19th Workshop on Modelling and Mining Networks

Network Diffusion — Framework to Simulate Spreading Processes in Complex Networks

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
[2]University of Technology Sydney

05.06.2024

Network Science Lab

Agenda

In this talk, I will introduce a computational library — `network-diffusion`. Here is the agenda:

Network Science Lab

Motivation

Spreading phenomena are one of the issues considered by a network science. They can be obeserved in various areas like: dynamics of political opinions, marketing campaigns, spread of epidemics, computer viruses, etc.

Figure: Artistic representation of a social network.[1]

[1] Source: www.uniroma3.it/articoli/seminario-biased-opinion-dynamics-when-the-devil-is-in-the-details-138122

Motivation

Analytical approaches are often insufficient for large graphs, prompting researchers to use computational methods, i.e. simulators.

Motivation

Analytical approaches are often insufficient for large graphs, prompting researchers to use computational methods, i.e. simulators.

Thus, like in other branches of computer science, there have been developed tools which addres that issue, allowing to avoid starting from scratch and enhancing the reproducibility of results.

Network Science Lab

Motivation

There is a bunch of tools that helps in sumulating diffusion processes in networks:

- **NDlib**[?],
- GLEaMviz[?],
- SimInf[?],
- STEM[?],
- EpiModel[?],
- Sispread[?],
- ...

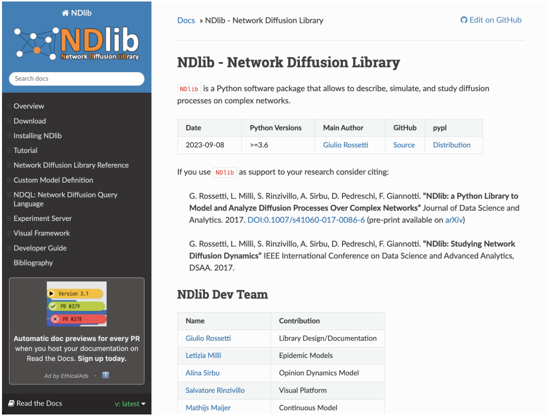

Figure: NDlib's website.

Motivation

However, if we consider...

...more complex network models,...

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

...**a gap among the available toolkits** emerges.

Motivation

A focus of our research group is oriented i.a. to: multilayer networks, temporal networks, spreading phenomena, data streams.

Network Science Lab

Motivation

A focus of our research group is oriented i.a. to: multilayer networks, temporal networks, spreading phenomena, data streams.

As a result of our recent activities, we decided to merge and wrap up a code we developed into a reusable library. We also decided to to share it with the community in an attempt of filling the gap in.

Network Science Lab

Motivation

A focus of our research group is oriented i.a. to: multilayer networks, temporal networks, spreading phenomena, data streams.

As a result of our recent activities, we decided to merge and wrap up a code we developed into a reusable library. We also decided to to share it with the community in an attempt of filling the gap in.

The main operating principles that we determined were:

1. compatibility with other tools commonly used in data science,
2. development of a tool as a framework with open interfaces,
3. supporting both multilayer and temporal networks,
4. supporting spreading models with discrete states.

Network Science Lab

Key Features

Functionalities of `network-diffusion`:

- end-to-end simulation workflow,
- predefined spreading models (for multilayer and temporal networks),
- an interface for implementing custom spreading models,
- support for the temporal network models (CogSNet + discrete windows),
- support for the multilayer networks,
- centrality measures for multilayer networks.

Key Features

Environmental requirements for `network-diffusion`:

- support for Linux, macOS, and Windows[2],
- Python (preferred 3.10) compatibility,
- C snippets in the CogSNet module to speed-up computations,
- NetworkX compatibility.

[2] Although we develop and use it mostly on Unix OSs

Key Features

To prepare the experiment we have to provide a network, a spreading model and auxiliary parameters. Then, the simulation unfolds as follows:

```
procedure PERFORM_PROPAGATION(network, model, epochs*)
    states_0 ← model.determine_initial_states()
    model.update_network(states_0)
    for e in [1, ..., epochs] do
        states_e ← model.network_evaluation_step(network)
        model.update_network(network, states_e)
    end for
    logs ← generate logs from experiment
return logs
end procedure
```

Example I - a Predefined Model

In this example, we will see how to trigger spread under the Linear Threshold Model within a simple, multilayer network.

