COMPARATIVE ANALYSIS OF AGENT-BASED MODELING FRAMEWORKS FOR SIGNAL PROPAGATION IN COMPLEX NETWORKS: NETLOGO AND PYTHON MESA

Cristian Berceanu[1] and Monica Patrascu[1,2,3]

Signal propagation is a prevalent research topic for an abounding number of complex network applications. The nonlinear characteristics of complex networks determined by complex interactions and topology require suitable agent-based modeling and simulation frameworks. This paper presents a comparison of two such frameworks in terms of usability and performance for studying signal propagation in complex networks. Our findings are based on two use cases that we implemented in both frameworks.

Keywords: agent-based modeling, complex networks, signal propagation

1. Introduction

The science of complex systems has registered increased interest in the past decades, as our technological universe continues to expand and interact with the sociological and human component, leading to the appearance of socio-technical systems [1] as a field of study. Complex systems are currently defined as comprised of connected entities which exhibit an emergent behavior at whole-system [2] level without the guidance of a central authority, but resulting from local interactions [3]. These component entities can be other systems, agents, humans, etc. with various levels of autonomy and/or intelligence. Due to the focus on interactions, complex networks are currently the prevalent form of organization and modeling of complex systems.

A complex network is a network formed of nodes (vertices) which connect through links (edges) with special properties [4]: non-trivial topology, small world effect, scale free node rank distribution, etc. All these are reflected in complex patterns of interaction, which makes communication between nodes critically important. In complex networks, communication between components is either direct and indirect [5]. Direct communication requires nodes to be linked, and thus remains localized within vicinities. Indirect communication

[1]Complex Systems Laboratory, Department of Automatic Control and Systems Engineering, University Politehnica of Bucharest, Romania, email: cristian.berceanu@stud.acs.upb.ro

[2]Centre for Elderly and Nursing Home Medicine, Department of Global Public Health and Primary Care, University of Bergen, Norway, email: monica.patrascu@uib.no

[3]Neuro-SysMed Center, Department of Global Public Health and Primary Care, University of Bergen, Norway

is either node-to-node or stigmergic. The latter implies that information is deposited and collected in and from the environment [6], whereas node-to-node indirect communication is a propagation phenomenon through which information travels on paths comprised of at least three nodes. This is known as signal propagation in complex networks and it models phenomena of diffusion, viral spread and epidemics, infodemics, cascading effects, social behavior, etc. A recent review [7] situates signal propagation as a key element to understanding emergent phenomena, while our recent survey [8] lists the current issues on routing, load, and protocols regarding the management of communication in networks in the absence of central authorities or controllers. In system science, signals can be either information, energy or matter [9]. In particle based systems, signal propagation is performed through the transfer of energy [10, 11, 12]. When particle based system are modeled as complex networks, the topology of the complex network is shifting to correlate with the state of the system [13, 14, 15].

Agent based modeling (ABM) [16] falls under the larger umbrella of multi-agent systems (MAS) and provides useful tools for simulating and understanding complex networks. ABM and MAS have also generated the agent-oriented programming (AOP) paradigm [17]. The resulting models are often accompanied by visualizations which showcase the interactions between agents. A wide range of application fields have benefited from ABM: social phenomena [18, 19, 20, 21], urban traffic simulation [22, 23, 24, 25, 26], financial market analysis [27, 28, 29, 30, 31], nonlinear biosystems dynamics [32, 33, 34, 35, 36].

An agent is an entity capable of sensing the environment or other agents, making decisions (either deliberatively or reactively), and performing actions upon the environment other agents [37]. This definition is not unique to the study of agents in general, but it does contain the main elements of an agent. Of note is that communication between agents is not explicit, but implied by their capabilities to interact with each other. An agent network is a set of interdependent agents $A = \{a_1, ..., a_n\}$, and a compatible relation R, $R \subseteq A \times A$ [38]. In this definition, R is reflexive and symmetric which means that the connection between two agents is bidirectional, so $\langle a_i, a_j \rangle \in R$ if and only if a_j is a neighbor of a_i [38].

Several frameworks have been proposed for ABM simulation since their early emergence in the '70s (e.g., Schelling's segregation model [39]) with various specifications and requirements for computational resources [40].

The belief–desire–intention (BDI) model is a representation of agent internal mental attitudes [41] that support the composition of reactive and deliberative agent behaviors [42]. FIPA is an organization established in 1996 dedicated to the standardisation of agent-based modelling that later became an IEEE Computer Society organization. One of the standards defined by FIPA is FIPA-ACL (Agent Communications Language) which facilitates the modelling and interoperability between different agent-based modeling software.

