Memory effect in news spreading networks

Roberto Bertilone, Francesco Fanchin, Nicola Sella

June 5, 2018

Contents

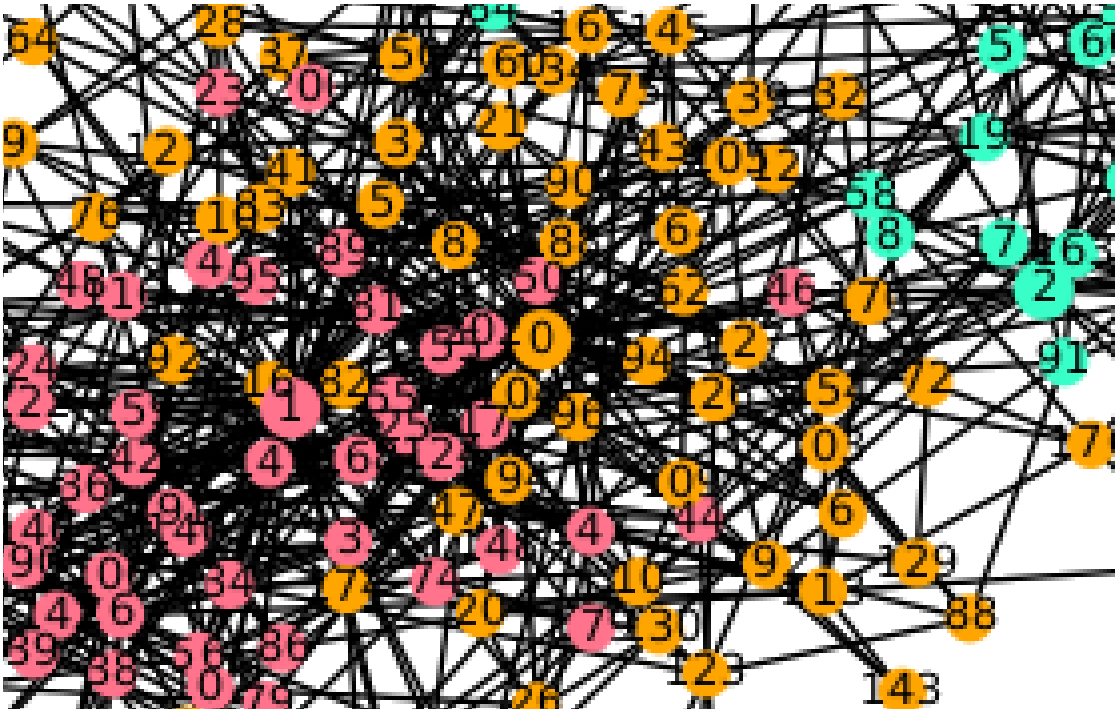


Figure 1: News Spreading Network Simulation[1]

Abstract

Our aim is to analyse the influence of memory in a news spreading dynamic. In order to do that, we have built a framework of agents connected in a network and equipped them with a set of basic functions. We wish to observe a diffusion-like process.
In this paper we expose the underlying methodology and try to explain it with a simple tutorial.

1 Introduction

For the purpose of modeling the interactions between users in the context of news spreading, it is convenient to talk about autonomous agents linked by friendly ties whose overall view constitutes the network. Network population is made of two breeds of agents: sources and users. These two breeds will interact in an autonomous approach during program execution. The network is initially a random connection of agents and modifies its own topology following a set of microscopic agents' actions.
We believe that news' content and agents social impact give rise to natural news diffusion in a social-like network. We hope to observe a spontaneous growth of scale-free regime

[1]Simulation run with: sources: 3, users: 200, cycles:800, average degree: 5, seed: 123456789

starting from dynamical micro-interactions.
We also hope to reveal a natural segregation behaviour that subjugate a great deal of real social networks. In addition to these items we wonder if there can be correlation between news spreading and agents' memory lenght. In practice, we have worked with the Swarm-like protocol in python 3 named SLAPP3.[2]

2 Overview

We have built our model focusing on two methodologies: agent based and network frameworks. We have blended these two frames considering a network of agents.

