ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

Simulation of Information Spreading in a Facebook Network

Urs Lustenberger , Nino Wili , Patrick Zöchbauer

Zurich
December 2013

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Urs Lustenberger Nino Wili Patric Zöchbauer

Contents

1 Abstract

2 Individual contributions

3 Introduction and Motivations

Everyone is on facebook
Commercials are personalized, the flow of info is interesting for companies
Are there "more important" persons in typical facebook network?

important questions (does the inhomogeneity of the real network influence the "total" evolution? are there "influentials"?)

4 Description of the Model

The process of information spreading has many similarities with epidemiology, so the well known SIR-model was adapted[1]. The "susceptibles", are not aware of the information and are called "ignorants" within this work. "Infected" individuals know about the information and are willing to share it with other people, in other words they spread it and are therefore called "spreaders". The adaptation of the epidemiological term "recovered" is not that straightforward because it is not entirely clear, what that means in this context but they can be looked at persons that are aware of the information but don't want to tell it others. They are called "stiflers".

In the SIR-model as well as in the Agent-based model, the following set of "reaction equations" was used(I: Ignorant, S: Spreader, R: Stifler):

$$\begin{cases} I + S \xrightarrow{\lambda} 2S & (1) \\ S + R \xrightarrow{\alpha} 2R & (2) \\ S + S \xrightarrow{\alpha} S + R & (3) \end{cases}$$

The main difference to the standard SIR-model is that spreaders don't become stiflers spontaneously but this this change is only induced by meeting an other spreader or stifler. Also it can be shown that for $\frac{\lambda}{\alpha} > 0$, i.e. for any positive rate, the number of stiflers is non-zero for large times. That means that there is no epidemic threshold as in the standard SIR-model.

Flow Chart of overall steps

Figure 1: Steps performed in each timestep.

4.1 Homogeneous SIR model

In the mathematical treatment of the model in a homogeneous system, the set of differential equations is expressed in terms of the densities $i(t) = I(t)/N$, $s(t) = S(t)/N$ and $r(t) = R(t)/N$.

$$\begin{cases} \dfrac{\mathrm{d}i(t)}{\mathrm{d}t} = -\lambda \cdot s(t)i(t) & (4) \\ \dfrac{\mathrm{d}s(t)}{\mathrm{d}t} = \lambda \cdot s(t)i(t) - \alpha \cdot s(t)[s(t) + r(t)] & (5) \\ \dfrac{\mathrm{d}r(t)}{\mathrm{d}t} = \alpha \cdot s(t)[s(t) + r(t)] & (6) \end{cases}$$

4.2 Agent-based model

In the inhomogeneous, agent-based model, the individuals are connected in a certain manner (facebook friends in this particular work). Only connected agents are able to meet. If a meeting occurs, transitions are induced at a certain probability depending on the relation between the agents (details are discussed later). Additional to the different degree of the agents, they also have a different activity, i.e. different probability to meet somebody. In total, the number of meetings in a certain time is proportional to the product of the degree (number of friends) and the activity. This has to be verified in the implementation of the model.

To convert the reaction equations into an agent-based model, time was discretized and in each time step a series of two steps is performed shown in Figure 1. First, the agents randomly meet another agent they know or nobody. Only two-agent meetings are possible. In the second step, the status of the agents change corresponding to the situation. There are six possible combinations of ignorants, spreaders and stiflers. Three of them correspond to the "reactions" 1-3, the other ones (I+I, I+R and R+R) have no other influence than "occupying" the agents. The probability λ was chosen to be dependent on the number of common friends. For simplicity, α was a constant.

5 Implementation

5.1 How did wezget our data?

To get the data for our simulation, we used the so-called Facebook Graph API. (reference) What does Graph API mean? A Facebook Grpah API is a programming tool designed to support better access to conventions on the Facebook social media platform.[1] It allows you to extract information that is published on facebook, which is exactly what we needed.

Unfortunately, there is no implementation for Matlab, so we programmed the data-fechting in python.

5.2 Coding

5.2.1 Get access

For the coding in python we used the facebook sdk for python[2].

First, one needs to get access to the social graph. Therefore, you need to enter an access token[3].

```
1 token = raw_input('Enter your Access Token: ')
2 anonym = input('Do you want to anonymize your Data? Yes (1) or No (0)? ')
3 graph = facebook.GraphAPI(token)
```

5.2.2 Get graph

Having enterd a valid token, we can now access all information this token provide us. For simplicty, we'll only discuss the implementation of the non-anonymized case in details. [For more details on the anonymization, look at subsection anonymization]

```
1 profile = graph.get_object("me")
2 friends = graph.get_connections("me", "friends")
```

In our case, we're interested in data about our own profile, as well as all available information we get from our friends. The object friends is nothing else but an array that contains all id's of our friends, and for each id, there is another array data, that contains all information about this specific friend.

[1] http://www.techopedia.com/definition/28984/facebook-graph-api

[2] https://github.com/pythonforfacebook/facebook-sdk

[3] https://developers.facebook.com/docs/facebook-login/access-tokens/

```
1 friend_list = [friend['id'] for friend in friends['data']]
```

With this simple for-loop, one can now get a list of all his friends. Since we're not primarly interesseted in our friends, but the connection in between them, we have to access at least the mutualfriends.

```
1 mutualfriends = graph.get_connections(tfriend, "mutualfriends")
```

The object mutualfriends provides us a list with all mutualfriends of "me" and friend "tfriend" (which is nothing else but one of the id's we have in the saved in "friend_list"

Repeat this proceedure for all friends, and we'll get exatly what we're looking for - a graph of all connection of our friends in between of them.

