Providing High Performance Computing based Models as a Service: Architecture and Services for Modeling Contagions on Large Networked Populations

Sherif Hanie El Meligy Abdelhamid

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Chris J. Kuhlman, Co-Chair
Madhav V. Marathe, Co-Chair
Mohamed Kholief
Chris North
Sekharipuram S. Ravi

Tuesday 15th November, 2016
Blacksburg, Virginia

Keywords: Social Behavior, Contagions, Networks, Control of Contagion Processes, Graph Dynamical Systems, Modeling and Simulation, Open Science, Software Systems, Web Services

Providing High Performance Computing based Models as a Service: Architecture and Services for Modeling Contagions on Large Networked Populations

Sherif E. Abdelhamid

(ABSTRACT)

Network science emerged as an interdisciplinary field over the last 20 years, and played a central role to address fundamental problems in other fields, e.g., epidemiology, public health, and transportation, and is now part of most university curriculums. Network dynamics is a major area within network science where researchers study different forms of processes in networked populations, such as the spread of emotions, influence, opinions, flu, ebola, and mass movements. These processes often referred to individually and collectively as contagions. Contagions are increasingly studied because of their economic, social, and political impacts. Yet, resources for studying network dynamics are largely dispersed and stand-alone. Furthermore, many researchers interested in the study of networks are not computer scientists. As a result, they do not have easy access to computing and data resources. Even with the presence of software or tools, it is challenging to install, build, and maintain software. These challenges create a barrier for researchers and domain scientists. The goal of this work is the design and implementation of a research framework for modeling contagions on large networked populations. The framework consists of various systems and services that provide support for researchers and domain scientists at different stages of their research workflow.

Providing High Performance Computing based Models as a Service: Architecture and Services for Modeling Contagions on Large Networked Populations

Sherif E. Abdelhamid

(GENERAL AUDIENCE ABSTRACT)

Network science is a field which studies complex networks. Network science emerged over the last 20 years as an interdisciplinary academic field which integrates data, tools, and theories from multiple disciplines, and use this integration to address fundamental problems in other fields, e.g., epidemiology, public health, and transportation, and is now part of most university curriculums. Network dynamics is a major area within network science where researchers study different forms of processes in networked populations, such as the spread of emotions, influence, opinions, flu, ebola, and mass movements. These processes often referred to individually and collectively as contagions. Contagions are increasingly studied because of their economic, social, and political impacts. To study contagions, researchers and domain scientists need both software and hardware that can collect, analyze, and manage large volumes of networked data. Furthermore, for non-computer scientists, it is more challenging to install, build, and maintain the software with the required hardware. These challenges create a barrier for researchers and domain scientists to conduct their experiments and replicate others' work. The goal of this work is the design and implementation of various systems and services that provide support for researchers and domain scientists at different stages of their research workflow.

Dedication

I would like to dedicate my PhD dissertation to my lovely sons Yaseen and Yusuf for the joy they brought to my life. My wife Mona for her endless love, kindness, and support. My parents Eman and Hany, my brother Karim, and all my family in Egypt for encouraging and guiding me all the time.

Acknowledgments

I am very grateful to my advisor, Dr. Chris J. Kuhlman who helped me with my PhD research, taught me various professional skills, and was always motivating and encouraging. I thank My advisor, Prof. Madhav V. Marathe, who guided me throughout my research, and I am very grateful to him for providing such a stimulating and interactive environment at Network Dynamics and Simulation Science Laboratory (NDSSL). I would like to thank Prof. S. S. Ravi, Prof. Chris North, and Prof. Mohamed Kholief, as they worked closely with me on different aspects of my research, and provided their valuable advice and feedback. I would like to thank all my friends, colleagues, and NDSSL members for their friendship, advice, and support. At last but not the least, I would like to thank my parents, my wife, and all my family for their unconditional support and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Contagion can be any entity that can spread through a networked population. Understanding how contagions spread within a population has applications in many fields. Examples include biology [108, 120], (discrete) mathematics [98], economics [80], health sciences [188], computer networks [164], politics [191], and epidemiology [168]. Examples of emotional contagions include mood [113], anxiety [39], fear [103], appreciation [89], depression [169] and enjoyment [88]. Examples of behavior contagions include hysteric [124], rule violation (e.g. speeding [66], criminality [119], substance abuse [83]), deliberate self-harm, financial contagion, consumer fashions and fads, aggressive behavior [22], and revolutions [15].

Contagions impact individuals; e.g. drinking, smoking, and obesity which are the three behaviors that contribute most to chronic health problems, and all are greatly affected by peer influence; e.g., [110]. Contagions can also affect a society or an entire country; e.g. 1992 Los Angeles riots which resulted in 1 billion in damages, 54 deaths,

and thousands of injures [216]. The annual cost of obesity alone is estimated at $147 billion [87]. Contagions can even cross countries' borders as in the recent Arab Spring [15, 220]. See [127] for further details.

1.2 Challenges for Domain Scientists

Reasoning about these systems requires simulation of networks at scale, as described in [214]. Inputs to simulations are, in turn, increasing in fidelity: attributes of population members are being computed [75], and information about population dynamics (e.g., topics transmitted over social media [94, 194]) are being gathered, including the development of new behavior models [175, 203]. Data mining increasingly plays a central role in these endeavors where, for example, differences in communication patterns between genders and among age groups are being uncovered [75, 94]. Further, individuals may have multiple behaviors and roles in different contexts within the same application domain [211]. These considerations have imposed challenges for researchers and domain scientists. There is a need to learn and use different tools, programming languages, and technologies for different tasks. Simulations at scale require access to powerful computing resources (e.g. clusters). The increased size of networks and generation of big output data from simulations requires special tools and services for data management, analysis, and post-processing. In addition, limited collaboration and sharing of findings among scientists, can lead to difficulties in reproducing or replicating others' work.

1.3 Research Motivation and Objectives

The challenges mentioned above provide motivation to design, build and evaluate systems and services to enable domain experts and other users (including computer scientists) to more effectively study contagion dynamics on networked populations.

Our first objective is to bridge the gap between the mathematical theory of simulation (using GDS) and high performance computing (HPC) design and implementation.

In discussing the Open Services Gateway Initiative Framework (OSGi) and Cyberinfrastructure Shell (CiShell), Borner [48] mentioned the following:
"*My aim here is to inspire computer scientists to implement software frameworks that empower domain scientists to assemble their own continuously evolving [software], adding and upgrading existing (and removing obsolete) plug-ins to arrive at a set that is truly relevant for their work with little or no help from computer scientists.*"
This statement serves as one of the primary drivers of our own work.

The second objective is to provide an infrastructure that supports research replicability or reproducibility and promotes collaboration among scientists. Different studies have shown that many, if not most, research papers are not reproducible [47, 112, 116]. There is evidence of challenges in various fields (e.g. psychological science [156], cancer biology [38], computer science [65]). Studying contagion dynamics depends on the input data, settings for simulation parameters, and on the specification of the hardware where the simulations run. However, sometimes computational experiments including simulations are not clearly described; the data and code needed to reproduce the results might not be available; and researchers use proprietary software and advanced hardware.

The third objective is to provide the medium for reviewers and other scientists to verify the simulation findings by others. Gezelter [93] mentioned the following:

"*As systems become more complex and the data sets become too large to reproduce, calculations that are verifiable in principle are no longer verifiable in practice without public access to the code (or data).*"
This statement also drives our own work to provide a research framework where all the original computations and data reside.

1.4 Graph Dynamical Systems Foundations

Our modeling approach is to represent populations as graphs, where graph nodes represent agents and edges represent pairwise interactions. This approach is named graph dynamical systems (GDS). GDS are discrete dynamical systems (discrete in both time and states of nodes) and are sometimes referred to as generalized cellular automata.

1.4.1 GDS Overview

A GDS is denoted by $\mathcal{S}(G, F, \mathcal{W}, K)$. Let G denote a directed graph, called a **dependency graph**, with vertex set $\mathrm{v}[G] = \{1, 2, \ldots, n\}$ and edge set $\mathrm{e}[G]$. We use the convention that directed edge (u, v) means that the state of vertex u is used to determine the next state of vertex v. To each vertex v we assign a state $x_v \in K$ and refer to this as the **vertex state**; K is the **vertex state set**. The 1-neighborhood of a vertex v is the set of vertices adjacent to v in G. Let $n[v]$ denote the sequence of vertices in the 1-neighborhood of vertex v sorted in increasing order such that for each $u \in n[v]$, there exists a directed edge $(u, v) \in \mathrm{e}[G]$. (If $v \in n[v]$, meaning that there is a directed self-loop, then the 1-neighborhood is closed.) In other words, each such u is an in-neighbor of v, and $d^{in}(v) = |n[v]|$, where d^{in} is the in-degree of v. We write the

sequence $x[v]$ of vertex states corresponding to the vertices in $n[v]$ as

$$x[v] = (x_{n[v](1)}, x_{n[v](2)}, \ldots, x_{n[v](d^{in}(v))}) \ .$$

We refer to $x[v]$ as the **restricted state**. Here, $n[v](i)$ is the ith entry in the sequence. We call $x = (x_1, x_2, \ldots, x_n)$ the **(system) state**. We denote the (system) state and restricted state at time t as $x(t)$ and $x(t)[v]$, respectively.

The dynamics of changes in vertex states are governed by a sequence $F = (f_v)_{v=1}^{n}$ of **vertex functions** where each $f_v \colon K^{d^{in}(v)} \longrightarrow K$ maps as

$$x_v(t+1) = f_v\big(x(t)[v]\big) \ . \tag{1.1}$$

That is, the state of vertex v at time $t+1$ is given by f_v evaluated for the restricted state $x[v]$ at time t. To reduce notation, we will often omit the time t from the restricted state.

An **update scheme** $\mathcal{W}$ governs how the list of vertex functions assemble to a **graph dynamical system map** (see e.g. [149, 155])

$$\mathbf{F} \colon K^n \longrightarrow K^n$$

producing the system state at time $t+1$ from that at time t; i.e., $x(t+1) = \mathbf{F}(x(t))$.

We first address the **synchronous** and **sequential** update schemes. In the former case we have the synchronous (parallel) GDS map

$$\mathbf{F}(x_1, x_2, \ldots, x_n) = (f_1(x[1]), f_2(x[2]), \ldots, f_n(x[n])) \ . \tag{1.2}$$

We refer to this subclass of GDS as **synchronous dynamics systems** (SyDS), since all vertex functions are executed simultaneously (i.e., in parallel); it is sometimes referred

to as **generalized cellular automata**. In the latter case we consider permutation update sequences. We first introduce the notion of X**-local functions**. Here, the X-local function $F_v \colon K^n \longrightarrow K^n$ is given by

$$F_v(x_1, \ldots, x_n) = (x_1, \ldots, x_{v-1}, f_v(x[v]), x_{v+1}, \ldots, x_n) \; ;$$

i.e., F_v updates only the vth component of the system state. Using $\pi = (\pi_1, \ldots, \pi_n) \in S_X$ (the set of all permutations of v$[G]$) as an update sequence, the corresponding asynchronous (or sequential) GDS map $\mathbf{F}_\pi \colon K^n \longrightarrow K^n$ is given by

$$\mathbf{F}_\pi = F_{\pi_n} \circ F_{\pi_{n-1}} \circ \cdots \circ F_{\pi_2} \circ F_{\pi_1} \; ,$$

which is the composition of the X-local functions. We refer to this class of asynchronous systems as **(permutation) sequential dynamical systems** (SDS).

A generalization of the two previous update schemes is **block sequential**. In this scheme, the vertices are partitioned into a sequence of q sets or blocks $B = (B_1, B_2, \ldots, B_q)$. The vertex functions for the vertices in each block are executed simultaneously, with sequential ordering between consecutive blocks. Let $\pi_B = (\pi_{B_1}, \ldots, \pi_{B_q})$ be a block permutation. We have the X-local function, for $l \in \{1, \ldots, q\}$, $F_{\pi_{B_l}} \colon K^n \longrightarrow K^n$, where the ith entry in $F_{\pi_{B_l}}$ is the identity map if vertex $i \notin B_l$ and is f_i if vertex $i \in B_l$. We have the block sequential GDS map

$$\mathbf{F}_{\pi_B} = F_{\pi_{B_q}} \circ F_{\pi_{B_{q-1}}} \circ \cdots \circ F_{\pi_{B_2}} \circ F_{\pi_{B_1}} \; ,$$

and refer to it as a **block sequential dynamical system** (BSDS). When the size of each block is one, a block sequential GDS map reduces to a sequential GDS map, and when all vertices are in one block, the block sequential map reduces to a synchronous GDS map. We describe all three types of maps here because different works in the

literature may use only one of the update methods.

The **phase space** $\Gamma(\mathbf{F})$ of the GDS map $\mathbf{F}$ is a directed graph with vertex set K^n and edge set $\{(x, \mathbf{F}(x)) \mid x \in K^n\}$. A state x for which there exists a positive integer p such that $\mathbf{F}^p(x) = x$ is a **periodic point**, and the smallest such integer p is the **period** of x. If $p = 1$ we call x a **fixed point** of $\mathbf{F}$. A state that is not periodic is a **transient state**. Classically, the **omega-limit set** of x, denoted by $\omega(x)$, is the set of accumulation points of the sequence $\{\mathbf{F}^k(x)\}_{k\geq 0}$. In the finite case, the omega-limit set (also called a **(limit) cycle**, **(periodic) orbit**, **limit set**, or **attractor**) is the unique periodic orbit reached from x under $\mathbf{F}$.

Given two update sequences π and π', if $\mathbf{F}_\pi$ and $\mathbf{F}_{\pi'}$ give the exact same state transitions; i.e., $\mathbf{F}_\pi(x) = \mathbf{F}_{\pi'}(x)$ for every $x \in K^n$, then the GDS maps are equal; i.e., $\mathbf{F}_\pi = \mathbf{F}_{\pi'}$, and we say that the maps are **functionally equivalent**. If the limit cycle structures for the two maps are the same, to within an isomorphism, then we say the two maps are **cycle equivalent** [149]. Cycle equivalence describes long-term dynamics. If two maps are functionally equivalent, then they are cycle equivalent. Functional and cycle equivalence can be computed for any pair of GDSs, irrespective of update sequence.

A procedural description of GDS for computing forward trajectory (e.g., as done in simulations) is provided in Algorithm 1. This is a serial algorithm for the synchronous update scheme. Each simulation consists of a number of diffusion instances (simulation runs) R. The parameter d is the duration of each run (number of time steps). At the beginning of each run, the simulation parameters are reset to their initial values. At each time step t, the vertex function f_v for vertex v is executed. If the state of v changes, then the new state is stored in temporary storage and $temp_v$ is assigned to $x(t+1)_v$ after all vertex functions have been executed. The values in $temp_v$ are printed to output to record state changes. Note that lines 7 through 15 of Algorithm 1 references Equation 1.2. This process is repeated for R times (number of

runs).

Algorithm 1: Synchronous Update Scheme

Input: Graph G, Vertex properties p_v, Edge properties p_e, Number of simulation runs R, Simulation Duration d, Sets of seed vertices s_v, Sets of seed edges s_e
Output: Changes in each Vertex states over time t
$i = 0$; // Initialize the run number to 0
while $i < R$ **do**
  $t = 0$; // Initialize the time to 0 at the beginning of each run
  while $t < d$ **do**
    $resetNodeProperties(p_v)$; // Reset all the node properties
    $resetEdgeProperties(p_e)$; // Reset all the edge properties
    $applyInitialConditions(s_v, s_e)$; // Load the seed conditions
    $clearTemp_v()$; // To store vertex state changes at time t+1
    foreach $v \in G$ **do**
      $x(t+1)_v \longleftarrow f_v(v, p_v, p_e)$; // Execute the local vertex function of vertex v per Equation 1.1
      if $x(t)_v \neq x(t+1)_v$ **then**
        $temp_v \longleftarrow x(t+1)_v$; // Store the new state in temporary storage
    $t \longleftarrow t+1$
    $loadNodeStates()$; // Apply the new state changes (assign $temp_v$ to $x(t+1)_v$)
  $i \longleftarrow i+1$

1.4.2 Illustrative Examples

Example 1. *To illustrate the above concepts, take $X = Circle_4$ as the graph (shown in the left in Figure 1.1), $K = \{0, 1\}$, and choose thresholds $(k_{01}, k_{10}) = (1, 3)$ for each vertex. The up-threshold k_{01} denotes the minimum number of distance-1 neighbors of v that are required to be in state 1 in order for v to transition to 1 when its state is 0. When $x_v = 1$, if the number of neighbors of v in state 1 (including v) (including v itself) is $< k_{10}$, then v changes to state 0. Otherwise x_v does not change. This is a **bithreshold** system, a generalization of a threshold GDS [131], in which each vertex is assigned a bithreshold vertex function. Using the update sequence $\pi = (0, 1, 3, 2)$, we obtain the single state transition $\mathbf{F}_\pi(0, 1, 0, 1) = (1, 0, 0, 0)$. The phase space of $\mathbf{F}_\pi$ is shown in Figure 1.1. We see that $\mathbf{F}_\pi$ has two fixed points and two*

2-cycles. In this system, $(1,0,1,0)$ *is a transient state. With* $\mathbf{F}_\pi$*, 12 of 16 system states lead to fixed points.*

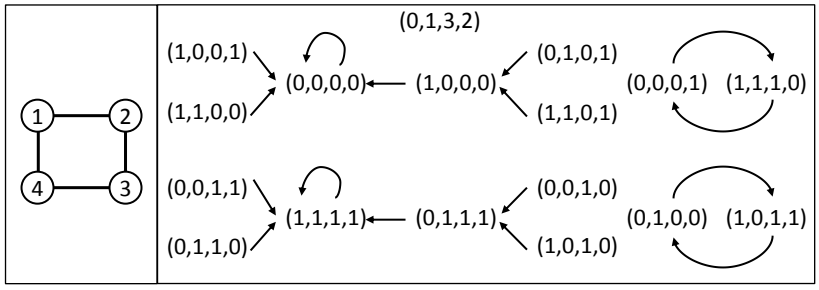


Figure 1.1. *The graph* $X = Circle_4$ *(left), and the phase spaces of* $\mathbf{F}_\pi$ *(right) with sequential update* $\pi = (0,1,3,2)$ *for Example 1.*

Example 2. *Take X as the 4-vertex graph on the left in Fig. 1.2, let* $K = \{0,1\}$*, and assign the nand vertex function to each vertex; we have nand* $= 1 + \prod_{u \in n[v]} x_u$*. The GDSs, in order from left to right in Fig. 1.2, are for synchronous, sequential (with* $\pi = (1,2,3,4)$*), and block sequential (with* $\pi_B = ([1,3],[2,4])$*) update methods. In the latter system,* $q = 2$*,* $B_1 = \{1,3\}$*, and* $B_2 = \{2,4\}$*. For the sequential GDS map, we have, for example,* $(1,0,1,1) = \mathbf{F}_\pi(0,1,1,0)$*. In all three GDSs, state* $(0,0,0,1)$ *is a transient state and state* $(1,1,1,1)$ *is a periodic point. By inspection, no two of these three GDSs are functionally equivalent because their phase spaces are different. No two GDSs are cycle equivalent because, from left, we have limit cycles of period (multiplicity): 2 (2); 2 (1) and 4 (1); and 2 (1) and 3 (1), respectively, for the maps. There are no fixed points in these GDSs, which is a characteristic of nand functions [155].*

Example 3. *Consider the* $n = 7$ *vertex graph G in Figure 1.3. We have* $K = \{0,1\}$ *with red (respectively, green) vertices indicating* $x_i = 0$ *(respectively,* $x_i = 1$*). These are the non-participating and participating states, respectively. Each vertex function* f_i*,* $1 \leq i \leq 7$*, is the progressive 2-threshold function. That is, if a vertex with* $x_i(t) = 0$ *has at least two*

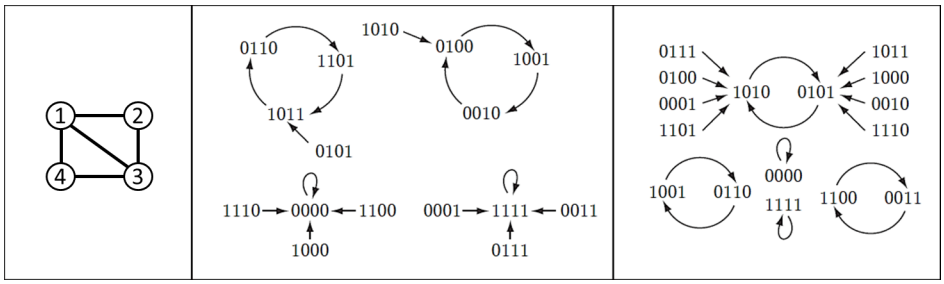


Figure 1.2. *The 4-vertex graph X (far left), and, in order, the phase spaces of* **F** *(synchronous update);* $\mathbf{F}_\pi$ *(sequential update with* $\pi = (1, 2, 3, 4)$*); and* $\mathbf{F}_{\pi_B}$ *(block sequential update with* $\pi_B = ([1, 3], [2, 4])$*). In all GDSs, each vertex is assigned the nand vertex function.*

neighbors in state 1 at time t*, then its state changes to 1 at time* $t + 1$*; otherwise, its state remains 0 at* $t + 1$*. If* $x_i(t) = 1$*, then* $x_i(t + 1) = 1$*; i.e., a vertex in state 1 remains in state 1. The update scheme is synchronous update. The initial state of the system* $x(0)$ *is given at the left: vertices 1 through 3 are in state 1, and all other vertices are 0. Since vertex 5 has two neighbors in state 1 (namely, 2 and 3), it changes to state 1 at time 1; no other vertex can change state at this time. At* $t = 1$*, vertex 4 has two neighbors in state 1 (namely, 1 and 5), so it changes to state 1 at* $t = 2$*. It can be verified that only vertex 6 changes state at the next time. Subsequent times produce no state changes, and consequently, the state at* $t = 3$ *is called a* fixed point.

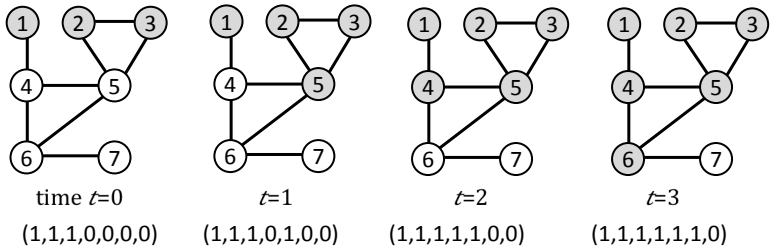


Figure 1.3. *GDS Example 1 showing threshold dynamics on a 7-vertex graph. The system state* $x = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ *at each time* $0 \leq t \leq 3$ *is provided under the respective time. The system reaches a fixed point at* $t = 3$*; i.e., for all* $t > 3$*, the system state* $x(t) = x(3)$*.*

Although not the focus of this dissertation, in the interest of completeness, we men-

tion that GDS is also a powerful framework for the evaluation of analysis problems. Representative examples of these works include the following, and references therein [10, 11, 24–26, 29, 29, 85, 117, 118, 129, 176].

1.5 Research Question, Solutions and Hypotheses

- **Research Question and Solutions**
  - **Can we design and implement a research framework to support domain scientists?**
    - S1 Developed framework consists of systems and services that are specifically designed for use by epidemiologists, social scientists, and (government) practitioners and other domain scientists who are not computing experts.
    - S2 Developed framework can also be used by computing experts to increase their productivity.
    - S3 Developed framework provides the suitable web based environment for scientists to share, reuse, and execute analyses simultaneously.
    - S4 Developed framework provides a suite of services that can integrate with several applications from different domains. The services provide the needed functionality to support the simulation process.
    - S5 Developed framework provides several capabilities to support data through the research process.
- **Research Hypotheses**
  - H1 By identifying the key challenges and problems during the research life cycle, we can have a better understanding of user requirements and the needed features for the proposed framework.

H2 Given the adequate training, domain scientists can effectively use the proposed framework.

H3 By delivering the required functionalities and features along with the ease of use and within a short learning time, the proposed framework can provide the needed support to domain scientists.

1.6 Research Overview, Approach and Contributions

The goal of this research is the design and implementation of an environment needed for modeling contagions in networked populations as well as studying and understanding network dynamics. In order to develop such an environment, we built four main components: (1) an open access distributed web application for characterizing GDSs (GDSC), (2) a web application for evaluation of network-based social dynamics (EDISON), (3) a suite of supporting network services (MARS), and (4) a web application for interactive exploration and understanding of contagion dynamics in networked populations (NEMO).

The dissertation is motivated by the need to overcome the previously mentioned challenges. Figure 1.4 illustrates the different components of the research framework. Case studies were conducted to illustrate the utility and usefulness of the developed systems and services within the framework. Additionally, the performance of different components was evaluated to assess the quality of services. Finally, we conducted a usability study to evaluate the usability and learnability of selected systems. The research approach is summarized in Figure 1.5.

In the following subsections an overview and contribution of each component is given. More details is presented in each relevant chapter.

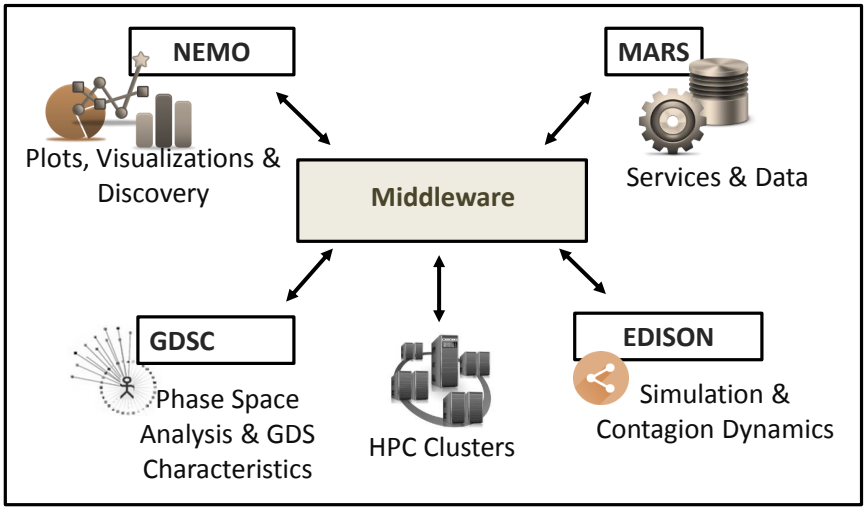


Figure 1.4. *An overview of the systems and services that form the main components of the developed research framework.*

1.6.1 GDSC

GDSC computes the complete dynamics (phase space) of a GDS. It can be used for both education and research. GDSC is a useful system for *experimental mathematics* and *computational mathematics*, where computational studies can be used to formulate theorems and guide their proofs. GDSC works on small to moderate size networks. The reason for this is inherent in the problem of computing the phase space: the number of state transitions that must be calculated for an n-vertex graph can be as large as $n! \cdot 2^n$. For a 100-vertex graph, this is 10^{188} state transitions. Nonetheless, specific reasoning can be done for a much larger class of networks.

GDSC contributions include: (i) the design and implementation of an open access distributed web application for characterizing GDSs, (i) a user-centered design of the front end that promotes usability and collaboration among scientists, and (i) use of high performance computing.

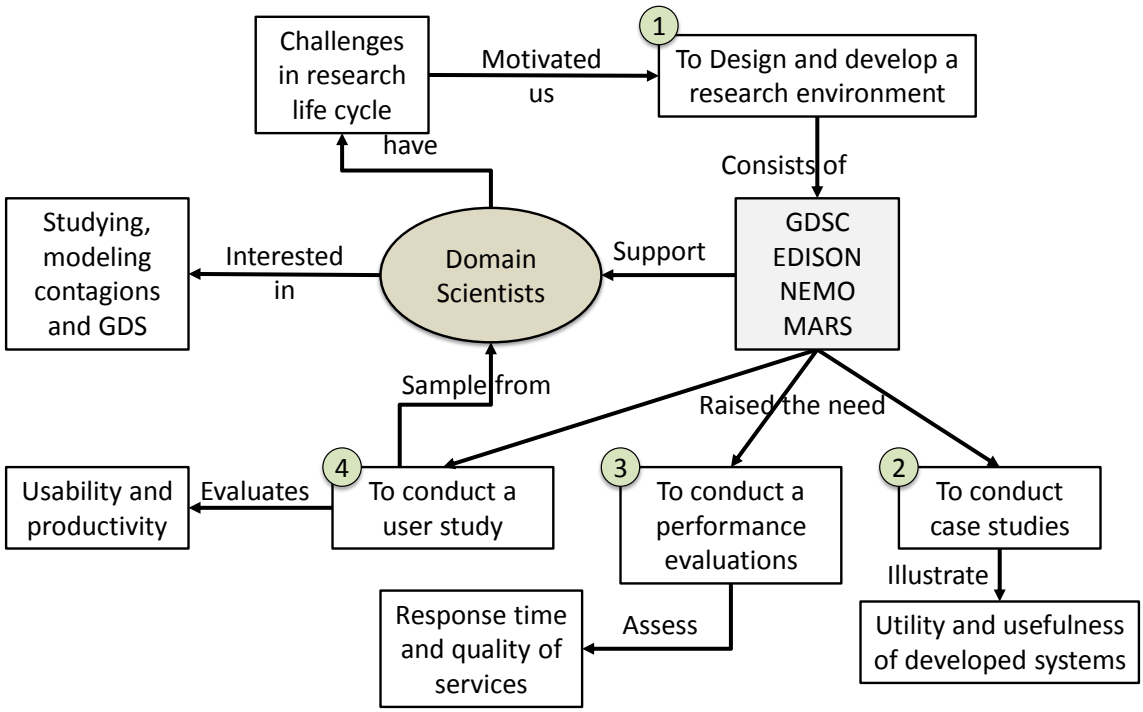


Figure 1.5. *An overview of the research approach. Numbered circles indicate the main stages of the dissertation.*

1.6.2 EDISON

EDISON performs agent-based simulations of forward trajectory analyses with emphasis on initial condition and interventions. Populations are modeled as networks (vertices and edges representing, respectively, population members and their interactions), and various dynamics are computed on them according to prescribed behavior models. Applications may range from public health epidemiology (e.g., spread of obesity) to the transmission of social influence and behaviors.

EDISON contributions include: (*i*) running experiments at scale (populations may range from a few hundred to millions of agents), (*ii*) it is web-based with a UI that is

specifically designed for use by epidemiologists, social scientists, and (government) practitioners who are not computing experts, and (iii) it is integrated with MARS services (described below) that contains a large number of open source social networks. EDISON utility is illustrated through case studies of disease and behavioral contagion transmission.

1.6.3 MARS

MARS services are essential for GDSC, EDISON, and other systems. They provide (meta)data storage, query, and analysis facilities that are used by both UIs and (HPC) simulation engines. In network science and dynamics, these data include agent dynamics models, their properties, and validity ranges; and networks and their vertex and edge attributes. Services include query, storage and retrieval, computations of network structural properties, and provenance.

MARS contributions include: (i) design and implementation of workflow system of services, (ii) processing and handling of big data, (iii) performance evaluation and load testing, and (iii) illustrative case study through integration with EDISON system.

1.6.4 NEMO

NEMO enables domain experts (e.g., network scientists, social scientists) to reason about the results of contagion simulations on social networks in terms of network properties. These properties can be (i) domain-specific attributes, such as the age and gender of humans that are represented by network vertices or weights of interactions that are represented by edges; and (ii) network structure measures, such as degree distributions or clustering coefficients.

NEMO contributions include the uses of a combination of data mining, computational analyses, and scientific data and network visualizations to enable knowledge discovery about networks that helps make sense of computed contagion dynamics.

1.7 Document Structure

The remaining of this dissertation is structured in the following way: In the next chapter (Chapter 2), we present a review of the relevant literature related to the developed research framework, grouped by each of the four major components. Chapters 3, 4, 5, and 6 cover in details the implemented systems and services, including GDSC, EDISON, MARS, and NEMO, respectively. Next, in Chapter 7, we discuss a case study on modeling and simulation of depression, as a contagion among students. Chapter 8 covers a study that involves a survey of research practices and a usability evaluation of EDISON system. Finally, Chapter 9 presents the research conclusion and anticipated future directions.

Chapter 2

Literature Review

In the following sections, the related work for each of the dissertation major components is provided.

2.1 GDSC Related Work

There are toolkits within Matlab [3] that are useful in analyzing selected dynamical systems; e.g., the Probabilistic Boolean Network toolkit developed by I. Shmulevich's research team [189]. Mathematica [2] has a cellular automata function that enables computation of dynamics on two-dimensional grid-like structures using synchronous update. Tools such as Dynamica [132] have been built on top of Mathematica. FiatLux [86] is a cellular automata simulator that sits on top of Ptolemy. It is used to study system robustness. It provides ten graph types and 12 dynamics models that are applied to vertices uniformly. Sequential and synchronous update schemes are available. The Ptolemy project at UC-Berkeley has developed software to model large physical systems [163] used in signal processing, telecommunications, network design, investment management, and can analyze both continuous and discrete sys-

tems [137]. BNS (Boolean Networks with Synchronous update) [78] simulates these systems, where update functions are given as truth tables (while flexible, the inputs are tedious, but more importantly, are exponential in the problem size). These are all open-source software packages, or add-ons to commercial products, that run on a user's machine, and in this sense are stand-alone applications.

GDSC, in contrast, is a web-based application with both (output) data and analysis management systems. This means that our system can be used for team science, including cases where team members are remotely located. If an analysis by one user is labeled public, then any other user can make a copy of that analysis, change particular inputs, and run the new analysis, without having to start from scratch.

The only other web-based tool, ADAM [109], also does not possess these management systems. ADAM is tailored for biological networks, and provides several of these. It also contains other biological models, such as Petri nets and probabilistic Boolean networks (PBNs). (We are developing a separate, stochastic modeling environment.) ADAM supports synchronous and sequential update; it does not support block sequential update and unfair word ordering. We also compute functional and cycle equivalence for (block) sequential update schemes. As there is no data persistence, using previous analyses as a basis for future analyses is not supported in ADAM, as it is in GDSC. In GDSC, analyses are stored on the host cluster, which is a nontrivial issue because one analysis can generate a 50 GB XML output file, and hence considerable storage space may be required to save a collection of files. GDSC not only stores this information for later retrieval and analyses, but also organizes data analyses. Furthermore, analyses run by a user can be marked public so that other users can access results, thereby promoting team science. Further, a user may load any public analysis, change one or two inputs, and submit the new job for parametric studies, which also promotes cooperative research [190].

The capability for entering vertex functions in ADAM is more general than that in GDSC. However, that generality comes at the cost of having to specify a great deal of information. For example, for a 30-node dependency graph, an ADAM user must *explicitly* specify 30 different functions that in general will contain different terms because the dependency graph is explicitly embedded in the equations. And each time the graph structure changes, the vertex functions must be re-entered. GDSC takes a different approach. The dependency graph is specified separately. A *type* of vertex function is specified by name. If the same vertex function type (e.g., NOR function) is applied to all vertices, then only two button clicks assign all vertex functions to all vertices because the function type works with the dependency graph to give a complete specification of the vertex functions. We claim that creating a 20-node graph, for example, is faster and less error-prone than specifying 20 vertex functions. Also, for example, if a user wants to run a second analysis using the same graph and a different vertex function for all vertices, then essentially two button clicks are required: one to load an existing analysis, and one to change the vertex function (and a few more clicks to navigate among input screens). With ADAM, 20 more equations are required for the second analysis.

See [1], which is a random Boolean network (RBN) toolkit for Matlab that will handle synchronous and sequential update. Another RBN software is RBNLab [92]. A more general dynamical systems tool is DDLab [219]. Reference [14] is a specialized tool for cell functions; other codes for specialized biological systems are at [189].

2.2 EDISON Related Work

There are other types of simulation tools, such as compartmental models, that are used for particular problems, such as SimSmoke [141]. These models use compartments of individuals that possess particular traits, rather than individual agents. For

example, properties for a particular agent cannot be changed because there is no notion of an individual agent.

Our focus for EDISON related work is primarily on open access simulation platforms that perform ABM and that scale to populations with at least millions of agents, so that experiments can be performed on populations at scale by users.

There are many simulators used for social science applications [148][1]. Here, we focus on selected open source simulation frameworks and provide references to others. We note that the frameworks can be used for more applications than social and epidemiological simulations, and most are discrete event frameworks. EDISON, in contrast, is a discrete time simulation environment.

At one end of the spectrum are parallel discrete event simulation (PDES) kernels such as μsik [157] that can utilize huge numbers of processes. This kernel has recently been used in simulations using cloud computing [221] (see references therein for different cloud-based simulation studies), and in simulations using 220 million MPI processes [158]. Its behavior models are specified through programming language source code.

Repast (HPC version)[2] is an ABM framework where simple behavior models can be built with higher level programming abstractions. Otherwise, models are programmed by a user. The software is downloaded and the system is compiled and run on user-provided or -accessible clusters. Swarm[3] is another advanced PDES framework, where behavior models are written in Objective-C and Java. Other large-scale ABM frameworks are MASON[4], Flame[5], and AnyLogic[6] (the latter is not open-

[1] See http://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software for over 80 simulation systems.

[2] https://repast.github.io/

[3] http://savannah.nongnu.org/projects/swarm

[4] http://cs.gmu.edu/ eclab/projects/mason/

[5] http://www.flame.ac.uk/docs/overview.html

[6] http://www.anylogic.com/

source). Each has strengths; e.g., AnyLogic comes with sophisticated simulation graphics; Repast and Flame run on HPC resources; and Flame was used to model the EU economy involving 55 agent types and hundreds of thousands of agents [154]. MASON differs from many others in using a multithreaded approach suitable for shared memory machines. See [13] for overviews of these and other frameworks.

At the other end of the spectrum is NetLogo[7]. Although it is not geared for HPC and large populations, it is a powerful, intuitive modeling environment with an IDE, high level abstractions for model building, and displays. Among its distinguishing features are ease of use and ease of learning. It, like the other simulators mentioned thus far, are locally built and run on user-provided resources.

