Proceedings of the 2011 Winter Simulation Conference
S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, eds.

A GENERAL-PURPOSE GRAPH DYNAMICAL SYSTEM MODELING FRAMEWORK

Chris J. Kuhlman
V. S. Anil Kumar
Madhav V. Marathe
Henning S. Mortveit
Samarth Swarup
Gaurav Tuli

Virginia Bioinformatics Institute, Virginia Tech
Blacksburg, VA 24061, USA

S. S. Ravi
Daniel J. Rosenkrantz

University of Albany—SUNY
Department of Computer Science
Albany, NY, 12222, USA

ABSTRACT

We describe InterSim, a general purpose flexible framework for simulating graph dynamical systems (GDS) and their generalizations. GDS provide a powerful formalism to model and analyze agent-based systems (ABS) because there is a direct mapping between nodes and edges (which denote interactions) in a GDS and agents and interactions in an ABS, thereby providing InterSim with great expressive power. We describe the design, implementation, capabilities, and features of InterSim; e.g., it enables users to quickly produce simulations of ABS in many application domains. We present illustrative case studies that focus on the simulation of social phenomena. InterSim has been used to simulate networks with 4 million agents and to execute large parametric simulation studies.

1 INTRODUCTION

A large number of complex systems such as those occurring in sociology, epidemiology, finance, biology, and statistical physics (e.g., (Karaoz et al. 2004, Centola and Macy 2007, Perumalla and Seal 2010)) can be modeled as graph dynamical systems (GDS) (Mortveit and Reidys 2007), which generalize a host of commonly studied models, such as cellular automata. Such systems are extremely hard to study purely analytically, and large-scale agent based simulations (ABS) are increasingly becoming fundamental tools in their study; e.g., (Epstein 2007, Gilbert 2007). Simulation-based studies are computationally expensive, since they involve complex and adaptive behaviors and a large parameter space (e.g., (Macy and Willer 2002, Hollander and Wu 2011)). For specific domains, such as epidemiology, there has been a lot of research devoted to developing highly tuned tools which scale to very large populations, e.g., (Bisset et al. 2009, Perumalla and Seal 2010). Such tools crucially exploit properties of the specific processes being modeled, e.g., the "susceptible-exposed-infectious-recovered" (SEIR) model in the case of epidemics, and it is not easy to adapt such tools to incorporate models in other applications. Further, in most realistic scenarios, distributed implementations are needed in order to scale to moderate or large instances and to execute large experimental designs.

There has also been a lot of research on developing more general discrete event frameworks that allow users to incorporate a wide variety of models, such as AnyLogic, BRACE, NetLogo, Repast and Repast SC++, SASSY, and Swarm; e.g., (Hybinette et al. 2006, North and Macal 2009, Wang et al. 2010). See also (Macal and North 2010) for a survey and (Aaby et al. 2010) for additional frameworks. The goal in such approaches is to provide a flexible framework for researchers who may not be computer programming experts. Such frameworks have proved to be very popular in the agent-based modeling and simulation (ABMS) community, and have large user populations.

Application contexts such as social interactions, epidemiology, (wireless) network communications, and biology have high levels of interactions among agents, making discrete *time* simulation—where agent interactions take place at every time step—applicable to many domains. Our work with discrete time and discrete event epidemiology simulators, for example, clearly demonstrates that discrete time simulations are much faster for specific applications because they do not incur the overhead of generating schedules and coordinating events of agents that might involve synchronization, particularly over multiple processing elements in distributed simulations for large problem sizes. Elegant solutions to distributed coordination can be classified as conservative (Chandy and Misra 1979), and anti-conservative (rollback-based) (Jefferson 1985); other studies that build on these works are compiled in (Carothers and Perumalla 2010). These issues are bypassed with discrete time simulation. A downside to discrete time simulation is that compute cycles are wasted when agents have no inputs at a particular time, but at least in some cases, these wasted cycles may be less costly than the overheads associated with discrete event simulation. Also, trends in current computer architectures toward many-core and GPU suggest that the effects of wasted cycles will diminish compared to communication costs on commodity clusters.

The goal of this paper is to describe and demonstrate InterSim, a new distributed tool for discrete time, high interaction ABS that uses a GDS-based formalization. InterSim provides a simple and flexible framework to capture any GDS, and is implemented using the Message Passing Interface (MPI). Our specific contributions include the following.

1. *General framework for simulating GDS.* InterSim implements the complete class of GDS and their generalizations; these include general kinds of (vector valued) update functions, interaction networks, update orders, and finite state machines (FSM) that describe state transitions (GDS and a variant are described later). The generality of GDS is reflected in the fact that it is a universal model of computation. Additionally, InterSim allows complex coupling of multiple systems, such as multiple networks and dynamics, in an easy manner. It uses "node interaction models" (NIM), which capture agent behaviors. The distributed processing aspects of the framework are completely transparent to the user, who only needs to be able to program the NIM serially using a well-defined interface. InterSim has been used to evaluate 4.8 million agent networks on a moderate sized cluster. A novel feature of our approach is the ability to use GDS theory to reason about system dynamics. We know of no other simulation framework that is explicitly theoretically grounded in GDS.
2. *Flexibility and reduced "end-to-end turnaround time."* Most studies of complex systems require a large experimental design, with a large number of simulation runs, in order to explore the whole parameter space. InterSim is flexible and easy to use. It is designed to optimize not only the time for each run, but also the turn-around time, which is defined as the time from problem definition to the time at which useful simulation results are generated—this includes the sum of software design, implementation, and verification times. We have executed parametric studies with thousands of parameter sets on 100000-agent systems using a 96-node commodity cluster (2 processors/node; 4 cores/processor) with 3 GHz Intel Xeon cores and 2 MB memory per core. This system is shared among many users and typically concurrent simulations within a given study use 2 nodes each. For very large networks, we may use 10 compute nodes. We realize turn around times on the order of 1 to 4 hours for more straight-forward models and total solution times on the order of a week of wall clock time. An immediate benefit of the flexibility and fast turnaround time of InterSim is that different kinds of complex dynamics models can be prototyped very quickly.
3. *Integration with a simulation steering tool.* InterSim has been specifically designed to integrate with Indemics (Bisset et al. 2010). Indemics is another framework that permits high performance computing in data-intensive simulations. A combination of these two frameworks leads to even greater flexibility in simulations; e.g., the ability for users to steer (or modify) simulations on-the-fly. The feasibility of this coupling has been demonstrated with an application-specific simulation code. InterSim integration with Indemics is a topic for future work and is not discussed further here.

