NDLIB: Studying Network Diffusion Dynamics

Giulio Rossetti*†, Letizia Milli*†, Salvatore Rinzivillo†, Alina Sirbu*, Dino Pedreschi* and Fosca Giannotti†
* University of Pisa,
Largo Bruno Pontecorvo, 2 Pisa, Italy
Email: name.surname@di.unipi.it
† KDD Lab. ISTI-CNR,
via G. Moruzzi, 1 Pisa, Italy
Email:name.surname@isti.cnr.it

Abstract—**Nowadays the analysis of diffusive phenomena occurring on top of complex networks represents a hot topic in the Social Network Analysis playground. In order to support students, teachers, developers and researchers in this work we introduce a novel simulation framework, NDLIB. NDLIB is designed to be a multi-level ecosystem that can be fruitfully used by different user segments. Upon the diffusion library, we designed a simulation server that allows remote execution of experiments and an online visualization tool that abstract the programmatic interface and makes available the simulation platform to non-technicians.**

I. INTRODUCTION

In the last decades Social Network Analysis, henceforth SNA, has received increasing attention from several, heterogeneous fields of research. Such popularity was certainly due to the flexibility offered by graph theory: a powerful tool that allows reducing countless phenomena to a common analytical framework whose basic bricks are nodes and their relations. Indeed, social relationships, trading, transportation and communication infrastructures, even the brain can be modeled as networks and, as such, analyzed. Undoubtedly, such pervasiveness has produced an amplification in the visibility of network analysis studies thus making this complex and interesting field one of the most widespread among higher education centers, universities and academies. Given the exponential diffusion reached by SNA, several tools were developed in order to make it approachable by the wider audience possible. SNA programming libraries are nowadays available to computer scientists, physicists as well as mathematicians; moreover, graphical tools were developed for social scientists, biologists as well as for educational purposes.

Although being a very active field of research per se, SNA is often used as a tool to analyze complex phenomena such as spreading of epidemic and diffusion of opinions, ideas, innovations. Even for such peculiar applications, we have witnessed during the last years the appearance of dedicated tools and libraries: however, the plethora of resources available often discourage the final users making hard and time-consuming the identification of the right tool for the specific task and level of expertise.

To cope with such issue, in this work we introduce a novel framework able to model, simulate and study diffusive phenomena that unfold on complex networks. NDLIB represents a multi-level solution: it is designed to offer a programmatic interface to developers, an experimental server to those centres that need to offer simulations as a service and, finally a visual interface for those, students as well as non-technicians, who want to run simulations and experiments but don't have the time to learn a new library or programming language.

The paper is organized as follows: in Section II we briefly introduce the network diffusion playground in order to make clear which are the phenomena that our NDLIB is designed to analyze; in Section III we review some of the main tools nowadays available to study and visualize diffusion simulations. In Section IV we introduce NDLIB: there we describe how the library is designed, how to use it and extend it. Moreover, we introduce NDLIB-REST and NDLIB-Viz: the former being a service designed to offer remote simulation facilities, the latter a web-based visual platform that abstracts from the coding complexity and allows the end user to setup, run and analyze diffusion experiments without writing any line of code. Finally in Section V we conclude the paper underlining the advantages of NDLIB w.r.t. its competitors and providing insights on the future evolution of our framework. Appendix A briefly describes the models made available by NDLIB.

II. NETWORK DIFFUSION

The analysis of diffusive phenomena that unfold on top of complex networks is a task able to attract growing interests from multiple fields of research. In order to provide a succinct framing of such complex and extensively studied problem it is possible to split the related literature into two broad, related, sub-classes: *Epidemics and Spreading* and *Opinion Dynamics*.

A. Epidemics and Spreading

When we talk about epidemics, we think about contagious diseases caused by biological pathogens, like influenza, measles, chickenpox and sexually transmitted viruses that spread from person to person. However, other phenomena can be linked to the concept of epidemic: think about the spread of computer virus [1] where the agent is a malware that can transmit a copy of itself from computer to computer, or the spread of mobile phone virus [2], [3], or the diffusion of knowledge, innovations, products in an online social network [4], the so-called "social contagion", where people are making decision to adopt a new idea or innovation. Several elements determine the patterns by which epidemics spread through groups of people: the properties carried by the pathogen (its contagiousness, the length of its infectious period and its severity), the structure of the network as well as the mobility patterns of the people involved. Although often treated as similar processes, diffusion of information and epidemic spreading can be easily distinguished by a single feature: the degree of *activeness* of the subjects they affect.

Indeed, the spreading process of a virus does not require an *active* participation of the people that catch it (i.e., even though some behaviors acts as contagion facilitators – scarce hygiene, moist and crowded environment – we can assume that no one chooses to get the flu on purpose); conversely, we can argue that the diffusion of an idea, an innovation, or a trend strictly depend not only on the social pressure but also by individual choices.

B. Opinion Dynamics

A different field related with modeling social behavior is that of opinion dynamics. Recent years have witnessed the introduction of a wide range of models that attempt to explain how opinions form in a population [5], taking into account various social theories (e.g. bounded confidence [6] or social impact [7]). These models have a lot in common with those seen in epidemics and spreading. In general, individuals are modeled as agents with a state and connected by a social network. The social links can be represented by a complete graph (mean field models) or by more realistic complex networks, similar to epidemics and spread. The state is typically represented by variables that can be discrete (similar to the case of spreading) but also continuous, representing, for instance, a probability of choosing one option or another [8]. The state of individuals changes in time, based on a set of update rules, mainly through interaction with the neighbors. While in many spreading and epidemics models this change is irreversible (susceptible to infected), in opinion dynamics the state can oscillate freely between the possible values, simulating thus how opinions change in reality. A different important aspect in opinion dynamics is external information, which can be interpreted as the effect of mass media. In general, external information is represented as a static individual with whom all others can interact, again present also in spreading models.

Hence, it is clear that the two model categories have enough in common to be implemented under a common framework, which is why we introduced both in our framework.

III. Complex Network Analysis Tools

When it comes to model and study complex networks and diffusive phenomena several resources are available to students, programmers and researchers. In this section we propose a review of the most used libraries for complex network analysis (III-A), visual tools (III-B) and simulators (III-C). Indeed, our review will not cover all the existing resources but only the ones that, in our opinion, provide interesting facilities to the end user at a reasonable learning cost.