Hybrid frameworks are those that provide both library-oriented modeling and IDEs (e.g., Repast). Hybrids offer a user the most flexibility in use; but these and those without IDEs (e.g., Repast, Swarm, and the others mentioned above) also require more detailed knowledge of the software [148]. EDISON is our solution to bridge this gap: to combine ease-of-use in specifying analysis inputs for large scale simulations, and HPC resources for scalable computations, without the need to understand software.

Arguably, the system closest to ours is described in [170]; see that work for additional references. However, that system is focused on SIR-style epidemics and does not have the breadth of models provided by EDISON. It does not handle directed and highly attributed population networks, does not have the system services to manipulate agents and interactions of networked populations, and uses different simulation engines.

In particular, to the best of our knowledge, no other system provides (i) a UI for streamlined analysis specification and management for multiple simultaneous users;

7https://ccl.northwestern.edu/netlogo/

(ii) specific functionality for querying and manipulating attributed big data sets, for fine-grained specification of properties for agents and interactions in large networks (e.g., those with millions of agents), and for parametric studies; (iii) a scalable HPC simulation engine; (iv) libraries of networked populations, and of disease and social behavior models; and (v) HPC hardware resources (HPC clusters and cloud services) for analysis execution. The web application obviates the need for programming skills.

2.3 MARS Related Work

MARS system is not a workflow *management* system (WMS) in the sense of [71] because our needs are different in some ways. Our workflows are geared toward a particular *application* space—that of network science. Our system is both the environment for executing workflows and the workflows themselves. Most large WMS are largely focused more on the physical infrastructure for executing workflows across domains; e.g., Triana [197] uses Grid resources and Pegasus [53] uses the Condor system and services many domains. Additional tools are sometimes used to construct the workflows; e.g., Wings [96].

The scope of individual workflows run on WMSs is generally larger; they often run larger applications that take longer times to execute (tens of thousands of CPU hours), and form end-to-end (i.e., complete) analysis systems, an example being earthquake analyses [53]. In contrast, our workflows tend to be more generic—within network science—and they are designed to service many applications and users simultaneously. Most of our workflows do not run for tens of hours, although some of them can, depending on input data size and the particular analyses requested. This leads to another differentiator, which is provenance, a key feature of WMSs [71]. Because MARS services multiple users for targeted needs, we do not keep a permanent log of every use of the system, although we could. However, we do store, index, and curate

results from service calls to improve response time in subsequent calls for service by the same or different users and applications (a system service called memoization). WMSs typically record data such as the calling program, date and time of execution, and other metadata to trace these interactions.

Other works compose their own workflows and workflow systems using particular approaches that suit their requirements, as do we. Examples include customized code [76], workbenches [171], and Semantic Automated Discovery and Integration (SADI) services for the sematic web [17]. Workflow systems are sometimes called problem solving environments and web services [150]

If we take a more general view of workflow and query systems. There are many other WMSs, like BioWMS [31], Taverna [217], and Kepler [146] that fit the same basic description given previously. Benefits and features of WMSs, along with other systems, are enumerated in [71, 95]. Repositories of workflows and workflow languages for execution on WMSs include [90, 97, 174].

There are many topics of WMS research. Provenance has received a lot of attention for relational [143] and RDF [60] contexts. see also [40, 182]. Other general topics include interoperability of workflows [82]; security [147]; evaluation of specified workflows [62]; finding common motifs in workflows [90]; and a language for iteratively refining workflows [178].

Although we chose SQL and relational databases for their familiarity among users, this is not required. There are SPARQL (i.e., RDF)-to-SQL translators [61] and reverse translators. [166]. There are also several extensions to SPARQL. [12]. Other query languages for networked data have been proposed; see a recent survey [16]. GraphDB [104] focuses on spatially embedded networks such as highway systems or power lines. PQL [139] is an SQL-based query language for querying biological pathway data. GOOD [105] is based on an object-oriented model and can be used to

modify graph structure; it is a starting point for other languages. Systems based on XML and the resource description framework (RDF) include SPARQL [162], which uses a semistructured data model to query graph networks in heterogenous web environments. Horton [179] enables execution of queries on large distributed graphs in parallel.

2.4 NEMO Related Work

There are various graph rendering and analysis tools, including those for time-varying networks [99, 208]. Additional works on graph visualization include [52, 210, 218]. A review of infectious disease epidemiology tools and visualizations [56] did not address sense-making of epidemiological contagion spreading predictions, which is our focus.

Top-down and bottom-up visualization theories are discussed in [99], and to some extent, in [59]. In the former case, one starts with formal representations and uses these to organize data. In the latter, data are first analyzed and representations are devised that fit the data.

Gephi [33] is an open source tool for visualizing networks and properties of networks. We use Gephi as a service within NEMO to render graphs and provide information about them. However, Gephi does not have the capabilities of NEMO for explaining contagion dynamics.

SocialNetSense [99] is designed to help analysts understand domain-based attributes and structural parameters of networks. Their work seems to be confined to networks with perhaps a few thousand vertices, whereas we seek to understand dynamics on networks that are 2 to 3 orders of magnitude greater in size.

Apolo [59] uses machine learning techniques to understand graphs. Moreover, be-

yond visualizing a network, it contains implementations of algorithms (e.g., PageRank) to annotate network elements.

Network Workbench (NWB) (e.g., [64]) is a tool for visualizing networks for a wide range of domains. It can investigate dynamics to some extent, but the tool is not used to explore and query networks, and to visualize networks and their properties, in order to explain contagion spreading on networks.

Contagion in networks is visualized in [208]. This work differs greatly from ours in that their definition of contagion is the process of removing vertices from graphs, not the spread of contagions like information on a network. Also, they are primarily studying financial networks and these are small (roughly 40 vertices). We seek to evaluate large graphs; e.g., those with millions of vertices. Furthermore, they are not concerned with explaining dynamics, only showing the dynamics (in terms of how a graph is reduced in size—for their type of contagion).

Sense-making data analysis tools for brain network data are called for in [192], where some of the stated needs are not only structural, but also related to brain functioning (i.e., dynamics).

Tools are mostly centered around the goal of visually displaying networks, although Apolo [59] and Gephi [33] contain algorithms for computing graph measures, as does NEMO. None of these tools address plotting network measures, which is at a lower level of detail than displays of vertices and edges. We will demonstrate the usefulness of these types of plots (in Chapter 6) to understand contagion dynamics. Most tools (beyond those identified here) seem to be focused on smaller graphs.

2.5 Agent-based Modeling of Depression Related Work

Depression is a primary factor for dropping out, and it can spread from person-to-person through social interactions. Several studies also identify peer influence as a main trigger for dropping out. Peer influence can exist in many forms. Mayer [151] found, for high school students, that the better a student's peers performed in school, the more likely the student is to drop out. Gaviria [91] also found peer-based effects for dropping out of high school. Crane [69] proposes a contagion (i.e., peer influence) approach to understanding social problems (including school dropping out), using an epidemic-like approach that is similar in spirit to the first epidemic-inspired social model [32]. The model is closely related to segregation models [183] and threshold models [100, 101, 212]. A relatively recent overview of contagion-like influence is given in [213]. Bank [23] identified different types of influence on students' persistence such as peer, faculty, and parental influences. Parents and peers were found to have stronger influences than were the faculty on the persistence of students.

Pyari [165] investigated the effects of anxiety among medical and engineering students. Results showed that medical students exhibited low anxiety in comparison to engineering students. Between 42% and 48% of PhD students in science and engineering at the University of California are depressed [42]. Vitasaria [206] studied the relationship between anxiety and academic performance. Results showed a significant correlation of high level anxiety and low academic performance among engineering students.

Students may find difficulty in keeping up with schoolwork; 36 percent of the students reported feeling frequently stressed [144]. No student reported a complete lack of worry about keeping up with schoolwork. Twenty-five percent of students reported frequent inability to pursue non-academic activities due to lack of time. Ten percent reported feeling most of time that they did not have a social life, while an-

other 41 percent reported occasionally feeling this way. These life imbalances can eventually contribute to development of depression. Another study [73] revealed relationships between mental health and year of study, academic program, and gender. Rizwan [173] found factors that affect the stress level of female engineering students. Results indicate that teachers' discouraging attitudes have the strongest effect on the stress levels of female engineering students. A study [185] by Cornell's College of Engineering involving 35.5% of the student population showed that the main sources of stress for engineering students include heavy workloads in engineering courses, large amount of time needed to finish assignments, not enough sleep, competition with classmates, and inflexibility of the Engineering curriculum.

Astin [19] in the theory of involvement, proposed that the greater the student's social involvement in college, the greater the student's learning and personal development and the less likely he or she is to leave. In the theory of attrition, Bean [35] identified three categories of reasons leading to student attrition: 1) background, 2) organizational and environmental factors, and 3) attitudinal and behavioral outcome. Spady [193] proposed a theory based on the use of Durkheim's [79] theory of suicides to explain freshman attrition. Spady's theory was the basis of Tinto's work [201]. It states that when a person shares values with a group, this person is less likely to commit suicide (or by analogy, drop out of school). Tinto identified in the interactionist theory three reasons for student departure: 1) academic difficulties, 2) inability of students to achieve their goals, and 3) failure to adapt to the institution social environment. Tinto's model focused more on academic and social integration.

Most of the mentioned studies follow an experimental or clinical approach, and may use quantitative, qualitative or mixed-method techniques to study student drop out phenomena and leading causal factors. These studies involve human subjects as well. Other studies take another approach through the use of agent-based modeling to study depression. Aziz [20] proposed a dynamic agent model of recurrences of de-

pression for an individual. Borkulo [205] shows the effect of interactions among different depression symptoms, which is called the causal interactions network. Both [50] proposed another model that has been used to simulate different scenarios in which personal characteristics determine the effect of stress on the (long-term) mood of a person. These agent-based models simulate the evolution of depression within a single agent/person. Our model goes beyond this point to simulate the evolution of depression both within a single agent and across agents of a population.

These theories, studies, and experiments indicate that depression is a major contributor to dropping out of school. Thus as a first step, we model the evolution of depression within agents and its transmission across agents. The results from simulations will be useful in follow-on work to forecast the impact on retention.

Chapter 3

GDSC: Graph Dynamical Systems Calculator

3.1 Introduction

Discrete dynamical systems are used to model various realistic systems in network science, from social unrest in human populations to regulation in biological networks. A common approach is to model the agents of a system as vertices of a graph, and the pairwise interactions between agents as edges. Agents are in one of a finite set of states at each discrete time step and are assigned functions that describe how their states change based on neighborhood relations. Full characterization of state transitions of one system can give insights into fundamental behaviors of other dynamical systems. In this chapter, we describe a discrete graph dynamical systems (GDSs) application called GDSC for computing and characterizing system dynamics [5]. It is an open access system that is used through a web interface. We provided an overview of GDS theory in Chapter 1. This theory is the basis of the web application; i.e., an understanding of GDS provides an understanding of the software features, while

abstracting away implementation details. We present a set of illustrative examples to demonstrate its use in education and research. Our perspective is that no single software tool will perform all computations that may be required by all users; tools typically have particular features that are more suitable for some tasks. We situate GDSC within this space of software tools.

3.1.1 Technical Challenges

Technical challenges of GDSC include: (i) designing and implementing an intuitive user interface (UI) that guides a user through the analysis process, yet provides flexibility [187]—and in particular for this application, a UI that is sufficiently explicit for a novice user and yet streamlined for an expert [187, 190] (also, see factors on frustration [43]); (ii) granting the user an appropriate degree of control of inputs [187]; (iii) providing a system of checks on user inputs that are executed at the appropriate times (e.g., upon input, when user leaves an input screen, checks across input screens) with meaningful directions for correction [198]; (iv) providing a full-featured regression test capability for administrators that facilitates system verification, for use after software changes and porting to new platforms; and (v) devising a distributed system architecture that is flexible and extensible (e.g., to incorporate different types of resources and handle large file sizes, on the order of tens of GB).

3.1.2 Contributions

1. *GDSC*—An open access distributed web application for characterizing GDSs.

There are three system entities: client, server, and compute platform. These enable separation of concerns: user inputs, data storage, and multiple HPC systems for calculations. These also improve system extensibility.

There are many system components and functional features. Graph generators: 13 composable graph template classes (e.g., lattice, biclique, Circle$_{n,r}$, trees); also, directed, undirected, and weighted graphs are supported. Vertex states: vertex states beyond 2-state Boolean sets are available. Vertex functions: 15 types of vertex functions, where each graph vertex can be assigned a different type. Update scheme generators: a completely general capability that includes synchronous (i.e., parallel), asynchronous (i.e., sequential), block sequential, and unfair word orderings [155]. That is, virtually any update scheme can be specified. This is particularly useful for evaluating hierarchical systems [21] and graphs with classic dyadic edges and hyperedges. Compute engine: computes all dynamics of the GDS, as well as functional and cycle equivalences. Middleware: connects the client-server web application and the backend high performance computing (HPC) cluster. Java remote procedure calls (RPC), Java Message Service (JMS) and secure copy program (SCP) are used for different types of data transmission of different file sizes. Database (DB): stores user information and analysis data for data persistence, and for fast copy-and-modify for new analyses. Job management system: a user's jobs are listed, along with the status (e.g., submitted for execution, completed) and links to inputs and results. Users can also view analyses by other users that are labeled public. Post processing results: XML output file and four different types of plots can be generated. The code also computes and displays functional equivalence and cycle equivalence across GDS maps [155], which is particularly useful for summarizing results for large analyses. Automated verification testing: administrators can select any number of existing analyses to rerun. After execution, previous (valid) results and new results are compared automatically to determine individual analysis and overall system verification.

2. System Usability. Usability is important for positive user experience. We overview selected usability features that address the technical challenges, as cited in the literature of Section 3.1.1. *User feedback* is a critical aspect of user interfaces [37, 198].

In GDSC, (i) screens adapt to inputs based on intra- and inter-screen dependencies, changing options provided to users; (ii) if a user changes inputs in an earlier screen, their effects are automatically propagated forward to appropriately modify existing inputs in later screens; (iii) there is dynamic status update of all jobs, including multiple states for jobs in-progress; and (iv) input checking using a three-tier system (immediately upon input, after all data in a screen have been inputted, and checks across screens). Thus, GDSC is a *human-computer interaction system*, as defined in [37]. An interaction-based taxonomy of human-computer systems (HCS) is offered in [37]. GDSC contains features from all three classes of this taxonomy. For example, it is a *medium for communication* in that remote users may not only share analyses and results, but also a user can take another user's public analysis, copy inputs, make changes to them, and run a new analysis, thereby fostering cooperative science and *rapid incremental exploration* [190]. It is argued in [37, 187] that *user control* is a central aspect of user interfaces that moves them from merely algorithmic to interactive, and thus more intricate to develop. As an example, every input screen either provides templates that enable arbitrary, user-defined composition of inputs (e.g., graph definition) or provides modifiable inputs on the screen that are highly suggestive of the range of inputs that can be specified. These ease and speed data input. We also implement a two-stage input model for user input screens. First, parameters may be assigned to a designated subset or all of GDS objects; they may also be modified in this way. Then, second, since all of these inputs are presented in a Work Display, individual inputs can be surgically altered. This flexibility is the first guideline of control for UIs [198].

3. Connection between theory and GDSC. We provided a short section on GDS theory in Section 1.4, which makes clear the foundations of the web-application. In particular, the theory is designed to produce a *mental model* [198] (as well as a formal model) for the user, which is then reflected in the UI, compute engine, and results of

GDSC.

4. High performance computing resources. The web application exposes the capabilities of the GDSC compute engine (backend), which runs on a high performance compute (HPC) cluster of Sandybridge nodes. Consequently, a user makes use of these resources free of charge. There is no need to download source code and run the application locally, nor to have personal access to HPC resources. Beyond use of the current system, the user automatically benefits from upgrades to the cluster and to the GDSC backend; the user need not keep track of, nor be concerned with, updates.

5. A mechanism for team science. Unlike stand-alone open source applications whose source code is downloaded and compiled on a user's machine for his or her individual use, a web-based application affords the opportunity for team science. With GDSC, a user can specify an analysis as public, in which case the analysis inputs and outputs can be viewed by any other user. (An analysis can also be designed private, in which case only the originator can see it.) Not only can public analyses be viewed, but also they can be loaded, modified, and run by any user. This facilitates team science, even with members remotely located.

As these contributions indicate, this work is about the entire GDSC system—the UI, middleware, and the backend compute engine—and its utility. We do not describe the details of the compute engine; its functionality is based on the theory in Section 1.4 and the UI described in Section 3.3. The general theme is that any feature produced for the UI has its counterpart in the compute engine at the backend.

The GDSC landing page [4] contains supporting materials including: a PowerPoint presentation for teachers and researchers to introduce/overview the system to users; a user manual; videos that describe the code's use to support immediate usability [133]; and a list of relevant publications.

Chapter Organization. GDSC system components are overviewed in the following

section, with a focus on client, server, and HPC cluster configurations, as well as middleware. GDSC functionality is presented through a control flow diagram that gives an overview of user interaction with GDSC. Thereafter, the UI and system outputs are presented, which gives more details. Finally, a case study is used to demonstrate the utility of the GDSC system; this example is taken from real research project using published data.

3.2 GDSC System Overview

An overview of the GDSC system is provided in Figure 3.1. The system is distributed—the client, server, and HPC cluster reside on distinct machines, although they need not. Reasons for this separation follow.

1. The client side separates user interaction from all other functionality. To improve response time, some data are located on the client, rather than the server, as described momentarily.

2. Results files, which can be tens of gigabytes in size, are stored on the server, and accessed through the database. The client, which requires access to these files, need only communicate with one resource.

3. The separation of server and compute cluster enables results to be stored on a system different from the HPC resources. For example, it is often not practical or desirable to store results on compute platforms (e.g., Future Grid cloud resources or Open Science Grid (OSG) resources).

4. This design facilitates the use of multiple compute platforms (e.g., different compute clusters and cloud resources), with one location for storage of output.

5. It improves extensibility: new compute resources can be added with localized, surgical changes to the system. Multiple compute resources currently exist, and more are set to be added.

We now provide additional details of the overall system.

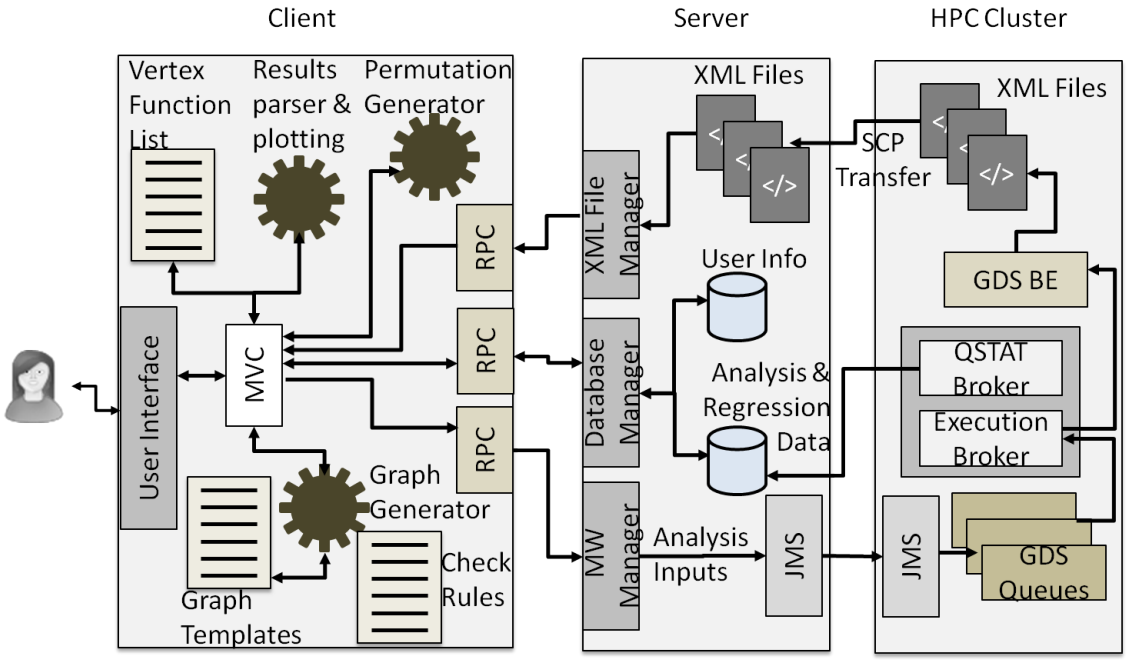


Figure 3.1. *Overview of GDSC components and hardware.*

The client-side contains the graphical UI that is run through a model-view-controller (MVC) design. Vertex functions and graph templates are stored here, for faster response time than could be provided through a database on the server side. Graph and permutation generators are also brought to the client, again for better user experience. For example, the display tool is slow in writing all permutations into the UI. Displaying all of these at once can take a long time, so instead the data are staged and only those permutations that are viewable on screen (e.g., as the user scrolls) are computed on the fly and populated in the table. Checking rules and associated values are also part of the client-side based on response time considerations. Remote procedure

calls (RPC) are used for communications. Data are transmitted to the server when the user submits the analysis, which also writes DB entries.

On the server side, inputs are transmitted to the HPC cluster via JMS. Various queues are used on a cluster. The execution broker translates the received inputs into ASCII data files required by the GDS BE (i.e., the C++ compute backend application). The QSTAT broker provides the status of an analysis to a DB on the server. This DB is polled by the web-app to obtain each analysis's status, and to update it within the Analysis Log screen so that each user has realtime knowledge of the status of all of his or her submissions. The output from GDS BE is an XML file. This file is transmitted to and stored on the server. File transfer is via SCP, since files can be as large as 50 GB, and hence conventional message passing would not suffice.

3.3 Web Application and System Features

In this section, we describe the main system features by focusing on the GDSC UI. Clearly, features and capabilities exposed by the UI must be implemented in the compute engine, which performs the computations described in Section gds-foundations. Furthermore, Middleware was described in Section 3.2.

Figure 3.2 provides control flow for GDSC, which also describes how a user interacts with the system. Almost every box in this figure, with two exceptions, corresponds to a user screen within GDSC. Some boxes are associated with multiple screens, which evolve based on other user inputs. After login, there are three user options: Input View, Analysis Log View, and Admin View. In the Input View, a user can start an analysis from scratch or by loading the inputs from any public analysis produced by any user. The user moves through four input screens: Dependency Graph, Vertex Functions, Update Scheme, and Initial States—these are the inputs for a GDS. (The

state set K is specified implicitly from the vertex functions.) These four screens must be traversed in the shown order (for the sake of non-expert users), but once values are selected for the screens, the user may move among the screens, changing values. A user inputs metadata and submits a job.

Continuing with Figure 3.2, all public analyses and/or all user jobs can be viewed, with their status, on the Analysis Log screen. Operations on results are shown under the Analysis Log. Options for viewing results are provided under View Analysis Results. Regression testing is valuable for verifying system performance (verification testing); this is only used by administrator users. It is important to note that from *any* screen, the user can go to the three main views located at the top, so there is considerable maneuverability and flexibility for a user.

We now address the three main elements of the UI: *Analysis Inputs*, *Analysis Log*, and *Regression Testing*.

3.3.1 Analysis Inputs

We describe the four main input screens for a GDS analysis.

Dependency graph definition. The dependency graph identifies the vertices whose states contribute to the next-state computation, f_v, of vertex v. Specifically, all vertices u with an edge from u to v have states x_u that are inputs to f_v. These edges overall v form the graph. The dependency graph can be specified in one of three ways: (i) list view (i.e., tabular view) with graph templates, (ii) list view to enter edges (and isolated vertices), and (iii) graph view. In the first two, a graph is specified by entering edges, in tabular form. Two vertices on a row of the table form an edge. The default is undirected edges, but per-edge directionality is changeable, and directed and undirected edges can be mixed. For example, if a graph has 25 undirected edges

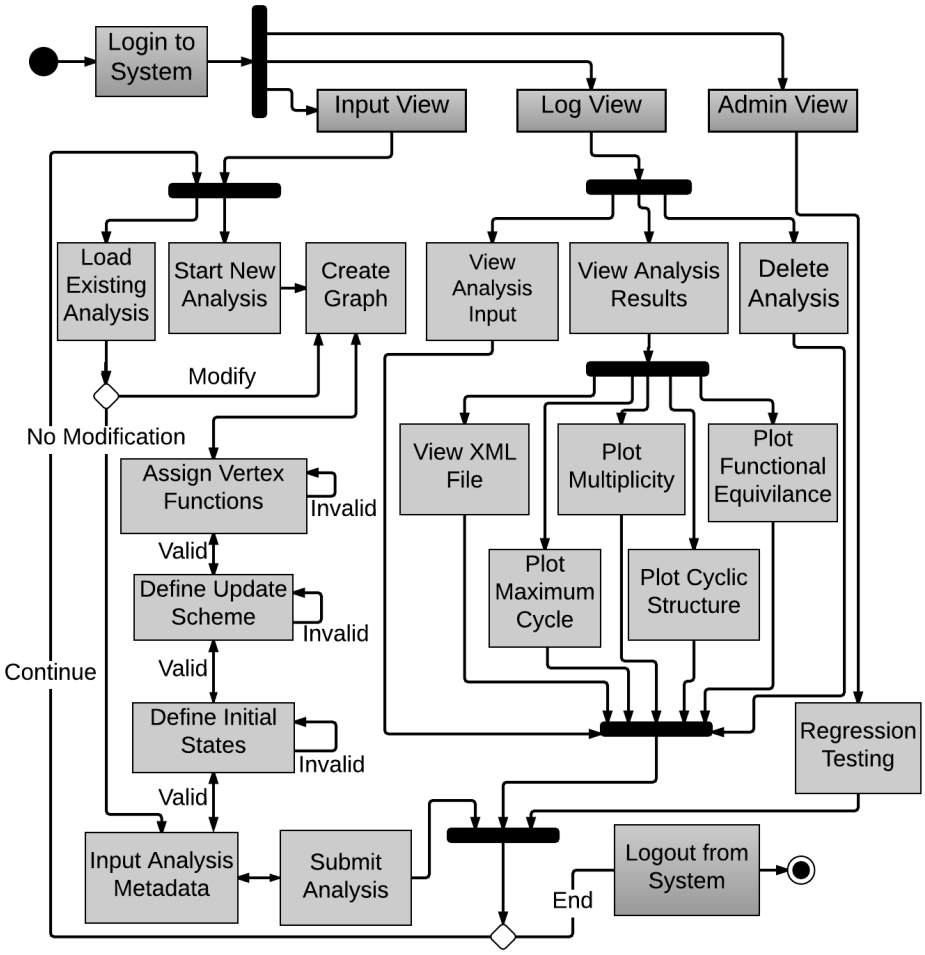


Figure 3.2. *Control flow through GDSC. At any time within GDSC, a user can go directly to "Open Input View," "Open Analysis Log," and if an admin user, "Open Admin View" (gray boxes) to initiate operations. Thus, system navigation is quite flexible. However, to guide the user in specifying analysis inputs, for example, some screens are presented in a prescribed order. Even in these cases, the user can move back and forth among screens.*

and one directed edge, the user may enter the edges in just this way; he or she need not enter 51 directed edges. A directed edge is *from* the first vertex *to* the second vertex, meaning that the *from* vertex influences the *to* vertex. These edges can be entered by a combination of graph templates or by adding edges one at a time. Graph templates enable a subgraph to be specified in a couple of clicks. To use graph templates, the user specifies *List (Basic Type)* for the *Input Mode* field. For the *Graph*, the user selects a template from the list in Table 3.1. Templates from this table can be combined to form larger graphs. Alternatively, a graph can be constructed visually, as shown in Figure 3.3, using graph view. This 20-vertex graph can be generate with a few clicks in list view mode, by joining a Wheel_5, $\text{Circle}_{6,2}$, and Tree subgraphs.

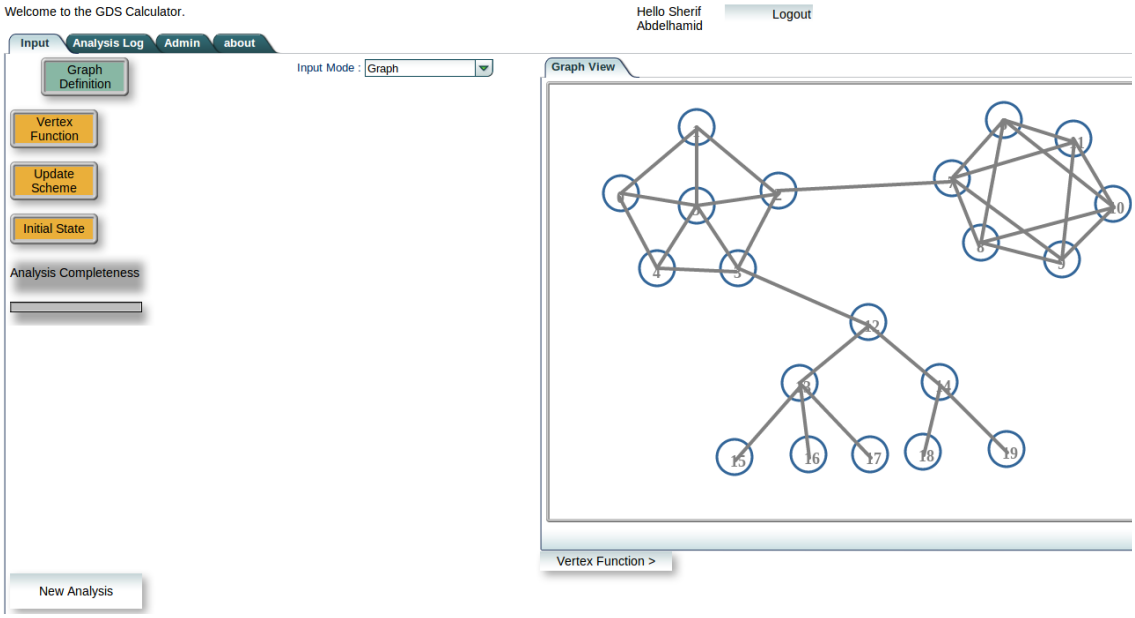


Figure 3.3. *Dependency graph shown in the Work Display area of the* Graph definition *input screen.*

Vertex functions. Each vertex of the dependency graph must be assigned a function that describes how that vertex state is updated. The Work Display for the vertex functions for a 4-vertex graph is provided in Figure 3.4. The drop-down *Function* list includes the vertex functions of Table 3.2. For a selected function type, the user speci-

Table 3.1. *Graph templates that can be used individually or in combination to form dependency graphs. Additional vertices and edges can also be added to these subgraphs, or a graph can be created from scratch.*

Number	**Template**	**Constraints**
1	Path	$n \geq 1$
2	Circle	$n \geq 3$
3	Wheel	$n \geq 4$
4	Star	$n \geq 3$
5	Degree-4 Lattice	$n \geq 4, (2 \times 2)$
6	Degree-4 Wrap-Around Lattice	$n \geq 4, (2 \times 2)$
7	Degree-8 Lattice	$n \geq 4, (3 \times 3)$
8	Degree-8 Wrap-Around Lattice	$n \geq 9, (3 \times 3)$
9	Clique	$n \geq 2$
10	Biclique	$n_1 \geq 1, n_2 \geq 1$
11	$\text{Circle}_{n,r}$	$n \geq 4, 1 < r < n/2$
12	Petersen	–
13	Full tree of arity (a) and depth d	$a \geq 2$, $depth \geq 1$

fies the vertices that are given this function assignment in the *Target Nodes* field: either *All* vertices or a *Selected* set are chosen. In the latter case, checked boxes in the Work Display identify the target vertices. For functions such as *Threshold Function*, parameter values may be required, and these are assigned in the same manner. Within the Work Display, individual entries may be changed. These features provide flexibility in assigning vertex functions with a minimal number of clicks, for different levels of granularity. The functional form for each vertex function is provided in the online user manual for GDSC [4].

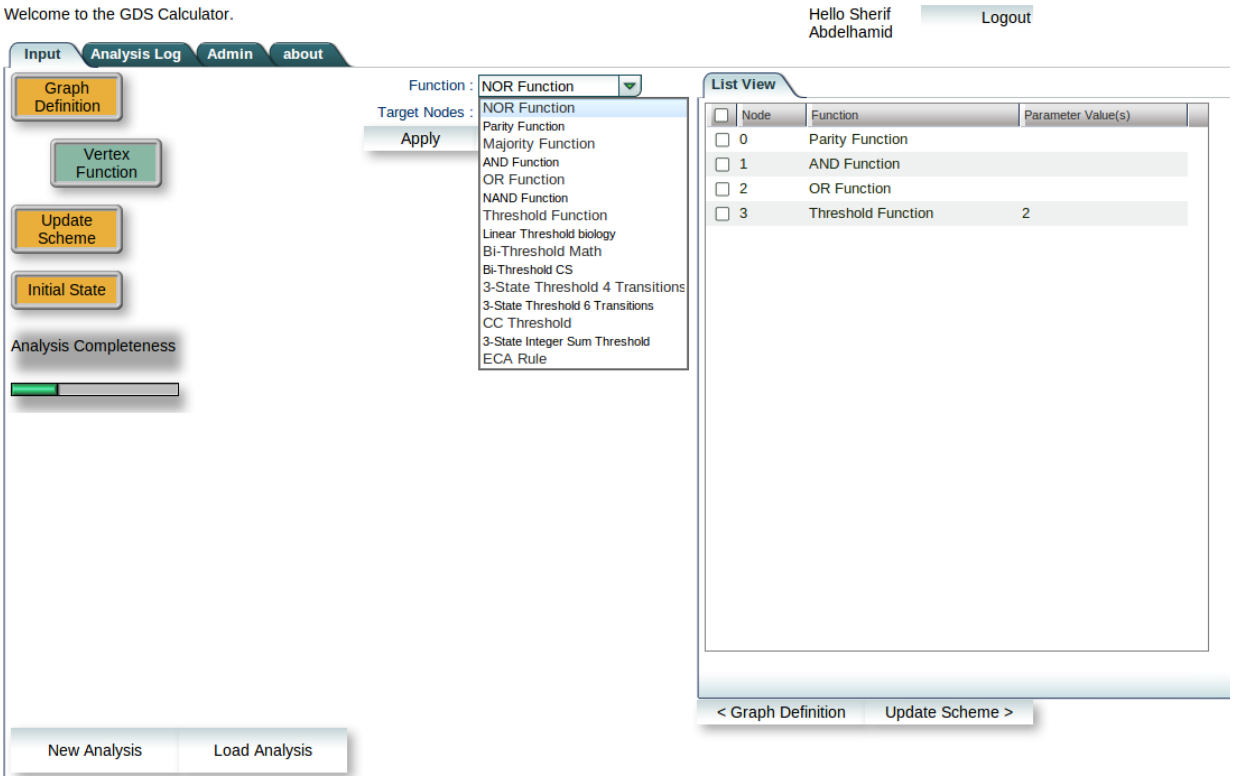


Figure 3.4. *Selected vertex functions in the Work Display of the Vertex Function screen for a four-vertex graph. The functions to the left of the Work Display can be selected and applied to all or any (target) subset of graph vertices. Each vertex is assigned a function from the list in Table 3.2, and can be mixed as long as their vertex state sets K are the same.*

Update schemes. An update scheme prescribes the sequencing of the execution of vertex functions at each time. See Section 1.4 for details of synchronous, sequential,

Table 3.2. *Vertex functions that can be assigned to any set of graph vertices. See the online user manual for the definition of each vertex function.*

Number	**Vertex function,** f_v	**Vertex State Set,** K
1	nor [155]	$\{0,1\}$
2	parity [155]	$\{0,1\}$
3	majority [155]	$\{0,1\}$
4	and [155]	$\{0,1\}$
5	or [155]	$\{0,1\}$
6	nand [155]	$\{0,1\}$
7	Threshold, progressive [100]	$\{0,1\}$
8	BiThreshold, mathematics [131]	$\{0,1\}$
9	BiThreshold, data mining [130]	$\{0,1\}$
10	3-State Back-and-Forth, 1-hop transitions, data mining. [152]	$\{0,1,2\}$
11	3-State Back-and-Forth, 1-hop transitions, mathematics.	$\{0,1,2\}$
12	3-State Back-and-Forth, any transition.	$\{0,1,2\}$
13	Connected Components Back-and-Forth [203]	$\{0,1,2\}$
14	ECA rules on Circle$_n$ (256 functions) [155]	$\{0,1\}$
15	Linear Threshold [72]	$\{0,1\}$

and block sequential, and fair and unfair word orderings. See Figure 3.5 for the Work Display and options to the left of it, in the Update Scheme screen. An unfair block sequential update scheme is selected. For this scheme, there are three choices: single sequence of vertex IDs; random sequence, each with a random number of blocks; and random sequences, each with the same number of blocks. Since the latter is selected in this figure, the user enters the number of sequences and the number of blocks per sequence. Blocks of vertex IDs are separated by "|" in a sequence, while vertices within a block are separated by "-". Clearly, some sequences are unfair, because a vertex appears in some of them more than one time. Also, for example, in entry 3, the vertex function f_3 never executes. Each entry in the Work Display can be modified, copied and pasted into another sequence entry field, or deleted. Modifications, in particular, can include vertex changes and movement or changes in "|" and "-" symbols. User control is again evident. While almost any combination of vertices is admissible for block sequential, unfair, the other update schemes demand fair word ordering so that each vertex must appear exactly once. System checks handle all cases. In Figure 3.5, if sequential update is chosen, there is one other option beyond the three options shown for *Number*. This is *All*, meaning all permutations will be evaluated. For synchronous, there is only one option: execute all vertex functions simultaneously at each time. Finally, the options that are presented to the user are based on rules, in order to preclude specification of conditions that lead to huge computational demands. These rules are based on the number of graph vertices and the specified update mechanism.

Initial system states (configurations). Given a GDS, system states x must be specified for which next states x' are computed. In a typical analysis, all system states will be specified; this is a requirement for phase space(s) to be computed. However, it is possible to evaluate any subset of system states in order to compute particular state transitions.