4. *Illustrative case studies*. We demonstrate the utility of InterSim by means of two case studies that illustrate selected types of models, networks, and behaviors that can be captured using NIM. The case studies show how multiple networks, and increasingly complex and adaptive behavioral models for agents can be incorporated. We show that models can be implemented in InterSim very easily, and discuss some sample results on the system dynamics.

In discussing the Open Services Gateway Initiative Framework (OSGi) and Cyberinfrastructure Shell (CiShell), (Borner 2011) mentions 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 also serves as a primary driver of our own work in the context of ABMS.

Organization. We discuss the GDS model in Section 2 and the high level architecture of InterSim in Section 3. In Section 4, we describe an implemented threshold-based dynamics model that is composed with InterSim. Illustrative case studies are discussed in Section 5. We conclude with Section 6.

2 EVOLVING GRAPH DYNAMICAL SYSTEMS: THEORETICAL FOUNDATIONS

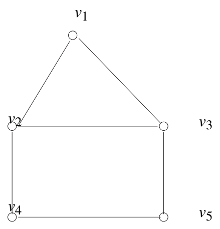
A **Graph Dynamical System** (GDS) is an abstract representation of a group of interacting entities (agents) and the nature of their interactions. This representation provides a sound basis to develop simulations of diffusion processes in such systems where the graph may evolve in time. Our formulation is contagion-centric, where a **contagion** is any entity, such as an opinion or virus, that can propagate through a system.

An **Evolving GDS** (EGDS) is given by $\mathscr{S}(\mathscr{C},\mathscr{X},\mathscr{K},\mathscr{J},\mathscr{F},\mathscr{D},\mathscr{R})$ where $\mathscr{C}$ is a set of contagions, $\mathscr{X}$ is a family of graphs, $\mathscr{K}$ is a family of vertex state spaces, $\mathscr{J}$ is a family of edge state spaces, $\mathscr{F}$ is a family of $\mathscr{X}$-local functions for vertices, $\mathscr{D}$ is a family of $\mathscr{X}$-local functions for edges, and $\mathscr{R}$ is a family of update sequences. For each contagion $c_j \in \mathscr{C}$, there is a time-evolving graph $X_j(V_j,E_j) \in \mathscr{X}$ over which c_j propagates (V_j is the vertex set and E_j is the edge set of the graph). Also for each c_j, there is a vertex state space $K_j \in \mathscr{K}$, an edge state space $J_j \in \mathscr{J}$, an X_j-local (vertex) function $F_j \in \mathscr{F}$, an X_j-local (edge) function $D_j \in \mathscr{D}$, and an update scheme $R_j \in \mathscr{R}$. Furthermore, for each vertex $v_i \in V_j$, there exists a **local (vertex) transition function** $f_i(S(v_i),S(n[v_i]),S(e[v_i])) \in F_j$ which determines the next state $s(v_i) \in K_j$ of the vertex for contagion c_j. Here, $S(v_i)$ is the entire state of v_i over all contagions, $S(n[v_i])$ is the entire state of all neighbors $n[v_i]$ of v_i, and $S(e[v_i])$ is the entire state of all edges $e[v_i]$ adjacent to v_i over all contagions. For each edge $e_k \in E_j$, there exists **local (edge) transition function** $d_k(S(e_k),S(n[e_k]),S(e[e_k])) \in D_j$ which determines the next state $s(e_k) \in J_j$ of e_k for contagion c_j. Here, $S(e_k)$ is the entire state of e_k over all contagions, $S(n[e_k])$ is the entire state of the two vertices incident on e_k, and $S(e[e_k])$ is the entire state over all contagions of all other edges $e[e_k]$ incident on either of the two vertices. The update scheme R_j determines the order in which f_i and d_k are executed, both within and across contagions. The contagion-centric formulation for EGDS $\mathscr{S}$ provides a natural framework to investigate the simultaneous diffusion of multiple contagions within ABMS. At any time l, the **configuration** $C(l)$ of an EGDS is a vector $(S_l(v_1),\ldots,S_l(v_n),S_l(e_1),\ldots,S_l(e_m))$, where n and m are the total number of nodes and edges, respectively, in the system. The time evolution of an EGDS is represented by the sequence of successive configurations of the EGDS.

Example: We now present a simple example with one contagion, only vertex states, and a static graph to illustrate the ideas mentioned above. The undirected graph $G(V,E)$ for this EGDS is shown in Figure 1. The state of each node is assumed to be from $K = \{0,1\}$, where 0 (1) indicates that the contagion has not (has) propagated to the node. For each node $v_i \in V$the local interaction function f_i, which depends on the current states of v_i and the neighbors of v_i, is defined as follows.

(a) If the current state of v_i is 1, then the value of f_i is 1 regardless of the states of the neighbors of v_i.

(b) If the current state of v_i is 0, then the value of f_i is 1 if at least one of the neighbors of v_i is in state 1; otherwise, the value of f_i is 0.



Note: Each configuration has the form $(s(1), s(2), s(3), s(4), s(5))$, where $s(v_i)$ is the state of node v_i, $1 \leq i \leq 5$. The sequence of configurations of the system are shown below. After time step 2, the system remains in the configurations $(1,1,1,1,1)$.

Initial Configuration:	(1, 0, 0, 0, 0)
Configuration at time 1:	(1, 1, 1, 0, 0)
Configuration at time 2:	(1, 1, 1, 1, 1)

Figure 1: An example of a synchronous Graph Dynamical System (GDS).

The above local transition function belongs to the general class of **simple threshold** functions. Such a function is characterized by the specification of a threshold value for each node; the value represents the *minimum* number of neighbors needed for the state of a node to change from 0 to 1. In this example, the threshold value for each node is 1. From the definition of local vertex transition functions, it is seen that once a node reaches the state of 1, it remains in that state for ever. In this illustration, the update scheme is assumed to be *synchronous*. In other words, all the local transition functions are computed in *parallel* using the current values of the inputs. A sequence of configurations for this system is also shown in Figure 1.

