisions are designed to capture the fact that the quality of information is not immediately clear to the reciever, and that, for example, if ALTER trusts the sender above the assigned threshold, she will still incorporate the information *regardless of its quality*.

We explore the parameter space by testing nine paired values of trust threshhold and average degree of the network, roughly corresponding to 'low', 'medium', and 'high' values for each parameter. One hundred runs of the model are averaged for each of the pairs of parameter values that we explore.

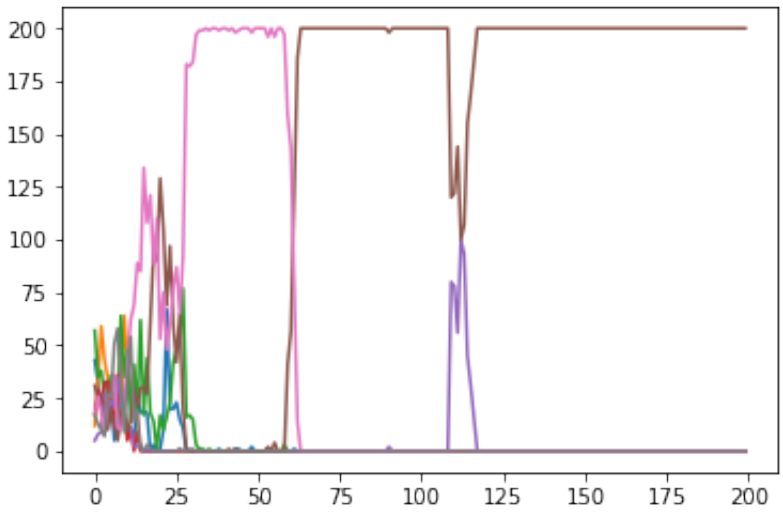
We are particularly interested in exploring two outcomes: the convergence of opinion and the average level of trust in the network. We extract measures of both for each model run. Our admittedly rough measure of opinion convergence is the percent of the population with the most common bit-string (of which there are 8 possibilities in our model: 2^3).

0.4 Results

Altogether, we have nine sets of results: trust threshhold and average degree are varied together, each at three levels. Average degree is varied between 5 and 20 and trust threshold is varied between 0.2 and 0.7 as can be seen in the table below.

avg degree	
	low
low	5, 0.2
medium	10, 0.2
high	20, 0.2

Figure 1: Modal Run of the Model Above (figure 1), we present a typical 'run' of the model where the threshold is set to 0.5 and the average degree is set to 20. Each line in the line plot



'Figure 1.'

represents the frequency of different opinions in the population. As mentioned above, there are 8 possible opinion types. On average, we see that the run begins with competition between various opinions, but that eventually a single opinion comes to dominate the population. This opinion is then occasionally overthrown by another opinion. This is despite the fact that the average level of trust across agents is stable throughout the duration of the model run (as shown in figure 2 below).

Figure 2: Trust 'change' across model runs Finally, we show in Figure 3 that across the nine different parameter pairs that we test, there is noticable variation in the speed with which an opinion becomes dominant that is more sensitive to the average degree of the network than it is to the change in the trust threshold. An opinion becomes dominant faster in populations with higher levels of connectedness and lower trust thresholds which determine the quality of sent information. Qualitatively, the cyclical behavior evident in Figure 1 is present in many parameter values of the model, but the length of the dominance varies with the average degree and trust threshhold level. This behavior is smoothed out in the 100 run averages presented in Figure 3.

Figure 3: Opinion Dominance Dynamics

0.5 Discussion and Future Directions

0.5.1 Discussion

In this study we are able to show that even simple agent based models regarding opinion dynamics with rather simple decision rules can generate non-trivial outcomes. Despite the homogeneity of the population and the decision rule that depends on an exogenously and non-varying threshold, our models produced a dynamic system where populations converge to a single opinion or create cyclical system behavior where two or more opinions race against each other.