Linear Threshold Model

Each node:

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

In case of multilayer networks the actors not the nodes[3] are a subject of the diffusion. Thus, we have to define how to aggregate impulses from the layers. In this example we will consider "OR" strategy, which says that the actor can be activated if any of nodes representing it in the network gets activated.

[3] which can be considered as avatars of the actors on the network's layers

Example I - a Predefined Model

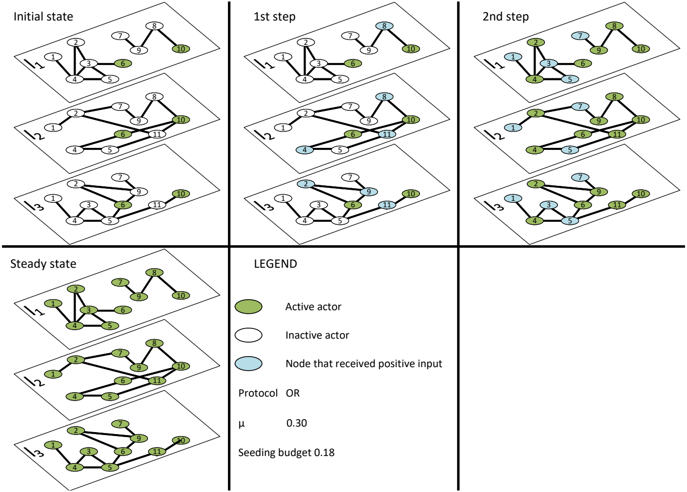

Figure: Propagation according to LTM with the *OR* strategy in a multilayer network - active actors: seeds $\{6, 10\}$, in a stable state: all of them.

Example I - a Predefined Model

Let's model this problem with `network-diffusion`!

Example II - a Custom Model

In this example we will consider a joint disease-awareness model (SIR-UA) that can be used e.g. to assess the effectiveness of various countermeasures against the spread of COVID-19 (**see the next presentation for details**):

Table: Transition weights with explanation.

Symbol	Description
α	probability of infection for unaware agents
α'	probability of infection for aware agents
β	probability of recovery
γ	probability of awareness for uninfected agents
δ	probability of awareness for infected agents

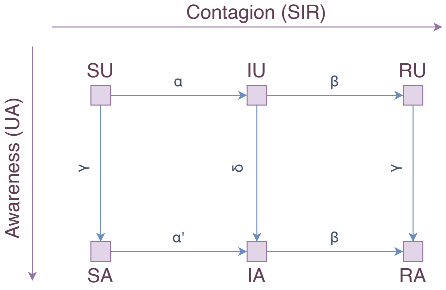

Figure: State and transition graph for SIR-UA.

Example II - a Custom Model

To create an own spreading model, we have to extend the abstract base class (`nd.models.BaseModel`) by implementing:

- a field `_compartmental_graph`,
- a field `_seed_selector`,
- a method `determine_initial_states`,
- a method `agent_evaluation_step`,
- a method `network_evaluation_step`,
- methods `__str__` and `get_allowed_states`.

The following entities (with implemented method `update_network`) are utilised in the class `nd.Simulator` which orchestrates the experiment.

Example II - a Custom Model

Let's model this problem with `network-diffusion`!

Resources and References

The library can be installed via:

```
pip install network-diffusion
```

Other useful resources have been also published:

- PyPI website: `pypi.org/project/network-diffusion`
- GitHub page: `github.com/anty-filidor/network_diffusion`
- Reference guide: `network-diffusion.readthedocs.io`
- A preprint of the paper: `arxiv.org/abs/2405.18085`

Network Science Lab

Limitations of `network-diffusion`

There is still some work to do...

- implemented in Python — performance can be better (only CogSNet implemented in C),
- limited to discrete spreading models,
- much less pre-defined spreading models than in NDlib,
- no user interface — a limitation for non-programmers,
- GPL licence.

References I

Thank you for your attention!

If you like `network-diffusion` please star the repo :)