A Web-Based System for Contagion Simulations on Networked Populations

Tanvir Ferdousi
Biocomplexity Institute and Initiative
University of Virginia
Charlottesville, VA 22904, USA
jcr5wj@virginia.edu

Aparna Kishore
Biocomplexity Institute and Initiative
University of Virginia
Charlottesville, VA 22904, USA
ak8mj@virginia.edu

Lucas Machi
Biocomplexity Institute and Initiative
University of Virginia
Charlottesville, VA 22904, USA
lhm4v@virginia.edu

Dustin Machi
Biocomplexity Institute and Initiative
University of Virginia
Charlottesville, VA 22904, USA
dm8qs@virginia.edu

Chris J. Kuhlman
Biocomplexity Institute and Initiative
University of Virginia
Charlottesville, VA 22904, USA
cjk8gx@virginia.edu

S. S. Ravi
Biocomplexity Institute and Initiative
University of Virginia
Charlottesville, VA 22904, USA
ssr6nh@virginia.edu

***Abstract*—Motivated by a wide range of applications, research on agent-based models of contagion propagation over networks has attracted a lot of attention in the literature. Many of the available software systems for simulating such agent-based models require users to download software, build the executable and set up execution environments. Further, running the resulting executable may require access to high performance computing clusters. Our work describes an open access software system (NetSimS) that works under the "Modeling and Simulation as a Service" (MSaaS) paradigm. It allows users to run simulations by selecting agent-based models and parameters, initial conditions, and networks through a web interface. The system supports a variety of models and networks with millions of nodes and edges.**

In addition to the simulator, the system includes components that allow users to choose initial conditions for simulations in a variety of ways, to analyze the data generated through simulations, and to produce plots from the data. We describe the components of NetSimS and carry out a performance evaluation of the system. We also discuss two case studies carried out on large networks using the system. NetSimS is a major component within *net.science*, a cyberinfrastructure for network science.

***Index Terms*—Agent-Based Simulation, Contagion, Networks, Modeling and Simulation as a Service, Cyberinfrastructure**

I. Introduction

A. Background and Motivation

Many problems are studied by computing contagion dynamics on networked populations. In these networks, nodes are entities such as humans, companies and institutions. Edges represent pairwise interactions between entities. Some examples of research on contagion processes are: the spread of COVID-19 [1] and Ebola [2] viruses, diffusion of invasive biological species over multi-pathway spatial networks [3], simulation of individual movements among urban locations [4], the spread of information on social media platforms [5], analyzing the spread of rumors in social networks [6], the flow of human behavior in social influence networks [7], the spread of incivility among people [8], and shock-waves or economic crises affecting financial institutions through their network of obligations [9]. It is essential to develop computational tools to study various aspects of contagions, as there is a wide range of applications.

Because of this need, many agent-based simulation systems have been developed (see Section II). Many simulators require software experience to configure a build environment, compile software, and specify execution environments. Input files must be constructed according to prescribed formats, and output from simulations typically require post-processing (e.g., data analyses) to visualize results. Several of these simulation systems are run from the command line. Most larger systems require high performance clusters or other hardware to run large-scale simulations and significant disk space for storing data. These factors can limit a wide range of users, particularly those not part of prominent universities or companies with significant computational resources. However, as described in this paper, these limitations can largely be obviated by providing modeling and simulation as a service (MSaaS).

We and our teammates are currently developing a cyberinfrastructure (CI) for network science called *net.science* [10]. "A *cyberinfrastructure* consists of computing systems, data storage systems, advanced instruments, data repositories, and visualization environments, all linked by high-speed networks to facilitate scholarly innovation and discoveries not otherwise possible" [11]. Further conceptual details regarding CIs are given in [12], [13]. The *net.science* CI is operational, and users can register for accounts and use the system free of charge. One of its components is Network Simulation as a Service (NetSimS). This component, which provides MSaaS, is described in this work.

B. Contributions

1. Open access web-based system for performing contagion dynamics simulations on networked populations. The three major computational subsystems of NetSimS are (i) simulation, (ii) data analysis, and (iii) data visualization.

Fig. 1: The major computational subsystems of NetSimS.

Note that these operations don't form a simple pipeline, since for a given simulation, many data analyses and plots maybe generated at user discretion. NetSimS is a major system within *net.science*, which provides CI-wide functionality such as data management and annotation (e.g., for provenance), control flow, job submission and monitoring, data storage, and file search, per the CI definition in Section I-A. In addition, a web application (web app) has been built that provides users with forms to input data and view outputs (data files, tables, and visualizations). The web app provides an intuitive interface even for users without programming experience. In this paper, we confine our scope to the NetSimS system and to those elements of *net.science* functionality that are specific to NetSimS, such as the web app screens for configuring a simulation, data analysis, and visualization parameters. See Section III and Figure 2. Furthermore, NetSimS without the web app is built according to the MSaaS paradigm because all NetSimS functionalities are accessible through system application programming interfaces (APIs). In fact, the web app uses those APIs to submit service requests for simulations and to retrieve files. These APIs and their direct usage will be presented in an expanded version of the paper.

2. Software module for specifying simulation initial conditions. The three computational subsystems of NetSimS are shown in Figure 1. The simulation subsystem is the largest and most complex. Its two major components are the simulation engine and the module for specifying the initial conditions of a simulation. The simulation engine Contagion Simulation on NETworks (CSonNET) has been covered in [14]. Hence, our contribution and focus here are on the new module for specifying initial conditions for simulations. Most simulation tools use simple seeding methods or put the burden on the user to generate seeding (i.e., initial conditions) files.

Section III-A2 presents the system design, description of operations, and an algorithm for the initial conditions module. It is a generalized framework for initializing nodes to states at the start of each simulation instance. Initially activated nodes are called *seeds* or *seed nodes*. Eleven filtering parameters (metrics) are provided; currently, these are structural parameters of networks (e.g., node degree, k-shell, clustering coefficient), but can be expanded to include properties (e.g., labels) of nodes and edges. For each metric, a $[min, max]$ range can be specified from which candidate seed nodes are filtered. Nodes can be prioritized in ascending or descending order for each metric. For example, a user may want to select nodes with high in-degree centrality and low clustering coefficient. The module allows users to specify a combination of metrics through a configuration file, where each metric may also have an associated weight. A weighted sum of normalized metric values is used for final node selection and sampling. This final step can be configured to sample nodes uniformly at random or using the probability distribution derived from the weighted sum of metrics. There are also options for deterministic selection. All of these inputs are specified through web app forms, and backend files are generated automatically. See Section III-A for details.

3. Performance evaluation. We present performance data for the simulation subsystem, and in particular, for the CSonNET simulation engine and the initial conditions module. The simulation system scales to networks with 3.5 million nodes on our Rivanna high performance computing (HPC) cluster with Intel 40-core compute nodes. These are denoted herein as large networks. Specifically, we provide strong scaling results for the simulation engine and timing data for the initial conditions module. For the latter, the metric computation times for all nodes may be significantly greater than or significantly less than the time to filter nodes for seeding. See Section IV. Performance results are hardware dependent (so even larger networks can be analyzed with suitable hardware); our subsystems will run on several operating systems since our codes are Python-based.

