High Performance Computational Social Science Modeling of Networked Populations

Christopher James Kuhlman

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

Madhav V. Marathe, Chair
Henning S. Mortveit
Sekharipuram S. Ravi
Eli Tilevich
V. S. Anil Vullikanti

March 25, 2013
Blacksburg, Virginia

Keywords: Social Behavior, Contagions, Networks, Control of Contagion Processes, Graph Dynamical Systems, Modeling and Simulation, Rapid Development

High Performance Computational Social Science Modeling of Networked Populations

Christopher James Kuhlman

(ABSTRACT)

Dynamics of social processes in populations, such as the spread of emotions, influence, opinions, and mass movements (often referred to individually and collectively as contagions), are increasingly studied because of their economic, social, and political impacts. Moreover, multiple contagions may interact and hence studying their simultaneous evolution is important. Within the context of social media, large datasets involving many tens of millions of people are leading to new insights into human behavior, and these datasets continue to grow in size. Through social media, contagions can readily cross national boundaries, as evidenced by the 2011 Arab Spring. These and other observations guide our work. Our goal is to study contagion processes at scale with an approach that permits intricate descriptions of interactions among members of a population. Our contributions are a modeling environment to perform these computations and a set of approaches to predict contagion spread size and to block the spread of contagions. Since we represent populations as networks, we also provide insights into network structure effects, and present and analyze a new model of contagion dynamics that represents a person's behavior in repeatedly joining and withdrawing from collective action. We study variants of problems for different classes of social contagions, including those known as simple and complex contagions.

Acknowledgments

I thank my committee. Each of these professors gave me valuable help along the way. In particular, my advisor, Professor Madhav V. Marathe, introduced me to contagion processes, gave me ideas on pursing study of complex contagions, and gave me helpful advice. Professor S. S. Ravi, through is friendship with Professor Marathe, worked closely with me on every aspect of this work. The professors on my committee graciously included me in other projects from which I benefited. I also thank Emeritus Professor Daniel J. Rosenkrantz, University at Albany, SUNY; he has helped me, too, and has been an integral part of all of this work.

I thank several professors who taught me things in a classroom, teachers in the noblest sense of the word: Jeffrey T. Borggaard, Leslie Kay, Peter A. Linnell, and Henning S. Mortveit (Virginia Tech); Chao-Kun Cheng and William J. Terrell (Virginia Commonwealth University); and Donald E. Carlson, John P. D'Angelo, and E. Graham Evans Jr. (University of Illinois); professional colleagues: Victor D. Aaron, G. Graham Chell, R. Craig McClung, and Richard A. Page (Southwest Research Institute); and George J. Foster (Computer Sciences Corporation); and those who taught me other things: Patricia Soriano, Donald Weed, and my kids Nora Claire and Finnegan Thomas.

I thank Peter D. Barnes, Jr. of Lawrence Livermore National Laboratory who spent a lot of time with me during my stay there.

I thank Vedavyas Duggirala, Bill Marmagas, and Mike Snow for expeditiously setting up a new Sandybridge cluster, merging two clusters (queues and hardware) to enable us to utilize over 1000 compute cores, providing access to additional clusters, and installing requisite software and tools to allow us to complete this work. Their efforts made many things possible.

I thank my parents and my wife's parents for giving my kids the love and attention which otherwise would have been lacking. I thank my siblings Kathy, Mark, and Karen for their encouragement and interest. I thank my Grandma Dugan.

I thank my wife Helen Renee for her help, encouragement, and patience in this process. In addition to helping me, she gave of herself, and then some, to see that our kids did not go without.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Contagion is a general term defining any entity or phenomena that can spread through a (human) population. Examples include fads, opinions [220], trust [134], attitudes [217], influence itself, emotions (e.g., anger and sadness) [42], stress [58], ideologies, diseases, folk knowledge [67], and malware [75]. *Dynamics* of contagion spread is important for understanding system *behavior* in various fields and applications. Among these are cell immunology [137], biology [160], ecology [59], (discrete) mathematics [125], economics [106], health sciences [285], computer networks [270], politics [290], statistical physics [63], epidemiology [273], and languages [60]. In the social sciences alone, there are many applications for contagion dynamics: technology adoption [67], rumors [163], contraception practices [132], participation in games [281], fanaticism [143], mass movements [99] and civil unrest [108], collective action [83, 290], revolutions [10], repression [291], and substance abuse [298]. Some of these domains and phenomena are listed at the top of Figure 1.1. Hence, contagions and contagion dynamics are pervasive, if not ubiquitous.

Impacts of these phenomena can be grave. The 1992 Los Angeles riots resulted in $1 billion in damage, 54 deaths, and thousands of injured [332]. Government responses to dissension can be equally destructive and the history of South Africa provides ample case studies. For example, the 1980s Botha-led government used unlawful imprisonment, torture, intimidation, mental abuse, and assassination as tactics against the anti-apartheid movement [305]. Furthermore, as byproducts, political instabilities can produce increased gang violence [305] and adversely affect foreign investment [123]. The Shining Path in Peru from 1980 to 1993 is estimated to have caused 30,000 deaths and $24 billion in property damage [254]; Peru's 1990 gross domestic product was $24 billion. Events such as the recent Arab Spring can result in significant widespread change [10, 335]. Drinking, smoking, and obesity are the three behaviors that contribute most to chronic health problems in the U. S., and all are

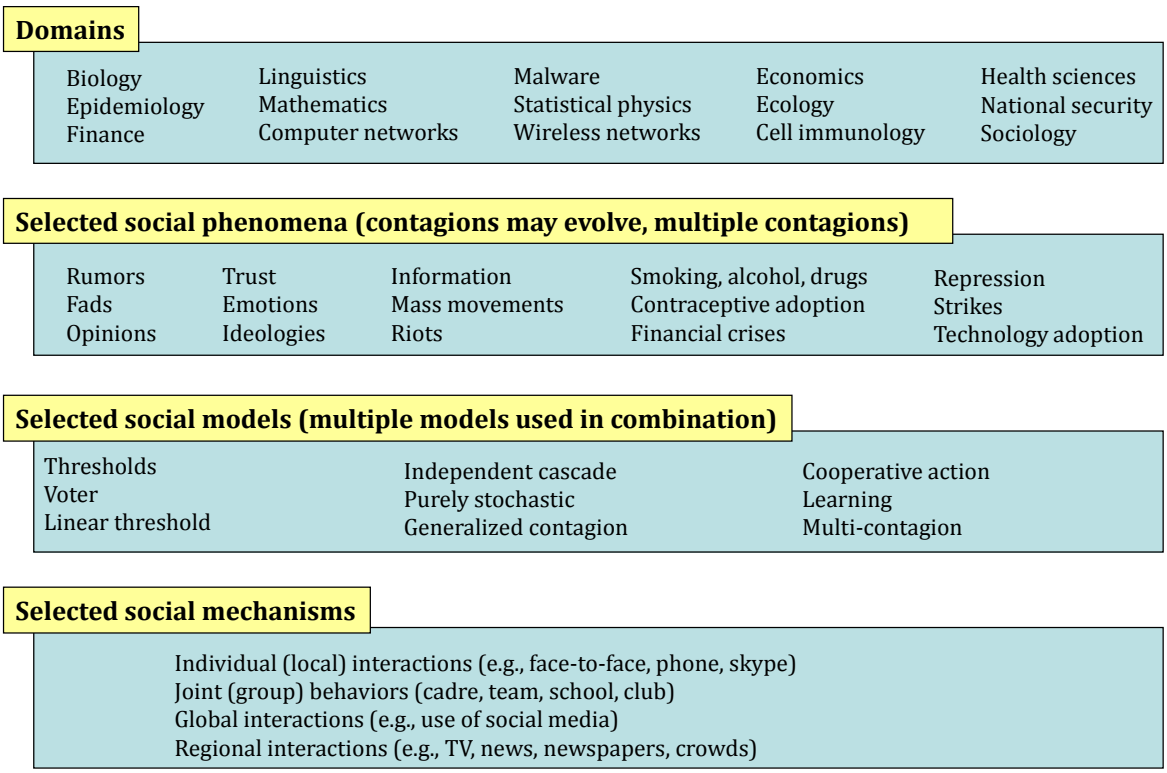


Figure 1.1: High level view of contagion processes along four dimensions, with examples.

greatly affected by peer influence (a form of contagion); e.g., [144]. Annual costs of these behaviors and drug abuse in the U. S. are given in Table 1.1.