A. Libraries

Nowadays, two languages among the others are widely considered the main players in the data science world: Python and R. Since SNA has acquired increasing importance in the data science community in the last years, we decided to focus our attention on libraries developed for such languages.

Python. The most famous, pure Python package, that provides graph data structures along with algorithms, synthetic generators and drawing tools is for sure *NetworkX*[1][9].

Upon such general graph modeling framework is built the *Nepidemix*[2] library: a suite tailored to programmatically describe simulation of complex processes on networks [10]. *Nepidemix* was developed by members of the IMPACT-HIV group; it is written in Python 2 and uses the module NetworkX to manage the network structure. It automates common diffusion simulation steps allowing the programmer to build a network according to some specifics and to run on top of it a set of epidemic processes for a specified number of iterations. Moreover, Nepidemix allows during simulation to save incremental results such as disease prevalence and state transitions. Another Python library dedicated to the simulation of diffusive models is *EoN*[3]. EoN is designed to study the spread of SIS and SIR diseases in networks [11]. It is composed of two sets of algorithms: the first set that deals with simulation of epidemics on networks (SIR and SIS) and the second designed to provide solutions of systems of equations. Also, this package is built on top of NetworkX graph structures.

R. One of the main library designed to handle, manipulate and analyze graph structures in R is *Igraph*[4][12]. Igraph is written in C and is released as Python and R packages. It can handle large graphs very well and provides functions for generating random and regular graphs, graph visualization, centrality analysis, path length and much more.

When it comes to simulating epidemic models in R one of the most famous package is undoubtedly *EpiModel*[5][13]. EpiModel provides facilities for build, solve, and plot mathematical models of infectious disease. It currently provides functionality for three classes of epidemic models – Deterministic Compartmental Models, Stochastic Individual Contact Models and Stochastic Network Models – and three types of infectious disease can be simulated upon them: SI, SIR, SIS. EpiModel allows generating visual summaries for the execution of epidemic models; it provides plotting facilities to show the means and standard deviations across multiple simulations

[1]NetworkX: https://networkx.github.io
[2]Nepidemix: http://nepidemix.irmacs.sfu.ca/
[3]https://github.com/springer-math/Mathematics-of-Epidemics-on-Networks
[4]Igraph: http://igraph.org/redirect.html
[5]EpiModel: http://www.epimodel.org/

while varying the initial infection status. It also includes a web-based visual application for simulating epidemics[6].

B. Network Analysis Visual Tools

Network analysis is nowadays adopted as an analytical tool by several, even multidisciplinary, fields. In order to simplify the approach to such discipline, several graphical tools have been designed: some of them have become widespread platforms adopted for both small scale analysis and educational purposes. Among them, we report two open source examples: *Cytoscape* and *Gephi*.

Cytoscape[7] is one of the first open source bioinformatic software platform born to visualize molecular interaction networks and biological pathways, integrating these networks with annotations, gene expression profiles and other state data [14]. Although Cytoscape was originally designed for biological research, it is extensively used to visualize and analyze graphs of any kind: additional features (i.e. community extraction) are made available through a plugin system thus making the platform easily extensible.

Gephi[8] is an open source platform for network analysis written in Java on top of the NetBeans evironment [15]. It employs a 3D render engine to display large networks in real-time and to speed up the exploration. Gephi provides easy access to a broad collection of network datasets and provides support for spatializing, filtering, navigating, manipulating and clustering graph entities.

C. Epidemics Simulators

The visual tools previously introduced provide general network analysis facilities but are not designed to support the simulation of spreading phenomena. Here we review some visual systems specifically designed for such task.

NetLogo[9] is a programmable modelling environment for simulating natural and social phenomena. It was developed by Uri Wilensky in 1999 [16] and has been in continuous development ever since at the "Center for Connected Learning and Computer-Based Modeling". It is particularly well suited for modeling complex systems that evolve over time describing them as agent-based processes. NetLogo enables users to run a predefined set of simulations and play with their parameters, exploring their behaviors under various conditions. NetLogo runs on the Java virtual machine.

GLEaMviz[10] is a publicly available software that simulates the spread of emerging human-to-human infectious diseases on world scale [17]. The GLEaMviz framework is composed of three components: the client application, the proxy middleware, and the simulation engine. The simulations it defines combine real-world data on populations and human mobility with elaborate stochastic models of disease transmission to simulate disease spread on the global scale. As output, it provides a dynamic map and several charts describing the geo-temporal evolution of the disease.

System Sciences[11] is an online project created by the "Institute of Systems Sciences, Innovation and Sustainability Research" at the University of Graz whose aim is to design an interactive electronic textbook for systems sciences based on software applications for tablet computers. In the disease spreading section offered by this tool, the user can choose a network from a set of classical network models (random, small world, scale free and complete network) and then fix the parameter of the SIR model (the only one implemented so far).

FRED[12] (A Framework for Reconstructing Epidemiological Dynamics) is an open source modeling system developed by the "Pitt Public Health Dynamics Laboratory" in collaboration with the "Pittsburgh Supercomputing Center and the School of Computer Science" at Carnegie Mellon University [18]. FRED supports research on the dynamics of infectious disease epidemics and the interacting effects of mitigation strategies, viral evolution and personal health behavior. The system uses agent-based modeling based on census synthetic populations data that capture the demographic and geographic distributions of the population. FRED epidemic models are currently available for every state and country in the United States, and for selected international locations.

IV. NDLIB ECOSYSTEM

Since the analysis of diffusive phenomena represents an hot topic for a number of communities having different backgrounds, we designed our framework so that it can be fruitfully used by the widest user segment possible. To do so we organized it in three incremental modules: the NDLIB core library (written in Python), a REST service accessible through API calls and, finally, a dynamic visual interface.

In this section we will describe and discuss the major characteristics of each of such components (as implemented in NDLIB v2.0.1), paying attention to underline the rationale behind the implementation choices made and their repercussions on the framework usability.