4. Case studies. Two case studies are performed: one on a 2.65 million node social contact network of Seattle, Washington, and another on a synthetic 120 thousand node preferential attachment network. These simulations and results evaluate the effects of seeding methods, filtering parameter values for seeding, numbers of seeds, and model parameters. Results change significantly with seeding parameters, justifying the construction of a robust simulation seeding module. These studies, presented in Section V, demonstrate the value of NetSimS to quantify input parameter regimes where results change most rapidly.

A note on contagion models: In subsequent sections, we consider several well-known models of contagion propagation in networks. Here, for space reasons, we limit ourselves to brief accounts of these models. In the **threshold** model [15], each node v has a threshold $\theta(v)$, a non-negative integer. Node v changes to the activated state when at least $\theta(v)$ of its neighbors are also in the activated state. In the **susceptible-infected-recovered** (SIR) model, each node is in one of the states from $\{\mathrm{S}, \mathrm{I}, \mathrm{R}\}$, and each edge e has a transmission probability $p(e)$. Details regarding the stochastic process that governs how a node in state S is infected by a neighbor in state I and how nodes change from I to R are presented in many standard references (e.g., [16]). A variant of the SIR model is the SEIR model, where there is an additional state E (for "exposed") between S and I states, and the transitions between the states are also stochastic [16].

II. Related Work

A variety of software tools are available for agent-based simulations on networks. Some are command line tools, while

others have graphical user interfaces. Many of these can operate as services. We address several of these categories.

A. Simulation Systems

These systems are typically run from the command line, using HPC hardware. Some use distributed computing, while others use multithreading. Examples of high performance simulation frameworks include RePast HPC [17], Swarm [18], MASON [19], Flame [20], NDlib [21], Rensselaer's Optimistic Simulation System (ROSS) [22], and AnyLogic [23]. A data-intensive simulation framework is discussed in [24]. Reviews of agent-based simulations (ABS) can be found in [25], [26].

B. Simulations with IDEs

At the other end of the spectrum are NetLogo [27] and Repast Symphony [28]. Although these are not necessarily geared for HPC and large populations, they are powerful and provide intuitive modeling environments with integrated development environments (IDEs), high-level abstractions for model building, and displays. Among their distinguishing features are ease of use and ease of learning. Like the other simulators mentioned thus far, these are locally built and run on user-provided resources.

C. Modeling and Simulation as a Service

Advantages of MSaaS were identified in Section I-A. More comprehensive discussion on MSaaS including architectural aspects, state of the art, and comparative analyses can be found in [29], [30], and [31]. A client-side web app and a crowd movement simulation (e.g., a crowd exiting a building) system are described in [32]. Simulations that use the μsik kernel [33] in conjunction with cloud computing are discussed in [34]. Another cloud-based simulation service [35] has been proposed to make large-scale urban simulations available to the general public. An early work [36] focuses on taking advantage of concurrent processing in the cloud. A service-oriented architecture suitable for military and defense applications is discussed in [37]. Taylor et al. [38] discuss a cloud-based simulation platform that can host a variety of industrial simulation applications. A large-scale multi-agent system simulation service is described in [39]. MSaaS has also been used for high school STEM education [40] to demonstrate traffic simulations. Our system, which is focused on carrying out contagion simulations on an underlying network, can run in both cloud and cluster environments.

D. Seeding Methods in Other Simulators

NDlib [21] handles initial conditions (i.e., seeding) in one of two ways: (i) a number of nodes that are initially activated can be specified, and these nodes are selected by NDlib by choosing them uniformly at random, or (ii) an initial conditions file is user-generated, where the initial state of each node is listed. Other simulators also follow these two common approaches. NetLogo [27] supports uniform random sampling of agents to allocate states. Flame [20] requires users to provide initial states in an XML file. Agent-based simulation tools like MASON [19] provide methods to define and initialize agents, leaving it up to the user to configure initial conditions.

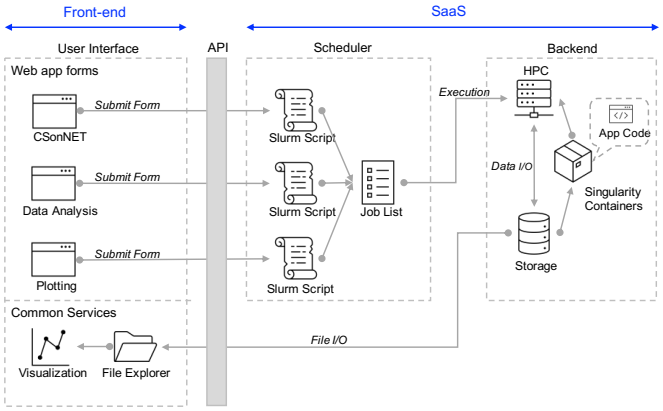


Fig. 2: The operational architecture of NetSimS system components: CSonNET, Data Analysis, and Plotting within the *net.science* CI. The underlying framework of *net.science* provides features to enable job submission, file I/O, execution of containerized code, data management, and storage.

III. NetSimS Software System

Figure 2 depicts the structure of the NetSimS system within the *net.science* CI. The diagram identifies several key subsystems, components, and services. The front-end contains components a user interacts with through web page forms, data tables, and visualizations. Jobs are constructed from user inputs and submitted to a Slurm scheduler that controls the job execution environment. Currently, all jobs run on an HPC cluster of multicore compute nodes, but other hardware can be supported (e.g., GPUs, cloud services). The system extracts from storage the relevant files, including input data and containerized code, and runs jobs from temporary workspaces allocated for those jobs. For NetSimS, containerized software includes the simulation, data analysis, and visualization subsystems—these three subsystems are addressed below. Results of various types are moved into storage upon job completion and data management operations are performed (e.g., database table updates for file searching and provenance).

Within the front end, there are many other forms such as those for displaying contents of data files, file metadata, visualizations of graphs, and job history with links to input and output files. But many of these capabilities are part of *net.science* system-wide functionality that many components (not just NetSimS) use. Here, our focus is on NetSimS.

A. Simulation Subsystem

This subsystem consists of the following components.

