Emergence in a real scale free network

Federico Malizia

20/04/2020

UNIVERSITÀ DEGLI STUDI DI TORINO

1 Introduction

The following report will illustrate an agent based modeling work representing the dynamics of a social network. In the first work with Professor Terna we manily focues on the emergence of a network (possibly scale-free) from the interaction of various nodes due to the similarities between the various users, without considering any type of information spreading between the various agents. The extension in this work concerns an addition of a new class of agents (*Supeuser*) and a greater complexity of the behavior of the agents in the "building" phase of the model, like the memory of an agent and the possibility of being able to break the connection with a node, and an additional exchange of information between them up to the observation of the echo chambers obtained as a final result at equilibrium obtained by the tuning of some parameters entered manually by the user. In literature there are many works where it is being studied, and tried to replicate, the dynamic structure of a social network. There are many machine learning models which try to predict the evolution of the network from a pre-established pattern. The research is still open to a real formulation of a model that replicates the intrinsic structure of a true social network. There are many studies applied to rumor spreading within social networks and all these have highlighted the difficulty of studying the intrinsic mechanisms of social media functioning, since there are various local mechanisms, for example degree of influence and degree of response of individual users to repeated exposure to all information and long-term and short-term interactions plus the existence of various sub-layers of the network. The following study will be formalized both on rumor spreading, as the nodes will propagate binary informations and how users approach with different users tastes and geographical positions, reproducing a dynamical study of a social networks involving multiple users and how they can interract between each other from the first stages of life of the system.

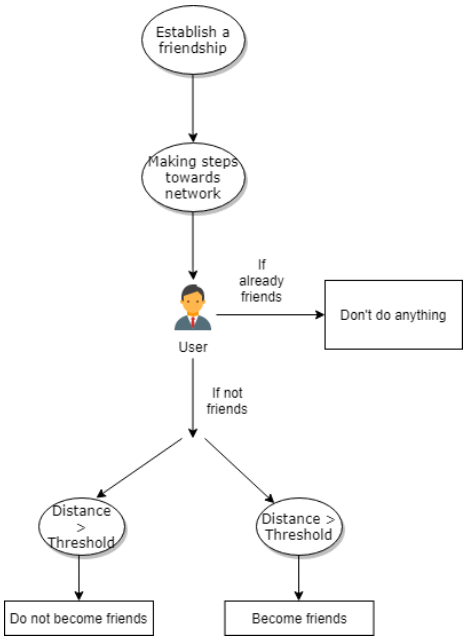


Figure 1: Logic representation of a node interacting with *Users* in the Econofisica work

2 The Model

Agents are generated randomly and they have the possibility to approach each other, and consequently also the ability to "respond" to the approach, using the assumption of reciprocity between the two agents. Names are randomly associated to each agent, coming from a list of typical Italian names. From the early stages of their creation they possess unique characteristics that are crucial for the evolution of the network. These unique features have been initialized within the agent in a multidimensional vector form, where each dimension represents an attribute that is assigned to the agent. Those attributes are: district of origin, age, political party and personality. In this study were used Italian provinces and the results of the 2019 european election in Italy. To each one of this attribute, has been associated a real number representing the coordinate that will be assigned to the user's multidimensional features vector. The agents also, once created, have some singular attributes that can be entered manually at the start of the simulation, two of there are a coefficient of shyness and a tolerance coefficient. The first one will be crucial in order for the agent to take, or not, the decision to approach one of the users near the time, the second one instead represents a threshold trought a measure of sympathy, which represents what the agents are able to tolerate with respect to the geographical differences and characteristics of other network agents while approaching each other. An-

other three parameters are: loquacity, hostility and foolishness. Those three parameters regulate the rumor spreading between the nodes, the first one is the tendency of a node to start a conversation, hostility is the tendency to a node to defend is opinion and the foolishess instead represent the tendency of a node of being manipuleted by the opinion of the approached node. The interaction between two nodes strongly depends on the distance between the two feature vectors, two nodes with a minimum distance have a very low probability of arguing despite the different polarities, while two nodes with a greater distance have a greater probability of quarreling given the differences and to remove their connection within the dynamic of the model. Each node will be initialized with a random opinion value between -1 and +1, as an initial state of polarization, corresponding to a value of sentiment towards a hypothetical topic of discussion in the dynamics. For a given user i, with a number a_i of interaction between nodes (defined as his/her activity), we can associate a time-ordered set of sentiments S_i and define his polarization P_i:

$$P_i = \frac{\sum_{t=0}^{a_i} s_t}{a_i}$$

which is bounded in the interval [-1; +1]. It has been developed a metric S for the correlation between node connections and polarities among the interactions.

$$S = \frac{\sum_i <S^{nn}>}{N}$$

where S^{nn} is the fraction of friends who shares the same opinion within each node, and N is the number of nodes. S is bounded in the interval [0; +1], a value of 0 means that the nodes does not share the same opinion among friends, and one means that every node shares the same opinion with their friends. A value of one would mean that the network is divided into two perfect isolated subgraphs where the two poles do not communicate with each other. Once the simulation has been completed, the agents belonging to the class *User* will be divided into 5 categories, so-called "states" of the agents, depending on the degree they got within the network: *Segregated, asocial, sociable, cool and influencers.*

2.1 Agents

Agents assigned to nodes are instances of a class called User

```
class User():
   def__init__(self,name,age,district,party,hostility,preferences,unique_id,
   shyness,loquacity,foolishness):
      self.name = name
      self.age = age
      self.district = district
      self.party = party
      self.preferences = preferences
      self.unique_id = unique_id
      self.shyness = shyness
      self.friends_count = 0
      self.friends_dict = {}
      self.subscriptions_count = 0
      self.subscriptions_dict = {}
      self.demand = random.random()
      self.sympaties_dict = {}
      self.hate_list = []
      self.opinion = np.random.choice([-1,1])
      self.loquacity = loquacity
      self.hostility = hostility
      self.foolishness = foolishness
      self.interaction = 0
      self.polarities = []
      self.polarization = 0
      self.state = "segregated"
```

The agents are randomly generated, each of them is associated with the initial status of "segregated", since at the time of their creation they are deprived of friends. The "District" and "Party" attributes will be assigned using probability values depending on the population distribution in the Italian provinces and the election results of the European elections of 2019. In this way, the model will be more realistic as the probability of having a greater number of users from more inhabited districs is higher.