2.1 Context

Let us focus on the dynamical process of news spreading in a social[3] network.
News diffusion is generally studied in a stochastic context, ruled by a set of stochastic differential equations.[6] There is an apparent similarity with epidemiological processes. However, while epidemic diffusion becomes a viral process, when a threshold is exceeded, news spreading process seems to be threshold-less. The epidemic model of information diffusion is usually a compartmental model in which agents coexist in the world in different stages.[4] Most of initial users stay in the compartment of *ignorants* whereas a minority of them stay in the *spreader* compartment. There is another compartment, the *stiflers*: users influenced by news who do not spread anymore (equivalent to *recovery* in the SIR[4]model).
This *Spreader-Ignorant-stifler* model (SIs) can be sketched by a set of transitions between compartments; one of the transition is spontaneous while the others are induced by contact:

- $I \longrightarrow S$
- $S + I \longrightarrow S + S$
- $S + S \longrightarrow s + S$
- $s + S \longrightarrow s + s$

S is the *Spreader*, I is the *Ignorant* and s the *stifler*. The first process corresponds to the spontaneous transition from the ignorant to spreader compartment; the second one corresponds to the contamination of an ignorant by contact with a spreader; the third and fourth interactions reproduce transitions by contact to the stifler compartment. Dynamic evolution is governed by series of transitions from a compartment to another one. Just as all spreaders reach the stifler compartment, ignorants remain so and the epidemic stops.

[2]For reference and download from: https://github.com/terna/SLAPP
[3]The word *social* can be thought in a general context.
[4]Acronym of Susceptible-Infected-Recovery, most famous model of epidemic spreading.

This approach is applicable to a network of users to predict the reproductive number[5] which enable us to estimate the future qualitative behaviour of spreading: if this number is above some epidemic threshold then virality of diffusion is guaranteed.
This can be a reliable description for a single news viral diffusion, but when we deal with multiple news, the analysis by a set of several differential equations would result more difficult.
A compartmental approach to study the phenomenon of information diffusion has been widely examined in the last years, with very different variants of naive models.[7][16] There are also several papers underlining interesting results in social science: for example social influence, infection, segregation,[9] homophily[1] or fake news diffusion;[16][11] the effects of fact-checking or bot-agent insertion[2] in a network are studied too. In news spreading literature, only few models are built on an agent architecture.[10][15][13][5][14]

2.2 Why Agents?

Agents are a very useful paradigm to model social interactions.[3] They can operate in their environment, take decisions and interact with each other. There is no communication protocol between them but they communicate and share news with "friends".
The environment is not deterministic. Every action can produce different effects with a different probability, but the simulation must be reproducible. Actions between agents are non-deterministic: they don't have complete control of another agent and have limited senses and sensors.
The required characteristics of each agent are:[17]

rationality: agents can take choices depending on their own belief and their surrounding environment;

reactivity: agents can check the world clock and news spreading nearby during the dynamic;

proactiveness: agents can express their own will, taking autonomous initiatives;

social ability: they know how to send and receive news, to determine sympathy with neighbours;

no *mobility* is required.

Each agent can receive information from the world or from another agent; it[6] can also interact with the world or with an agent, in order to meet its own "character" (a.k.a. state of mind). They can modify their surrounding environment by means of the adding and removal of edges in social network. The only accessible variable for each agent is the clock number; they can also access some neighbours' information.
Each agent makes local fair decisions: global behaviours may emerge. When active,

[5]The reproductive number R_0 is defined by characteristic parameters that affect spreading like average degree (in first approximation) and diffusion rate.

[6]There is no sexual division among agents

each agent can control the environment and reacts to the changes. During the inactive state the environment can change so an agent's previous buffered actions may not be performed. For this reason the agent can act unpredictably and somehow irrational.

2.3 Network of Agents

Our network is made of agents eventually connected by weighted and undirected edges. The network is composed by of a fixed number of sources and users.
Users are tied with a attachment probability computed from the desired average degree and inserted by external input. In this way we obtain a random network of users, with exponential trend for the degree distribution in the termodynamical limit. We can see intial degree distribution in linear scale (figure 2).

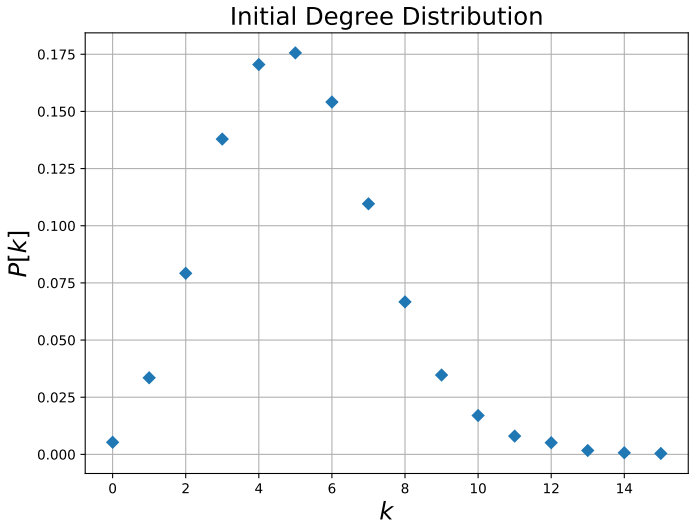


Figure 2: Degree Distribution, seed 17, 3 sources, 10000 users, 5 initial average degree, 1 time step

