

# Simulating Spreading of Multiple Interacting Processes in Complex Networks

1<sup>st</sup> Michał Czuba

Department of Artificial Intelligence  
Wrocław University of Science and Technology  
Wrocław, Poland  
michal.czuba@pwr.edu.pl

2<sup>nd</sup> Piotr Bródka

Department of Artificial Intelligence  
Wrocław University of Science and Technology  
Wrocław, Poland  
piotr.brodka@pwr.edu.pl

**Abstract**—Investigating the interaction between spreading processes in complex networks is one of the most important challenges in network science. However, whether we would like to know how the information campaign will affect virus spreading or how the advertising campaign of the new iPhone will affect the sales of Samsung phones, we need an environment that will allow us to evaluate under what conditions our spreading campaign will be effective. *Network Diffusion* is a Python package that should help do that. In this paper, we introduce its operating principle and main functionalities, including simple examples of simulations that can be performed using it.

**Index Terms**—interacting spreading processes, epidemics, network science, multilayer network

## I. INTRODUCTION

For almost two years now, the resources of the entire world have been used for the fight against the "invisible" enemy. We spent countless hours and trillions of dollars to stop the COVID-19 pandemic, and still, millions of people have died from the virus or the collapse of healthcare systems around the world. One of the challenging tasks is understanding how different countermeasures, such as information campaigns, lockdowns, or vaccinations, affect the progression of the virus to choose those that will save both human lives and the economy.

In this paper, we would like to present the *Network Diffusion* package [1] that allows simulating the spread of many coexisting processes in single and multilayer networks. An example of such processes would be, of course, coronavirus spread, interacting with an information campaign and vaccinations, but also the interaction between two advertising campaigns (e.g. iPhone vs Samsung or Cannon vs Nikon), two political campaigns (e.g. Biden vs Trump), diseases (e.g., AIDS and Tuberculosis), to name just a few [2]. However, before we can present the *Network Diffusion* package, we would like to introduce a few key concepts from network science [3].

### A. Complex networks

Complex networks are the backbone of all complex systems [3] starting from the nervous system of living organisms [4], through our infrastructure systems (electric grid, water pipes, airports, etc.) [5], [6], and ending with relationships

between people [7]. Complex networks can be represented and analysed in multiple ways, e.g., adjacency matrix [3] or property vectors [8], however, the most common approach is to represent a complex network as a graph [3] or a set of graphs (for multilayer networks) [9], [10]. The multilayer network, according to [9], is defined as quadruple  $M = (N, L, V, E)$ ; where:  $N$  is a set of actors;  $L$  is a set of layers;  $V$  is a set of nodes,  $V \subseteq N \times L$ ;  $E$  is a set of edges  $(v_1, v_2) : v_1, v_2 \in V$ , and if  $v_1 = (n_1, l_1)$  and  $v_2 = (n_2, l_2) \in E$  then  $l_1 = l_2$ .

### B. Spreading phenomena

Spreading phenomenon covers a wide spectrum of processes, starting from information diffusion [11], through virus [12], opinion [13], innovation [14] spreading and ending with the spread of influence [15]. Fortunately, all these processes are similar if we look at their high-level components [3] (i) *What* phenomenon is spreading (e.g. information, opinion, virus), (ii) *How/Who* it is spreading (e.g. network nodes, actors, agents), and (iii) *Where* it is spreading (i.e. network type). All these elements carry a different context to the simulation results, but, at the same time, they can be imagined as "containers" for algorithmic structures, which will be introduced later. An example of those three components can be a viral video that spreads on a social networking site. Here, *what* is the video or digital content, *how/who* is a link to the website or a post on social media, and *where* is a social network that is the foundation of this particular social networking site.

1) *Epidemic modelling framework*: The initial models developed by epidemiologists are based on two assumptions [3]: compartmentalisation and homogeneous mixing. The first condition says that the state of each individual is discrete. This means (continuing with epidemiological analogies) that the relation of a given node to a disease can be described by  $n$  states, e.g. *healthy*, *sick*. The second assumption concerns the ability of the nodes to change their states. Each individual can interact with all nodes in a given time unit. Using that hypothesis, we can build many various spreading models like SI (Suspected-Infected), SIS (Suspected-Infected-Suspected), SIR (Suspected-Infected-Recovered), SIRS (Suspected-Infected-Recovered-Suspected), and so on. Epidemic models can be used to simulate other

spreading processes. The most common example would be information spreading, where we have models like UAU [16] (Unaware-Aware-Unaware) or UAF [17] (Unaware-Aware-Forgot), which are based on SIS and SIR models, respectively.

2) *Independent cascade model*: Independent cascade model (ICM) [15] has a much different mechanism and is widely used to simulate influence diffusion. Its principles reject the homogeneous mixing assumption because of the way the phenomenon propagates - it cannot be captured holistically for a given network. Here, the spreading takes the form of a cascade. Each newly activated node has one chance to activate its neighbours in each step. It is based on the likelihood of activation (aka. propagation probability) stored at the edge connecting them [18].

3) *Linear threshold model*: This model is based on the concept of the activation threshold that is defined for each node of the network [15]. It determines a minimum value of the influence of its neighbours to change its state. In other words, the minimum value above must be the sum of the intensity of the connections from the neighbouring activated nodes to result in activation [18]. When comparing ICM and Linear Threshold Model (LTM), one can say that the first one is based on the *push* mechanism, i.e., a node pushes its influence to its neighbours. In contrast, LTM is based on the *pull* mechanism, i.e. the node pulls the influence from its neighbours.

### C. Interacting spreading processes in networks

When analysing how various phenomena in networks propagate, we have to ask the following questions: *what is being propagated?* and *where is it being propagated?*. As an answer, we get four general possibilities: (1) a single process in a single network, (2) a single process in a multilayer network, (3) multiple processes in a single network, and (4) multiple processes in a multilayer network.

The last two options bring the possibility of spreading multiple processes that can interact with each other. Thus, in addition to modelling each phenomenon, we also need to model the interactions between them, i.e., how they influence each other (and that is exactly a problem that *Network Diffusion* solves). Below, we briefly describe the four possible variants of interactions.

Firstly, we can distinguish supporting processes (about 11% of research in the domain [2]) that mutually support each other by increasing coverage and velocity. Here, a good illustration is that chronic diseases (such as asthma) are a catalyst for contracting COVID-19.

The next genres are competing processes (about 36% publications [2]). In this case, one process causes suppression of the propagation of the other. A good example of this is the presidential election: if the number of people influenced by candidate X increases, the number of people influenced by candidate Y has to decrease, and at the end of the day, only one candidate can win.

The third case, the mixed approach, covers about 46% of the publications [2]. It considers instances where one process supports the second, but the second competes with the first.

An interaction between disease and awareness can be a good example. People who get sick become aware of a disease, so the more people get sick, the more people will be aware. On the other hand, people aware of the pandemic will take preventive actions to limit its spreading.

The last case is when processes do not interact with each other. It appears only in 7% of the papers in the domain [2].