1) Web App: The graphical interface of the web app in *net.science* is shown in Figure 3. It contains sections to specify a network, select a contagion model and enter contagion model parameters, initial conditions, simulation parameters, and output filename. A graph file (e.g., edge list) is specified as an input. One of several contagion model classes can

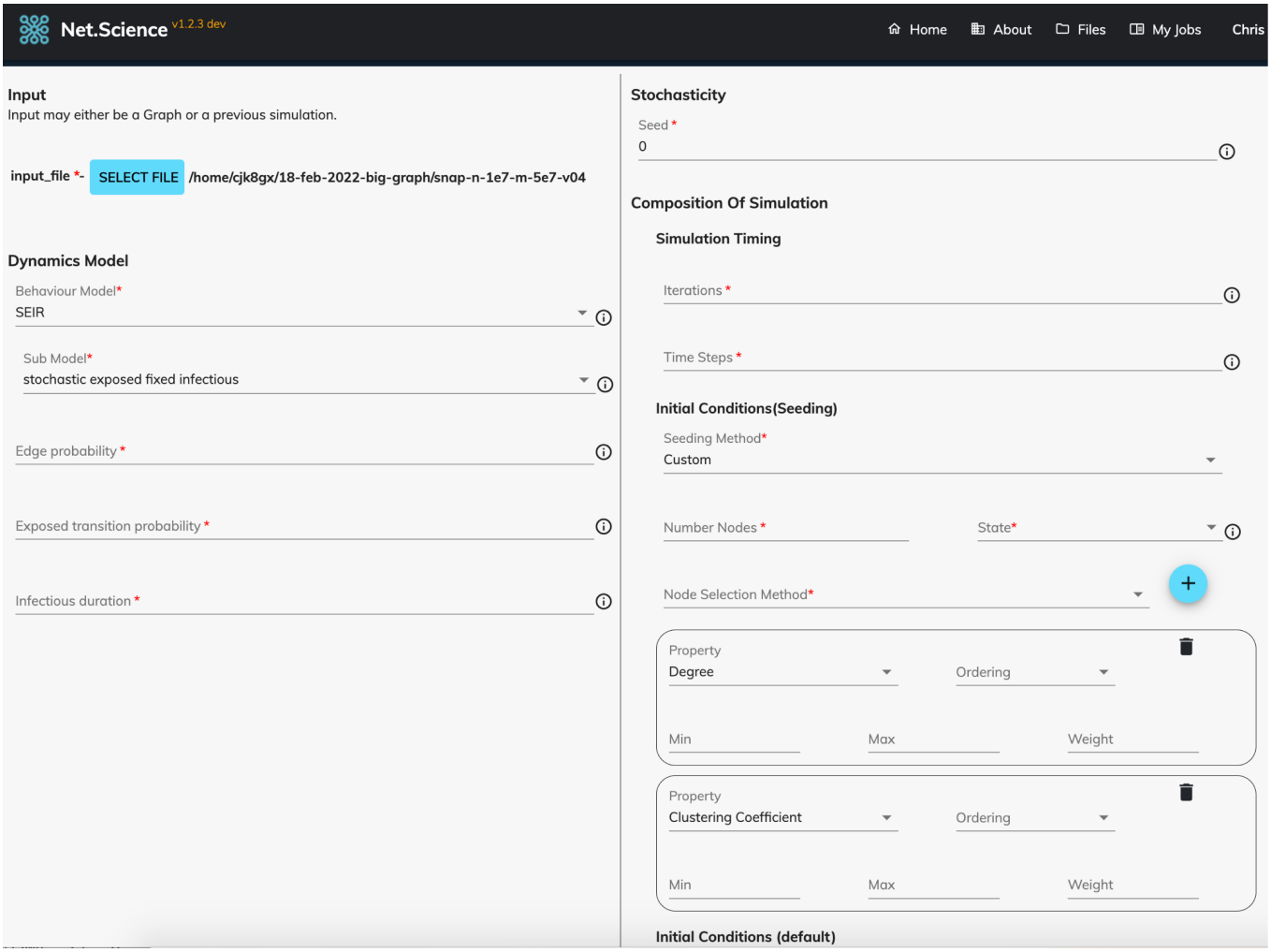


Fig. 3: The web interface of *net.science* depicting the input form in NetSimS for specifying simulation input parameters. A few parameters are not shown owing to space limits.

be selected (e.g., SIR, SEIR, SIS, and threshold models), along with submodels from each model class, and model parameters vary depending on the model. Input fields for properties of one SEIR model variant are shown in Figure 3. The number of iterations (i.e., simulation instances) to run and the duration (time steps) of each iteration are specified as a part of the simulation parameter set. Initial condition files are generated by adding metric-based configurations to filter and prioritize nodes. Figure 3 shows filtering based on node degree and clustering coefficient, as one example. These nodes are eventually sampled and assigned a particular activated state (e.g., I) alongside their default states (e.g., S).

We now discuss the two primary modules that compute initial conditions and execute simulations, and hence perform the operations specified by a user in Figure 3.

2) Initial Conditions Module: The graph-seeding component is responsible for generating initial conditions. In CSonNET, the initial condition is a complete list of nodes with their assigned states for each iteration prior to simulation runs. An operational diagram of the graph seeding module is shown in Figure 4. The Seed Set Generator executable, run at the backend of Figure 2, takes three categories of inputs: (i) required and optional parameters in the form of command line arguments (CLA), (ii) a configuration file that defines metrics and filtering, and (iii) a graph file used to compute metrics and generate seed node sets. For this discussion, let V and E denote the sets of nodes and edges in the graph $G(V, E)$, and $n = |V|$ denote the total number of nodes. The CLAs specify all input and output file paths, the number of iterations, default and activated state symbols, sampling method, etc. The configuration file is used to specify graph metrics (e.g., degree, betweenness centrality, authority/hub scores, clustering coefficient, eigenvector centrality, pagerank) and their upper and lower limits to filter nodes. For a metric with index m, the m^{th} entry (line) in the configuration file specifies its upper (k^m_{upper}) and lower (k^m_{lower}) limits, sorting order, and metric weight, w_m. An arbitrary number of metrics can be specified by adding entries to the configuration file to incorporate those measures in seed node selection.

We now formally describe the methodology that uses these

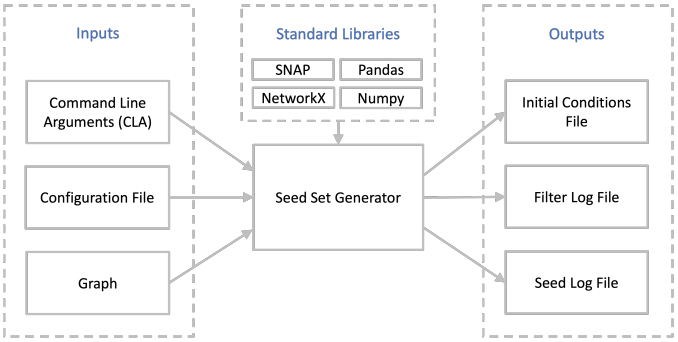


Fig. 4: Operational diagram of the graph-seeding module that generates initial conditions for contagion simulation on networks. The module uses either SNAP [41] or NetworkX [42] libraries to compute graph properties, depending on the metrics given in the configuration file. The command line arguments provide paths to all input and output files and other parameters.

metrics to select the seed sets. Let M be the total number of metrics specified in the configuration file. Let $k_{v_i}^m$ denote the value of metric m for node $v_i \in V$. For each metric m, the seed generator uses the appropriate graph analysis function and computes the values for all nodes. Once the $k_{v_i}^m$ values are computed for all $v_i \in V$ and $m \in \{1, 2, \ldots, M\}$, nodes are filtered as follows. Let

$$V^m = \{v_i | v_i \in V \text{ and } \; k_{lower}^m \leq k_{v_i}^m \leq k_{upper}^m\}. \quad (1)$$