The attributes will be converted into a multidimensional vector thanks to python dictionaries; to each province or political party, a numerical value is associated (in ascending order by geographical proximity regarding the provinces and ordered by "political closeness" regarding the parties). A random number will represents the personality attribute. Once the agents has been created, an undirected graph will be initialized using the NetworkX library, then to each node an agent is assigned. The graph represents the dynamic of a social network, where the edges are precisely the friendships that have been established over time, and the nodes represent the agents. At the initial stage of the network, the nodes will be without any link; within the simulation, one agent at a time will move with small steps, looking around for another agent to interact with,

x	Province	p	Popolazione
95	Agrigento	0,0072046599	434870
3	Alessandria	0,0069795754	421284
58	Ancona	0,007807017	471228
1	Aosta	0,0020819573	125666
49	Arezzo	0,0056768817	342654
59	Ascoli Piceno	0,0034324148	207179
4	Asti	0,003555991	214638
83	Avellino	0,0069302377	418306
76	Bari	0,02074227	1251994
77	Barletta - Andria - Trani	0,0064614634	390011
29	Belluno	0,0033623513	202950
84	Benevento	0,0045894646	277018
15	Bergamo	0,0184658447	1114590
5	Biella	0,0029089848	175585
27	Bolzano	0,0088002319	531178

Figure 2: A small portion of districts dataset file

once found, the process is divided into two phases: there will be the approaching agent and the agent being approached. The extension in this work compared to the previous one was to give agents the ability to access the friend list of the various nodes within the network, interacting with "friends of friends" and adding the "hate list" , the agents can have a memory of the nodes with which they previously had a negative interaction, thus ignoring the aforementioned nodes contained in the list for future interactions. By accessing the friend list of the various nodes in the network, users can decide to terminate the relation with a node if there are nodes from the first's hate list in the friend list of the second one, with a probability depending on the distance between the vector features of the two nodes. Another important addition is the *Superuser* class of agents. The *Superuser* class represent the content creators on the social network, comparable as the Facebooks pages. The differences between *User* and *Superuser* are that Superuser will never interact on his own initiative, but the interaction will take place only by the will of the User, Superuser during the interaction could try to influence user opinion by sharing contents in the network.

```
class SuperUser():
   def __init__(self,topic,unique_id,quality):
      self.topic = topic
      self.unique_id = unique_id
      self.subscribers_count = 0
      self.subscribers_dict = {}
      self.ban_list = []
      self.quality = random.random()# quality
      self.opinion = np.random.choice([-1,1])
      self.state = "p" #superuser

   def get_id(self):
      return(self.unique_id)
```

```
  def get_user_state(self):
     return(self.state)

  def get_topic(self):
     return(self.topic)


  def got_approached(self,subscriber,success):
     if success == True:
        self.add_subscriber(subscriber)
     if success == None:
        self.remove_subscriber(subscriber)


  def add_subscriber(self,subscriber):
     self.subscribers_count += 1
     self.subscribers_dict[subscriber.get_id()] = subscriber

  def remove_subscriber(self,subscriber):
     if subscriber.get_id() in self.subscribers_dict.keys() and
         subscriber.get_id() != self.get_id():
        self.subscribers_count -= 1
        del self.subscribers_dict[subscriber.get_id()]
        self.ban_list.append(subscriber)
     else:
        pass
```

Nodes in the model can interact in several ways, first they will try to establish friendships among each others, or users can make a subscription with the superusers. Another way to interact for nodes is to share conents, which are represented as binary numbers.

```
def look_around(self,N):
   if self.friends_count <= 2:
      randomsteps = [i for i in range(int(-N/10),int(N/10))]
      self.step = np.random.choice(randomsteps)
   if self.friends_count > 2 and self.friends_count <=5:
      randomsteps = [i for i in range(int(-N/5),int(N/5))]
      self.step = np.random.choice(randomsteps)
   if self.friends_count > 5:
      randomsteps = [i for i in range(-N,N)]
      self.step = np.random.choice(randomsteps)
   return(self.step)
```

The *make_your_decision* function rules the decision of the approaching agents at the start of the approaching stage. If both agents are not friends, the approaching one will try to establish a friendship. During the approach phase, a random number will be generated that if it is greater than or equal to the shyness degree of the approaching agent, then the approach will take place, oth-

erwise the agent will withdraw. If the agents that has been selected from the approaching one is a Superuser, the ones who represent the content spreaders in the network, the agent will decide between subscribe or to interact within the Superuser contents.

```
def make_your_decision(self,friend):
   decision = False
   if friend.state == "p":
      if friend.get_id() in self.friends_dict.keys():
         decision = "t" # talk
      else:
         decision = "s" #submit subscription
   else:
      if self.friends_count != 0:
         if random.random() <= self.loquacity:
            decision = "t" #talk
         else:
            if random.random() >= self.shyness:
            decision = True
      else:
         if random.random() >= self.shyness:
            decision = True
   return(decision)
```

```
def approach(self,friend,threshold,decision):
   success = False
   if decision == True:
      if friend.get_id() in self.friends_dict.keys() and friend.state !=
          "p" and  len(friend.friends_dict) != 0 and friend.get_id() !=
          self.get_id():
         success = self.check_friend_list(friend) #"friend"
      if friend.get_id() not in self.friends_dict.keys() and
          friend.state != "p" and  friend.get_id() not in self.hate_list
          and friend.get_id() !=  self.get_id():
         if self.sympathy(friend) <= (threshold*100):
            success = True
            self.add_friend(friend)
         else:
            self.hate_list.append(friend)
   elif decision == "t" and friend.get_id() in self.friends_dict.keys()
       and friend.get_id() != self.get_id() or friend.get_id() in
       self.subscriptions_dict :
      if friend.state != "p": #superuser
         success = self.talk_to_friends(friend,threshold)
      else:
         success = self.content_interaction(friend)
   elif decision == "s":
      success = self.submit_subscription(friend)
   return(success)

def got_approached(self,friend,success):
   if success == True:
      self.add_friend(friend)
   if success == None:
      self.remove_friend(friend)
```