However, most ABM frameworks are not compliant with the FIPA-ACL protocol because it would increase the performance overhead and without a real benefit in the context of small-scale simulations. While BDI-based and FIPA-compliant agent platforms produce conceptually heavy models [43], researchers outside computer science (e.g., sociology, epidemiology, etc.) often turn toward lightweight simulation frameworks. This is because models of agent behaviors are often abstracted to the essentials specific to each studied phenomenon, e.g., humans in a crowd are reduced to particles with direction and speed. While this practice has its limitations [44], a plethora of other emergent processes can still benefit from abstraction and ABM simulation [45].

This paper aims to perform a comparative analysis of two widely-used simulation frameworks to study emergent behaviors of simple agents, NetLogo [46] and Mesa Python library [47], investigating the user interface, friendliness, dependencies, and performance, by creating two models in both frameworks.

The rest of this paper is organized as follows. Section 2 presents an overview of the two frameworks. Section 3 presents an information diffusion model and its results. Section 4 presents a particle collision model implemented in both frameworks and the usability and performance results. Section 5 presents a comparative analysis between the two ABM frameworks based on the two case studies. Section 6 concludes the paper.

2. NetLogo and Mesa Python Library

NetLogo inherited the turtle-patch-observer (fig. 1) model [48] from StarLogoT, a blend of Logo (a member of the Lisp family) and StarLisp [49]. There are three types of agents in NetLogo: *turtles* are agents that move around the simulated world and comprise the agent network, *patches* are cells of the grid over which turtles move and comprise the environment, and a single *observer* agent that allows the model designer to observe the model outcomes [49]. NetLogo is a popular and well documented multi-agent programmable modeling environment [46]. NetLogo is well suited for prototyping multi-agent systems and instructions may be assigned to hundreds of independent agents that operate concurrently [43].

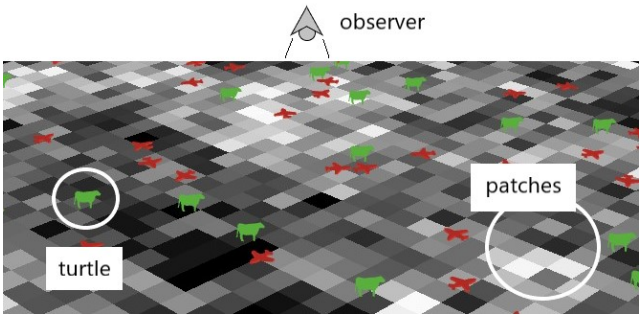


Fig. 1. NetLogo turtle-patch-observer exemplification

Mesa [47] is an open-source Python 3 library and ABM framework providing a web interface that may be enhanced using javascript. Python has become in recent years an increasingly popular language and it is a general purpose programming language, unlike NetLogo's custom scripting language [50]. The architecture of the Mesa library may be divided into three categories (fig. 2 [50]): modeling, analysis and visualization. The *Model* class is a container for model level parameters; the *Agent* classes describe the model agents; the *Scheduler* controls the agent activation regime and handles the model time in general while the *Space* class describes the network and/or space the agents are situated in.

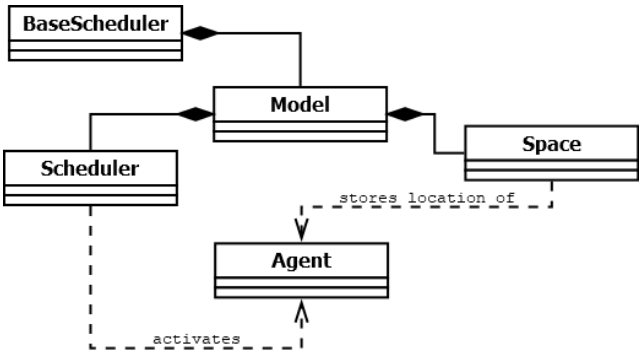


Fig. 2. Mesa (simplified) architecture [50]

3. **Case Study: Information Diffusion Model**

To compare NetLogo and Mesa Python library, we implement a classic signal propagation as information diffusion scenario. In this scenario we evaluate the accessibility and usability of the frameworks. The agent environment consists of a square 2D map. Each agent in the network may have one of two states: red or blue. All agents are initialized with the color blue at the first step of simulation, excepting a single agent which is red. At each simulation step, red agents send a message to each of their neighbouring agents to change their state from blue to red. Blue agents react to the message and change their state; they keep their blue state if no message arrives. Thus, the red state propagates through the map until all agents become red. In this model, neighboring agents are defined as agents with a predefined radius around the message sender. Algorithm 1 shows the behavior of agents, where r is the radius of signal propagation. This model is a basic version of a gossiping algorithm [51], in which all communication is one-to-one and only one piece of information is transmitted through the network. This type of information diffusion is often used instead of one-to-all broadcast, to avoid the overload of the communication network. The same algorithm was implemented in both NetLogo and Mesa. The radius of signal propagation r does not have the

same value in both ABMS tools because the environment is not defined in the same way. The Mesa environemnt is defined by the number of pixels while the environment in NetLogo is defined by the number of patches.

