Simulation of diffusion in dynamic networks

Contents

1 Introduction

The program explained in this manual realizes simulations of diffusion on dynamic networks, and analyzes the results. This program allows you to define your own diffusion model and run it on dynamic networks with or without a (static or dynamic) community structure.

This manual contains a lot of details about the programm, **if you want a quick explanation, see the README file instead**.

Required language and libraries :

- python 3
- matplotlib

2 Formats used

2.1 Diffusion model definition

A diffusion model allows to represent mathematically how a propagtion works. In a model, there are several states for nodes: healthy, infected, cured, etc. Rules define the passage of a node from one state to another: which nodes influence this transition, how does the node change state if it has been influenced. This program allows to create a diffusion model by defining the number of states, influencers for the change of state, and state change rules.

2.1.1 Example

The diffusion model is defined in xml. Here is an example of SI (Susceptible / Infected) model defined in xml. Nodes can be in two states, S or I. It is possible for a node in state S to become in state I if it is contamined with a certain probability by its neighborhood.

```
<model>
    <states>
        <state id="S"/>
        <state id="I" />
    </states>
    <edges>
        <edge source = "S" target = "I" rule = "neighborhood"
         transition = "probability 0.1" />
    </edges>
</model>
```

For each transition of state in the model, you have to create a line <edge source target rule transition /> like in the example.

2.1.2 Nodes who have influence for changing state

Three different cases can be used in the model defintion:

- a node depends on nothing to change its state (like the transition for I to R in the SIR model). In this case, use the command rule = "none" in the model definition.
- a node depends on its neighborhood to change its state, like in the SI model. In this case, use the command rule = "neighborhood" in the model definition (like in the example).
- a node depends on its community to change its state. In this case, use the command rule = "community" in the model definition (and do not forget to give a community file to the program).

2.1.3 Rules for changing state

Three different rules can be used to define how to change state:

- it is possible to use a probability. For instance in the SI model, if a S-node has an infected influencer node, it has a proability of becoming a I-node. In this case, use the command transition = "probability x" in the model definition. (x is the value of the probability).
- you can use a percentage of nodes in a state (like in an adoption model). For instance, if a node has x % of its influencer nodes in a given state, it moves to this state. In this case, use the command transition = "percentage x" (x is the value of the percentage).
- you can use a number of nodes in a state. For instance, if a node has at least 3 influencer nodes in a given state, it moves to this state. In this case, use the command transition = "number 3".

2.2 Graph format

The first two lines of the graph indicates if it is oriented and weighted. Then, each line describes a link between two nodes n_1 and n_2, as well as the start and end times of the link. **The file must be sorted by increasing start time**, or results will be inconsistent.

Example:

```
oriented = yes
weighted = no
3 4 1 5
1 2 1 10
2 5 1 10
2 6 2 4
```

```
5 6 2 6
1 4 2 7
3 6 3 8
5 7 3 9
2 3 4 5
4 7 4 7
1 3 4 8
3 7 7 9
6 7 7 10
```

2.2.1 Weights

In the graph definition, you can give weights: you can use these weights in the diffusion model to define the transition rule. For instance, if you use a SI model, you can define for each link a weight equal to a probability. If, in your model definition, you want to use these weights, use this command:

```
<edge source = "S" target = "I" rule = "neighborhood"
transition = "probability weight" />
```

If you do so, each link will have a different influence according to its weight.
You can also use weights if your transition is defined by a number or a percentage.

2.3 Initial states of nodes

This file contains the initial states of the nodes: each line contains the initial state of a node.
Example:

```
1 I
2 S
3 I
4 S
5 S
6 S
7 S
```

2.4 Communities format

This file is optional: it should be used only if the diffusion model uses communities, or if the user wants statistics about the diffusion related to communities. Each line contains an id

community, a node as well as the beginning and end times of its presence in the community. **The file must be sorted by increasing beginning time**, or results will be inconsistent.