Table 1.1: Estimated annual costs in U. S. of chronic health problems.

Behavior	**Annual Cost**
Alcohol abuse	$192 billion [227]
Tobacco use	$168 billion [227]
Obesity	$147 billion [112]
Drug Abuse	$181 billion [247]

1.2 Document Structure

To justify modeling of contagion processes, we first describe direct and indirect experimental evidence of contagion processes in populations. Thereafter, we present background work on the topics of this thesis, which include social dynamics models, graph dynamical systems (the modeling approach we use for our investigations), software frameworks and simulators for modeling contagion processes, and blocking diffusion in particular popular classes of dynamics models. While doing so, we motivate our work in the context of these other studies. In the last part of the Introduction, we overview our research contributions. Subsequent chapter topics are provided in Figure 1.2, where we emphasize the role of graph dynamical systems as a foundation of our work.

Throughout this work, we represent populations as graphs. We will use the term *node* or *vertex* to mean a member of a population (often an individual) and *edge* to mean an interaction between a pair of nodes, since these are closest to our (abstract) representation.

Tabled 1.2 and 1.3, respectively, provide symbols and acronyms used throughout this thesis and their meanings. In many cases, more than one variant of a symbol is commonly used.

1.3 Experimental and Empirical Evidence

1.3.1 Evidence for Contagions

We sample a number of different fields to provide an overview of experimental evidence of contagions, empirical studies that indirectly infer the existence of contagion processes, and explanations of observed behavior using contagion models. This is not an exhaustive review; the literature is vast.

Drug use and interventions. Network science and contagion dynamics ideas have been employed in public health to thwart undesirable behaviors [93, 165, 168, 203, 307, 320, 321].

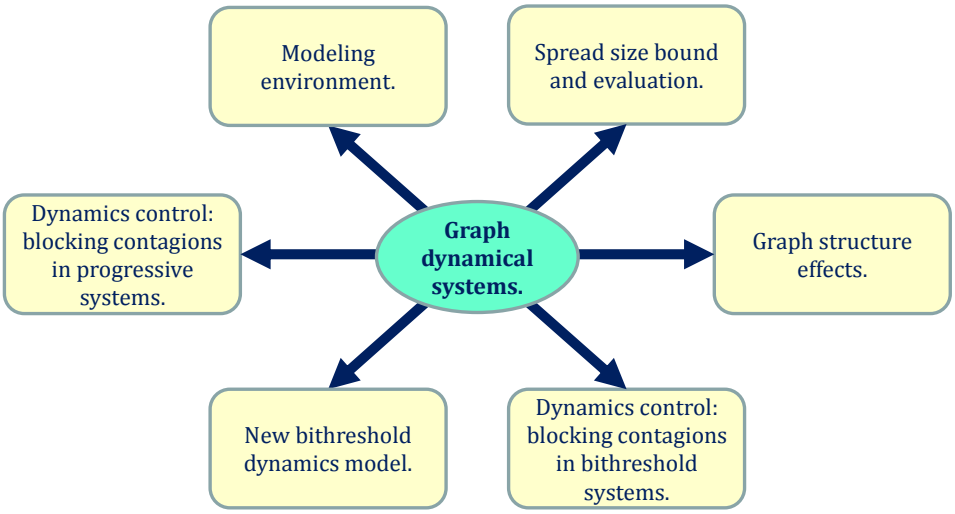


Figure 1.2: Overview of thesis topics: graph dynamical systems (Chapter 2), modeling environment for network-based social contagions (Chapter 3), method for bounding the extent of contagion propagation (Chapter 4), blocking propagation of progressive contagions (Chapter 5), network structure effects (Chapter 6), and a bithreshold dynamics model and contagion blocking (Chapter 7).

Table 1.2: Common symbols used in this work.

Symbol	**Meaning**
a_i	Function.
d_{ave}, $d_{ave}(v)$	Average degree of a network, or a node.
e	Edge.
f, f_v, f_i	(Local) vertex (for vertex v, or vertex i)
f, f_e, $f_{(u,v)}$	(Local) edge (for edge e, directed edge (u, v).
m	Number of edges in graph.
n	Number of nodes in graph.
n_c	Number of contagions.
n_s	Number of seed nodes.
s_i, s_v, s_{v_i}, $s_v(t)$, x_v, x_i, $x_v(t)$	State for vertex i, v, v_i (at time t).
s_e, $s_{(u,v)}$, x_e, $x_{(u,v)}$, $x_{(u,v)}(t)$	State for an edge e or (u, v) (at time t).
s'_i, s'_v, s'_{v_i}	Next state for vertex i, v, v_i (newly computed state value).
t	Time.
(u, v)	Directed edge from u to v.
$\{u, v\}$	Undirected edge between u and v.
v, v_i	Vertex, node.
$x[v]$	A restricted state; the states of v and all of v's neighbors.
$\mathbb{B}$	Boolean state space; i.e., $\{0, 1\}$.
C	Set of contagions.
$\mathcal{C}$, $\mathcal{C}(t)$, x, $x(t)$	System configuration; i.e., a system state. Configuration at time t.
CC_{ave}	Average clustering coefficient for a graph.
E	Set of edges.
$\mathcal{E}$	ENSIM process; typically runs one MPI process instance.
$\mathbf{F}$, $\mathbf{F}_\pi$	Family of vertex and edge functions; GDS map. GDS map for a particular permutation.
G, $G(V, E)$	Graph, network.
I	Seed set.
K, K_v	Vertex state set.
K_e	Edge state set.
N	Number of compute (or ENSIM) processes.
$\mathcal{S}$	Graph dynamical system (GDS), sequential dynamical system (SDS), or synchronous dynamical system (SyDS); depends on context.
V, V_i	Set of vertices.
W	Update scheme.
π	Permutations. Frequently a sequence of nodes and/or edges.
π_B	Block sequential permutations. Frequently a sequence of nodes and/or edges.
θ, θ_i, θ_v	Threshold for vertex i or v.
τ	Epidemic threshold.

Table 1.3: Common acronyms used in this work.

Acronym	**Meaning**
BCH	Betweenness centrality critical node heuristic for blocking contagion spread.
CBH	Covering based hueristic for blocking contagion spread.
ECH	Eigenvector centrality critical node heuristic for blocking contagion spread.
ER	Erdos-Renyi random graph.
ED	Exponential-decay, or a graph with such a degree distribution.
GDS	Graph dynamical system.
HCH	High degree critical node heuristic for blocking contagion spread.
PBH	Potential based hueristic for blocking contagion spread.
RCH	Random critical node heuristic for blocking contagion spread.
SDS	Sequential dynamical system.
SF	Scale-free, or a graph with such a degree distribution.
STC	State transition criterion; the conditions under which a node or edge changes state.
STM	State transition model; the set of state transitions capturing the dynamics of vertices and edges.
SyDS	Synchronous dynamical system.

Studies often examine drug use in populations (typically comprised of hundreds of individuals) and how these activities might spread diseases such as HIV or hepititus B and C; see [319] for an overview. Field survey of participants is a dominant method of data collection, and is used to identify opinion leaders. These individuals participate in a training program that provides them with information on how to adapt their practices in order to reduce the risk of harm (e.g., disease spread). These leaders then provide this information to their friends in their natural environment, and in this way the goal is to educate the community and promote healthier practices. This method is called "snowballing" [319]. A motivating idea is that people in a community will listen to opinion leaders because they have similar backgrounds, perspectives, and habits [203] (this is homophily; informally, "like attracts like" [223]) . Hence, these studies are essentially intervention programs. By surveying people before and after the intervention, a study can determine whether the information is propagating through the group, to whom, by what routes of the human network, and ultimately, whether the intervention is successful. Post-training surveys of population members, to measure change in behavior, are typically conducted 6 to 18 months after training. Data collected from these studies demonstrate (multi-) contagion dynamics in the form of information spread to, and alterations of habits of, members. Hence, these studies involve both indirect and direct evidence of contagion processes in the form of information diffusion and behavior change in networked populations.