Here, V^m is the set of remaining nodes after filtering based on the m^{th} metric. After completing M filtering steps, the final node set is given by

$$U = \bigcap_{m=1}^{M} V^m \,. \quad (2)$$

For each metric, the values are normalized to be in the range $[0, 1]$ across all nodes to ensure fair participation of that measure in final node selection. The normalization is performed as follows:

$$\begin{aligned} k_{max}^m &= \max(k_{v_1}^m, k_{v_2}^m, ..., k_{v_n}^m) \,, \\ k_{min}^m &= \min(k_{v_1}^m, k_{v_2}^m, ..., k_{v_n}^m) \,, \quad \text{and} \\ q_{v_i}^m &= \frac{k_{v_i}^m - k_{min}^m}{k_{max}^m - k_{min}^m} \,. \end{aligned} \quad (3)$$

Here, $q_{v_i}^m$ is the normalized quality score of node v_i for metric m. Metrics are usually specified to be used in descending order of normalized scores (e.g., prioritizing high-degree nodes). If a user wants nodes to be prioritized in an ascending order for a metric (e.g., pick nodes with low clustering coefficients), the quality scores for that metric are complemented for all nodes $v_i \in V$:

$$q_{v_i}^m = 1 - q_{v_i}^m \,. \quad (4)$$

Once quality scores for all the required metrics have been computed for all the nodes, a weighted sum determines the combined score Q_{v_i} for each node $v_i \in V$:

$$Q_{v_i} = \sum_{m=1}^{M} w_i q_{v_i}^m \,. \quad (5)$$

The filtered subset of nodes U, computed in Equation (2), is now available for sampling. A probability distribution is computed from the combined scores as follows:

$$p(u_i) = \frac{Q_{u_i}}{\sum_{u_i \in U} Q_{u_i}} \,. \quad (6)$$

The final set of seed nodes is obtained by randomly sampling U using this probability distribution. A pseudocode description of our method of generating initial conditions is given in Algorithm 1.

Algorithm 1: Initial Condition Generation

1 **Inputs** (1) A graph $G = (V, E)$, (2) Configuration file with M entries (i.e., metrics), and (3) Parameters related to file paths, input graph type, sampling method, iterations, etc.

2 **Output** Initial condition file

3 **Steps:**

A. Read command line argument parameters. Let R be the number of iterations and let n_A be the number of nodes to be activated in each iteration.
B. Read the graph into program memory.
C. Read the configuration file. Let M be the no. of entries.
D. **if** $M \neq 0$ **then**
1. **for** $m = 1$ **to** M **do**:
i. Compute metric values $k_{v_i}^m$ specified by entry m.
ii. Generate filtered node subsets V^m using Equation (1).
iii. Compute normalized node score $q_{v_i}^m$ using Equations (3) and/or (4).
2. Compute the combined node quality metric Q_{v_i} using Equation (5).
3. Obtain node set U for sampling using Equation (2).
4. Compute the probability distribution $p(u_i)$ from Equation (6).
5. **for** $r = 1$ **to** R **do**:
i. Initialize each node in V to the default state.
ii. Sample n_A nodes using distribution $p(u_i)$ and assign the activated state to those nodes.
iii. Append node list and their states to the output file for iteration r.
E. **else**
1. **for** $r = 1$ **to** R **do**:
i. Initialize each node in V to the default state.
ii. Sample n_A nodes uniformly randomly from V and assign the activated state to those nodes.
iii. Append node list and their states to the output file for iteration r.

3) Computational Simulation Engine: The simulation engine is an agent-based simulation (ABS) *framework*, meaning that it is explicitly designed to enable quick, surgical insertion and validation of new contagion models. File input/output (I/O), data structures, a general API for adding new models, simulation control flow, concurrency, and optimizations for controlling simulations are part of the framework, and switches in a simulation configuration file enable customization of a simulation. The Python implementation is a discrete-time simulation framework that provides concurrency by forking processes: one simulation instance of a collection of instances is assigned to one forked process. (A simulation typically

consists of multiple instances, where each instance starts at time $t = 0$ and steps through time up to a maximum time, whereupon the instance is completed. Multiple instances are typically required, e.g., they can incorporate different seed node sets (i.e., initial conditions), or the same seed set if the contagion model is stochastic.) At this time, 15 contagion models have been exposed to users through the web app; more will follow. Owing to space reasons, we refer the reader to [14] for more details on the simulation system.

B. Data Analysis Subsystem

The data analysis subsystem performs computations on simulation output and produces data files whose contents can be directly visualized. Currently, the module takes as input a raw simulation output file, a specified state of a contagion model, and analysis type and produces time history data for the specified state for all the nodes and simulation instances (i.e., iterations). Three analysis types can be specified, namely (i) new, (ii) cumulative, and (iii) current. These three types compute respectively the number of new, cumulative, and current nodes in the specified state at each time step. Often, this code is run multiple times for one simulation since the code is executed once for each analysis type.

C. Visualization Subsystem

The goal of the visualization subsystem is to produce customized, publication-quality plots of simulation results. The input to the plotting module is the output from the data analysis module. A user can customize x- and y-axis labels, tick labels, the ranges on each axis, type of axis (i.e., linear or logarithmic), and font sizes. One can also specify the text for curves in the legend along with the color and opacity for each curve. The user can select any/all of the plot types: (i) error bar plot (where error bars are displayed on data), (ii) line plot (curves only; no data points), (iii) scatter plot (data only, no curves), and (iv) bar chart. The plotting module can produce output files in .pdf, .ps, .png, .eps, or .svg formats.

Figure 5 shows a portion of the web app form for plotting. Plot types are shown on the left, and an expandable set of legend text (one for each curve) is shown at the right. X-axis configurations are shown at the bottom right; commensurate inputs for the y-axis are not shown for space reasons.

IV. Performance Evaluation

A. Networks Used in Studies

Details regarding the networks used in our studies are provided in Table I. These vary over five orders of magnitude in the number of nodes and edges. The Jazz network is a graph of musicians [43]. The Wiki and Slashdot are taken from [44]. The PA-120k and PA-500k networks are synthetic preferential attachment (PA) networks [45] and were generated in *net.science*. The Seattle networks are social contact networks, where each node is a person, and each edge means that the two nodes are in contact (i.e., are co-located) at overlapping times. These networks are generated according to the procedures in [46]. The suffix (18-75) means that only people between the ages of 18 and 75, inclusive, are part of the network; the network without the suffix contains all persons.

TABLE I: Networks used for analysis. There are mined (M), synthetic (S), or constructed (C) networks. The *net.science* cyber-infrastructure [10] was used to construct the two synthetic graphs and to generate all network properties.