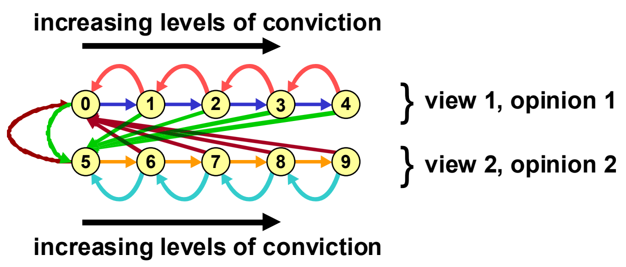
To make the EGDS formulation more concrete, we provide illustrative model features in Figure 2, with additional discussion in (Bisset et al. 2011). Currently, some features associated with $\mathscr{D}$ are rudimentary.

1. **Graphs**: (a) Directed (to capture *asymmetric* interactions) and undirected; (b) Time-varying multiple networks, with a different network for each contagion.
2. **Extended set of states for nodes and edges**: (a) A finite state machine (FSM) for each contagion and node, representing the set of state transitions for a node (e.g., Figure 3); (b) States for edges (e.g., time labels to indicate the time intervals for which an edge exists); (c) Multi-dimensional states for nodes and edges for each contagion.
3. **Generalized local transition functions**: Each local function f_i at node v_i may be Markovian or depend on the history of the state values during the last T time steps. Such a function has the general form

$$s_{l+1}(v_i) = f_i(S_\tau(v_i), S_\tau(n[v_i]), S_\tau(e[v_i])), \qquad l - T + 1 \leq \tau \leq l$$

   where (as before) $S_\tau(v_i)$ is the total state over all contagions of node v_i at time τ, $S_\tau(n[v_i])$ is the total states of all neighbors of v_i at time τ, $S_\tau(e[v_i])$ is the total state of all incident edges of v_i at time τ. The local functions may be multi-valued or stochastic.
4. **Different state update schemes**: (a) Synchronous, sequential ordering based on a permutation or a word (Mortveit and Reidys 2007); (b) Hybrid (which combines the synchronous and sequential models).

Figure 2: Evolving Graph Dynamical Systems.


increasing levels of conviction
0
1
2
3
4
5
6
7
8
9
view 1, opinion 1
view 2, opinion 2
increasing levels of conviction

Note: Arrows represent state transitions and different arrow colors indicate different mechanisms giving rise to the state changes. Within each view, agents can change to stronger and weaker levels of conviction. If an agent changes from one view to another, the transition goes to the lowest conviction state. This FSM is used in case study 2 to model the evolution of opinions and ideology, and hence consensus-building.

Figure 3: Example FSM including multiple opinions and levels of conviction.

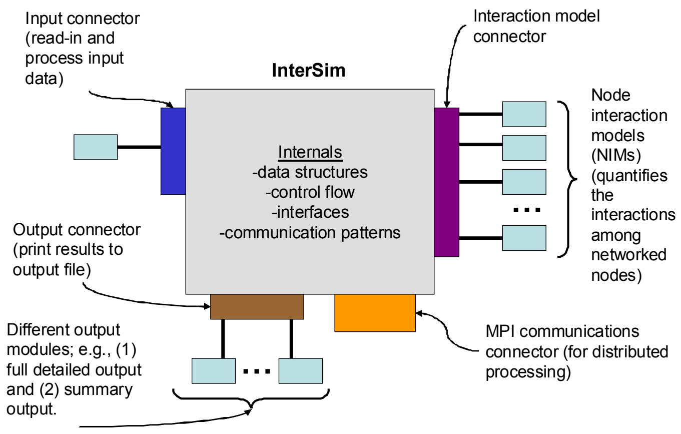

Input connector (read-in and process input data)
InterSim
Interaction model connector
Internals
-data structures
-control flow
-interfaces
-communication patterns
Node interaction models (NIMs) (quantifies the interactions among networked nodes)
Output connector (print results to output file)
Different output modules; e.g., (1) full detailed output and (2) summary output.
MPI communications connector (for distributed processing)

Figure 4: Schematic of an InterSim MPI process executing on a processing element.

Algorithm 1: Simulation

```
input : Graph(s) $G_i$, agent properties $v_P$, edge
        properties $e_P$, number of iterations $n_i$,
        maximum time per iteration $l_{max}$, base
        conditions $B_0$, NIM identifier $NIM$, NIM
        properties $NIM_P$, seed conditions $I$.
output: Changes in agent states $S$ as a function of time
        $l$.

Read in all Inputs: $G_i$, $v_P$, $e_P$, $NIM$ and $NIM_P$, $n_i$,
$l_{max}$, $B_0$.
for (each iteration $j$) do
    Read (initial) seed conditions $I_j$.
1   Reset all properties and NIM
    parameters to their base
    conditions.
    for ($l = 0; l \leq l_{max}; ++l$) do
        Loop over the agents owned by this process.
        for ($v = 0; v \leq v_{maxowned}; ++v$) do
2           Invoke the NIM for agent $v$.
3           if (state of $v$ changes) then Reset model
            parameters of new state for
            agent $v$.
        Send state changes to output.
        Send state changes to other processes.
```

3 InterSim FRAMEWORK

To set up the framework description, we need some terminology. An **iteration** is a diffusion instance. Initial conditions at time $l = 0$ are specified for the network (e.g., agent initial states) and for NIM properties. Network dynamics and structure evolve over time until the maximum time l_{max} is reached, at which point the iteration terminates. A **simulation** is a set of iterations. Possible variations across iterations include differences in initial agent states, evolutionary NIM parameters, interventions, and network structure.

3.1 Description

This high-level description of the InterSim simulation framework follows the theoretical description. InterSim is a distributed framework, with an instance running on each process element (e.g., CPU, core) of a cluster. MPI is used for communications. A schematic of an MPI process instance is given in Figure 4. InterSim is the gray box that contains control and data structures, and the set of *connectors* to node interaction models (NIM), input and output modules, and MPI. InterSim and the modules that plug into it (light blue rectangles) through the connectors combine to form the simulator. The input module is general and hence has only one implemented module. The MPI connector handles communications among the MPI processes along with marshalling and demarshalling of data.