## II. SIMILAR SOFTWARE PACKAGES

To gain valuable recognition of similar solutions to *Network Diffusion*, an appropriate tool review methodology was adopted. For this, the general state of the software available in the field was taken into account. We started from a simple reconnaissance using a search engine and focused more on the tools available for the Python language; however, it was not a strict condition. Our final goal was to see a general cross-section of state-of-the-art. As a result, information on more than twenty different tools was obtained. We analysed them and divided them into two groups: *packages dedicated for network spreading simulations* and *general complex network analysis software*.

### A. Software for spreading processes simulations

1) *GLEaMviz*: The first application that has functionalities corresponding to the designed software is a GLEaMviz. It works with real data, population density, and migration around the world, combined with stochastic models of disease propagation. As a result, it provides a sophisticated simulation environment. Due to the large scale of the experiments (the whole world), a single node is a population of a given size (defined by the user). A very interesting feature is the manual definition of the epidemiological model. GLEaMviz makes this possible by manipulating the compartments (understood in the same way as in sec. I-B1). Allowable transitions between them are also fully definable. The user can also select the geographical start of the disease, the initial percentages of individuals belonging to a given compartment, its duration, etc. There is also an option to generate various visualisations at the end of the experiment. Despite the interesting functionalities mentioned above, GLEaMviz has a rather large disadvantage: it only allows the propagation of one process at a time [19].

2) *NDLIB*: Network Diffusion LIBrary is a Python package based on the NetworkX library. It allows performing simulations with many predefined epidemiological models (such as: SIS, SIR, SEIR, etc.), influence group (LTM, ICM, Profile, etc.), opinion group (Voter, Sznajd, etc.), and even dynamics (models with the capacity to change the topology of network). Moreover, the user can create its own customised models. Results visualisation is also possible via Matplotlib or Bokeh with the flexibility to append a custom graphical engine. NDLIB also has some interesting run-time features. First of all, it includes an option to perform a "multi-execution" of the simulation by parallel computing. As this kind of experiment is generally stochastic, this feature gives a chance to see the general behaviour of the observed phenomena. It also enables running the simulation on a server (as well

TABLE I  
TOOLS WITH CORRESPONDING FUNCTIONALITIES TO *Network Diffusion*.

| Name      | Type           | Environment | Functionalities  | Access |
|-----------|----------------|-------------|--|--------|
| GLEaMviz  | app.           | desktop     | single model propagation, high freedom of model definition, visualisation of the experiment [19]     | free   |
| NDLIB     | library        | Python      | single model propagation, high freedom of model definition, visualisation of the experiment [20]     | free   |
| SimInf    | library        | R           | propagation of custom model, visualisation [21]  | free   |
| Sisspread | app.           | console     | SI, SIS or SIR model propagation, returns numerical data [22]  | free   |
| STEM      | Eclipse plugin | desktop     | single model propagation, visualisation, workflow like finite element method (FEM) environments [23] | free   |

as locally). For users unfamiliar with Python, the authors created NDQL, a query language (based on SQL syntax) that supports elementary NDLIB commands. For those who cannot programme, they also provided a "visualisation framework" to play with some of the models implemented with a GUI-based tool [20]. NDLIB is a very useful library with many features. However, it does not directly support experiments where many processes interact together.

3) *SimInf*: The next software is SimInf, a process diffusion package for R. Its API has been designed in a very interesting way. This tool allows a user to define his own models by parsing the appropriate string. Interestingly, no network is needed to run the simulation, as the algorithms work on the assumption of homogeneous mixing. After a successful simulation, it is possible to display a summary graph. Nevertheless, we must note that no support for operations on real networks and the lack of an option to define multiprocess experiments [21] are distinct shortcomings.

4) *Sisspread*: Sisspread is a simple application implemented in C. It is a console tool without a graphical interface. It focuses on three models (SI, SIS, SIR) with the possibility of deep analysis of the experiments performed. This tool supports very basic IO operations - the user can upload a custom network that meets certain requirements (without the option to manipulate it). As a result of the experiment, numerical data are returned for further analysis [22].

5) *STEM*: Another advanced system is STEM. As an extension of the well-known Eclipse IDE, it uses its graphical layout and the general philosophy of user interaction. It is reminiscent of FEM systems in the way it works. Just like them, it requires the user to specify a medium (equivalent to a model of a physical object) where the simulation is performed with its discretisation level (i.e. whether the node is a municipality or an entire county). The next step is to attach an appropriate solver propagation model and set the starting parameters of the experiment. Once this is done, it is possible to visualise the spread progress. As in other programmes, it does not support multiple spreading processes [23].

## B. Software for complex network analysis

Due to the large number of tools found (see tab. II), we describe below only the most interesting of them.

1) *Gephi*: Gephi is an application for visualising networks in many aspects, with extensive graphical capabilities. It allows

the user to perform exploratory data analysis (EDA) in real-time on basic structures and connections between objects. Gephi is also designed for social network analysis - it has an easy way to map different portals (Facebook, Twitter, etc.) and Small World networks with each other. It can also visually represent biological data and serve as a tool for creating scientific posters. Additionally, within Gephi, the basic metrics for network analysis have also been implemented. The package works with many file types and supports integration with plugins written by external developers. In terms of non-functional aspects, it has an interface that does not require programming skills [27].

2) *iGraph*: iGraph is a library designed for several programming languages (i.e. C, R, Python, and Mathematica). The three main paradigms of iGraph are: easy handling of graph algorithms, fast handling of large networks, and enabling interaction with high-level languages like R. In many aspects, this tool is similar to pure NetworkX, but it is also able to run simple epidemic models like SIR. For clarity, it should be added that this functionality is rather residual [30].

3) *NetMiner*: NetMiner is a comprehensive environment for network analysis and visualisation. Unlike the tools described above, it is not free software. Among the functionalities that distinguish it from the others, we can list for sure the possibility of EDA (e.g., analysis of written text and, on this basis, creation of semantic network analysis), clustering, classification, etc. NetMiner also has extensive visualisation capabilities, both static and dynamic, and, what is interesting, three-dimensional. Another important feature of this tool is the presence of a Python-based console. It can even be upgraded by adding external packages. Moreover, NetMiner allows the compilation of written programmes, which is a big advantage of it in terms of securing the code against third parties [34].