A. NDLIB*: Network Diffusion Library*

At the core of our tool there is NDLIB, whose name stands for *"(N)etwork (D)iffusion Library"*, a Python package built upon the network facilities offered by NetworkX. The library, available for Python 2.7.x and 3.x, is currently hosted on GitHub[13] , on pypi[14] and has its online documentation on ReadTheDocs[15]. A complete list of the diffusion models

[6]EpiModel viz: https://statnet.shinyapps.io/epinet/
[7]Cytoscape: http://www.cytoscape.org/
[8]Gephi: https://gephi.org/
[9]NetLogo: https://ccl.northwestern.edu/netlogo
[10]GLEaMviz: http://www.gleamviz.org/
[11]System Sciences: http://systems-sciences.uni-graz.at/etextbook/
[12]FRED: http://fred.publichealth.pitt.edu/measles
[13]NDLIB GitHub: https://github.com/GiulioRossetti/ndlib
[14]NDLIB pypi: https://pypi.python.org/pypi/ndlib
[15]NDLIB docs: http://ndlib.readthedocs.io

implemented in NDLIB v2.0.1 – 8 from the epidemics and 5 from the opinion dynamics categories –, along with their short descriptions, is reported in Appendix A. Indeed, NDLIB is intended for developers and all those users that have basilar knowledge of Python programming that want to run simulation experiments on their own machine.

NDLIB Rationale. NDLIB models diffusive phenomena as a discrete-time agent-based processes: given a network $G = (V, E)$ and its actual state S_i, the request of a diffusion iteration will return a novel state S_{i+1} obtained by applying the model evolution rules to all the nodes in V.

The library decomposes the diffusion process into three components: (i) the graph topology on which the process take place; (ii) the specific diffusion model to simulate; (iii) the configuration of the model and infection initial state.

The first component, the graph topology, is borrowed by the available entities exposed by the NetworkX library: indeed, the implementation of all NDLIB models is agnostic w.r.t. the directedness of the graph, thus allowing the user to use both undirected as well as directed networks in his simulations.

The second component, the model selection, is designed so to expose to the final user a minimal and coherent interface: all the models extend a generic template that takes care of handling the initialization phase and to expose step-by-step simulation facilities.

Finally, the third component, the simulation initialization interface, allows the user to fully specify three different classes of information:

(i) model specific parameters (e.g. the β parameter for the *SI* model);

(ii) nodes' and edges' attributes (e.g. node/edge-wise thresholds for the *Threshold/Independent Cascade* models);

(iii) the initial state of the epidemic (e.g. the percentage of nodes in each status - randomly chosen - or a planted initial configuration of node statuses).

The configuration object plays a fundamental role in the library logic: it acts as the focus of experiment description thus making the simulation definition and invocation coherent across all the exposed models.

The following code shows an example of experiment definition, configuration and execution.

```
1  import networkx as nx
2  import ndlib.models.ModelConfig as mc
3  import ndlib.models.epidemics.SIRModel
4                                          as sir
5
6  # Network topology
7  g = nx.erdos_renyi_graph(1000, 0.1)
8
9  # Model selection
10 model = sir.SIRModel(g)
11
12 # Model Configuration
13 cfg = mc.Configuration()
14 cfg.add_model_parameter('beta', 0.001)
15 cfg.add_model_parameter('gamma', 0.01)
16 cfg.add_model_parameter("percentage_infected",
17                                   0.05)
18 model.set_initial_status(cfg)
19
20 # Simulation execution
21 iterations = model.iteration_bunch(200)
```

In lines 1-4 are imported all the required modules; in line 7 an Erdös Renyi graph `g` is built using a NetworkX generator; in line 10 the *SIR* `model` is *attached* to the graph `g`; in lines 13-18 the `model` initial status is configured; finally, line 21 shows how 200 iterations of the simulation can be obtained by the invocation of the `model.iteration_bunch(bunch_size=n)` method (where $n = 200$ identifies the number of desired iterations). An alternative to the iteration bunch simulation request is offered by the step-by-step `model.iteration()` method, a call that returns as output a single simulation iteration. The status returned by both `model.iteration()` and `model.iteration_bunch(bunch_size=n)` is incremental: each iteration describes the configurations of those nodes that changed their status from the previous model iteration.

In order to allow the final user to easily analyze the behavior of a simulation NDLIB exposes a set of visual facilities. By exploiting the Bokeh[16] library, NDLIB defines a visualization package `ndlib.viz.bokeh`. Among the facility offered by such package, the `DiffusionTrend` object that takes care of generating trend line plots starting from the configured `model` and computed iterations.

```
22 from bokeh.io import show
23 from ndlib.viz.bokeh.DiffusionTrend import
24         DiffusionTrend
25
26 viz = DiffusionTrend(model, iterations)
27 p = viz.plot(width=400, height=400)
28 show(p)
```

In addition, within the same package is also made available a `DiffusionPrevalence` plot:

```
29 from bokeh.io import show
30 from ndlib.viz.bokeh.DiffusionPrevalence
31         import DiffusionPrevalence
32
33 viz = DiffusionPrevalence(model, iterations)
34 p1 = viz.plot(width=400, height=400)
35 show(p)
```

The `plot()` method, implemented by both classes, uses the model configuration and iteration results to generate plots as the ones in Figure 1: labels, headings, parameter values are retrieved by the metadata within the configured `model` object thus making the plot template agnostic w.r.t. the specific model used during the simulation. In particular the `DiffusionTrend` plot shows the temporal trend for the node statuses specified by the model while the `DiffusionPrevalence` one underline their variation across consecutive iterations (e.g. the delta of the number of nodes that have a status s w.r.t. consecutive iterations).

[16]Bokeh: http://bokeh.pydata.org

Moreover, in order to better compare runs of different models (as well as different configurations of the same model) on the same graph the `ndlib.viz.bokeh` package exposes a multiplot facility with grid auto-layout as shown in the following snippet:

```python
from ndlib.viz.bokeh.MultiPlot
     import MultiPlot

vm = MultiPlot()

# Add the generated plots
vm.add_plot(p)
vm.add_plot(p1)

# Visualize them
multi = vm.plot()
```

Extend NDLIB. As discussed before, all the diffusion models implemented in NDLIB extend the same template defined by `ndlib.models.DiffusionModel`. The `DiffusionModel` abstract class takes care of: (i) validate the coherence and completeness of the `ModelConfig` object used to instantiate the simulation; (ii) initialize the simulation; (iii) define a common interface for parameter passing and model execution. Extending the NDLIB library is easy: a novel model, identified by a Python class that extends `DiffusionModel`, must implement two methods: (i) its constuctor (e.g. `__init__(self, graph)`), and (ii) the iteration step (e.g. `iteration()`). The `__init__` method is used to provide a meta description for the model parameters (both global and node/edge specific) as well as for the node statuses. Such meta-data, whose example is reported in the code below, has two roles: it allows `DiffusionModel` to check the consistency of the model configuration and enables the `VisualizeDiffusion` object to customize the simulation visualizations.

```python
from ndlib.models.DiffusionModel
     import DiffusionModel

class MyModel(DiffusionModel):
  def __init__(self, graph):
    super(self.__class__, self)
          .__init__(graph)

    # Method name
    self.name = "MyModel"

    # Available node statuses
    self.available_statuses = {
        "Susceptible": 0,
        "Infected": 1
    }

    # Exposed Parameters
    self.parameters = {
        "model":
          "name": {
             "descr": "Infection Rate"
             "range": [0,1],
             "optional": False
           },
          },
          "nodes": {},
          "edges": {},
        }
```

The core of each diffusion model is thus defined in its `iteration()` method. As discussed before the entire diffusion process can be seen as an agent-based discrete-time simulation: the `iteration()` method describe the rules that decide for each *agent* (i.e. a node), during a round of simulation, if it will maintain its status or change it.