NIM are written by the user and are compiled and linked to the infrastructure. NIM provide the node and edge behaviors for a particular application such as social behavior, statistical physics modeling, or communications networks. A NIM can also contain interventions. One NIM is instantiated for each system agent so that the NIM may evolve its own properties in an agent-specific fashion. A threshold-based NIM is described in Section 4. The framework and interaction model connector enable great flexibility and complexity in behaviors across applications; e.g., a NIM can solve a nonlinear system of equations $As = b$ for an agent state vector s at each time step. The output module connector enables different types of outputs. We have implemented output modules that print to file every state change, and that write to file only summary data. The latter module can reduce 5 to 10 GB output files to 100KB files, which is useful for large parametric studies where disk space becomes a limiting resource.

The gray box contains data structures for networks, agent and edge properties, and messaging. Each agent and edge of any network may have any number of integer, double, and character property values. Other primitive data types can be added in a straight-forward fashion. This enables elaborate descriptions of state and allows a user to specify any combination of parameters needed for a NIM that may evolve multiple contagions. Differences in property types across simulations are handled through input files; there are no code modifications. A subset of agents are *owned* by each MPI process; i.e., an MPI process is responsible for updating state of its owned agents and edges. Each MPI process provides the properties (including state) of its owned agents and all agents that influence its owned agents (i.e., its *pure influencers*).

InterSim controls overall execution. A high-level description is provided in the algorithm above, which is executed on all MPI processes. Inputs are read in at the beginning of a simulation. Seed values (explained in Section 5) are the only data read on a per iteration basis. The algorithm loops over iterations j, times l, and owned nodes v. The NIM for each agent is reset at the start of an iteration. At each time, the NIM for each agent v is invoked, wherein state changes are determined. If state changes, then the next state for v is reset; this is required for the case of cycles in the FSM. State changes are written to a file and they are shared among MPI processes so that each process' pure influencers have their current states for the next time step.

There is a well-defined interface for a NIM, given in Figure 5. The three methods are executed according to the corresponding identifiers in the algorithm, and are sufficiently general for any agent interaction model.

1. `resetForNewIteration()`: at the beginning of a new diffusion instance, reset local transition function state.
2. `computeNextState()`: at time l, compute the next state for agent v_j at time $(l+1)$.
3. `resetForNextState()`: at time l, if a state change was computed for agent v_j, reset parameters for the next state local transition function.

Figure 5: Programming interface to which NIM must conform.

3.2 Impacts

Here we tie InterSim features to the goals of Section 1. The GDS foundation enables one to reason about simulations using the body of proven results for these systems (e.g., (Mortveit and Reidys 2007)). The wide range in application domains and expressiveness in dynamics models are achieved because the framework is application agnostic, it automatically manages simulation parameters and provides them to a NIM for its use, and a NIM can implement any behavior consistent with the generic interface of Figure 5 while making use of any combination of simulation parameters (including agent and edge properties). The applicability of the infrastructure to a wide range of dynamical systems is addressed by the following lemma; the proof is omitted for space reasons.

Lemma 1 Given one or more agent-based contagion diffusion processes that can each be modeled as GDS as described in Section 2, the InterSim framework can simulate the system dynamics.

The same set of features enable fast turn around times and ease of use (note that one of these features does not imply the other). First, the infrastructure handles all distributed processing; the application programmer only needs to think *serially* in designing and constructing NIM software to capture the requisite dynamics. Second, the application programming interface (API) of Figure 5 provides structure for NIM. Third, the NIM of the system form a library and these can be used as starting points for new NIM development. Fourth, the framework and NIM interact in one place in the simulation code, so the programmer can focus on this one interface (in many codes, dynamics functionality is dispersed throughout the code). Fifth, there is a well-defined set of methods for retrieving agent and edge properties and for storing state updates. A number of test cases with system developers and external developers have demonstrated that turn around times from 1.5 to 4 hours can routinely be realized and that inexperienced programmers can write NIM and use the simulator.

4 ILLUSTRATION: IMPLEMENTING A THRESHOLD MODEL IN InterSim

While the infrastructure is agnostic to the form of local functions, threshold functions are attractive in social contexts because they capture influence of one agent on another; e.g., (Bischi and Merlone 2009). Furthermore, in (Watts 2002) it is argued that even when more intricate models of social interaction are "wired" into a person, threshold behavior may remain dominant for a number of reasons.

We describe features of the threshold model implementation. The case studies demonstrate the use of this model for multiple contagions and networks, and state transition mechanisms. There are two submodels

1. Constant contagion amount.
2. Contagion amount a function of edge weight.
3. Contagion amount selected from a uniform distribution with user specified limits.

Figure 6: Contagion amount models currently implemented. Additional models can be readily added.

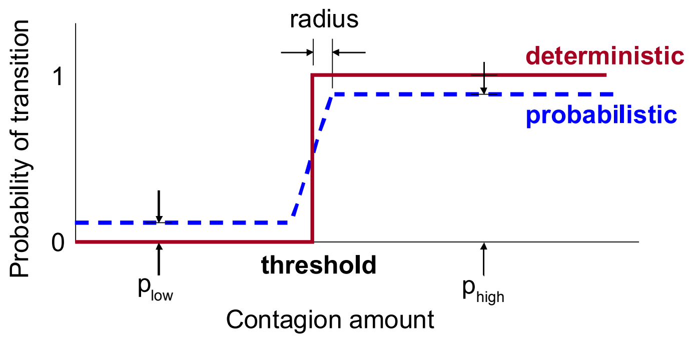


Figure 7: Schematic representation of state transition threshold model for an agent. The total amount of contagion is plotted against the probability of transition. The transition threshold and other parameters are inputs to the models.

of the threshold model: (i) a contagion amount (CA) model that quantifies the contagion passed in one interaction from one agent to another along their common edge and (ii) a state transition (ST) model that describes the criterion for an agent state change. Figure 6 displays three options for specifying contagion amount (through input file values). Parameters for a state transition model are given in Figure 7; these parameters permit deterministic and stochastic state transitions. All parameters can be specified as a function of state or on a per-agent basis, and can also be functions of global simulation parameters such as time.