4) *NetworkX*: NetworkX is one of the most important packages among Python data science libraries. Its capabilities are large and mainly concern static network analysis. NetworkX also has visualisation functionality, but in its official documentation, it recommends the use of Cytoscape, Gephi, and Graphviz to perform any visualisation [36]. Furthermore, because of its integration with the Python language, the long stability of this library can be assumed. NetworkX has a large number of algorithms and metrics that are useful for network analysis, which makes it virtually unrivalled among other Python programming tools. Moreover, this library is a

TABLE II  
SOFTWARE FOR COMPLEX NETWORK ANALYSIS.

| Name         | Type    | Environment     | Functionalities   | Access     |
|--------------|---------|-----------------|---|------------|
| AllegroGraph | DBMS    | desktop         | graph database management system, visualisation, static analysis of networks [24]           | free       |
| Arcgis       | app.    | desktop, web    | static analysis of multilayer networks, visualisation, IO [25]                              | free       |
| Cytoscape    | app.    | desktop         | static analysis of networks, visualisation [26]   | free       |
| Gephi        | app.    | desktop         | static analysis of networks, visualisation [27]   | free       |
| Giraph       | library | Java            | static analysis of networks [28]  | free       |
| Graphviz     | app.    | console         | visualisation [29]  | free       |
| iGraph       | library | Python, R, C, M | static analysis of multilayer networks, visualisation, IO [30]                              | free       |
| Multinet     | package | R Python        | manipulation of multilayer networks, static analysis, visualisation IO [31]                 | free       |
| Muxviz       | app.    | desktop         | manipulation of multilayer networks, static analysis, static and dynamic visualisation [32] | free       |
| Neo4j        | app.    | desktop         | graph database management system, visualisation, static analysis of networks [33]           | paid       |
| NetMiner     | app.    | desktop         | static analysis of networks, visualisation, IO, Python-like scripting [34]                  | paid       |
| Network      | library | R               | static analysis of multilayer networks, IO [35]   | free       |
| NetworkX     | library | Python          | static analysis of networks, visualisation, IO [36]   | free       |
| Nodexl       | plugin  | Excel           | static analysis of networks, visualisation, IO [37]   | free, paid |
| Pymnet       | library | Python          | static analysis of multilayer networks, visualisation, IO [38]                              | free       |
| Tulip        | app.    | desktop         | static analysis of networks, visualisation, IO, Python-like scripting [39]                  | free       |
| yEd          | app.    | desktop         | static analysis of networks, visualisation, IO [40]   | free, paid |

base for several tools listed in this review.

### C. Summary

Having the analysis of the existing solutions done, it is easy to see that they can be divided into several types based on the main functionality: (1) software for static analysis of networks, (2) software for visualising networks, and (3) software for simulation of process propagation in networks. We can also distinguish the following groups based on the type of the tool: (1) web applications, (2) locally executed applications, (3) libraries for languages popular in data science, and (4) others (database management systems, plugins for other applications, etc.). However, it is worth noting that, after careful reconnaissance, virtually no simulation environment was found for the simultaneous propagation of multiple processes.

## III. OPERATING PRINCIPLE

The core functionality of the software requires adequate attention, especially since, as demonstrated in sec. II, no solution was found that covers the problem of spreading multiple processes in networks. Hence, before the start of designing the software, we had to address and answer two questions: (1) how to run any number of processes during a single experiment, and (2) how to determine (in a general way) the interactions between processes? To make further deliberations easier to understand, we defined a domain dictionary with all important terms (see tab. III)

### A. Generalisation of single processes modelling

According to the concepts described in sec. I, we can notice a similarity between many discrete processes. Moreover, due to the numerical nature of the simulations carried out, it is not possible to hold the assumption of homogeneous mixing. Taking these into account, it is easy to conclude that models

like SIS, SIR, SIRS, etc. can be reduced to specific cases of a model with  $T = [t_1, t_2, \dots, t_k]$  possible states, where each of the network nodes can be in one of them at a given time step. Moreover, the transition from one state to another takes place under a certain weight, which can be called pseudo-probability. Let us further note that, e.g. in the SIR model, it is impossible to transit directly from  $S$  to  $R$ . Thus, a state change is only possible between neighbouring states, i.e. from  $t_k$  one can only move to  $t_{k-1}$  and  $t_{k+1}$ .

*Definition 3.1:* It follows that every model  $\mathfrak{M}$  belonging to the mentioned group can be described for the purposes of the designed system by: vector of states  $T = [t_1, t_2, \dots, t_k]$  and vector of transition weights between neighbouring states  $W = [w_1^2, w_2^3, \dots, w_{k-1}^k]$  (where  $w_n^m$  means a weight of transition  $n \rightarrow m$ ); which can be combined in a matrix:

|          | $t_1$   | $t_2$   | $t_3$   | $\dots$  | $t_k$   |
|----------|---------|---------|---------|----------|---------|
| $t_1$    |         | $w_1^2$ |         |          | $w_1^k$ |
| $t_2$    | $w_2^1$ |         | $w_2^3$ |          |         |
| $t_3$    |         | $w_3^2$ |         |          |         |
| $\vdots$ |         |         |         | $\ddots$ |         |
| $t_k$    | $w_k^1$ |         |         |          |         |

For example, a SIRS model with coefficients:  $\beta$  (individual's infection rate),  $\mu$  (individual's recovery rate), and  $\xi$  (loss of immunity) can be expressed as a vector of states:  $T = [s, i, r]$ ; vector of transition weights between neighbouring states:  $W = [\beta, \mu, \xi]$ , and process matrix:

|     | $s$   | $i$     | $r$   |
|-----|-------|---------|-------|
| $s$ |       | $\beta$ |       |
| $i$ |       |         | $\mu$ |
| $r$ | $\xi$ |         |       |

TABLE III  
DOMAIN DICTIONARY FOR TERMS INTRODUCED TO DESIGN THE *Network Diffusion* PACKAGE.

| Term              | Symbol   | Description   |
|-------------------|--|---|
| Epoch             | $\varepsilon_i \in \epsilon : \{1, \dots, \varepsilon\}$         | A discrete-time unit of the Experiment when all Nodes of the Network have updated their State against each of the Processes.  |
| Experiment        | $X = (\mathfrak{M}, M, \epsilon)$                                | A complete run of the Propagation model on the Network for a defined number of Epochs.  |
| Network           | $M = (N, L, V, E)$   | A multilayer network (as introduced in sec. I-A).   |
| Process           | $T = [t_1, t_2, \dots, t_k]$                                     | Singular phenomena affecting the Network, described by an ordered set of States. Each Process must consist of at least two States and, in the package, is identified with the particular Network Layer. |
| Propagation model | $\mathfrak{M} = (T_1 \times \dots \times T_n, W)$                | A definition of how Processes should interact with each other, i.e. a set of all possible global States of the Nodes and a set of Weights under which Transitions are allowed during Experiment.        |
| State             | $t_{ajx} :$<br>$t_a \in T_1; t_j \in T_2; t_x \in T_3$           | Categorical value denoting a relationship of the Node against a particular Process (a local State - here e.g. $t_a$ ) or a set of interacting Processes (a global State - here $t_{ajx}$ ).             |
| Transition        | $t_{aj} \rightarrow t_{ak} :$<br>$t_a \in T_1; t_j, t_k \in T_2$ | Possible change of State of the Node at potential denoted by the Weight. Transitions are allowed between neighbouring States, e.g. they must differ only by one State.                                  |
| Weight            | $w_{t_j}^{t_k} \in W$  | Pseudo-probability of particular Transition occurrence (here: $t_j \rightarrow t_k$ ), valued by a number in range $[0, 1]$ .   |