Links are the only possibility to establish a relation between the agents; a random value of edge's weight represents a previous bias in friendship.[7] The result of such a mechanism of graph generation doesn't actually return a real network, because social real networks possess the property of scale-free,[8] instead, in random graphs, the majority of node-degree is near the average-degree. Furthermore, in scale-free network, variance is very large and guarantees the existence of hubs in the network.[12] Despite the above-

[7]This particular choice for our links underlines bilaterality and intensity of communication between agents. Taking as example an exchange of information between two people or between a person and a newspaper, weight represents feeling, previous chemistry;

[8]The scale-free property is mainly described by power law degree distribution, with a cut-off for high degrees (i.e. hubs).

mentioned theory, we start with a random network hoping to observe a natural evolution in this direction.

Sources are a news reservoir: users can pick one or more news from them and eventually spread. Sources' degree is higher than users' one because we assume that newspapers have more links than a common user. Sources contain a fixed number of news and eventually create fresh news run-time.
Each single news contains a unique string which identifies it, a vector of topics, the initial source's ID, creation date, its own relevance (a sort of "impact factor").

Users inculde the "state of mind", a vector representing their own personality; sources, like users, have a "personality" too (i.e. another vector) which reflects news' content. Users can remember a limited number of news, according to the affinity between incoming news and their state of mind. Furthermore, affinity also regulates diffusion among users. State of mind is not fixed since is affected by news spreading; edges' weights can be modified by users' state of mind. Time is externally imposed and, according to it, users can be active or inactive: they can interact with external world when active.

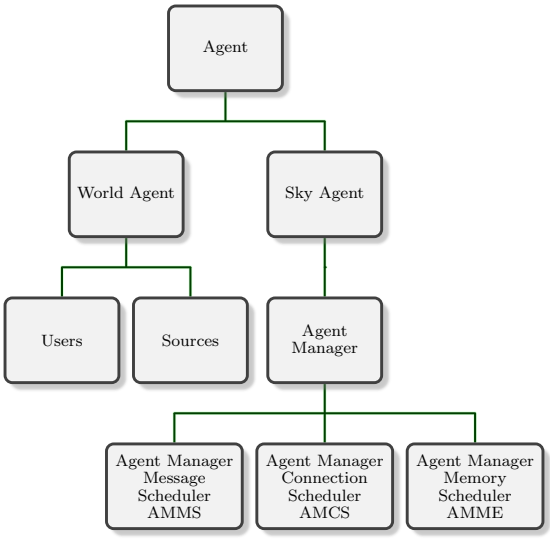


Figure 3: Hierarchy of classes

3 Our Model

We use the SLAPP3 platform for agent simulation: this environment provides agent based protocol in Python 3. We implemented a hierarchy of agents to manage the different abilities of each breed.
The class *Agent* is the common and oldest ancestor: this is required by SLAPP3.
Then we have two main classes defining the main branches, one for the 'real' agents, *WorldAgent*, and another for the abstract ones, *SkyAgent*. *WorldAgent* can represent living creatures or tangible objects; *SkyAgent* is an external agent, living outside the world.
Only five breeds of all the implemented classes are produced during the execution of the program. This structure allows easy changes and possible future implementations of the code.

3.1 WorldAgent

Two leafs of the tree sprout from this branch: the classes *User* and *Source*. The common variables passed by *WorldAgent* are the vector of mind state and the database of news, used both for memory and storage. The common members are trivial. Specifically, the state of mind is a normalized vector of a specified dimension `dim` defined inside *commonVar.py*.

3.1.1 Source

Sources have a peaked mind state initialized during construction. The initialization is binary with a number of non-zero values from one to three: after that, noise is added and the vector is normalized. Each *Source* has a method, *generateNews*, that can produce

news described with a vector of topics "near" the state of the source. We use near, talking of states, like we talk of geometrical vectors: we will measure the mind distance with the scalar product of these vectors.