We believe that this model can be applied to understand the dynamics of opinion diffusion in affiliation networks, classrooms, or other settings where network structure is exogenous and

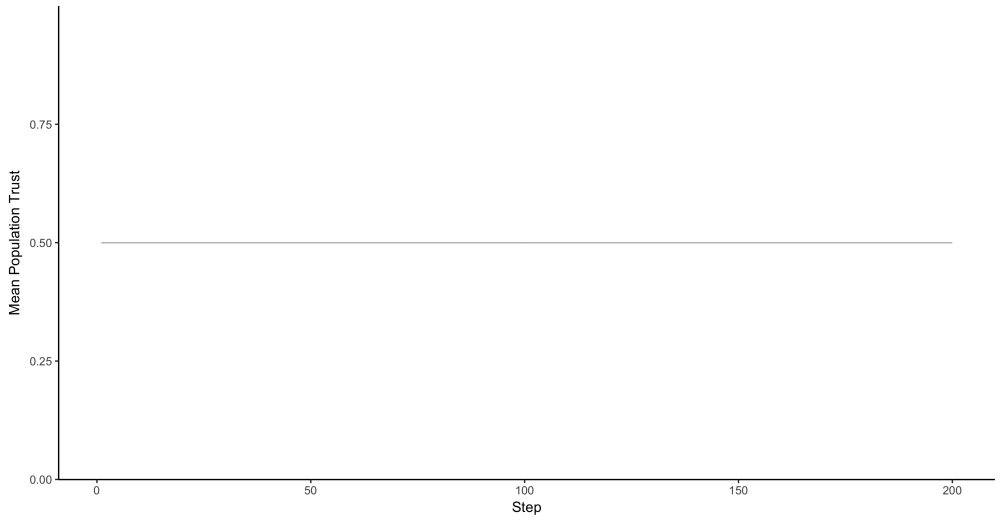


Figure 2.

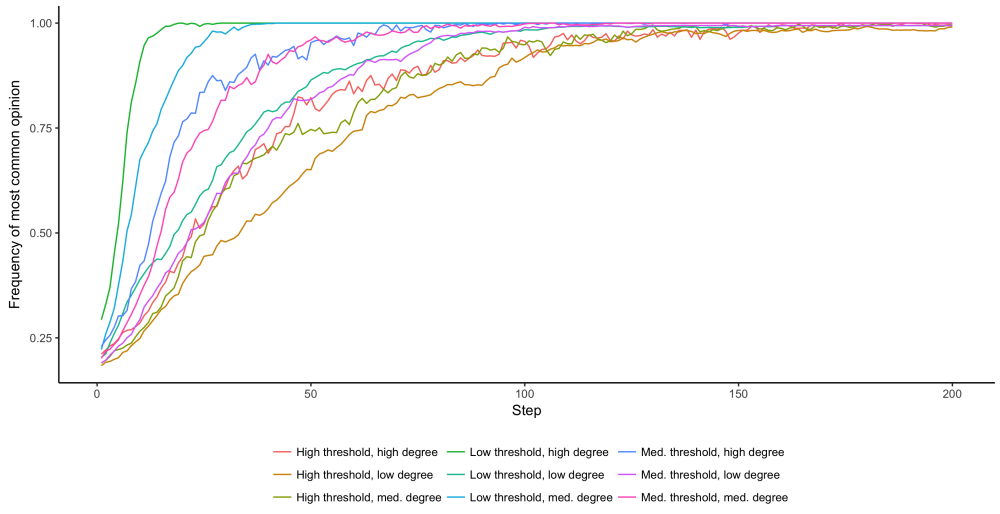


Figure 3.

difficult to alter. One other example that this model can speak to is gossip networks, where opinion dynamics are possibly highly dependant on the network structure as well as the trust each person has for their 'friends.' Additionally, one could also extend the model to explore collective decision dynamics in newly formed groups: the model is initialized with diverse yet unpolarized opinions and exogenous trust values, which might be analogous to trust based on simple signals of background or social status.

0.5.2 Future Directions

The model presented in this study is flexible enough to support a great variety of future inquiries. First, we are interested in exploring the dependence of our above results to the distribution of initial trust across an agent's neighbors. Currently these are drawn from a uniform distribution between zero and one, which may drive the stability of average trust across the period of the model. This dependence on the initial conditions or initial distribution could potentially be explored via the use of an Active Nonlinear Test (ANT).