As shown in the example below, the iteration step is composed of three stages: (i) collection of the actual nodes' statuses, (ii) update cycle over the nodes, and (iii) computation of the incremental updates. Note that the first step is mandatory since we consider each iteration as atomic and we expect a synchronous updates of all the nodes (e.g. the model rules must be applied to all the agents starting from the same initial configuration).

```python
    def iteration(self):
      # Set initial node statuses
      actual_status = {node: nstatus for node,
        nstatus in self.status.iteritems()}

      # first iteration
      if self.actual_iteration == 0:
        self.actual_iteration += 1
        return 0, actual_status

      # iteration inner loop
      for u in self.graph.nodes():
        # Iteration updates

      # Incremental result
      delta = self.status_delta(actual_status)
      self.status = actual_status
      self.actual_iteration += 1

      return self.actual_iteration - 1, delta
```

B. NDLIB-*REST: simulation web service*

As discussed before, the simulation facilities offered by NDLIB are specifically designed for those users that want to run experiments on their local machine. However, in some scenarios, e.g. due to limited computational resources or to the rising of other particular needs, it may be convenient to separate the machine on which the definition of the experiment is made from the one that actually executes the simulation. In order to satisfy such needs, we developed a RESTfull service, NDLIB-REST[17] , that builds upon NDLIB an experiment server queryable through API calls.

NDLIB-REST rationale. The simulation web service is designed around the concept of *experiment*. An experiment, identified by a unique identifier, is composed of two entities: (i) a network and (ii) one (or more) configured models. Experiments are used to keep track of the simulation definition, to return consecutive model iterations to the user

[17]NDLIB-REST: https://github.com/GiulioRossetti/ndlib-rest

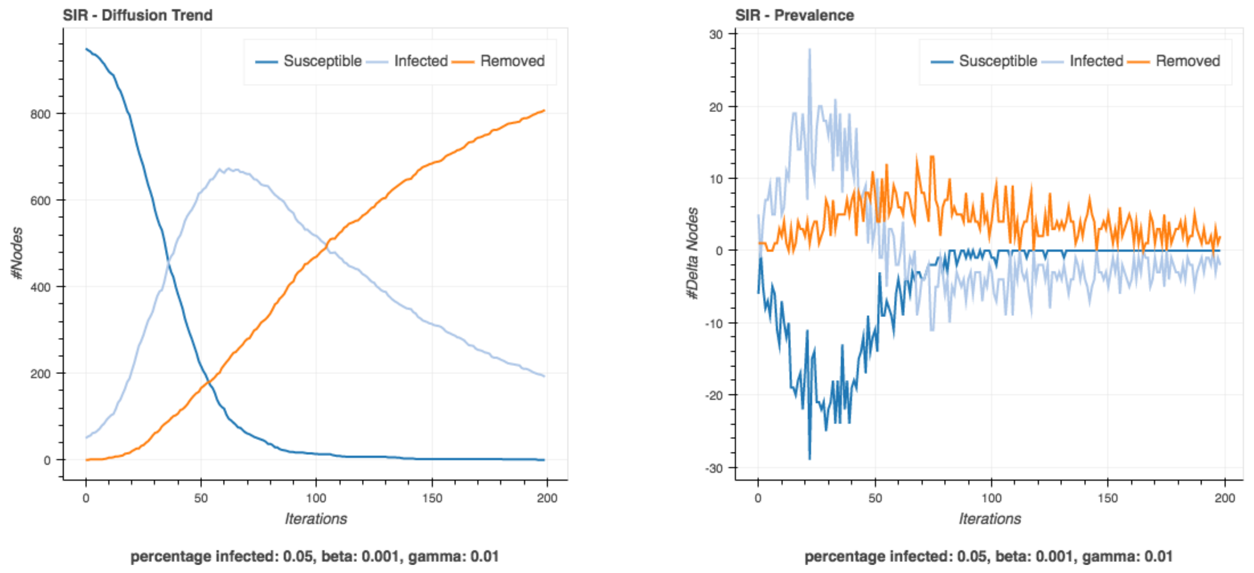
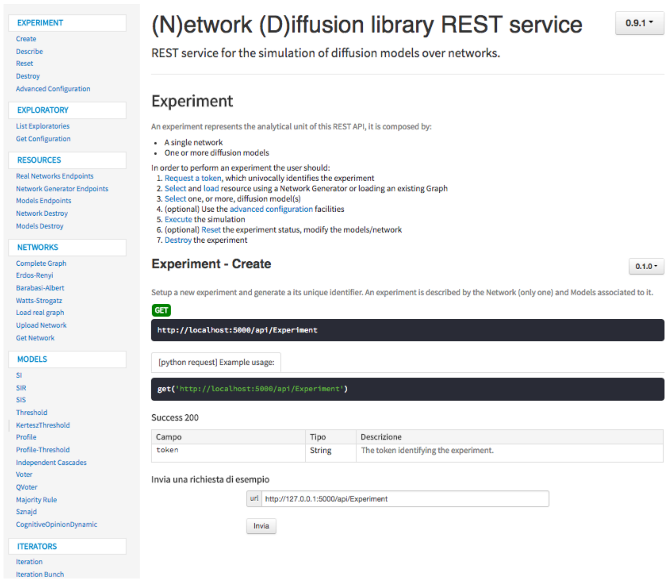


Fig. 1. NDLIB `DiffusionTrend` and `DiffusionPrevalence` SIR plots.