Algorithm 1: Agent behavior

```
if agent-state is red then
    send message to neighbor (in-radius r);
else
    if message received from neighbor then
        switch agent-state to red;
    end
end
```

3.1. NetLogo

To simulate this scenario, we first define a new breed of agents which we name "particles". Defining a breed of agents in NetLogo is the object oriented programming (OOP) equivalent to inheriting from the turtle class. The syntax for defining a new breed of agent is very simple:

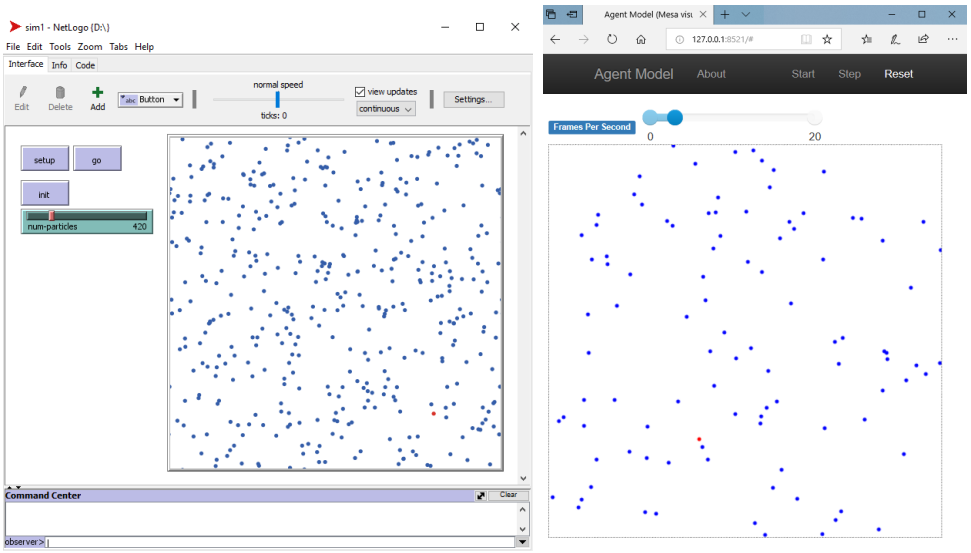
```
breed [particles particle]
```

After defining the breed of agents, we set up the environment of the simulation. We only need a white plane where our particles (turtles) can communicate with each other. To initialize the environment, we create a new function called *setup* and map this function to a button with the same name in the graphical user interface (GUI). The *setup* function first clears everything from the simulation world, including variables, turtles, patches and drawings. Then, *setup* resets the simulation step counter (ticks in NetLogo) and sets the color of all patches to white. After this, the *setup* function generates a number *num-particles* of blue particles with shape "dot" at random positions:

```
to setup
  clear-all
  reset-ticks
  ask patches [ set pcolor white ]
  create-particles num-particles
  [ set color blue
    setxy random-xcor random-ycor
    set shape "dot" ]
end
```

Now that the environment is formed and the blue particles are present on the 2D simulation environment, we also initialize one agent with state color red. Fig. 3a shows this initial network state and the NetLogo model GUI. For this, we create a function called *init* that changes the color of a single random particle to red:

```
to init
  ask one-of particles [ set color red ]
end
```



(a) NetLogo (b) Mesa Python Library

Fig. 3. Initial states and GUIs

After initialization, we define and run the behavior of all agents at each simulation step. NetLogo provides the standard *go* function:

```
to go
  tick
  ask particles with [color = red] [
    ask particles in-radius 3 [ set color red ] ]
end
```

At each simulation step, we iterate through the red particles, and for each of the red particles we propagate the red color to the neighbouring particles within a certain radius.

3.2. Mesa Python Library

To implement the signal propagation model, we extend the *Agent* and *Model* classes. We name *AgentModel* the class extended from *Model* and *Turtle* the class extended from *Agent*:

```
class AgentModel(Model):
        def __init__(self,N,width,height):
                self.num_agents = N
                self.space = ContinuousSpace(width,height,True,0,0)
                globalSpace = self.space
                self.schedule = RandomActivation(self)
```

```
self.running = True
for i in range(self.num_agents):
        a = Turtle(i, self)
        if (i == 0):
                a.change_color("red");
        else:
                a.change_color("blue");
        a.set_space_context(self.space)
        self.schedule.add(a)
        x = random.randrange(self.space.x_max)
        y = random.randrange(self.space.y_max)
        agent_position = np.array((x,y))
        self.space.place_agent(a, agent_position)
```

Python Mesa does not provide an implementation for a continuous space out of the box, but there is an example in the official repository for Craig Reynolds's Boids flocker model [47], where a custom visualization module for drawing agents with continuous positions was implemented. We use this visualization module to place the agents at random locations in the constructor of the *AgentModel* class. Fig. 3b shows the initialization setup of the model and the GUI.