Secondly, we are interested in how the model would change if network structure were altered in several relevant ways. It is reasonable in the modelling of an opinion dynamic or gossip diffusion across a network to assume that the network might not be stable over the steps of the model. An agent may lose ties to agents who she no longer trusts or may be interested in forming new ties with agents who are trusted by those who she trusts (e.g. transitive triads). Additionally, the initial network structure could be a factor in the outcomes we observe, and further exploration of other network structures might shed additional light on how some 'opinions' come to dominate the population.

Thirdly, the dependence of the model results on the length of the bitstring and its diverse character should be further explored. One possibility is to look at a bimodal initialization of the bit-strings, such as in a polarized political setting. Although the use of 3-bit strings aid interpretability of the model, it constrains the size of the possible 'opinion space.' Further diversity may make it more difficult for a single opinion to dominate.

Finally, one could also explore behavioral rules beyond trust that motivate agents' decision-making. It is easy to imagine that beyond simple updating of the trust, some punishment might take place when an agent discovers she has been sent low-quality information. The current parameterization of the model assumes that the trust 'cost' for sending low quality information is lower than the trust 'reward' when an agent 'verifies.' In the same manner, the model could also be extended by implementing a more thorough representation of reputation, in which cost is not internal to a betrayed recipient, but where other agents are also able to observe this betrayal and update their trust values accordingly. Furthermore, it may be the case that the model overstates the importance of single ties – one could compare the current results to a model in which agents update their 'opinions' based on an aggregation or average of their neighbors' opinions.