3.1.2 User

The main class of the program is *User*. This object contains all the functions to act independently in the world. We will provide a description of the most important ones. As mentioned before, a user can compute distance, with the namesake function, among mind states and between a mind state and a topic inside news. He can get an opinion on what he sees. He has a bunch of function to detect who are his neighbors, if they are sources or users, and if there are only sources around him. A user can also decide to become active or inactive using internal rules. He can create or remove an edge between another agent. Other important users' actions are the way they spread: *activeDiffusion* and *passiveDiffusion* are the two functions used to spread news. We will talk about them later. Finally, an agent can choose how and who manages edges, using *createEdge* and *deleteEdge*.

3.2 SkyAgent and AgentManager

SkyAgent has an only child: *AgentManager*: he is responsible of all the logs. Other skyagents can be implemented but they are not needed now.

3.2.1 The Schedulers

There are three leafs out of the *AgentManager's* branch: *MessageScheduler*, *ConnectionScheduler* and *MemoryScheduler*. We will refer to them with their respective abbreviations: *AMMS*, *AMCS* and *AMME*. The purpose of the logs is to provide a post simulation tool to perform a complete statistical analysis of the model.

AMMS This scheduler registers all the spreading during the execution. It focuses on the news and registers their creation and diffusion, also marking the passive and the active diffusion.

AMCS We can take note of all operations correlated to edges: creation, destruction and weight change will be registerd in a log file.

AMME This is unlike the other logs. It is not called from a user but from the observer.[9] For every cycle of the program this scheduler will print the database of each agent and the activation state of anyone.

3.3 Modeling Variables

It is not easy to infer a good human mind model. One has to make a lot of assumptions. For instance, what is important and what is not. As we said before, the agents live in

[9] TODO specificare observer nel protocollo swarm

a network; the connections are the possibility to communicate and the edge's weight represents the previous trust.

3.3.1 Mind State

One way to model mind is to divide it into topics: the number is arbitrary. The mind now is a vector: each component is a topic. We have choosen non-negative values as if anyone could be interested in some topic or not. One can have or not an opinion: there is no distinction between having a good or a bad point of view. The initialization of these mind states is binary: the vector starts with only zeros or ones as said for the sources before. Then rumor is added and the vector is normalized. We chose to have only one peak each user.

3.3.2 News

The news is a piece of information that an agent can like or dislike. For this reason it must be comparable with the mind state. We have built news as a vector, like the mind state: its components, geometrically speaking, are near the source's one. Other important variables for the news are the relevance, which measures the impact of the news in the world, the date of creation and the source who made it.

3.4 Execution

Dynamic is regulated by the scheduled structure of SLAPP3. An observer defines the world time while the model itself provides for a finer structure of actions. Furthermore, several actions are scripted and interpreted by the scheduler.

3.4.1 Observer

The observer actions used, included in the file `observerActions.txt`, are `modelStep`, `ask_all`, `ask_one`, `visualizeNet` and `clock`.

`modelStep` is called with its default implementation

`ask_all` calls the *AMME*

`ask_one` calls the finalization of the program. It is used in the last line of the file: the line number has to correspond to the final clock cycle in order to make this function work. To end the execution, log files have to be written, calling `writeLog` which is implemented in each of the *AgentManager's* sons. The `.gml` file of the graph is saved as well as the logs in a folder created ad hoc: `logs`. If the execution interrupts or, for some reason, `ask_one` is not called properly, the files will be saved as well in a temporary folder, `temp`, and the logs will be saved up to the latest automatic save (every 1000 lines of log).

`visualizeNet` calls the net visualization. The function called is `drawGraph` which is contained in the file `graph.py`. It is also possible to draw the net, to save it in a file or both.

`clock` increments the world time.