A simulation step function in the *AgentModel* class tells the scheduler when a new simulation step is triggered:

```
def step(self):
        self.schedule.step()
```

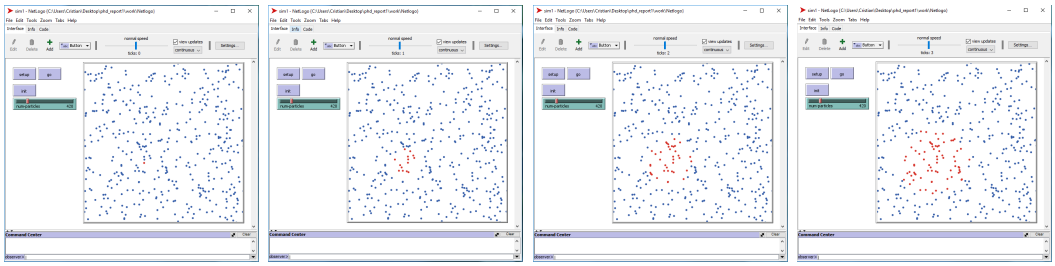
After this, the scheduler triggers the step function defined in the *Turtle* class for each agent, based on the scheduling rules. Python Mesa library uses the Tornado Python package to launch a Tornado web server instance on the current machine and automatically opens the main page in the default browser when the instance is started.

3.3. Results

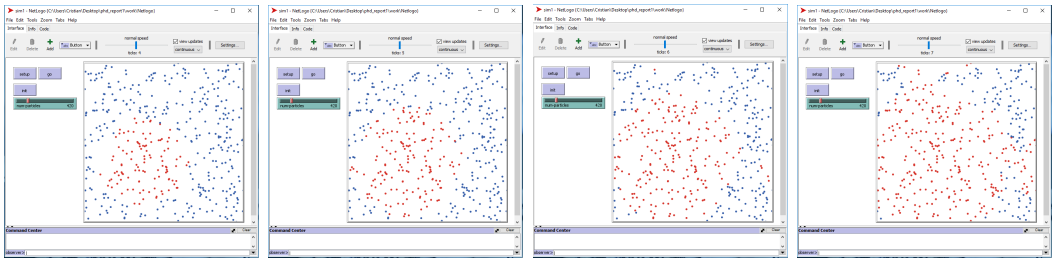
The output of the models in NetLogo and Python Mesa are illustrated in figs. 4 and 5, which show the first 8 steps of the simulation.

4. Case Study: Particle Collision

Newton's Third Law of motion states that "to every action there is always opposed an equal reaction" [52]. We thus build the second signal propagation case study, in which particles do not exchange information, but energy. The law of momentum conservation is closely related to Newton's Third Law of motion [53]. Kinetic energy is conserved in case of elastic collisions but not for inelastic collisions [53]. For this case study, we consider that all collisions between particles in both NetLogo and Mesa Python library are elastic.

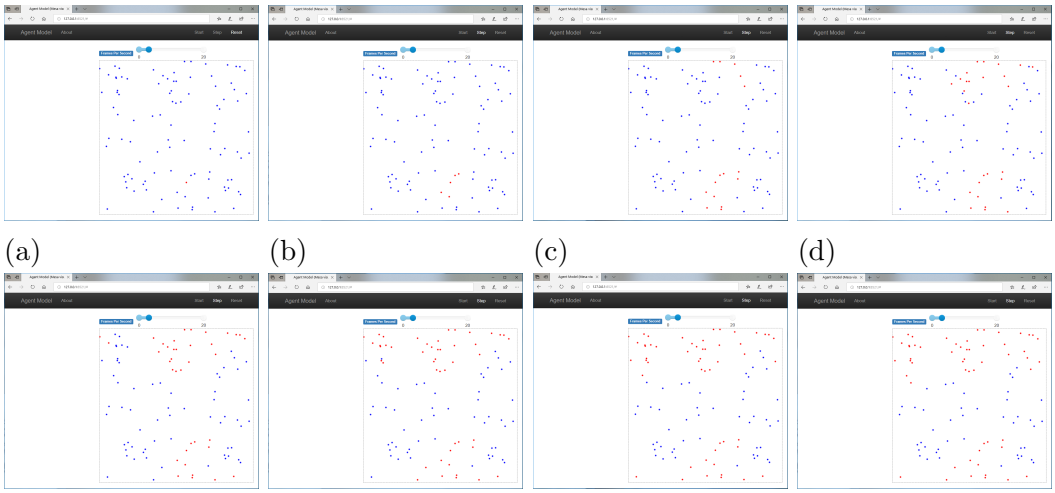


(a) (b) (c) (d)



