Network Diffusion — Framework to Simulate Spreading Processes in Complex Networks

Wrocław University of Science and Technology

Michał Czuba [1], Mateusz Nurek [1], Damian Serwata [1], Yu-Xuan Qi [2], Mingshan Jia [2], Katarzyna Musial [2], Radosław Michalski [1], Piotr Bródka [1]

[1]Wrocław University of Science and Technology, PL, EU
[2]University of Technology Sydney, NSW, AU

Network Science Lab

Introduction

Spreading phenomena are one of the issues considered by network science. They can be observed in various areas, such as dynamics of political opinions, marketing campaigns, spread of epidemics, etc. With the advancement of network science, analytical approaches have become insufficient for large graphs, prompting researchers to use computational methods. Moreover, recent years brought a significant expansion of the scope of network science beyond static graphs to encompass more complex structures. The introduction of streaming, temporal, multilayer, and hypernetwork approaches has brought new possibilities and imposed additional requirements. Unfortunately, the pace of advancement is often too rapid for existing computational packages to keep up with the functionality updates.

Problem

There are many very good and efficient tools that help in simulating diffusion processes in networks, e.g. `ndlib` (which we love).

However, if we consider...

...more complex network models,...

...spreading multiple processes at the same time...

...a gap among the available toolkits emerges.

Our Contribution

In order to address the issue, we decided to redesign, polish, and share the simulation environment we are internally using in the lab. Thus, in our recent work [1], we presented `network-diffusion`. To start using the library, just type in your shell:

```
pip install network-diffusion
```

Or scan this QR code to reach the docs with more examples:

Key Features of `network-diffusion`

- End-to-End Simulation Workflow for Discrete Diffusion Phenomena
- Support for Temporal Network Models
- Support for Multilayer Network Models
- A Bunch of Predefined Spreading Models
- An Interface for Implementing Custom Spreading Models
- New Centrality Measures
- `networkx` Compatibility

Example

Linear Threshold Model in Multilayer Networks

In a diffusion under the Linear Threshold Model (LTM), each node:

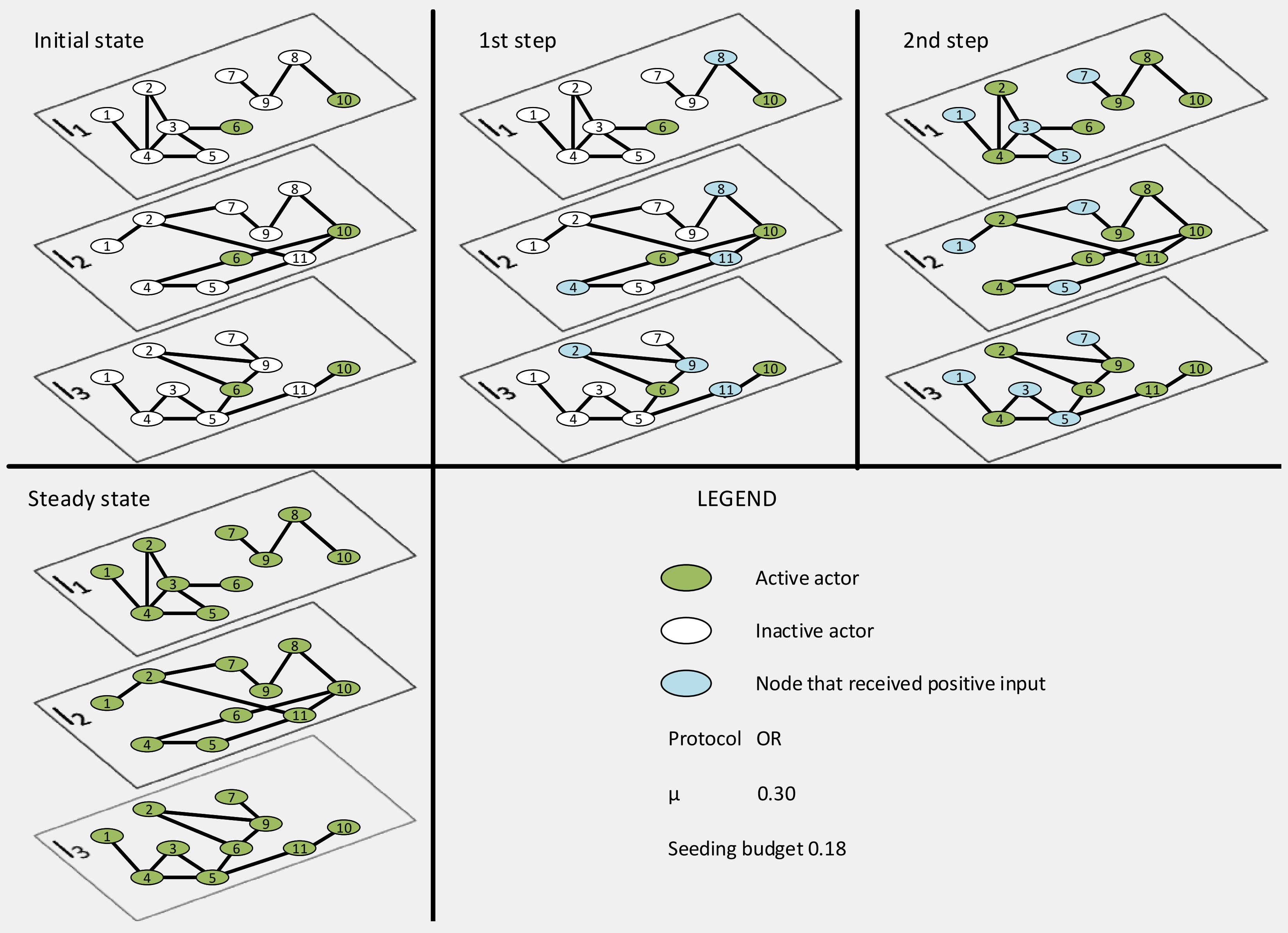
- can fall in two states: *active* and *inactive*,
- becomes *active* if the fraction of its *active* neighbors to all neighbours exceeds a certain threshold (μ).

In the case of multilayer networks — actors are the subject of the process, while the nodes are their auxiliary representation. Thus, we have to define how to aggregate impulses from the layers. In this example, we will consider OR strategy (aka protocol), which says that the actor can be activated if any of the nodes representing it gets activated.

Example

Scenario to Simulate

The Figure below shows the case we would like to model with `network-diffusion` — mLTM with homogeneous threshold $\mu = 0.3$ and protocol OR. The process will diffuse in a network with eleven actors, starting from the agents six and ten.