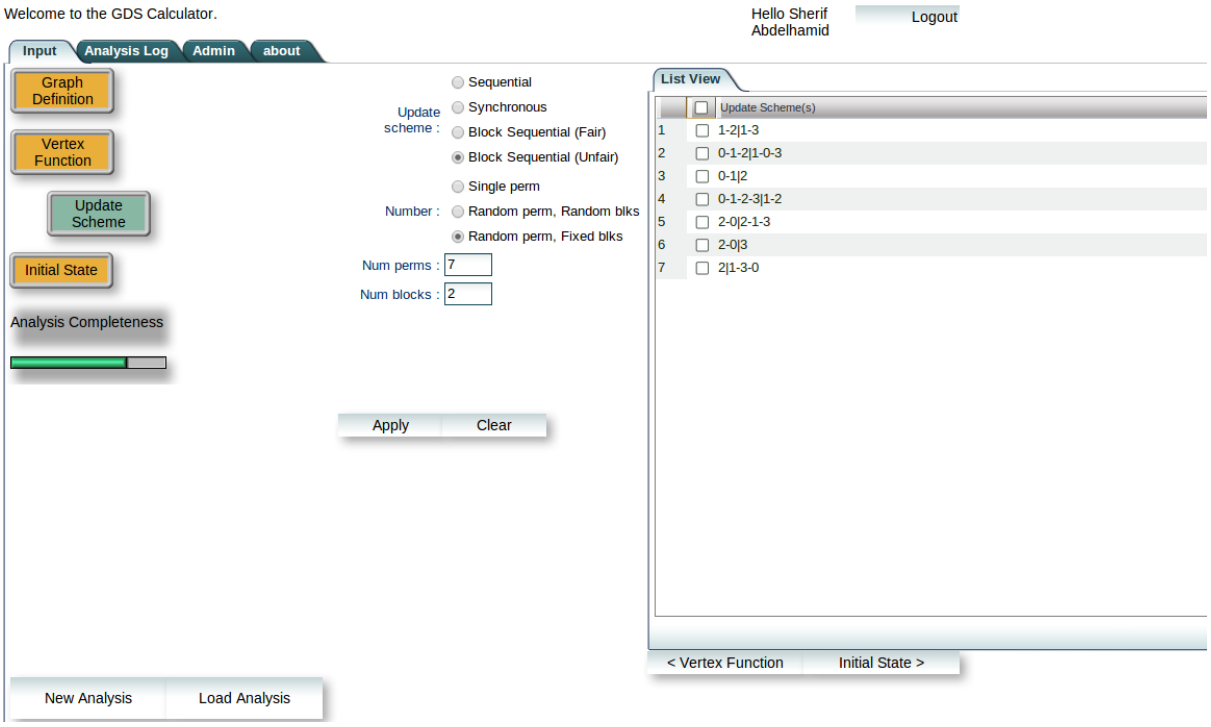


Figure 3.5. *Update scheme screen for a 4-vertex graph. A user selects the type of update (sequential, synchronous, block sequential fair word order, or block sequential unfair word order). For each choice, there are further options to help the user tailor update sequences to analyze. Here, the choice is block sequential, unfair, with seven random vertex sequences, each with two blocks. However, the user can change the number of blocks of a vertex sequence within the Work Display for additional flexibility.*

3.3.2 Analysis Log

Figure 3.6 shows a portion of the *Analysis Log* screen. The table summarizes either a user's analyses alone, or a user's analyses and also all public analyses in GDSC. An analysis can be selected, and its inputs (*Inputs*) and outputs (*Results*) can be viewed, or the analysis can be deleted. Several different types of plots can be generated using the XML-based output file. Some of these will be presented in the forthcoming example, and include plots of cycle structures and functionally equivalent and cycle equivalent GDS maps.

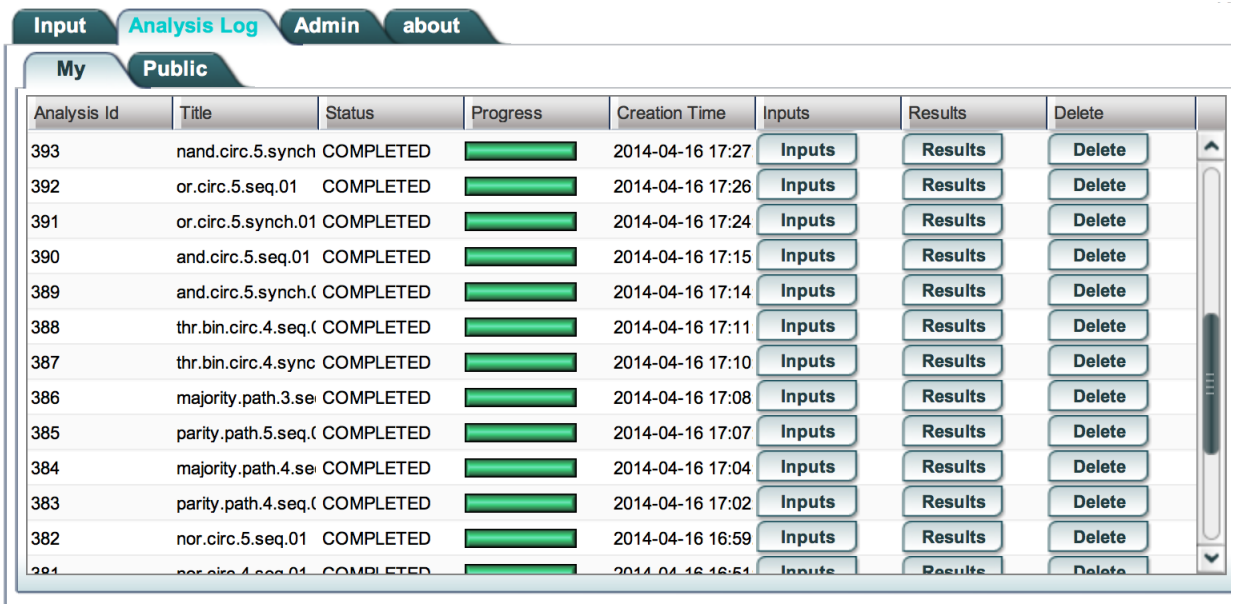


Figure 3.6. *The Analysis Log screen provides a listing of* (i) *a user's own analyses (under* `My` *tab) and* (ii) *all public analyses completed within GDSC (under* `Public` *tab). For each analysis, its status is given, along with name and date, and options for viewing inputs and results, and for removing the analysis.*

Example 4. *This example is a continuation of Example 1 in Section 1.4. See that example for the graph and vertex functions used. Here, we expand the evaluation by examining all permutations for sequential update scheme, not just* $\pi = (0, 1, 3, 2)$ *as in the earlier example. We first look at all* $n! = 24$ *permutations for the 4-vertex graph. Figure 3.7 shows the 14*

permutation classes that give different GDS maps. For example, there is a class of permutations with $[(1,3,0,2)]$ *that contains four permutations that all give the same GDS map (see the bar with x-axis value of 10). The XML file can be interrogated to see these permutations. From these 14 classes, Figure 3.8 shows that there are two cycle-equivalent equivalence classes,* $[\pi] = [(0,1,2,3)]$ *and* $[\pi'] = [(0,1,3,2)]$*, each identified by one permutation of the class. The limit cycles for permutation* π' *are given in Figure 3.9.*

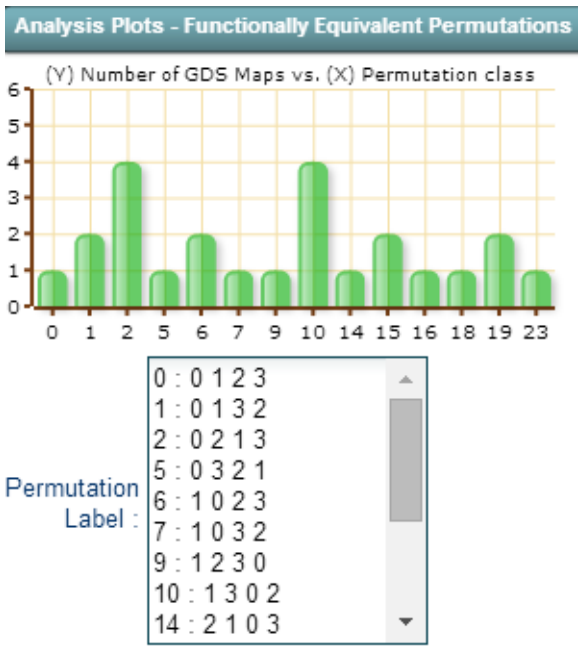
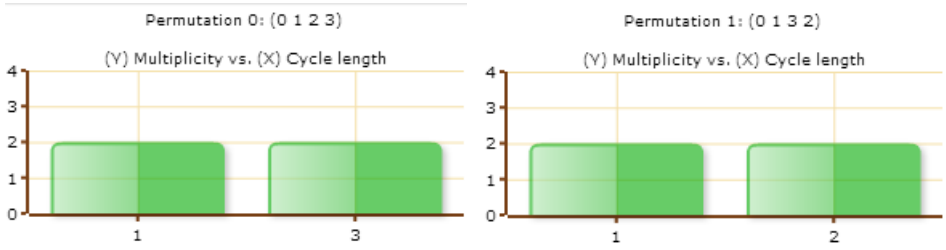


Figure 3.7. *The number of GDS maps within a functional equivalence class (i.e., the number of permutations within a class of permutations that gives the same GDS map). The x-axis gives an integer corresponding to the table below the figure. This table contains a permutation for each integer: a permutation that represents the equivalence class. In this figure, permutation class [(0, 1, 2, 3)] contains one GDS map—the one corresponding to that permutation. Permutation classes [(0, 2, 1, 3)] and [(1, 3, 0, 2)] have the largest number of GDS maps, which is four. See Example 4.*



(a) **(b)**

Figure 3.8. *There are two cycle equivalent permutation classes in Example 4. (a) For permutation class [(0, 1, 2, 3)], there are two fixed points (i.e., 1-cycles) and two 3-cycles. (b) For permutation class [(0, 1, 3, 2)], there are two fixed points and two 2-cycles. That is, each of the 24 permutations is an element of one of these equivalence classes. Hence, to within an isomorphism, the 24 original permutations all give one of two long-term cycle structures. Note that the permutation in (b) is the one addressed in Example 1 and Figure 1.1.*

3.3.3 Auto Regression Testing

A facility for regression testing is provided for administrator users. This feature significantly streamlines system verification stemming from software updates and new deployments. See Figure 3.10 and the *Admin* tab. In the left side, the administrator can choose to rerun any combination of existing analyses by clicking the *Rerun* button beside the desired analyses. These rerun jobs also appear in the left table and are marked as "Rerun of ID" where ID is the ID of the original job. The original job ID and the companion rerun ID are placed in the right side table of Figure 3.10. When the rerun job is completed, the *Diff* button may be clicked, and the XML files from companion runs are compared. Differences are displayed on screen. After comparison, each or both of the analyses may be deleted; one presumably keeps the (an) analysis with valid results, which become the verification cases for the next software update. Note that testing the UI is a separate issue, but regression testing does test assembly of input data, passing of data from the client through the rest of the web application, generation of input files for the backend computations, GDS computa-

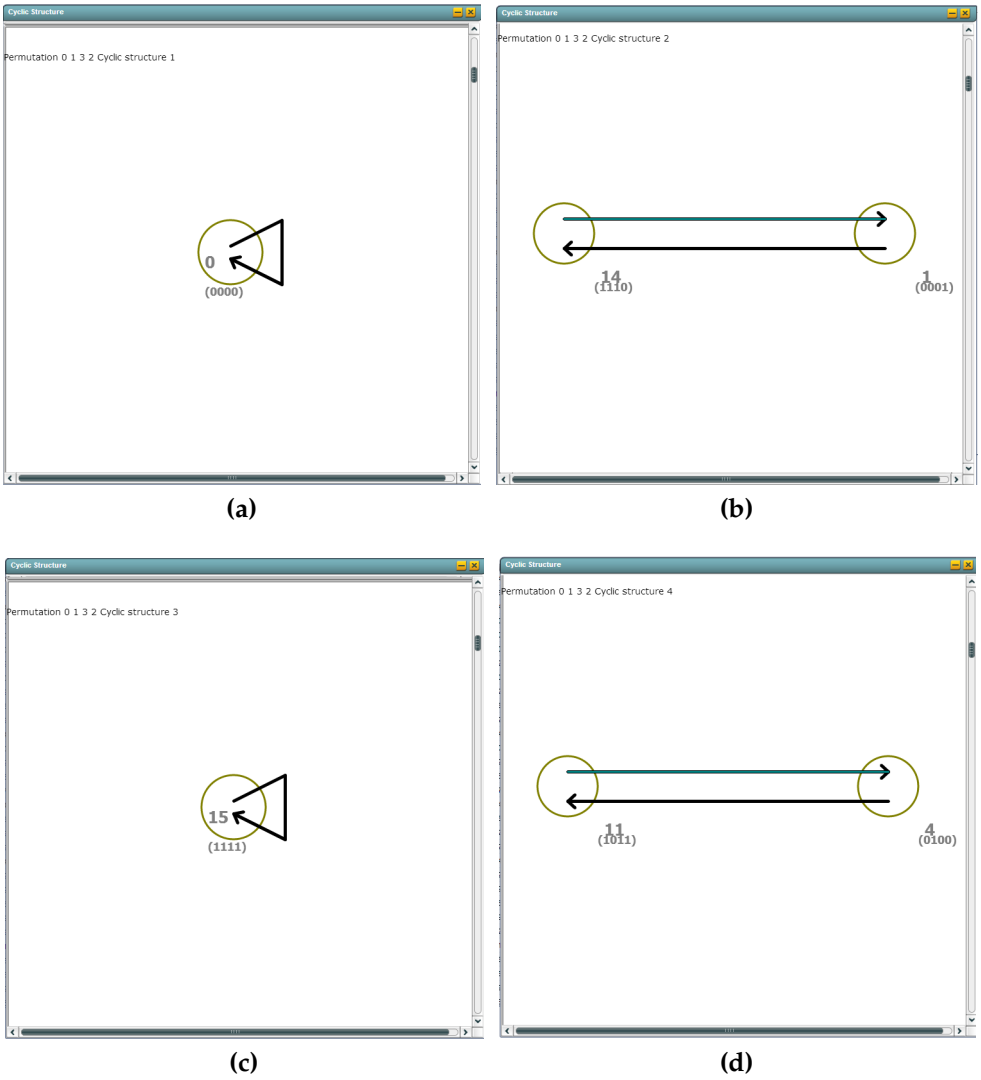


Figure 3.9. *There are four cycle structures for permutation class (0, 1, 3, 2); these four structures are the ones addressed in Example 1.*

tions, database operations, and data transfer back to the client-side application, which is a significant part of the entire process. This capability is used with each software update that we provide.

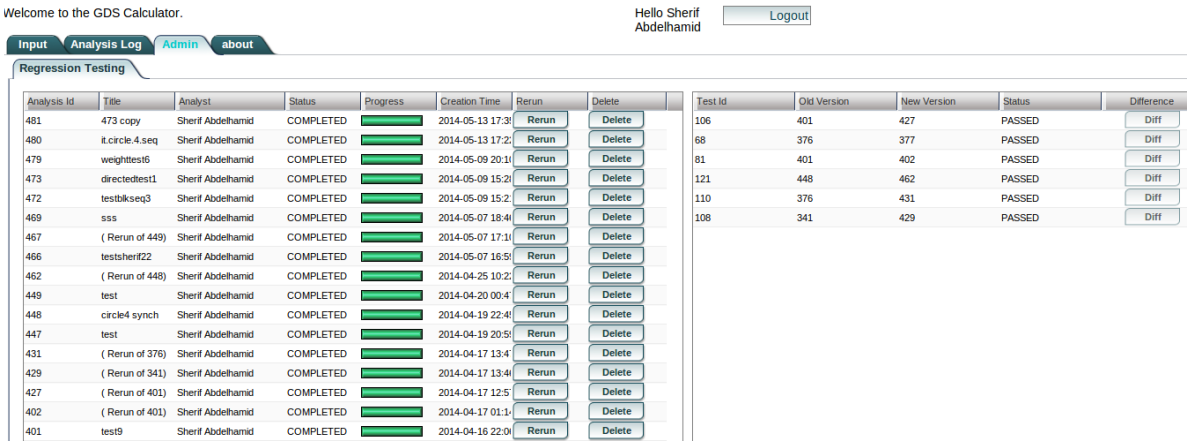


Figure 3.10. *The Admin tab, for selected users, allows regression testing and verification to test the system when features are added or modified, or when the system is ported to different hardware.*

3.4 Illustrative Case Study

We provide a case study using a biological network. The case study illustrate that we can also model dynamical systems used by other researchers (e.g., [72]), demonstrating that GDSC has wider applicability than just our work. Experimentally, we can evaluate 20-vertex graphs. This limitation is due to memory consumption, and output file sizes generated and their transfer across machines; we are working to increase this limit.

Before moving to the case study, we note that the power of GDSC grows with user expertise in dynamical systems (although this expertise is not required). As one example, it was proved in [149] that for any network G that is a tree, all sequential update permutations produce the same phase space. This means, for example, that if π and

$\hat{\pi}$ are two permutations for a tree network, then $\mathbf{F}_\pi = \mathbf{F}_{\hat{\pi}}$; i.e., they produce the same state transitions and hence generate the same phase space. Consequently, naively computing phase spaces for all permutations of a 20-vertex tree requires determining $20! \cdot 2^{20} \approx 10^{24}$ state transitions. Using the above result, this number can be reduced by a factor of $20! \approx 10^{18}$ because now only one of the $n!$ permutations need be evaluated. Other examples can be found in, for example, [72, 98, 131, 149, 177].

Let X be the variant of the genetic regulatory network that is Fig. 6 of [72], reproduced here as Fig. 3.11. Directed edge (u, v) means that u influences v; i.e., $u \in n[v]$.

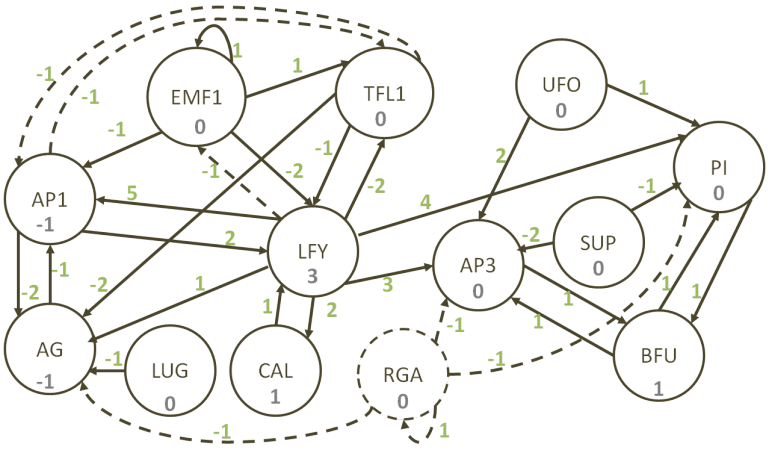


Figure 3.11. *Dependency graph for a genetic regulatory network [72].*

Edge weight $w_{u,v}$ is the magnitude of influence; $w_{u,v} < 0$ means that u inhibits the transition of v. The threshold k_v for vertex v is given inside each vertex in the figure. The GDS is a Boolean system; i.e., $K = \{0, 1\}$. A vertex may transition from state 0 to state 1, and from 1 to 0. Each vertex v has the vertex function

$$f_v(x[v]) = \sum_{u \in n[v]} (w_{u,v} x_u) - k_v \tag{3.1}$$

with the next state of x_v given by $x_v = 1$ if $f_v(x[v]) > 0$, and $x_v = 0$ otherwise. Synchronous update yields 50 fixed points and 24 2-cycles. They are interested in six

physically-meaningful fixed points, which are among our set of 50. Interestingly, the original work [153], on which the current work is based, used a block sequential update scheme, which was motivated by different activation times among the genes. GDSC can model this behavior, but the fixed points will be the same irrespective of the update scheme.

3.5 Limitations

Limitations of this work include: (i) a serial code for back end computations (a parallel code will speed computations); (ii) graphical representations of phase spaces could be improved; (iii) using XML file formats lead to larger file sizes (JSON formats would reduce file sizes); and (iv) inability to move between graphical representation of a network and a list view of the network in building networks for analysis. New features include the ability to sample phase spaces for larger networks and to specify vertex functions within the system itself.

Chapter 4

EDISON: A Web Application for Evaluation of Network-Based Social Dynamics

4.1 Introduction

Public health issues, from virus and disease transmission, to the spread of unhealthy behaviors (such as smoking and obesity) are global priorities. They can lead not only to fatalities, but also to decreased quality of life, large expenditures for health care, and the onset of other ailments.

In this chapter, we describe EDISON: a web-based modeling environment that can be used to perform complex computational experiments involving very general (epidemiological) contagion processes over social networks [7]. EDISON is publicly accessible by scientists and domain experts interested in carrying out in-silico social and epidemiological experiments.

4.1.1 Technical Challenges

To straight-forwardly incorporate high-level of heterogeneity into simulations, we need to build populations that contain various attributes [27], and simulations need to manipulate and process this information quickly. Specifically, applications need to enable a user to efficiently (i) search populations to identify members with particular attributes, (ii) assign to these members particular behaviors, (iii) transparently move large amounts of data within the system, and (iv) compute dynamics at scale. We address these issues herein. As a concrete example, one action by a user might be to identify all members of a population that are female, whose ages are between 15 and 31, and who interact with at least 12 males; EDISON must quickly search a population of, say, 10 million agents and return those matching these multiple criteria, who may then be assigned particular properties by a user. The payoff in doing so is large: these more nuanced inputs improve the quality of simulation results, and commensurately, our understanding of social epidemiological phenomena.

4.1.2 Contributions

1. EDISON: Open Access Web Application for Large Population Dynamics. This web application is a modeling environment for running social dynamics on network representations of populations. Populations are modeled as networks (vertices and edges representing, respectively, population members and their interactions). The spread of contagions is computed according to prescribed disease and behavior models. Multiple contagions and multinetworks maybe be used. For example, individuals may interact within social face-to-face contact networks and through social media such as Twitter and Facebook networks. Our focus is on large population dynamics; e.g., those with 10s of millions of agents and 100s of millions of interactions. The problems for which EDISON is ideally suited are those in the class called graph dy-

namical systems (GDS) [155] which generalizes several other models [30]. GDS was described earlier in Section 1.4.

EDISON consists of (i) a user interface (UI), (ii) a set of services (see Contribution 2), (iii) a simulation engine, InterSim, (iv) libraries of networks and disease and behavior models, and (v) middleware. The simulation engine is a distributed modeling framework suitable for the study of large networks, and an earlier version of it is described in [128]. EDISON runs on Virginia Tech-provided computers so that users need not have access to high performance computing (HPC) resources. A sophisticated middleware connects these various software components that run on different hardware. Finally, these components scale to large populations and result in a system that can serve multiple simultaneous users.

2. Integration with MARS Services. There are several software services of MARS services (more information will be presented in Chapter 5) that were integrated to meet big data challenges (e.g., querying a network based on its attributes, assigning particular properties to subsets of network elements). We highlight four. First, a labeled graph storage service stores directed and undirected networks that contain any number of attributes of types `short`, `int`, `long`, `double`, and `string` for both vertices and edges. Attributes from disparate sources can be added to the networks over time. Second, a graph measure service computes measures such as betweenness centrality, and these are also stored as attributes. Some measures are pre-computed at the time of network upload and others are computed on-the-fly as needed. Third, a graph query service uses a custom-built grammar that enables a user to select subsets of vertices and edges based on their attributes. These subsets are then assigned behavior model properties. In this way, properties of agents, for even large networks, can be specified quickly. In addition, multiple queries can be represented by one compact statement, making it easy to obtain multiple sets of graph elements. Fourth, a search service provides a user with all queries produced by all users to date. A user can se-

lect a query and use it directly or modify it. Together, these services enable a user to incorporate fine-grained population member attributes into simulations.

3. Usability. The web application significantly increases usability and productivity for two sets of users: novice users and simulation experts. The system provides automatic generation of simulation input files, which would otherwise have to be generated using scripting, which takes time, even for programming experts. More importantly, the UI and MARS enable domain experts to use the system who would not otherwise be able to, without a large learning curve (see Related Work in Section 2.2). The web app also promotes team science, since public analyses may be viewed and used by all users.

4. Computational Experiments. We provide computational experiments that illustrate its practical use for modeling the spread of contagions.

Chapter Organization. Following this section, an overview of EDISON is then described, with particular emphasis on the UI (Section 4.3), since this is the part of EDISON that is exposed to a user. However, we also introduce other system components in Section 4.4. Then we provide computational experiments that illustrate its practical use for modeling the spread of epidemiological contagions. Based on these descriptions, we holistically describe the usability of the system in Section 4.6.

4.2 Behavior Models

The simulation framework InterSim can implement any GDS. Consequently, GDS characteristics reveal the types of modeling that can be done with EDISON. We will use the term *behavior models* beyond this section because it fits with epidemiological and social science domains, but in particular for this chapter, this term essentially means *vertex function*. Generalizations of this system can be found in [128], which

includes edge functions, time-varying vertices and edges, and multiple simultaneous contagions.

Selected behavior model families currently implemented in EDISON are summarized in Table 4.1. For example, there are 6 different SIR model variants for Item 4. These models can be classified as *unilateral* models in the sense that a vertex updates its state independent of how other vertex states are updated. There are classes of models, such as *joint action* models (e.g., [63]), where multiple vertices change state through coordination. These models do not fit the GDS paradigm and hence cannot be efficiently implemented in EDISON.

GDS can be used to model many social and disease phenomena. However, each of these different types of models requires different properties. For example, the Ebola model in Figure 4.1 requires 12 double values per vertex and one double property per edge.

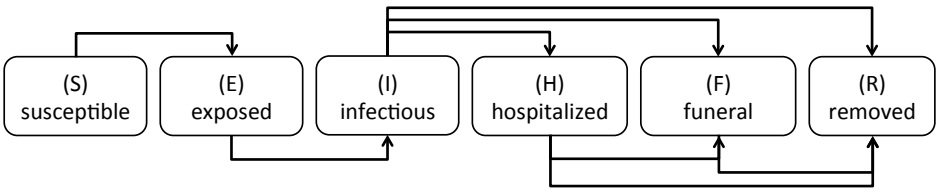


Figure 4.1. *An Ebola state transition model for a vertex, represented by a vertex function. Details of model equations are omitted due to space constraints; see [138].*

4.2.1 Need For Web Application Functionality

GDS can be used to model many (social) phenomena. However, each of these different types of models requires different properties. For example, the Ebola model in Figure 4.1 requires 12 double values per vertex and one double property per edge. The structured resistance model [168] requires over 10 double parameters per ver-

tex, for just three structural levels. Specifying a probability curve for the stochastic threshold model of [175] can easily require six double and six integer parameters, depending on data. The point is, to expose these model-dependent parameters so that they can be assigned values per vertex and per edge, and to enable versatile selection of vertices and edges to which these properties are assigned, EDISON is required to provide fine-grained manipulation of vertices and edges (and their properties), and this drives many of the features of the web application, described next.

Table 4.1. *Selected behavior model families in EDISON.*

Item	Model	Source/Example
1	2-state progressive threshold.	Simple, complex contagions (e.g., contraception) [58, 100].
2	2-state back-and-forth systems.	Starting and stopping behavior (e.g., yoyo dieting) [44].
3	Probability curve.	Generalization of basic threshold model; Twitter hashtag propagation [175].
4	Various forms of epidemic SIR models.	Examples: SIR, SEIR, SIS, with simple and complex contagions [58].
5	Generalized contagion.	For epidemics and social contagions [74].
6	Ebola.	Specialized to selected African cultures [138].
7	Structured resistance model.	For epidemics and social contagions [168].
8	Linear threshold model.	Social model used in many influence maximization studies [122].
9	Independent cascade model.	Social model used in many influence maximization studies [122].

4.3 User Interface

The UI guides a user through a series of screens whose contents adapt based on decisions made by the user up to that point. An analysis log screen, Figure 4.2, summarizes all analyses completed to date. Team science is possible, even among remote users, in that public analyses can be viewed by all users. Selection screens (e.g., for networks and dynamics models) provide a user with detailed information about graph and model characteristics to help the user make informed choices. We now focus on particular aspects of usability and big data processing.

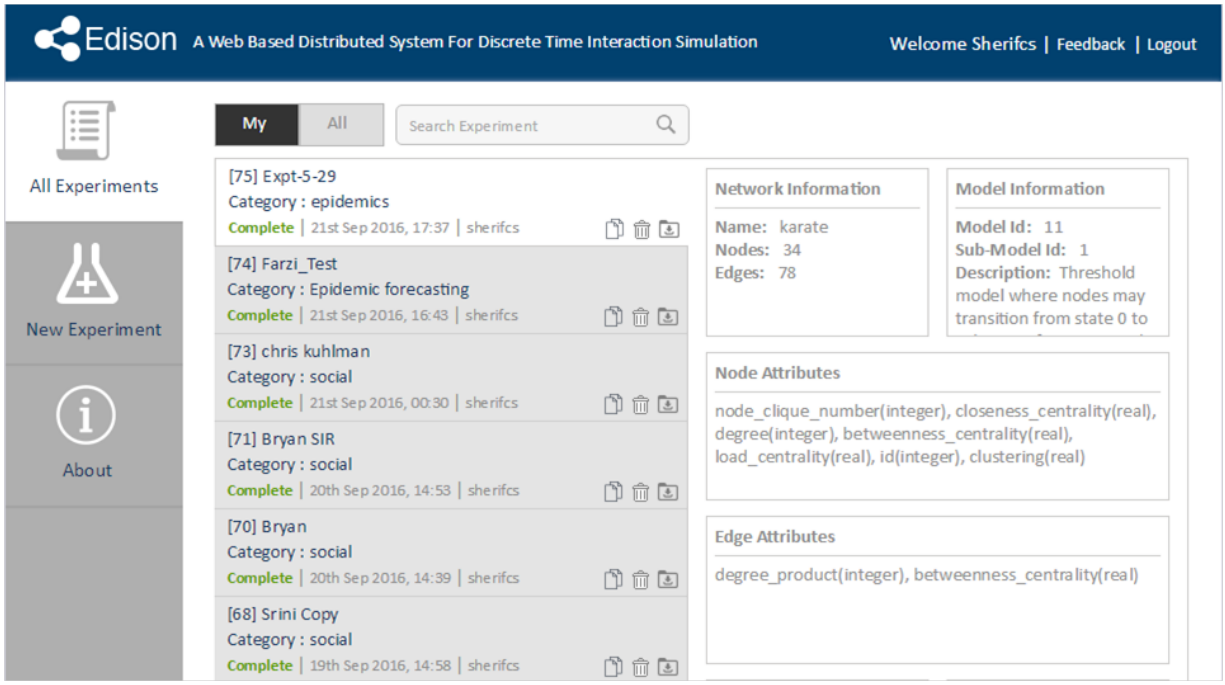


Figure 4.2. *Analysis Log screen of EDISON.*

Figures 4.3 and 4.4 illustrate key functionality of the system with respect to the manipulation of data in the form of population attributes and dynamics model properties. In the lower half of Figure 4.3, properties are listed; here, integer node states and traits. There are two integer node states: state_of_node (either 0 or 1) and is_fixed_state

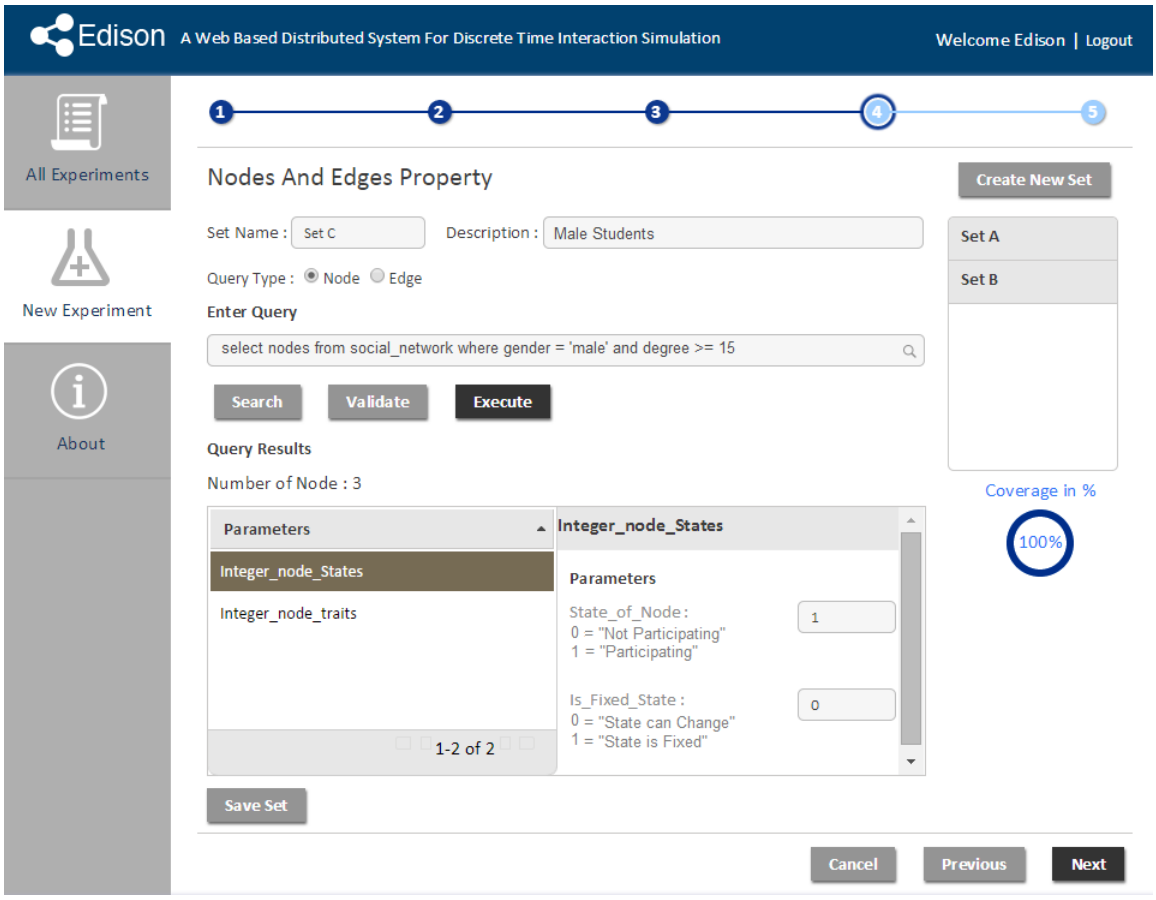


Figure 4.3. *The property specification screen.*

(either 0 for no or 1 for yes). If the latter value is specified as 1, then the group of selected nodes will not change their states in the simulation. This is an example of a simple intervention in that vertices may be fixed in state 0 and thus inhibit the transitions of their neighbors to state 1; other interventions will be given in the experiments. Values for a set of elements can also be assigned by specifying a distribution. Furthermore, model parameters for nodes and edges may be used directly from network attributes. For example, a model parameter may be an agent's age. If age is also an attribute of network vertices, then these may be loaded directly as values for model parameters. However, other ages may also be assigned (e.g., for sensitivity studies).

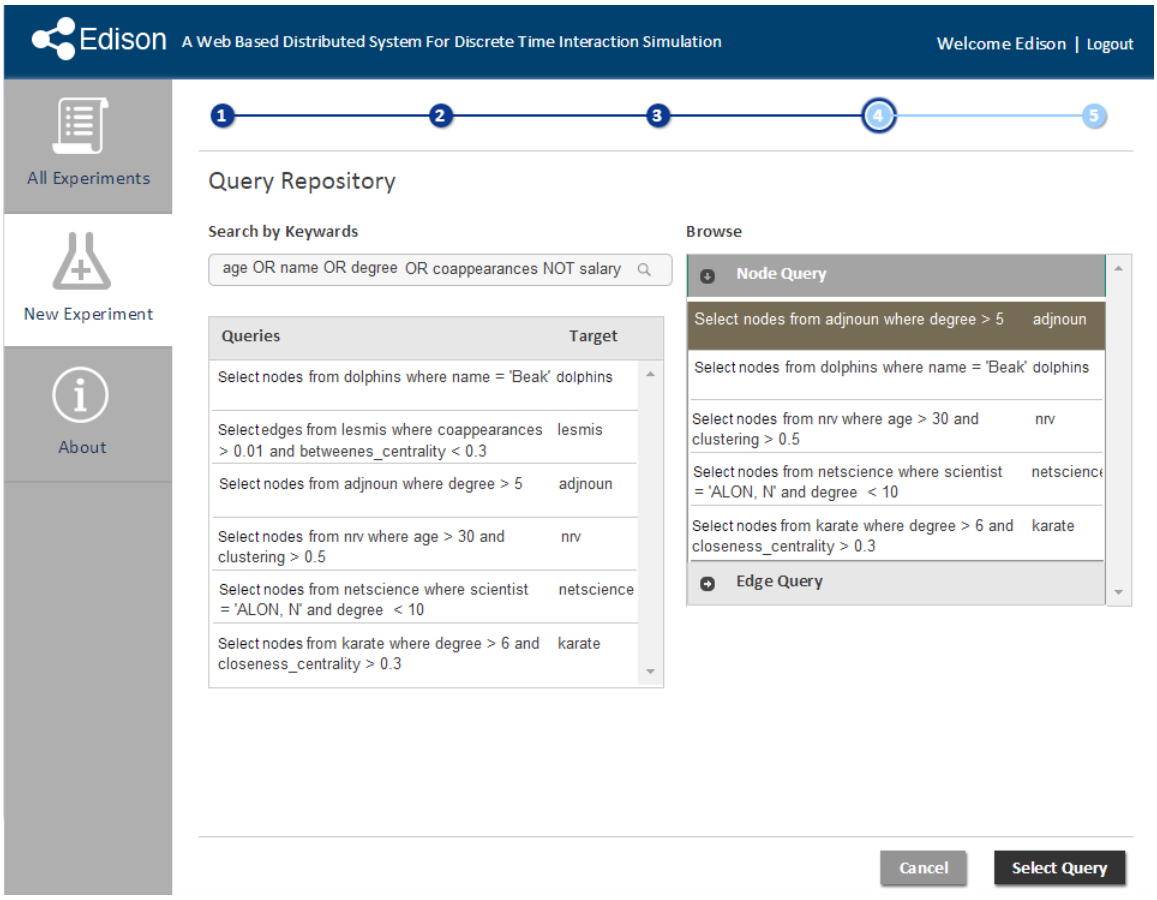


Figure 4.4. *The query search screen.*

The question is, what nodes will be assigned these properties? The upper portion of the screen enables a user to select vertices (and edges) of the graph through queries and then assign properties to the elements of the sets. A user can query the network about its attributes and structural parameters such as degree, clustering coefficient, and edge betweenness. The set of nodes or edges returned from the query are stored and these nodes or edges are assigned properties as described above. The sets of nodes and edges can be queried and set operations can be executed. To assist query specification, if a user clicks the *Search* button below the query, he or she is taken to Figure 4.4.