### B. Example of two processes model

The way we defined the coexistence of several processes looks similar, but it is scaled into additional dimensions. As we know from sec. I-C, processes interactions can be supporting, competing, mixed, or independent. Let us consider a two-process model  $\mathfrak{M} = (T_1 \times T_2, W)$  representing propagation of SIR-like disease ( $T_2$ ) and UA-like vaccine against it ( $T_1$ ) - a classic example of competing processes. Thus, we have:

- $T_1 = [u, v]$ ;  $u$  - unvaccinated individual,  $v$  - vaccinated individual
- $T_2 = [s, i, r]$ ;  $s$  - suspected individual (i.e. not yet affected by illness),  $i$  - infected individual,  $r$  - recovered individual

Both processes affect each node of the network. Therefore, we can describe all possible states in the model with the following set (a Cartesian product -  $T_1 \times T_2$ ):  $\{us, ui, ur, vs, vi, vr\}$ . Moreover, under the assumption that at each simulation step, it is possible only to change the state of a node in the dimension of the singular process, we can list all available transitions (see fig. 1 for visual presentation):

- for  $T_1$  (vaccination process); for  $s$  constant:  $us \rightarrow vs$ , for  $i$  constant:  $ui \rightarrow vi$ , for  $r$  constant:  $ur \rightarrow vr$ ,
- for  $T_2$  (disease process); for  $u$  constant:  $us \rightarrow ui \rightarrow ur$ , for  $v$  constant:  $vs \rightarrow vi \rightarrow vr$

Please note, that if we were to assign appropriate weights to particular transitions that reflect the probability of a state change in one process given the others, we could obtain the effect of reinforcing or suppressing between processes. For example,  $w_{us}^{ui} > w_{vs}^{vi}$  means that it is easier for unvaccinated individuals to get infected than for those who are vaccinated.

Moreover, with a more detailed analysis of fig. 1, we can see the advantages of the assumption that transitions are allowed only in the direction determined by the singular process. For our example, it means that changes of node's state can occur only in the  $T_1$  axis (e.g.  $us \rightarrow vs$ ) or the  $T_2$  axis (e.g.  $us \rightarrow ui$ ), but never "diagonally". This eliminates obviously incorrect transitions such as  $ui \rightarrow vr$ , which denotes a change

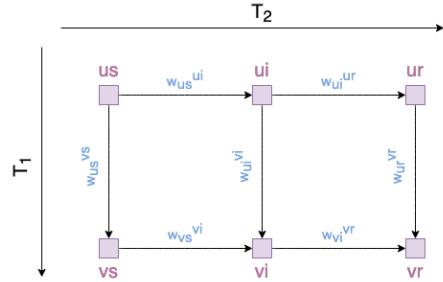


Fig. 1. Example of two process model  $\mathfrak{M} = (T_1 \times T_2, W)$ ;  $T_1 = [u, v]$ ;  $T_2 = [s, i, r]$ ;  $W = [w_{us}^{ui}, w_{ui}^{ur}, w_{vs}^{vi}, w_{vi}^{vr}, w_{us}^{vs}, w_{ui}^{ui}, w_{ur}^{vr}]$ .

of an unvaccinated and suspected individual to vaccinated and recovered one in one time step without even getting infected.

This assumption is also beneficial for theoretically possible transitions, where, at the same time step, not one but two or more processes change their states. First of all, it greatly simplifies defining the impact of one process on another. Furthermore, looking ahead, it greatly simplified the implementation of *Network Diffusion*. Let us consider all theoretically possible transitions  $us \rightarrow vi$ , (unvaccinated, suspected)  $\rightarrow$  (vaccinated, infected). Let us also assume that the fact of being vaccinated significantly reduces the probability of getting infected. Then we have three ways to get from  $us$  to  $vi$ :

- 1) (unvaccinated, suspected)  $\xrightarrow{w_{us}^{vs}}$  (vaccinated, suspected)  $\xrightarrow{w_{vs}^{vi}}$  (vaccinated, infected)
- 2) (unvaccinated, suspected)  $\xrightarrow{w_{us}^{ui}}$  (unvaccinated, infected)  $\xrightarrow{w_{ui}^{vi}}$  (vaccinated, infected)
- 3) (unvaccinated, suspected)  $\xrightarrow{w_{us}^{vi}}$  (vaccinated, infected)

In such a context, it is easy to see that setting  $w_{vs}^{vi} \ll w_{us}^{ui}$  models the real-world scenario that vaccination significantly reduces the probability of infection. The other weights are also intuitive to interpret, with one exception. The value of  $w_{us}^{vi}$ , makes impossible to deduce the relation between  $T_1$  and  $T_2$ .

### C. Generalisation of modelling multiple processes

Using the example presented in the sec. III-B, it is possible to define a general scheme of the multiprocess model that is used in the *Network Diffusion*:

**Definition 3.2:** The mutual impact between  $n$  processes meeting def. 3.1, can be represented by a  $n$ -dimensional orthogonal grid (spanned by points formed by the Cartesian product of the sets of states of each process), in which the connections between points exist only in directions along the axes denoting processes (i.e. transitions are allowed only between points differing by a single coordinate).

This approach sufficiently answers the two questions posed at the beginning of this section; i.e. it allows us to define any number of processes within a single experiment and to determine their influence on each other clearly.

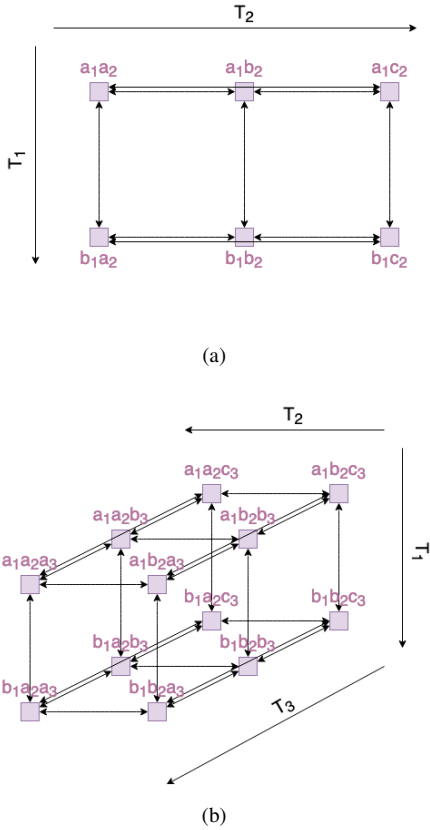


Fig. 2. General examples of multiprocess models (top)  $\mathfrak{M} = (T_1 \times T_2, W)$ ,  $T_1 = [a_1, b_1]$ ,  $T_2 = [a_2, b_2, c_2]$ ; (bottom)  $\mathfrak{M} = (T_1 \times T_2 \times T_3, W)$ ,  $T_1 = [a_1, b_1]$ ;  $T_2 = [a_2, b_2]$ ;  $T_3 = [a_3, b_3, c_3]$ ; For readability of diagrams vectors of weights  $W$  are not depicted.