(e) (f) (g) (h)

Fig. 4. NetLogo simulation output, steps 1 (A) to 8 (F)



(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 5. Mesa simulation output, steps 1 (A) to 8 (F)

Momentum and kinetic energy is conserved in case of elastic collisions [54]. Therefore, if two particles collide (fig. 6), the resulting velocities of the two particles V_1 and V_2 are calculated by solving a system of two equations consisting of: the conservation of momentum and the conservation of kinetic energy. Thus, we can write:

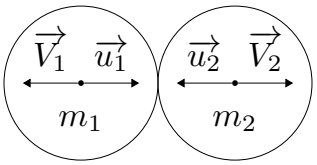


Fig. 6. Elastic collision of 2 particles

$$\begin{cases} m_1\overrightarrow{u_1} + m_2\overrightarrow{u_2} = m_1\overrightarrow{V_1} + m_2\overrightarrow{V_2} \\ m_1\overrightarrow{u_1}^2 + m_2\overrightarrow{u_2}^2 = m_1\overrightarrow{V_1}^2 + m_2\overrightarrow{V_2}^2 \end{cases} \Rightarrow \begin{cases} \overrightarrow{V_1} = \frac{m_1\overrightarrow{u_1} - m_2\overrightarrow{u_1} + 2m_2\overrightarrow{u_2}}{m_1+m_2} \\ \overrightarrow{V_2} = \frac{2m_2\overrightarrow{u_2} - m_2\overrightarrow{u_1} + m_1\overrightarrow{u_1}}{m_1+m_2} \end{cases} \tag{1}$$

4.1. NetLogo

One of the limitations of NetLogo is that it does not provide functions for vector operations. However, we can define local variables using the `let` keyword to store the modulus of two orthogonal vectors $\overrightarrow{v_x}$ and $\overrightarrow{v_y}$, so that:

$$\begin{cases} \overrightarrow{v} = \overrightarrow{v_x} + \overrightarrow{v_y} \\ \theta = \arctan \frac{\|\overrightarrow{v_y}\|}{\|\overrightarrow{v_x}\|} \end{cases}, \tag{2}$$

where θ is the heading attribute of the agent. The heading θ is defined on the interval $[0; 360)$.

Let $\overrightarrow{V_{1x}}$ and $\overrightarrow{V_{1y}}$ be the orthogonal projections of $\overrightarrow{V_1}$, and let $\overrightarrow{V_{2x}}$ and $\overrightarrow{V_{2y}}$ be the orthogonal projections of $\overrightarrow{V_2}$. The moduli of the orthogonal projections are calculated using the solutions of equation 1. The arctangent function is available out of the box in NetLogo and we use it to calculate the direction (heading) of vectors $\overrightarrow{V_1}$ and $\overrightarrow{V_2}$, as per equation 2.

4.2. Mesa Python Library

The Mesa implementation of the elastic collisions is straightforward and uses the *numpy* Python package for vector operations. In Mesa we do not have to define the direction (heading) of the agent because the library is written in Python and supports vector operations out of the box.

4.3. Results

Using Windows Performance Analyzer we ran the two models with 100 particles (fig. 7) and observed that the CPU utilization is slightly higher in case of Mesa, but the Mesa ABM requires less RAM memory. An important difference between the two frameworks is that Mesa is a web application and some CPU and memory resources are also utilized by the web browser. The CPU used in both cases to generate the results is an Intel Core i7-1370P.

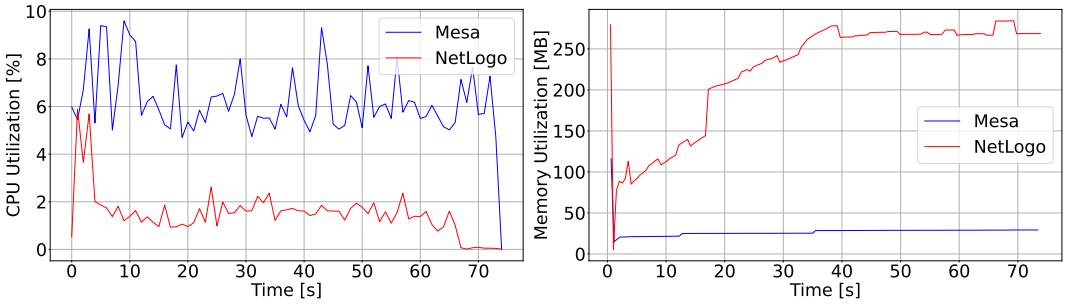


Fig. 7. Performance comparison: CPU and memory utilization for NetLogo and Mesa Python library

5. **Comparative analysis**

Netlogo has an user-friendly interface (fig. 3a) and allows customizing the interface of the simulation environment by adding buttons and slide controls. Buttons may be mapped to environment functions and slide controls may be assigned to environment parameters. Mesa library does not have GUI-related features available out of the box, but Python and javascript may be used to extend or modify the Mesa library code. The model implementation effort is low for NetLogo and suitable for inexperienced users, whereas Mesa requires a more advanced level of programming literacy.