In this figure, a user queries a repository using keywords. Queries matching those keywords are returned. The user can select from among them. Furthermore, node and edge queries can be browsed (at the right). A selected query is returned to Figure 4.3 where it can be used as is, or modified before it is executed. Queries can also be tested for validity before submission. These features greatly enhance a user's experience in specifying properties for an analysis, and at fine granularity.

The same features in Figures 4.3 and 4.4 are also used to select seed sets (i.e., initial conditions) for each run of an analysis. A run is a single diffusion instance. An analysis can have any number of runs; a typical simulation involves somewhere between 20 and 100 runs, but larger numbers are used, depending on circumstances. Special queries are provided to enable users to generate multiple sets of nodes and edges so that properties of collections of sets (for more than one run; and often, all runs) can be specified with one query. There are random set capabilities also, so that a user obtains sets of nodes or edges at random from a superset of elements. This is another aspect of the system's usefulness in streamlining inputs while giving *more* control to a user.

Example 5. Suppose we want to specify properties for an Ebola model for citizens of Liberia, Africa. We want all citizens to have the same properties of infectiousness, etc., except for people between the ages of 10 and 19 years with at least 15 interactions a day with other people. We do this in two steps. First, we form and execute the query:

```
select nodes from liberia
```

We then assign Ebola model properties to all nodes (agents), consistent with the query. Then, we form and execute a second query:

```
select nodes from liberia where age>=10 and age<=19 and degree>=15
```

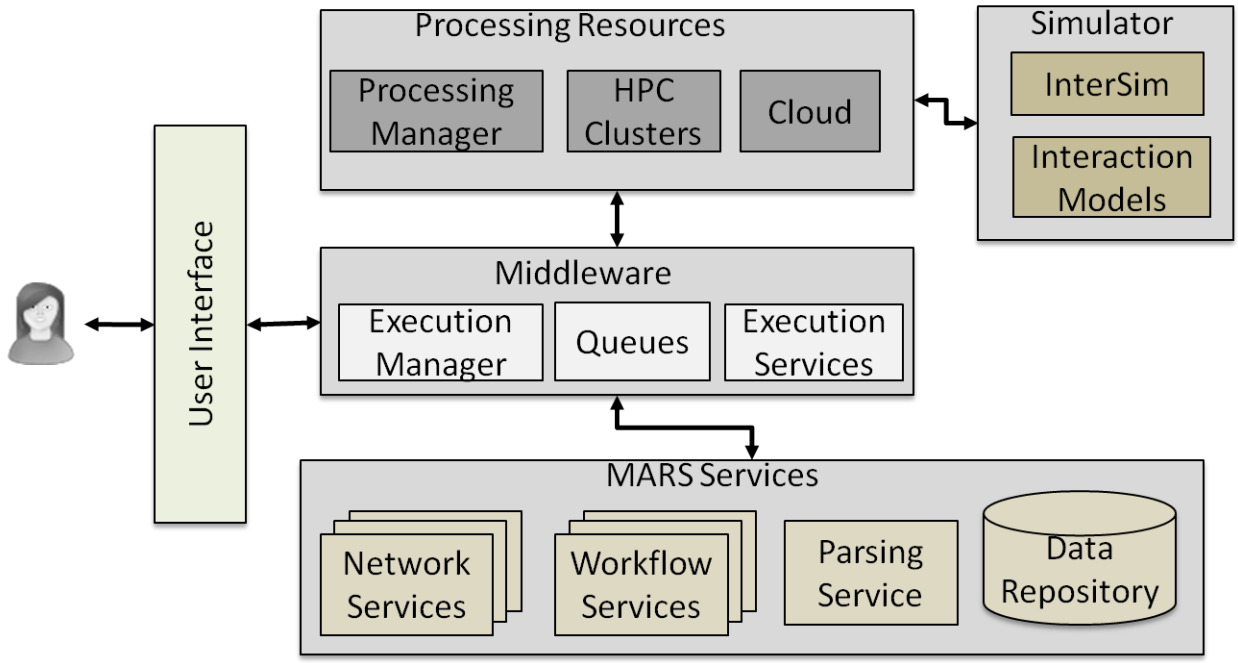


Figure 4.5. *EDISON main system components: UI, MARS services and data repository, hardware resources, simulation engine, middleware, networks (graph and attribute), and behavior (interaction) models.*

We then specify Ebola properties for these nodes (agents), which overwrite the properties from the first query—but only for this subset of agents. In two queries (and with specifying two sets of properties), the model assignments are complete. This is quite fast; and note that the model properties are assigned based on both agent attributes (age) and network measures (degree).

4.4 System Architecture

A logical view of the EDISON architecture is given in Figure 4.5. The UI was covered in more detail in Section 4.3, since it exposes system functionality. Here, we provide brief descriptions of the architecture and selected components of the system that are

particularly relevant for our application space and big data processing. These are the middleware, selected MARS services, simulation engine, and processing resources.

4.4.1 Middleware

The Java Messaging Service (JMS) is used for routing requests for services and data through the system. A Job or Execution Manager places requests on a message queue (not a classic queue, but rather a component that guarantees that each request is processed only once). Each job (request) is routed to particular services that can fulfill that request, where the first available service takes ownership of that job. There may be many service requests that are executed for one user on each of the screens in Figures 4.3 and 4.4. Requests contain various information/requirements; e.g., data, metadata, required output, and possibly quality-of-service requirements. The manager also exposes REST APIs for consumption by various services (e.g., database, query, computation). Rules are provided to direct requests to appropriate services. From a practical standpoint, the middleware is designed to service many users simultaneously. There is also a monitoring system to provide (realtime) information on system execution.

4.4.2 Integration with MARS

MARS plays an important role within EDISON. There are different software services identified in Figure 4.5. In Chapter 5, we will address in more detail all the *Services* that are particularly important to EDISON and big data.

4.4.3 InterSim Simulation Engine

InterSim is the modeling framework used for simulations in EDISON. An earlier MPI-based version is described in [128]. The current version is a hybrid system (MPI + multithreading), best suited for HPC multicore clusters. It can implement any behavior model that conforms to a GDS (see Section 1.4). The system isolates the behavior of vertices and edges, so that new functionality can be surgically integrated without affecting the framework or other implemented behavior models. It can simulate the spread of multiple contagions over multiple networks. The experiments discussed in this chapter provide example analyses, including the use of interventions, which are important for supporting policy development. Sample performance data on a 9 million vertex network of Chicago is provided in Figure 4.6. From these data, InterSim exhibits strong scaling; i.e., the system continues to make efficient use of resources as more compute cores are used in computations.

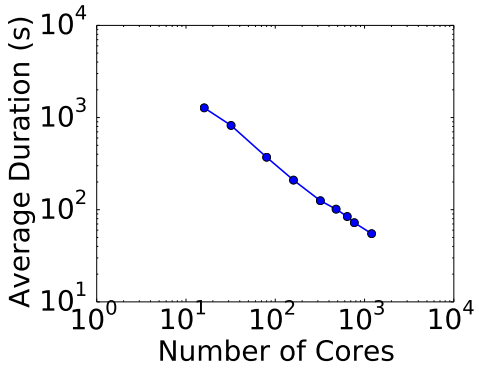


Figure 4.6. *Strong scaling using a Chicago network: execution time for one run of 100 time steps, using a stochastic behavior model of social participation.*

4.4.4 HPC Hardware Resources

Jobs are submitted through a resource (processing) manager to either Virginia Tech-owned compute clusters or to cloud services, based on a set of pre-defined rules. In this way, multiple jobs can run concurrently.

4.5 Illustrative Case Studies

4.5.1 Networks and Simulations

4.5.1.1 Networks

Networks are provided in Table 4.2, along with the type of network, numbers of vertices and edges, average clustering coefficient c_{ave}, and average degree d_{ave}. The clustering coefficient of a vertex v is the number of edges between pairs of distance-1 neighbors of v, divided by the number of possible edges. NRV is the New River Valley, Virginia. MVC is Montgomery County, Virginia. The human contact networks were developed from synthetic population generation procedures [27] that capture people's activity patterns and daily interactions. The remainder of the networks [140] represent remote forms of communication between people.

4.5.1.2 Simulations

All simulations are performed with EDISON. A simulation is a set of contagion diffusion runs. Each run is comprised of network, with properties of interaction models assigned to vertices (agents) and edges (interactions). A subset of vertices are "seed nodes," meaning that they are initially "infected" or "participating" at time $t = 0$; these are typically different for each run. Dynamics emanates from these seed nodes

Table 4.2. *Networks used in experiments [27, 140].*

Networked Pop.	**Pop. Type**	**Num. of Vertices**	**Num. of Edges**	c_{ave}	d_{ave}
Liberia, Africa	Human contact	3.674M	31.67M	0.408	17.2
Portland, OR, USA	Human contact	1.57M	19.48M	0.511	24.7
NRV, USA	Human contact	153K	4.15M	0.393	54.4
MVC, USA	Human contact	77.57K	2.01M	0.396	52.0
Epinions	Online	75.87k	0.5M	0.138	10.7
Slashdot	Online	77.36K	0.905M	0.0555	14.1
Enron	Email	33.69K	180.8K	0.509	10.7

over discrete times, until some specified maximum time t_{max}. All of these data are specified in EDISON; we use 50 runs per simulation. We plot the point-wise averages of results in the figures; e.g., if we perform 50 runs and we plot the average fraction of infected vertices at time $t = 23$, then this value is the average of the fraction of infected vertices at $t = 23$ over the 50 values that come from the 50 diffusion instances.

4.5.2 Simulation Results

Three experiments and results are described. These are: (i) SIR influenza epidemics on human contact networks of regions of the U.S., (ii) the process of registering for an internet-based health forum via online influence, and (iii) the 2014 Ebola outbreak in Liberia, Africa. For each, we present motivation and results generated with EDISON. Then we briefly illustrate how EDISON's capabilities are used to produce the results.

4.5.2.1 Influenza in Three Populations: Case Study 1

The application. We use an SIR dynamics model to simulate influenza on three networks of Table 4.2: MVC, NRV, and Portland. We choose MVC and NRV because the networks, though of different sizes (the latter has twice as many vertices and edges

as the former), are from roughly the same geographic region in Virginia. We also use Portland, OR, which has an order of magnitude greater number of vertices (people) but only one-half the average degree. We assume that all edges have a nominal interaction duration of 2 hours. Infectiousness (given by edge weight w), and duration δ of infectiousness, are both varied in simulations. In each run, 50 vertices are chosen uniformly at random and seeded with influenza (flu). For each susceptible vertex i at each time t, each infectious distance-1 neighbor j attempts to infect i and succeeds with probability $w_{ji} = w$.

Figure 4.7 shows results for Portland. Figure 4.7a contains numbers of new infections at each time (so-called epi-curves), and Figure 4.7b shows the cumulative number of people infected as a function of time. The different colors correspond to different edge weights (transmission levels) and the dashed vs. solid curves correspond to infectious durations of 5 days and 3 days, respectively. As w and δ increase, the peak in numbers of new infections increases in magnitude and occurs at earlier times.

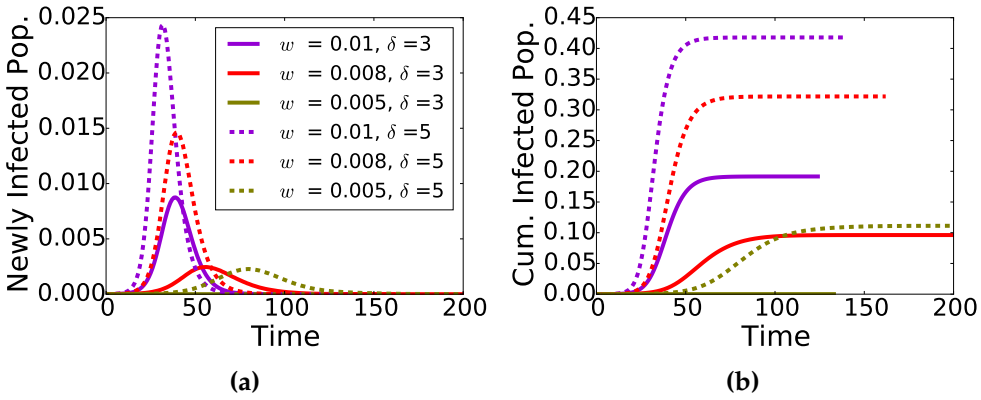


Figure 4.7. *Epidemic simulation results for Portland. (a) Fraction of new infections in time. (b) Cumulative fraction of infections in time. The legend applies to both plots: δ is the infectious duration in days and w is the edge probability of disease transmission.*

Figure 4.8 shows the time to infect 10% of MVC, NRV, and Portland populations for $\delta = 3$ and three edge weights. MCV and NRV are in close agreement (as are their

respective epi-curves) for all combinations of w and δ. The Portland data differ significantly; this is a size effect, Portland is much larger than the other two regions. The data also show that there is a significant difference between $w = 0.008$ and 0.009 for Portland: as w increases by 0.001, the time to infect 10% of the people decreases by a factor of four.

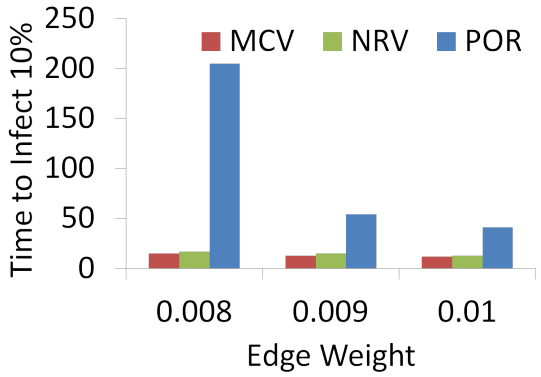


Figure 4.8. *The time taken for 10% of each of the NRV, MVC, and Portland populations to become infected for various edge weights w. Here, $\delta = 3$ days.*

Modeling with EDISON. A total of 36 simulations are performed (3 networks $\times$ 6 values of w $\times$ 2 values of δ). For a given network, once one set of simulation inputs is specified, these input sets can be copied, and then only the values that need to be changed in the copy are modified, and the job is submitted. That is, only w and/or δ change in successive sets of inputs. So specifying the simulation parameters is fast. All SIR model parameters are specified with one vertex query (to assign δ) and one edge query (to assign w). All 50 seed sets per simulation are generated with one query (even though the seed sets are different).

The net result is that inputs for a simulation for each network, started from scratch, might take about 10 minutes for a novice; less than 3 minutes for an experienced user. The remaining analyses, which only require selected inputs to change, may take roughly 1-2 minutes. This is why web applications—even for experienced software

developers—are far more efficient than using command-line processing to execute jobs through scripting.

4.5.2.2 Participation in a Health Care Forum: Case Study 2

The application. This study is motivated by online social experiments [57]. These experiments were designed to determine the social influence required by users to join an online health care forum. Specifically, the purpose was to determine whether the process of registering was one of complex contagion [58], where users require reinforcement to participate. That is, a user v's threshold θ_v means that v, who is currently not participating, will participate if at least θ_v of its immediate neighbors are already participating. Otherwise, v will remain as not participating. Complex contagion occurs when a user's threshold is strictly > 1, so that he or she requires more than one participating neighbor (i.e., reinforcement) to join. Once participating, a user remains in this state; i.e., once a user joins the forum, one remains registered, and one contributes influence to each of his or her distance-1 neighbors who are not yet participating.

Centola [57] finds that experimental subjects participate after 2 to 4 of their neighbors are already participating; i.e., that $\theta \in \{2, 3, 4\}$, as evidence for complex contagion. Here, we study thresholds $\theta = 1$, 3, and 5. Our threshold model is deterministic; if an agent's threshold is met, he or she transitions to the participating state.

Actual networks from the study are not available and are smaller in size (the maximum number of vertices in experimental networks was 144), so we use three online networks from the literature with tens of thousands of vertices: Epinions, Slashdot, and Enron email networks (Table 4.2). We use these three because there is an order of magnitude range in c_{ave} among the networks, and we choose clustering because it is a measure of friend-of-friend relationships.

We choose seed vertices—those that are initially participating—in two ways. First, we compute the median clustering coefficient γ of all vertices in each of the three networks (0, 0, and 0.500 for Epinions, Slashdot, and Enron, respectively). Then, we perform one set of experiments where all 300 seed vertices for each run of a simulation are chosen from vertices that have clustering coefficients c less than or equal to γ and another set of experiments where all 300 seed vertices have $c > \gamma$.

Figure 4.9a provides data for the Epinions network: curves are time histories of the cumulative fraction of experimental subjects that are participating. Each curve represents data where all subjects have the same specified threshold θ. The value of $c \leq \gamma$ (dashed curves) or $c > \gamma$ (solid curves) dictates the subset of vertices from which seed vertices are selected. As θ increases from 1 to 5, the cumulative fraction of participating agents decreases from 1.0 (red curves) to 0.2 or less (green curves). This figure shows interesting interactions between θ and c. For smaller thresholds ($\theta \leq 3$), the clustering coefficient of seeded vertices has no effect on the final fraction of vertices participating. It does, however, have an effect on the early stages of the dynamics: initially, contagion speeds faster from the higher-clustering vertices (solid red and blue curves). As threshold further increases to $\theta = 5$, the seeding now has an effect on the final fraction of participating vertices: when $c > \gamma$, the final fraction is 0.2; when $c \leq \gamma$, the final fraction is an order of magnitude less (green curves).

Figure 4.9b contains data for all three networks. The final fraction of participating agents is plotted against the uniform threshold value assigned to all vertices. These data are for seeding of the high clustering agents ($c > \gamma$). Data from Figure 4.9a are represented by the orange curve, and the other two curves are generated from comparable data on Slashdot and Enron networks. It is interesting that although Slashdot has one-half the average clustering coefficient c_{ave} of Epinions, it shows larger contagion spread than Epinions. Often, the greater the clustering, the better suited a network is to propagate complex contagion because of the greater local connectivity.

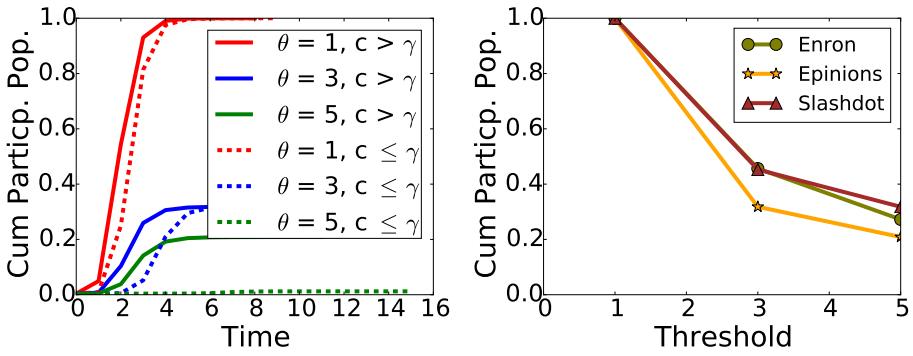


Figure 4.9. *Complex contagion simulations ,examining the effects of seeding and threshold θ on contagion spread. Here the application is people joining a health care forum, and those who have registered are* participating *(Partcip.). (a) Cumulative fraction of participating agents for the Epinions network. Data are given for three thresholds θ and two types of seeding: (i) vertices with lower clustering coefficient ($c \leq \gamma$) and (ii) those with greater clustering coefficient ($c > \gamma$). (b) Cumulative fraction of infections at steady-state as a function of threshold for three networks, where seeds are vertices with $c > \gamma$. The average clustering coefficient c_{ave} ranges over an order of magnitude for the three networks.*

However, in this case, Slashdot has a 40% greater average degree, meaning greater global connectivity; see Table 4.2. Enron also shows greater spreading. Its average degree is about the same as that of Epinions, but its average clustering coefficient is over 3× that of Epinions, so there are more friend-of-friend relationships to provide reinforcement for contagion spreading.

Modeling with EDISON. This study is comprised of 18 simulations using EDISON (3 networks × 3 values of θ × 2 seeding conditions). For each network, one simulation requires specification of all inputs. For subsequent simulations, the network and the dynamics model do not change, so the simulation specification can be copied and used as a starting point for all subsequent simulations for a given network. The single query within EDISON for specifying thresholds for all vertices must be repeated for each simulation because the threshold of vertices changes. The single query for specifying 50 sets of 300 seed nodes per run of each simulation is repeated for each

simulation because we want to randomize the seeding process. The times to specify inputs for a simulation are similar to those described in Section 4.5.2.1.

4.5.2.3 Ebola in Liberia, Africa: Case Study 3

The application. In this experiment, we model the Ebola outbreak in Liberia, which reached its peak in 2014. Roughly 10,000 cases of Ebola were confirmed or suspected in Liberia, in the main outbreak. We use a pre-existing synthetic population [27] and social network of Liberia. The contagion model of Ebola transmission in each human is specifically designed to include cultural factors (e.g., hospital visitations, funerals) [138], and has been used to model other Ebola outbreaks over the last ten. See [172] for more details from a compartmental modeling perspective. We run 15 simulations to calibrate the disease transmissibility τ in the model to produce roughly 10,000 Ebola infections in 400 days. The transition of an agent from state S (susceptible) to E (exposed, meaning an agent has contracted the virus but is not yet infectious) is governed by τ. We find that $\tau = 1.76 \times 10^{-6}$ gives an average outbreak size of 9870 in 400 simulated days (over 20 distinct runs), which agrees well with observations of about 10,000 infections in a little over a year. We use all other model parameters similar to those from an existing study of Ebola conducted on Liberia [172].

With this calibration effort, we investigate various interventions, using age and gender. We studied the effects of (i) providing vaccinations to people based on (a) age and (b) gender; and (ii) different efficacies of vaccines. For example, if a vaccine is 80% effective, then the people receiving the vaccine have their transmissibility reduced to $(1 - 0.8)\tau$. Vaccines are in various stages of development[1].

Figure 4.10a shows the base fraction of people infected and the fractions resulting from vaccinating all people in age ranges 21-30, 31-40, and 41-50. These age ranges

[1] See, for example, http://www.who.int/medicines/emp_ebola_q_as/en/

are the most effective to vaccinate (confirmed through simulation), since they have the most aggressive combinations of numbers of people and numbers of activities of the people. Activities dictate numbers of interactions with others, which are necessary to spread infection. These data are for a very potent vaccination that is 80% effective; they indicate that vaccinations can reduce outbreak sizes to 1/3 of base levels, or less. For each of the age groups 21-30, 31-40, and 41-50 years, the numbers of (males, females) are, respectively, (308956, 316221), (238375, 232595), and (136568, 147596). Hence, vaccinating those in the 31-40 age range, with 25% fewer people than in the 21-30 age range, produces less than one-half the illnesses that occur when vaccinating the latter group.

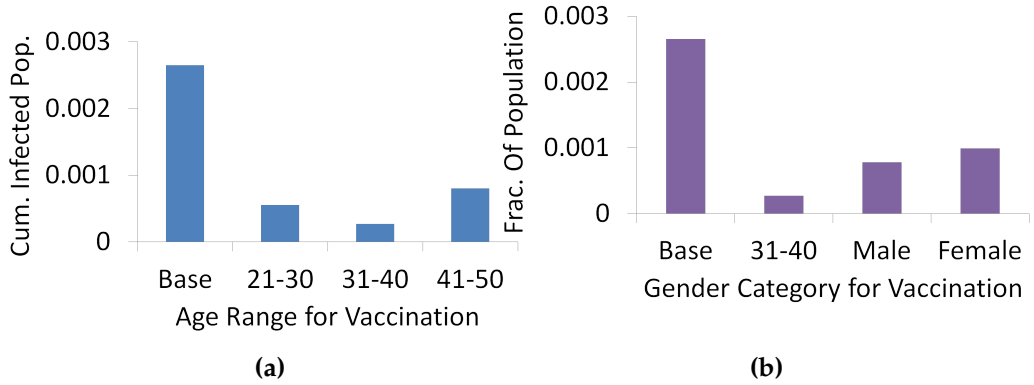


Figure 4.10. *Simulations of Ebola outbreaks in 2014 in Liberia, Africa. In each plot, the fraction of the population that has contracted Ebola over 400 days is plotted as the* Base *case. (a) shows the effect of vaccinating all people in the specified 10-year age ranges, where the vaccine is assumed to be 80% effective. (b) shows the effect of vaccinating all people in the 31-40 age range, by gender.*

Figure 4.10b further breaks down the people vaccinated in the 31-40 age range by vaccinating each gender separately. The vaccination of males is more effective. Simple numbers of men and women explain these results; there are 2.4% more men than women. Note that this contrasts with Figure 4.10a, where the most effective group to vaccinate is not the largest group.

To assess the impact of vaccination efficacy, we vaccinate the 31-40 age group with different levels of effectiveness. Figure 4.11 contains the results. The incremental effectiveness of a vaccine starts to decrease after the effectiveness increases beyond 40%.

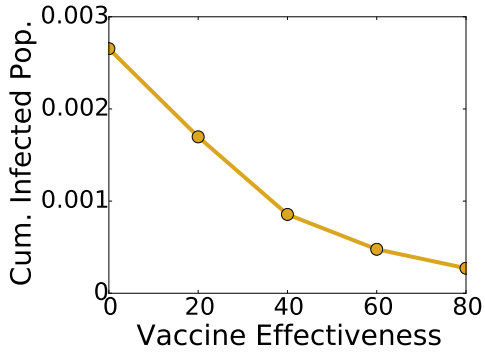


Figure 4.11. *The effect of vaccine effectiveness on cumulative fraction of population infected with Ebola for Liberia, when all people in the 31-40 age range are vaccinated.*

Modeling with EDISON. We conducted 25 simulations; 15 for calibration and nine more for the study. As with other studies herein, EDISON enables users to query and obtain subsets of nodes (and edges) and then specify particular properties of these subsets. Here, we query and obtain sets of vertices based on age and gender, as described above, and assign these different sets different values of transmissibility.

We focus here on the differences between this study and the others. Here we perform calibration analyses to obtain virus transmissibility that matches observations, namely, number of infected people in Liberia. For this, we ran three sets of simulations. In the first, we ran the model with transmissibility by decade, from 0.01 to 1×10^{-7}. Then we ran another set of simulations at a prescribed interval between 10^{-5} and 10^{-6}, and then finally a third level of calibration to arrive at the τ value given above. It is important to note that all simulations at one level were run simultaneously in EDISON, to reduce the wall clock time to achieve calibration. This calibration process is quite standard. With the calibrated transmissibility, all other simulations

could be run simultaneously.

4.6 Usability

We have presented several aspects of usability: power (what you can do), ease of use (how easily can you do it), productivity (how fast can you do it), concurrency (how many users can do these things at the same time), and accessibility (who can use the system). Power comes from the GDS model and simulation engine; from the MARS services that enable a user to assign any (valid) values to behavior model parameters and to select vertices and edges to which those properties are assigned; and the UI that exposes this functionality. Ease of use and productivity come from the UI; crucial elements are the screens that allow query-like statements to select subsets of vertices and edges for property assignments (including packing multiple queries in one statement) and a search service where a user can utilize and adapt previously executed queries. Productivity also comes from the scalability of the simulation engine. Concurrency is realized from the system architecture (e.g., client-server architecture of the web app, scalable middleware) and compute resources that can execute multiple service requests and simulations simultaneously. Accessibility of the system is provided by the compute resources provided through Virginia Tech. Usability evaluation of EDISON is provided in chapter 8.

4.7 Limitations

Among the limitations of this work are: (i) the inability to add new models through the UI; (ii) the absence of post-processing capabilities (e.g., plotting of results); (iii) the inability to specify seed sets beyond specifying random sets; and (iv) the inability to

specify different behavior models across nodes and edges of the same network.

Chapter 5

MARS: Network Services and Their Compositions for Network Science

5.1 Introduction

Network science is moving more and more to computing *dynamics* on networks (so-called contagion processes), in addition to computing structural network features (e.g., key players and the like) and other parameters. Generalized contagion processes impose additional data storage and processing demands that include more generic and versatile manipulations of networked data that can be highly attributed. In this chapter, we describe a new network services and workflow system called MARS that supports structural network analyses and generalized network dynamics analyses [8]. It is accessible through the internet and can serve multiple simultaneous users and software applications. In addition to managing various types of digital objects, MARS provides services that enable applications (and UIs) to add, interrogate, query, analyze, and process data. We focus on several network services and workflows of MARS in this chapter. We also provide a case study using EDISON that MARS supports,

and several performance evaluations of scalability and work loads. We find that MARS efficiently processes networks of hundreds of millions of edges from many hundreds of simultaneous users.

5.1.1 Technical Challenges

There are many challenges for building a system to support generalized contagion dynamics simulations on large networks (those with 1 million or more vertices). These include the following. (*i*) *Designing modular, interoperable* categories *of services, where each service is a distinct, relocatable process.* These designs enable more flexible compositions of workflows and can exploit data locality. (*ii*) *Providing stateless (REST-ful) and stateful (i.e.,* session*) services.* REST APIs are ubiquitous in service oriented architectures, but network science applications also require stateful sessions with MARS. A session may be comprised of multiple interacting service and workflow requests to accommodate domain logic that requires software control structures (such as IF or WHILE) to execute between calls to the MARS API. A simple example is bookkeeping for multiple queries of network edges to ensure that properties for all edges are specified in a dynamics model. (*iii*) *Developing an SQL-like query grammar with special features for network dynamics.* Querying networks for dynamics involves not only querying for sets of vertices and edges, but also manipulating recursively the return sets and performing set operations. These are handled by our grammar, which can be viewed as an extension of SQL for the relational model. As part of a larger effort on cyberinfrastructures for network science, we are also studying the resource description framework (RDF) of subject-predicate-object triples and SPARQL query approaches. (*iv*) *Fast processing of highly attributed big data to support UI responsiveness.* Supporting UIs provides an important use case for MARS. For example, if a client application's sole job is to compute measures on large networks, and these are sub-

mitted as batch jobs, latency is less of an issue. However, in MARS, computing graph measures is not the end goal; it is a human-in-the-loop task that is part of the larger goal of computing network dynamics. Hence, for user experience when working through a UI, the services must be efficient. We provide performance evaluation data in Section 5.6.

5.1.2 Contributions

A summary of our major contributions follows.

1. MARS Workflow Service. MARS is a server-based workflow system developed for network science scientific computing. It operates by concurrently servicing multiple users and multiple applications and is accessed through the internet, using a well-defined API. It was designed to support both GUI functionality as well as HPC computational requirements, but the system is application-agnostic and thus not tied to either. It contains a repository along with several categories of services, but we confine ourselves to network services and their workflows. These services store networks, compute measures on them, manipulate subsets of network vertices and edges, and query data. MARS also uses external applications to support services, thereby increasing its capabilities. These applications currently run on an HPC computing cluster (MARS uses a PBS scheduler to launch executables) and are oblivious to the services that invoke them. MARS can be reconfigured to run these third-party codes on other platforms.

The services are stand-alone executing processes that can reside on different compute nodes for increased performance through data locality and for flexibility in mapping processes to hardware. As we will demonstrate, workflows are composed not only of sequences of services, but also of interleaved functions across services. Both stateless (i.e., REST-ful) and stateful (i.e., *session*) interactions with external applications

are supported. All aspects of stateful interactions (hand-shaking, coordinating multiple requests by the same application for the same session) are handled completely by MARS. A customized grammar for SQL-like queries is used to support special requirements for network dynamics. While network science is the target application for this work, a large part of this system is general; e.g., a new query parser could be inserted for another application and the software used to compute measures on networks can be changed out for other applications.

2. MARS and Big Data. The MARS repository houses many types of digital objects (DOs). There are multiple categories for DOs; we focus here only on the networks that are stored for simulations because they are central for computations in this domain. We use the relational database model and an SQL-like grammar. Currently, the largest networks within MARS have 9 million vertices and 406 million edges. Networks may be directed or undirected, and may have any number of vertex and edge attributes, of any primitive type (e.g., integers, doubles, strings). With advancements in data mining and machine learning (e.g., [107]) to infer properties of networks, the ability of MARS to store domain attributes greatly expands the range of possible network dynamics simulations. For the services described herein, MARS works with multiple data formats, taking computing performance and data storage into consideration: (unformatted) flat ASCII files, relational DB tables, and formatted (e.g., JSON) data objects. All of these are used in particular places, depending on requirements.

3. Illustrative Case Study. We provide a case study of contagion spread on a Facebook network using EDISON. The dynamics model is a complex contagion and we show that the dynamics are affected by the seeding conditions (i.e., what vertices of the network initially possess the contagion), and by threshold assignments. Threshold is the parameter that characterizes dynamics. We then identify the services and workflows of MARS that are used to support the computations. Other EDISON studies that utilize MARS are provided in [7].

4. Performance Evaluation. A collection of performance evaluations is provided. They focus mainly on the services for network query execution and computation of network structural parameters, since these are most compute-intensive. Results are computed for various networks to reveal how performance changes with network size, and demonstrate that MARS services can be executed at scale on networks that range from a few thousand to hundreds-of-millions of edges, with hundreds of simultaneous users. When used with UIs, where a human is in-the-loop, execution time affects user perception of the UI application.

Chapter Organization. MARS is overviewed in Section 5.2. Selected network services and workflow services are presented in Section 5.3 and 5.4. For these services a case study is provided in Section 5.5 and performance evaluations are provided in Section 5.6. Related Work and Conclusions form the last two sections.

5.2 MARS System Overview

Figure 5.1 provides a system view of MARS. The boxes in the large central gray box represent categories of services. The Repository (including the relational database management system, RDMS) is central, as it houses a variety of DOs through an expanding collection of tables. The Search Engine Service (SES) is a low-level service that provides support for several other services.

In this chapter, we focus on four of the Network Services and two of the System Workflow Services (SWS) of Figure 5.1. While many services may provide isolated results, almost all of them also interact/cooperate through workflows in order to accomplish user-requested tasks. Each service is a stand-alone process; services communicate via messages through a REST implementation and use well-defined APIs. The services may be constructed in any programming language. All services to date are written in

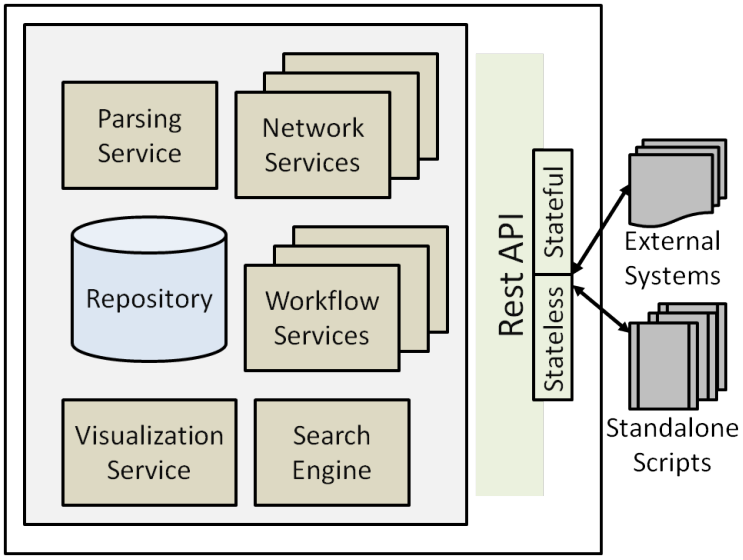


Figure 5.1. *Overview of the MARS system.*

Python.

External systems may request stateful interactions, and in these cases, MARS handles connections, handshaking, state storage, and storage updates with successive service requests within one session. When a session ends, these resources are automatically recouped. These tasks are transparent to the application making the requests.