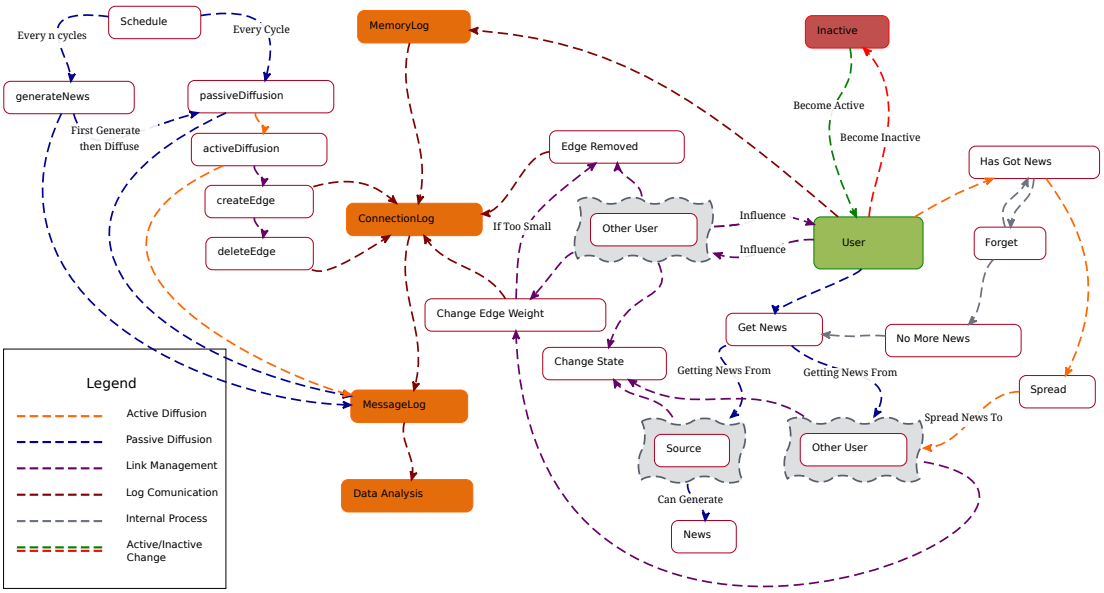


Figure 4: Mind map of the model

3.4.2 Model and Schedule

The file `modelActions.txt` calls the script with `read_script` at every cycle. The `reset` and `move` functions are not implemented but still called as placeholders. The file `schedule.xls` makes three groups of calls:

timestep 1 during the first cycle the program generates news contained in the sources;

timesteps 2 - 10000 after news generation and for all the execution, users perform passive and active diffusion;

timesteps 10 - 10000 after some time of diffusion the agents start to modify the net structure, adding and removing edges with a probability of 0.03.

3.5 Overview on simulation: technical difficulties

Let us focus on a complete simulation cycle, dealing with some practical details, while the overview of the main process is given to be clear to the reader. The figure 4 explains

the main program workflow during a generic simulation. From left to right: the schedule runs the main blocks, then the logs start the acquisition of data, finally we see the actions of a generic agent during one time step.

3.5.1 The problem of 'log calling' and 'memory unload'

The breed order creation is log first, then users and finally sources. We will discuss how and where to call a log during the execution of the program. As we previously said, logs are agents themselves, not external functions or tools. The problem is how to call a log at run time; this is solved only when a single instance of every *AgentManager's* son is created. Each scheduler, automatically builds himself, during initialization, inside the `commonVar.py` file. It is obvious that this trick will not work for more than one log for breed. The agent nature of the log is not a bad thing at all because all the logs are able to manage the memory autonomously: they fill the memory of the machine up to a limit value of lines and then autosave, appending all the work done in a temporary file. This allows a limited number of accesses to the disk and a reasonable amount of memory used during the simulation.

3.5.2 How to create a graph of agents: 'further implementation issue'

The class *WorldAgent* creates the graph during the agent initializiation. The trick used is clever but can produce some problems in further implementations. The first agent creates the graph if he is the first called and if the timestep is 1; all the agents add themselves into the graph durig their initialization and the last one creates the edges randomly. This is a problem if we want to implement a feature for adding nodes. What we want to avoid is the re-initialization of edges at every new *WorldAgent* initialization. The simulation works properly if the number of the agents does not change.

3.5.3 Determinism and Random

Agents' actions are deterministic and random. Let us focus on the action `createEdge`. An agent of the breed user can decide to add an edge choosing a node from the other agents in the net. He chooses one user randomly and one between his second nearest neighbors, computing the distance between his mind state and the possible future linked agent's mind state. To add a bit of randomness the agent adds all the second nearest neighbors to a list, sorted by distance and counted with their multiplicity. Then he chooses randomly from the first ten elements of the list. Thus we can model deterministic action rules maintaining the behavior unpredictable. Other examples of determinism/random can be found in all the spread and diffusion functions.

3.6 What can this model do?

This model allows to observe different behaviors of agent communities' interactions. You can notice the news spreading and the emerging phenomenom of rumor spreading correlated to it. Besides, the actions of the agent changing state, while spreading news, bring

out agents' segregation in the network. All of these effects can be studied also altering agents' memory. Memory dependency is a new variable introduced as a fundamental part of the agent's structure.