In its simplest Markovian form, a state transition of agent v_i for c_j occurs when the contagion amounts a_j received from all neighbors v_k, denoted $n[v_i]$, exceed a threshold t at the current time; i.e., $\sum_{v_k \in n[v_i]} a_j \geq t$, where this equation is a threshold-based specialization of the equation in Figure 2. The CA and ST models combine to produce a model that extends a generalized contagion model (Dodds and Watts 2005).

5 CASE STUDIES

Two cases studies are presented that demonstrate features of the infrastructure, the types of NIM that can be plugged into it, and new insights into network dynamics. Some features are beyond those of Sections 2 and 3. *We emphasize that these examples serve to illustrate capabilities of the simulation system and describe general trends; to simulate these phenomena for a particular application, model parameters would be developed from data.* The first case study uses a simpler dynamics model and highlights the use of multiple networks and contagions, while the second utilizes a multi-mechanism NIM with a single contagion. We also focus on threshold functions for reasons mentioned in Section 4. We use one social network which we call `epinions`, obtained from Jure Leskovec's website at Stanford, a 75879-agent network of trust relationships between agents who trust one another's opinions, and rewirings of it.

5.1 Case Study 1

The first case study illustrates the use of multiple undirected networks in modeling people getting out of the stock market during a financial downturn. We investigate the effect on population dynamics of alterations in three networks over which different contagions spread, and of Markovian and history-dependent dynamics models. We show here novel results of how thresholds affect the probability of a cascade (i.e., the probability that most of the agents that can be "affected"—by leaving the stock market—are indeed affected) for multiple weighted networks, and how memory affects results. Previous work only examined a single contagion and unweighted network with no memory in the dynamics model (Centola and Macy 2007). The turn-around-times for the two NIM of this study were 1.5 hours and 2.5 hours.

There are three distinct networks which spread gossip, general news, and financial news, respectively, which are distinct contagions c_j. For example, the gossip network describes pairs of agents that share gossip. The great majority of agents ($\sim 99.5\%$) in each simulation are initially invested in the stock market and each form of information encourages an agent to get out of the market. Gossip, general news, and financial news contribute contagion amounts of $a_j = 1$, 2, and 3, respectively. When an agent receives a critical amount of information through a combination of these three sources in one day (i.e., when the sum of contagions exceeds a threshold t_T), it will cash out its stock portfolio. An agent still in the stock market will only accept information transmitted to it from neighboring agents that have already exited the stock market. The FSM in the NIM for this study is a ratchet-up or progressive model (i.e., the only state transition is from being invested in the stock market to being divested) and is of much interest to the sociology community (Granovetter 1978, Centola and Macy 2007).

The financial news network in all simulations is the `epinions` network. The other two networks are generated by *rewiring* edges of the `epinions` network. For each simulation, a rewiring probability r is specified. The gossip and general news networks are each generated by randomly rewiring a fraction r of edges (so all three networks are different). Here, rewiring means that an edge is unhooked from one agent and randomly assigned to a new agent; thus, the degrees of agents are different across networks. Values of r investigated are 0, 0.2, 0.5, and 0.8.

All agents are initially invested in the stock market except for a seed set of five agents; the seed agents are divested. For each of the 100 iterations of a simulation, a different set of seed agents is used. These same agent seed sets are used across the 104 simulations, for comparison purposes.

To streamline the presentation, we sum contagion amounts and thresholds over all three forms of information (contagions c_j), and an agent gets out of the stock market when the following condition is met: $c_T = \sum_{j=1}^{3} a_j \geq \sum_{j=1}^{3} t_j = t_T$. This criterion is evaluated at every time step. For NIM1, there is no memory; each invested agent is influenced by every divested neighboring agent on every day. For NIM2, there is history dependence: each divested agent only spreads information to its invested neighbors once—at the time of its divestiture—and the invested agent only remembers the information for two time steps.

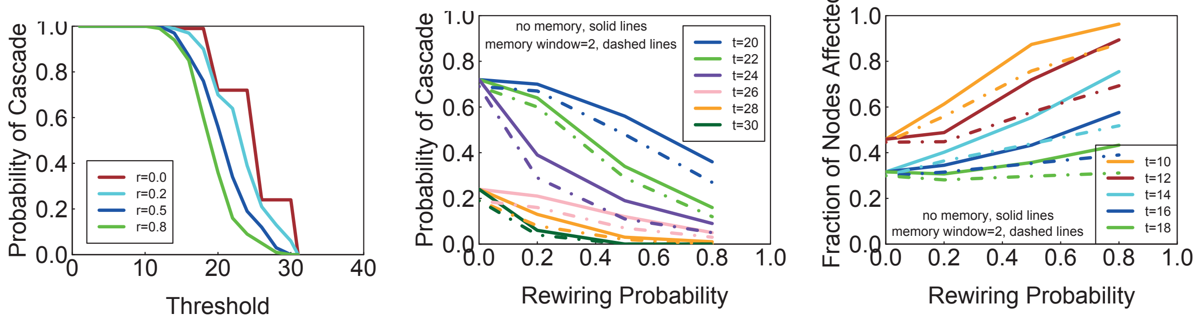


Figure 8: For the financial contagion, (a) probability of cascade as a function of threshold t_T for different rewiring probabilities r for NIM1; (b) probability of cascade as a function of rewiring probability for different thresholds; and (c) average cascade size—for iterations that cascade—as a function of rewiring probability for different thresholds. In (b) and (c), solid curves are results for NIM1; dashed curves are for NIM2. Note that these results also apply to probabilistic diffusion, where for example $p_{low} = 0$, $p_{high} = 0.5$ in Figure 7; simulations will merely take longer to reach the final states.