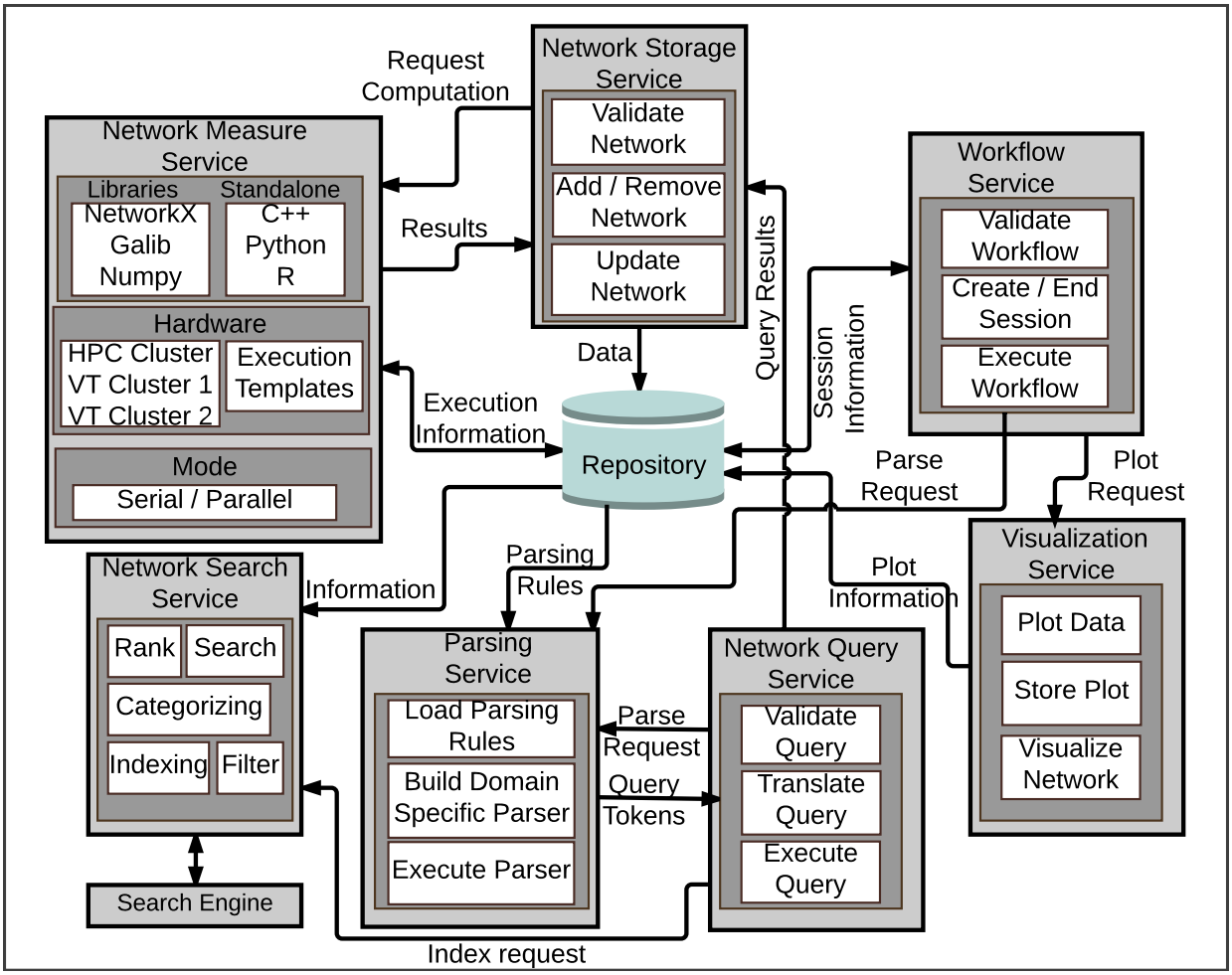


Figure 5.2. *Selected interactions among services are illustrated. Boxes here correspond to boxes in Figure 5.1, with same colors. Each service can be a separate process and can run on different servers.*

5.3 Network Services

Selected major Network Services from Figure 5.1 are provided in Figure 5.2, along with a few representative interactions. All are used in various workflows (Section 5.4). Several low-level details are not addressed here, but are implemented; an example is DB table indexing.

5.3.1 Network Storage Service, NStS

NStS supports directed and undirected networks, and any number and type of vertex and edge labels (also called attributes or properties). There are two types of attributes: domain-based and structural. Structural attributes are generated within MARS using the Network Measure Service, NMS (below). For each network added to the system, two tables are created to store vertices and edges and their attributes, and properties of the network are added to a network table, immediately and over time. Thus, these tables of attributes can grow and shrink and table schemas correspondingly change. NStS supports these and other actions; cf. Figure 5.2. Many networks have five to seven vertex attributes (one has 59) and a few edge attributes. Simple and multi-edge graphs, and those with self-loops, can be stored with these schema, but they are not well suited for hypergraphs.

5.3.2 Network Query Service, NQS, and Network Query Parsing Service, NQPS

The NQS and NQPS parse, validate, and execute queries submitted through the API. NQPS contains a custom-built grammar that interprets SQL-like queries and specialized queries, and prevents SQL-injection. We chose SQL because of its widespread use. These queries are performed on networks, and return sets of vertices or edges, or

subgraphs in JSON format.

Illustrative queries are provided in Table 5.1. *Simple queries* return sets of vertices or edges based on their properties. The properties may be domain-based or structural parameters; e.g., see the first query involving clustering coefficient and age. *Mixed queries* possess WHERE clauses that are of different type than the return set type. For example, one may select a set of edges based on the properties of incident vertices (and also of edge properties). *Sampling multisets* returns *number* of sets of vertices or edges from a *single* query. The query is separated into a *number* of distinct queries that can be executed in parallel if the DBMS supports concurrency. This capability is useful in obtaining seed sets for multiple simulation instances. NQS is an example where connection *sessions* (i.e., state-ful interactions with MARS) can be useful. In these sessions, users may require multiple sets of vertices, for example, and specify different dynamics properties for each vertex set. A session is required to keep track of which vertices have been selected (all vertices must be assigned properties). NQS also performs *set operations*, which are useful in manipulating previously existing vertex and edge sets and their elements. The return sets from a query contain the vertex (or edge) IDs, and may also contain all of the graph element properties, depending on an interface argument. This enables flexibility and efficiency in terms of response time and data storage for the calling application, for large data manipulation.

Table 5.1. *Examples of four types of queries handled by the NQS and NQPS.*

Query Example	**Query Type**
select nodes from chicago where degree >40 and clustering ≤ 0.6 and age > 18 and age ≤ 25	simple
select edges from epinions where u.degree >72 or v.degree >72 and u.clustering <0.5 and v.clustering ≥ 0.5 and u.gender = M and u.age >60	mixed
select sample(number= 10, [10,30]) nodes from google-web where age >30	sampling multisets
setA union setB except setC	set operation

5.3.3 Network Query Search Service, NQSS

To improve user experience, all valid queries from all users are retained by the system and are provided to a user in searchable catalog form. The goal is that a user does not have to compose queries from scratch, but rather can search for an existing query that matches one's needs, or one close to the desired query, which can then be edited.

5.3.4 Network Measure Service, NMS

NMS computes structural measures of networks, such as degree, clustering coefficient, and betweenness centrality of vertices of a graph. NMS uses a software repository of MARS that includes paths to executables and command line arguments. New executables can be added to the repository. NMS also uses a Job Submission Service (JSS) that generates qsub files for the Torque/PBS scheduler (these executables run on an HPC cluster, but could be reconfigured to run on other hardware). This service is generic in that while we use it to compute structural properties of graphs, NMS can launch any executable that operates on a graph and then add the results to the Repository (with appropriate interactions with other services). This service uses serial and parallel standalone executables and different libraries, some of which are listed in Table 5.2. There can be multiple codes for the same measure, particularly for those that do not scale well for large graphs, such as betweenness centrality. In this case, one code may compute an exact solution and another may compute an approximate one.

Table 5.2. *Selected measures supported by the NMS for vertices, edges, and graphs.*

Network Measure	**Programming Language**	**Package**	**Target**
Betweenness centrality	Python	NetworkX	Edge
Degree product	Python	NetworkX	Edge
Degree	Python	NetworkX	Node
InDegree	Python	NetworkX	Node
OutDegree	Python	NetworkX	Node
Degree	SQL DBMS	Standalone	Node
InDegree	SQL	Standalone	Node
OutDegree	SQL	Standalone	Node
Betweenness centrality	Python	NetworkX	Node
Clustering coefficient	Python	NetworkX	Node
Clustering coefficient	Python	Standalone	Node
Node Clique number	Python	NetworkX	Node
Load centrality	Python	NetworkX	Node
Closeness centrality	Python	NetworkX	Node
Redundant: Connected component	Parallel C++	Galib	Node
K-shell	Serial C++	Standalone	Node
Diameter	Parallel C++	Galib	Network
Radius	Python	NetworkX	Network
Average degree	Parallel C++	Galib	Network
Connected components	Parallel C++	Galib	Network
Average clustering coefficient	Parallel C++	Galib	Network
Number of triangles	Parallel C++	Galib	Network

5.4 System Workflow Service

The *System* Workflow Service (SWS) orchestrates the execution of sequences of MARS services to accomplish tasks. Only system architects have access to these workflows; they can be created (currently by coding the sequencing manually), modified, or deleted. All users may use the workflows. Figure 5.3 presents two system workflows. The red "Error" ovals indicate that MARS detects an error in the inputs and hence cannot complete the workflow. It is important to note that each of these diagrams represents an algorithm, and that each of these algorithms invokes a sequence of lower-level algorithms that are encoded in the services.

5.4.1 Query Execution Workflow

The first workflow (Figure 5.3a) executes user-supplied queries. The NQSS may help a user construct a valid query (not shown in the figure). The Parsing Service parses the query against a grammar. If the query exists (determined by the Memoization Service) in the Repository and it is not a sampling multiset query, then the NQS returns the stored results immediately. Otherwise, a query validation workflow (described in Figure (b)) is launched and the workflow ends if the query is not valid. Otherwise, MARS determines whether the query requires measures that are not currently stored (using the Memoization Service), but for which an executable exists. If an executable does not exist, the workflow returns the result "invalid." If an executable does exist, the NMS computes the required measure, NStS stores the results in the Repository, and query execution resumes through the NQS.

5.4.2 Query Validation Workflow

The second workflow (Figure 5.3b) is used to validate new queries entered by users. This process is launched either as part of the query execution workflow, or individually, if a user wants to validate a query before execution. NQS is only used in the workflow. This example illustrates how a workflow can be executed by a single service, in contrast to Figure 5.3a where five services interoperate.

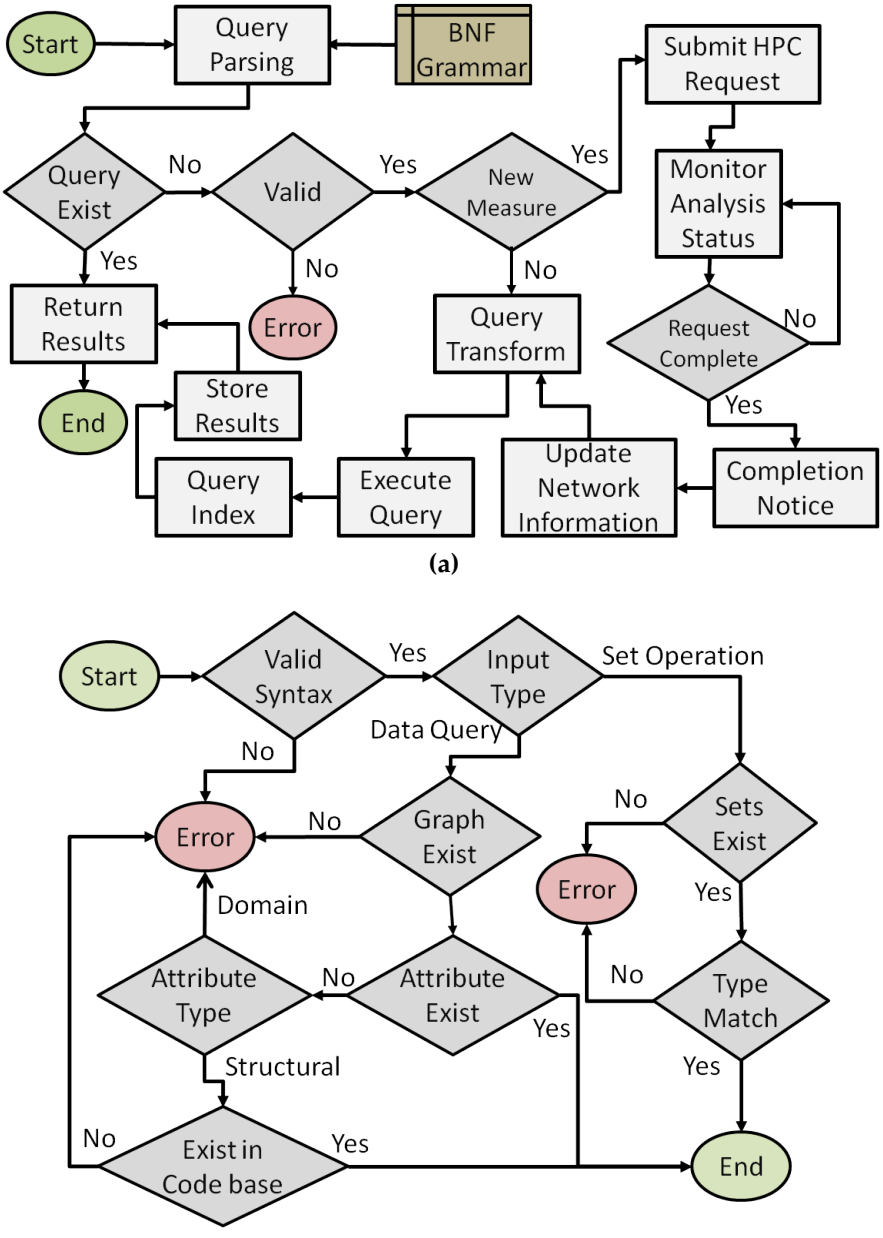


Figure 5.3. *Examples of system workflows for (a) query execution, which orchestrates five different services, the NQPS (in yellow), NQS (in pink), NMS (in blue), NStS (in grey) and NQSS (in green);(b) validating new queries, performed solely by the NQS (in pink), and can be part of query execution workflow in (a).*

5.5 Illustrative Case Study

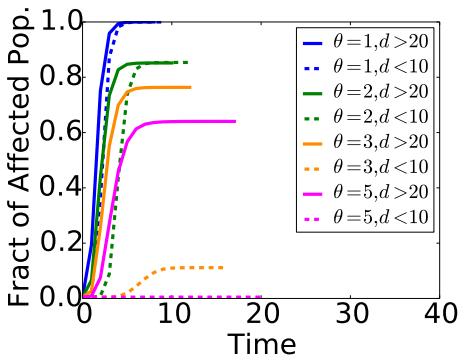


Figure 5.4. *Complex contagion simulations on a Facebook network of 63392 vertices, where the time histories of the cumulative fraction of nodes in state 1 (i.e., the affected state) are plotted against time, for different thresholds θ.*

We examine the spread of complex contagions in networks, where vertices can be in state 0 (resp., 1), meaning that a vertex does not (resp., does) possess a contagion. We investigate the contagion spread size (i.e., the fraction of vertices in state 1) as a function of time. See Figure 5.4; these results were generated using EDISON [7]. The threshold θ is the same for all vertices in one simulation, and is given in the legend. The threshold of a vertex v is the minimum number of neighbors that it requires to be in state 1 for v to transition from state 0 to 1. Once in state 1, a vertex remains in that state, which can represent many things, for example rumor-spreading. The 300 seed nodes for each of the 50 runs (the curves in the plot are averages over these runs) were determined randomly from all vertices with degree $d > 20$ (solid curves) or $d < 10$ (dashed). For a given d regime, the seed vertex sets are the same for all thresholds, for comparison. As vertex threshold θ increases, the final fraction of vertices in state 1 decreases, and the time to reach this maximum fraction increases. As threshold increases, the disparity in results between these cases for d, for a fixed θ, increases.

Several MARS services are used in this study: NMS, NStS, NQS, NQPS, NQSS, and

Memoization Service. Of particular note, to select seed vertices, the compact query feature and special features of the query grammar were used to specify all 50 sets of 300 seed nodes for each analysis, with a single query. The workflow of Figure 5.3a is used twice in each of the 8 EDISON analyses.

5.6 Performance Evaluation

Several experiments were completed to evaluate various aspects of MARS and directly addressed issues brought up in Sections 5.1, 5.3, and 5.4. We show five results here. In all of these studies, the networks are treated as undirected and range up to 400 million edges in size. We tested the NQS and NMS because these involve the most processing among the network services. The test system is a virtual machine comprised of an Intel Xeon X5670, 2.93 GHz processor running CentOS release 6.7. The services tested were implemented with Python 2.7.5.

Experiment 1: **Effect of using memoization on query time.** We executed the same query, to select all vertices of a graph, over selected networks stored in MARS: `select nodes from g`, where `g` is the name of the network. The returned data (in JSON format) from the first execution of a query is also stored as an entry in a table, indexed by a query ID, for fast retrieval for repeated querying. The Memoization Service enters new data and interrogates existing data. In the later case, the Memoization Service returns the JSON object immediately if a submitted query is matched with a previously executed query. Without memoization, each query gets executed in full. Figure 5.5 provides data for four networks and clearly that shows as the size of the network increases, the benefits of memoization increases. Also, if we use the memoization times as conservative estimates for checking whether requested results already exist, then this time is a small fraction of the computation time when the results

do not already exist; i.e., memoization overhead is low.

Experiment 2: **Time to calculate degree and degree product using stand-alone codes.** Computing graph structural parameters is one of the network services (NMS). First networks are downloaded from DB tables, the degree for each vertex and the product of the degrees of the incident vertices for each edge are computed. Once the computations are completed, the new data are uploaded into the DB and indexes are re-established. Figure 5.6 the times to compute selected vertex and edge parameters using stand-alone Python codes, and overhead activities shows for several networks. These results show that overhead times can rival the compute times, but that as more measures are computed with one download of the network, these costs are amortized more effectively. "Double storage" of networks, where they are housed in DB tables and on the hard-drive in ASCII format, is selectively done in MARS to reduce overheads, and these data explain why.

Experiment 3: **Time to calculate different structural parameters.** The time to compute different structural parameters can vary considerably. Here, we compare the times to compute the k-shell and degree of each vertex of selected networks using stand-alone executable codes for the structural computations, including the overheads of the previous experiment. Figure 5.7 demonstrates that execution times can be significant when the graph is large (e.g., almost 300 million edges), although this will depend on the graph parameter. Data such as these explain why a set of measures is computed for a network when it is initially uploaded into MARS (as part of a workflow), and is not made visible until these computations have finished: to hide latency.

Experiment 4: **Time to calculate vertex degree using the DBMS versus programming language.** The results of a previous experiment demonstrated that overhead times associated with downloading a graph and uploading vertex and edge prop-

erties can be significant. Consequently, if graph measures can be computed within the DB, then these overheads can be eliminated. To compare these approaches, the degree of each node is computed within the DB and by the approach of Experiment 2. Results in Figure 5.8 show that for larger networks, the time savings of performing the computations within the DB is significant. For example, for Pokec (1.6 million vertices, 30 million edges), the compute time within the DB is equivalent to 16% of the total time download the network and to compute measures with a stand-alone Python code. Hence, measures that are sufficiently simple should be computed within the DBMS. The DB-generated results use indexed tables. As with virtually all DB operations on large datasets, results are highly dependent on indexing. It also highlights the benefits of using parallel programs for structural computations (here the Python computations are serial).

Experiment 5: **Network services load testing.** Since MARS may simultaneously service multiple applications and multiple users, the expected number of concurrent service requests could be in the 100s or more. To investigate scalability, we used the open source load testing software SoapUI (https://www.soapui.org/) to generate different types of loads on MARS, based on a simulated number of users, each making one service request. Here, we evaluate a particular query on the netscience (1.5K vertices) and Google (875K vertices) networks: `select nodes from g where id < 50`, where `g` is the name of the network. Results for the tests are shown in Figure 5.9, where the number of user requests increases slowly over time from 1 to 2500. The average time to execute the query across the entire set of users whose size is indicated on the x-axis, is relatively stable at about 0.1 seconds, and thus indicates that the system is scalable to at least roughly 2500 users.

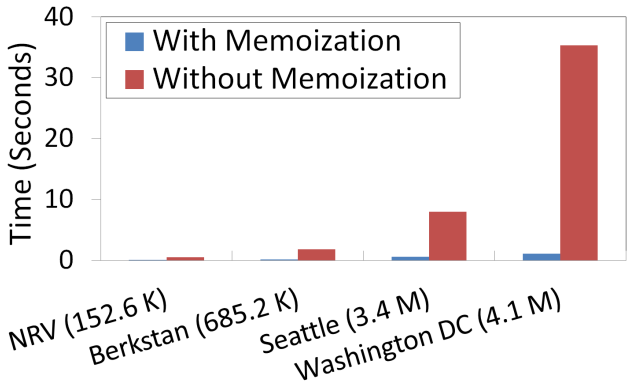


Figure 5.5. *Time to return query results for four networks when using the Memoization Service (and not) for networks up to 4.1 million vertices (400 M edges).*

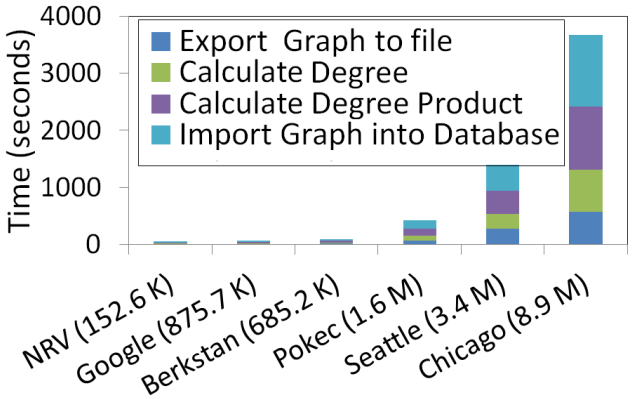


Figure 5.6. *Total time to compute degree per vertex (vertex property) and degree product (edge property), along with overhead times, for several networks.*

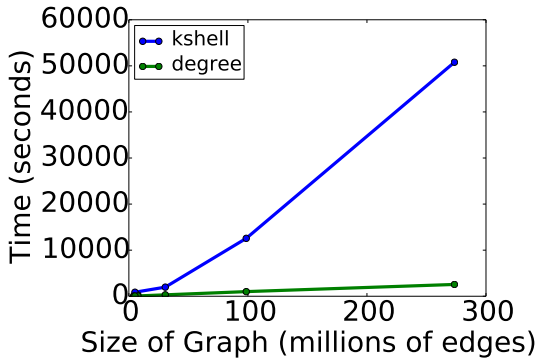


Figure 5.7. *Times to compute graph measures (degree and k-shell per vertex) for several networks, up to 9 million vertices and 273 million edges.*

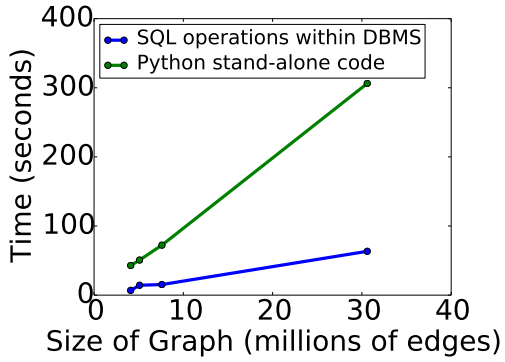


Figure 5.8. *Execution times that may be realized for network structural properties that can be computed within the DBMS for networks up to 1.6 million vertices and 30 million edges.*

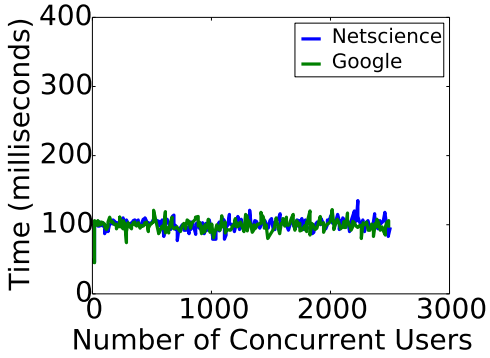


Figure 5.9. *Average execution time to return query results across different values of simultaneous query requests (x-axis). The average execution time is stable, indicating scalability. Netscience (1.5K) and Google (875K) vertices.*

5.7 MARS Implementation

MARS services is implemented using Python 2.7. External python modules and packages are used to deliver various features. Modules are listed in Table 5.3.

Table 5.3. *List of python packages used for MARS implementation.*

Package	**Purpose**	**Source**
SQLITE	Data Repository	https://sqlite.org/
Cherrypy	Multi-threaded python server	http://cherrypy.org/
Whoosh	Search Engine	https://pypi.python.org/pypi/Whoosh/
Pyparsing	Parser Generator	http://pyparsing.wikispaces.com/
Bottle Framework	Rest API	http://bottlepy.org/docs/dev/
MPipe	User-defined Workflows	http://vmlaker.github.io/mpipe/
matplotlib	Data Plotting	http://matplotlib.org/
pythonds	Internal Data Structures (Stack)	https://pypi.python.org/pypi/pythonds
requests	Send/Receive HTTP Requests	http://docs.python-requests.org/en/master/
NumPy	Data Analysis	http://www.numpy.org/
NetworkX	Network Structure Measures	https://networkx.github.io/

5.8 Limitations

Limitations of this work include: (i) difficulty in handling large query results which can reach hundreds of GB of data; and (ii) the usage of a file-based DB instead of a client/server DB which might raise issues with large numbers of concurrent requests. Areas for improvement include: (i) inclusion of a full-featured open source workflow system; (ii) inclusion of services for dynamics models; and (iii) evaluation of RDF DBMSs.

Chapter 6

Understanding Contagion Dynamics in Networked Populations Through Data Exploration and Visualization

6.1 Introduction

Modeling and simulation of contagion processes on networked populations are used to understand protests, social unrest, the spread of information, and virus and disease epidemics, among other phenomena. Network structure and attributes of vertices and edges are often useful in explaining contagion spreading processes. However, particularly for larger networks (e.g., those with hundreds of thousands or millions of vertices), reasoning about and making sense of contagion propagation results is difficult owing to the scale of these simulations. In this chapter, we describe a web application called NEMO for assisting an analyst in understanding contagion processes and in establishing causality. NEMO has several features to query, analyze and visualize networks, subnetworks, and their properties. In addition to explaining

NEMO's features, we provide a real case study using NEMO. We demonstrate how NEMO can be used to interactively explore networks to understand the reasons for the effectiveness of different interventions.

6.1.1 Contributions

A summary of our main contributions follows.

1. NEMO functionality for understanding contagion diffusion on large networked populations. NEMO is a web application with the following features: uploading attributed networks into the system; computing structural characteristics and measures of networks; browsing networks and their properties. NEMO can query networks to return sets of vertices and edges, and other information. It can visualize network attributes, computed data (e.g., measures), data returned from queries, and entire networks. Additionally, NEMO promotes sharing of data across users. We describe these features in more detail below. The system is designed for use by researchers, domain scientists, policy planners, and others with no computing expertise. We demonstrate its use on networks up to 4 million vertices in size.

2. Description of NEMO in the context of contagion dynamics. NEMO's purpose is sense-making of contagion diffusion on networks, through knowledge discovery about network properties using scientific computation and visual analytics. It is a distinct web application that can be run in isolation. Nonetheless, it integrates and exposes some of MARS services (described in Chapter 5) that can be profitably used in conjunction with other external systems. EDISON web application (described in Chapter 4) is one of these examples.

3. Case study. A case study is provided and demonstrates how NEMO can be used to reason about the efficacy of different interventions for impeding the spread of Ebola

in a 4 million vertex population of Liberia, Africa. In this case study, we demonstrate how exploration, analyses, and visualizations aid in the sense-making process.

Importantly, the case study illustrates that one cannot just compute graph structural parameters a priori, and then simply use the resulting static plots to explain dynamics. This is because contagion processes on networks change with the contagion model, model properties, initial simulation conditions, and interventions, among other factors. There is no way to a priori list these myriad combinations of simulation conditions and pre-compute all required network characteristics. Consequently, it is vital to have an *interactive* system like NEMO that enables *exploration* of network—and subnetwork—properties for knowledge discovery.

Chapter Organization. NEMO is overviewed in Section 6.2. Illustrative case study is provided in Section 6.3.

6.2 NEMO Overview

6.2.1 Summary of User Features

Figure 6.1 provides features available to users through NEMO. A user can upload networks into the system, and specify structural parameters (e.g., number of connected components) and distributions (e.g., degree distribution) to be computed. Users can browse networks, including their attributes. Attributes may be domain-based, such as the ages of network vertices that represent people and duration of contact on an edge between two vertices, or structural parameters such as k-cores or betweenness centrality. Users can perform queries on these networks, returning sets of vertices or edges which can be plotted. Other types of plots include networks themselves or data associated with them. Example plots for each of these classes are given below in

this section or in the case study of Section 6.3. Data can be shared among users. For example, a network uploaded by one user can be used by another; e.g., to generate plots of properties. A user can couple these activities; e.g., browse graphs looking for a network with at least some specified number of vertices and with a maximum degree of at least a certain value.

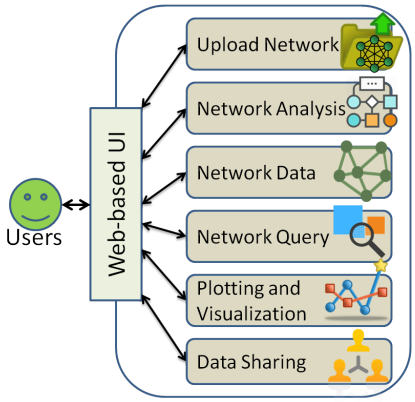


Figure 6.1. *User functionality through NEMO.*

Some of the functionality required to realize these features is provided by MARS [8], such as the Network Analysis and Query features. NEMO uses the MARS application programming interface (API) to complete queries and user-defined workflows. Other functionality is provided within NEMO, but here we focus on system use and visualization.

6.2.2 Additional Details

Figure 6.2 contains a screen capture of a NEMO UI screen for performing queries. Queries use an SQL-like grammar (provided by MARS services), with some additional features for the queries to support contagion dynamics on networks. Results from the queries may be returned and downloaded in different formats, and the results from the queries may contain vertex or edge properties, or just vertex/edge IDs.

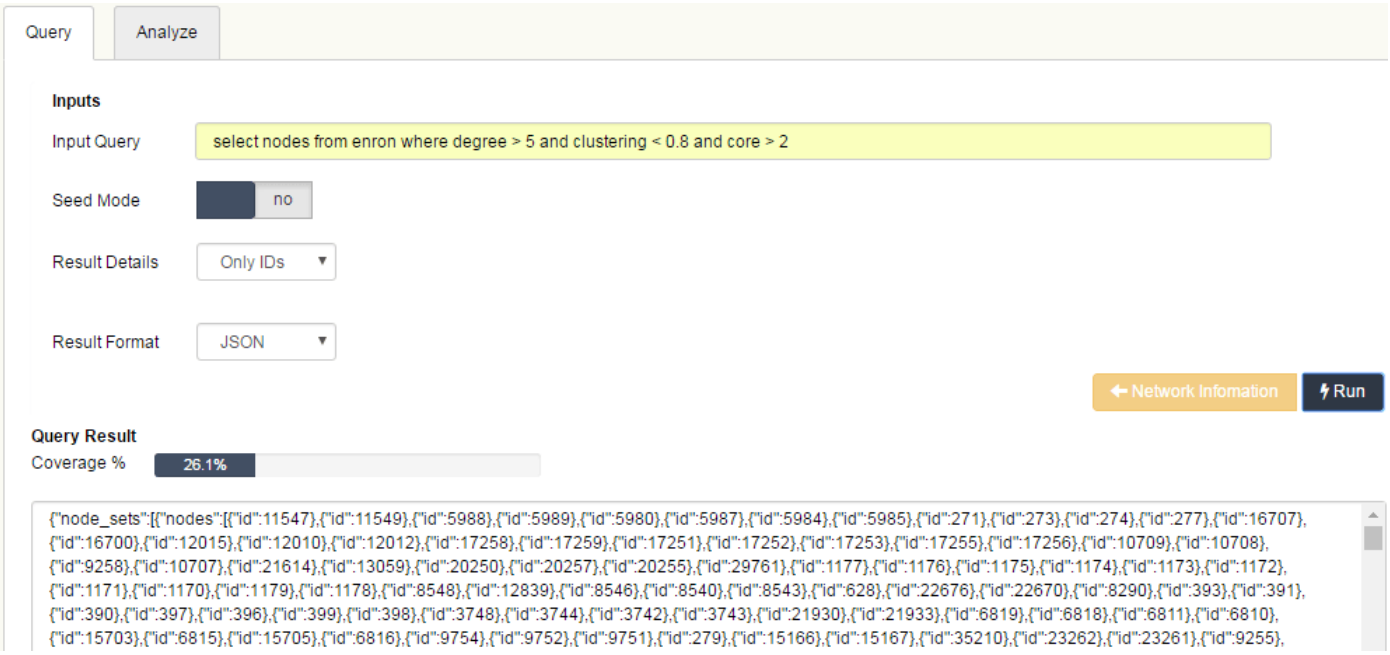


Figure 6.2. *NEMO UI screen for performing queries. The specified query returns all vertices that have degree* > 5*, clustering coefficient* < 0.8*, and k-core of 3 or more. Return data can be vertex IDs only, or IDs with all properties associated with each vertex. Similarly for edge-based queries. This is useful for queries that produce large return sets; if all that is needed are vertex (resp. edge) IDs, then much storage can be saved. Result formats include JSON, XML, and CSV. The particular query returned 26.1% of network vertices.*

This feature is useful for memory management when dealing with large networks. A feature that builds on the query capability is that results from the queries can be plotted. For example, results from the queries that return counts can be plotted. We provide examples in Section 6.3 below.

Plots may be generated by specifying plot type and data; e.g., scatter, bar, or pie charts. For example, fractions of vertices in the five largest communities of a Wikivote network [140], computed using the Louvain modularity-based community detection method [46], are displayed in Figure 6.3. Axis labels and types (logarithmic and linear), axis numeric ranges and tick label increments, and font sizes for axis and tick labels may all be customized per plot (e.g., for bar and scatter plots). Similarly, data may be plotted as symbols only, lines only, or both. Line colors and widths and sym-

bol type, size, and color are user-specified. We use the Matplotlib library [115] for plot generation. Plots generated with NEMO are publication quality. Many of the plots in this chapter were generated directly with NEMO. Plots may be downloaded in PDF and PNG. Moreover, the data in a plot can also be downloaded for further processing.

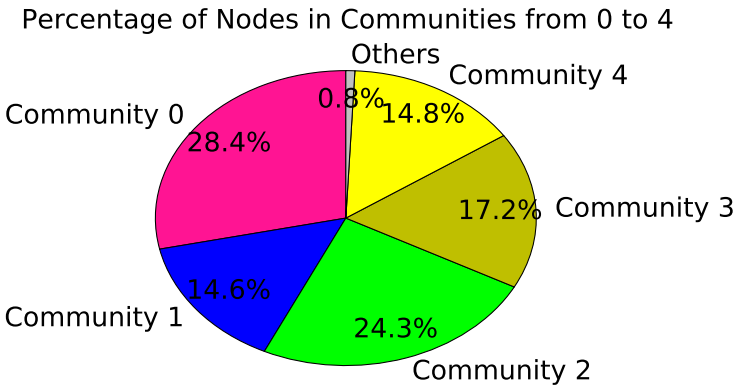


Figure 6.3. *Fraction of vertices in each of the five largest communities for a Wikivote network with 7115 vertices.*

The visualization in Figure 6.4 is generated with NEMO, using Gephi and a thin layer consisting of a Javascript GEXF Viewer, available for download[1]. Network visualization can be accessed through NEMO or directly through a supported separate URL. Visualizations can also be embedded into external websites. The visualization component resides on a separate machine from NEMO, which allows re-configuration or upgrading the visualization layer without affecting NEMO.

These plots link back to Figure 6.1. The *upload network* and *network analysis* features in NEMO enable users to load *attributed* or *labeled* networks into the system and to compute measures on networks (e.g., k-shell and degree distributions, communities, and numbers and sizes of connected components). Then, *network query* enables a user to parse, combine, and even operate on the data. All of these data are then candidates

[1] https://github.com/raphv/gexf-js

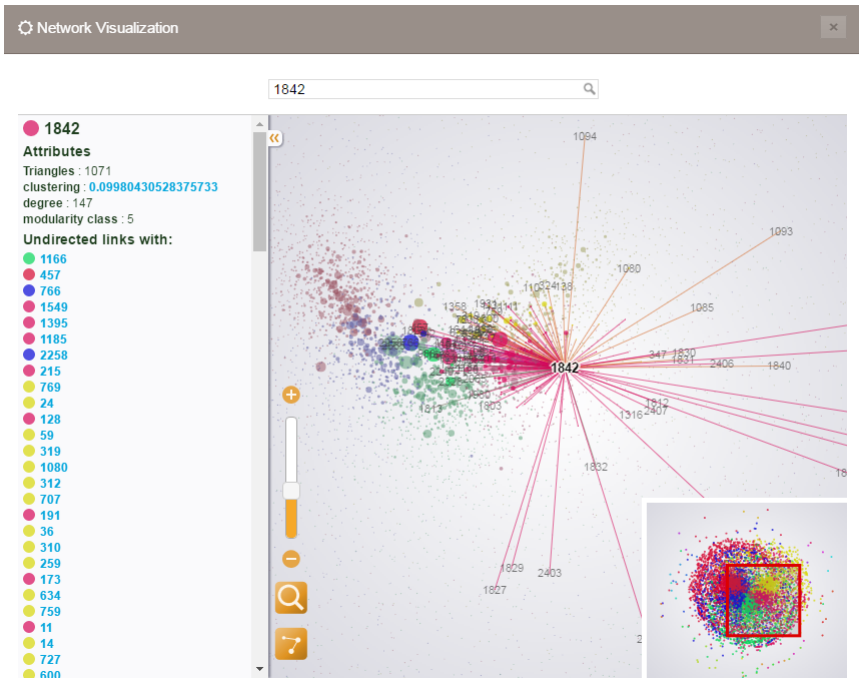


Figure 6.4. *Illustrative graphics generated through NEMO. Visualization of Wikivote network with 7115 nodes and 100762 edges. Vertex colors represent communities, while vertex sizes are proportional to vertex degrees. The legend on the left shows some of the structural attributes of selected vertex (Id = 1842), along with a list of its direct neighbors.*

for plotting.

6.3 Illustrative Case Study

We present a case study that evaluates the effectiveness of interventions in modeling an Ebola outbreak in Liberia, Africa, which is similar to the one that peaked in 2014. This study is a continuation of case study 3 that is described in Chapter 4, which shows that some interventions are more effective than others. See Figure 4.10a.

NEMO can interrogate networks and visualize data through the NEMO UI; these features help us to reason about the contagion dynamics. Figure 6.5 conveys the operations of NEMO for reasoning about the simulation results. The analyses described here represent one session with NEMO; i.e., a user logs into the system and completes all analyses in one sitting (although these operations may be broken up among multiple sessions).

First, it is reasonable to suspect that there are simply more people in the network of the 31-40 age range, so vaccinating these people produces the greatest vaccine coverage. To test this hypothesis, the user constructs a workflow within NEMO using point-and-click operations. The workflow counts the number of social network vertices that are in each age range, and constructs and displays the bar chart shown by workflow 1 (the orange circle with "1" inside), which contains the green bars. From this plot, we see that approximately 0.5 million people are in the 31-40 age range, but that other groups have far more people.