Netlogo supports both 2D and 3D simulations out of the box. Mesa library supports only 2D simulations but it is very easy to resize the simulation environment. Mesa does not require any other dependency besides Python and the GUI (fig. 3b) is displayed in the web browser, therefore it is easy to transfer the code across different platforms. NetLogo does not require other dependencies and is available as a downloadable program to run on a machine without requiring internet connection, but also as a web app. NetLogo also offers a community-populated model library.

In figs. 4 and 5 the signal propagation algorithm returns similar patterns. This is because although the environment is defined differently (number of patches for NetLogo and number of pixels for Mesa), we chose the signal propagation radius so that signal propagates proportionally in both models. It is possible to define the same initial conditions (position of agents in the environment, number of agents, and position of the red agent) in both simulators and thus obtain the same results since the behavior is the same in both simulation scenarios.

In terms of performance, both models perform reasonably well (fig. 7). While NetLogo is a stable, mature and reliable ABM framework, a considerable effort has been devoted to further developing Mesa Python library over the last two years. The BehaviorSpace feature in NetLogo allows simulating the same

experiment multiple times with a certain degrees of randomness. A similar feature is not available in Mesa.

Both are open source software: Mesa is Apache2 licensed and NetLogo is GPL licensed. Neither is FIPA-compliant, using lightweight messages for communication. For both of them there is documentation available online. According to Abar et al. [55], NetLogo is suitable for medium scale to large scale simulation while Mesa is suitable for small scale to medium scale simulations [55]. However, in the case of Mesa library, the architecture allows optimizations by limiting agent activation function calls using a custom scheduler. Both Mesa and NetLogo are flexible enough to allow studying physical dynamic phenomena because there are no restrictions for solving differential equations in the agent model.

6. Conclusions

In this study we compared the features of NetLogo and Mesa Python library and identified advantages and disadvantages for modeling signal propagation in complex networks, by implementing and simulating the same two agent models in both frameworks. Both NetLogo and Mesa are useful tools for simulating agent interactions and non-trivial network structures. While NetLogo is very easy to learn and has an intuitive syntax, Mesa library offers the possibility to customize and extend its modules, making it a versatile simulation tool. This study serves as an argument for framework choice in a larger context of modeling the co-evolution of infodemics and pandemics, as well as information diffusion in complex social networks. Further work includes the implementation of decision-making behavior in agents, preferential attachment and changing states, and propagation of multiple pieces of information through the network.

References

[1] *M. Sony and S. Naik.* Industry 4.0 integration with socio-technical systems theory: A systematic review and proposed theoretical model. In Technology in society, vol. 61, 2020 , p. 101248

[2] *S. Johnson.* Emergence: The connected lives of ants, brains, cities, and software. Simon and Schuster, 2002

[3] *N. Boccara.* Modeling complex systems. Springer Science & Business Media, 2010

[4] *A. D. Broido and A. Clauset.* Scale-free networks are rare. In Nature communications, vol. 10, no. 1, 2019 , p. 1017

[5] *R. Lambiotte, M. Rosvall and I. Scholtes.* From networks to optimal higher-order models of complex systems. In Nature physics, vol. 15, no. 4, 2019 , pp. 313–320

[6] *K. Chen, R. Li, J. Crowcroft, Z. Zhao and H. Zhang.* The implementation of stigmergy in network-assisted multi-agent system. In Proceedings of the 26th Annual International Conference on Mobile Computing and Networking, 2020, pp. 1–2

[7] *P. Ji, J. Ye, Y. Mu, W. Lin, Y. Tian, C. Hens, M. Perc, Y. Tang, J. Sun and J. Kurths.* Signal propagation in complex networks. In Physics Reports, vol. 1017, 2023 , pp. 1–96