### D. General simulation framework

The concepts presented allow us to assume that there is only one process running in a given layer of the network. This means that the network should have as many layers as many processes are simulated. As a result, with the multilayer structure, each node will be described in the context of each of the defined processes. However, if the layers do not have

an identical set of vertices, it will mean that a given node is indifferent to the process in whose layer it does not exist (e.g., a person who does not use social media and cannot be affected by information spreading via Twitter).

Continuing this line of reasoning, one can arrive at a general simulation scheme. For the convenience of the reader, it is presented as alg. 1. This allows simulations to be run on custom, discrete models with a virtually infinite number of processes and internal states.

---

#### Algorithm 1 Schema of the simulation in *Network Diffusion*.

---

**Require:**  $\mathfrak{M}$

$$\mathfrak{M} = (T_1 \times \dots \times T_k, W) \quad (1)$$

**Require:**  $M$

$$M = (N, L, V, E) : \quad (2)$$

$$\bigwedge_{i \in \{1, \dots, k\}} L_i \rightarrow T_i;$$

$$\bigwedge_{n \in N} n \rightarrow \tau_n = \{\tau_n^1, \dots, \tau_n^k\} \equiv \{t_1 \in T_1 \vee \emptyset, \dots, t_k \in T_k \vee \emptyset\};$$

**Require:**  $\epsilon$

$$\epsilon = \{1, \dots, \varepsilon\} \quad (3)$$

```

1: procedure  $X(\mathfrak{M}, M, \epsilon)$ 
2:   for  $\varepsilon$  in  $\epsilon$  do
3:     for  $l$  in  $L$  do
4:       for  $v$  in  $\{N \times l\}$  do
5:         for  $\bar{v}$  in  $\{(N \times l) - v\}$  do
6:           if  $\tau_v^l \neq \tau_{\bar{v}}^l$  then
7:             if  $\text{BernoulliTrial}(p = w_{\tau_n^l - \tau_v^l + \tau_{\bar{v}}^l}^{\tau_n}) = 1$  then
8:                $\tau_v^l := \tau_{\bar{v}}^l$ 
9:             go to line 4
10:          end if
11:        end if
12:      end for
13:    end for
14:  end for
15: end procedure

```

---

### IV. Network Diffusion PACKAGE

Keeping in mind the way we defined the multiprocess interaction framework, it is possible to move on to the package description. We also find it useful to present some high-level aspects of its architecture.

The most important functional requirements were paired with the questions posed in the introduction of sec. III, i.e. the possibility (1) to simulate any number of processes during the experiment and (2) to define interactions between them. We also wanted to support experiments on external data (the commonly used format to store multilayer networks - *.mpx* has been chosen) and generate data and figures reflecting changes of states in the network during experiments.

When it comes to nonfunctional requirements, the main was the programming language. Due to the popularity of particular

ones in data science, we faced an alternative: Python or R. The popularity and support of both languages are very similar, so in the end, the first one was chosen subjectively. It is worth noting that we were not focused on efficiency. Therefore, the implementation was reflecting alg. 1 without optimisation.

#### A. Implementation

The whole package is based on the NetworkX library since, as shown in sec. II-B, it has a lot of built-in functionalities, is widely used, and has great community support. This significantly reduces the so-called entry threshold of the package for new users. Following the theoretical concepts, we divide the code into three main components: *MultilayerNetwork*, *PropagationModel*, *MultiSpreading*.

The *MultilayerNetwork* was able to store multilayer networks (as a dictionary of *networkx.Graph* objects keyed by the layer names). We also added integrated I/O methods with the NetworkX library for *.mpx* files.

As for the *PropagationModel* class, the biggest problem to solve was to define the model unambiguously according to def. 3.2. We solved it by defining the following procedure of propagation model initialisation: (1) user passes processes to the model in the form of name and internal states for each process, (2) user inserts the default weight for transition, (3) user calls a compilation function (then the model is converted to an interpretable form for the simulation environment), (4) user manually changes the weights of particular transitions (usually those which are significant for the experiment). Regarding the problem of defining a data structure to keep the model in, during compilation, a unique graph with only possible transitions has been created for each process. This made easy referring to particular transitions in the model by operating strings of type *<state>.<process>*, similar to the SimInf library II-A.

The *MultiSpreading* was compacting the modules described already into a coherent whole. It had two fields *\_network* and *\_model*, which were associated with each other by process names (i.e. for each process in the model, there had to be a corresponding layer in the network with the same name). The other relevant methods were *set\_initial\_states* and *perform\_propagation*. The first one was used to set the initial conditions of the simulation - what percentage/number of nodes is currently in which state? The second method, on the other hand, was responsible for the simulation itself, according to the alg. 1. To separate the preprocessing of the experiment and the results from the simulation, a class *ExperimentLogger* has been prepared, whose object is returned after a successful experiment run.

#### V. EXAMPLE OF USAGE

Finally, a practical example can be presented. We prepared three comprehensive simulation scripts for this purpose: (1) example of epidemic propagation combined with vaccinations and awareness of the population, (2) marketing campaign of two competitive products, (3) gossip spreading on two different social networks. All are included in the GitHub

TABLE IV  
ARTEFACTS RELATED TO THE *Network Diffusion* PACKAGE.

| Artefact                       | Link  | Description                                       |
|--------------------------------|---|---|
| Main repository                | <a href="https://github.com/anty-filidor/network_diffusion">https://github.com/anty-filidor/network_diffusion</a>                   | Main repository with package's code [1]           |
| PyPi package                   | <a href="https://pypi.org/project/network-diffusion/">https://pypi.org/project/network-diffusion/</a>                               | Built package ready to download                   |
| Anaconda package               | <a href="https://anaconda.org/anty-filidor/network_diffusion">https://anaconda.org/anty-filidor/network_diffusion</a>               | Built package ready to download                   |
| Documentation                  | <a href="https://network-diffusion.readthedocs.io">https://network-diffusion.readthedocs.io</a>                                     | Web page with comprehensive documentation         |
| Examples (source code)         | <a href="https://github.com/anty-filidor/network_diffusion_examples">https://github.com/anty-filidor/network_diffusion_examples</a> | Auxiliary repository with examples                |
| Runnable capsule with examples | <a href="https://codeocean.com/capsule/8807709/tree/v4">https://codeocean.com/capsule/8807709/tree/v4</a>                           | Examples ready to run online via web browser [41] |

repository [1] and the Code Ocean capsule [41]. In this section, we present the first one (i.e. *epidemic.py*) in shortened form<sup>1</sup>. To avoid coping scripts, we will refer the reader to the corresponding lines of code in the repository/capsule.

#### A. Scenario to simulate

In this experiment, we will simulate the spreading of three processes: an illness affecting the network, awareness of its existence, and vaccination against it. Keeping in mind the notation introduced in tab. III we will first define a propagation model:

- *illness* :  $T = [s, i, r]$ ,  $W = [w_s^i, w_i^r]$ , where  $s$  denotes suspected node,  $i$  infected and  $r$  recovered,
- *awareness* :  $T = [n, a]$ ,  $W = [w_n^a]$ , where  $n$  means not-aware node and  $a$  respectively aware one,
- *vaccination* :  $T = [u, v]$ ,  $W = [w_u^v]$ , where  $u$  means not vaccinated node and  $v$  vaccinated one.