The *approach* function is the one who rules the interaction between nodes, it is ruled by the "record" between the two nodes. If the resulting decision from *make_your_decision* function is to try to establish a new friendship. If there is still a relation between nodes, if the approached node is from the *User* class, the approaching one will first look at its friends list and if he finds someone who hates he will automatically remove the friendships with the approached node previously selected. Otherwise, the approaching node will pick a new node from the friends list of the approached one, and it will repeat the loop again. This procedure is ruled by a recursive function outside the agents class. If the approached node is a *Superuser* and there is already a connection between them, the approaching node will interact with the contents of the *Superuser*. When an agent decides to approach for the first time with some other user, the two agents will begin to interact with each other. Using a distance function, the agents will become friends if only the Euclidean distance between the agents features vectors will be less than a "tolerance" value.

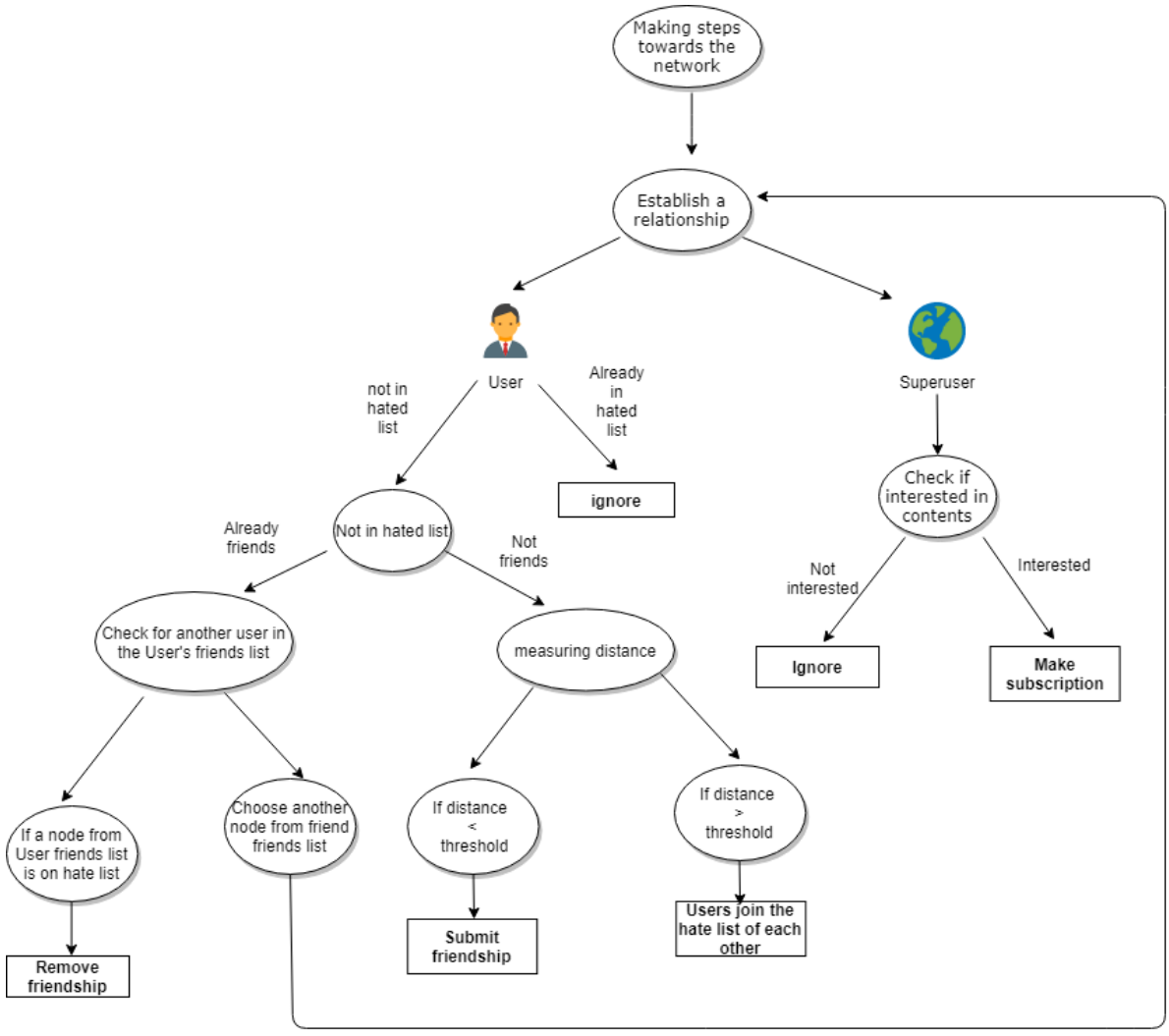


Figure 3: Logic representation of a node approaching to *Users* and *Superusers* contents.

```
 def distance(self,friend):
   v_1 = np.array(self.features, dtype = "int")
   v_2 = np.array(friend.features, dtype = "int")
   sympathy_degree = np.linalg.norm(v_1 - v_2)
   self.sympaties_dict[friend.get_id()] = sympathy_degree
   return(sympathy_degree)

 def add_friend(self,friend):
   self.friends_count += 1
   self.friends_dict[friend.get_id()] = friend

def remove_friend(self,friend):
   if friend.get_id() in self.friends_dict.keys() and friend.get_id()
       != self.get_id():
      self.friends_count -= 1
      del self.friends_dict[friend.get_id()]
   else:
      pass

 def submit_subscription(self,friend):
   success = False
   if friend.get_id() not in self.subscriptions_dict.keys():
      if (any([item in friend.topic for item in self.preferences]))
          == True and self.demand <= friend.quality:
        success = True
        self.add_subscription(friend)
   return(success)

 def add_subscription(self,friend):
   self.subscriptions_count += 1
   self.subscriptions_dict[friend.get_id()] = friend

def remove_subscription(self,friend):
   self.subscriptions_count -= 1
   del self.subscriptions_dict[friend.get_id()]

def check_friend_list(self,friend):
   if any([item in self.hate_list for item in
       friend.friends_dict.values()]) == True or any([item in
       self.hate_list for item in
       friend.subscriptions_dict.values()]) == True:
      success = None #remove friendship
      self.remove_friend(friend)
      self.hate_list.append(friend)
   else:
      success = "friend" #node will pick a new node from friend list