For example, there are small population studies that provide evidence for utilizing information diffusion to prevent overdoses among drug users [285]. In [193], opinion leaders were incentivized to spread information to encourage the use of condoms and cleaning of

needles among injecting drug users (IDUs). In that study, the spread of information was observed with the result of increased condom usage. The study used not only social networks through the solicitation of nominated opinion leaders, but also tracked information exchange within it. Evidence shows that HIV intervention strategies are effective in reducing spread of HIV and sexually transmitted infections (STIs); e.g., see [93] for a composite or meta study of African American women in the U.S. There are multiple strategies, including those based on social networks, information diffusion, and social influence. Risk of unprotected sex dropped in Bulgaria when network leaders were convinced to spread safety information [165]. In [203], 523 individuals in Hartford, CT were studied to "reduce risky drug use and sexual practices." Of these, 112 were trained, representing a significant fraction ($> 20\%$) of the population. Various survey data indicated that risk-reducing information spread by the 112 trainees propagated into the network and by 6 months, 60% of the population received the information and 57% of the 411 remaining members adopted safer practices. Those with shorter paths to the trainees were more likely to be exposed to the information.

Emotional contagions. Human emotions (e.g., happiness, fear, disgust, anger, sadness) are well-known to transfer among interacting people [102, 140]. Scales for emotions and the factors that go into them are provided in [102, 114], indicating that there degrees of feeling. Experiments on fear and anger are discussed in [235], and on anger and sadness in [42]. These two studies are not focused on emotions as contagions, but rather provide evidence that mood affects decision-making processes, and therefore could affect one's decision in adopting a contagion. Further, [235] indicates that emotions are not binary, but rather inherently have different degrees. This is interesting because many social models of contagion assume a binary state space: a node either possesses a contagion or it does not. A simulation study [312] demonstrates that making fear a binary variable reduces agreement with measurements of actual events of fear propagation in a crowd, compared to making fear a continuous or multi-state variable. Although perhaps not strictly an emotion, stress can evoke worry, which is an emotion [140]. Spreading of stress between spouses has been found [58].

Financial contagions. Financial markets offer an interesting perspective on contagion dynamics because these contagions (e.g., instabilities, downturns, subsequent rebounds) are often studied at the company, institution, or country level, rather than at the individual human level. Evidence for financial contagions comes from [195, 216, 336, 337]. Contagion spread across countries from the 2007 subprime mortgage crisis in the U.S. is studied in [195]. The recovery in Japan is shown to have helped lift the U.S. economy following the 1987 U.S. stock market crash [336]. The 1997 financial crisis in East Asia was exaccerbated by contagion spreading across countries [337]. A study of broader market fluctuations from 1996 to 2007 [216] again showed evidence of contagion processes, where local instabilities were determined to cause—in some cases—global problems. At the individual level, evidence of contagion spread among investors is given in [287].

Multiple contagions. Here we are interested in multiple interacting contagion processes. Twitter data around the time of public demonstrations in Spain in 2011 were analyzed [127]. Two simultaneous processes were identified: the dynamics of recruitment to the protest

cause (i.e., influence) and the dynamics of information spread. Moreover, these factors can be mutually reinforcing. In [316], two mechanisms of contagion were identified in the use of Facebook: the process of joining Facebook and dynamics of early usage. Multinetworks and multiple contagions have been observed in analyses of social movements and protests [128,149,220]. Furthermore, some simulation studies model simultaneous contagions, such as the spread of viruses and fear [109], which tacitly assume the existence of multiple contagions. Simple examples of multiple contagions can be found in advertising, where two or more products compete in a market; e.g., marketing campaigns of Coke and Pepsi each seek to influence consumers to use their products.

Political campaigns can be viewed as multi-contagion efforts in the following sense. A campaign seeks to promulgate information to win voters to their side. However, this is not enough: campaigns must also convince their constituents to act, in the form of actually voting. Each each of these processes can be considered contagions.

A multi-contagion episode involving language spread and disease occurred in England in the years 1348-1350. At that time, Latin was the language of the educated and it hampered the spread of English. However, the bubonic plague, also known as the Black Death, disproportionately hit urban centers and monestaries, the sources of Latin's usage. As stated in [52], "Though the Black Death was a catastrophe, it set in train a series of social upheavals which would speed the English language along the road to full restoration as the recognized language of the natives."

Mass protests and social movements. Data support what seems to be the generally accepted notion that social media is playing an ever increasing role in mass protests. Lysenko and Dsouza [207–209] have done what appears to be pioneering work in reconstructing timelines and determining factors affecting the success of revolutions and protests. One focus is the use of information and communication technologies (ICTs), such as cellular telephones and social media. From these works and others, a trend emerges. In the 2004 Ukrainian Orange Revolution, ICTs were little used [208]. In the 2009 Moldovian Revolution, Twitter was not used to instigate protests, but rather was used to inform people post-event [209]. That is, social media was used in a reactive capacity. In the Arab Spring events in Tunisia, Egypt, and Libya of 2011, ICTs (and particularly social media) were reportedly used extensively for information dissemination, fostering participation in demonstrations and playing an integral part in revolters' successes; i.e., social media was now used in a proactive or instigating manner [11, 12, 150, 289]. Hence, consensus wisdom is that social media is used for information (contagion) spread, and conventional wisdom is that this information influences certain (perhaps many) individuals (and thus there is an influence contagion).

These results should not be surprising since even before the internet age, printed media was used to galvanize people to act and even revolt. In the 1840s in Europe, governments in 13 countries across Europe were exposed to criticisms from their citizens. In some cases (France, Prussia, the Hapsburg Empire), these evolved into protests and demonstrations. While in the end none of these movements grew to revolution status, some governments were temporarily

unseated or fled in fear, and/or permanently gave greater freedoms and made concessions to their citizens. For example, France declared universal manhood suffrage in 1848. The spread of information within countries, and across national boundaries, was credited with fomenting social unrest throughout the region. Information diffusion occurred through printed media and people traveling throughout Europe (e.g., to conferences) [296]. So, too, in South Africa, during the anti-apartheid movement, printed media served to unite and give courage to the oppressed [305]. Media also played a role in East German anti-communist demontrations in 1953 [205].

Even today, face-to-face interaction may be the primary means of influence, all the moreso in regions where social media is limited. The spreading of ideology among villages in Afghanistan has been studied in [152], where both Talaban and U. S. personnel try to win tribal leaders to their side. This is a classic use of network science and notions of contagion, since the tribal leaders hold great sway over the opinions of the community. Hence, these practices are focused on opinion leaders, just as in public health [193].

That said, conventional wisdom is that social media are ever more useful in instigating and coordinating collective behavior. This is seen in the actions of governments such as that of China in monitoring social media to prevent (protest) gatherings of its citizens [170].

Social media and online data. Social media and online interactions enable experiments of contagion processes at unprecedented scales. One example is the mining of behavior of tens of millions of Facebook users [316], which resulted in a new experimentally-determined model of contagion dynamics. The model is based on *contexts*. For example, a teenager may have interactions in the contexts of family, friends, school (classmates and teachers), and sports teams. Each context acts as a distinct aggregated influencer; e.g., if a node operates in five contexts, then there are five sources of contagion. In this model, multiple individuals within a context do not provide separate influence. Threshold models have also been used to understand the behavior of tens of thousands during mass protests in Spain, through Twitter [127]. Voting behavior of 61 million individuals has been demonstrated to be peer-influenced [46]. These and other studies [14, 15, 130, 190, 251, 277] are producing results from online social populations with up to 100s of millions of users. As one example, it was found that younger Facebook users are more easily influenced than older ones, and that influential users are less susceptible to influence than are non-influential ones [15]. We note that while much of the data come from Twitter and Facebook, there are data from other systems: online purchase recommender systems [196], videos of crowd behavior in Greece [312], Flickr [70], LiveJournal [172], customized online experiments [68], email [176], and blogs [133].

Sociology studies. We conclude with a description of some sociology studies. Many of the previously-mentioned works are sociologically driven. Here, we are interested in higher level ideas rather than data for a specific problem. The goal is to put contagions in a larger context, thus supplementing the earlier works.