[8] *C. Berceanu and M. Pătrașcu.* Engineering emergence: a survey on control in the world of complex networks. In Automation, vol. 3, no. 1
[9] *Y. Wang.* Towards the abstract system theory of system science for cognitive and intelligent systems. In Complex & Intelligent Systems, vol. 1, 2015 , pp. 1–22
[10] *L. Kondic.* Energy propagation through dense granular systems. In Granular Matter, vol. 21, no. 4, 2019 , p. 85
[11] *D. Luís, V. Giangaspero, A. Viladegut, A. Lani, A. Camps and O. Chazot.* Effect of electron number densities on the radio signal propagation in an inductively coupled plasma facility. In Acta Astronautica, vol. 212, 2023 , pp. 408–423
[12] *M. Kim.* Active plasma layer manipulation scheme during hypersonic flight. In Aerospace Science and Technology, vol. 35, 2014 , pp. 135–142
[13] *C. Wang and S. Chen.* Application of the complex network method in solid-state sintering. In Computational materials science, vol. 69, 2013 , pp. 14–21
[14] *M. G. Quiles, E. E. Macau and N. Rubido.* Dynamical detection of network communities. In Scientific reports, vol. 6, no. 1, 2016 , p. 25570
[15] *W. Zhang, C. Yuan, S. Zhang, W. Xiao, N. Zhang and R. Chen.* Correlation mechanism of friction behavior and topological properties of the contact network during powder compaction. In Particuology, vol. 82, 2023 , pp. 98–110
[16] *P. Hansen, X. Liu and G. M. Morrison.* Agent-based modelling and socio-technical energy transitions: A systematic literature review. In Energy Research & Social Science, vol. 49, 2019 , pp. 41–52
[17] *O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci and A. Santi.* Multi-agent oriented programming with JaCaMo. In Science of Computer Programming, vol. 78, no. 6, 2013 , pp. 747–761
[18] *M. Will, J. Groeneveld, K. Frank and B. Muller.* Combining social network analysis and agent-based modelling to explore dynamics of human interaction: A review. In Socio-Environmental Systems Modelling, vol. 2, 2020 , pp. 16325–16325
[19] *G. Scholz, T. Eberhard, R. Ostrowski and N. Wijermans.* Social identity in agent-based models—exploring the state of the art. In Conference of the European Social Simulation Association. Springer, 2019, pp. 59–64
[20] *M. He and J. Lee.* Social culture and innovation diffusion: a theoretically founded agent-based model. In Journal of Evolutionary Economics, vol. 30, 2020 , pp. 1109–1149
[21] *A. Baktash, A. Huang, E. De La Mora Velasco, M. F. Jahromi and F. Bahja.* Agent-based modelling for tourism research. In Current Issues in Tourism, vol. 26, no. 13, 2023 , pp. 2115–2127
[22] *F. F. Bastarianto, T. O. Hancock, C. F. Choudhury and E. Manley.* Agent-based models in urban transportation: review, challenges, and opportunities. In European Transport Research Review, vol. 15, no. 1, 2023 , pp. 1–20
[23] *G. Ben-Dor, E. Ben-Elia and I. Benenson.* Assessing the impacts of dedicated bus lanes on urban traffic congestion and modal split with an agent-based model. In Procedia computer science, vol. 130, 2018 , pp. 824–829
[24] *B. Zhao, K. Kumar, G. Casey and K. Soga.* Agent-based model (ABM) for city-scale traffic simulation: A case study on San Francisco. In International Conference on Smart Infrastructure and Construction 2019 (ICSIC) Driving data-informed decision-making. ICE Publishing, 2019, pp. 203–212
[25] *K. Hager, J. Rauh and W. Rid.* Agent-based modeling of traffic behavior in growing metropolitan areas. In Transportation Research Procedia, vol. 10, 2015 , pp. 306–315
[26] *A. Ion, C. Berceanu and M. Patrascu.* Applying agent based simulation to the design of traffic control systems with respect to real-world urban complexity. In Multi-Agent Systems and Agreement Technologies: 13th European Conference, EUMAS 2015, and

Third International Conference, AT 2015, Athens, Greece, December 17-18, 2015, Revised Selected Papers 13. Springer, 2016, pp. 395–409

[27] *T. Mizuta.* An agent-based model for designing a financial market that works well. In 2020 IEEE symposium series on computational intelligence (SSCI). IEEE, 2020, pp. 400–406

[28] *L. Wang, K. Ahn, C. Kim and C. Ha.* Agent-based models in financial market studies. In Journal of Physics: Conference Series. IOP Publishing, 2018, vol. 1039, p. 012022

[29] *A. Todd, P. Beling, W. Scherer and S. Y. Yang.* Agent-based financial markets: A review of the methodology and domain. In 2016 IEEE symposium series on computational intelligence (SSCI). IEEE, 2016, pp. 1–5

[30] *A. Mandes and P. Winker.* Complexity and model comparison in agent based modeling of financial markets. In Journal of Economic Interaction and Coordination, vol. 12, 2017 , pp. 469–506

[31] *N. Schmitt and F. Westerhoff.* Heterogeneity, spontaneous coordination and extreme events within large-scale and small-scale agent-based financial market models. In Journal of Evolutionary Economics, vol. 27, 2017 , pp. 1041–1070

[32] *S. Sadhukhan, P. Mishra, S. K. Basu and J. Mandal.* A multi-scale agent-based model for avascular tumour growth. In Biosystems, vol. 206, 2021 , p. 104450