5.2.3 Get activity

5.3 Anonymization

5.4 The Network

how did we get the network?

how did we get the coordinates with gephi?

5.5 Homogeneous SIR-model

Urs

5.6 Agent-based model

5.6.1 Initial condition

At the beginning of each simulation, all agents are ignorant but one, which is a spreader. This agent was determined pseudo-randomly.

5.6.2 Determine the meetings

(See `talkstep.m` for details.)

In order to determine who meets who, the program goes through the vector of agents (1:N) randomly (line 14-16). With a probability corresponding to the activity of that agent, he may meet somebody (line 19). The person he meets is determined randomly and must be one he knows and one which is not already meeting another

agent in this time step (line 30). In order to be able to implement that agents with more friends meet more people and to keep the simple data structure first a random person is chosen out of all the persons in the network and after that, the program checks whether they know. Like that, lot of "finding attempts" land on pairs which are not connected, that's why more than one attempt is performed each round (line 22-39). `attempt` is basically just a scaling factor. Tests show that the number of meetings is still proportional to the product `activity * number of friends`.

5.7 Status changes in meetings

6 Simulation Results and Discussion

most important question!

7 Summary and Outlook

blub

8 Appendix

Code

../../code/talkstep.m

```
1  %script thats let people meet, eventually meet and talk, updates status
2
3  %%
4  %Determine who meets who
5
6
7  %vector which stores if someone's already in a meeting with who
8  %0: free, k: already meeting person k, i:not meeting anybody
9  meeting=zeros(1,N);
10
11 %random vector, so it doesnt start at node 1 always
12 choose=randperm(N);
13
14 for k=1:N
15
16     i=choose(k);
17
18     if(meeting(i)==0)%check if i already meets somebody
19         if(rand>(person(i).activity))
```

```
20              meeting(i)=i; %doesnt meet anybody
21          else
22              who=0; %who: meeting partner of i
23              attempt=1; %dont check forever, maybe no partner available
24              while(who==0);
25                  who=round(rand*N+0.5); %random partner
26
27                  attempt=attempt+1;
28
29                  %check if who is not meeting, and who and i know each other
30                  if(meeting(who)==0 && connect(i,who))
31                      meeting(i)=who;
32                      meeting(who)=i;
33
34                      nummeetings(i)=nummeetings(i)+1;
35                      nummeetings(who)=nummeetings(who)+1;
36                  else
37                      if(attempt<50) %change this!
38                          who=0;
39                      else who=1; %just not 0
40                      end
41                  end
42              end
43          end
44      end
45
46      %some can be 0, not cool for further programming
47      if(meeting(i)==0)
48          meeting(i)=i;
49      end
50  end
51
52
53
54
55  %% change status for each meeting
56
57  check=zeros(1,N); %vector to check if status was updated, otherwise
58  %multiple updates in one round are possible
59
60
61
62  for i=1:N
63
64      p1=i;                       %person1
65      p2=meeting(i);              %person2
66
67
68      %check if status wasnt updated yet and if i is actually meeting sombody
69      if (check(p1)==0 && check(p2)==0 && p1~=p2)
```

```
70
71          %if both are ignorant or both are stiflers nothing happens...
72
73          %if one is ignorant and the other is a spreader, both become
74          %spreaders
75          if((status(p1)+status(p2))==1)
76              %make p1 the one whos already infected
77               if status(p2)==1
78                  dummy=p2;
79                  p2=p1;
80                  p1=dummy;
81               end
82
83              %make pinform a function of p1 and p2
84              pinform=common(p1,p2)/maxcommon;
85
86              if(rand<pinform) %probability that they talk about this info
87
88                  infections(p1)=infections(p1)+1;
89
90                  status(p2)=1;
91
92                  SaveMeeting;
93              end
94
95
96          %if both are spreaders, one becomes a stifler
97          elseif(status(p1)==1 && status(p2)==1)
98              if(rand<pforget) %probability that they forget
99                  if(rand<0.5)%only one of them forgets(randomly chosen)
100                     status(p1)=2;
101                 else
102                     status(p2)=2;
103                 end
104             end
105
106         %if one is a stifler and one a spreader, both become stiflers
107         elseif((status(p1)+status(p2))==3)
108             if(rand<pforget)  %only by a certain probability
109                 status(p1)=2;
110                 status(p2)=2;
111             end
112         end
113     end
114     %remember that you updated the status
115        check(p1)=1;
116        check(p2)=1;
117
118 end
```

../../code/parameters.m

```
1 %pmeet=0.8; %probability, that a person wants to meet another person
2             %used in "talkstep" to determine meeting-vector
3
4 %pinform=0.9; %probability that a spreader meeting a ignorant tells him
5             %as "lambda" in (Alain Barrat)
6
7 pforget=0.2; %p, that if a spreader meets a spreader, one of them becomes a
8             %stifler. or if a stifler meets a spreader, both become
9             %stiflers
```

References

[1] A. Barrat, M. Barthlemy, A. Vespignani. Dynamical Processes on Complex Networks. Chapter 10.