These data do not provide an explanation, so the analyst constructs a second workflow, denoted by the orange circle with the "2" inside, in Figure 6.5. This workflow generates the accompanying plot and is motivated by the idea that people in the 31-40 age range may have more total interactions than those in other age ranges. But the plot of sum-of-degrees by age bin shows that, again, this is not the case. This age group has slightly more than 10 million interactions, but there are two other groups with more contacts.

Workflow 3 is used to determine whether the average number of interactions per person in each age range explains the effects of interventions. In this case, the 31-40 and 41-50 age ranges have the greatest average degrees, but only slightly more than the bins 21-30 and 51-60. These four bins represent the most effective age ranges for interventions in Figure 4.10a. While informative, the average degrees among these four bins do not explain the differences in the efficacies of the interventions.

Workflows 4 and 5 are used to compute k-shell distributions, and for each k-shell, the number of vertices in the shell for the different age bins is computed. A k-core is a subgraph of a graph in which all vertices have a degree that is at least k. A k-shell is the set of vertices in the k-core subgraph, and not in the $(k+1)$-core subgraph. For each age bin, the more vertices that are in greater shells, the more well-connected those vertices are to other high-degree vertices. Workflow 4 produces the counts of network vertices in each k-shell, by age bin, for the five age groups that are least effective in stopping the outbreak when vaccinated, and generates the plot shown. Workflow 5 generates analogous data for the four age groups that are most effective to vaccinate and the plot generated in this workflow is also shown. From this latter plot, we see that for some of the greatest shells, around $k = 30$ (see the red arrow), the ordering by age bin for the number of vertices in shells 30 through 35, exactly matches the ranking of the efficacy of vaccination. The largest shell in the Liberia network is 37, so these shells are large. This suggests that the vertices in the social network with a high degree, that are the most well-connected to other high degree vertices, are most effective in reducing the Ebola spread.

After workflow 5, the analyst terminates the session. Note that for all plots, the user can customize the plotted ranges on the x- and y-axes, the axis labels and all font sizes, and the colors and types of lines for each data line. The goal is to enable a user to generate publication-quality graphics while interactively exploring the data.

This case study does not prove that vaccinating high k-shell vertices are most effective in thwarting Ebola outbreaks. Typically, the best option the analyst can do is to use inductive reasoning: given the evidence (i.e., simulation results), identify likely causes that produce the result. Additional simulations, perhaps on other networks, could be used to test this hypothesis.

This study also illustrates that one must look in detail at data. High-level graphics, such as a network visualization with vertices colored by community, does not yield useful information, although identifying the vertices in communities that are most connected to other communities could yield useful information. Social networks with high-degree hub nodes often produce communities of very large size, which are not helpful in understanding how contagion is transmitted. Other possibilities are computing the durations of contact for the vertices of different age bins (not shown), since the duration of contact affects the probability of disease transmission.

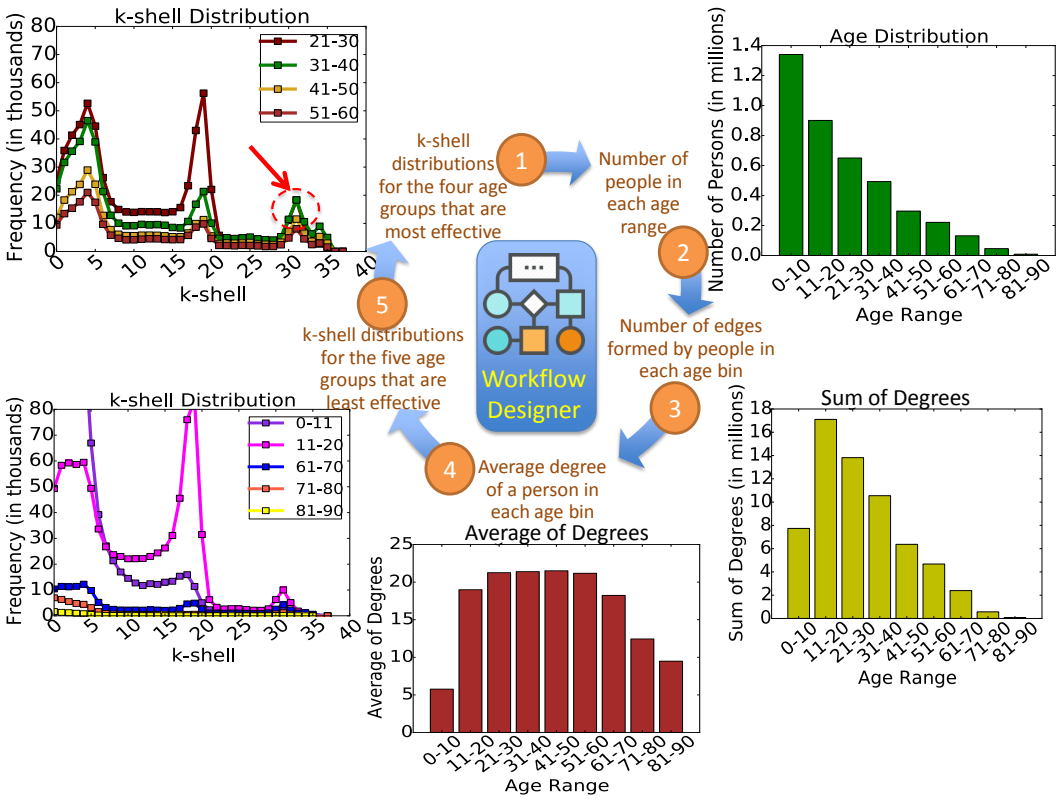


Figure 6.5. *Iterative and interactive use of NEMO to generate different plots on-the-fly through the web console, in order to understand the effects of interventions on simulated Ebola outbreaks in Liberia, Africa. Given the simulation results in Figure 4.10a, the goal is to understand the reasons for the effectiveness of different interventions. The data here were generated during one user session with NEMO. Each plot is the result of a different workflow that is specified within NEMO by a user. Between workflows, the human-in-the-loop evaluates the results and decides on the next analysis (workflow) to be completed, if any. Here, the results of five analyses (workflows labeled 1 to 5) are provided. The utility of these results is described in the text.*

6.4 Limitations

There are several limitations of this work. These include: (i) the lack of control flows (e.g. branching and looping) in user-defined workflows; and (ii) the inability to visualize large networks entirely due to memory limitations. Possible enhancements are: (i) integration of a full-featured open source workflow system; and (ii) the incorporation of multiple types of data in knowledge discovery (e.g., simulation data, data from other simulation runs and other networks), beyond data from a single network.

Chapter 7

Agent-Based Modeling and Simulation of Depression and Its Impact on Students' Success and Academic Retention

7.1 Overview

In the U.S., major depressive disorder affects approximately 14.8 million American adults. Furthermore, depression can lead to a several other illnesses and disabilities. Economic burden of depression is estimated to be $53 billion annually in the U.S. alone. Depression can reach such high levels that it can lead to suicide, which is the third leading cause of death among the U.S. college-aged population.

Studies show a direct relation between mental health and academic success. In particular, depression is a significant predictor of a lower GPA and an increased dropout

rate. A 15 point increase on the depression scale correlates with a 0.17 drop in GPA and corresponds to a 4.7 percent increase in probability of dropping out [81]. High dropout rates also adversely impact both universities and society.

In this chapter, we construct and exercise an agent-based model (ABM) of the evolution of depression among a population of roughly 19,000 college students [6]. This model includes within-agent interactions among depression symptoms and agent-to-agent interactions defined by a college student social network. We conduct simulation studies to identify and model parameters and initial conditions that most influence population outcomes. Connectivity among within-agent symptoms is demonstrated to have a large effect on population levels of depression.

7.2 Introduction

7.2.1 Background

Dropout rates of engineering students are high in the United States [34, 136]. Increased dropout rates have negative impacts on students, institutions and society. From a student's perspective, quitting school can lower self-confidence and self-esteem [36]. From an institutional perspective, student dropouts represent a loss of talented students and indicate an institution's lack of attention to the needs of students [134]. Dropout rates also affect society. The national cost of these dropouts amounted to $4.5 billion in lost earnings and taxes to state and federal governments in 2012 [186]. Factors contributing to students' dropping out include preparation, ability, motivation, engagement within the institution, college grade point average, financial aid, age, ethnicity and socioeconomic status [9, 135, 196, 223].

The side effects resulting from social interactions among students including friends,

classmates, and roommates is another important factor in withdrawing from school [54, 55, 145, 204]. Depression is contagious [51, 70, 106, 113, 126, 161, 195, 207], meaning that it can spread from person to person through different types of social contacts, such as face-to-face interactions and online social media. Studies show a direct relation between mental health and academic success. Students report depression and anxiety among top impediments to academic performance [18]. Sixty four (64) percent of young adults who are no longer in college cite a mental health-related reason for not attending [102]. In this work, we use agent-based modeling (ABM) to simulate the evolution of depression within a synthetic social contact network of undergraduate students at a university.

7.2.2 Motivation for Agent-based Modeling of Depression

ABM of depression is important for several reasons. First, there has been a large amount of data gathered and analyzed since the 1970s regarding factors that affect academic retention and attainment. These data can be used to develop and inform social behavior models that can be used in simulations. Second, simulations resolve behaviors in time. This is often critical for understanding causality and how local agent behaviors give rise to population-level outcomes. This is fundamentally different from performing statistical analyses using final outcomes [84]. Third, simulations on appropriately labeled agents that compromise a population (e.g., in the form of social networks) can produce disaggregated results. Validated models can also be used in other settings (e.g. for different academic institutions), and can be exercised to explore counterfactuals. Finally, results from ABMs can inform experimental studies, including surveys and human-subjects testing. This is because ABMs can be exercised to identify which variables have the most impact on outcomes. These are the variables that are most important to characterize through experiments. Hence, there is a

feedback loop between experiments and modeling.

7.2.3 Contributions

A summary of our major contributions follows.

1. A synthetic population of 19,000 college undergraduates. A social network of the state of Virginia was generated using the procedure in [27] and used as the starting point for our work. The social network contains synthetic individuals whose traits match in distribution the attributes of the actual population. From this network, we extracted college students (those agents in the age range 18-22 years that have college activities and are located in the vicinity of a major university). Edges in the original social network are retained if the incident vertices are both college students. This produces a social network of 18,866 students (agents) and 119,139 pairwise student daily interactions.

2. Agent-based model of depression. We extend a model of within-host depression evolution [205] to include the effects of social interactions. Consequently, our model accounts for internal, environmental, and social factors of the evolution of depression, which is consistent with many research studies [70, 113, 161]. In particular, each agent has an internal network with 14 vertices that represent depressive symptoms. These symptoms can influence each other, and we take these interactions from the literature. Each agent is connected to peer students in the social network with whom the student comes in contact. Influence is transmitted through the edges. A student becomes depressed if a sufficient number of symptoms becomes activated through within-host and external interactions. A student may transition back and forth between depressive and healthy states.

3. Simulation of depression in a college population. We code the agent-based model

in a simulation system (InterSim) and exercise it using various inputs. InterSim is described in [128]. We show that the number of depressed students changes with the strength of influence in the symptom interactions and in the social network. We explore the effect of initial conditions, and illustrate the interesting result that within-agent symptom connectivity changes the magnitude of the steady state level of depression within the population.

7.3 Data and Methodology

7.3.1 College Social Network

We use a realistic college population over which we study depression dynamics and peer influence. We model the undergraduate student body of a large university. A modeling process [27] was used to construct this population, which creates anonymous students and endows them with traits such as age, gender, and sets of activities that result in daily face-to-face interactions with other students. The result of this process is a college social network, where nodes/agents represent students and undirected edges represent interactions between students.

To produce this network, we start with the social contact network of the state of Virginia. We then extract from this network, people with the ages between 18 and 22 (inclusive) years that have at least one edge (i.e., interaction) with another college-age student, and who are geographically located in the vicinity of a particular public university. These agents and their interactions in a normative day form the social network, whose traits are given in Table 7.1. An agent has roughly 12 interactions a day with other students. There are over 200,000 "friend of friend" relationships (i.e., triangles) in the network. The diameter of the network is quite large compared to many social networks. Although there are 105 connected components, the network has a

giant component that contains 98.7% of all nodes (agents).

Table 7.1. *College network structural characteristics.*

Network Property	**Description**	**Value**
Number of Nodes	Total number of students	18866
Number of Edges	Total number of peer-to-peer interactions	119139
Number of Triangles	Number of student groups of size three and form a cycle	202318
Average Degree	Average number of edges connected to a node	12.63
Diameter	Longest of all the calculated shortest paths in the network	16
Average Path Length	Sum of shortest paths between all pairs of nodes divided by the total number of pairs	4.785
Density	Ratio of the number of edges to the number of possible edges	0.001
Modularity	Fraction of edges that fall within a group, minus the expected number of edges within group	0.484
Number of Communities	Number of node groups in the network	151
Average Clustering Coefficient	Measure of the degree to which the nodes tend to cluster together	0.348
Number of Connected Components	Number of node groups that are mutually reachable by undirected edges	105
Size of Giant Component	The fraction of nodes in the largest connected component	0.987

7.3.2 Contagion (Behavioral) Model

We propose a model that quantifies the diffusion of activated depression symptoms among students. The model accounts for the major factors that affect the dynamics of depression. We study two types of dynamics: (i) internal dynamics within a single agent and (ii) external dynamics between agents. Internal dynamics describe how depression evolves within a student as a result of symptom interactions. Symptoms

include depressed mood, loss of interest, weight loss, weight gain, decreased appetite, increased appetite, insomnia, hypersomnia, psychomotor agitation, psychomotor retardation, fatigue, worthlessness or guilt, concentration problems, and suicidal thoughts [205]. It is based on the hypothesis that symptoms of mental disorders have direct causal relations with one another and this is called the causal network perspective [49, 68, 184, 205]. External dynamics focus on students' peer interactions with their roommates, classmates, or friends. The social contact network edges are the external interactions.

Our modeling approach follows graph dynamical systems (GDSs) [155]. GDS is described in Section 1.4. Figure 7.1 shows an illustrative example of a six-agent network, with an internal view of two agents 4 and 6. Within-agent edges represent the causal relations between symptoms of that agent, while undirected edges across agents are the peer interactions. In the following discussion, we will refer to the internal symptom network as G^1, and the across agent network as G^2.

In Figure 1, there is a single edge between agents 4 and 6. This single edge represents the multiple edges between symptoms of agents 4 and 6. We do not show all of these to reduce clutter. In general, each symptom of agent 4 can be connected to any number of symptoms of agent 6. In this work we confined ourselves to between-agent edges that connect the same symptoms, so there are 14 edges between the symptoms of agents 4 and 6.

In our model, the network is described as $G(V, E)$ where G is a composite graph of G^1 and G^2. $G^1(V^1, E^1)$ is the undirected graph of depressive symptoms within each student, and is fixed for all students. It represents the causal network of within-host symptom interactions. The set V^1 is the vertex set of symptoms, $n^1 = |V^1| = 14$. Let $v^1_{ki} \in V^1$ be the stress-generating symptom k for agent i. The edge set E^1 represents the direct causal relations between two symptoms where $e^1_{kl,i} \in E^1$ is the undirected

edge between symptoms k and l for agent i and m^1=$|E^1| = 17$ (see Figure 7.1). The state x^1_{ki} of v^1_{ki} is either 0 if symptom k is not activated, or 1 if it is. The state set $K^1 = \{0, 1\}$.

Graph $G^2(V^2, E^2)$ is the network describing the student contacts in the population, where V^2 is the vertex set of (human) agents and E^2 is the edge set of their daily interactions, $n^2 = |V^2|$ and m^2=$|E^2|$. For this problem, n^2 and m^2 are given in Table 7.1. Each element $v^2_i \in V^2$ is a student i that can be considered a supernode, that contains a graph G^1 describing internal dynamics. Each edge in E^2 represents a set of connections between pairs of symptoms in neighboring agents. Specifically, $e_{k_1 i, k_2 j}$ represents an undirected edge between symptom k_1 of agent i and symptom k_2 of agent j. To simplify simulations and interpret results throughout this study, we take $k_1 = k_2$. That is, only the same symptoms are connected between two neighboring agents in the social network. This choice conforms to research findings [67, 111, 125], which show that homophily in depression symptoms may stem from peer influence.

The set K^2 of agent vertex states is defined as $K^2 = \{0, 1\}$. Student i can be in one of two states, healthy $x^2_i = 0$ or depressed $x^2_i = 1$. A student is considered depressed if there are eight or more developed/activated symptoms. Thus, an agent can transition back and forth between depression and no depression depending on its number of activated symptoms. The probability for symptom v^1_{ki} to become activated is represented as p^1_{ki}.

Two types of vertex functions are presented in the model: The set F^1 of vertex functions f^1_{ki} determines the state of each stressing symptom v^1_{ki} for student v^2_i at time step t. The set F^2 of vertex functions f^2_i determines the state of each student v^2_i at each time step t. (See Table 7.2.)

The specification W of the order in which every f^1_{ki} and f^2_i is executed, is as follows. At each time t, each agent i computes the state x^1_{ki} for symptom k, using f^1_{ki}. Then

Table 7.2. *List of model variables and user input parameters. rnd denotes a random number in* $[0, 1]$.

Name	Definition	Type	Range	Equation
p^1_{ki}	Activation probability for symptom v^1_{ki} in G^1.	$\mathbb{R}$	$[0, 1]$	$\frac{1}{1+e^{a^1_{ki}(b^1_{ki}-A^1_{ki})}}$
A^1_{ki}	Total amount of stress on symptom v^1_{ki} in G^1.	$\mathbb{R}$	$[-\infty, +\infty]$	$z^1_{ki} + c^2_i + q^1_{ki} + r^2_{ki}$
r^2_{ki}	Student's peer influence in G^2. $N^2(v^2_i)$ is the set of peers (distant-1 neighbors) for student v^2_i. w^2_{ij} is edge weight between student v^2_i and student v^2_j whose state is x^2_j.	$\mathbb{R}$	$[0, +\infty]$	$\sum_{v^2_j \in N^2(v^2_i)} (w^2_{ij} x^1_{kj})$
q^1_{ki}	Symptom's distant-1 neighbors influence in G^1. $N^1(v^1_{ki})$ is the set of neighbors for symptom v^1_{ki}. w^1_{kici} is edge weight between symptoms v^1_{ki} and v^1_{ci} whose state is x^1_{ci}.	$\mathbb{R}$	$[0, +\infty]$	$\sum_{v^1_{ci} \in N^1(v^1_{ki})} (w^1_{kici} x^1_{ci})$
τ^2_i	Number of activated symptoms for student v^2_i.	$\mathbb{N}$	$[0, 14]$	$\sum_{k=0}^{n^1-1} f^1_{ki}$
c^2_i	Amount of stress on symptoms network in student v^2_i.	$\mathbb{R}$	$[-8, 8]$	Model Parameter
z^1_{ki}	Individual stress level of symptom v^1_{ki}.	$\mathbb{N}$	$[-5, 5]$	Model Parameter
a^1_{ki}	Symptom-specific parameter controls steepness of p^1_{ki}.	$\mathbb{R}$	$[-\infty, +\infty]$	Model Parameter
b^1_{ki}	Symptom-specific parameter for the threshold of symptom v^1_{ki}.	$\mathbb{R}$	$[-\infty, +\infty]$	Model Parameter
w^1_{kici}	Edge weight between symptom v^1_{ki} and symptom v^1_{ci} within same agent i in G^1.	$\mathbb{R}$	$[0, 1]$	Model Parameter
w^2_{ij}	Edge weight between student v^2_i and student v^2_j in G^2 $(i \neq j)$. This is the weight between all corresponding symptoms of two agents.	$\mathbb{R}$	$[0, 1]$	Model Parameter
f^1_{ki}	Symptom k of agent i state transition function in G^1.	$\mathbb{N}$	$\{0, 1\}$	$f^1_{ki} = \begin{cases} 1 & \text{if } p^1_{ki} - rnd > 0 \\ 0 & \text{otherwise} \end{cases}$
f^2_i	Student's i state transition function in G^2.	$\mathbb{N}$	$\{0, 1\}$	$f^2_i = \begin{cases} 1 & \text{if } \tau^2_i > \tau_{crit} \\ 0 & \text{otherwise} \end{cases}$

each agent i executes f_i^2 to determine the state x_i^2 at time t. An agent i is depressed if $x_i^2 = 1$ and not depressed if $x_i^2 = 0$. As we will see below, because of the forms of f_{ki}^1 and f_i^2, each agent can perform these computations in parallel. The critical number τ_{crit} of activated symptoms causing a person to change state is $\tau_{crit} = \lfloor n^1/2 \rfloor$. That is, at least one-half of a person's symptoms must be activated to cause a person to transition to, or remain in, state 1.

Definitions of each variable and constant are provided in Table 7.2. The values for a_{ki}^1 and b_{ki}^1 are based on the VATSPUD data used in [123, 160]. The variable A_{ki}^1 is the total amount of stress on symptom v_{ki}^1. It consists of (i) the individual stress level z_{ki}^1, (ii) the amount of external activation c_i^2 which is the amount of stress on the symptom network for student v_i^2, (iii) the influence of the activation of neighbors of symptom v_{ki}^1, which is denoted by q_{ki}^1, and (iv) the student's peer influence on symptom v_{ki}^1 denoted is by r_{ki}^2. The variable q_{ki}^1 depends on whether or not the symptom's neighbors are activated and on the strength of the connection between the activated neighbor and symptom v_{ki}^1. The variable r_{ki}^2 quantifies peer influence in G^2 and depends on whether or not the symptom v_{ki} of student i's direct contacts (e.g., roommates, classmates or friends) are activated and on the strength of the relation between the depressed student and his or her peer v_j^2.

We relate particular equations in Table 7.2 to the GDS model and networks of Figure 7.1. First, the equation for q_{ki}^1 captures the interactions among symptoms of the within-agent network G^1. For symptom k of agent i, $N^1(v_{ki}^1)$ denotes the neighbors v_{ci}^1 for symptoms c. The states x_{ci}^1 of these distance-1 neighboring symptoms, along with the edge weights w_{kici}^1 between symptoms k and c, contribute to the next state of symptom v_{ki}^1 through q_{ki}^1.

Second, r_{ki}^2 captures social influence on agent i in G^2. For each of the neighbors v_j of v_i in G^2, the symptom v_{kj} influences the symptom v_{ki} of v_i. Here, we take the

strength w^2_{ij} of the interaction between corresponding symptoms of two agents i and j as the same for all symptoms k. We further assume that only corresponding symptoms interact. Neither of these assumptions is limiting and can be relaxed. In this case, then, the general interaction term r^2_{ki} would be $r^2_{ki} = \sum_{v^2_j \in N^2(v^2_i)} \left(\sum_{k'=0}^{n^1-1} (w^2_{k'jki} x^1_{k'j}) \right)$. In this case, $w^2_{k'jki}$ is the weight of an edge in G^2, the *social* network, between symptom k' in v^2_j and symptom k in v^2_i. If no such edge exists, the weight is zero. The outer sum in the last equation is over all neighbors of v^2_i.

Third, the state of depression of an agent i is computed in a two-step process. At each time t, f^1_{ki} is computed and symptom k of agent i is activated if $f^1_{ki} = 1$. Then, f^2_i is evaluated by determining whether τ^2_i—the number of activated symptoms—is more than one-half of symptoms for agent i. If so, then the agent i is depressed. Note that in these equations, an agent j may not be depressed, but if it has activated symptoms, then it can contribute to the depression of its neighbors. Note also that the depressive state x^2_i of an agent i is not directly used in the depression evaluation of its neighbors.

Model Behavior

The model has many parameters. Here we focus on the effects of stress level z^1_{ki}, edge weight w^2_{ij}, and degree of agents in the social network on the activation probability p^1_{ki} of symptoms, since activated symptoms govern depression of agents. We fix all other factors. Figures 7.2a shows the effect of varying the symptom individual stress z^1_{ki} from -5 up to 5 on symptom activation probability for all 14 symptoms. Other factors c^2_i, z^1_{ki} and w^2 are set to 0, 0 and 0.5 respectively. We use w^2 as the value of w^2_{ij} for all i and j, as seen in Table 7.2. Symptom 0 contributes the most to the increase in activation probability. Differences in results across symptoms are large for z^1_{ki} between 0 and 3.

In Figure 7.2b, we investigate the effect of symptom 0 on the activation probability

for each agent as the numbers of depressed friends increases and the edge weights w^2 changes. Other factors c_i^2 and z_{ki}^1 are set to 0. As the number of depressed friends increases, the internal symptom's activation probability increases. This effect is more pronounced as w^2 increases. This leads to an increase in the number of active symptoms and hence to an increase in the agent's vulnerability to depression.

We repeated the experiment but now with all of the symptoms. The weights w^2 were fixed to 0.05 and factors c_i^2, z_{ki}^1 are set to 0. The results are shown in Figure 7.2c. We find the same effect of depressed friends on all symptoms, but the magnitude in activation probability changes across symptoms. These results are expected as symptom 0 is highly connected in the symptom network of Figure 7.1.

Simulation Description

Simulation starts at time $t = 0$. At each time step, a symptom v_{ki}^1 within student v_i^2 has a state x_{ki}^1, and its function f_{ki}^1 computes the symptom state change from state $x_{ki}^1(t)$ at time t to state $x_{ki}^1(t+1)$ at time $t+1$. Similarly at each time step, a student v_i^2 has a state x_i^2, and its function f_i^2 describes how the student changes his or her depressive state from state $x_i^2(t)$ at time t to state $x_i^2(t+1)$ at time $t+1$ based on the number of activated symptoms at time $t+1$. Healthy students are susceptible to depression at any time based on their current state and their peer influence. Students may also develop some symptoms based on external sources, such as experiencing or watching stressful events.

The main simulation steps are described in Algorithm 2, which is the entry point for the simulation. It describes how students' states change over time based on the current number of activated symptoms. The process of identifying and counting active symptoms is described in Algorithms 3 and 4. We take $simDuration = 12$ (months) and $depressionThreshold = 8$.

Algorithm 2: depressionDevelopment

Input: Simulation parameters, including simDuration and depressionThreshold
Output: Students' depressionStatus
simTime = 0; // Start simulation at time 0
while *simTime* < *simDuration* **do**
// Continue while the current time is less than the desired duration
  foreach $v_i^2 \in V^2$ **do**
// Loop over the entire student population
    $n \longleftarrow countActiveSymptoms(i, k, a_{ki}^1, b_{ki}^1, z_{ki}^1, c_i^2, v_{ki}^1, N^1(v_{ki}^1), N^2(v_i^2), v_i^2)$;
    // Calculate the number of active symptoms within each student
    if $n >=$ *depressionThreshold* **then**
      $depressionStatus \longleftarrow 1$; // Student is currently depressed
    else
      $depressionStatus \longleftarrow 0$; // Student is currently not depressed
  $simTime \longleftarrow simTime + 1$

Algorithm 3: countActiveSymptoms

Input: $i, k, a_{ki}^1, b_{ki}^1, z_{ki}^1, c_i^2, v_{ki}^1, N^1(v_{ki}^1), N^2(v_i^2), v_i^2$
Output: count
count = 0; // Initialize number of active symptoms to 0
foreach $v_{ki}^1 \in V^1$ **do**
// Loop over the 14 symptoms within the same student
  $p_{ki}^1 \longleftarrow activationProbability(i, k, a_{ki}^1, b_{ki}^1, z_{ki}^1, c_i^2, v_{ki}^1, N^1(v_{ki}^1), N^2(v_i^2), v_i^2)$; // Calculate the activation probability
  $rnd \longleftarrow uniformRandomNumber()$
  if $p_{ki}^1 - rnd > 0$ **then**
    $symptomState \longleftarrow 1$; // Symptom is currently activated
    $count \longleftarrow count + 1$; // Increment the number of active symptoms by 1
  else
    $symptomState \longleftarrow 0$; // Symptom is currently not activated
return count

Algorithm 4: activationProbability

Input: $i, k, a^1_{ki}, b^1_{ki}, z^1_{ki}, c^2_i, v^1_{ki}, N^1(v^1_{ki}), N^2(v^2_i), v^2_i$
Output: p^1_{ki}
$r^2_{ki} \longleftarrow 0$; // Initialize student peer influence to 0
$q^1_{ki} \longleftarrow 0$; // Initialize symptom distant-1 neighbors influence to 0
foreach $v^1_{ci} \in N^1(v^1_{ki})$ **do**
    // Loop over the neighboring symptoms within the same student
    $q^1_{ki} \longleftarrow q^1_{ki} + w^1_{kici} x^1_{ci}$; // Calculate the student peer influence
foreach $v^2_j \in N^2(v^2_i)$ **do**
    // Loop over the corresponding symptoms in neighboring students (peers)
    $r^2_{ki} \longleftarrow r^2_{ki} + w^2_{ij} x^1_{kj}$; // Calculate the symptom distant-1 neighbors influence
$A^1_{ki} \longleftarrow c^2_i + q^1_{ki} + r^2_{ki} + z^1_{ki}$; // Calculate the total amount of stress on the symptom
$p^1_{ki} \longleftarrow \frac{1}{1+e^{a^1_{ki}(b^1_{ki} - A^1_{ki})}}$; // Calculate the activation probability
return p^1_{ki}

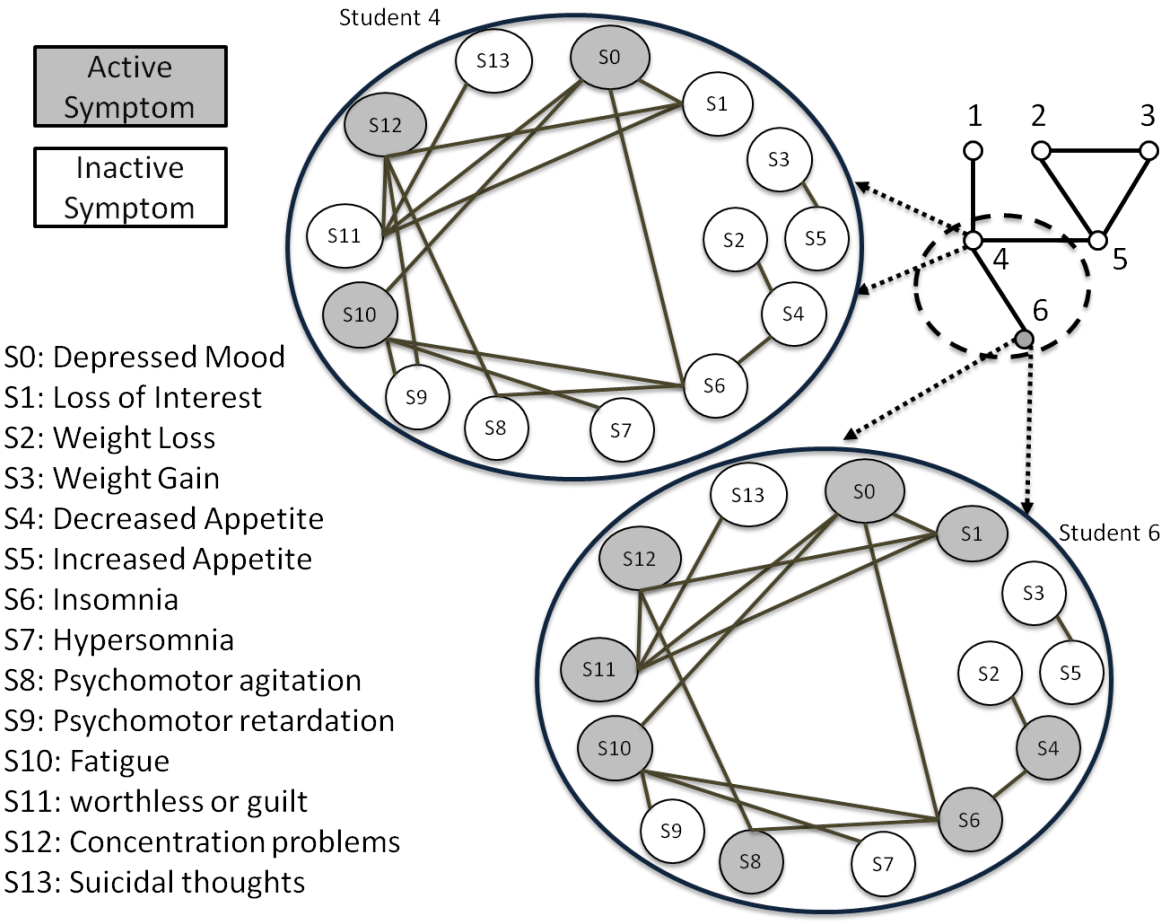


Figure 7.1. *Illustrative example network, showing local interactions between two students labeled 4 and 6, who are part of a larger student population of six agents; this is network G^2. The color of symptoms in the two large circles determine the symptom state which can be orange (active) or green (inactive). Each within-agent symptom network is an instance of G^1. The total number of active symptoms determines whether the agent will be in a healthy or a depressed state. For example, agent 6 has 8 active symptoms, and as a result, the agent color turns to orange (depressed). The with-in agent network is taken from [205].*

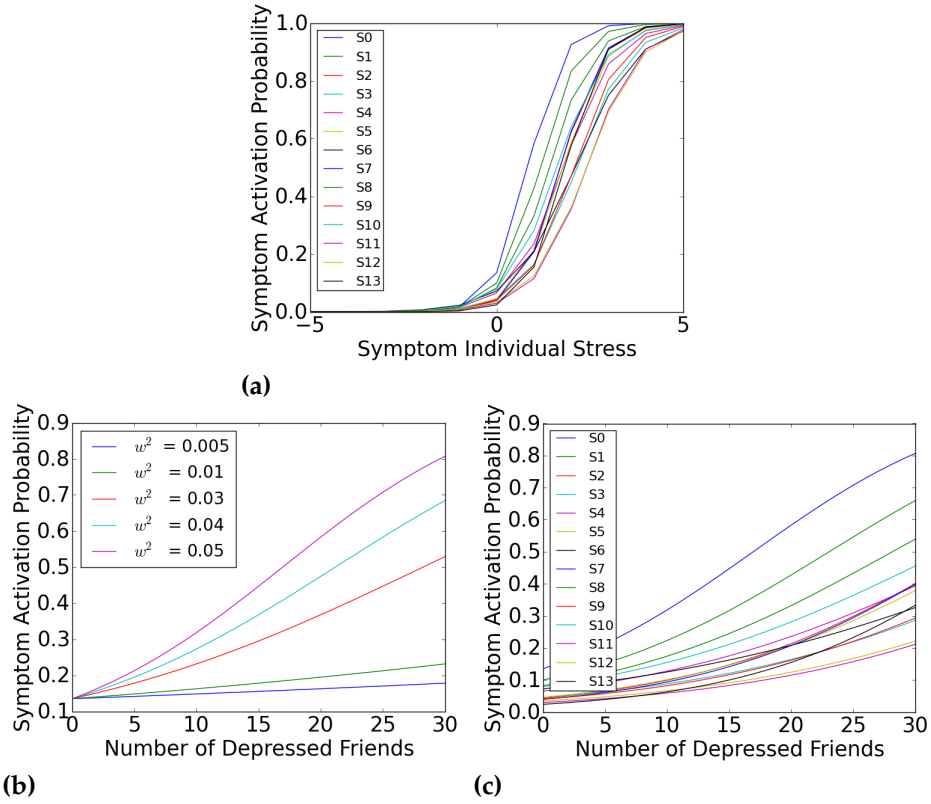


Figure 7.2. *Results for the depression model. (a) The changes in symptom activation probabilities for all symptoms as a function of stress level for each symptom. (b) The changes in symptom activation probability for symptom 0 as a function of number of depressed neighbors and the strength of peer influence. (c) The changes in symptom activation probabilities for all symptoms as a function of number of depressed neighbors.*

7.4 Simulation Results

7.4.1 Effect of peer influence on the number of depressed students

We start by assuming that all agents are free of depression symptoms. We evaluate the impact of peer influence on the final number of depressed students. We fix the model input parameters c_i^2, z_{ki}^1 and w_{kici}^1 to 0.5, 0 and 0.5 respectively. We systematically increase the edge weights w^2 between agents from 0.005 to 0.5. We use w^2 as the value of w_{ij}^2 for all i and j. Peer influence r_{ki}^2 increases as w^2 increases, as seen in Table 7.2. Each curve in the results of Figure 7.3a represents the average results from 50 simulation runs. Each curve also corresponds to a single value of w^2 for all edges in the social network. As w^2 increases, the peer influence and the number of depressed students increases. When w^2 increases by 67%, from 0.03 to 0.05, the number of depressed students increases by a factor of six, from 400 to 2400. Hence, we see the importance of modeling between-agent interactions through the social contact network.

7.4.2 Effect of symptom influence on the number of depressed students

We study the symptom influence within the same agent. All agents are initially free of depression symptoms. We fix other factors, including the peer influence, to isolate this effect. The fixed values for model parameters c_i^1, z_i^1 and w^2 are 0.5, 0 and 0.05 respectively. The results are shown in Figure 7.3b. We take $w_{kici}^1 = w^1$, for all symptoms k and c, and all agents i. As w^1 increases by 2.5×, from 0.2 to 0.5, the average number of depressed students increases by 16×, from 150 to 2400. The fraction of depressed students increases from 0.8% to 12.7%.