   return(success)

def pick_from_friend_list(self,friend):
```

```
friend_r = random.choice(list(friend.friends_dict.values()))
return(friend_r)
```

The second way the nodes can interact is between the contents that every node, both user and superuser, shares as binary numbers -1 and +1. If the opinions among *users* are discordant they will argue and the probability of losing their connection depends on the distance between their feature vectors. Subsequently, if the two nodes maintain their relationship, the approaching node can be manipulated by the opinion of the second node, with a probability related to its *foolishness* coefficient. The interaction between *user* and *superuser* is pretty similar, *user* could not like the content created by the *superuser* and this would lead to an argue between the two nodes depending from the *hostility* rate of the approaching node, that could lead the node to share the superuser content or to a possible ban from the superuser.

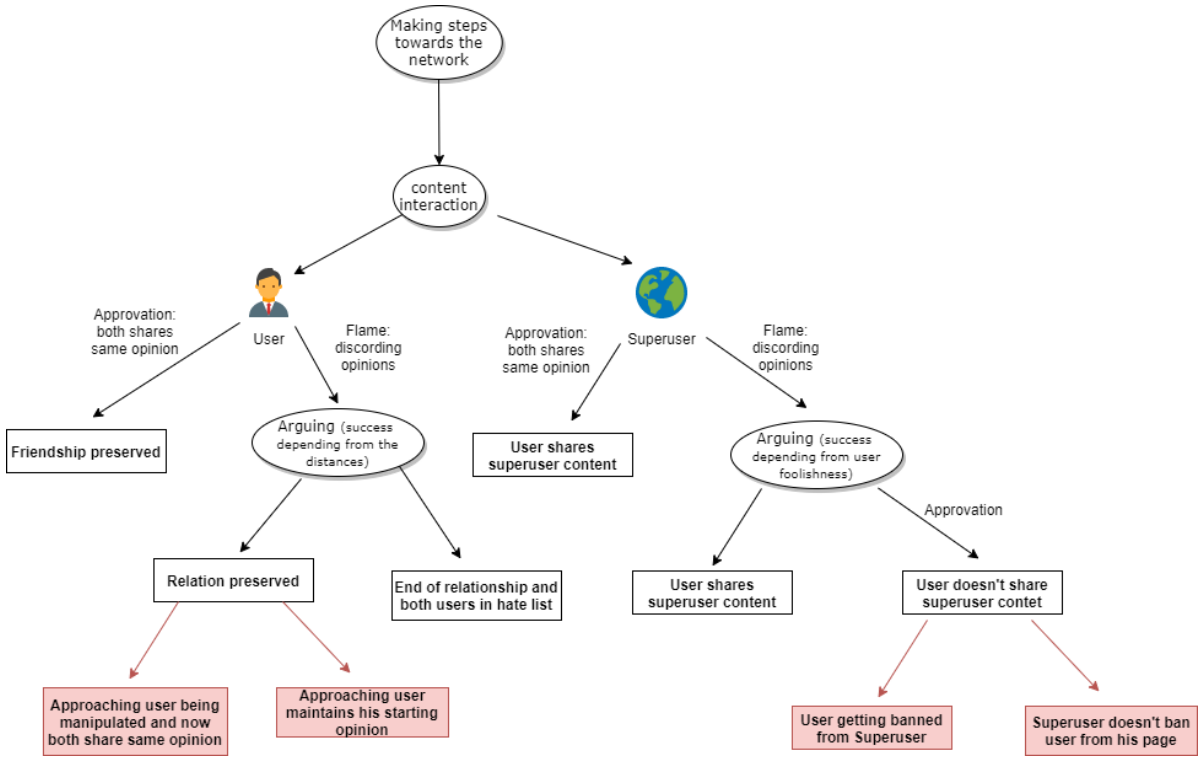


Figure 4: Logic representation of nodes sharing opinions

```
def talk_to_friends(self,friend,threshold):
success = False

if friend.get_id() in self.friends_dict.keys() and friend.state != "p"
    and friend.get_id() != self.get_id():
self.interaction += 1
if self.opinion != friend.opinion:
if self.sympathy(friend)/100 > random.random()
self.remove_friend(friend)
self.hate_list.append(friend)
success = None #remove friendship
if self.foolishness > random.random():
self.opinion = friend.opinion
self.polarities.append(self.opinion)
self.polarization = sum(self.polarities)/self.interaction
return(success)

def content_interaction(self,superuser):
success = False
if superuser.get_id() in self.subscriptions_dict.keys():
self.interaction += 1
content = np.random.choice([-1,1])
if content != superuser.opinion:
if self.hostility >= random.random():

self.remove_subscription(superuser)
self.hate_list.append(superuser)
success = None
else:
if self.foolishness > random.random():
self.opinion = content
self.polarities.append(self.opinion)
self.polarization = sum(self.polarities)/self.interaction
return(success)
```

2.2 States

The agents of class *User* will all be generated with the status of Segregates, since in the first moments of their generation they will not be connected with any friend. During the simulation, their status will be updated according to the number of friends they will have within the network compared to the maximum degree that can be observed inside it. Here is the list of all the five states that the nodes can have during the simulation:

- The nodes that will not be able to connect with anyone, will take the status of Segregated
- the nodes with a degree less than or equal to 25% of the maximum degree

of the network will take the status asocial.