Network	Structural Properties				
	Num. Nodes	*Num. Edges*	*Avg. Degree*	*Max. Degree*	*Largest k-core*
Jazz, M	198	2,742	27.7	100	29
Wiki, M	7,066	100,736	28.51	1,065	53
Slashdot, M	77,360	469,180	12.13	2,539	54
PA-120k, S	120,000	2.4 M	39.99	2,686	20
PA-500k, S	500,000	9.99 M	39.99	5,405	20
Seattle (18-75), C	2.56 M	40.34 M	31.49	664	42
Seattle, C	3.52 M	66.51 M	37.82	879	43

B. Initial Conditions Computation

The seeding module that computes the initial conditions for a simulation (i.e., assigns initial states to nodes) is evaluated over the networks listed in Table I. Networks are evaluated under three distinct scenarios, listed in Table II. The most basic scenario, called *Uniform*, does not use any metrics or filtering. In this case, all nodes are available for generating the initial condition file. The nodes are sampled uniformly at random. The second scenario, *Deg-Clust* uses a combination of two metrics: node degree and node clustering coefficient. For each metric, nodes whose values are *outside* the range $[\min, \max]$ (specified in the configuration file) are filtered out. The remaining nodes are available for sampling, which is done using a node probability distribution derived from the node score (as discussed in Section III-A2). The third scenario, *Deg-Betw* is similar to the second scenario, except that it uses node betweenness centrality as the second metric.

TABLE II: Graph seeding scenarios for initial condition (IC) file generation. Under "Filter Method," nodes that have metric values within the range $[\min, \max]$ are kept as candidate nodes for seeding. Values for $\min$ and $\max$ are network-dependent and are not specified here for lack of space.

Scenario	Metrics	Filter Method	Sampling
Uniform	None	None	Uniform Random
Deg-Clust	Node Degree Clustering Coeff.	$[\min, \max]$ $[\min, \max]$	Node Probability
Deg-Betw	Node Degree Betweenness Cent.	$[\min, \max]$ $[\min, \max]$	Node Probability

The performance analysis results are shown in Figure 6. The bars indicate the total time for generating initial condition files for simulations. The total time is the sum of the times required for metric computations, node filtering, node sampling, and writing the output file. The breakdown of the time values for a few selected networks under three different scenarios are shown in Table III. For large networks, some of the analysis results for Deg-Betw scenarios are missing in Figure 6. This is due to the fact that the computations of betweenness centrality could not be finished within reasonable times for such large graphs. This is typical for large networks and betweenness

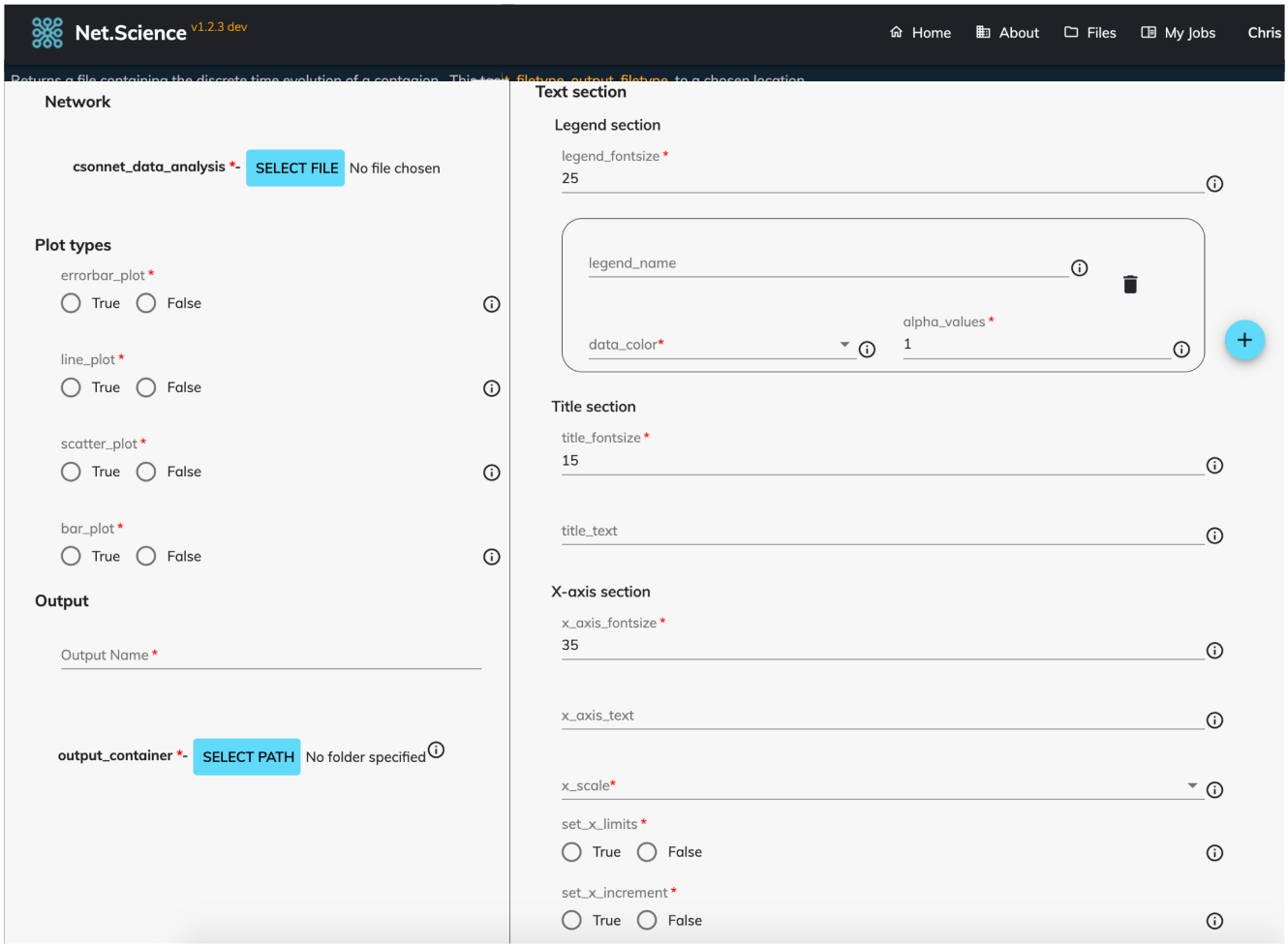


Fig. 5: The web interface of *net.science* depicting the input form in NetSimS for specifying inputs for simulation results plots. Some inputs on the form continue at the bottom of this snapshot.

centrality, since the running time for this centrality measure is $O(|V|\,|E|)$ [47]. Another use of these computations beyond providing the specified running times is to identify combinations of parameters that are particularly onerous to compute.

TABLE III: Seeding performance analysis on selected networks. Here, NA denotes "not applicable." (Uniform sampling scenarios do not require metric computation or filtering.)