Influence is considered a contagion process. An individual may influence another, causing the second person to acquire a contagion (e.g., a point of view, a habit, information). Overviews

of the ideas of social influence are given in [217, 220].

Some simple quantitative models are based on **thresholds**. A threshold θ is an intrinsic property of a node that describes how much influence must be exerted on it in order for it to acquire a contagion. The greater the threshold, the greater the resistance to accepting a contagion. For example, if a node v's threshold is 3, then one interpretation of threshold is that at least 3 of v's neighbors must already possess the contagion and be willing to pass it on, in order for v to acquire it. Some of the seminal works on these threshold ideas are [67, 132, 281]. Examples of threshold phenomena from these works include participation in revolts, technology adoption, and use of contraceptives.

Schelling [281] provides some examples of slightly more complicated behaviors, describing how individuals may acquire a contagion, divest themselves of it, re-acquire it, and so on. We call these behaviors **back and forth**, which contrast with **progressive** systems where once a choice is made, it is irrevocable. To this point of back and forth behavior, he states, "Numerous social phenomena display cyclic behavior ...", see [281, p. 86]. He provides commonplace examples, such as pick-up volleyball games among college students where some people attend weekly games sporatically. His example of illegally crossing a street embodies many ideas, as follows. Some people will initiate the action [of illegally crossing a street] (i.e., zero-threshold individuals), others will immediately follow (i.e., low-threshold individuals), and others will wait until many others have started (high-threshold individuals). He further describes cyclic behavior in that some will initiate the action (crossing a street) and will then observe others, and if there is an insufficient number of followers, they will reverse course (e.g., return to the curb). Other cyclic or back-and-forth sociological examples are provided in [35], and in modeling social movements [249]. It is well-recognized that dieters engage in cycles of dieting and non-dieting over time, the phenomenon being so common that it has a name: yo-yo dieting [20]. Similarly, with respect to alcohol consumption, we have "on the wagon" (i.e., abstinence from drinking) and "[falling] off the wagon" (i.e., a return to drinking after giving it up). The social impact in terms of health costs of these cyclic behaviors are summarized in Table 1.1. The cyclic nature of rumor spreading is discussed in [163].

Reducing levels of chronic illnesses represent special problems because they require modifying people's ingrained habits such as over-eating. Hence, the difficulty in halting undesirable behavior may be greater than that of starting the behavior; i.e., a start threshold may be less than a threshold to stop. Particularly relevant for addictive behavior is the notion of **self-efficacy**, stemming from social cognitive science, which describes an addict's perceived ability to change her behavior. The greater a person's self-efficacy, the more empowered she feels to conquer an addiction. Furthermore two of the four factors that produce self-efficacy are peer-driven: observations of others and verbal persuasion. These ideas are summarized in [278]. The points are: (*i*) self-efficacy can be modeled as an internal threshold phenomenon, (*ii*) that it is driven significantly by peer influences, and (*iii*) that cyclic behavior can entail changing (i.e., time-varying) thresholds for nodes.

This last point—the notion of time-varying thresholds—has been put forth by Macy [212] as part of a learning model. If a person participates in a group action for a common cause, and a goal is not achieved, his internal threshold for subsequent participation may increase. Similarly, if a goal is achieved, then his threshold may decrease. The notion of contagions morphing over time, in the form of rumors, is described in [162]. The effects of burstiness in contagion propagation (i.e., increased frequency of transmission) has been studied [127, 300] and demonstrated to increase contagion spread in some cases; see also [174].

Another interesting idea is that of *levels* of commitment. Many sociological studies (e.g., [67, 132, 281]) treat contagions as binary phenomena. In Macy's contagion dynamics study [212] he quotes Elster [107], "although the assumption of a dichotomous independent variable—the decision [by humans] to cooperate [to meet some goal]—is convenient for many purposes, it is often unrealistic. Often, the problem facing the actor is not *whether* to contribute, but *how much* to contribute." This can apply to various situations, such as joining a strike or adopting a new technology [67, 132]: one makes a partial commitment. The idea of varying levels of commitment is also described in [248]. Add Health longitudinal data [139] generated on 7-12 graders in 1994-1995 and a subset of the initial group most recently in 2008, contain survey results that clearly illustrate that drinking and smoking behavior of young adults have various levels of participation.

Watts [329] argues that humans often use threshold-based approaches to decision making (e.g., in choosing to commit or acquire a contagion) even when more intricate decision processes are available. For example, there may be insufficient time to take a more elaborate approach or the data required for more complicated reasoning may be lacking. Thus, he argues that threshold-based decisions are not uncommon.

Threshold-based approaches are most commonly applied within the context of a networked population. That is, the agents in a population that may influence an agent v are precisely those agents that interact with v. These agents and interactions are naturally represented as graphs where nodes represent agents and edges represent interactions. Sociological works such as [89, 128, 129, 303, 304] advocate for a network-science type of approach for modeling behavior, where agent interactions are dependent on the network(s) in which they are embedded. Many studies along these lines have been completed; e.g., [83, 100, 129, 188, 189, 290, 292, 339].

1.3.2 An Opposing View

There is not unanimity in the notion of contagions and their effects on populations. Over the last few years, a debate has arisen based on the following question: does one node change state owing to interactions with its neighbors or does an edge form between two nodes that are similar in some way (e.g., homophily [21], where like attracts like). The problem is well-explained in [14], and there are several other works [15, 202, 225]. Either way, these types of dynamics can be modeled. For example, dynamic networks and edge existence can

be modeled with stochastic actor models [293, 297]. In this work, we take the view that contagion processes propagate through (networked) populations.

1.4 Social Dynamics

The modeling approach we use is defined formally in Section 1.5. Here, we informally describe contagion dynamics in providing an overview of some social dynamics models. The **neighbors** of a node v_i are the nodes v_j that share edges with v_i. The number of neighbors of v_i is v_i's **degree**. Unless stated otherwise, we use the terms *graph* and *network* interchangably. Every node occupies a discrete state from a finite set of permissible states, with respect to each contagion, at every point in time. Contagion(s) propagate on the network in discrete time increments and we compute node state changes as a function of time t; we informally call this process a **simulation**. Time in a numerical simulation of a contagion process is $t \in \{0\} \cup \mathbb{N}$, with $t = 0$ corresponding to the start time and a specified t_{max} corresponding to the end time of a simulation. The states of all nodes at $t = 0$ are specified based on their physical meanings taken from the application domain. A node v interacts with its neighboring nodes, defined by the edges incident on v. Nodes may transition from one state to another at each time based on these interactions.

In this section, we focus on state transition models (STMs). A **state transition model** is a set of states that nodes can occupy and admissible state transitions (changes). It does not include the conditions under which state transitions occur (i.e., the **state transition criteria** (STC)). This approach enables us to describe in general a variety of STMs and discuss particular STC with respect to particular applications. Thereafter, we describe one type of STC—the threshold model—that quantifies the conditions under which a node changes state. Threshold approaches are popular in the social science literature and we use them in this work. There are other state spaces, sets of state transitions, and state transition criteria, and we note several review articles on various aspects of dynamics models and networked populations that go beyond this description [8, 41, 63, 104, 238, 323].

Node states and state transitions. There are many models of contagion processes whose state changes are represented by the STMs of Figure 1.3 where numbers inside circles represent node states and arrows represent permissible transitions. Each of these STMs may be used in a multiple domains. Many models focus on binary (i.e., $\{0, 1\}$) node state systems, where a node either does not (state 0) or does (state 1) possess a contagion (e.g., [132]). Arguments for the use of binary models in many realistic situations are given in [329]. We also use the generic terms **unaffected** (**affected**) nodes to denote nodes that do not (do) possess a contagion. These are analogous to the **uninfected** and **infected** states used in epidemiology. Other terms employed include **inactive** and **active**.