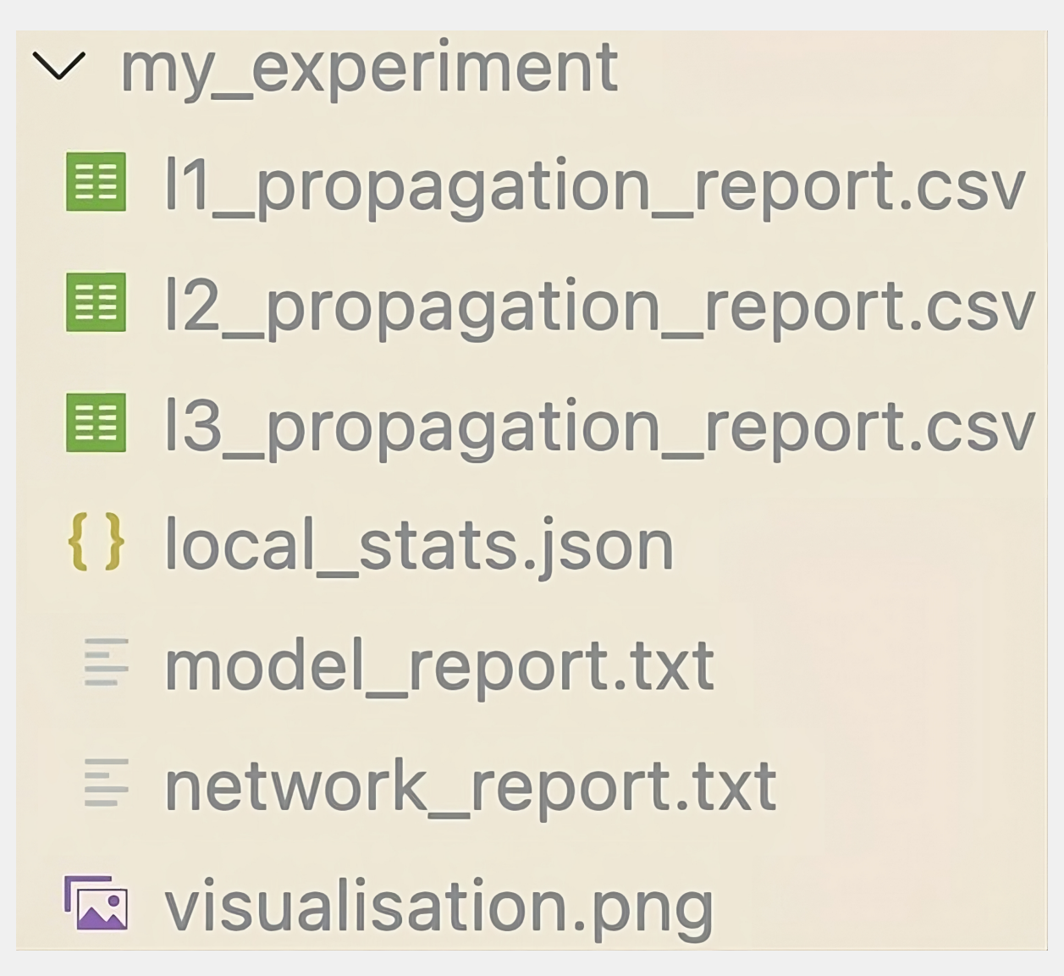


Modelling it with `network-diffusion`

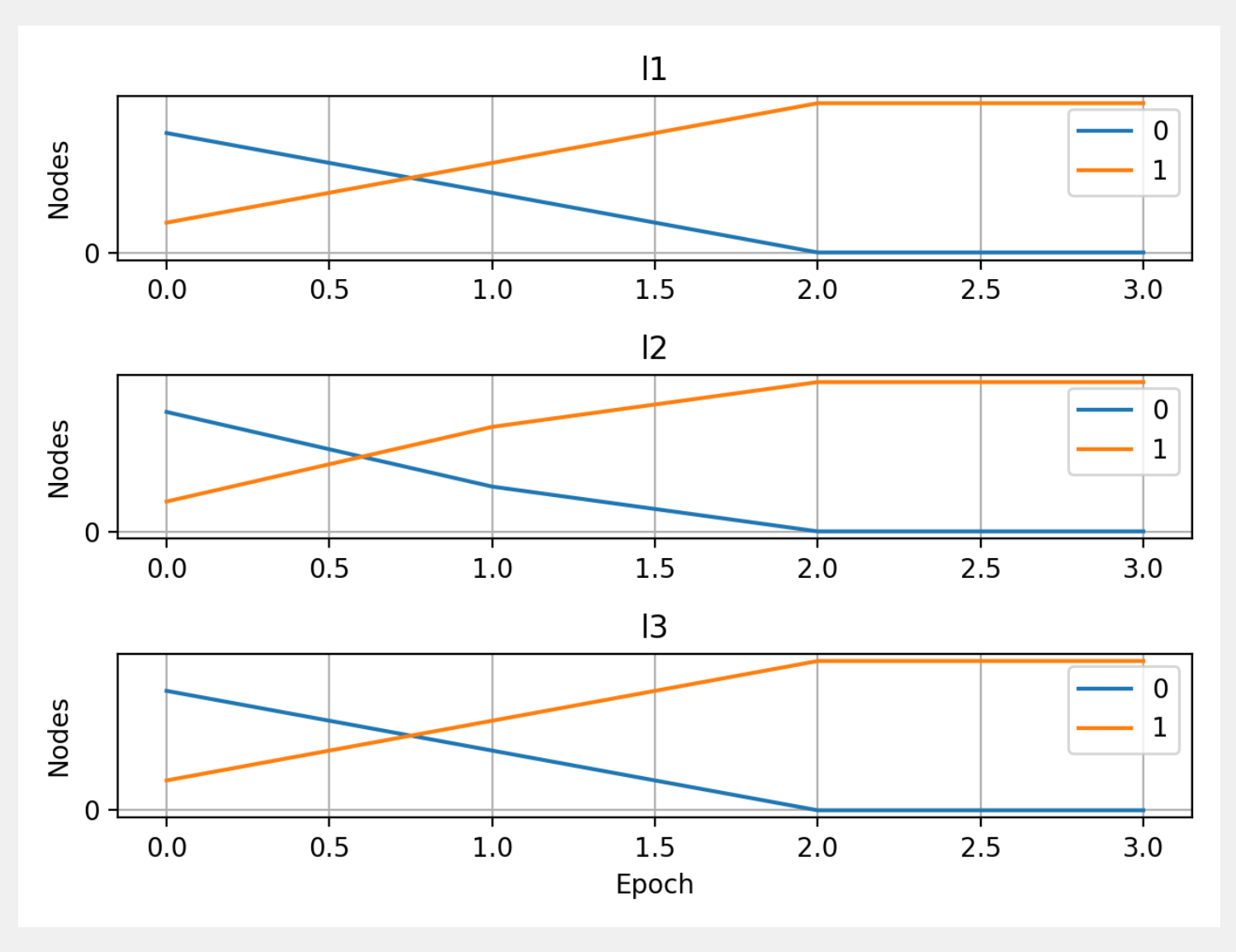
```python
import network_diffusion as nd

# get the graph - a medium for spreading
net = nd.mln.functions.get_toy_network_piotr()

# set actor 6 and 10 as seeds (we can use another heuristic here as well)
ac_6, ac_10 = net.get_actor(6), net.get_actor(10)
ranking_list = [ac_6, ac_10, *set(net.get_actors()).difference({ac_6, ac_10})]
seed_selector = nd.seeding.MockingActorSelector(ranking_list)
seed_quota = 100 * 2 / net.get_actors_num()

# define the model according to the given parameters
spreading_model = nd.models.MLTModel(
    seeding_budget=[100 - seed_quota, seed_quota],
    seed_selector=seed_selector,
    protocol="OR",
    mi_value=0.3,
)

# perform the simulation that lasts four epochs
simulator = nd.Simulator(model=spreading_model, network=net)
logs = simulator.perform_propagation(n_epochs=3)

# obtain detailed logs for each actor in the form of JSON
raw_logs_json = logs.get_detailed_logs()

# or obtain aggregated logs for each of the network's layer
aggregated_logs_json = logs.get_aggragated_logs()

# or just save a summary of the experiment with all the experiment's details
logs.report(visualisation=True, path="my_experiment")
```

Outcomes

The output logs contain a description of the network and the propagation model, a report of the spreading for all simulated phenomena, a capture of the states of each node during the simulation, and a brief visualisation of the propagation. They can be easily composed into subsequent data analysis pipelines.



```
my_experiment
  l1_propagation_report.csv
  l2_propagation_report.csv
  l3_propagation_report.csv
  local_stats.json
  model_report.txt
  network_report.txt
  visualisation.png
```



References

[1] Michał Czuba et al. "Network Diffusion — Framework to Simulate Spreading Processes in Complex Networks". In: *Big Data Mining And Analytics* (2024), pp. 1–13. DOI: `10.26599/BDMA.2024.9020010`. URL: `https://doi.org/10.26599/BDMA.2024.9020010`.