Network	**Scenario**	**Computation Times (seconds)**			
		Metric	*Filtering*	*Sampling*	*Total*
Wiki	Uniform	NA	NA	0.80	0.80
Wiki	Deg-Clust	0.38	0.34	0.81	1.53
Wiki	Deg-Betw	81.58	0.34	0.80	82.73
PA-120k	Uniform	NA	NA	14.15	14.15
PA-120k	Deg-Clust	8.10	87.76	12.13	107.99
PA-120k	Deg-Betw	64,260	86.25	11.87	64,358
Seattle	Uniform	NA	NA	407.44	407.44
Seattle	Deg-Clust	202.76	84,783	399.52	85,385

C. Agent-Based Simulations

Figure 7 contains strong scaling timing results for simulations on five networks. All simulations use an SIR model. The transmission (edge) probability of a susceptible node v_i being infected at time $(t+1)$ by an infected neighbor v_j is $\Pr(s_i^{t+1} = \mathrm{I} \,|\, s_j^t = \mathrm{I}\,,\, s_i^t = \mathrm{S}) = 0.004$. A node is infectious (i.e., in state I) for four days; it then transitions to state R. Results are the total time to run 100 iterations of 100 time steps (i.e., days) on the specified networks. Each data point is the average of ten measured values. Initial conditions are that 300 nodes are initially in state I (those nodes with the greatest degree) and the remaining nodes are in state S. For these conditions, roughly 0.50 to 1.0 fraction of nodes across the networks get infected in the simulations. The linear nature of the data, on the log-log plot in Figure 7, demonstrates that CSonNET exhibits strong scaling.

V. Case Studies

Two case studies are performed using NetSimS. The first case study uses the threshold model and explores the effects

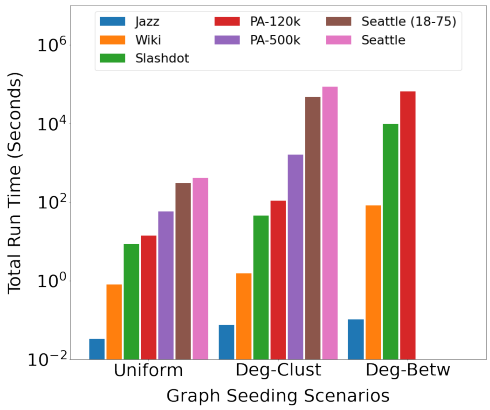


Fig. 6: A comparison of computational times for generating initial condition files across networks listed in Table I under the three seeding scenarios listed in Table II. The total run time consists of the times to compute metrics, filter nodes, sample seed nodes, and write output files. Each bar is the average of five runs. A few of these cases are elaborated in Table III. For the scenario Deg-Betw, some data are missing for very large networks. This is due to the long computation times required by the betweenness centrality metric.

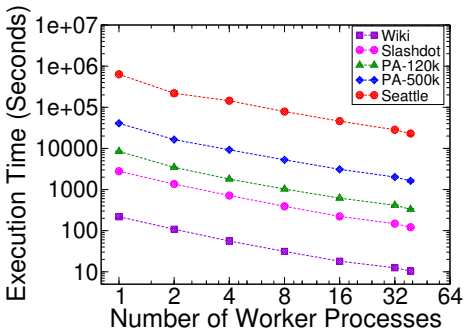


Fig. 7: Strong scaling results for CSonNET for five networks. The approximately straight lines in the plot indicate that the simulation code exhibits strong scaling on graphs up to millions of nodes.

of seeding method, number of seed nodes, and threshold values on the spread of a social contagion, such as a protest event [48], or joining an online platform [49]. The second case study uses an SIR model [50] to assess the effect of seed node selection methods and seed set sizes, in spreading a virus.

A. Case Study 1: Simulation of Social Protests

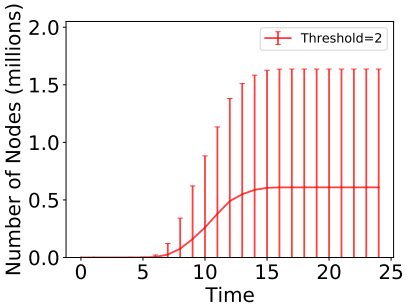
The spread of a contagion that represents participating in a protest is simulated on the Seattle (18-75) network of Table I, where individuals (graph nodes) are those people between 18 and 75 years, and edges represent social (face-to-face) contacts. The contagion spread is modeled with the Granovetter threshold model [15]. All nodes are assigned the same threshold θ and values range from one (simple contagion) to eight (complex contagion) [51]. Initial conditions for simulations are that 50 to 50,000 nodes (in powers of 10) are chosen in two distinct ways: i) sampled uniformly at random from all nodes and ii) filtered nodes with clustering coefficient in the range [0.05, 1.0] (this range avoids degree-1 nodes) followed by random sampling using the probability distribution derived from the node clustering coefficient. These nodes are assigned the activated state (state 1), which indicates that a node (person) is participating in a protest. The remaining nodes are initially assigned the inactive or non-participating state 0. A node changes from state 0 to state 1 when at least θ of its neighbors are already in state 1. Over 40 simulations were performed by systematically varying the numbers of seed nodes, the compositions of seed node sets, and node thresholds. Each simulation consisted of 50 simulation instances (i.e., iterations), and the duration of each instance was 30 time steps (days).

Figures 8a and 8b show results generated with NetSimS. Figure 8a shows time history data of the cumulative number of activated nodes for a simulation with $n_s = 50$ randomly selected seed nodes. The error bars represent $\pm$ one standard deviation from the mean curve generated from 50 iterations. If we take the final average value from Figures 8a for those input conditions, and repeat this process for different seeding methods and values of n_s and θ, then the plot of Figure 8b can be generated. The results show a precipitous reduction in the final percentage of activated nodes with small increases in threshold, except for very large seed set sizes. Generally, for a given set of conditions, the percentage of activated nodes is greater for seed nodes chosen randomly from all nodes [denoted "(U)"in the legend] than for random seeding of nodes whose clustering coefficient is in the aforementioned range [denoted "(C)" in the legend]. Systems like NetSimS are valuable because they can identify and quantify these sensitivities of results to inputs.

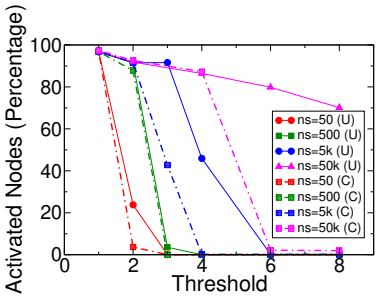
B. Case Study 2: Simulation of Virus Transmission

An SIR model in NetSimS is used to compute the attack size (i.e., numbers and fractions of infected nodes) in the PA-120k network of Table I. The probability of transmission from an infected node v_i to a susceptible node v_j, along undirected edge $\{v_i, v_j\}$, is given by $\Pr(\mathrm{I} \,|\, \mathrm{S}) = 0.002$. The infectious duration for each infected node is 4 days (time steps). Fifty simulation instances were computed in single-day increments, each from time $t = 0$ through 30 days.

Seed nodes, i.e., nodes initially in the infected state I, are chosen based on node degree in two steps. First, a subset of nodes is selected (filtered) based on a $[min, max]$ range specified in the configuration file. Next, nodes are sampled randomly based on probabilities derived from node scores computed from degrees. We run different schemes of node filtering. The first scheme (A) filters nodes with degree values in the range $[20, 30]$. The second scheme (B) filters nodes with degree values in the range $[80, 250]$. The number n_s of seed nodes is varied across simulations, using eight values of n_s, from 50 to 5,000 nodes. Nodes in the graph that are not seed nodes are assigned the susceptible state S.