3.6.1 News Spreading

This model can be used to simulate the spread of one or more news generated by sources at any cycle of time. The news can also be added or regenerated every time. It can run for an arbitrary long mind state of agents, for an arbitrary memory length. It is written using dictionaries: it provides a very short computational time. Other mathematical operations are performed including numpy. Furthermore it can run very long simulations choosing not to save graphs and produce logs only for post simulation analysis.

3.6.2 Segregation

The strength of this model is its abstraction level: it simulates the spread of a positive normalized vector. We used it to analyze a group of people sending messages, emails or talking about news. It is possible to perform a mind state spreading. We modeled the spread of a mind state when we used the source as a spreader of a news, which is near his mind state. If the assumptions on the vector are good, segregation will emerge from this model as a common behavior of agent based and network systems. The segregation, due to different languages, can be modeled as well: let us think of the mind state as a vector describing the way of speaking. Each language has a different accent and a different tone: all these things can be represented as a positive vector.

4 Tutorial

4.1 Before the simulation: what to do

Let us see in practice what we have previously learnt in a simple tutorial: first of all, we need the whole program, available at https://github.com/BFSteam/memory.
Once downloaded, it is convenient to put `path_to/memory/src` inside `path_to_SLAPP3/project.txt` as described in *SLAPP_Reference_Handbook.pdf pag20*. Inside SLAPP3 we can run the program `runShell.py` from terminal or from jupyter notebook using `iRunShell.ipynb`.
The program asks which project we want to run: if we set up the path correctly we can confirm *memory* path and project. Afterwards we have to set all the necessary input variables to start the simulation:

`Random number seed:` insert the seed to make the simulation reproducible.

`Number of sources:` insert the number of sources inside the network.

`Number of users:` insert the number of users inside the network.

`Average degree for users:` insert the value of the average degree for users only.

`Number of cycles:` insert the maximum number time can reach.

In order to simplify our first approach, default values are provided: answering enter at each line will make the program run, and that's it!

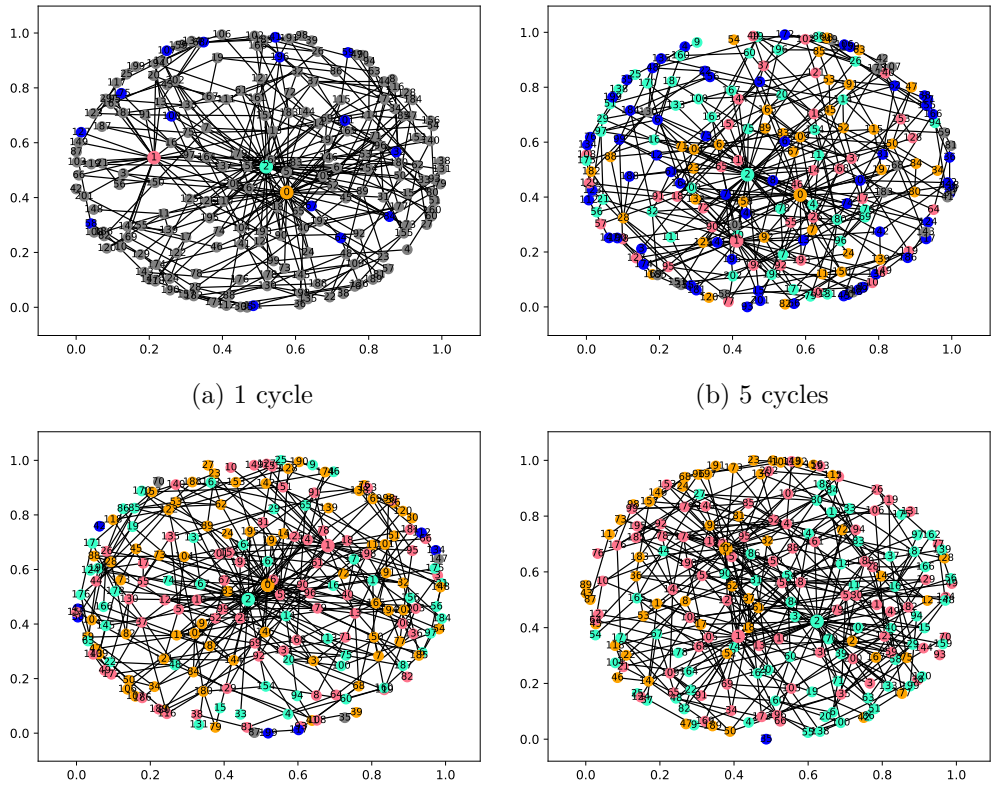
4.2 During the simulation: where to put the accent