In some models, once a node transitions from state 0 to state 1 (or starts in state 1), it remains in that state. The $1 \rightarrow 0$ transition is not permitted (Figure 1.3(a)). These models

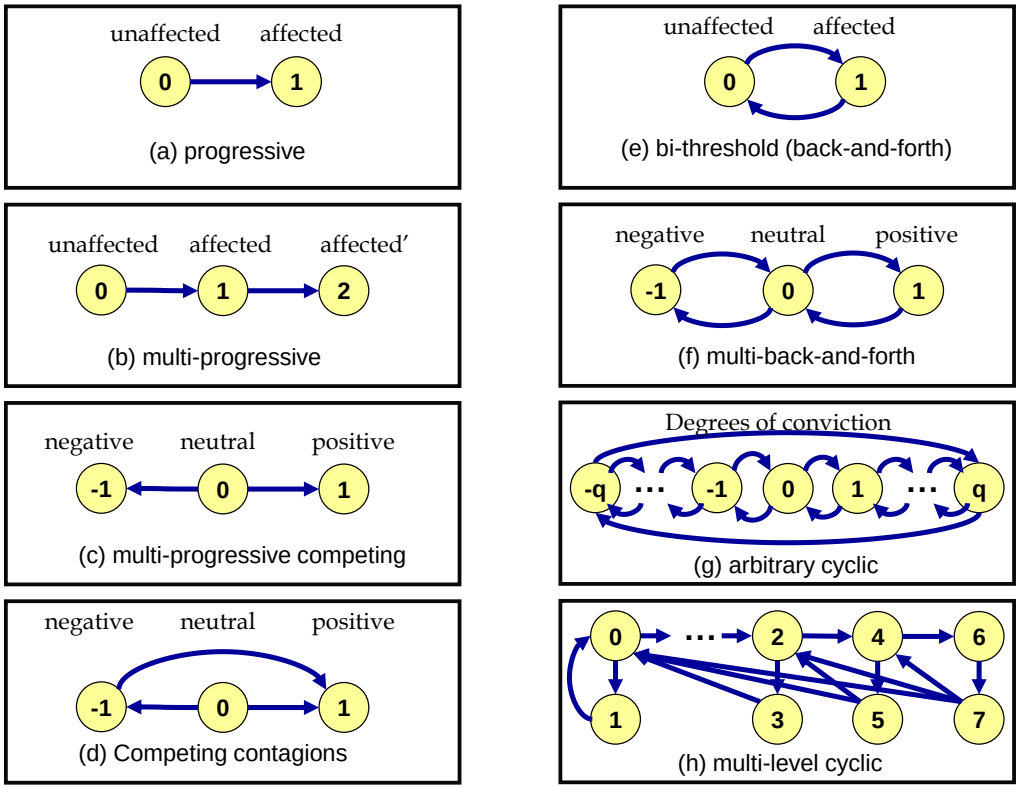


Figure 1.3: Different types of STMs employed in various domains to represent social behaviors. A reference to an example implementation is given for each. (a) Progressive [172], ratcheted [180], irreversible systems [105] modeling techology adoption [132]. (b) Progressively greater levels of conviction as in commitment to a revolution [209]. (c) Two mutually exclusive terminal states [57]. (d) Competing contagions [244]. (e) Voter models [145]. (f) Back-and-forth systems with neutral state [311]. (g) Arbitrary levels of participation [107]. (h) Multiple levels of conviction [64].

are known variably as **progressive** [166], **ratcheted** [180], and **irreversible** [105], and these and other references (e.g., [65,67,127,132,133,277,281,329]) provide illustrative applications. Two of the most common STC employing this STM are the independent cascade (IC) model and the linear threshold (LT) model. In the IC model, an unaffected node v has one chance to become affected by each of its neighbors (the contagion transfer occurs with probability p). If v gets affected, it transitions $0 \rightarrow 1$. At the next time, v has one chance to affect each of its unaffected neighbors with probability p. Thereafter, v remains affected, but cannot affect others. In the LT model, each edge has a weight. A node v that is unaffected at a particular time step sums the weights on edges that it shares with affected neighbors. If the sum of these edge weights is greater than or equal to v's threshold θ_v, then v transitions $0 \rightarrow 1$. (A threshold model is described below.) In this model, an affected node repeatedly tries to affect its unaffected neighbors at each time step. Note that the IC and LT models possess very different dynamics even though the STM is the same.

A multi-progressive STM (Figure 1.3(b)), where there are multiple transitions, all leading to a terminal state, represents many epidemiological models [110, 159] such as susceptible-infectious-recovered (SIR) and susceptible-exposed-infectious-recovered (SEIR); the SEIR model requires four states. The STM also is used to model multiple levels of commitment for social contagions [224]. We elaborate on these two models to illustrate how STCs can be different for the same STM. In the SEIR model, where S, E, I, and R correspond to states 0, 1, 2, and 3, respectively, the transition $0 \rightarrow 1$ is typically threshold-based, as it is in the multiple levels of commitment model. However, the $1 \rightarrow 2$ state transition in the SEIR model is typically time based: a node remains in state 1 (i.e., incubating) for a particular length of time, which is disease-specific and can range from a day up to several months [222]. The $2 \rightarrow 3$ transition is also typically time based. In contrast, for the multiple levels of commitment model, all transitions are threshold-based.

Three-state progressive competing STMs (Figure 1.3(c)) may model participation in strikes and riots and adoption of fads, where a node can be in a neutral state and change state to one of two other states. The assumption is that once one commits, the position is permanent or irreversible. This model is implemented in [76, 142] for two competing contagions. A generalization to any number of competing contagions is given in [177] so that Figure 1.3(c) would be a star with center state 0.

The competing contagions model of Figure 1.3(d) is used in cases where a node can transition to an undesirable state (-1 in this case) and later be influenced to adopt a more desirable state (in this case 1). Or, a node may go directly to state 1 from the neutral state [57].

Back-and-forth models (Figure 1.3(e)) were identified by social scientists (e.g. [101, 281]), where a node can move back and forth between states 0 and 1, and hence can visit a state any number of times. Voter models [85, 145] fit this description. In the basic voter model, a node chooses at random a neighboring node and changes its state to that of the chosen neighbor. There are myriad variants on this process and voter models are widely used [33, 63, 111, 148, 229]. A mean field sociologically-inspired model where nodes can transition

back-and-forth between two states is given in [35]. The epidemiological SIS (susceptible-infectious-suseptible) model is another example [101], where a node can become susceptible again (state 0) after being infectious (state 1), once its contagious duration expires. The SIS model has been used for social contagions (e.g., [71]). However, no threshold-based contagion model has been put forth that takes into account the effects of network structure and assigns thresholds to both state transitions. One subject of this thesis is to devise and analyze such a bithreshold model of social behavior; see Chapter 7. Figure 1.3(f) shows a back-and-forth model with a neutral state 0. This model is used for the acquisition of one of two competing rumors, where state 0 is neutral, and one can relinquish their belief in either rumor and again become neutral [311].

Cyclic models are generalizations of back-and-forth models where the node state transitions contain cycles so that states can be revisited (e.g., [64]). Motivations for this type of model were provided in Section 1.3. The model in [3] can be viewed as an STM of the type in Figure 1.3(g) if the real-valued states in that work are binned; then any transition between two states is admissible. A competing contagion model [260] also employs this STM. The idea of multiple levels of immunity for human viruses [273] (Figure 1.3(h)) has also been applied to understanding addictive smoking behavior [314]. (Epidemiological models have been modified and applied to many different domains; e.g., revolutions and conflict [64], as well as transmission of computer viruses and hand-held device malware [75, 175, 270].)

A state transition criterion based on thresholds. Threshold models that describe the conditions under which a node changes state (and possibly to which state a node transitions) were motivated in Section 1.3 and earlier in this section. There are many other models, including voter models, pure probabilistic transitions, and other formulations that incorporate threshold-like ideas (e.g., [83, 166, 290]). Our goal here is to give some insight into the formulation of contagion models by explaining a threshold model, as these models are widely used; e.g., [67, 132, 166, 281, 329].

A threshold model assigns to each node v_i a threshold value $\theta_{i,jk}$for each state transition $j \to k$. Our focus here will be on one contagion and one state transition, and so we will drop the subscripts j and k. More concretely, one can also consider this description as applying to the model of Figure 1.3(a).