(N)etwork (D)iffusion library REST service

REST service for the simulation of diffusion models over networks.

Experiment

An experiment represents the analytical unit of this REST API, it is composed by:

- A single network
- One or more diffusion models

In order to perform an experiment the user should:

1. Request a token, which univocally identifies the experiment
2. Select and load resource using a Network Generator or loading an existing Graph
3. Select one, or more, diffusion model(s)
4. (optional) Use the advanced configuration facilities
5. Execute the simulation
6. (optional) Reset the experiment status, modify the models/network
7. Destroy the experiment

Experiment - Create

Setup a new experiment and generate a its unique identifier. An experiment is described by the Network (only one) and Models associated to it.

```
http://localhost:5000/api/Experiment
```

[python request] Example usage:

```
get('http://localhost:5000/api/Experiment')
```

Success 200

Campo	Tipo	Descrizione
token	String	The token identifying the experiment.

Invia una richiesta di esempio

url http://127.0.0.1:5000/api/Experiment

Fig. 2. NDLIB-REST documentation webpage.

and to store - locally on the experiment server - the current status of the diffusion process.

In particular, in order to perform an experiment, a user must:

1. Request a token, which univocally identifies the experiment;
2. Select or load a network resource;
3. Select one, or more, diffusion model(s);
4. (optional) Use the advanced configuration facilities to define node/edge parameters;
5. Execute the step-by-step simulation;
6. (optional) Reset the experiment status, modify the models/network;
7. Destroy the experiment.

The last action, involving the destruction of the experiment, is designed to clean the serialization made by the service of the incremental experiment status. If an experiment is not explicitly destroyed its data is removed, and the associated token invalidated, after a temporal window that can be configured by the service administrator. NDLIB-REST is shipped as a Docker[18] container image so to make it configuration free and easier to setup. Moreover, the simulation server is, by default, executed within a Gunicorn[19] instance allowing parallel executions of multiple experiments at the same time. NDLIB-REST is built using Flask[20] and offers a standard online documentation page (shown in Figure 2) that can also be directly used to test the exposed endpoints both configuring and running experiments.

REST API. As a standard for REST services, all the calls made to NDLIB-REST endpoints generate JSON responses. The APIs of the simulation service are organized in six categories so to provide a logic separation among all the exposed resources. In particular, in NDLIB-REST are exposed endpoints handling:

- *Experiments*: endpoints in this category allow to create, destroy, configure, reset and describe experiments;
- *Exploratories*: endpoints in this category allow to load predefined scenarios (e.g. specific networks/models with explicit initial configuration);
- *Resources*: endpoints in this category allow to query the system to dynamically discover the endpoints (and their descriptions) defined within the system;
- *Networks*: endpoints in this category handle a load of network data as well as the generation of synthetic graphs

[18]Docker: https://www.docker.com/
[19]Gunicorn: http://gunicorn.org/
[20]Flask: http://flask.pocoo.org/

(Barabasi-Albert, Erdös-Renyi, Watts-Strogatz, ...);

- *Models*: endpoints in this category expose the NDLIB models;
- *Iterators*: endpoints in this category expose the step-by-step and iteration bunch facilities needed to run the simulation.

Since the simulation service allows to attach multiple diffusion models to the same experiment both the single iteration and the iteration bunch endpoints expose additional parameters that allow the user to select the models for which the call was invoked. By default, when such parameter is not specified, all the models are executed and their incremental statuses returned. A particular class of endpoints is the *E*xploratories one. Such endpoints are used to define the access to pre-set diffusion scenarios. Using such facilities the owner of the simulation server can describe, beforehand, specific scenarios, package them and make them available to the service users. From an educational point of view such mechanism can be used, for instance, by professors to design emblematic diffusion scenarios (composed by both network and initial nodes/edges statuses) so to let the students explore their impact on specific models configurations (e.g. to analyze the role of weak-ties and/or community structures).

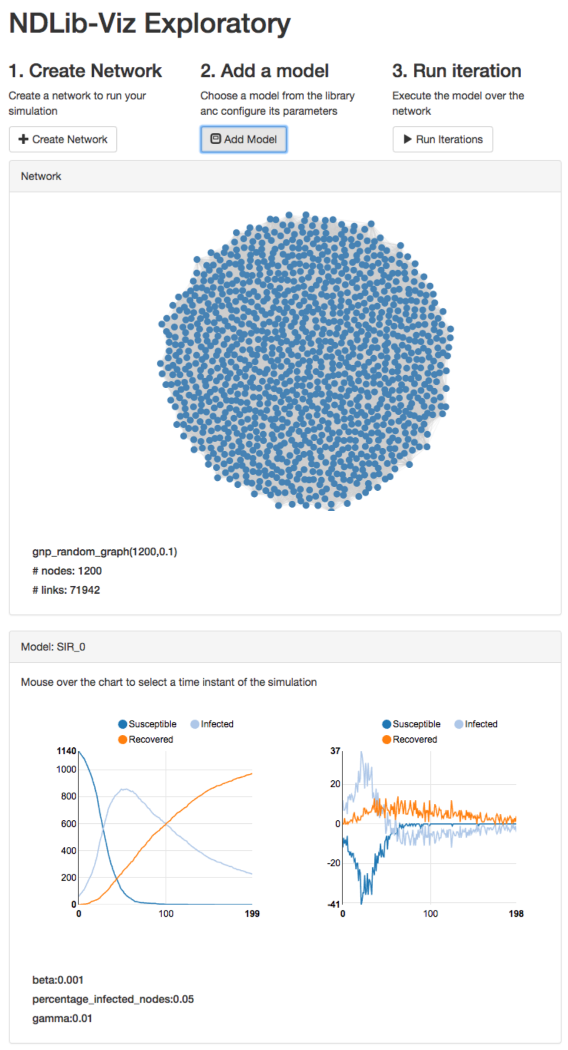
Python API wrapper. In order to provide a simplified interface to query the NDLIB-REST service, we defined a Python wrapper that organizes and exposes all the implemented API. Such API wrapper, shipped along with the web service, allows to define and run remote experiments as shown in the example below:

```python
from NDlibClient import NDlibClient

# Connecting the simulation service
e = NDlibClient("http://127.0.0.1:5000")

# Configuring the experiment
e.create_experiment()
x = e.add_erdos_renyi_graph(300, 0.01)
e.add_SIR(infected=0.1,
          beta=0.001, gamma=0.01)

# Execute the experiment
res = e.get_iteration_bunch(bunch=200)

e.destroy_experiment()
```