Figure 8 summarizes the results. Each curve represents the average results of 100 iterations. Figure 8(a) shows the probability of a cascade as a function of threshold for the four rewiring probability values using only NIM1. The curves show a relatively sharp transition from high probability to low probability as threshold increases from approximately $t_T = 17$ to 30. Figure 8(b) exchanges the roles of r and t_T to show more clearly the influence of r: as r increases, the cascade probability decreases. Also, results for NIM1 are the solid curves, while those for NIM2 are dashed. Figure 8(c) shows the cascade size—only for those iterations that cascade—as a function of r and threshold. Now, we see the opposite trend: as r increases, cascade size increases. In the latter two plots, the 2-step memory window decreases the probability of cascade and cascade size. To our knowledge, this is the first time multiple weighted networks and multiple (complex) contagions have been investigated with threshold models of diffusion.

5.2 Case Study 2

The second case study examines a single contagion, which can be thought of as the diffusion of opinions or ideologies (Centola and Macy 2007), dynamics of consensus building (Olfati-Saber et al. 2007), changes in degrees of human emotions (e.g., happiness and unhappiness (Fowler and Christakis 2008)), or willingness to join a rebellion (Granovetter 1978) within a group that could number hundreds or even millions of people. The novel aspect here is that three mechanisms for changing opinions are incorporated, and we show how different permutations (i.e., prioritizations) of these mechanisms lead to different network dynamics. This feature extends generalized contagion concepts (Dodds and Watts 2005) and to our knowledge, this is the first time different mechanisms have been used as the source of variation in state transitions. In addition, we run multiple sets of simulations to show the effects on network dynamics of dynamics properties, probabilistic state transitions, and interventions.

The FSM of Figure 3 is used to model the dynamics of a population. States 0 through 4 represent one view, with the strength of conviction increasing as states progress from 0 to 4. States 5 through 9 represent a second view, with increasing conviction in going from state 5 to state 9. (Note that additional views can be added to this setup.) The state transitions are also given in the figure, with six threshold-based mechanisms in different colors. The blue and orange arrows among states 0 through 4 indicate an increase and decrease in conviction, respectively, while the green arrows represent a change in opinion. Analogous transitions exist for states 5 through 9. When an agent changes its opinion (e.g., from state 2 to state 5), it starts with minimum conviction. To simplify the analysis, we assume that the thresholds for state transition mechanisms for states 0 through 4 are the same as those for the analogous transition mechanisms for states 5 through 9, and likewise for contagion amounts.

Table 1 shows the thresholds used; values are given in pairs that reflect the symmetry between opinions. In Table 1(a), thresholds increase with increasing levels of conviction, meaning that it is more difficult to achieve greater levels of conviction. Reductions in conviction have the same threshold across convection levels in Table 1(b). Thresholds are generally greatest for changing opinions, as reflected in Table 1(c).

Three execution orders (priorities) of the three mechanisms, per time step, are provided in Table 2. For example, mechanism order 1 (M01) is such that at every time step, an agent is updated for (i) increasing conviction, then for (ii) change of opinion, and finally for (iii) decreasing conviction. A state transition occurs for the first mechanism that meets its transition criterion.

Table 1: Thresholds for state transitions for the three sets of mechanisms: (a) for increasing conviction; (b) for decreasing conviction; and (c) for changing opinion. The thresholds are the same for the same conviction level for both opinions in the FSM of Figure 3 and hence the first column contains two states.

(a)

State	Threshold
0/5	1
1/6	1
2/7	2
3/8	3
4/9	–

(b)

State	Threshold
0/5	–
1/6	2
2/7	2
3/8	2
4/9	2

(c)

State	Threshold
0/5	1
1/6	2
2/7	4
3/8	4
4/9	4

Table 3 provides the contagion amounts transmitted to agents in column 1 by agents in the states in the remaining columns. States not listed in columns 2 through 6 do not influence the state in column 1. Table 3(a) conveys the notion that higher levels of conviction have greater influence on increasing conviction than do lower levels of conviction. The values in Table 3(b) mean that the same and higher conviction

Table 2: For the FSM of Figure 3, there are six state transition mechanisms indicated by the colored arrows. The analogous mechanisms for states 0-4 and 5-9 are treated the same herein; thus there are 3 sets of mechanisms. This table provides 3 priority orderings of mechanisms, giving 3 types of simulations.

Name	**Sequencing of State Transition Mechanisms**
MO1	(i) increasing conviction, (ii) change of opinion, and (iii) decreasing conviction
MO2	(i) increasing conviction, (ii) decreasing conviction, and (iii) change of opinion
MO3	(i) change of opinion, (ii) decreasing conviction, and (iii) increasing conviction

states oppose reductions in conviction (i.e., the contagion amounts are negative). In Table 3(c), where an agent changes opinion, opposing agents with as great or greater conviction are most influential. (Note that the thresholds and contagion amounts in Tables 1 and 3, here functions of state, can instead depend on individual agents if that level of heterogeneity is needed.)

Table 3: Contagion amounts transmitted to each agent in the state of column 1 by the agents in the states of columns 2 through 6. The tables reflect the three sets of state transition mechanisms in Figure 3: (a) for increasing conviction; (b) for decreasing conviction; and (c) for changing opinion. The contagion amounts are the same for the same conviction level for both opinions in the FSM and hence there are pairs of states in column 1 and pairs of states in the other column headings.

(a)

affected	0/5	1/6	2/7	3/8	4/9
0/5	2	2	3	4	4
1/6	2	2	3	4	4
2/7	2	2	3	4	4
3/8	2	2	2	3	4
4/9	2	2	2	2	4

(b)

affected	0/5	1/6	2/7	3/8	4/9
0/5	–	–	–	–	–
1/6	1	-1	-1	-1	-1
2/7	1	1	-1	-1	-1
3/8	1	1	1	-1	-1
4/9	1	1	1	1	-1

(c)

affected	5/0	6/1	7/2	8/3	9/4
0/5	2	2	2	3	3
1/6	2	2	2	3	3
2/7	1	1	2	3	3
3/8	1	1	1	2	3
4/9	1	1	1	1	3

The initial conditions in most simulations are the same: 20 agents are in each of the states 1, 2, 3, 6, 7, and 8, and the remaining 75759 agents are in state 0. Thus, the great majority of agents (75819) have view 1 (states 0 through 4), with different convictions, and we investigate whether 60 nodes with an alternative view 2 (states 5-9) can persuade the 75819 to change their view. Each simulation consists of 50 iterations and the differences among iterations are the agents that are seeded in states 1, 2, 3, 6, 7, and 8. Agents in each of these six states form a connected subgraph, and each agent has at least 20 neighbors, so the agents form well-connected groups. The *i*th iteration of each simulation has the same initial conditions so that we can compare dynamics across simulations. Results are averaged over all iterations of a simulation.