A state transition occurs for node v_i when the following equation holds

$$a_i \geq \theta_i \tag{1.1}$$

where a_i is a function that describes the **driving force** for state change and θ_i is a function that describes the **resistance** or **inertia** that has to be overcome for that state change to occur. We assume that θ_i is a constant and a_i is a function of v_i's current state and those of its neighbors. The equation can also describe to what state v_i transitions. A concrete instance for a progressive two-state model with the only state transition being $0 \to 1$, where the current and next states for v_i are respectively s_i and s_i', is

1. If $s_i = 1$, then $s_i' = 1$ (i.e., if the current state of v_i is 1, then v_i remains in state 1).
2. If the number a_i of neighbors of v_i in state 1 is $\geq \theta_i$, then $s_i' = 1$. (Note: $s_i = 0$.)
3. Otherwise, $s_i' = 0$.

Also, since all models considered in this work are discrete models—discrete in time and discrete in state—when we write *state change*, *next state*, or similar, we are assuming for this discussion that the current state corresponds to time t and that the new state corresponds to time $(t + 1)$.

We now present some often-used contagion classes. A **simple contagion** is one in which $\theta_i = 1$ for all v_i in a graph. A **complex contagion** means that at least one node v_k has $\theta_k > 1$ [67]. As stated in [67], "Complex contagions require social affirmation from multiple sources." A **generalized contagion** [101] incorporates features of the first two, but in addition, nodes have different **status** and thereby contribute different amounts of influence, or more generally, contagion, to their neighbors. These different amounts of contagion can be node-dependent (e.g., through homophily; i.e., nodes are influenced more by neighbors with similar attributes [21, 69, 223]), relationship-dependent (i.e., edge-dependent), time-dependent, or any combination thereof. In these cases, a_i of Equation (1.1) would be the sum of all contagion amounts contributed by all neighbors of v_i.

The seemingly straight-forward differences in contagion type (i.e., simple, complex, generalized) have large ramifications for system dynamics. For example, it is known that the *weak link* effect is an important phenomenon in providing nodes with new information that they do not receive from the nodes with which they interact regularly [131]. The weak link effect essentially says that if a node has most of its interactions with the same group of people who are also (reasonably) well connected to each other, then all nodes in the group will basically have the same information—because they all interact. However, infrequent interactions (i.e., weak interactions) with people outside of this group provide pathways for new information (i.e., contagion) to flow into the group. It has been demonstrated that the weak link effect is far less prominent in complex contagions than in simple contagions [67]. Part of our work proves other differences between simple and complex contagions.

Other models. There are many types of models beyond those in Figure 1.3, including those tailored specifically to a particular application. Table 1.4 provides some *categories* of contagion models, some of which were mentioned here, and others are provided in Appendix A.

Another aspect of our work is a modeling framework that enables simulation of *any* STM and *any* collection of STC, so long as we preserve the notion that a node v_i's state update is a function of the state of v_i itself and those of its neighbors. We need this versatility to enable modeling of the various problem domains described at the outset of Chapter 1, the different types of social behaviors discussed there and in Section 1.3, and the different types of models explored in this section.

Table 1.4: Sampling of categories of contagion models. Generally, there are many works utilizing different models within each of these categories, but we list only one here. See Appendix A for many more models.

Number	Category	References
1	Edge weight-based thresholds.	[166]
2	Time-based thresholds.	[124]
3	Simple contagions.	[132]
4	Complex contagions.	[67]
5	Generalized contagions.	[101]
6	Connected components models.	[316]
7	Learning models.	[212]
8	Voter models.	[324]
9	Statistical physics models (e.g., Ising model).	[63]
10	Joint action.	[83]
11	Multiple complementing contagions.	[109]
12	Multiple competing contagions.	[57]
13	Multi-mechanisms and vector-valued vertex functions.	[191]

Actually, our modeling environment can handle more general problems, including **joint action models**, where a collection of nodes change state simultaneoulsy and cooperatively [83] (see the fourth box in Figure 1.1). One of the results we show is how computation time can vary greatly for simulations of contagion dynamics using different models in Figure 1.3, which is important because one of our interests is to simulate large populations (e.g., 100s of millions of nodes and billions of edges). The basic model we use is reviewed in Section 1.5.

1.5 Graph Dynamical Systems

Models in all of the categories in Table 1.4, and models in Appendix A can be treated in a unified fashion using the formalism of graph dynamical systems (GDSs), also known as **generalized graph automata** or **generalized cellular automata** [28]. The GDS model is presented here, and an extended GDS is provided in Chapter 2. For additional descriptions of discrete dynamical systems, see, for example, [30, 194, 211, 232, 234]. We begin with the necessary definitions.

1.5.1 Basic GDS

Let $\mathbb{B}$ denote a finite domain. A **Graph Dynamical System** (GDS) $\mathcal{S}$ over $\mathbb{B}$ is specified as a triple $\mathcal{S} = (G, \mathbf{F}, W)$, where

(*i*) $G(V, E)$, an undirected graph with node set V and edge set E and where $n = |V|$ and $m = |E|$, represents the underlying social network over which a contagion propagates;

(*ii*) each node of G has a state value from $\mathbb{B}$;

(*iii*) $\mathbf{F} = \{f_1, f_2, \ldots, f_n\}$ is a collection of functions in the system, with f_i denoting the **local vertex (transition) function** associated with node v_i, $1 \leq i \leq n$ ($\mathbf{F}$ is called a GDS **map** with $\mathbf{F} : \mathbb{B}^n \longrightarrow \mathbb{B}^n$); and

(*iv*) W specifies a scheme for updating node states.

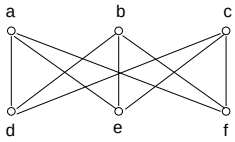
Each function f_i specifies the local interaction between node v_i and its neighbors in G. The inputs to function f_i are the state of v_i and those of the neighbors of v_i in G; function f_i maps each combination of inputs to a value in $\mathbb{B}$. The value computed by f_i is the next state of node v_i. To express this formally, let s_i and s_i' denote respectively the current and next states of node v_i, $1 \leq i \leq n$. Also, let the neighbors of node v_i be v_{i_1}, v_{i_2}, ..., v_{i_r} for some $r \geq 0$. Then

$$s_i' \;=\; f_i(s_i,\, s_{i_1},\, s_{i_1},\, \ldots,\, s_{i_r}). \tag{1.2}$$

When the domain $\mathbb{B} = \{0, 1\}$, one class of functions that are used as local transition functions in many social systems are the **threshold** functions (see e.g., [28, 67]). Given an integer θ, a θ-threshold function has the value 1 if at least θ of its inputs are 1; otherwise, the value of the function is 0. When $\theta = 1$, the GDS models a system which propagates a simple contagion; when $\theta \geq 2$, the corresponding system propagates a complex contagion.

A **configuration** or **system state** $\mathcal{C}$ of a GDS at any time is an n-vector $(s_1, s_2, \ldots, s_n)$, where $s_i \in \mathbb{B}$ is the value of the state of node v_i ($1 \leq i \leq n$).

Several different sequencing schemes for updating the states of nodes have been proposed in the literature (e.g. [28, 232]). We discuss three of these schemes here. In the **synchronous update** scheme, all the local transition functions are computed in parallel and the states of the nodes are also updated in parallel. In the **sequential update** scheme, a permutation π of the nodes is specified; in each time step, computations of the local transition functions and the state updates are carried out in the order specified by π. In the **block-sequential update** scheme, which generalizes the other two schemes, a partition of the node set V into a certain number $k \geq 1$ of blocks is produced and a total ordering on these blocks is specified. Suppose the blocks and the ordering are denoted by the sequence $\langle V_1, V_2, \ldots, V_k \rangle$. In each time step, the blocks are considered in the specified sequential order while the local transition function computations and state updates of all the nodes in a block are carried out synchronously. When the number of blocks is one, the block-sequential scheme reduces to the synchronous update scheme; when the number of blocks is n (i.e., each block has exactly one node), the result is a sequential update scheme. Beyond these three is a variety of other schemes, such as **unfair word** ordering, in which a node is never updated [232].



Note: Each configuration of the system has the form $(s_a, s_b, s_c, s_d, s_e, s_f)$, where s_a through s_f denote the the states of nodes a through f respectively.