C. NDLIB-*Viz: Visualization Framework*

Finally, upon the NDLIB-REST service, we design a visualization platform[21]. NDLIB-Viz aims to make non-technicians able to design, configure and run epidemic simulations, thus removing the barriers introduced by the usual requirements of programming language knowledge. Indeed, apart from the usual research-oriented audience, we developed NDLIB-Viz to support students and facilitate teachers to introduce epidemic models: in order to better support the educational nature of the proposed platform and

[21] Available at: https://github.com/GiulioRossetti/ndlib-viz


NDLib-Viz Exploratory
1. Create Network
Create a network to run your simulation
+ Create Network
2. Add a model
Choose a model from the library anc configure its parameters
Add Model
3. Run iteration
Execute the model over the network
▶ Run Iterations
Network
Model: SIR_0
Mouse over the chart to select a time instant of the simulation
Susceptible
Infected
Recovered
1140
1000
800
600
400
200
0
100
199
37
20
-20
-41
198

gnp_random_graph(1200,0.1)
# nodes: 1200
# links: 71942

beta:0.001
percentage_infected_nodes:0.05
gamma:0.01

Fig. 3. NDLIB Visualization Framework appearance during a simulation. The top toolbar presents a schematic workflow to execute a simulation. The central view presents a visualization of the status of each node. The bottom part presents a synthetic visualization of properties of the simulation. Mouse interaction allow the user to select a specific time instant of the simulation to update all the other views accordingly

collect user feedbacks, we currently employ NDLIB-Viz in a Social Analysis course of the Computer Science Master Degree at the University of Pisa, Italy. The platform itself is a web application: it can be executed on a local as well as on a remote NDLIB-REST installation. The visual interface guides the user to follow the NDLIB-REST workflow through a toolbar on top of the page with a schematic representation of the expected steps (see Figure 3). As a first step, the user should create an experiment and a network. The parameters to import or generate the network are exposed via a web form, providing also suggestions and error checking for the parameters entered by the user. Once the network has been created, it is rendered on the screen in a viewport. At the second step, the user may create one or more diffusion models to attach to the network. Each model is simulated according to the specifics of NDlib-REST. The simulation is handled by the user in the third step, where she can choose the number of iteration to execute. The time discrete simulation is presented to the user by mean of a set of linked displays [19], where the selection and interaction performed on a view are propagated to the other views. The main view is the visualization of the network, showing all the nodes and their links. Each node is assigned a color to represent its status in a specific time instant of the simulation. The choice of the time instant to visualize is determined by the user, by means of selection of the mouse on the linked displays. The bottom part of the interface shows the result of simulation for each model. An aggregated visualization of each model is presented in a block containing the reference to the model and its parameters, and two charts to show `DiffusionTrend` and `DiffusionPrevalence` plots, like those presented in Section IV. Exploiting the web interface, the plots are interactive. Exploring the plots with the mouse the user may receive additional information on specific time instant (see tooltip box in the example in Figure 3). The selection on the plot is directly linked with the block of network visualization, showing the node statuses in the corresponding time instant.

Architecture. The Visualization Framework is a single page web application implemented using Javascript and HTML 5. The decoupling of the simulation engine and the visual interface allow us to exploit modern browsers to provide an efficient environment for visualization of models and interactions. The structure and layout of the page are managed with Bootstrap[22]. The business logic and visualization of graphical widgets are implemented in D3.js[23]. Nodes and edges of the networks are drawn using the Force Layout library provided by the D3 library. The network visualization is implemented using Canvas object provided by standard HTML5. This allows a very efficient update of the network view. The charts showing the Diffusion Trend and Prevalence (presented in section IV) are created using NVD3 library[24].

The Visualization Framework is implemented using a Model-Control-View (MCV) design pattern. The model is managed by a central component that implements a REST API client that handle the status of the experiment. When the user interacts with one of the views (charts, network layout, toolbar) the controller notifies the model to update the experiment. Each interaction with the visual interface is managed by the model component that centralizes all the communications with the REST server. The calls to the server are executed asynchronously and the component updates the visual interface as soon as a response arrives from the server.

V. Conclusions and Future Works

In this paper, we introduced the NDlib environment, a modular framework designed to provide an easy access to network diffusion simulation models to a broad user base. The framework, composed by a standalone library NDlib, a RESTfull simulation service, and a visualization tool is built upon the NetworkX library and released as free software.

Indeed, several tools are nowadays available to students, developers and researchers to simulate and study diffusive phenomena on complex networks, however NDlib is one of the first that offers multiple interfaces specifically designed to solve specific use case. Among its competitors, partially described in Section III-A, only the EpiModel R package seems able to offer a comparable framework in terms of library and visualization facilities: however, NDlib is tailored for a wider audience making easy to set up remote experimental services. Moreover, the number of implemented models (see Appendix A) as well as the extensibility of the library make NDlib a valid solution for those users that need to compare different diffusive schema other than the classical compartimental ones (SI, SIS, SIR).

As future work we plan to further increase the number of models implemented in the NDlib library and to extend it to support not only static graphs but also evolving ones thus allowing the simulation of both dynamics *on* and *of* networks. Moreover, NDlib-REST is in process of being integrated into a visual simulation platform developed within the CIMPLEX H2020 EU project.

Acknowledgments

This work is funded by the European Community's H2020 Program under the funding scheme "FETPROACT-1-2014: Global Systems Science (GSS)", grant agreement # 641191 CIMPLEX "Bringing CItizens, Models and Data together in Participatory, Interactive SociaL EXploratories"[25]. This work is supported by the European Community's H2020 Program under the scheme "INFRAIA-1-2014-2015: Research Infrastructures", grant agreement #654024 *"SoBigData: Social Mining & Big Data Ecosystem"*[26].