The compiled model can be represented by a three-dimensional grid with twelve points. Regarding the fact that we defined only a subset of all possible transitions (e.g.,  $w_i^s$  was discarded), we have to define "only" twenty ones:

- for states varying in the *illness*:  $w_{snu}^{inu}, w_{inu}^{rnu}, w_{sau}^{iau}, w_{iau}^{rau}, w_{snv}^{inv}, w_{inv}^{rvv}, w_{sav}^{iav}, w_{iav}^{rav}$ ,
- for states varying in the *awareness*:  $w_{snu}^{sau}, w_{inu}^{iau}, w_{rnu}^{rau}, w_{snv}^{sav}, w_{inv}^{iav}, w_{rvv}^{rav}$ ,
- for states varying in the *vaccination*:  $w_{sau}^{sav}, w_{iau}^{iav}, w_{rau}^{rav}, w_{snu}^{snv}, w_{inu}^{inv}, w_{rnu}^{rvv}$ ,

Let us also add more constraints to the experiment: we want both vaccination and awareness to reduce the probability of getting sick, and awareness of the disease should make the nodes more willing to get vaccinated. We can obtain such an effect by setting some of the transitions listed above to predefined values. Moreover, we note that some of them

<sup>1</sup>It is worth noting that the remaining two examples utilise other mechanisms of phenomena spreading. They are all in the form of the Jupyter Notebook, and we welcome readers to investigate them.



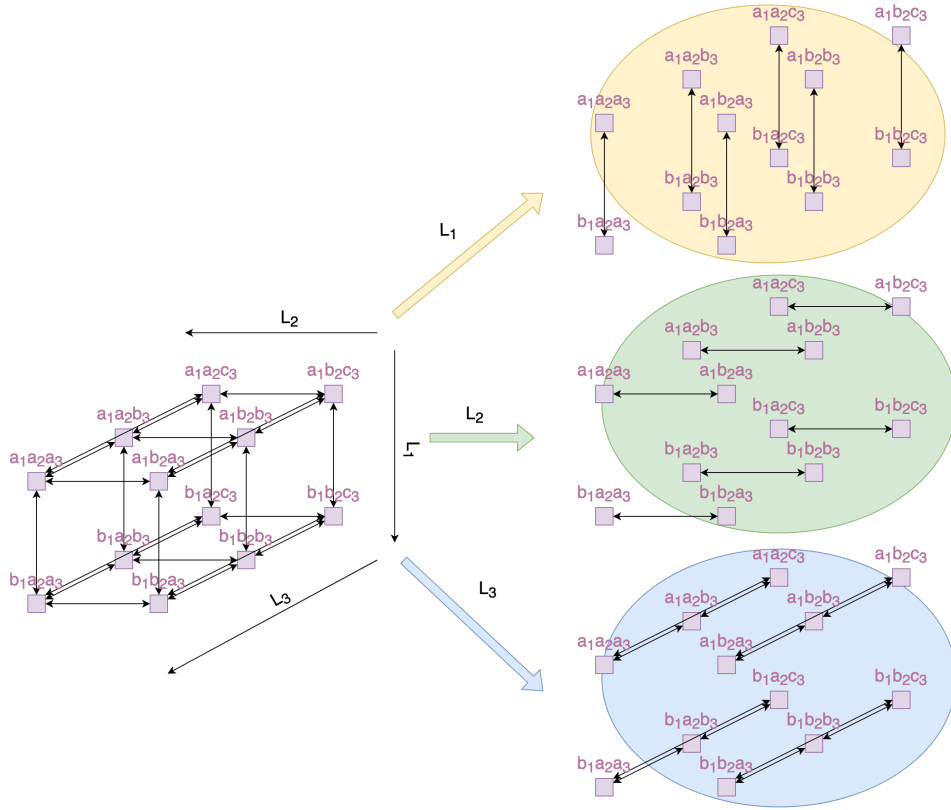


Fig. 3. Scheme for compiling a propagation model. A single, three-dimensional model is stored as a structure of three separate NetworkX graphs, one for each process, with edges covering allowed only transitions.

seem illogical. For example,  $w_{snu}^{snv}$  one, because it is hard to imagine that an individual can be vaccinated without even being aware of the disease. Hence, in our model, we can limit the transitions to the following set of non-zero ones:

- for states varying in *illness*:  $w_{snu}^{inu} = 0.4$ ,  $w_{inu}^{rnu} = 0.1$ ,  $w_{sau}^{iau} = 0.2$ ,  $w_{iau}^{rau} = 0.3$ ,  $w_{sav}^{iav} = 0.05$ ,  $w_{iav}^{rav} = 0.7$ ,
- for states varying in *awareness*:  $w_{snu}^{sau} = 0.05$ ,  $w_{inu}^{iau} = 0.2$ ,  $w_{snv}^{sav} = 1$ ,
- for states varying in *vaccination*:  $w_{sau}^{sav} = 0.03$ ,  $w_{iau}^{iav} = 0.1$ .

On the other hand, in rare cases, situations like  $snu \rightarrow snv$  can be possible. Therefore, we will set up some tiny weights (0.005) for all possible but unlikely to happen transitions.

#### B. Defining a propagation model

With these assumptions, we can finally prepare the script. As a first step, we have to import all required packages (see lines 2:16). After that, we can step forward to define a propagation model (see lines 29:48). Please pay attention to the function `compile`. When the keyword argument is set, all theoretically possible transitions in the model will have a weight assigned to 0.005. With this operation, we can increase the model's flexibility to include some edge cases.

#### C. Defining a network

The next step is to define the network. As mentioned in sec. III-D, it should have as many layers as the processes

spreading within it. We can achieve this by duplicating a flat NetworkX graph into three layers. In this experiment, we will use a predefined graph reflecting interactions in “Les Misérables” (see lines 50:52).

#### D. Defining and running an experiment

After that, we can define and start the experiment (see lines 54:99). We set the initial state of the “*ill*” process to have 65 nodes “*s*”, 10 “*i*” and 2 “*r*” at the first epoch. Here, we did it manually for particular nodes to obtain reproducibility, but *Network Diffusion* contains another function that does it for randomly selected nodes at once: `nx.Experiment.set_initial_states`.

#### E. Results and discussion

Once the simulation is completed, an `ExperimentLogger` object is returned. We can use it to prepare a report (see line 99). As a result, we obtain (snippets available in documentation - see tab. IV): propagation report for each process as set of .csv files containing information about the distributions of particular states in each epoch (*awar\_propagation.csv*, *ill\_propagation.csv*, *vacc\_propagation.csv*), information about the network used for experiment (*network\_report.txt*), the model (*model\_report.txt*) and visualisation of the experiment (fig. 4a).