Figure 1.4: An example of a graph dynamical system.

A different sequence may be specified at each time step; e.g., at each time step, a different permutation of the nodes can be utilized.

If a deterministic GDS (i.e., a GDS in which all local vertex functions are deterministic) has a transition from configuration $\mathcal{C}$ to configuration $\mathcal{C}'$, we say that $\mathcal{C}'$ is the **successor** of $\mathcal{C}$ and that $\mathcal{C}$ is a **predecessor** of $\mathcal{C}'$. A configuration $\mathcal{C}$ is called a **fixed point** if the successor of $\mathcal{C}$ is $\mathcal{C}$ itself. For a given graph G, collection of vertex functions $\mathbf{F}$, a state space where there are $q = |\mathbb{B}|$ states, and synchronous update, the total number of transitions that must be computed to describe all transitions from $\mathcal{C}$ to $\mathcal{C}'$ is q^n. For sequential update, the number of configuration transitions increases to $n!\ q^n$, where $n!$ is the number of permutation update sequences. The set of all such transitions for *one* update permutation is called the **phase space** of the GDS, and forms a directed graph. For graphs of smaller size (i.e., small n, on the order of 100 nodes), computing the phase space is computationally prohibitive, and one resorts to sampling the space of all such configuration transitions.

We now present a simple example of a GDS to show that different update schemes can lead to different dynamics.

Example: Consider the graph shown in Figure 1.4 with domain $\mathbb{B} = \{0, 1\}$. Suppose the local interaction function at each node is the **3-threshold** function. Thus, the system models the diffusion of a complex contagion. For the three update schemes examined below, let the initial configuration (at time $t = 0$) of the system be $(0, 0, 0, 1, 1, 1)$.

The dynamics for three update schemes are provided in Table 1.5. Under the synchronous update model, the system cycles between the two configurations $(0, 0, 0, 1, 1, 1)$ and $(1, 1, 1, 0, 0, 0)$ starting at $t = 0$. Under the sequential update model where the permutation π is $\langle v_a, v_b, v_c, v_d, v_e, v_f \rangle$, the system reaches the fixed point $(1, 1, 1, 1, 1, 1)$ in one time step. Finally, when we consider the block-sequential update model, where the nodes are partitioned into three blocks with $V_1 = \{v_a, v_d\}$, $V_2 = \{v_b, v_e\}$ and $V_3 = \{v_c, v_f\}$ and the permutation of the blocks is $\langle V_1, V_2, V_3 \rangle$, the system reaches the fixed point $(0, 0, 0, 0, 0, 0)$ in two time steps.

We now detail one system state transition, from the configuration at time $t = 0$, $\mathcal{C}(0) = (0, 0, 0, 1, 1, 1)$ to the configuration at time $t = 1$, $\mathcal{C}'(1) = (1, 1, 1, 1, 1, 1)$, for the sequential update and permutation $\langle v_a, v_b, v_c, v_d, v_e, v_f \rangle$. Node v_a is updated first. Its current state is $s_a = 0$. It has three neighbors in state 1 in $\mathcal{C}(0)$, namely v_d, v_e, and v_f, and because its threshold criterion ($\theta = 3$) is satisfied, it transitions to $s'_a = 1$. Node v_b is updated next and

its current state is $s_b = 0$. It has the same three neighbors in state 1, and hence $s'_b = 1$. (Note that if the state of v_a was an input to f_b, then the input value would be 1, reflecting the state update for v_a.) The same neighborhood conditions exist for v_c and hence $s'_c = 1$. For node v_d, it now has three neighbors in state 1, namely v_a, v_b, and v_c, plus itself, for a total of four nodes in state 1. This exceeds the node threshold, so it remains in state 1 (i.e., $s'_d = 1$). The same arguments hold for the state updates for v_e and v_f so that $s'_e = 1$ and $s'_f = 1$. Hence $\mathcal{C}'(1) = (1, 1, 1, 1, 1, 1)$, which can be verified as a fixed point. □

Table 1.5: System configurations versus time for three update schemes, giving rise to different long-term configurations (attractors).

Time	**Synchronous**	**Sequential**	**Block-Sequential**
0	(0,0,0,1,1,1)	(0,0,0,1,1,1)	(0,0,0,1,1,1)
1	(1,1,1,0,0,0)	(1,1,1,1,1,1)	(1,0,0,0,0,0)
2	(0,0,0,1,1,1)	(1,1,1,1,1,1)	(0,0,0,0,0,0)
long-term	2-cycle	fixed point	fixed point

1.5.2 Practical considerations

Connection of vertex functions to previous description. The transition criterion of Equation (1.1) in Section 1.4 is a particular representation of the general expression of Equation 1.2.

Interpretation of dynamics results. The above example shows that for the same underlying social network (Figure 1.4), different update schemes may produce completely different dynamics. The importance of these differences can be appreciated by considering a specific complex contagion such as a protest movement, with state 1 (0) indicating a person who has joined (not joined) the movement. From the results of Table 1.5, for the specific initial state, under the synchronous update model, at any time, half the population has joined the movement. Under the sequential update model, eventually the entire population is part of the movement, and remains so, while under a block-sequential update model, no member of the population is part of the movement in the long-term (i.e., the movement dies).

Effects of update schemes. Different update schemes are of practical importance. Synchronous update is natural when all nodes react to their surroundings at each time. Virus propagation based on daily contact is a natural example. It is attractive from a computational standpoint because it makes maximum use of parallelism in distributed and/or multithreaded simulation. Sequential update is useful because some algorithms have provable performance only for this update scheme (e.g., [160]). Hence, from a computational standpoint, one pays a penalty in performance in exchange for provable behavior. Block sequential may be used when all nodes of a given *type* execute simultaneously. For example, in an organization, executive agents may execute first in making a decision, and then successive

levels of management and workers may implement or react to the decision sequentially, with all members at one level acting simultaneously.

Benefits of combining theory and simulation. This example highlights another benefit of our modeling approach. That is, there are many theoretical results for GDS (e.g., [232]) and these can be used to guide studies or verify aspects of computed dynamics. For example, in [187], it is proven that the long-term dynamics of (*i*) the synchronous update must be either a fixed point or a 2-cycle, here it is a 2-cycle; and (*ii*) the sequential update must result in a fixed point. Our example results in Table 1.5, which for larger sized graphs would be generated with simulation, conform to these theoretical results. Combining simulation and theoretical results provides greater insights into system dynamics.

Control of dynamical processes. Of general interest in dynamical systems is how to **control** their behavior (i.e., alter the dynamics in a prescribed way to achieve some objective), and this includes contagion dynamics [39,204,319]. For example, if a harmful rumor is spreading, a question is: how can we alter the graph or node behaviors in order to slow down or stop the spread of it? In the case of a social movement, how might one take action to increase/decrease the probability of it consuming a large fraction of a population? Examples are plentiful. Typically, contagion processes are controlled using any of a number of options: modifying the graph (adding or removing nodes and edges, or changing node connectivities), changing properties of nodes and edges, changing the local vertex transition functions, or changing parameters in those functions. Any of these types of alterations is an **intervention**: a change in the GDS. Thus, a GDS provides a way to reason about interventions. We mention this for two reasons. First, we investigate particular intervention strategies for different dynamics models in this work. Second, the modeling framework that we develop as part of this work, to perform simulations of contagion dynamics, is particularly flexible and general in its ability to modify vertex functions to incorporate interventions. Our framework is also useful in modifying graph structure during the simulation.

1.6 Modeling Frameworks and Simulation

One aspect of our work is the development of a modeling environment that implements the concepts of GDSs described in Section 1.5 and thus simulates contagion dynamics on populations. In this section, we review other modeling environments and simulators.

We first address general types of simulation, from continuous time to discrete event and discrete time. We then describe some of the major issues and advances in parallel discrete event simulation (PDES), which is used for agent-based modeling, in addition to other types of simulation. We present some of the hardware platforms used for agent-based modeling and simulation (ABMS). While providing references to many simulation frameworks, we focus on some simulation issues that are particular to the types of problems of interest to us, illustrating how these problems present unique challenges.