References

[1] P. Szor, "Fighting computer virus attacks." *USENIX*, 2004.
[2] S. Havlin, "Phone infections," *Science*, 2009.
[3] P. Wang, M. C. González, R. Menezes, and A. L. Barabási, "Understanding the spread of malicious mobile-phone programs and their damage potential," *International Journal of Information Security*, 2013.
[4] R. S. Burt, "Social Contagion and Innovation: Cohesion Versus Structural Equivalence," *American Journal of Sociology*, 1987.
[5] A. Sîrbu, V. Loreto, V. D. Servedio, and F. Tria, "Opinion dynamics: Models, extensions and external effects," in *Participatory Sensing, Opinions and Collective Awareness*. Springer International Publishing, 2017, pp. 363–401.
[6] G. Deffuant, D. Neau, F. Amblard, and G. Weisbuch, "Mixing beliefs among interacting agents," *Advances in Complex Systems*, vol. 3, no. 4, pp. 87–98, 2000.
[7] K. Sznajd-Weron and J. Sznajd, "Opinion evolution in closed community," *International Journal of Modern Physics C*, vol. 11, pp. 1157–1165, 2001.
[8] A. Sîrbu, V. Loreto, V. D. Servedio, and F. Tria, "Opinion dynamics with disagreement and modulated information," *Journal of Statistical Physics*, pp. 1–20, 2013.

[22]Bootstrap: http://getbootstrap.com/
[23]D3.js:https://d3js.org/
[24]NVD3: http://nvd3.org/
[25]CIMPLEX: https://www.cimplex-project.eu
[26]SoBigData: http://www.sobigdata.eu

[9] 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., 2008, pp. 11–15.
[10] L. Ahrenberg, S. Kok, K. Vasarhelyi, and A. Rutherford, “Nepidemix,” 2016.
[11] I. Z. Kiss, J. C. Miller, and P. Simon, *(Book) Mathematics of epidemics on networks: from exact to approximate models*. Springer, Forthcoming.
[12] G. Csardi and T. Nepusz, “The igraph software package for complex network research,” *InterJournal, Complex Systems*, vol. 1695, no. 5, pp. 1–9, 2006.
[13] S. Jenness, S. M. Goodreau, and M. Morris, “Epimodel: Mathematical modeling of infectious disease. r package version 1.3.0.” 2017. [Online]. Available: http://www.epimodel.org
[14] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker, “Cytoscape: a software environment for integrated models of biomolecular interaction networks.” *Genome Research*, vol. 13, pp. 2498–2504, Nov. 2003.
[15] M. Bastian, S. Heymann, and M. Jacomy, “Gephi: An open source software for exploring and manipulating networks.” in *ICWSM*. The AAAI Press, 2009.
[16] U. Wilensky, “Netlogo,” 1999.
[17] W. Van den Broeck, C. Gioannini, 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, 2011.
[18] J. J. Grefenstette, S. T. Brown, R. Rosenfeld, J. DePasse, N. T. Stone, P. C. Cooley, W. D. Wheaton, A. Fyshe, D. D. Galloway, A. Sriram *et al.*, “Fred (a framework for reconstructing epidemic dynamics): an open-source software system for modeling infectious diseases and control strategies using census-based populations,” *BMC public health*, vol. 13, no. 1, p. 940, 2013.
[19] C. M. Newton, “Graphics: from alpha to omega in data analysis,” in *Graphical Representation of Multivariate Data*, P. C. Wang, Ed. Academic Press, 1978, pp. 59 – 92. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780127347509500083
[20] W. O. Kermack and A. McKendrick, “A Contribution to the Mathematical Theory of Epidemics,” *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 115, no. 772, pp. 700–721, Aug. 1927.
[21] M. Granovetter, “Threshold models of collective behavior,” *The American Journal of Sociology*, vol. 83, no. 6, pp. 1420–1443, 1978.
[22] Z. Ruan, G. Iñiguez, M. Karsai, and J. Kertész, “Kinetics of social contagion,” *Phys. Rev. Lett.*, vol. 115, p. 218702, Nov 2015.
[23] D. J. Watts, “A simple model of global cascades on random networks,” *Proceedings of the National Academy of Sciences*, vol. 99, no. 9, pp. 5766–5771, 2002.
[24] D. Kempe, J. Kleinberg, and E. Tardos, “Maximizing the spread of influence through a social network,” in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’03, 2003, pp. 137–146.
[25] P. Clifford and A. Sudbury, “A model for spatial conflict,” *Biometrika*, vol. 60, no. 3, pp. 581–588, 1973.
[26] R. Holley and T. Liggett, “Ergodic theorems for weakly interacting infinite systems and the voter model,” *Ann. Probab.*, vol. 3, no. 4, pp. 643–663, Aug 1975.
[27] P. L. Krapivsky, S. Redner, and E. Ben-Naim, *A kinetic view of statistical physics*. Cambridge University Press, 2010.
[28] C. Castellano, M. A. Munoz, and R. Pastor-Satorras, “The non-linear q-voter model,” *Physical Review E*, vol. 80, p. 041129, 2009.
[29] S. Galam, “Minority opinion spreading in random geometry,” *Eur. Phys. J. B*, vol. 25, no. 4, pp. 403–406, 2002.
[30] R. Friedman and M. Friedman, *The Tyranny of the Status Quo*. Orlando, FL, USA: Harcourt Brace Company, 1984.
[31] D. Vilone, F. Giardini, M. Paolucci, and R. Conte, “Reducing individuals’ risk sensitiveness can promote positive and non-alarmist views about catastrophic events in an agent-based simulation,” *arXiv preprint arXiv:1609.04566*, 2016.

Appendix
Diffusion Methods implemented in NDlib

NDlib exposes several network diffusion models, covering both epidemic approaches as well as and opinion dynamics. In particular, the actual release of the library (v2.0.1) implements the following algorithms:

A. Epidemic Models.

SI: This model was introduced in 1927 by Kermack [20]. In the SI model, during the course of an epidemics, a node is allowed to change its status only from Susceptible (S) to Infected (I). SI assumes that if, during a generic iteration, a susceptible node comes into contact with an infected one, it becomes infected with probability β: once a node becomes infected, it stays infected (the only transition is $S \rightarrow I$).

SIR: this model was still introduced in 1927 by Kermack [20]. In the SIR model, during the course of an epidemics, a node is allowed to change its status from Susceptible (S) to Infected (I), then to Removed (R). SIR assumes that if, during a generic iteration, a susceptible node comes into contact with an infected one, it becomes infected with probability β, than it can be switch to removed with probability γ (the only transition allowed are $S \rightarrow I \rightarrow R$).

SIS: as SIR, the SIS model is a variation of the SI model introduced in [20]. The model assumes that if, during a generic iteration, a susceptible node comes into contact with an infected one, it becomes infected with probability β, than it can be switch again to susceptible with probability λ (the only transition allowed are $S \rightarrow I \rightarrow S$).