[33] *M. Ponce-de Leon, A. Montagud, V. Noël, A. Meert, G. Pradas, E. Barillot, L. Calzone and A. Valencia.* PhysiBoSS 2.0: a sustainable integration of stochastic boolean and agent-based modelling frameworks. In NPJ Systems Biology and Applications, vol. 9, no. 1, 2023 , p. 54

[34] *A. Ghandar, G. Theodoropoulos, M. Zhong, B. Zhen, S. Chen, Y. Gong and A. Ahmed.* An agent-based modelling framework for urban agriculture. In 2019 Winter Simulation Conference (WSC). IEEE, 2019, pp. 1767–1778

[35] *M. Ginovart.* Discovering the power of individual-based modelling in teaching and learning: The study of a predator–prey system. In Journal of Science Education and Technology, vol. 23, 2014 , pp. 496–513

[36] *B. Modu, N. Polovina and S. Konur.* Agent-based modeling of malaria transmission. In IEEE Access, vol. 11, 2023 , pp. 19794–19808

[37] *Y. Rizk, M. Awad and E. W. Tunstel.* Decision making in multiagent systems: A survey. In IEEE Transactions on Cognitive and Developmental Systems, vol. 10, no. 3, 2018 , pp. 514–529

[38] *D. Ye, M. Zhang and D. Sutanto.* Self-adaptation-based dynamic coalition formation in a distributed agent network: A mechanism and a brief survey. In IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 5, 2013 , pp. 1042–1051

[39] *T. C. Schelling.* Dynamic models of segregation. In Journal of mathematical sociology, vol. 1, no. 2, 1971 , pp. 143–186

[40] *R. J. Allan et al.* Survey of agent based modelling and simulation tools. Science & Technology Facilities Council New York, 2010

[41] *G. Kardas, B. T. Tezel and M. Challenger.* Domain-specific modelling language for belief–desire–intention software agents. In IET Software, vol. 12, no. 4, 2018 , pp. 356–364

[42] *U. KC and J. Chodorowski.* A case study of adding proactivity in indoor social robots using belief–desire–intention (BDI) model. In Biomimetics, vol. 4, no. 4, 2019 , p. 74

[43] *J. Carbo, N. Sanchez-Pi and J. Molina.* Agent-based simulation with netlogo to evaluate ambient intelligence scenarios. In Journal of Simulation, vol. 12, no. 1, 2018 , pp. 42–52

[44] *A. Sieben, J. Schumann and A. Seyfried.* Collective phenomena in crowds—where pedestrian dynamics need social psychology. In PLoS one, vol. 12, no. 6, 2017 , pp. 1–19

[45] *G. Sotnik, B. A. Cassell, M. J. Duveneck and R. M. Scheller.* A new agent-based model provides insight into deep uncertainty faced in simulated forest management. In Landscape Ecology, vol. 37, no. 5, 2022 , pp. 1251–1269
[46] Net Logo website. `http://ccl.northwestern.edu/netlogo/`. Accessed: 2023-11-13
[47] Mesa python 3 library github repository. `https://github.com/projectmesa/mesa/`. Accessed: 2023-11-13
[48] *M. Batty and B. Jiang.* Multi-agent simulation: new approaches to exploring space-time dynamics in GIS. In Centre for Advanced Spatial Analysis (UCL)
[49] *S. Tisue and U. Wilensky.* Netlogo: A simple environment for modeling complexity. In International conference on complex systems. Boston, MA, 2004, vol. 21, pp. 16–21
[50] *D. Masad and J. Kazil.* MESA: an agent-based modeling framework. In Proceedings of the 14th Python in Science Conference (SCIPY 2015), 2015, pp. 53–60
[51] *A. Mosebach and J. Lunze.* A deterministic gossiping algorithm for the synchronization of multi-agent systems. In IFAC-PapersOnLine, vol. 48, no. 22, 2015 , pp. 1–7
[52] *S. Hawking.* On the shoulders of giants: The great works of physics and astronomy, 2002
[53] *T. Bryce and K. MacMillan.* Momentum and kinetic energy: Confusable concepts in secondary school physics. In Journal of Research in Science Teaching: The Official Journal of the National Association for Research in Science Teaching, vol. 46, no. 7, 2009 , pp. 739–761
[54] *Z. Tan, J. Ning, Y. Liu, X. Wang, G. Yang and W. Yang.* ECRModel: An elastic collision-based rumor-propagation model in online social networks. In IEEE Access, vol. 4, 2016 , pp. 6105–6120
[55] *S. Abar, G. K. Theodoropoulos, P. Lemarinier and G. M. O'Hare.* Agent based modelling and simulation tools: a review of the state-of-art software. In Computer Science Review, vol. 24, 2017 , pp. 13–33