7.4.3 Students predisposed to depression

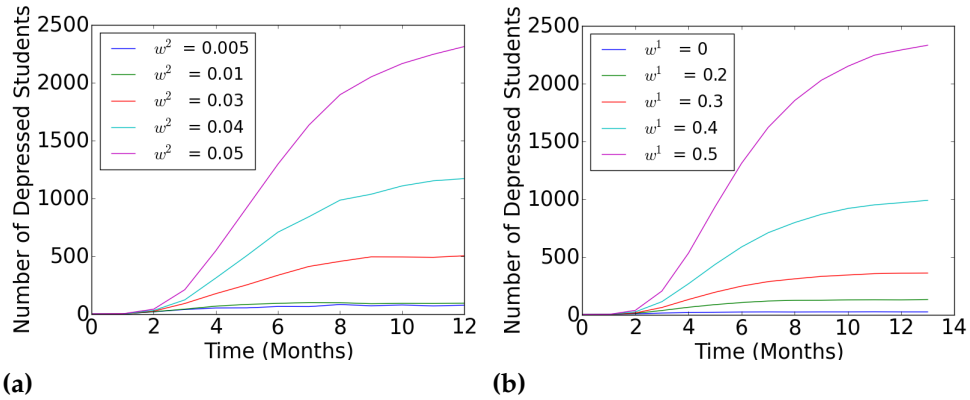


Figure 7.3. *(a) The time evolution of number of depressed students, showing that as peer influence increases, the number of depressed agents increases. (b) The time evolution of number of depressed students, showing that as the strength of symptom-to-symptom influence increases, the number of depressed agents increases. These two plots illustrate the importance of capturing agent interactions in the social network (in (a)), and capturing within-agent interactions (in (b)).*

Students may start their college years with symptoms and signs of depression. In previous simulations we assumed that the entire population is free of depression symptoms. In this experiment, we study how pre-existing depression symptoms might affect the speed at which depression spreads among students. Model parameters c^1_{ki}, z^1_{ki}, w^1 and w^2 are set to values 0.5, 0, 0.5 and 0.05, respectively. We used p and n to represent the probability p of having n active symptoms at time = 0. Figure 7.4a shows that as n and p increase, the speed of depression contagion increases earlier. The final number of depressed students converges over time.

As mentioned earlier, the 14 symptoms are connected through an internal causal network shown in Figure 7.1 that has 2 connected components, including a giant component of 12 of the 14 symptoms. In Figure 7.4b, we examine this symptom connectivity and two different (extreme) cases: all the symptoms are fully connected and all are disconnected. The partially connected curve corresponds to the connectivity shown

in Figure 7.1. Results illustrate the impact of having all symptoms fully connected in speeding the contagion process and having a larger number of depressed students at steady state. These data suggest the interesting result that internal agent connectivity dictates population level depression, at least for the conditions of these simulations. Also, the results for the connectivity of Figure 7.1 are much closer to those for disconnected symptoms than those for fully connected symptoms.

Another experiment on the symptom causal network aims to determine the effect of the most important symptoms, those most highly connected, being initially activated. Importance of symptoms is proportional to their degrees. The highest degree symptoms are (0, 1, 10 and 11) with degrees (4, 3, 4 and 4), respectively. The lowest degree symptoms are (3, 5, 7 and 13) with degrees (1, 1, 2 and 1), respectively. The results are shown in Figure 7.4c. The results show that activating the highest degree symptoms accelerates the depression at earlier times.

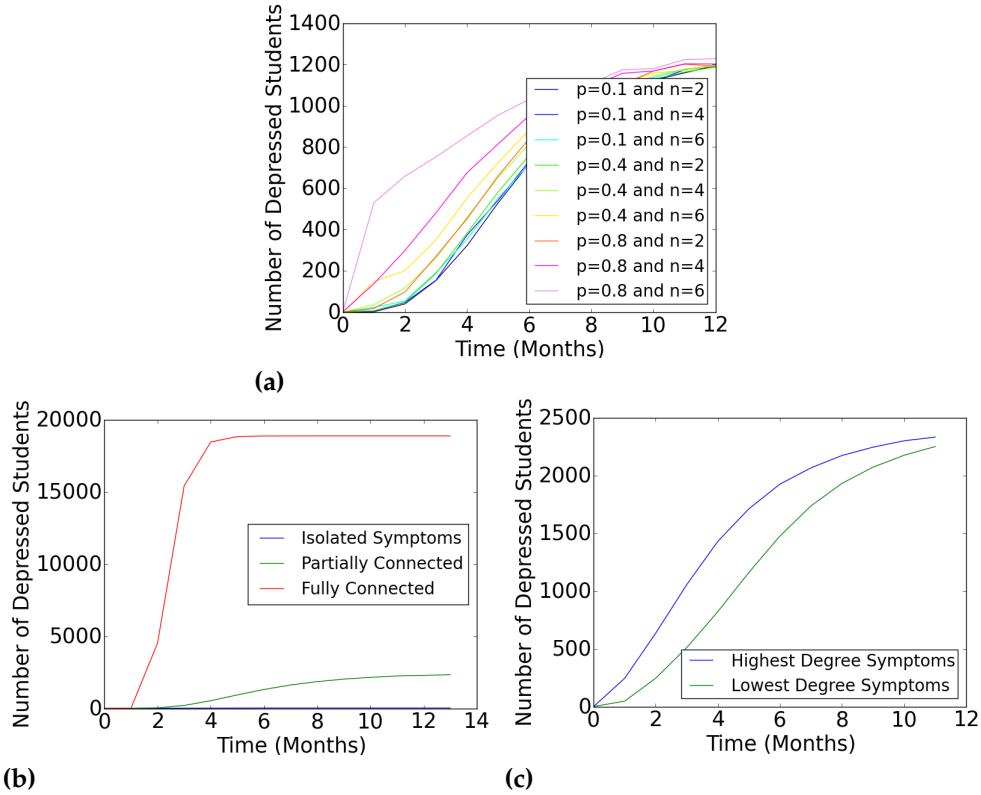


Figure 7.4. *(a) The time evolution of number of depressed students, showing that as p and n increase, the contagious speed of depression increases. (b) The time evolution of number of depressed students, showing the impact of having all symptoms fully connected in speeding the contagion process and having a larger number of depressed students at steady state. (c) The time evolution of number of depressed students, showing the impacts of having the highest degree and lowest degree symptoms initially activated.*

7.5 Discussion

Clearly our model is focused mainly on modeling and simulation of depression among students. We are motivated by several studies, which are cited earlier in Section 2.5. These studies identify and emphasize the importance of depression on a student's academic performance and retention. Therefore, we believe that the simulation results on depression can be used as the basis for evaluating the impact of depression on retention.

7.6 Implications

In order to decrease the rate of depressed students, both students and universities must fully understand the illness, the causes, and its effects. There is a need for developing programs that improve prevention, identification, and treatment of depression. Collected data reveals that around three percent of first-year students experiencing depressive symptoms seek help and counseling services [167]. Another study showed that living off campus was related to less knowledge of campus mental health services [222]. Existing programs focus on within individual student depression symptoms. Considerations should also be given to the contagious aspect of depression through peer influence. Both students and universities need to be able to identify risk factors, external stressors, and signs and symptoms of the disease.

Some work [114] suggested that university mental health programs can also benefit from adapting ideas that have already shown promising results in settings outside of college campuses. Some of Tinto's recommendations [200] to promote retention can be utilized to control depression through providing academic, social and personal support, particularly in and before the first year. Student led organizations may help reduce the effect of depression symptoms by increasing the student's engagement to the social network, specially for those who are vulnerable to depression [121].

7.7 Limitations

Among the limitations of this work are: (i) the lack of model validation and calibration using depression data from an actual population; and (ii) the absence of some agent traits that can affect the simulation results (e.g. age, gender, or household income). Possible extensions are: (i) a more highly resolved student population that accounts for other (social) activities such as extra circular activities; (ii) other student

populations to investigate network structure effects (e.g., universities primarily serving a commuter population versus a classic student population); and (*iii*) studying the effects of depression on academic achievement.

Chapter 8

A Survey on Research Practices and Usability Evaluation of EDISON System

8.1 Introduction

According to Greg Wilson [215], scientists spend a high percentage of research time developing software. However, only few received an adequate training to do this effectively. Technical excellence is no longer enough. Simulation tools also need to be easy to use and fit in the work practices and activities of the users [77]. In this chapter, we conduct a survey of scientists' research practices and a usability study to monitor, evaluate, and assess the usage of EDISON system.

8.2 Study Design

8.2.1 Mixed Methods Approach

The study follows a mixed method approach, specifically, we use the concurrent nested design approach as discussed in [209]. Both qualitative and quantitative data were collected on two concurrent strands. We conducted a survey of research practices. The goal is to identify common patterns among participants and their challenges during their research workflow. On another strand, we conducted a usability evaluation of EDISON system. EDISON is expected to have a large user base, due to its provided features, functionalities, and applicability of contagion dynamics in several domains. The reasons for using a mixed method approach is to combine the best of the in-depth, contextualized view of qualitative research with the objectivity of quantitative research. The goal was to use the strengths of one to overcome the weakness of the other. An overview of the study is given in Figure 8.1

8.2.2 Study Participants

The study includes 21 participants from Virginia Tech University. A request was sent to participants by email. Out of the total participants, fifteen were chosen for EDISON usability evaluation. The purposeful sampling strategy by John W Creswell was used to define the sampling pool. Participants were chosen based on their experience with agent-based modeling. Also, the final sample represents researchers with a different subject area and age groups. The main idea is that if participants were purposefully chosen to be different in the first place, then their responses will be different, and this will eventually lead to better qualitative and quantitative results.

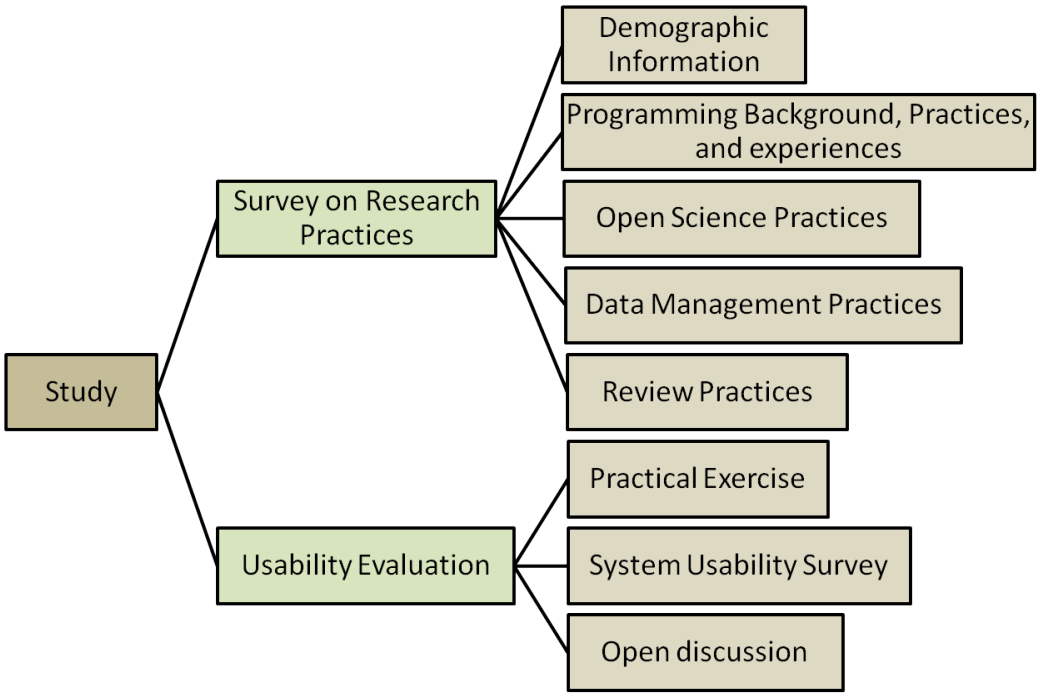


Figure 8.1. *An overview of the study which consists of a research practices survey and a usability evaluation of EDISON systems.*

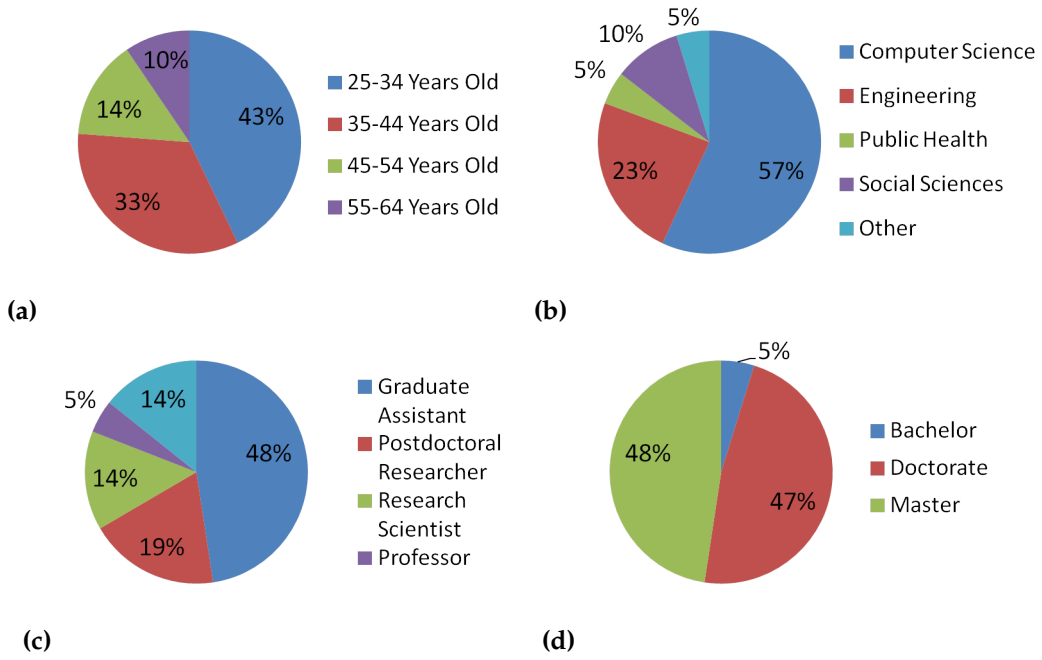


Figure 8.2. *Participants' distribution based on their(a) age, (b) subject area, (c) job, and (d) highest degree achieved.*

8.3 Research Practices Survey

8.3.1 Overview

The survey gathered both quantitative and qualitative data related to different forms of scientists' research practices. The survey took a questionnaire form. The questions followed either a Likert or a rating scale, in addition to, yes/no and open-ended questions. Some of the questions were selected from two studies [159] and [199]. The original studies address the various aspects of scientific computing and the data practices of researchers including data accessibility, discovery, re-use, preservation, and sharing.

8.3.2 Data Collection

Participants were given an overview of the study at the beginning. The facilitator was present to clarify any questions. Questionnaires were conducted separately in a face-to-face setting to avoid any validity threats. Each participant had to complete five sections: (i) demographic information, (ii) programming background and practices, (iii) open science practices, (iv) data management practices, and (v) review practices.

8.3.3 Results and Findings

The results and findings of the survey are addressed in the following subsections.

8.3.3.1 Programming Background, Practices & Experiences

Responses show that 25% of participants consider themselves below average regarding programming skills. At least 57% of participants spend more than 50% of their re-

search time programming. The three most frequently used programming languages are Python, Matlab, C/C++. Around 20% of scientists spend on average more than 12 hours to complete an experiment execution, which can last sometimes up to days or weeks. Around 76% of participants use a combination of multiple programming languages during research, which imposes a challenge to researchers with no or little programming skills. Some of the results are illustrated in Figure 8.3.

8.3.3.2 Open Science Practices

Results show that more than 75% of participants agree that lack of access to data generated by other researchers or institutions can affect their progress in research. More than 60% of participants believe that data can be misinterpreted due to poor standardization. Also more than 25% of participants, most of the time, find difficulty running others' code, and around 50% find challenges in modifying or adding features to the code. Around 90% of participants mention that sometimes they find difficulty replicating or reproducing others' work. The reasons are grouped and listed in Table 8.1.

8.3.3.3 Data Management Practices

Although 100% of participants feel that long-term storing of data is important for their research, 62% feel unsatisfied with their current practices. Additionally, around 95% of participants believe organizing data is necessary for their research. However, 75% feel unsatisfied with the current process of organizing data.

8.3.3.4 Review Practices

This subsection focuses on the problems and challenges researchers have while reviewing others' work. These challenges usually happen during the publication pro-

Table 8.1. *Sources of problems embedding researchers from replicating or reproducing others' work .*

Documentation or Details Related	**Code or Data Related**
Unclear or missing software or code settings	Missing input/out data or code
Unclear or missing heuristics description	Difficulty in producing executable
Unclear or missing pre- or post-processing steps details	Data format is not described
No clear description or missing steps of algorithm(s) used	Unexpected errors with missing troubleshooting
Unclear or missing experiment steps	Specific platform needed (e.g. Cluster or Operating system)
–	Code and data versions are not compatible
–	Very large and complex code.

cess, a phase of the research life cycle. Around 50% of participants feel that availability of code and data is necessary for the publication process. More than 70% of participants agree that providing code and data at the time of manuscript submission is important. The unavailability of code, unavailability of data, and the unclear description of experiments are the three most common challenges during the review process.

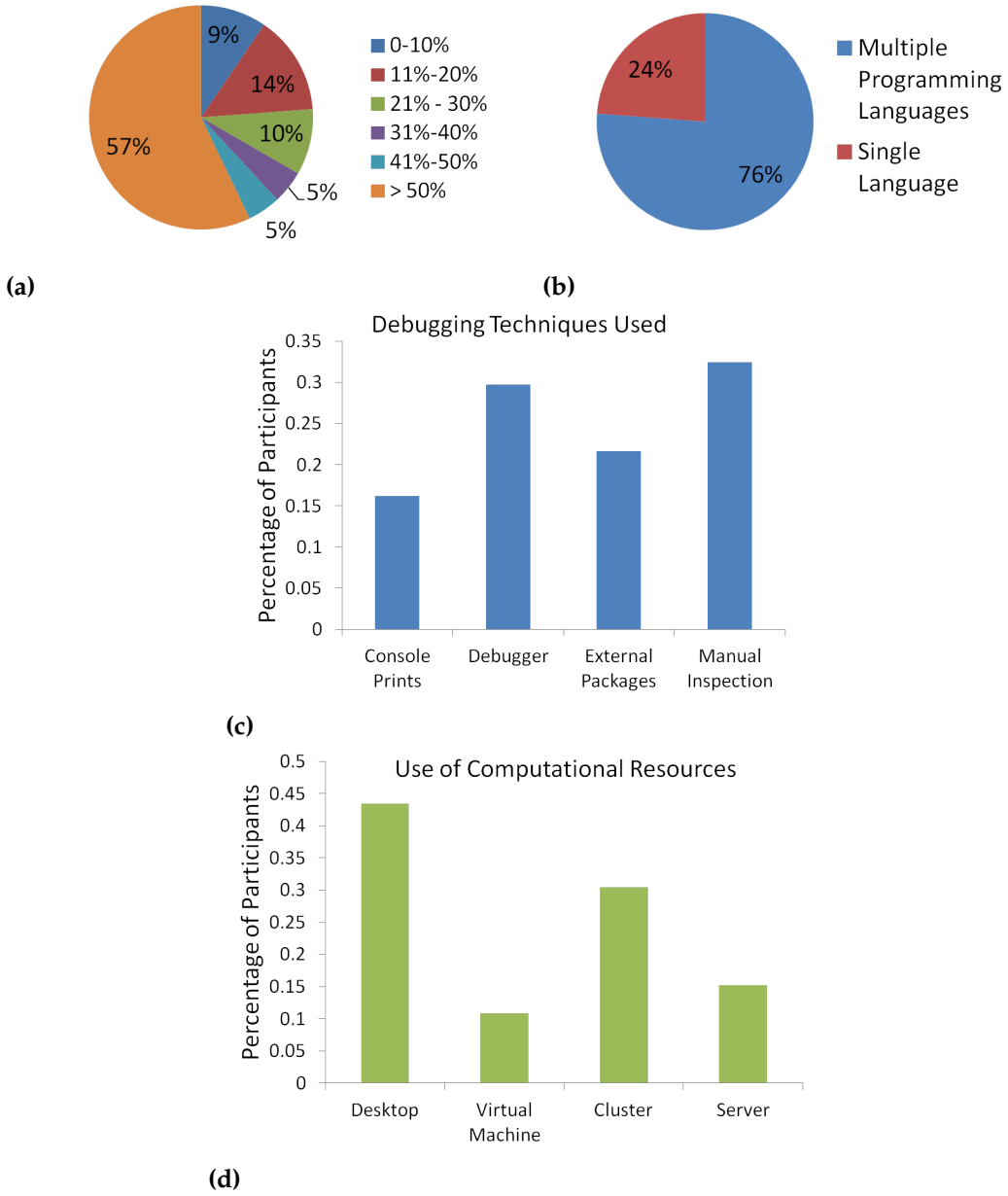


Figure 8.3. *(a) Time spent programming as percentage of research time. (b) Number of programming languages used. (c) Distribution of debugging techniques. (d) Computational Resource Use.*

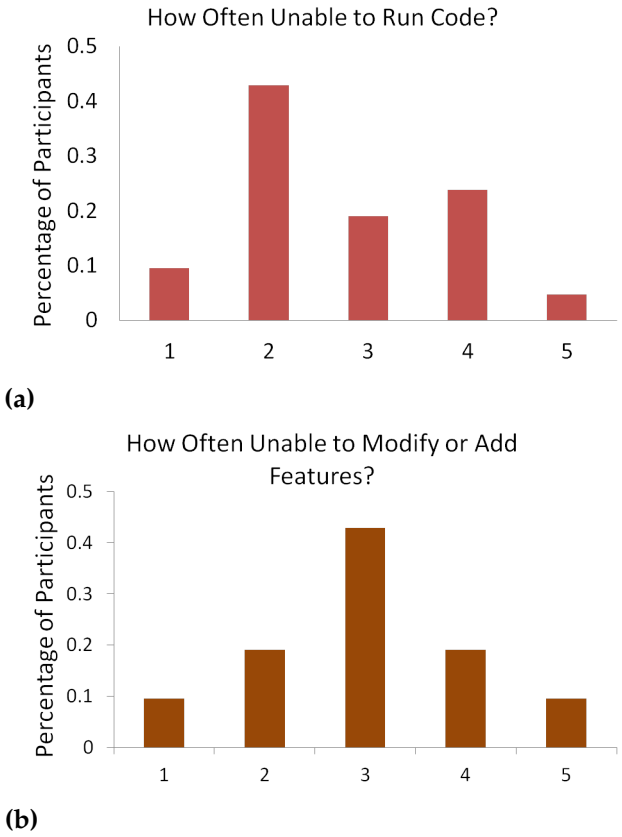


Figure 8.4. *(a) Response distribution of the inability to run others' code. (b) Response distribution of the inability to modify or change others' code. The x-axis scale is from 0 (rarely) to 5 (always).*

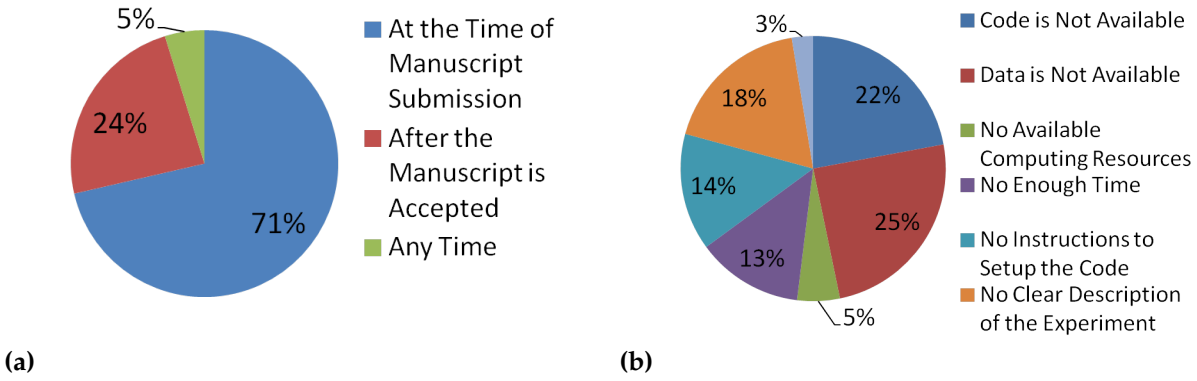


Figure 8.5. *(a) Response distribution of the preferred timing to provide code or data. (b) Response distribution of the challenges that can prevent scientists from reviewing others' work.*

8.4 Usability Evaluation of EDISON System

8.4.1 Overview

This part of the study is to evaluate the web-based user interface of EDISON, and measure the usability of the system. Inorder to do so, participants' feedback and interaction with the UI were recorded.

8.4.2 Data Collection

The data collection is described in Figure 8.7. Participants were given an overview demo of EDISON; then they were provided similar exercises. A facilitator was available to provide help, answer questions, and take observations. Following each exercise, participants took the System Usability Scale (SUS) questionnaire. The final step was an open discussion to capture additional user feedback and suggestions for future improvements. Additional one-to-one interviews were conducted with those participants who had experience using back-end simulators. The simulators included InterSim, the back-end simulator for EDISON, EpiFast [45], and EpiSimdemics [28].

SUS questionnaire was chosen for different reasons: (i) it has been widely used and proven to be reliable [41], (ii) SUS can interpret EDISON's usability rank among other systems using Sauro's benchmark database [181], (iii) SUS can address both usability and learnability [142], and (iv) it has shown better performance when compared with other questionnaires (e.g. QUIS (Questionnaire for User Interface Satisfaction), CSUQ(Computer System Usability Questionnaire), and Words (adapted from MicrosoftâĂŹs Product Reaction Cards)) [202]. Following each user session, we calculated the individual SUS and the average SUS scores. The percentile rank was determined based on the work by Sauro [181].

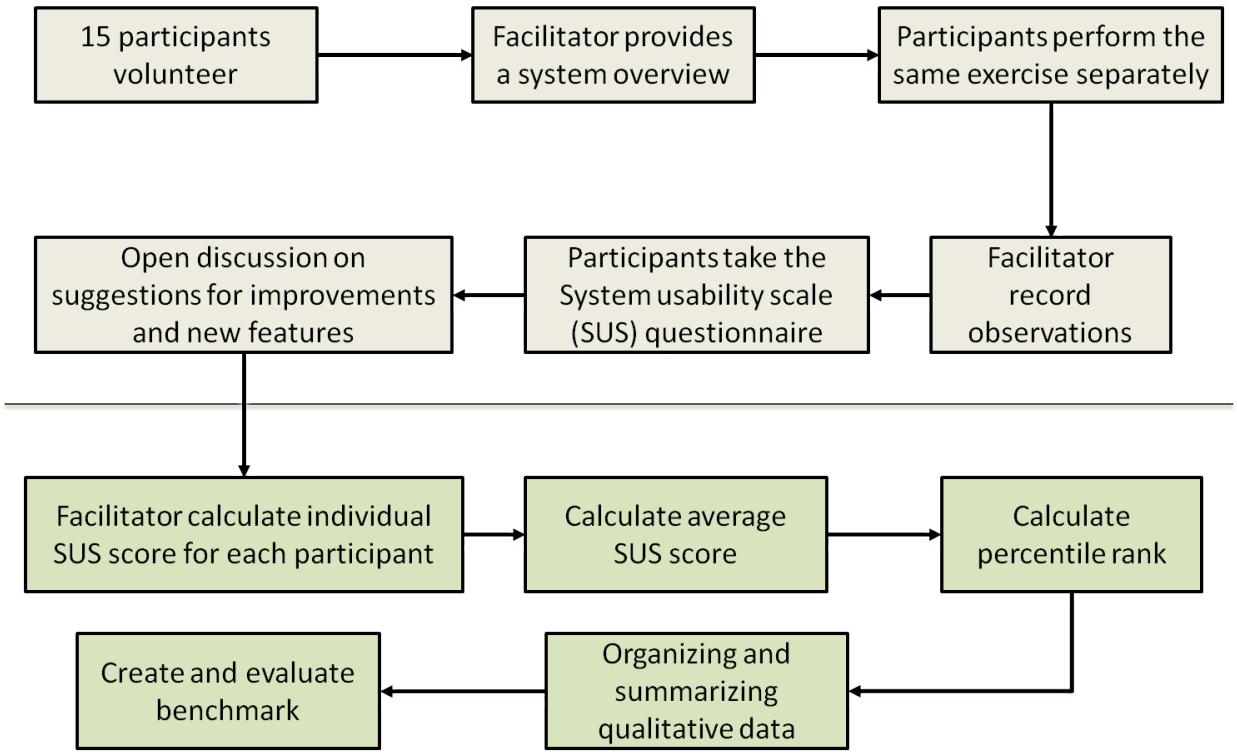


Figure 8.6. *An overview of EDISON usability evaluation steps.*

8.4.3 Results and Findings

8.4.3.1 System Usability Scale

The average SUS score is seventy six which translates into a 77% percentile rank. This means that EDISON is more usable more than 77% of products that were evaluated in the Sauro Database. According to [180], SUS scores above 80.3 translate into letter grade *A*, while scores around the average score of 68 mean a grade *C*. Below a fifty one is considered a grade *F*.

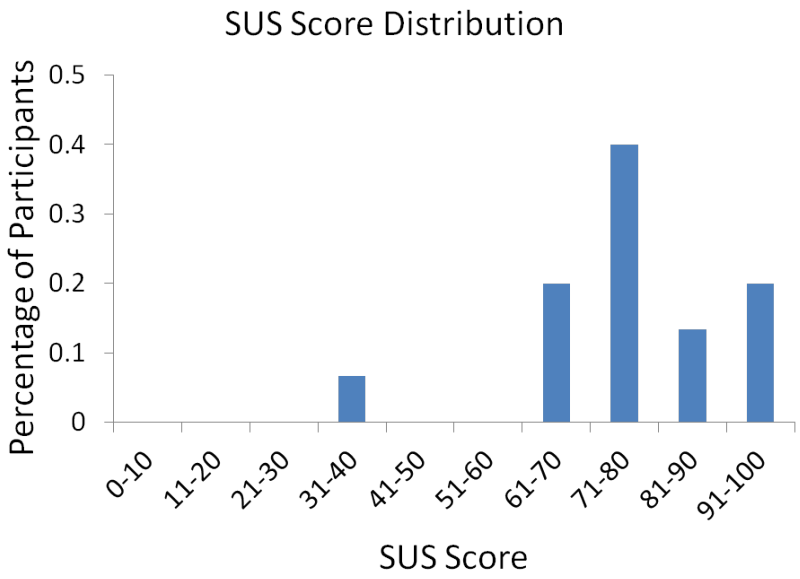


Figure 8.7. *Distribution of the raw SUS scores from the 15 participants. The average score is 76.*

8.4.3.2 Usability Benchmark

While the SUS questionnaire has shown reliability and applicability in several studies, it provides a non-detailed assessment of the system usability. To have a detailed view, a benchmark was created as described in Table 8.2 based on the open discussion with participants with back-end simulator experience. The open discussion revealed that the minimum time to prepare, create and submit an experiment was on average 12 hrs, which was the best case scenario where users encountered no problems. The maximum time was around (3.5 days). Some of the common problems they encountered were debugging code for errors, understanding the existing manual, and post-processing the generated output files. Moreover, when a new model needs to be added to the system, it can take around two to three days to test and validate the model.

Table 8.2. *Benchmark to assess EDISON usability and productivity*

Criteria / Task	**Expected**	**Actual**
Time to create an experiment	19 mins	8.66 mins
Enter experiment metadata	2 mins	2 mins
Choose network	2 mins	< 1 min
Choose dynamic model	2 mins	< 1 min
Enter model properties	7 mins	3 mins
Enter seeds properties	6 mins	2 mins
SUS Score	68	76
Participants ask for help	Yes	Yes
Participants will need future training	No	Yes
Time to replicate experiment	< 1 min	< 1 min
Percentage of successful experiments	100%	100%
Time and usability score correlation	Strong	Medium

8.5 Limitations

Among the possible improvements to this work are: (i) use of some technologies that gather more valuable data on the human behavior (e.g. eye-tracking and clicks heatmap); and (ii) use of a larger sample size that includes greater diversity in users, since software use cuts across almost all academic disciplines. A limitation of this work is the late evaluation of EDISON usability, which should start earlier in the design process.

Chapter 9

Conclusions and Future Directions

The goal of this research is to design, build, and evaluate systems and services that enable domain experts and other users (including computer scientists) to study contagion dynamics on networked populations more effectively. The systems and services collectively form a research framework to study network dynamics. The primary objectives are to: (*i*) bridge the gap between the mathematical theory of simulation (using GDS) and high-performance computing (HPC) design and implementation; (2) provide an infrastructure that supports research reproducibility, increases users' productivity and promotes collaboration among scientists; and (3) provide the medium for reviewers and other researchers to verify the simulation findings. We contributed towards these objectives by implementing four main components: (1) an open access distributed web application for characterizing GDSs (GDSCalc), (2) a web application for evaluation of network-based social dynamics (EDISON), (3) a suite of supporting network services (MARS), and (4) a web application for interactive exploration and understanding of contagion dynamics in networked populations (NEMO). Illustrative case studies were conducted to illustrate the utility and usefulness of the developed systems and services. Additionally, the performance of several

components was evaluated to assess the quality of services. Finally, we conducted a usability study to evaluate the usability and learnability of selected systems. Figure 9.1, provides an overview of the future directions for the implemented systems and services.

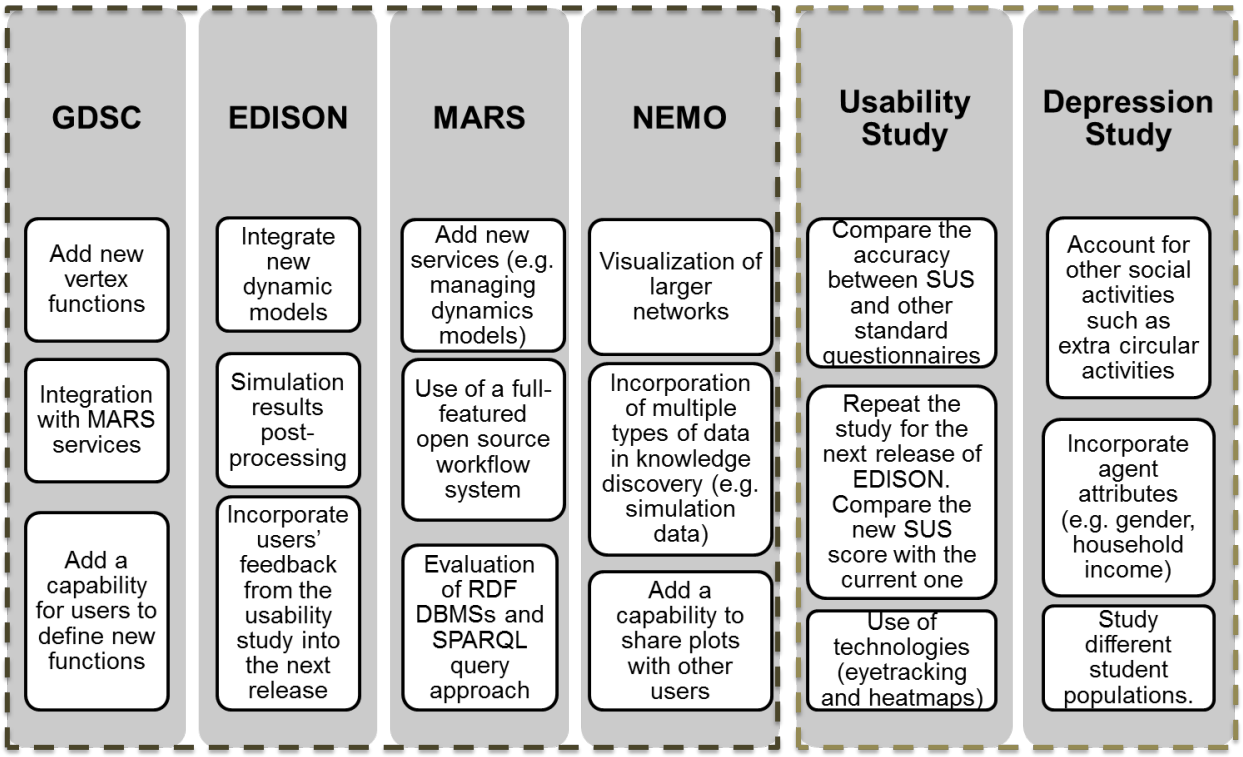


Figure 9.1. *Possible future work for each component of the implemented framework.*

Bibliography

1. *MATLAB Random Boolean Network Toolkit*. MathWorks, Inc., 2007.

2. *Mathematica*. Wolfram Research, Inc., 2012. Edition 9.0.

3. *MATLAB and Statistics Toolbox Release 2012b*. MathWorks, Inc., 2012.

4. GDSCalc, 2014. http://taos.vbi.vt.edu/gdscalc/welcome.html.

5. Abdelhamid, S. E., Kuhlman, C. J., Marathe, M. V., Mortveit, H. S., and Ravi, S. GDSCalc: A web-based application for evaluating discrete graph dynamical systems. *PloS one 10*, 8 (2015), e0133660.

6. Abdelhamid, S. E., Kuhlman, C. J., Marathe, M. V., Ravi, S. S., and Reid, K. Agent-based modeling and simulation of depression and its impact on student success and academic retention. In *2016 ASEE Annual Conference & Exposition* (New Orleans, Louisiana, June 2016), no. 10.18260/p.26545, ©2016 American Society for Engineering Education. https://peer.asee.org/26545.

7. Abdelhamid, S. E. M., Kuhlman, C. J., Korkmaz, G., Marathe, M. V., and Ravi, S. S. EDISON: a web application for computational health informatics at scale. In *Proceedings of the 6th ACM BCB Conference* (2015), ACM, pp. 413–422.

8. Abdelhamid, S. E. M., Kuhlman, C. J., Marathe, M. V., and Ravi, S. Network services and their compositions for network science applications. *Procedia Computer Science 80* (2016), 472–483.

9. Adelman, C. Answers in the tool box. academic intensity, attendance patterns, and bachelor's degree attainment, 1999.

10. Adiga, A., Kuhlman, C. J., Marathe, M. V., Ravi, S., Rosenkrantz, D. J., and Stearns, R. E. Inferring local transition functions of discrete dynamical systems from observations of system behavior. *Theoretical Computer Science* (2016).

11. Adiga, A., Kuhlman, C. J., Mortveit, H. S., and Vullikanti, A. Sensitivity of diffusion dynamics to network uncertainty. *Journal of Artificial Intelligence Research (JAIR) 51* (2014), 207–226.

12. Alkhateeb, F., Baget, J., and Euzenat, J. Extending SPARQL with regular expression patterns (for querying RDF). *Web Semantics: Science, Services and Agents on the World Wide Web 7* (2009), 57–73.