Example:

```
1 3 1 9
2 5 1 10
1 7 2 14
2 4 2 11
2 6 3 7
1 2 5 8
```

On the first line, we see that node 3 is in community 1 from $t = 1$ to $t = 9$.

3 Simulation of the diffusion

The simulation is made by the diffusion.py program.

3.1 Parameter configuration

The simulation of diffusion is launched by the following command:

```
>> python3 diffusion.py model_file.xml graph_file.txt initial_states.txt
 <communities.txt>
```

model_file.xml is the file containing the diffusion model.
graph_file.txt contains the graph.
initial_states.txt contains the initial states of the node for the diffusion.
As explained above, the communities file is optional and depends on the diffusion model.

3.2 Result file

The *diffusion.py* program creates a file named *result_modelname_graphname.txt*. This file contains, for each timestep, and for each state in the diffusion model, the list of nodes in this state at this instant.

Example:

```
time 0
state I
1
3
state S
2
```

```
4
time 1
state I
1
2
3
state S
4
```

4 Analysis of the results

The analysis of the results is launched by the command:

```
>> python3 trace_analysis.py result_file.txt model_diffusion_file.xml
<communities_file>.txt
```

The communities file is optional, and can be used if you are interested by statistics about diffusion in communities.
result_file.txt contains the result produced by the *diffusion.py* program.
model_diffusion_file.xml contains the diffusion model used to produce the results file.

This program produces several curves:

- the number of nodes in each state over time (*nodes_state_modelname.png*)
- the percentage of nodes in each state over time (*percentage_nodes_state_modelname.png*)
- if a community file has been given as an input, for each community, the number of nodes in each state over time (*commmunity_idcomm_nodes_state_modelname.png*)
- if a community file has been given as an input, for each state, the percentage of nodes in this state in each community (*percentage_state_statename_in_each_comm_modelname.png*).

5 Case study

In this section, we realize a case study using a random dynamic graph with a community structure. First we perform two simulations with different diffusion models on this graph, and then, we compare the obtained results files. The files that have been used to this case study are in the repository of the program.

5.1 Random dynamic graph

The graph used here contains 100 nodes. This graph has four static communities with 25 nodes each. The idea is to have a lot of nodes inside a community, and very few between two communities. The dynamic is simple: at each time step, a few links in each community and bewteen commmunities are suppressed, and we create the same number of new links.

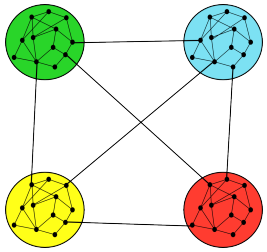

Figure 1: The graph contains four communities, with very few links between communitites

5.2 Diffusion models used

On this graph, we make two simulations of diffusion, with two different models:

- SI model: each node is either sane (S), or infected (I). An infected node has a given probability to contaminate its neighbours.
- adoption model: two states, S or I. A node in state S become in state I if at least $x\%$ of its neighbours are in state I.

The simulations have been launched with the commands:

```
>> python3 diffusion.py si.xml newman.txt ini_states_newman.txt
>> python3 diffusion.py adoption.xml newman.txt ini_states_newman.txt
```

5.3 Results

We analyze the traces of the two diffusion with the commands:

```
>> python3 trace_analysis.py result_si_newman.txt si.xml comm_newman.txt
>> python3 trace_analysis.py result_adoption_newman.txt adoption.xml comm_newman.txt
```

Here is an example of the curves given by the program:

In each diffusion, the infected nodes are in community 1 at the first step. We see on these curves that with the SI model, the diffusion quickly goes to other community. On the opposite, with the adoption model, the diffusion stays in the initial community.