- The nodes with a degree between 25% and 50% of the maximum value reached in the network, will take the name of Socievoli
- The nodes with a degree between 50% and 75% will take the status of Cool users.
- The nodes with a degree greater than or equal to 75% will be called "Influencers", or rather the hubs of the network.

2.3 User interface

At the beginning of the simulation there will be four parameters that can be chosen by the user, which will determine the evolution of the network.

- *Number of nodes*: it represents the number of nodes within the network and consequently the number of agents that will be randomly generated. A greater number of nodes will produce greater randomization and network amalgamation, as there will be more users and therefore greater behavioral diversity among individuals, but will require more computing power. It is not advisable enter a number of nodes larger than 300-400 units.
- *Shyness rate*: Represents the shyness degree of each individual node. It is possible to enter a value between 0 and 1 equal for every node of the network or by entering the value -1 to generate a different random value for each node of the network. The greater the value, the less the agents' tendency to approach with other network agents.
- *Tolerance rate*: Represents the degree of tolerance in the interaction between each pair of agents. Also in this case it will be possible to enter a value between 0 and 1 or initialize a different random value for each agent. Obviously, the lower the value, the less likely the agents to make friends with evident geographic and behavioral inequalities
- *Loquacity rate*: Represents the tendency of an agent to start an interaction with another agent from the class of *User* instead to looking for a new connection. Also in this case it will be possible to enter a value between 0 and 1 or initialize a different random value for each agent. Obviously, the lower the value, the less likely the agents to starts interaction within nodes.
- *Foolishness rate*: It represents the tendency of a node of being manipulated within the interaction with both User and Superusers, leading to a change of opinion states if the opinion of the two interacting nodes are discording. Also in this case it will be possible to enter a value between 0 and 1 or initialize a different random value for each agent. Lower the value, lower is the tendency of a node of being manipulated.

- *Hostility rate*: It represents the tendency of a node of argue against the Superuser content. It is strictly correlated to the *Democratiness* rate of *Superusers*. Greater the hostility value is, greater is the probability that the node could get banned from the Superuser contents.

- *Number of interactions*: It represents the number of interactions between nodes, successful or failing attempts to approaches, and is therefore the quantity that regulates the time of the simulation. It is advisable to enter a value equal to or greater than the maximum number of edges in an indirect network: $\frac{N(N-1)}{2}$ to observe the evolution of the network up to an equilibrium state between the nodes.

The agents state will be updated in real time according to the distribution of the nodes degree within the graph. In the simulation the nodes will have different colors depending on their state: black for *segregated*, blue for *asocials*, green for *sociable* users, orange for *cool* users, red for *influencers* representing hubs, fraction of nodes that own the higher number of edges within the network and yellow for *Superuser* class.

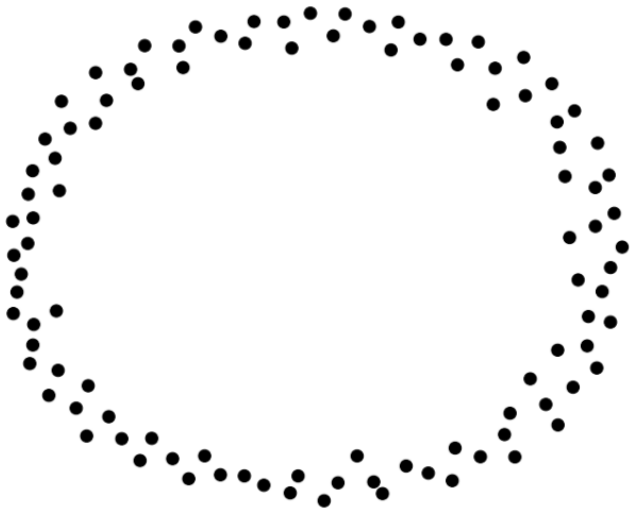


Figure 5: An example of simulation with 100 agents at early stages before they begin to approach. The black color nodes represents the segregate state.

3 Results

The following section will report some cases obtained by entering different parameters in the user interface. The network conformation is totally dependent on the randomization of the nodes, different simulations with the same number of parameters will not be equal since the parameters of the agents are randomly generated.

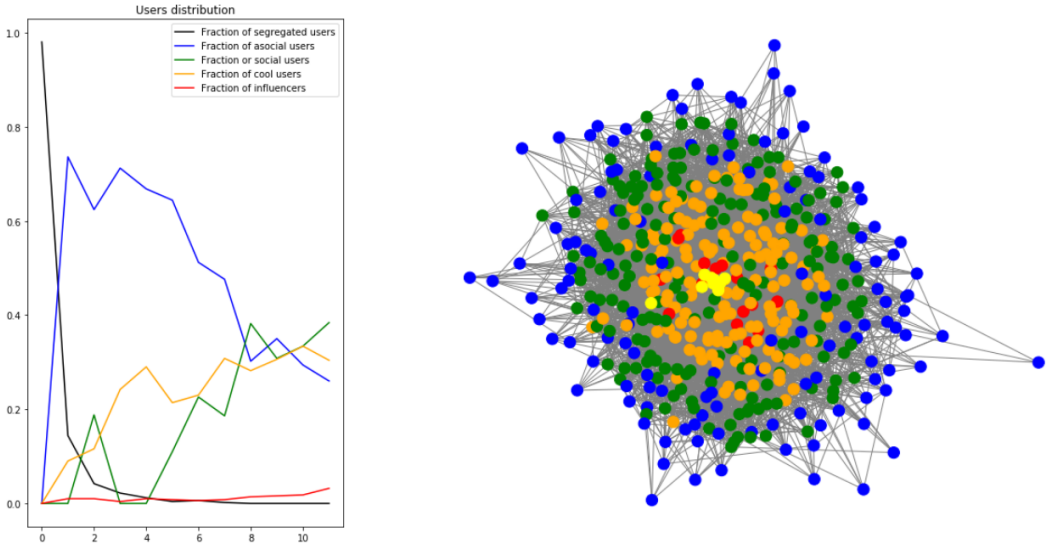


Figure 6: Graphical representation of 750 nodes generated with random parameters after one million interactions