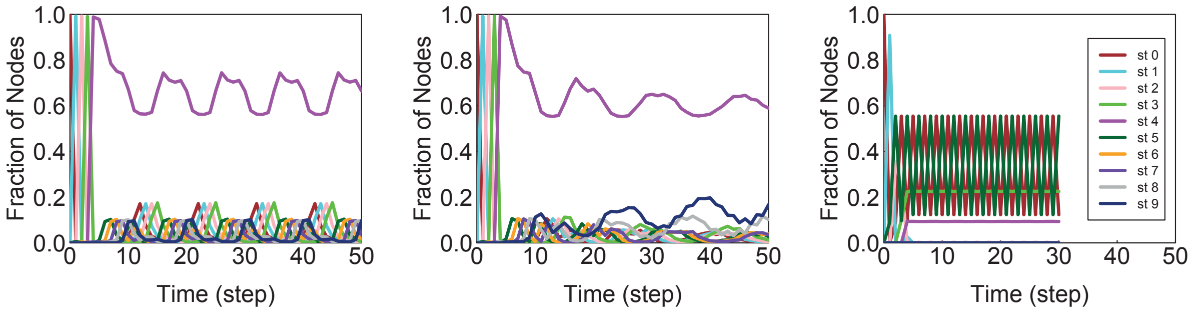


Figure 9: States of agents in the `epinions` network as a function of simulation time for the FSM of Figure 3 for different precedence orderings of state transition mechanisms based on Table 2: (a) MO1; (b) MO2; and (c) MO3. The legend is the same for all plots and state is abbreviated "st."

The three plots in Figure 9 show results, in order, for the three mechanism orderings in Table 2. State 4 emerges as the dominant state for MO1 (Figure 9(a)), with periodic behaviors of the other states never reaching more than 20% of nodes. For MO2 in Figure 9(b), state 4 once again dominates, but the periodicity of other states is destroyed, and the maximum conviction state for view 2 (state 9), although oscillating, is increasing and more than doubles its maximum fraction from that in Figure 9(a). For MO3, over 65% of agents cycle back-and-forth between the minimum conviction states for the two views (states 0 and 5); state 4 is no longer dominant. These results clearly show that the priority order of state transition mechanisms matters, that for studies of particular behaviors these priorities must be determined, and that the framework readily handles different mechanisms and orderings.

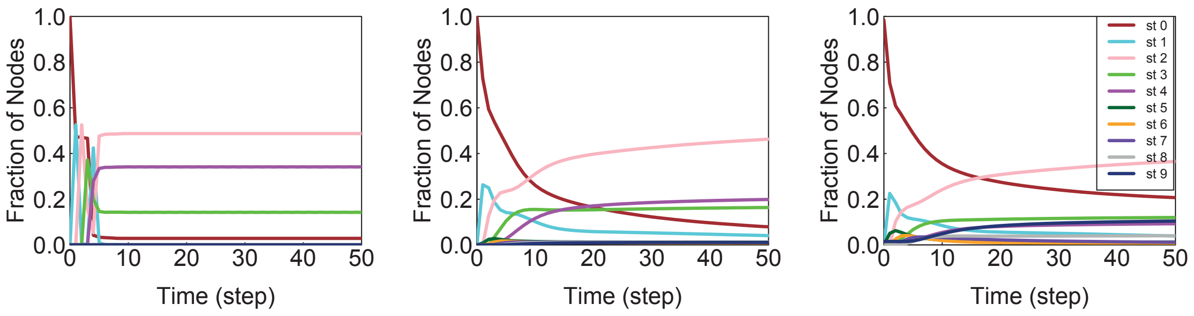


Figure 10: States of agents in the `epinions` network as a function of simulation time for the FSM of Figure 3, the MO2 ordering of state transition mechanisms, and all thresholds in Table 1 are increased by a factor of four: (a) deterministic results; (b) probabilistic results with $plow = 0$ and $p_{high} = 0.5$ in Figure 7; and (c) the conditions of (b) plus 500 agents fixed in state 9. The legend is the same for all plots and state is abbreviated "st."

In Figure 10, we focus on MO2 and all thresholds are increased by a factor of four. Figure 10(a) shows that state 2 now dominates, not state 4. Figure 10(b) depicts results for stochastic diffusion where $p_{low} = 0$ and $p_{high} = 0.5$ in Figure 7. State transitions occur more gradually in time and the fraction of agents in state 4 decreases. Overshooting is observed for state 1, where the fraction of agents initially increases and then monotonically decreases. This is of much interest in the sociology community and has been well documented as occurring in real groups (Bischi and Merlone 2009). In the two plots thus far, the fraction of agents with view 2 is minuscule. To investigate how to make view 2 more prevalent, we run simulations with the intervention that 500 agents are initially in state 9 and these agents do not change their state; otherwise conditions are the same as those in Figure 10(b). Figure 10(c) illustrates mixed results. On the one hand, 500 agents is less than 1% of the population, and yet it causes 11% of agents to migrate to that state. However, despite the disproportionate increase of state-9 agents, 11% is not nearly a majority. More agents in state 9 are required to change the majority view.

6 CONCLUSIONS

We have described a general-purpose, agent-based, discrete time simulation framework based on GDS. Our framework is flexible and can handle a range of agent behaviors arising out of diverse applications. We have demonstrated some of the features of the system through two case studies. The framework (i) permits developers to construct software plug-ins that describe agent behavior using only serial programming conventions (i.e., no parallel or distributed programming skills are required), (ii) provides a general plug-in interface to allow agent behaviors across many problem domains, where all user code is confined to one part of the simulation software, and (iii) enables fast turn around times (the duration from problem specification to generating useful simulation results), often in the range of 2 to 4 hours. While it is demonstrably scalable to large networks and parametric studies, future work includes improving these features.