0.6 Code Appendix

```
In [1]: import random
        from random import choice
        from mesa import Agent, Model
        from mesa.time import RandomActivation
        from mesa.space import NetworkGrid
        import math
        from enum import Enum
```

```
import networkx as nx
import numpy as np
import statistics
from collections import Counter
import pandas as pd
import matplotlib.pyplot as plt


"""defining a function for generating the 'bit strings'
for the agents. that took too long."""

def rbitstring(n,bits):
    seq = list(bits)
    out = [choice(seq) for i in range(n)]
    return(out)

""" m dimensions"""
m = 3

""" initial trust threshold"""
th = 0.2

"""update amount"""
update_amt = 0.2

num_nodes = 200

initial_trust = []
mean_trust_step = pd.DataFrame()
step_count = []
update_high = []
update_low = []
mean_trust_agent_step = []
var_trust_step = pd.DataFrame()
counts = pd.DataFrame()
most_common = pd.DataFrame()

class human(Agent):
    """ An agent representing a single person."""
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.abit_string = rbitstring(m,[0,1])
        self.sig_qual = []
        self.State = 'UPDATING'
        self.uncertain_ids = [] #list of agents that sent signals to this a
        self.prev_bit_signals = []
        trust_val = []
        for i in range(num_nodes):
```

```
            trust_val.append(random.random())
        self.trust_val = trust_val

    def get_neighbors_nodes(self):
        self.neighbors = model.grid.get_neighbors(self.pos, include_center

    def assign_trust(self):
        self.neighbor_trust = dict(zip(self.neighbors,self.trust_val[0:len

    def updating(self):
        if self.neighbors == []:
            pass
        else:
            for i in range(len(self.neighbors)):
                dec = []
                if self.neighbor_trust[self.neighbors[i]] > th:
                    dec = 'high trust'
                elif self.neighbor_trust[self.neighbors[i]] <= th:
                    dec = 'low trust'
                for a in model.schedule.agents:
                    if a.unique_id == self.neighbors[i]:
                        B = a
                if dec == 'high trust': # A trusts B

                    if B.neighbor_trust[self.unique_id] > th: # if B trusts
                        for i in range(len(B.abit_string)):
                            if random.random() <= B.neighbor_trust[self.uni

                                B.abit_string[i] = self.abit_string[i]

                    elif B.neighbor_trust[self.unique_id] <= th: # if B doe
                        B.State = 'VERIFYING'
                        B.sig_qual.append('HIGH')
                        B.uncertain_ids.append(self.unique_id)
                        B.prev_bit_signals.append(self.abit_string)

                elif dec == 'low trust': # if A does NOT trust B

                    if random.random() >= 0.5: # coin flip -- high signal
                        if B.neighbor_trust[self.unique_id] > th: # B trust
                            for i in range(len(B.abit_string)):
                                    if random.random() <= B.neighbor_trust[
                                        B.abit_string[i] = self.abit_string
                        elif B.neighbor_trust[self.unique_id] < th: # B doe
                            B.State = 'VERIFYING'
                            B.sig_qual.append('HIGH')
                            B.prev_bit_signals.append(self.abit_string)
                            B.uncertain_ids.append(self.unique_id)
```

```
                    else:
                        rdm = rbitstring(m,[0,1])
                        if B.neighbor_trust[self.unique_id] > th: # B trust
                            for i in range(len(B.abit_string)):
                                if random.random() <= B.neighbor_trust[self
                                    B.abit_string[i] = rdm[i]
                        elif B.neighbor_trust[self.unique_id] < th: # B doe
                            B.State = 'VERIFYING'
                            B.sig_qual.append('LOW')
                            B.prev_bit_signals.append(rdm)
                            B.uncertain_ids.append(self.unique_id)


    def verify(self):

        if self.neighbors == []:
            pass
        else:

            sig_ids = dict(zip(self.uncertain_ids,self.sig_qual))

            bit_ids = dict(zip(self.uncertain_ids,self.prev_bit_signals))

            for i in self.uncertain_ids:

                if sig_ids[i] == 'HIGH':
                    update_high.append('high')
                    # update the trust for agent A

                    if self.neighbor_trust[i] < 1 - update_amt:
                        self.neighbor_trust[i] += update_amt
                    else:
                        self.neighbor_trust[i] = 1

                    # incorporate info

                    for j in range(len(self.abit_string)):
                            if random.random() <= self.neighbor_trust[i]:
                                self.abit_string[j] = bit_ids[i][j]

                    self.State = 'UPDATING'

                elif sig_ids[i] == 'LOW':

                    if self.neighbor_trust[i] > 2 * -update_amt:
                        self.neighbor_trust[i] += 2 * -update_amt
                    else:
                        self.neighbor_trust[i] = 0
```

```
                    self.State = 'UPDATING'

                    update_low.append('low')

            self.sig_qual = []
            self.prev_bit_signals = []
            self.uncertain_ids = []

    def step(self):
        if model.counter == 0:
            initial_trust.append(np.mean(self.trust_val))
            for a in model.schedule.agents:
                a.get_neighbors_nodes()
                a.assign_trust()
        if model.counter > 0:
            pass
        if self.State == 'UPDATING':
            self.updating()
        elif self.State == 'VERIFYING':
            self.verify()


class info_model(Model):
    """A model with some number of agents in a network."""
    def __init__(self, num_nodes, avg_node_degree):
        self.counter = 0
        self.num_nodes = num_nodes
        prob = avg_node_degree / self.num_nodes
        self.G = nx.erdos_renyi_graph(n=self.num_nodes, p=prob)
        self.grid = NetworkGrid(self.G)
        self.schedule = RandomActivation(self)
        # Create agents
        for i, node in enumerate(self.G.nodes()):
            a = human(i, self)
            self.grid.place_agent(a, node)
            self.schedule.add(a)

    def step(self):
        self.schedule.step()
        self.counter +=1
        mean_trust_step_l.append(np.mean([np.mean(a.trust_val) for a in mod
        step_count.append(self.counter)
        # get the counts of each possible bit string at each step of the mo
        agent_strings = [a.abit_string for a in model.schedule.agents]

        cnt = Counter()
        for z in agent_strings:
```

```
                    cnt[str(z)] += 1
                counts_l.append(cnt)
                most_common_l.append(cnt.most_common(1)[0][1] / 200)

        #       """Advance the model 1 step."""
```

In [2]:
```
        # counts_plot = []
        # for i in range(0,100):
        #     random.seed(a = i)

        #     model = info_model(num_nodes = 200, avg_node_degree = 5)

        #     initial_trust = []
        #     mean_trust_step_l =[]
        #     step_count = []
        #     update_high = []
        #     update_low = []
        #     mean_trust_agent_step_l = []
        # #     var_trust_step_l = []
        #     counts_l = []
        #     most_common_l = []

        #     for j in range(200):
        #         model.step()

        #     most_common[str(i)] = most_common_l
        #     counts_plot = counts_l
        #     mean_trust_step[str(i)] = mean_trust_step_l
```