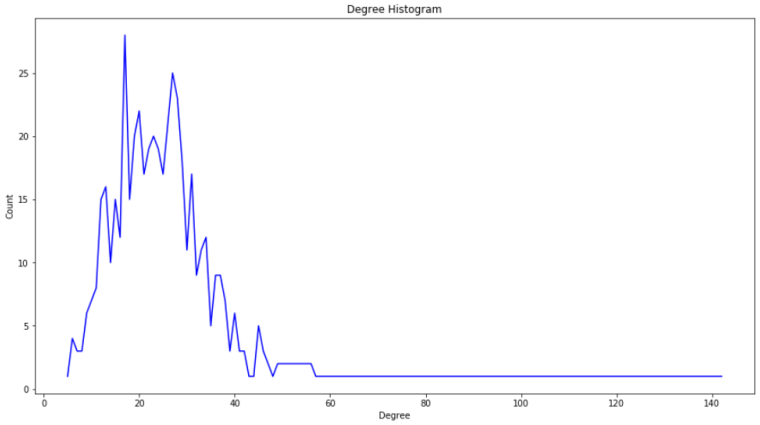
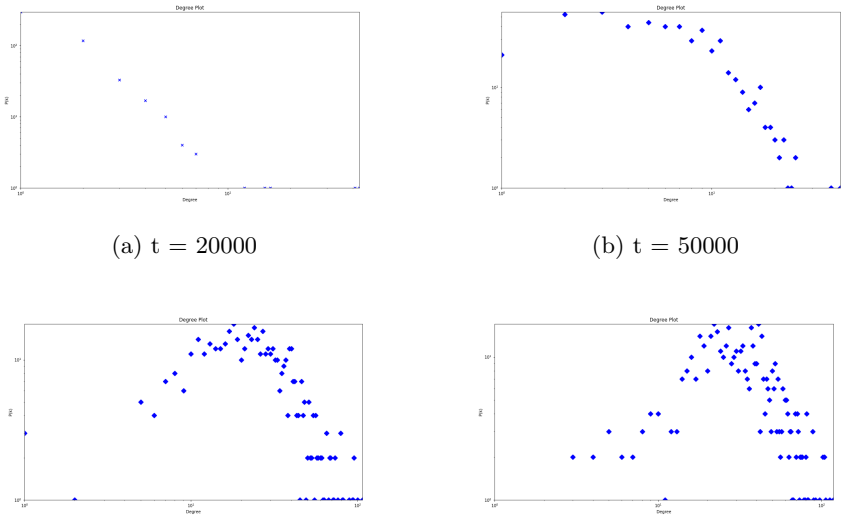


Figure 7: Degree distribution of the network with random values as parameters, the x-axes represents the degree and the y-axes is the fraction of node with that specific degree

The network is homogeneously connected, in this case the degree distribution will be more similar to a erdos renyi random graph, with a pair of nodes having a very high degree, and advancing towards the bounds, degree decreases almost linearly. What we were expacting was something similair to a scale-free network, similarly to a real network. The discrepancy of the degree distribution is mainly due to the degrees of freedom present within the model and especially by the number of nodes fixed in the network. Since the network is not evolutionary and the number of nodes remains constant, when the dynamics of the network will begin to saturate and the nodes will have already been connected with their affine, they will begin to interact randomly due to the opinion spreading. The result of the interaction between the nodes will lead to a fairly random diffusive dynamic as the nodes could remove their connections in case the opinions are divergent. A power law degree distribution can be observed in the early stages of the network emergence.



(a) t = 20000

(b) t = 50000

(c) t = 100000

(d) t = 200000

Figure 8: Degree distributions at early stages, t represents number of interactions

At very long interactions nodes will struggle to find new connections depending on feature vectors, so the interaction with their friends will inevitably lead to a decrease of their connections, disrupting the dynamics leading in the long run to a random degree distribution. The most relevant parameters for the emergence of the network are the tolerance and shyness parameters of the nodes, with a very low value of tolerance many nodes in the network will struggle to establish friendships with other nodes, this will involve a very high fraction of segregated nodes, with small marginal communities formed with a bigger populated component.