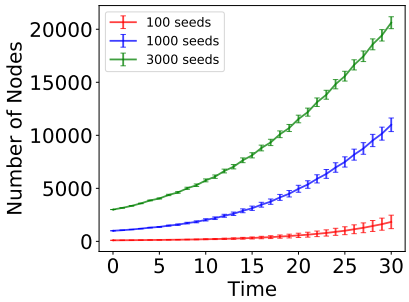


(a)

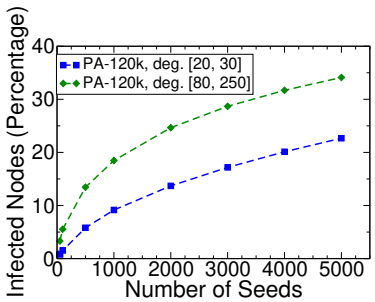


(b)

Fig. 8: (a) Visualization using plotting module of NetSimS. Cumulative activation curve for the Seattle (18-75) graph, for a simulation using a deterministic threshold model ($\theta = 2$) with $n_s = 50$ randomly activated seed nodes per iteration. Simulation duration is 30 days. Error bars represent one standard deviation from the average curve over 50 iterations. (b) Final fraction of agents (i.e., nodes in the Seattle (18-75) graph) participating in a protest as functions of seeding method, numbers of seeds, and thresholds of agents. In the legend, results with seed nodes sampled from all nodes are denoted "(U)" while those with randomly chosen seed nodes from those with clustering coefficient in the range [0.05, 1.0] are denoted "(C)".



(a)



(b)

Fig. 9: (a) Cumulative infection curves for an SIR model run on the PA-120k network for three values of n_s, per the legend. All seed nodes were selected from the nodes with degrees in the range [20, 30]. The duration of the simulations is 30 days (time steps). Each curve presents the average over 50 iterations, with error bars as $\pm$ one standard deviation. (b) Final fraction of nodes in the PA-120k graph infected after 30 days from the spread of a virus using an SIR model in agent-based simulations performed with NetSimS. The number of seed nodes and the method used to select seed nodes were varied across a set of simulations.

Figure 9a displays the cumulative number of infected nodes as a function of time in days, averaged over all 50 iterations, with a $\pm$ one standard deviation error bar at each day. All curves show the classic nonlinear increase in attack size with time. Data at $t = 30$ days for these conditions and others are used to produce the following result.

Figure 9b shows how the final fraction of infected agents (at $t = 30$ days) increases with the number of seed nodes for both seeding schemes. The second scheme (B) shows a greater percentage of infected agents compared to the first scheme (A) for all seeding configurations (50 to 5,000 seed nodes). This is expected since nodes with greater degrees are selected in the second scheme (B). The growth rate of the final fraction of infected nodes with respect to the number of seeds is nonlinear; final infected fractions grow most rapidly for smaller numbers of seeds. Again, we demonstrate the use of tools like NetSimS to identify and quantify interesting input parameter spaces and trends in simulation output.

VI. Summary

The simulation system NetSimS models the spread of contagions on populations represented as networks. It provides modeling and simulation as a service (MSaaS). Our contributions are listed in Section I-B. Our focus is on the use of the system through a web app, but the web app uses the same API as third party codes that make computationally demanding service requests (e.g., large problems, large numbers of simulation requests). The system is accessible through the freely available cyberinfrastructure (CI) called *net.science* [10]. One limitation of the system is that contagion models have to be explicitly added to both frontend and backend. Also, the sizes of networks used in simulations are limited by available memory and computational power. We continue to extend the system by adding new features. Directions for extension include adding more contagion models to the simulation framework, seeding methods to the initial conditions module, data analyses, and visualizations. We are investigating the parallelization of the Initial Conditions Module, e.g., by using NetworKit [52].

Acknowledgments: We thank the reviewers for providing helpful comments, and Research Computing at UVA for providing computational resources and technical support. This research is supported by University of Virginia Strategic Investment Fund award number SIF160, VDH grant VDH-21-501- 0135-1, and NSF Grants OAC-1916805 (CINES), CCF-

1918656 (Expeditions) and CMMI-1916670 (CRISP 2.0).

REFERENCES

[1] B. Prasse, M. A. Achterberg, L. Ma, and P. Van Mieghem, "Network-inference-based prediction of the COVID-19 epidemic outbreak in the Chinese province Hubei," *Applied Network Science*, vol. 5, no. 1, pp. 1–11, 2020.

[2] A. Rizzo, B. Pedalino, and M. Porfiri, "A network model for Ebola spreading," *Journal of theoretical biology*, vol. 394, pp. 212–222, 2016.

[3] A. Adiga, N. Palmer, Y. Y. Baek, H. Mortveit, and S. Ravi, "Network models and simulation analytics for multi-scale dynamics of biological invasions," *Frontiers in Big Data*, vol. 5, 2022.

[4] G. Chowell, J. M. Hyman, S. Eubank, and C. Castillo-Chavez, "Scaling laws for the movement of people between locations in a large city," *Phys. Rev. E*, vol. 68, p. 066102, Dec 2003.

[5] M. Cinelli, W. Quattrociocchi, A. Galeazzi, C. M. Valensise, E. Brugnoli, A. L. Schmidt, P. Zola, F. Zollo, and A. Scala, "The COVID-19 social media infodemic," *Scientific Reports*, vol. 10, no. 1, pp. 1–10, 2020.

[6] B. Doerr, M. Fouz, and T. Friedrich, "Why rumors spread so quickly in social networks," *Communications of the ACM*, vol. 55, no. 6, pp. 70–75, 2012.

[7] N. A. Christakis and J. H. Fowler, "Social contagion theory: examining dynamic social networks and human behavior," *Statistics in medicine*, vol. 32, no. 4, pp. 556–577, 2013.

[8] C. Porath and C. Pearson, "The price of incivility," *Harvard business review*, vol. 91, no. 1-2, pp. 114–121, 2013.

[9] P. Glasserman and H. P. Young, "Contagion in financial networks," *Journal of Economic Literature*, vol. 54, no. 3, pp. 779–831, 2016.

[10] N. K. Ahmed *et al.*, "net.science: A Cyberinfrastructure for Sustained Innovation in Network Science and Engineering," in *Gateway Conference*, 2020, pp. 71–74.

[11] C. A. Stewart, S. Simms, B. Plale *et al.*, "What is cyberinfrastructure," in *Proceedings of the 38th Annual ACM SIGUCCS Fall Conference: Navigation and Discovery*, 2010, p. 37–44.

[12] N. S. F. C. Council, "Cyberinfrastructure vision for 21st century discovery," 2007, TR.

[13] M. Welshons (Ed.), "Our cultural commonwealth: The report of the american council of learned societies commission on cyberinfrastructure for the humanities and social sciences," American Council of Learned Societies, Tech. Rep., 2006.