Simulation is running and our network is evolving. A window will appear with the drawing network[10] and flowing time is displayed in terminal.
Nodes of the network are painted with different colors. If they don't spread: grey for inactive state and blue for active one; if they spread can be orange, pink or cyan depending on the spreaded news. In detail, every source initially generates a single news, tracked by its color. The source's size are bigger than users'one; sources are labeled with 0, 1 and 2. Numbering goes on with users.
We show network dynamic in the pictures below:

4.3 After the simulation: what to notice

We show graphs of a previous run. Used parameters are: seed 17, 3 sources, 200 users, initial average degree of 3, and 500 time steps.
In figure 5a we see the network during initialization. Diffusion has not started yet and users are mostly inactive: sources have created one news and the few active users are unaware.

[10] Graphs drawn using networkx and matplotlib



(a) 1 cycle

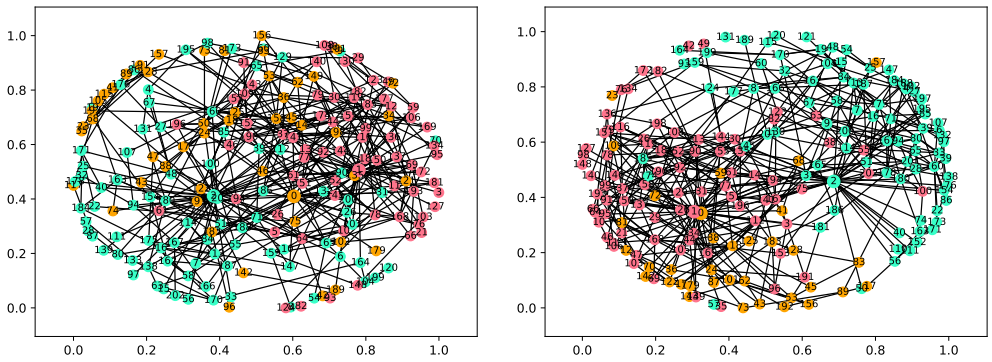
(b) 5 cycles

(c) 10 cycles

(d) 50 cycles

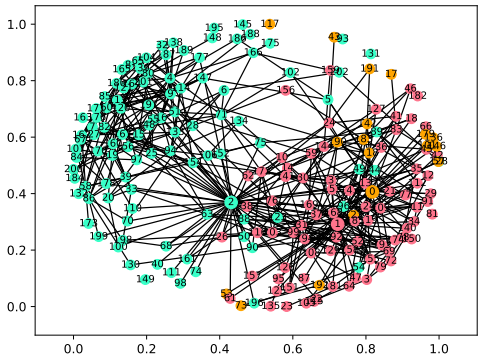
Figure 5: Plot at different initial time steps for a simulation with 3 sources, 200 users, initial average degree of users 3 and 500 time steps. Random seed initialized to 17.

After five time steps, figure 5b, some users have changed their state to active and have started to spread. However, a few of uninformed users still remain. At tenth iteration, in figure 5c, almost every user has a news inside and the whole network might be homogeneously mixed. We have also the first edge creation and removal: consequently, the network's topology starts changing.

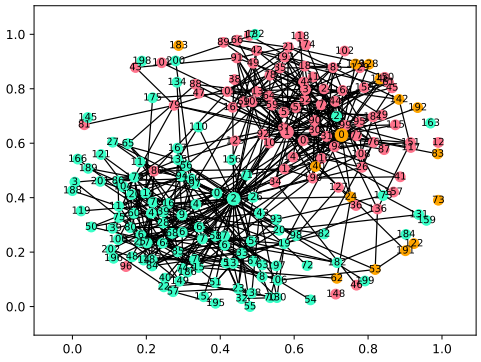


(a) 100 cycles

(b) 200 cycles



(c) 400 cycles



(d) 500 cycles

Figure 6: Plot at different final time steps for a simulation with 3 sources, 200 users, initial average degree of users 3 and 500 time steps. Random seed initialized to 17.

Now we jump to the fiftieth iteration, figure 5d, to see that all the users except one are involved in the dynamic. There might be a clue of segregation. After fifty steps we detect the first clear symptom of segregation. Moving on of a hundred in a hundred time steps, we see how the orange news tends to vanish and a strong segregation arise in two main communities due to past influence between users.