As we can see in fig. 4a, interactions between processes reflect the assumed scenario, the more individuals vaccinated



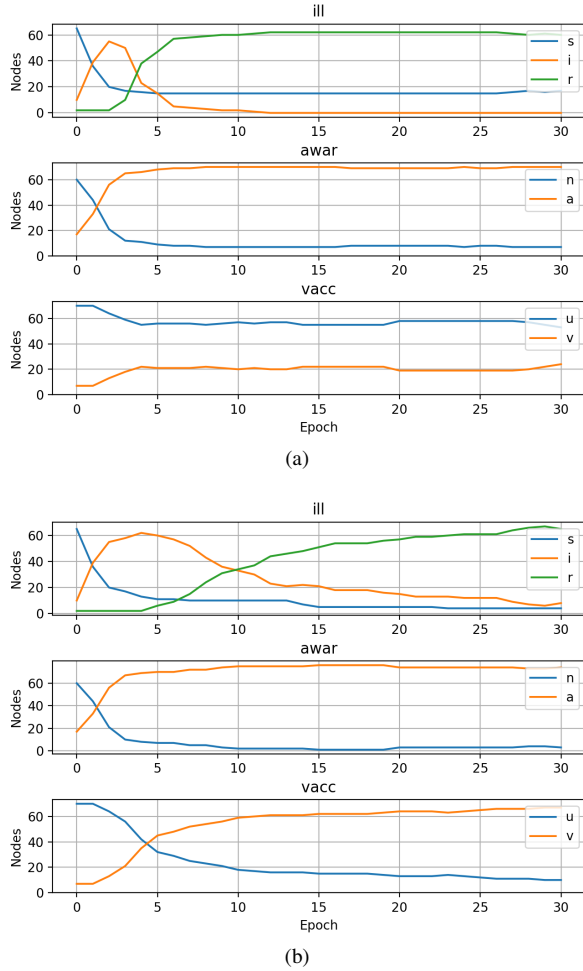


Fig. 4. Visualisations of experiments - dynamics of processes in the network (a) according to presented assumptions, (b) with modified *illness* to make it more contagious.

and aware, the less increase of ill nodes in the network. On the other hand, in fig. 4b, we can see a variation of the experiment. In order to obtain the effect of more malicious illnesses that cannot be so easily mitigated with vaccines and awareness of the population, we have modified weights  $w_{inu}^{rnu}$ ,  $w_{iau}^{rau}$  and  $w_{iav}^{rav}$ . That resulted in a slower falling curve for state  $i$  that allowed the *vaccination* process to continue spreading among the network.

## VI. CONCLUSIONS

In this paper, we have introduced a computational approach to simulate the spreading of multiple processes in networks. Keeping in mind the analysis of the existing tools from sec. II, we find it more valuable since it fulfils the observed gap in the domain. Despite the basic character of *Network Diffusion*, we hope that it will help other researchers in analysing and understanding spreading processes in networks.

We would like to mention that there is great potential in the further development of the package. As we described in sec. III, for now, we are limited to a fairly simple schema of the experiment. We plan to extend it in the next releases

by aspects of time dynamics, such as adding layer switching cost for the spreading process [42]. We also noticed a need to provide the user with the possibility to customise a function that changes the states of nodes. Currently, we have a fairly simple mechanism that depends on iteration through neighbours combined with the Bernoulli test. However, we can use another approach, e.g. that takes into account all neighbours of the node at once. Another important aspect is an enhancement of the package's computational capabilities (see sec. A for details). Finally, the concept that is currently investigated by the authors is a development of a training mechanism that sets up weights of the models in order to fit with real data, which allows for predicting further spreading of the phenomena.

Having written all that, we can say that *Network Diffusion* may help to address the problem of multiple spreading processes within the networks. This issue is certainly real, and the COVID-19 crisis shows that we need to be able to simulate such phenomena by taking into account coexisting phenomena. As we demonstrated in the article, the package has a fine theoretical background and examples proving its usability. We strongly believe that our concept can be helpful and become an inspiration to other projects.

## REFERENCES

- [1] M. Czuba and P. Bródka, "Network diffusion," 2021. [Online]. Available: [https://github.com/anty-filidor/network\\_diffusion](https://github.com/anty-filidor/network_diffusion)
- [2] P. Bródka, K. Musial, and J. Jankowski, "Interacting spreading processes in multilayer networks: a systematic review," *IEEE Access*, vol. 8, pp. 10 316–10 341, 2020.
- [3] A.-L. Barabási and M. Pósfai, *Network Science*. Cambridge: Cambridge University Press, 2016, ch. 10.1, pp. –.
- [4] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers, "Biogrid: A general repository for interaction datasets," *Nucleic acids research*, vol. 34, pp. D535–D539, 2006.
- [5] M. De Domenico, A. Solé-Ribalta, S. Gómez, and A. Arenas, "Navigability of interconnected networks under random failures," *National Academy of Sciences*, vol. 111, no. 23, p. 8351, 2014.
- [6] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.
- [7] P. Kazienko, K. Musial, E. Kukla, T. Kajdanowicz, and P. Bródka, "Multidimensional social network: Model and analysis," in *International Conference on Computational Collective Intelligence*. Springer, 2011, pp. 378–387.
- [8] P. Bródka, A. Chmiel, M. Magnani, and G. Ragozini, "Quantifying layer similarity in multiplex networks: A systematic study," *Royal Society open science*, vol. 5, no. 8, p. 171747, 2018.
- [9] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter, "Multilayer networks," *Journal of complex networks*, vol. 2, no. 3, pp. 203–271, 2014.
- [10] M. Magnani and L. Rossi, "The ml-model for multi-layer social networks," in *2011 International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 2011, pp. 5–12.
- [11] A. Guille, H. Hacid, C. Favre, and D. A. Zighed, "Information diffusion in online social networks: A survey," *ACM Sigmod Record*, vol. 42, no. 2, pp. 17–28, 2013.
- [12] W. O. Kermack and A. G. McKendrick, "A contribution to the mathematical theory of epidemics," *royal society of london. Series A, Containing papers of a mathematical and physical character*, vol. 115, no. 772, pp. 700–721, 1927.
- [13] R. A. Holley and T. M. Liggett, "Ergodic theorems for weakly interacting infinite systems and the voter model," *The annals of probability*, pp. 643–663, 1975.
- [14] F. M. Bass, "A new product growth for model consumer durables," *Management science*, vol. 15, no. 5, pp. 215–227, 1969.
- [15] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *9th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, p. 137.

[16] H. Zang, "The effects of global awareness on the spreading of epidemics in multiplex networks," *Physica A: Statistical Mechanics and its Applications*, vol. 492, pp. 1495–1506, 2018.

[17] M. Scatà, A. Di Stefano, P. Liò, and A. La Corte, "The impact of heterogeneity and awareness in modeling epidemic spreading on multiplex networks," *Scientific reports*, vol. 6, no. 1, 2016.

[18] P. Shakarian, A. Bhatnagar, A. Aleali, E. Shaabani, and R. Guo, *Diffusion in Social Networks*. Springer: Springer, 2016, ch. 4.