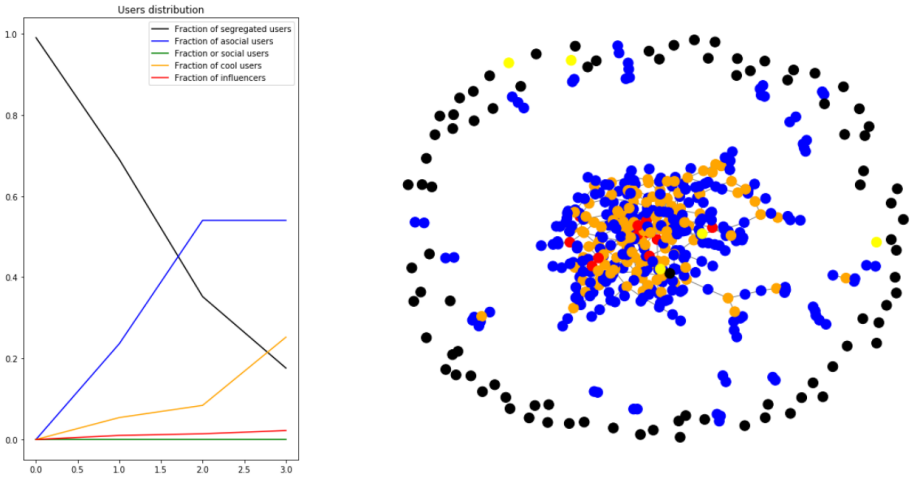


Figure 9: Graph representation of 800 nodes with a low tolerance value

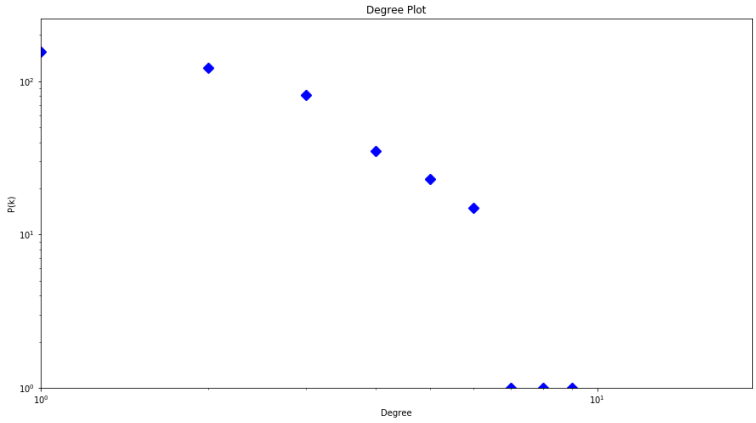


Figure 10: Degree distribution of the graph with low 800 intolerant nodes

The following representations will show the network topology regarding the polarity coefficents of every node. In addition to the feature vector, the connections between the nodes are heavily dependent on their opinions. The parameters that rules this type of interaction are mainly two, hostility and foolishness. We will find very different conformations of our network as the parameters rules the aggressiveness of the nodes during the interactions between the contents shared on the network. In this work we will present some examples of networks as these parameters change.

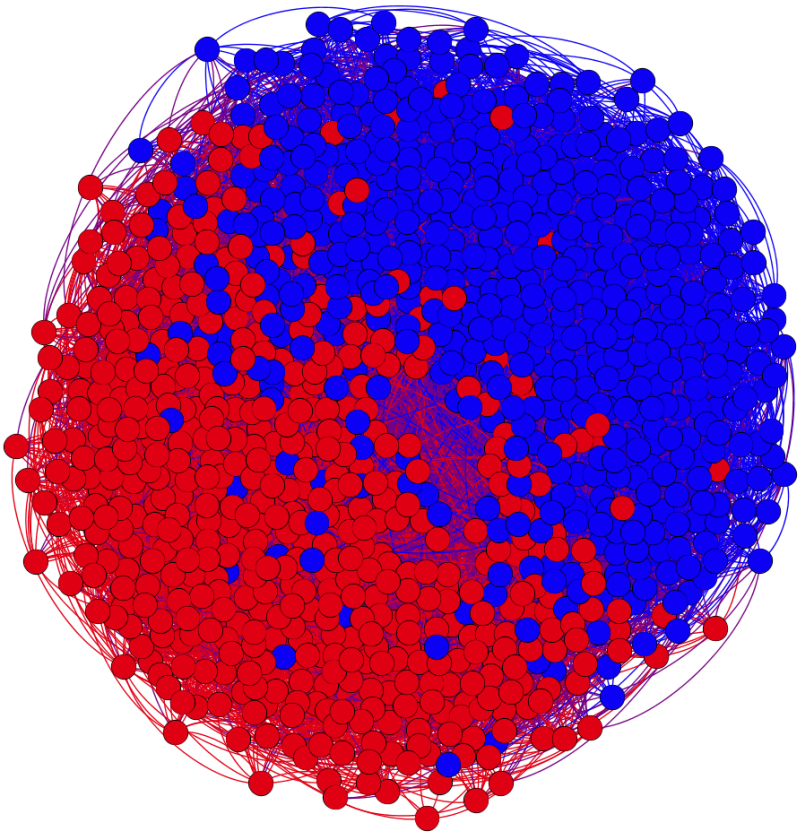


Figure 11: Graph representation by nodes polarity, blu nodes represents nodes with a negative sentiment (-1), red nodes represents positive sentiments (+1). *Superuser*

Figure 9 represents the topology of a network of 2000 "slightly aggressive" nodes within one million interaction. It is denoted how the little aggressiveness induces the network to a clear distinction between two factions with divergent opinions, the nodes will tend to have relationships only with nodes with similar polarity. A lower competitiveness of the nodes leads to a spontaneous emergence of the condition of stability of the network, in something that can be denoted as "echo chambers". Nodes tend to connect and interact only with nodes who share the same opinions. Due to the fair number of degrees of freedom in the interaction between the nodes, the network will reach a condition of absolute equilibrium at a very high number of interactions. A network made up by much more competitive nodes will lead to a different topology, it is not possible to achieve any equilibrium as the nodes during the interactions tend to disagree with each other, leading the more naive nodes to a change of opinion or more

aggressive nodes to "flame" which can also lead to an interruption of friendships.