[14] J. Priest, A. Kishore, L. Machi *et al.*, "CSonNet: An agent-based modeling software system for discrete time simulation," in *Winter Simulation Conference (WSC)*, 2021, pp. 1003–1014.

[15] M. Granovetter, "Threshold models of collective behavior," *The American Journal of Sociology*, vol. 83, no. 6, pp. 1420–1443, 1978.

[16] D. Easley and J. Kleinberg, *Networks, Crowds and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

[17] N. Collier and M. North, "Parallel agent-based simulation with repast for high performance computing," *SIMULATION*, vol. 89, no. 10, pp. 1215–1235, 2013.

[18] N. Minar, R. Burkhart, C. Langton, and M. Askenazi, "The Swarm simulation system: A toolkit for building multi-agent simulations," Santa Fe Institute, Tech. Rep. 96-06-042, 1996.

[19] S. Luke, C. Cioffi-Revilla, L. Panait *et al.*, "MASON: A Multi-Agent Simulation Environment," *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 82, pp. 517–527, 2005.

[20] M. Kiran, P. Richmond, M. Holcombe, L. S. Chin, D. Worth, and C. Greenough, "FLAME: Simulating Large Populations of Agents on Parallel Hardware Architectures," in *AAMAS*, 2010, pp. 1633–1636.

[21] G. Rossetti, L. Milli *et al.*, "NDlib: a Python library to model and analyze diffusion processes over complex networks," *International Journal of Data Science and Analytics*, vol. 5, no. 1, pp. 61–79, 2018.

[22] D. W. Bauer Jr., C. D. Carothers, and A. Holder, "Scalable time warp on blue gene supercomputers," in *Workshop on Principles of Advanced and Distributed Simulation*, 2009, pp. 35–44.

[23] J. Huang, L. Liu, and L. Shi, "Auction Policy Analysis: An Agent-Based Simulation Optimization Model of Grain Market," in *Winter Simulation Conference (WSC)*, 2016, pp. 3417–3428.

[24] P. Bhattacharya, S. Ekanayake, C. J. Kuhlman *et al.*, "The matrix: An agent-based modeling framework for data intensive simulations," in *AAMAS*, 2019, p. 1635–1643.

[25] R. J. Allan, *Survey of Agent Based Modelling and Simulation Tools*. Science & Technology Facilities Council New York, 2010.

[26] K. Kravari and N. Bassiliades, "A Survey of Agent Platforms," *Journal of Artificial Societies and Social Simulation*, pp. 1–18, 2015.

[27] S. Railsback, D. Ayllón, U. Berger *et al.*, "Improving execution speed of models implemented in netlogo," *Journal of Artificial Societies and Social Simulation*, vol. 20, no. 1, 2017.

[28] M. J. North, N. T. Collier, J. Ozik *et al.*, "Complex adaptive systems modeling with repast simphony," *Complex adaptive systems modeling*, vol. 1, no. 1, pp. 1–26, 2013.

[29] M. Shahin, M. A. Babar, and M. A. Chauhan, "Architectural design space for modelling and simulation as a service: a review," *Journal of Systems and Software*, vol. 170, p. 110752, 2020.

[30] E. Cayirci, "Modeling and simulation as a cloud service: a survey," in *2013 Winter Simulations Conference (WSC)*. IEEE, 2013, pp. 389–400.

[31] J. E. Hannay, T. van den Berg, S. Gallant, and K. Gupton, "Modeling and simulation as a service infrastructure capabilities for discovery, composition and execution of simulation services," *The Journal of Defense Modeling and Simulation*, vol. 18, no. 1, pp. 5–28, 2021.

[32] S. Wang and G. Wainer, "A simulation as a service methodology with application for crowd modeling, simulation and visualization," *Simulation*, vol. 91, no. 1, pp. 71–95, 2015.

[33] K. S. Perumalla, "μsik–a micro-kernel for parallel/ distributed simulation systems," in *PADS*, 2005.

[34] S. B. Yoginath and K. S. Perumalla, "Efficient parallel discrete event simulation on cloud/virtual machine platforms," *ACM TOMACS*, 2015.

[35] D. Zehe, A. Knoll, W. Cai, and H. Aydt, "SEMSim Cloud Service: Large-scale urban systems simulation in the cloud," *Simulation Modelling Practice and Theory*, vol. 58, pp. 157–171, 2015.

[36] M. Rak, A. Cuomo, and U. Villano, "Mjades: Concurrent simulation in the cloud," in *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*. IEEE, 2012, pp. 853–860.

[37] D. Procházka and J. Hodickỳ, "Modelling and simulation as a service and concept development and experimentation," in *International Conference on Military Technologies (ICMT)*. IEEE, 2017, pp. 721–727.

[38] S. J. Taylor, T. Kiss, A. Anagnostou, G. Terstyanszky, P. Kacsuk, J. Costes, and N. Fantini, "The cloudsme simulation platform and its applications: A generic multi-cloud platform for developing and executing commercial cloud-based simulations," *Future Generation Computer Systems*, vol. 88, pp. 524–539, 2018.

[39] C. Hüning, M. Adebahr, T. Thiel-Clemen, J. Dalski, U. Lenfers, and L. Grundmann, "Modeling & simulation as a service with the massive multi-agent system mars," in *Proceedings of the Agent-Directed Simulation Symposium*, 2016, pp. 1–8.

[40] F. Caglar, S. Shekhar, A. Gokhale *et al.*, "Cloud-hosted simulation-as-a-service for high school stem education," *Simulation Modelling Practice and Theory*, vol. 58, pp. 255–273, 2015.

[41] J. Leskovec and R. Sosič, "Snap: A general-purpose network analysis and graph-mining library," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 1, p. 1, 2016.

[42] 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 (SciPy2008)*, 2008, pp. 11–15.

[43] P. M. Gleiser and L. Danon, "Community structure in Jazz," *Advances in Complex Systems*, vol. 6, no. 4, pp. 565–573, 2003.

[44] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[45] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.

[46] C. L. Barrett, R. J. Beckman, M. Khan *et al.*, "Generation and analysis of large synthetic social contact networks," in *Winter Simulation Conference (WSC)*, 2009, pp. 1003–1014.

[47] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.

[48] S. Gonzalez-Bailon, J. Borge-Holthoefer, A. Rivero, and Y. Moreno, "The dynamics of protest recruitment through an online network," *Scientific Reports*, vol. 1, p. 7, 2011.

[49] D. Centola, "The spread of behavior in an online social network experiment," *Science*, vol. 329, pp. 1194–1197, 2010.

[50] H. W. Hethcote, "The mathematics of infectious diseases," *SIAM Review*, vol. 42, pp. 599–653, 2000.

[51] D. Centola and M. Macy, "Complex contagions and the weakness of long ties," *American J. of Sociology*, vol. 113, no. 3, pp. 702–734, 2007.

[52] C. Staudt, A. Sazonovs, and H. Meyerhenke, "NetworKit: A tool suite for large-scale complex network analysis," *Network Science*, vol. 4, pp. 508–530, 2016.