13. Allan, R. J. Survey of agent based modelling and simulation tools. TR 1999-66, STFC, 2010.

14. Amir-Kroll, H., Sadot, A., Cohen, I. R., and Harel, D. Gemcell: A generic platform for modeling multi-cellular biological systems. *Theoretical Computer Science 391* (2008), 276–290.

15. Angenendt, S., Asseburg, M., Bank, A., Difraoui, A. E., Freitag, U., Glosemeyer, I., Lacher, W., Niethammer, K., Perthes, V., Posch, W., Roll, S., Thimm, J., and Westphal, K. Protest, revolt, and regime change in the arab world 2012. Tech. rep., German Institute for International and Security Affairs, 2012.

16. Angles, R., and Gutierrez, C. Survey of graph database models. *ACM Computing Surveys (CSUR) 40*, 1 (2008), 1.

17. Aranguren, M. E., and Wilkinson, M. D. Enhanced reproducibility of sadi web service workflows with Galaxy and Docker. *GigaScience 4*, 59 (2015).

18. Association, A. C. H. American college health association national college health assessment spring 2006 reference group data report (abridged). *Journal of American College Health 55*, 4 (2007), 195.

19. Astin, A. W. Student involvement: A developmental theory for higher education. *Journal of college student personnel 25*, 4 (1984), 297–308.

20. Aziz, A. A., Klein, M. C. A., and Treur, J. An agent model of temporal dynamics in relapse and recurrence in depression. *Next-Generation Applied Intelligence* (2009), 36–45.

21. Baas, N. A., and Helvik, T. Higher Order Cellular Automata. *Advances in Complex Systems 8* (2005), 169–192.

22. Bandura, A. *Aggression: A social learning analysis.* Prentice-Hall, 1973.

23. Bank, B. J., Slavings, R. L., and Biddle, B. J. Effects of peer, faculty, and parental influences on students' persistence. *Sociology of Education 63* (1990), 208–225.

24. Barrett, C., Hunt, H. B., Marathe, M. V., Ravi, S., Rosenkrantz, D. J., and Stearns, R. E. Analysis problems for sequential dynamical systems and communicating state machines. In *International Symposium on Mathematical Foundations of Computer Science* (2001), Springer, pp. 159–172.

25. Barrett, C., Hunt, H. B., Marathe, M. V., Ravi, S., Rosenkrantz, D. J., and Stearns, R. E. On some special classes of sequential dynamical systems. *Annals of Combinatorics 7*, 4 (2003), 381–408.

26. Barrett, C., Hunt, H. B., Marathe, M. V., Ravi, S., Rosenkrantz, D. J., and Stearns, R. E. Reachability problems for sequential dynamical systems with threshold functions. *Theoretical Computer Science 295*, 1 (2003), 41–64.

27. Barrett, C. L., Beckman, R. J., Khan, M., Anil Kumar, V. S., Marathe, M. V., Stretz, P. E., Dutta, T., and Lewis, B. Generation and analysis of large synthetic social contact networks. In *Winter Simulation Conference* (2009), Winter Simulation Conference, pp. 1003–1014.

28. Barrett, C. L., Bisset, K. R., Eubank, S. G., Feng, X., and Marathe, M. V. Episimdemics: an efficient algorithm for simulating the spread of infectious disease over large realistic social networks. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (2008), IEEE Press, p. 37.

29. Barrett, C. L., Hunt, H. B., Marathe, M. V., Ravi, S., Rosenkrantz, D. J., and Stearns, R. E. Complexity of reachability problems for finite discrete dynamical systems. *Journal of Computer and System Sciences 72*, 8 (2006), 1317–1345.

30. Barrett, C. L., Hunt, H. B., Marathe, M. V., Ravi, S. S., Rosenkrantz, D. J., Stearns, R. E., and Thakur, M. Predecessor Existence Problems for Finite Discrete Dynamical Systems. *Theor. Comput. Sci. 386*, 1-2 (2007), 3–37.

31. Bartocci, E., Corradini, F., Merelli, E., and Scortichini, L. BioWMS: A web-based workflow management system for bioinformatics. *BMC Bioinformatics 8*, S-1 (2007).

32. Bass, F. M. A new product growth for model consumer durables. *Management Science 15* (1969), 215–227.

33. Bastian, M., Heymann, S., and Jacomy, M. Gephi: An open source software for exploring and manipulating networks.

34. Bean, J. P. Dropouts and turnover: The synthesis and test of a causal model of student attrition. *Research in higher education 12*, 2 (1980), 155–187.

35. Bean, J. P. The synthesis of a theoretical model of student attrition, 1981.

36. Beane, J. A., and Lipka, R. P. *Self-concept, self-esteem, and the curriculum*. Columbia University, Teachers College, 1986.

37. Beaudouin-Lafon, M. Human-computer interaction. In *Interactive Computations: The New Paradigm* (2006), pp. 227–254.

38. Begley, C. G., and Ellis, L. M. Drug development: Raise standards for preclinical cancer research. *Nature 483*, 7391 (2012), 531–533.

39. Behnke, R. R., Sawyer, C. R., and King, P. E. Contagion theory and the communication of public speaking state anxiety. *Communication Education 43*, 3 (1994), 246–251.

40. Belhajjame, K., Zhao, J., Garijo, D., Gamble, M., Hettne, K., Palma, R., Mina, E., Corcho, O., Gómez-Pérez, J. M., Bechhofer, S., Klyne, G., and Goble, C. Using a suite of ontologies for preserving workflow-centric research objects. *Web Semantics: Science, Services and Agents on the World Wide Web 32*, 0 (2015).

41. Berman, M. I., Buckey, J. C., Hull, J. G., Linardatos, E., Song, S. L., McLellan, R. K., and Hegel, M. T. Feasibility study of an interactive multimedia electronic problem solving treatment program for depression: a preliminary uncontrolled trial. *Behavior therapy 45*, 3 (2014), 358–375.

42. Bernstein, R. Depression afflicts almost half of STEM graduate students at UC berkeley. *Science* (may 2015).

43. Bessiere, K., Newhagen, J. E., Robinson, J. P., and Schneiiderman, B. A model for computer frustration: the role of instrumental and dispositional factors on incident, session, and post-session frustration and mood. *Computers in Human Behavior 22* (2006), 941–961.

44. Bischi, G., and Merlone, U. Global dynamics in binary choice models with social influence. *J. Math. Soc. 33* (2009).

45. Bisset, K. R., Chen, J., Feng, X., Kumar, V. S., and Marathe, M. V. Epifast: a fast algorithm for large scale realistic epidemic simulations on distributed memory systems. In *Proceedings of the 23rd international conference on Supercomputing* (2009), ACM, pp. 430–439.

46. Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment 10* (2008), P10008–1–P10008–12.

47. Bonnet, P., Manegold, S., Bjørling, M., Cao, W., Gonzalez, J., Granados, J., Hall, N., Idreos, S., Ivanova, M., Johnson, R., Koop, D., Kraska, T., MÃijller, R., Olteanu, D., Papotti, P., Reilly, C., Tsirogiannis, D., Yu, C., Freire, J., and Shasha, S. Repeatability and workability evaluation of sigmod 2011. *ACM SIGMOD Record 40*, 2 (2011), 45–48.

48. Borner, K. Plug-and-Play Macroscopes. *Communications of the ACM 54*, 3 (2011), 60–69.

49. Borsboom, D. Psychometric perspectives on diagnostic systems. *Journal of clinical psychology 64*, 9 (2008), 1089–1108.

50. Both, F., Hoogendoorn, M., Klein, M. C. A., and Treur, J. Modeling the dynamics of mood and depression, 2008.

51. Boyko, M., Kutz, R., Grinshpun, J., Zvenigorodsky, V., Gruenbaum, S. E., Gruenbaum, B. F., Brotfain, E., Shapira, Y., and Zlotnik, A. Establishment of an animal model of depression contagion. *Behavioural brain research 281* (2015), 358–363.

52. Burch, M. Dynamic graph visualization with multiple visual metaphors. In *Proceedings of the 8th International Symposium on Visual Information Communication and Interaction (VINCI)* (New York, NY, USA, 2015), VINCI '15, ACM, pp. 27–34.

53. Callaghan, S., Deelman, E., Gunter, D., Juve, G., Maechling, P., Brooks, C., Vahi, K., Milner, K., Graves, R., Field, E., Okaya, D., and Jordan, T. Scaling up workflow-based applications. *J. Comput. Syst. Sci. 76*, 6 (Sept. 2010), 428–446.

54. Callahan, K. M. Academic-centered peer interactions and retention in undergraduate mathematics programs. *Journal of College Student Retention: Research, Theory & Practice 10*, 3 (2008), 361–389.

55. Carmichael, C. L., Reis, H. T., and Duberstein, P. R. In your 20s it's quantity, in your 30s it's quality: The prognostic value of social activity across 30 years of adulthood. *Psychology and aging 30*, 1 (2015), 95.

56. Carroll, L., Au, A., Detwiler, L., Fu, T., Painter, I., and Abernethy, N. Visualization and analytics tools for infectious disease epidemiology: A systematic review. *Journal of Biomedical Informatics 51* (10 2014), 287–298.

57. Centola, D. The spread of behavior in an online social network experiment. *Science 329* (2010), 1194–1197.

58. Centola, D., and Macy, M. Complex contagions and the weakness of long ties. *American Journal of Sociology 113*, 3 (2007), 702–734.

59. Chau, D. H., Kittur, A., Hong, J. I., and Faloutsos, C. Apolo: Making sense of large network data by combining rich user interaction and machine learning. In *SIGCHI, Human Factors in Computing Systems* (2011), pp. 167–176.

60. Chebotko, A., Lu, S., Fei, X., and Fotouh, F. RDFPROV: A relational rdf store for querying and managing scientific workflow provenance. *Data and Knowledge Engineering 69* (2010), 836–865.

61. Chebotko, A., Lu, S., and Fotouh, F. Semantics preserving SPARQL-to-SQL translation. *Data and Knowledge Engineering 68* (2009), 973–1000.

62. Chen, J., Boulakia, S. C., Froidevaux, C., Goble, C. A., Missier, P., and Williams, A. R. Distillflow: removing redundancy in scientific workflows. In *Conference on Scientific and Statistical Database Management, SSDBM '14, Aalborg, Denmark, June 30 - July 02, 2014* (2014), pp. 46:1–46:4.

63. Chwe, M. S. Structure and strategy in collective action. *American Journal of Sociology 105*, 1 (1999), 128–156.

64. Colizza, V., Pastor-Satorras, R., and Vespignani, A. Reaction–diffusion processes and metapopulation models in heterogeneous networks. *Nature Physics 3*, 4 (2007), 276–282.

65. Collberg, C., and Proebsting, T. A. Repeatability in computer systems research. *Communications of the ACM 59*, 3 (2016), 62–69.

66. Connolly, T., and Åberg, L. Some contagion models of speeding. *Accident Analysis & Prevention 25*, 1 (1993), 57–66.

67. Coyne, J. C. Depression and the response of others. *Journal of abnormal psychology 85*, 2 (1976), 186.

68. Cramer, A. O. J., Waldorp, L. J., van der Maas, H. L. J., and Borsboom, D. Comorbidity: a network perspective. *Behavioral and Brain Sciences 33*, 2-3 (2010), 137–150.

69. Crane, J. The epidemic theory of ghettos and neighborhood effects on dropping out and teenage childbearing. *American Journal of Sociology 96*, 5 (1991), 1226–1295.

70. De Choudhury, M., Counts, S., and Horvitz, E. Social media as a measurement tool of depression in populations. In *Proceedings of the 5th Annual ACM Web Science Conference* (2013), ACM, pp. 47–56.

71. Deelman, E., Gannon, D., Shields, M., and Taylor, I. Workflows and e-science: An overview of workflow system features and capabilities. *Future Gener. Comput. Syst. 25*, 5 (May 2009), 528–540.

72. Demongeot, J., Goles, E., Morvan, M., Noual, M., and Sene, S. Attraction basins as gauges of robustness against boundary conditions in biological complex systems. *PLoS One 5*, 8 (2010), e11793–1–e11793–18.

73. Deziel, M., Olawo, D., Truchon, L., and Golab, L. Analyzing the mental health of engineering students using classification and regression. In *EDM* (2013), pp. 228–231.

74. Dodds, P. S., and Watts, D. J. A Generalized Model of Social and Biological Contagion. *Journal of Theoretical Biology 232*, 4 (2005), 587–604.

75. Dong, Y., Yang, Y., Tang, J., Yang, Y., and Chawla, N. V. Inferring user demographics and social strategies in mobile social networks. In *KDD* (2014).

76. Droop, A. P. qsubsec: a lightweight template system for defining sun grid engine workflows. *Bioinformatics* (2014).

77. Dube, E., Naidoo, S., and Addresses, E. Usability and information access challenges in complex simulation models, 2008.

78. Dubrova, E. Bns (boolean networks with synchronous update), 2013.

79. Durkheim, E. *Suicide: A Study In Sociology*. Free Press, 1997.

80. Easley, D., and Kleinberg, J. *Networks, Crowds and Markets: Reasoning About A Highly Connected World*. Cambridge University Press, New York, NY, 2010.

81. Eisenberg, D., Golberstein, E., and Hunt, J. B. Mental health and academic success in college. *The BE Journal of Economic Analysis & Policy 9*, 1 (2009).

82. Elmroth, E., Hernï£¡ndez, F., and Tordsson, J. Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. *Future Generation Comp. Syst. 26*, 2 (2010), 245–256.

83. Ennett, S. T., Flewelling, R. L., Lindrooth, R. C., and Norton, E. C. School and neighborhood characteristics associated with school rates of alcohol, cigarette, and marijuana use. *Journal of Health and Social Behavior* (1997), 55–71.

84. Epstein, J. *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton University Press, 2006.

85. Eubank, S., Kumar, V. A., Marathe, M. V., Srinivasan, A., and Wang, N. Structure of social contact networks and their impact on epidemics. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 70* (2006), 181.

86. Fates, N. Fiatlux, 2013.

87. Finkelstein, E., Trogdon, J., Cohen, J., and Dietz, W. Annual Medical Spending Attributable to Obesity: Payer- and Service-Specific Estimates. *Health Affairs 28*, 5 (2009), w822–w831.

88. Freedman, J. L., Birsky, J., and Cavoukian, A. Environmental determinants of behavioral contagion: Density and number. *Basic and Applied Social Psychology 1*, 2 (1980), 155–161.

89. Freedman, J. L., and Perlick, D. Crowding, contagion, and laughter. *Journal of Experimental Social Psychology 15*, 3 (1979), 295–303.

90. Garijo, D., Alper, P., Belhajjame, K., Corcho, O., Gil, Y., and Goble, C. Common motifs in scientific workflows: An empirical analysis. *Future Generation Computer Systems* (2013).

91. Gaviria, A., and Raphael, S. School-based peer effects and juvenile behavior. *The Review of Economics and Statistics 83* (2001), 257–268.

92. Gershenson, C. *RBNLab*. 2014.

93. Gezelter, J. D. Open science and verifiability, 2015.

94. Ghose, A., and Han, S. P. An empirical analysis of user content generation and usage behavior on the mobile internet. *Management Science* (2011), 1–21.

95. Gil, Y. Intelligent workflow systems and provenance-aware software. In *Proceedings of the Seventh International Congress on Environmental Modeling and Software* (2014).

96. Gil, Y., Ratnakar, V., Kim, J., González-Calero, P. A., Groth, P., Moody, J., and Deelman, E. WINGS: Intelligent workflow-based design of computational experiments. *IEEE Intelligent Systems 26*, 1 (2011).

97. Goble, C. A., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D. T., Newman, D. R., Borkum, M., Bechhofer, S., Roos, M., Li, P., and Roure, D. D. myexperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic Acids Research 38*, Web-Server-Issue (2010), 677–682.

98. Goles, E., and Martinez, S. *Neural and Automata Networks*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.

99. Gou, L., Zhang, X., Luo, A., and Anderson, P. F. Socialnetsense: Supporting sensemaking of social and structural features in networks with interactive visualization. In *VAST* (2012), pp. 133–142.

100. Granovetter, M. Threshold Models of Collective Behavior. *American Journal of Sociology 83*, 6 (1978), 1420–1443.

101. Granovetter, M., and Soong, R. Threshold models of diffusion and collective behavior. *J. Mathematical Sociology 9* (1983), 169–179.

102. Gruttadaro, D., and Crudo, D. College students speak: A survey report on mental health. *Survey Report). Arlington, VA: National Alliance on Mental Illness* (2012).

103. Gump, B. B., and Kulik, J. A. Stress, affiliation, and emotional contagion. *Journal of personality and social psychology 72*, 2 (1997), 305.

104. Güting, R. H. Graphdb: Modeling and querying graphs in databases. In *VLDB* (1994), vol. 94, Citeseer, pp. 12–15.

105. Gyssens, M., Paredaens, J., Van den Bussche, J., and Gucht, D. V. A graph-oriented object database model. *Knowledge and Data Engineering, IEEE Transactions on 6*, 4 (1994), 572–586.

106. Haeffel, G. J., and Hames, J. L. Cognitive vulnerability to depression can be contagious. *Clinical Psychological Science 2*, 1 (2014), 75–85.

107. Hallac, D., Leskovec, J., and Boyd, S. Network lasso: Clustering and optimization in large-scale graphs. In *KDD* (2015).

108. Hancioglu, B., Swigon, D., and Clermont, G. A dynamical model of human immune response to influenza A virus infection. *Journal of Theoretical Biology 246*, 1 (2007), 70–86.

109. Hinkelmann, F., Brandon, M., Guang, B., McNeill, R., Blekherman, G., Veliz-Cuba, A., and Laubenbacher, R. Adam: Analysis of discrete models of biological systems using computer algebra. *BMC Bioinformatics 12* (2011), 1–11.

110. Hoffman, B. R., Sussman, S., Unger, J. B., and Valente, T. W. Peer influences on adolescent cigarette smoking: A theoretical review of the literature. *Substance Use and Misuse 41* (2006), 103–155.

111. Hogue, A., and Steinberg, L. Homophily of internalized distress in adolescent peer groups. *Developmental psychology 31*, 6 (1995), 897.

112. Hornbæk, K., Sander, S. S., Bargas-Avila, J. A., and Grue Simonsen, J. Is once enough?: on the extent and content of replications in human-computer interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2014), ACM, pp. 3523–3532.

113. Hsee, C. K., Hatfield, E., and Chemtob, C. Assessments of the emotional states of others: Conscious judgments versus emotional contagion. *Journal of social and clinical psychology 11*, 2 (1992), 119–128.

114. Hunt, J., and Eisenberg, D. Mental health problems and help-seeking behavior among college students. *Journal of Adolescent Health 46*, 1 (2010), 3–10.

115. Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering 9*, 3 (2007), 90–95.

116. Ioannidis, J. P., Allison, D. B., Ball, C. A., Coulibaly, I., Cui, X., Culhane, A. C., Falchi, M., Furlanello, C., Game, L., Jurman, G., Mangion, J., Mehta, T., Nitzberg, M., Page, G. P., Petretto, E., and van Noort, V. Repeatability of published microarray gene expression analyses. *Nature genetics 41*, 2 (2009), 149–155.

117. Istrate, G., Marathe, M. V., and Ravi, S. Adversarial models in evolutionary game dynamics. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms* (2001), Society for Industrial and Applied Mathematics, pp. 719–720.

118. Istrate, G., Marathe, M. V., and Ravi, S. Adversarial scheduling in discrete models of social dynamics. *Mathematical Structures in Computer Science 22*, 05 (2012), 788–815.

119. Jones, M. B., and Jones, D. R. Preferred pathways of behavioral contagion. *Journal of psychiatric research 29*, 3 (1995), 193–209.

120. Karaoz, U., Murali, T., Letovsky, S., Zheng, Y., Ding, C., Cantor, C., and Kasif, S. Whole-genome annotation by using evidence integration in functional-linkage networks. *Proceedings of the National Academy of Sciences 101*, 9 (2004), 2888–2893.

121. Keith, T. Depression and its negative effect on college students. *Undergraduate Research Journal for the Human Sciences 9*, 1 (2010).

122. Kempe, D., Kleinberg, J., and Tardos, E. Maximizing the spread of influence through a social network. In *KDD* (2003).

123. Kendler, K. S., and Prescott, C. A. Genes, environment, and psychopathology. *New York: Guilford* (2006).

124. Kerckhoff, A. C., and Back, K. W. *The June bug: A study of hysterical contagion*. Appleton-Century-Crofts, 1968.

125. Kiuru, N., Burk, W. J., Laursen, B., Nurmi, J., and Salmela-Aro, K. Is depression contagious? a test of alternative peer socialization mechanisms of depressive symptoms in adolescent peer networks. *Journal of Adolescent Health 50*, 3 (2012), 250–255.

126. Kramer, A. D. I., Guillory, J. E., and Hancock, J. T. Experimental evidence of massive-scale emotional contagion through social networks. *Proceedings of the National Academy of Sciences 111*, 24 (2014), 8788–8790.

127. Kuhlman, C. J. *High Performance Computational Social Science Modeling of Networked Populations*. PhD thesis, Virginia Tech, 2013.

128. Kuhlman, C. J., Kumar, V., Marathe, M. V., Mortveit, H. S., Swarup, S., Tuli, G., Ravi, S., and Rosenkrantz, D. J. A general-purpose graph dynamical system modeling framework. In *Proceedings of the Winter Simulation Conference* (2011), Winter Simulation Conference, pp. 296–308.

129. Kuhlman, C. J., Kumar, V. A., Marathe, M. V., Ravi, S., and Rosenkrantz, D. J. Inhibiting diffusion of complex contagions in social networks: theoretical and experimental results. *Data mining and knowledge discovery 29*, 2 (2015), 423–465.

130. Kuhlman, C. J., Kumar, V. S. A., Marathe, M. V., Ravi, S. S., Rosenkrantz, D. J., Swarup, S., and Tuli, G. Inhibiting the Diffusion of Contagions in Bi-Threshold Systems: Analytical and Experimental Results. In *Proceedings of the AAAI Fall 2011 Symposium on Complex Adaptive Systems (CAS-AAAI 2011)* (November 2011), pp. 91–100.

131. Kuhlman, C. J., Mortveit, H. S., Murrugarra, D., and Kumar, V. S. A. Bifurcations in Boolean networks. *Discrete Mathematics and Theoretical Computer Science* (2011), 29–46.

132. Kulenovic, M. R., and Merino, O. *Discrete Dynamical Systems and Difference Equations with Mathematica*. Chapman-Hall, 2002.

133. Kules, B., Kang, H., Plaisant, C., Rose, A., and Shneiderman, B. Immediate usability: a case study of public access design for a community photo library. *Interacting with Computers 16* (2004).

134. Larsen, M. R., Sommersel, H. B., and Larsen, M. S. *Evidence on Dropout Phenomena at Universities*. Danish Clearinghouse for educational research, 2013.

135. LeBold, W. K. Research in engineering education: An overview. *Engineering Education 70*, 5 (1980), 406.

136. LeBold, W. K., Delauretis, R., and Shell, K. D. The purdue interest questionnaire: An interest inventory to assist engineer students in planning their career. *annual Frontiers in Education Conference* (1977).

137. Lee, E. A., and Zheng, H. Operational semantics of hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control* (2005), pp. 25–53.

138. Legrand, J., Grais, R. F., Boelle, P. Y., Valleron, A. J., and Flahault, A. Understanding the dynamics of ebola epidemics. *Epidemiology and Infection 135* (2007).

139. Leser, U. A query language for biological networks. *Bioinformatics 21*, suppl 2 (2005), ii33–ii39.

140. Leskovec, J. Stanford Network Analysis Project. http://snap.stanford.edu/. [Online; accessed 1-May-2014].

141. Levy, D., de Almeida, L. M., and Szklo, A. The brazil simsmoke policy simulation model: The effect of strong tobacco control policies on smoking prevalence and smoking-attributable deaths in a middle income nation. *Plos One* (2012).

142. Lewis, J. R., and Sauro, J. The factor structure of the system usability scale. In *International Conference on Human Centered Design* (2009), Springer, pp. 94–103.

143. Lim, C., Lu, S., Chebotko, A., and Fotouh, F. Storing, reasoning, and querying OPM-compliant scientific workflow provenance using relational databases. *Future Generation Computer Systems 27* (2011), 781–789.

144. Loshbaugh, H. G., Hoeglund, T., Streveler, R., and Breaux, K. Engineering school, life balance, and the student experience. In *Proceedings of the American Society for Engineering Education Annual Conference, Chicago, IL* (2006).

145. Lu, J., and Churchill, D. The effect of social interaction on learning engagement in a social networking environment. *Interactive learning environments 22*, 4 (2014), 401–417.

146. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., and Zhao, Y. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper. 18*, 10 (Aug. 2006), 1039–1065.

147. Ma, J., Cao, J., and Zhang, Y. Efficiently supporting secure and reliable collaboration in scientific workflows. *J. Comput. Syst. Sci. 76*, 6 (Sept. 2010), 475–489.

148. Macal, C. M., and North, M. J. Tutorial on agent-based modelling and simulation. *J. Simulation 4* (2010), 151–162.

149. Macauley, M., and Mortveit, H. Cycle equivalence of graph dynamical systems. *Nonlinearity 22*, 2 (2009), 421–436.

150. Majithia, S., Shields, M., Taylor, I., and Wang, I. Triana: a graphical web service composition and execution toolkit. In *Proceedings. IEEE International Conference on Web Services* (July 2004), pp. 514–521.

151. Mayer, S. E. How much does a high school's racial and socioeconomic mix affect graduation and teenage fertility rates? In *The Urban Underclass*, C. Jencks and P. E. Peterson, Eds. The Brookings Institution, 1991.

152. Melnik, S., Ward, J. A., Gleeson, J. P., and Porter, M. A. Multi-Stage Complex Contagions, 2013.

153. Mendoza, L., and Alvarez-Buylla, E. R. Dynamics of the genetic regulatory network forarabidopsis thalianaflower morphogenesis. *Journal of theoretical biology 193*, 2 (1998), 307–319.

154. Mike Holcombe, et. al. Large-scale modeling of economic systems. *Complex Systems 22* (2013), 175–191.

155. Mortveit, H., and Reidys, C. *An Introduction to Sequential Dynamical Systems*. Springer, New York, NY, 2007.

156. Open Science Collaboration. Estimating the reproducibility of psychological science. *Science 349*, 6251 (2015), aac4716.

157. Perumalla, K. S. μsik–a micro-kernel for parallel/ distributed simulation systems. In *PADS* (2005).

158. Perumalla, K. S., and Park, A. J. Simulating billion-task parallel programs. In *Summer Sim* (2014).

159. Prabhu, P., Jablin, T. B., Raman, A., Zhang, Y., Huang, J., Kim, H., Johnson, N. P., Liu, F., Ghosh, S., Beard, S., Oh, T., Zoufaly, M., Walker, D., and August, D. I. A survey of the practice of computational science. In *State of the Practice Reports* (New York, NY, USA, 2011), SC '11, ACM, pp. 19:1–19:12.

160. Prescott, C. A., Aggen, S. H., and Kendler, K. S. Sex-specific genetic influences on the comorbidity of alcoholism and major depression in a population-based sample of us twins. *Archives of General Psychiatry 57*, 8 (2000), 803–811.

161. Prinstein, M. J. Moderators of peer contagion: A longitudinal examination of depression socialization between adolescents and their best friends. *Journal of Clinical Child and Adolescent Psychology 36*, 2 (2007), 159–170.

162. Prud'hommeaux, E., and Seaborne, A. Sparql query language for rdf. *W3C recommendation 15* (2008).

163. Ptolemaeus, C. *System Design, Modeling, and Simulation using Ptolemy II*. lulu.com, 2013.

164. Puzis, R., Tubi, M., Elovici, Y., Glezer, C., and Dolev, S. A Decision Support System for Placement of Intrusion Detection and Prevention Devices in Large-Scale Networks. *ACM Transactions on Modeling and Computer Simulation 22* (2011), 1–2.

165. Pyari, D. A comparitive study of depression and anxeity among medical and engineering students. *American research thoughts 1*, 9 (7 2015), 12.

166. Rachapalli, J., Khadilkar, V., Kantarcioglu, M., and Thuraisingham, B. RETRO: A framework for semantics preserving SQL-to-SPARQL translation. In *Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn)* (2011).

167. Raunic, A., and Xenos, S. University counselling service utilisation by local and international students and user characteristics: A review. *International Journal for the Advancement of Counselling 30*, 4 (2008), 262–267.

168. Reluga, T. C., Medlock, J., and Perelson, A. S. Backward bifurcations and multiple equilibria in epidemic models with structured immunity. *Journal of Theoretical Biology 252* (2008), 155–165.

169. Reynolds, A. D., and Crea, T. M. Peer influence processes for youth delinquency and depression. *Journal of adolescence 43* (2015), 83–95.

170. Richard Beckman, et. al. ISIS: A networked- epidemiology based pervasive web app for infectious disease pandemic planning and response. In *KDD* (2014).

171. Rifaieh, R., Unwin, R., Carver, J., and Miller, M. A. SWAMI: Integrating biological databases and analysis tools within user friendly environment. In *4th International Workshop on Data Integration in the Life Sciences (DILS)* (2007).

172. Rivers, C. M., Lofgren, E. T., Marathe, M. V., Eubank, S., and Lewis, B. L. Modeling the impact of interventions on an epidemic of Ebola in Sierra Leone and Liberia. *Plos Currents Outbreaks* (2014).

173. Rizwan, A., Farooq, S., Alvi, M. S. I., and Nawaz, S. Analysis of factors affecting the stress level of female engineering students. *Global Journal of Human-Social Science Research 12*, 10-A (2012).

174. Romano, P., Bartocci, E., Bertolini, G., Paoli, F. D., Marra, D., Mauri, G., Merelli, E., and Milanesi, L. Biowep: a workflow enactment portal for bioinformatics applications. *BMC Bioinformatics 8*, S-1 (2007).

175. Romero, D., Meeder, B., and Kleinberg, J. Differences in the Mechanics of Information Diffusion Across Topics: Idioms, Political Hashtags, and Complex Contagion on Twitter. In *WWW* (2011).

176. Rosenkrantz, D. J., Marathe, M. V., Hunt, H. B., Ravi, S., and Stearns, R. E. Analysis problems for graphical dynamical systems: A unified approach through graph predicates. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems* (2015), International Foundation for Autonomous Agents and Multiagent Systems, pp. 1501–1509.

177. Ruz, G. A., and Goles, E. Reconstruction and update robustness of the mammalian cell cycle network. In *Comptuational Intelligence in Bioinformatics and Computational Biology (CIBCB 2012)* (2012), pp. 397–403.

178. Santos, E., Koop, D., Vo, H. T., Anderson, E., Freire, J., and Silva, C. T. Using workflow medleys to streamline exploratory tasks. In *21st International Conference on Scientific and Statistical Database Management (SSDBM* (2009), pp. 292–301.

179. Sarwat, M., Elnikety, S., He, Y., and Kliot, G. Horton: Online query execution engine for large distributed graphs. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on* (2012), IEEE, pp. 1289–1292.

180. Sauro, J. Measuring usability with the system usability scale (sus). 2011. [Online; accessed 27-October-2016].

181. Sauro, J., and Lewis, J. R. *Quantifying the user experience: Practical statistics for user research*. Elsevier, 2012.

182. Scheidegger, C. E., Koop, D., Freire, J., and Silva, C. T. Querying and re-using workflows with VisTrails. In *ACM International Conference on Management of Data (SIGMOD)* (2008), pp. 1251–1254.

183. Schelling, T. *Micromotives and Macrobehavior*. W. W. Norton and Company, 1978.

184. Schmittmann, V. D., Cramer, A. O. J., Waldorp, L. J., Epskamp, S., Kievit, R. A., and Borsboom, D. Deconstructing the construct: A network perspective on psychological phenomena. *New Ideas in Psychology 31*, 1 (2013), 43–53.

185. Schneider, L. Perceived stress among engineering students. In *St, Lawrence Section Conference, Toronto, Canada* (2007).

186. Schneider, M., and Yin, L. The high cost of low graduation rates: How much does dropping out of college really cost? *American Institutes for Research* (2011).

187. Schneiderman, B., and Plaisant, C. *Designing the User Interface*. Pearson, 2010.

188. Sherman, S. G., Ganna, D. S., Tobin, K. E., Latkin, C. A., Welsh, C., and Bielenson, P. The life they save may be mine: Diffusion of overdose prevention information from a city sponsored programme. *International Journal of Drug Policy 20* (2009), 137–142.

189. Shmulevich, I. *PBN*. 2014.

190. Shneiderman, B. Creativity support tools: Accelerating discovery and innovation. *Communications of the ACM 50* (2007), 20–32.

191. Siegel, D. Social networks and collective action. *Americal Journal of Political Science 53* (2009), 122–138.

192. Sorns, O. Making sense of brain network data. *Nature Methods* (2013).

193. Spady, W. G. Dropouts from higher education: An interdisciplinary review and synthesis. *Interchange 1*, 1 (1970), 64–85.

194. Spasojevic, N., Yan, J., Rao, A., and Bhattacharyya, P. Lasta: Large scale topic assignment on multiple social networks. In *KDD* (2014).

195. Stevens, E. A., and Prinstein, M. J. Peer contagion of depressogenic attributional styles among adolescents: A longitudinal study. *Journal of Abnormal Child Psychology 33*, 1 (2005), 25–37.

196. Swail, W. S. The art of student retention: A handbook for practitioners and administrators. In *Educational Policy Institute. Texas Higher Education Coordinating Board 20th Annual Recruitment and Retention Conference Austin, TX June* (2004), vol. 21, p. 2004.

197. Taylor, I., Shields, M., Wang, I., and Harrison, A. The Triana workflow environment: Architecture and applications. *Workflows for e-Science* (2007), 320–339.

198. Teeni, D., Carey, J., and Zhang, P. *Human Computer Interaction*. Wiley, 2007.

199. Tenopir, C., Allard, S., Douglass, K., Aydinoglu, A. U., Wu, L., Read, E., Manoff, M., and Frame, M. Data sharing by scientists: practices and perceptions. *PloS one 6*, 6 (2011), e21101.

200. Tinto, V. Taking retention seriously: Rethinking the first year of college. *NACADA journal 19*, 2 (1999), 5–9.

201. Tinto, V. Research and practice of student retention: what next? *Journal of College Student Retention: Research, Theory & Practice 8*, 1 (2006), 1–19.

202. Tullis, T. S., and Stetson, J. N. A comparison of questionnaires for assessing website usability. In *Usability Professional Association Conference* (2004), Citeseer, pp. 1–12.

203. Ugander, J., Backstrom, L., Marlow, C., and Kleinberg, J. Structural diversity in social contagion. *Proceedings of the Naitonal Academy of Sciences (PNAS 2012) 109*, 9 (2012), 5962–5966.

204. Umar, S. S., Shaib, I. O., Aituisi, D. N., Yakubu, N. A., and Bada, O. The effect of social factors on students' academic performance in Nigerian tertiary institutions. *Library Philosophy and Practice (e-journal)* (2010), 334.

205. Van Borkulo, C. D., Van der Maas, H. L. J., Borsboomm, D., and Cramer, A. O. J. Netlogo vulnerability_to_depression, 2013.

206. Vitasari, P., Wahab, M. N. A., Othman, A., Herawan, T., and Sinnadurai, S. K. The relationship between study anxiety and academic performance among engineering students. *Procedia-Social and Behavioral Sciences 8* (2010), 490–497.

207. von Dawans, B. *Neuropeptidergic modulation of social behavior in health and social phobia*. Cuvillier Verlag, 2008.

208. von Landesberger, T., Diel, S., Bremm, S., and Fellner, D. W. Visual analysis of contagion in networks. *Information Visualization* (2013).

209. W. Creswell, J. *Research design: Qualitative, quantitative, and mixed methods approaches*. SAGE Publications, Incorporated, 2009.

210. Wang, C., Feng, K., Shen, H., and Lee, T. Coherent time-varying graph drawing with multifocus+context interaction. *IEEE Transactions on Visualization and Computer Graphics 18*, 8 (2012), 1330–1342.

211. Wang, S., Hu, X., Yu, P. S., and Li, Z. Mmrate: Inferring multi-aspect diffusion networks with multi-pattern cascades. In *KDD* (2014).

212. Watts, D. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences (PNAS) 99* (2002), 5766–5771.

213. Watts, D., and Dodds, P. Threshold models of social influence. In *The Oxford Handbook of Analytical Sociology* (2009), pp. 475–497.

214. Wellman, M. P. Putting the agent in agent-based modeling. *Journal of Autonomous Agents and Multiagent Systems (to appear)* (2015).

215. Wilson, G. V. Where's the real bottleneck in scientific computing? *American Scientist 94*, 1 (2006), 5.

216. Wilson, S. Riot anniversary tour surveys progress and economic challenges in Los Angeles. In *Cable New Network (CNN)*. 2012.

217. Wolstencroft, K., Haines, R., Fellows, D., Williams, A., et al. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research* (2013), W557–W561.

218. wook Ahn, J., Taieb-Maimon, M., Sopan, A., Plaisant, C., and Shneiderman, B. Temporal visualization of social network dynamics: Prototypes for nation of neighbors. In *SBP 2011* (2004), pp. 309–316.

219. Wuensche, A. *DDLab*. 2014.

220. Yadlin, A., Eran, O., Stein, S., Magen, Z., Schweitzer, Y., Brom, S., Kam, E., Guzansky, Y., Berti, B., Lindenstrauss, G., Friedman, D., Asculai, E., Landau, E. B., Kurz, A., Siboni, G., and Heller, M. One year of the arab spring: Global and regional implications. Tech. rep., The Institute for National Security Studies.

221. Yoginath, S. B., and Perumalla, K. S. Efficient parallel discrete event simulation on cloud/virtual machine platforms. *ACM TOMACS (under review)* (2014).

222. Yorgason, J. B., Linville, D., and Zitzman, B. Mental health among college students: do those who need services know about and use them? *Journal of American College Health 57*, 2 (2008), 173–182.

223. Zimmerman, B., and Schunk, D. H. Competence and control beliefs: Distinguishing the means and ends. *Handbook of educational psychology* (2006), 349–367.