[19] W. V. d. Broeck, C. Giannini, B. Gonçalves, M. Quaggiotto, V. Colizza, and A. Vespignani, "The gleamviz computational tool, a publicly available software to explore realistic epidemic spreading scenarios at the global scale," *BMC Infectious Diseases*, vol. 11, no. 1, p. 37, Feb 2011. [Online]. Available: <https://doi.org/10.1186/1471-2334-11-37>

[20] G. Rossetti, L. Milli, S. Rinzivillo, A. Sirbu, D. Pedreschi, and F. Giannotti, "Ndlb: A python library to model and analyze diffusion processes over complex networks," *Int. J. Data Sci. Anal.*, vol. 5, no. 1, pp. 61–79, Feb 2018.

[21] S. Widgren, P. Bauer, R. Eriksson, and S. Engblom, "SimInf: An R package for data-driven stochastic disease spread simulations," *Journal of Statistical Software*, vol. 91, no. 12, pp. 1–42, 2019.

[22] F. Alvarez, P. Crépey, M. Barthelemy, and A.-J. Valleron, "Sispread: A software to simulate infectious diseases spreading on contact networks," *Methods of information in medicine*, vol. 46, p. 19, 2007.

[23] J. Douglas, S. Bianco, S. Edlund, T. Engelhardt, M. Filter, T. Günther, M. HuKun, E. Nixon, N. Sevilla, A. Swaid, and J. Kaufman, "Stem: An open source tool for disease modeling," *Health security*, sie 2019.

[24] Franz Inc., "Allegrograph website," [26 May 2022]. [Online]. Available: <https://allegrograph.com/products/allegrograph>

[25] E. Inc., "Arcgis pro - 2d and 3d gis mapping software," [26 May 2022]. [Online]. Available: <https://www.esri.com/en-us/arcgis/products/arcgis-pro/overview>

[26] Cytoscape Consortium, "Cytoscape 3.8.2 user manual," [26 May 2022]. [Online]. Available: <https://manual.cytoscape.org/en/stable>

[27] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," *International AAAI Conference on Weblogs and Social Media*, 2009.

[28] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 135–146. [Online]. Available: <https://doi.org/10.1145/1807167.1807184>

[29] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, *Graphviz and Dynagraph — Static and Dynamic Graph Drawing Tools*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, ch. -, pp. 127–148.

[30] G. Csardi and T. Nepusz, "The igraph Software Package for Complex Network Research," *InterJournal*, vol. Complex Systems, 2006. [Online]. Available: <https://igraph.org>

[31] M. Matteo, V. Davide, and D. Mikael, *Multinet: Analysis and Mining of Multilayer Social Networks*, 2020. [Online]. Available: <https://cran.r-project.org/web/packages/multinet/index.html>

[32] M. De Domenico, M. A. Porter, and A. Arenas, "Muxviz: A tool for multilayer analysis and visualization of networks," *Journal of Complex Networks*, vol. 3, no. 2, pp. 159–176, 10 2014.

[33] Neo4j, Inc., "Neo4j graph platform the leader in graph databases," [26 May 2022]. [Online]. Available: <https://neo4j.com>

[34] G.-H. Ghim, N. Cho, and J. Seo, *NetMiner*. New York, NY: Springer New York, 2014, pp. 1025–1037.

[35] C. T. Butts, *Network: Classes for Relational Data*, The Statnet Project (<http://www.statnet.org>), 2020, r package version 1.16.1. [Online]. Available: <https://CRAN.R-project.org/package=network>

[36] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[37] M. Smith, A. Ceni, N. Milic-Frayling, B. Shneiderman, E. M. Rodrigues, J. Leskovec, and C. Dunne, *NodeXL: A Free and Open Network Overview, Discovery and Exploration Add-in for Excel 2007/2010/2013/2016*, 2010. [Online]. Available: <http://nodexl.codeplex.com/>

[38] D. Auber, D. Archambault, R. Bourqui, M. Delest, J. Dubois, A. Lambert, P. Mary, M. Mathiaut, G. Melançon, B. Pinaud, B. Renoust, Mikko Kivela, "Multilayer networks library for python (pymnet) documentation," Aug 2017, [26 May 2022], <http://www.mkivela.com/pymnet>.

and J. Vallet, "TULIP 5," in *Encyclopedia of Social Network Analysis and Mining*, R. Alhajj and J. Rokne, Eds. Springer, Aug. 2017, pp. 1–28. [Online]. Available: <https://hal.archives-ouvertes.fr/hal-01654518>

[40] yWorks GmbH, "yed graph editor," [26 May 2022]. [Online]. Available: <https://www.yworks.com/products/yed>

[41] M. Czuba and P. Bródka, "Demo-code for the paper "network diffusion: A package to simulate spreading of multiple interacting processes in complex networks"," 11 2021. [Online]. Available: <https://codeocean.com/capsule/8807709/tree/v3>

[42] B. Min, S.-H. Gwak, N. Lee, and K.-I. Goh, "Layer-switching cost and optimality in information spreading on multiplex networks," *Scientific reports*, vol. 6, no. 1, pp. 1–12, 2016.

## APPENDIX

### A. Computational efficiency of the package

Since *Network Diffusion* has been considered a prototype framework, we have decided to investigate its time efficiency compared to NDlib [20]. We performed an experiment on a single-layer SIR model (due to limitations of the NDlib, we could not test a more complex case) spreading within the Erdos-Renyi graph, the size of which varied. For each network size, both implementations were executed 10 times (to mitigate the side effects that burden the processor). Each call to the function was measured in milliseconds. The results are presented in fig. 5. The time efficiency of Network Diffusion is a field for improvement, especially for large networks.

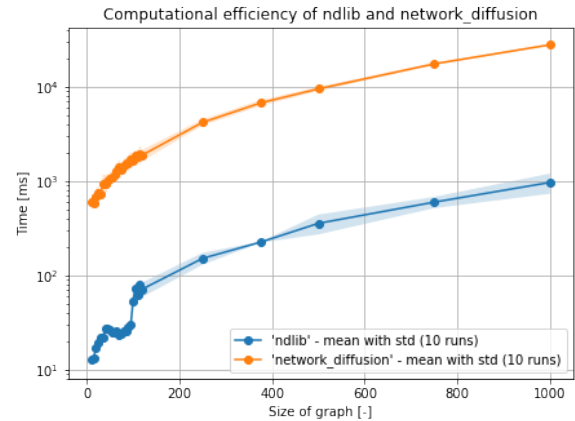


Fig. 5. Comparison of the efficiency of the Network Diffusion and the NDlib.

### B. Running attached code examples

There are two ways to run the examples: (1) local execution, (2) via the Code Ocean capsule [41]. Despite the differences in code handling, both are expected to produce the same results. To follow the first, the user needs to have Python installed. It is recommended to create a virtual environment based on the *requirements.txt* file. Then, to run the chosen example, the corresponding script needs to be executed. The second method requires far less effort. To run our code, we recommend first that you duplicate the capsule into a self-owned account. Then, to run the engine, the "reproducible run" button has to be pressed. To run a Jupyter notebook, the user has to select a Jupyter icon from the list below the mentioned button.