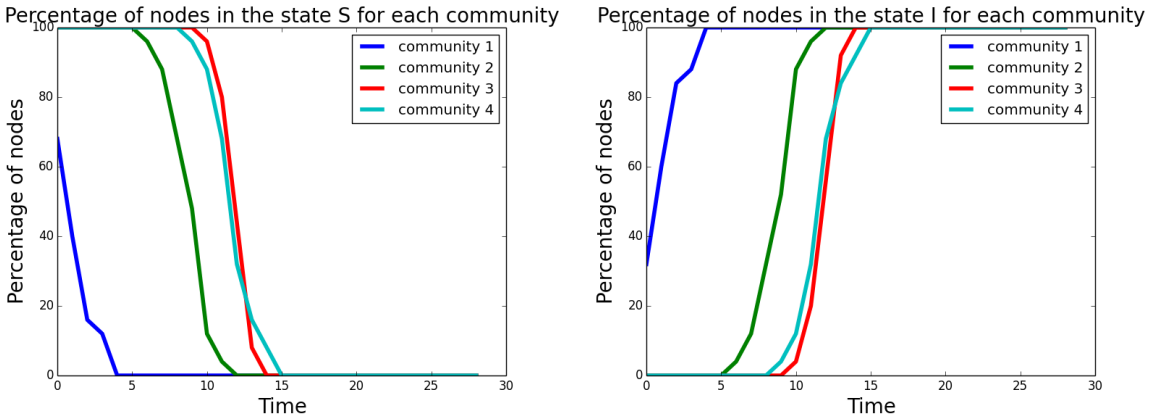

Figure 2: Diffusion with the SI model: for both states S and I, percentage of nodes in a state for each community

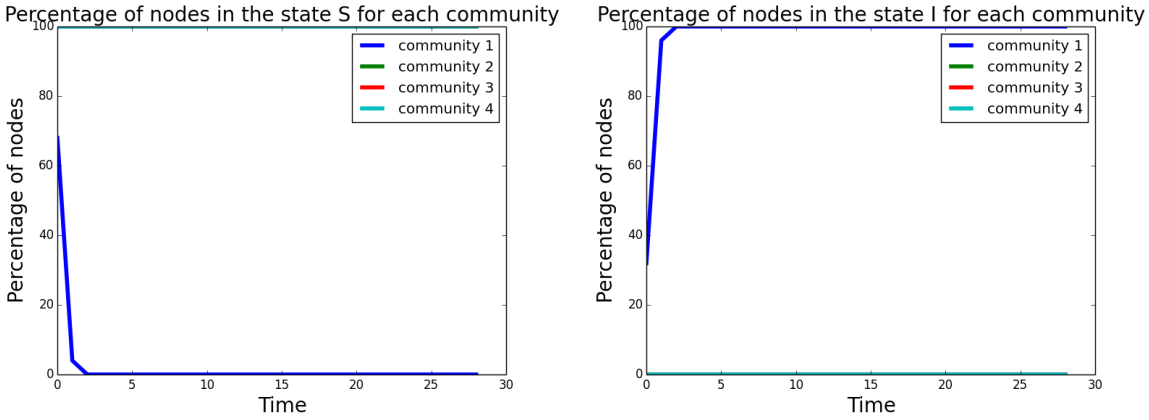

Figure 3: Diffusion with the adoption model: for both states S and I, percentage of nodes in a state for each community

6 Ideas to add functionalities

In this section, we describe how the program works in order to help the addition of new functionalities.

The file diffusion.py launches the diffusion. If you add new rules for your diffusion models, you must change this file so that the diffusion can manage with these new rules.

The file rules.py contains the rules to interpret the diffusion model. If you want to create new rules, add them in this file (and do not forget to change the diffusion.py file).

The file parser.py contains all the functions to read the files: the diffusion model file, the graph file, the community file.

The file trace_analysis.py contains the code to analyze the trace of the diffusion. If you want to compute new statistics or curve on the trace, change this file.

The file plots.py contains the code to plot the curves. If you want to change the look of your curves, or add new curves, use this file.