ACKNOWLEDGMENTS

We thank the conference reviewers, our external collaborators, and members of the Network Dynamics and Simulation Science Laboratory (NDSSL) for their suggestions and comments. This work has been partially supported by NIH MIDAS project 2U01GM070694-7, NSF PetaApps Grant OCI-0904844, DTRA R&D Grant HDTRA1-0901-0017, DTRA CNIMS Grant HDTRA1-07-C-0113, NSF Netse CNS-1011769, NSF SDCI OCI-1032677, NSF CNS-0845700, and DOE DE-SC0003957.

REFERENCES

Aaby, B., K. Perumalla, and S. Seal. 2010. “Efficient Simulation of Agent-Based Models on Multi-GPU and Multi-Core Clusters”. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*.

Bischi, G., and U. Merlone. 2009. “Global Dynamics in Binary Choice Models with Social Influence”. *J. Math. Sociology* 33:277–302.

Bisset, K., J. Chen, X. Feng, Y. Ma, and M. Marathe. 2010. “Indemics: an Interactive Data Intensive Framework for High Performance Epidemic Simulation”. In *Proceedings of the 24th ACM International Conference on Supercomputing*, 233–242.

Bisset, K., J. Chen, C. Kuhlman, V. Kumar, and M. Marathe. 2011. “Interaction-Based HPC Modeling of Social, Biological, and Economic Contagions Over Large Networks”. In *Proceedings of the 2011 Winter Simulation Conference*. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Bisset, K., X. Feng, M. Marathe, and S. Yardi. 2009. “Modeling Interaction Between Individuals, Social Networks, and Public Policy to Support Public Health Epidemiology”. In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. Rossetti, R. Hill, B. Johansson, A. Dunkin, and R. Ingalls, 2020–2031. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

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

Carothers, C., and K. Perumalla. 2010. “On Deciding Between Conservative and Optimistic Approaches on Massively Parallel Platforms”. In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 678–687. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Centola, D., and M. Macy. 2007. “Complex Contagions and the Weakness of Long Ties”. *American J. Sociology* 113 (3): 702–734.

Chandy, K., and J. Misra. 1979. “Distributed Simulation: A Case Study in Design and Verification of Distributed Systems”. *IEEE Transactions on Software Engineering* SE-5:440–452.

Dodds, P., and D. Watts. 2005. “A Generalized Model of Social and Biological Contagion”. *J. Theo. Biology* 232 (4): 587–604.

Epstein, J. 2007. *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton University Press.

Fowler, J., and N. Christakis. 2008. “Dynamic Spread of Happiness in a Large Social Network: Longitudinal Analysis Over 20 Years in the Framingham Heart Study”. *British Medical Journal*.

Gilbert, N. 2007. *Agent-Based Models*. Sage Publications.

Granovetter, M. 1978. “Threshold Models of Collective Behavior”. *American J. Sociology* 83 (6): 1420–1443.

Hollander, C., and A. Wu. 2011. “The Current State of Normative Agent-Based Systems”. *Journal of Artificial Societies and Social Simulation* 14 (6): 1–47.

Hybinette, M., E. Kraemer, Y. Xiong, G. Matthews, and J. Ahmed. 2006. “SASSY: A Design for Scalable Agent-Basd Simulation System Using a Distributed Discrete Event Infrastructure”. In *Proceedings of the*

2006 Winter Simulation Conference, edited by L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 926–933. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Jefferson, D. 1985. “Virtual Time”. *ACM Transactions on Programming Languages and Systems* 7:404–425.

Karaoz, U., T. Murali, S. Letovsky, Y. Zheng, C. Ding, C. Cantor, and S. Kasif. 2004. “Whole-Genome Annotation By Using Evidence Integration in Functional-Linkage Networks”. *Proceedings of the National Academy of Sciences* 101 (9): 2888–2893.

Macal, C., and M. North. 2010. “Tutorial on Agent-Based Modelling and Simulation”. *Journal of Simulation* 4:151–162.

Macy, M., and R. Willer. 2002. “From Factors to Actors: Computational Sociology and Agent-Based Modeling”. *Annual Review of Sociology* 28:143–166.

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

North, M., and C. Macal. 2009. “Foundations of and Recent Advances in Artificial Life Modeling with Repast 3 and Repast Simphony”. In *Artificial Life Models in Software*, edited by A. Adamatzky and M. Komosinski, 37–60. Springer.

Olfati-Saber, R., J. Fax, and R. Murray. 2007. “Consensus and Cooperation in Networked Multi-Agent Systems”. *Proceedings of the IEEE* 95 (1): 215–233.

Perumalla, K., and S. Seal. 2010. “Reversible Parallel Discrete-Event Execution of Large-Scale Epidemic Outbreak Models”. In *Proceedings of the 24th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*.

Wang, G., M. Salles, B. Sowell, X. Wang, T. Cao, A. Demers, J. Gehrke, and W. White. 2010. “Behavioral Simulations in MapReduce”. *Proceedings of the VLDB Endowment* 3 (1): 952–963.

Watts, D. 2002. “A Simple Model of Global Cascades on Random Networks”. *PNAS* 99 (9): 5766–5771.

AUTHOR BIOGRAPHIES

CHRIS J. KUHLMAN is a graduate student in the Computer Science Department at Virginia Tech. His email address is ckuhlman@vbi.vt.edu.

V. S. ANIL KUMAR is an Associate Professor in the Computer Science Department at Virginia Tech. His email address is akumar@vbi.vt.edu.

MADHAV V. MARATHE is a Professor in the Computer Science Department at Virginia Tech. His email address is mmarathe@vbi.vt.edu.

HENNING S. MORTVEIT is an Associate Professor in the Mathematics Department at Virginia Tech. His email address is hmortveit@vbi.vt.edu.

SAMARTH SWARUP is a Post Doctoral Research Associate in NDSSL at Virginia Tech. His email address is swarup@vbi.vt.edu.

GAURAV TULI is a graduate student in the Computer Science Department at Virginia Tech. His email address is gtuli@vbi.vt.edu.

S. S. RAVI is a Professor in the Computer Science Department at the University of Albany—SUNY. His email address is ravi@cs.albany.edu.

DANIEL J. ROSENKRANTZ is an Emeritus Professor in the Computer Science Department at the University of Albany—SUNY. His email address is drosenkrantz@gmail.com.