Threshold: this model was introduced in 1978 by Granovetter [21]. In the Threshold model during an epidemics, a node has two distinct and mutually exclusive behavioral alternatives, e.g., it can adopt or not a given behavior, participate or not participate in a riot. Nodes individual decision depends on the percentage of its neighbors have made the same choice, thus imposing a threshold. The model works as follows: each node starts with its own threshold τ and status (infected or susceptible). During the iteration t every node is observed: iff the percentage of its neighbors that were infected at time $t-1$ is grater than its threshold it becomes infected as well.

Kertesz Threshold: this model was introduced in 2015 by Ruan et al. [22] and it is an extension of the Watts threshold model [23]. The authors extend the classical model introducing a density r of blocked nodes - nodes which are immune to social influence - and a probability of spontaneous adoption p to capture external influence. Thus, the model distinguishes three kinds of node: Blocked (B), Susceptible (S) and Adoptiong (A). A node can adopt either under its neighbors influence or due to endogenous effects.

Independent Cascades: this model was introduced by Kempe et all in 2003 [24]. The Independent Cascades model starts with an initial set of active nodes A_0: the diffusive process unfolds in discrete steps according to the following randomized rule:

- When node v becomes active in step t, it is given a single chance to activate each currently inactive neighbor w; it succeeds with a probability $p_{v,w}$.
- If w has multiple newly activated neighbors, their attempts are sequenced in an arbitrary order.

- If v succeeds, then w will become active in step $t+1$; but whether or not v succeeds, it cannot make any further attempts to activate w in subsequent rounds.

The process runs until no more activations are possible.

Node Profile: this model is a variation of the Threshold one, it assumes that the diffusion process is only apparent; each node decides to adopt or not a given behavior - once known its existence - only on the basis of its own interests. In this scenario the peer pressure is completely ruled out from the overall model: it is not important how many of its neighbors have adopted a specific behaviour, if the node does not like it, it will not change its interests. Each node has its own profile describing how many it is likely to accept a behaviour similar to the one that is currently spreading. The diffusion process starts from a set of nodes that have already adopted a given behaviour H: for each of the susceptible nodes in the neighborhood of a node u that has already adopted H, an unbalanced coin is flipped, the unbalance given by the personal profile of the susceptible node; if a positive result is obtained the susceptible node will adopt the behaviour.

Node Profile-Threshold: this model, still extension of the Threshold one, assumes the existence of node profiles that act as preferential schemas for individual tastes but relax the constraints imposed by the Profile model by letting nodes influenceable via peer pressure mechanisms. The peer pressure is modeled with a threshold. The diffusion process starts from a set of nodes that have already adopted a given behaviour H: for each of the susceptible node an unbalanced coin is flipped if the percentage of its neighbors that are already infected excedes its threhosld. As in the Profile Model the coin unbalance is given by the personal profile of the susceptible node; if a positive result is obtained the susceptible node will adopt the behaviour.

B. *Opinion Dynamic Models.*

Voter: this model is one of the simplest models of opinion dynamics, originally introduced to analyze competition of species [25] and soon after applied to model elections [26]. The model assumes the opinion of an individual to be a discrete variable ± 1. The state of the population varies based on a very simple update rule: at each iteration, a random individual is selected, who then copies the opinion of one random neighbor. Starting from any initial configuration, on a complete network, the entire population converges to a consensus on one of the two options [27]. The probability that consensus is reached on opinion $+1$ is equal to the initial fraction of individuals holding that opinion.

Snajzd: this model [7] is a variant of spin model employing the theory of social impact, which takes into account the fact that a group of individuals with the same opinion can influence their neighbors more than one single individual. In the original model the social network is a 2-dimensional lattice, however, we also implemented the variant on any complex networks. Each agent has an opinion $\sigma_i = \pm 1$; at each time step, a pair of neighboring agents is selected and, if their opinion coincides, all their neighbors take that opinion. The model has been shown to converge to one of the two agreeing stationary states, depending on the initial density of up-spins (transition at 50% density).

Q-Voter: this model was introduced as a generalization of discrete opinion dynamic models [28]. Here, N individuals hold an opinion ± 1. At each time step, a set of q neighbors are chosen and, if they agree, they influence one neighbor chose at random, i.e. this agent copies the opinion of the group. If the group does not agree, the agent flips its opinion with probability ϵ. It is clear that the voter and Sznajd models are special cases of this more recent model ($q = 1, \epsilon = 0$ and $q = 2, \epsilon = 0$, respectively). Analytic results for $q \leq 3$ validate the numerical results obtained for the special case models, with transitions from an ordered phase (small ϵ) to a disordered one (large ϵ). For $q > 3$, a new type of transition between the two phases appears, which consist of passing through an intermediate regime where the final state depends on the initial condition. We implemented in NDlib the model with $\epsilon = 0$.

Majority Rule: this model is a different discrete model of opinion dynamics, proposed to describe public debates [29]. Agents take discrete opinions ± 1, just like the voter model. All agents can interact with all other agents (also in our implementation), so the social network is always a complete graph. At each time step, a group of r agents is selected randomly and they all take the majority opinion within the group. The group size can be fixed or taken at each time step from a specific distribution. If r is odd, then the majority opinion is always defined, however, if r is even there could be tied situations. To select a prevailing opinion, in this case, a bias in favor of one opinion $(+1)$ is introduced. This idea is inspired by the concept of social inertia [30].

Cognitive Opinion Dynamics: this model was introduced by Vilone et all. [31], which models the state of individuals taking into account several cognitively-grounded variables. The aim is to simulate a response to risk in catastrophic events in the presence of external (institutional) information. The individual opinion is modeled as a continuous variable $O_i \in [0, 1]$, representing the degree of perception of the risk (how probable it is that the catastrophic event will actually happen). This opinion evolves through interactions with neighbours and external information, based on four internal variables for each individual i: risk sensitivity ($R_i \in \{-1, 0, 1\}$), tendency to inform others ($\beta_i \in [0, 1]$), trust in institutions ($T_i \in [0, 1]$) and trust in peers ($\Pi_i = 1 - T_i$). These values are generated when the population is initialized and stay fixed during the simulation. In our implementation, we allow some control on the distribution of these parameters. The update rules define how O_i values change in time (see original paper [31] for details). The model was shown to be able to reproduce well various real situations; in particular, it is visible that risk sensitivity is more important than trust in institutional information when it comes to evaluating risky situations.