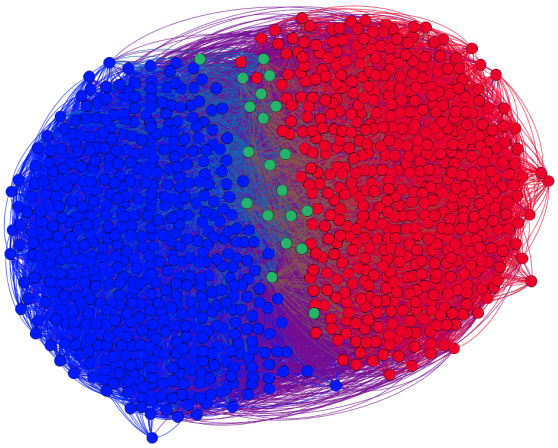


Figure 12: Graph representation by nodes polarity at equilibrium after two million interactions, blu nodes represents nodes with a negative sentiment (-1), red nodes represents positive sentiments (+1), green nodes represents *Superuser*

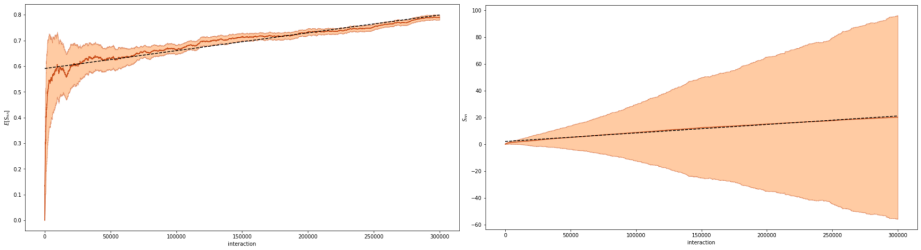


Figure 13: Degree of the nearest-neighbor polarity of whole network by interaction number (left) and average number of nearest-neighbor with same polarity of nodes by interaction

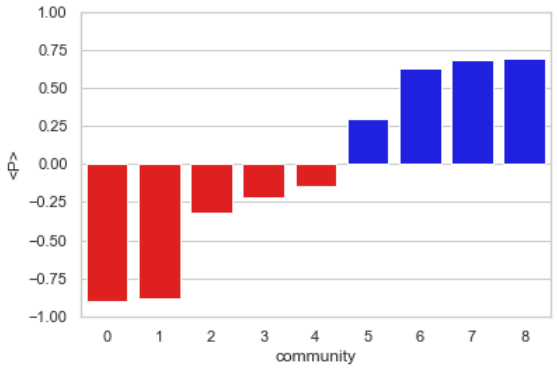


Figure 14: Average polarization of different communities identified by the Louvain algorithm.

Note that the heterogeneity of the bonds combined with the tendency of the nodes to initiate debates means that the equilibrium condition will never be reached, even at very long times, unlike a graph consisting mainly of less aggressive nodes tending not to generate any conflict, but simply tend to have interactions with nodes with the same degree of polarity. Since the nodes will initially interact to create connections only through the similarities between the feature vectors and subsequently they will begin to interact according to their degree of polarity, the heterogeneity will depend heavily on the initial parameters of tolerance, hostility and "foolishness" of the individual nodes and also by the number of Superusers within the network, which act as "content creators". By balancing these parameters, each simulation will result on a totally different diffusion of informations.

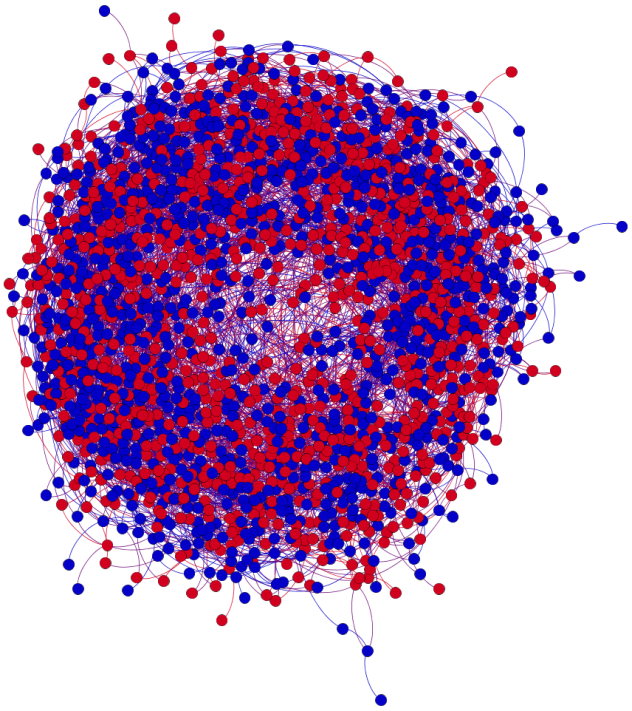


Figure 15: Graph representation by nodes polarity containing very aggressive nodes. *Superuser*

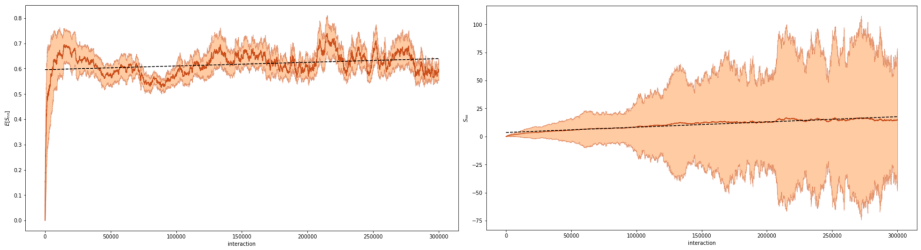


Figure 16: Degree of the nearest-neighbor polarity of whole network with very aggressive agents by interaction number (left) and average number of nearest-neighbor with same polarity of nodes by interaction

4 Conclusions

The proposed model represents a small incipit to the potential of agent-based programming for the schematic representation of a social network. This model only with only few variable parameters, which in some way represent two of the many macro peculiarity of an average social network user and a fixed number of nodes are not enough to represent the dynamical behavior of a social network. None of the four fundamental models, circular network, random, small-world, and preferential attachment, are able to faithfully reproduce the typical features of a social network. A model with multiple parameters and a larger number of data able to use greater features besides political affiliation and geographical proximity and that can simulate the temporal expansion of the number of nodes in the network could be a very promising model to simulate the dynamical behavior of a social network. Anyhow agent programming is undoubtedly a very powerful tool that can be useful in this task.