4.4 Further implementations

Some possible future developments are:

adding and removing nodes during execution can yield big changes in the process;

different algorithm of network generation e.g. Barabasi-Albert algorithm or Watts-Strogatz algorithm;

analyze the activation time starting from microscopic behaviors it is possible to reproduce macroscopic phenomena such as bursty patterns.[8]

4.5 Conclusion

Despite the vast and growing literature on rumor spreading based on SIR model we have built agents' microscopic actions establishing on common human behaviour and social rules. Furthermore the environment is a random network and the edges change dynamically. Agents alter their mind state interacting with neighbors. The diffusion process takes into account from one to several news. Sources are agent themselves: they are part of the network and are not external.
The originality of our model lies in the brand new variable memory which strongly affects agent's actions and decisions. We expect that memory length will lead to measurable macroscopic dynamical effects: this is what we would like to investigate. This project will advance our understanding on this hypotesis: several simulations, performed with different parameters, could lead us to a proof which decides the conjecture.
We observed segregation in almost all examples and some parameters might be more relevant than others. Two open questons hold: which are the key parameters and, if we found them out, which their values and their ranges would be. This is pure speculation on our part but if it were right we could state which parameters are crucial. A second issue we have previously discussed is the emerging scale free dynamic: the authenticity of our guess will be validated through statistical observations.

References

[1] Luca Maria Aiello et al. "Friendship prediction and homophily in social media". In: *ACM Transactions on the Web (TWEB)* 6.2 (2012), p. 9.

[2] Luca Maria Aiello et al. "People are Strange when you're a Stranger: Impact and Influence of Bots on Social Networks". In: *Links* 697.483,151 (2012), pp. 1–566.

[3] Robert Axtell. "Why agents?: on the varied motivations for agent computing in the social sciences". In: (2000).

[4] Alain Barrat, Marc Barthelemy, and Alessandro Vespignani. *Dynamical processes on complex networks*. Cambridge university press, 2008.

[5] Maira A de C Gatti et al. "A simulation-based approach to analyze the information diffusion in microblogging online social network". In: *Simulation Conference (WSC), 2013 Winter*. IEEE. 2013, pp. 1685–1696.

[6] Wei Chen, Laks VS Lakshmanan, and Carlos Castillo. "Information and influence propagation in social networks". In: *Synthesis Lectures on Data Management* 5.4 (2013), pp. 1–177.

[7] Nick Fedewa, Emily Krause, and Alexandra Sisson. "Spread of A Rumor". In: *Society for Industrial and Applied Mathematics. Central Michigan University* 25 (2013).

[8] K.-I. Goh and A.-L. Barabási. "Burstiness and memory in complex systems". In: *EPL (Europhysics Letters)* 81.4 (2008), p. 48002. ISSN: 0295-5075. DOI: 10.1209/0295-5075/81/48002. URL: http://stacks.iop.org/0295-5075/81/i=4/a=48002 (visited on 06/07/2017).

[9] Adam Douglas Henry, Paweł Prałat, and Cun-Quan Zhang. "Emergence of segregation in evolving social networks". In: *Proceedings of the National Academy of Sciences* 108.21 (2011), pp. 8605–8610.

[10] Dechun Liu and Xi Chen. "Rumor Propagation in Online Social Networks Like Twitter–A Simulation Study". In: *Multimedia Information Networking and Security (MINES), 2011 Third International Conference on*. IEEE. 2011, pp. 278–282.

[11] Maziar Nekovee et al. "Theory of rumour spreading in complex social networks". In: *Physica A: Statistical Mechanics and its Applications* 374.1 (2007), pp. 457–470.

[12] M. Newman. "Introduction". In: *Camden Third Series* 94 (1963), pp. vii–xiv.

[13] Eunsoo Seo, Prasant Mohapatra, and Tarek Abdelzaher. "Identifying rumors and their sources in social networks". In: *SPIE defense, security, and sensing*. International Society for Optics and Photonics. 2012, pp. 83891I–83891I.

[14] Emilio Serrano and Carlos A Iglesias. "Validating viral marketing strategies in Twitter via agent-based social simulation". In: *Expert Systems with Applications* 50 (2016), pp. 140–150.

[15] Emilio Serrano, Carlos Ángel Iglesias, and Mercedes Garijo. "A Novel agent-based rumor spreading model in Twitter". In: *Proceedings of the 24th International Conference on World Wide Web*. ACM. 2015, pp. 811–814.

[16] Marcella Tambuscio et al. "Network segregation in a model of misinformation and fact checking". In: *arXiv preprint arXiv:1610.04170* (2016).

[17] Michael Wooldridge and Nicholas R Jennings. "Intelligent agents: Theory and practice". In: *The knowledge engineering review* 10.02 (1995), pp. 115–152.