There are wide variations in types of simulations. Physics codes that compute dynamics, such as finite element software, most often model continuous time [342] in (very small) discrete time steps to capture fine-grained temporal evolution. Time stepping schemes are routinely used to vary the time increment in simulations according to rate changes of important physical quantities. But all time increments are generally very small. PDES uses concurrency across processors to jump forward to times at which some action (i.e., event) takes place, skipping times at which no event occurs. Time jumps are often much larger than those in finite element codes, for example. PDES can be used to model physics phenomena (e.g., [301]), and is also used in agent-based systems. A unified view of continuous time and discrete event simulation is provided in [24]. Here, we focus on ABMS systems. Overviews of some differences between general PDES and PDES applied to ABMS are given in [72, 154]. The dominant difference is that in ABMS, agents are autonomous and intelligent, whereas agents are often not used in PDES.

Common dimensions along which agent-based simulators are divided are discretization (by time or events) and scheduling of simulation computations (conservative and optimistic) [61, 118, 257, 263, 326]. Some simulators and frameworks are hybrids, either using both simultaneously [27] or switching back and forth between them at runtime [262]. Parallel discrete event simulators (PDES) require some scheme to time order events properly, to ensure correctness, and this is accomplished through distributed scheduling (i.e., synchronization across distributed processes of the simulation). This can be realized through a guaranteed minimum lookahead time (the sum of the least current time on any processor plus the global lookahead time), or conservative [73, 74] or optimistic scheduling [61, 62, 157, 263]. A valuable reference list to sources beyond those in this document is provided in [61]. In conservative scheduling, a process of a simulation advances time only when it is sure that no event at an earlier time can be received from other processes. In optimistic scheduling, a process executes events in time order according to its current set of events. If events from earlier times are received later from another process (i.e., if "past" events are received), the system reverts back to an appropriate earlier time, obtains the saved state at that time, and resumes computation from that point. Two of the most common themes are rollback and reverse computation. In the former, system state is preserved at prescribed intervals, and system state reverts back to the saved state at the greatest time that is less than or equal to the past event's time. In the latter case, computations from the current time are reversed and execute back to the point of the past event's time, and simulation again marches forward from that point. Reverse computation has the advantage of requiring less memory for state saving, and is still widely considered as state-of-the-art. Different synchronization mechanisms are experimentally evaluated through performance evaluation, for large simulations (i.e., those involving thousands of processors) in [61]. They find that for roughly 1000 processors, conservative scheduling can deliver performance comparable to optimistic scheduling, but that for larger numbers of processors—up to roughly 10000—optimistic scheduling has increasingly better performance.

In our work, we use discrete time simulation, which is the appropriate paradigm for GDS. For

GDS problems, discrete event simulation would reduce to discrete time simulation because in GDS, all state update functions are executed at every time step and the time steps occur at regular intervals. Discrete time simulation obviates the need for the infrastructure and event ordering computations of discrete event semantics. Discrete time is also used by other ABMSs; e.g. [36, 326].

We now narrow the focus to agent-based simulators. Agent-based simulations are run on various hardware. These include serial codes on single processors [294], commodity (multicore) clusters [36, 91, 154], supercomputers of up to 65,000 cores or more [1, 26, 91, 265], internet-based centralized systems that use remote clients to carry out computatons [154, 256, 257], graphics processing units (GPUs) [264], and hybrid systems of GPUs and multicore processors [1]. ENSIM, our modeling environment, is designed to run on multi-core processors of commodity clusters.

There are many agent-based simulation platforms, ranging from smaller scale with feature-rich pre- and/or post-processors (e.g., [294,306]) to bare bones very high performance systems that execute on tens of thousands of processors (e.g., [265]). Many simulators that span this spectrum are identified in [1, 210]; Collier and North [90, 91] also provide a valuable survey. In many simulation environments, a physical process handles many agents [154,257,262], but some systems use a dedicated process for each agent [121,276]. Some follow a master-worker paradigm [36, 154, 257] while others are peer-to-peer systems [262]. ENSIM is a peer-to-peer system. Steering simulations via runtime user interactions has also been investigated [37,154].

In our work, we focus on one class of problems: systems that can be represented by networked agents that therefore fit within the realm of graph dynamical systems. This choice is driven by the need to simulate a wide range of behaviors in the social sciences, but we have demonstrated that the modeling environment can be used for other domains, many of which were identified in the opening paragraphs of Section 1.1. Motivation for social modeling was also given in Section 1.1. Our choice of model also enables a lot of desirable framework features, such as flexibility in models, fast turn around times, and a single model interface, as described in Chapter 3.

The need for social simulations incorporating millions of nodes is also described in [264] and their work addresses agent systems of up to 10 million nodes that are simulated using graphics processing units (GPUs). Simulations on multicore processors involving 1 billion agents are presented in [1]. In these latter two works, we crucially note that the simulations are for, essentially, cellular automata, where the population is a grid of agents that only interact with their nearest neighbors. These types of regular problems are much easier problems to parallelize because of regular communications patterns that can also exploit data locality. In contrast, graph problems, where connectivity among nodes can be arbitrary, give rise to irregular, more expensive, communications patterns that cannot always exploit locality. Moreover, those simulations do not require storage of a graph to identify agent (node) interactions, which, as we demonstrate, can be in the billions of edges.

Other simulators that scale to 100s of millions to billions of nodes (e.g., [258, 265]) do so

using agent-location graphs, where agents travel to locations and interact if they are present at the same location at the same time. Again, this significantly reduces data storage requirements because the approach obviates the need to store graph edges, as required in network-based modeling environments such as ENSIM. While enabling large scale simulation, these approaches lack the granularity to uniquely specify pairwise interactions—direct influence, information flow, peer pressure—required for many social interactions, as argued in myriad works (see e.g., [48,117,129,132,205,217,220,249,281,318]). Our simulation environment is currently limited by memory: the largest network that we have been able to load into memory on our compute cluster is a 2.8 billion edge graph.

A discrete time simulation framework called BRACE based on MapReduce is discussed in [326]. BRACE provides programmability (with interactions being specified through a high-level language called BRASIL) and scalability to behavioral simulations. We could find no information on maximum problem sizes that BRACE can handle, in terms of number of agents. In [326], scalability is measured by throughput; the BRACE system is capable of processing 1-2 million agents per second. Although model dependent, ENSIM can routinely process 1 million nodes per second, even under heavy loading scenarios where a majority of the 100 million nodes in a network *change state* at the same time, thus generating large computation and communication loads.

Another simulation framework for the social sciences is called FLAME [86,97]. Their system is based on the concept of Communicating Stream X-Machines, which is essentially a GDS (one can translate back and forth between a Stream X-Machine and a GDS, and both can simulate a Turing machine). Their software implementation is essentially a conservative event-based scheme except that agents block on the reception of input messages so that the agent does not execute its vertex function until all input messages are received. Hence, dependencies among tasks are strictly enforced. It is an MPI task-based system. The provide data for a 500000-agent system. Interestingly, they represent the dynamics of agents in the form of finite state machines (FSMs), but do not allow cycles in their FSMs.

A high performance version of Repast [91] has been developed. It seeks to provide a feature-rich environment for modeling, along with high performance. It is a discrete event simulation engine with conservative scheduling. It also enables agents to interact in two types of environments. The first is a network representation of agents, as is studied herein. The second is a two-dimensional spatial grid where agents move within the grid. Their MPI implementation has been run with up to 10^9 agents on 32,768 cores.

Pandora [333] is a cloud-based HPC social simulation framework. They focus on low-density simulations, meaning agents numbering in the tens of thousands. Moreover, because their simulation is spatial, compute processes only communicate with their nearest neighbors, enabling more efficient communications through data locality. They implement a recurring three-step state evolution process in which, at each time increment, agents gather information about their environment in parallel, select an action in parallel, and then update their states sequentially to avoid clashes. Pandora, written in C++, has an interface to Python to service

non-computing experts, among whom Python is a popular programming choice.

Many simulators are developed for particular domains, and cleverly exploit problem semantics to achieve extreme performance. For example, a simulator discussed in [36] for modeling the spread of viruses can scale to billions of nodes. It is also very fast. For example, it is able to model a contact network of New York City, consisting of 18 million nodes and 480 thousand edges, simulating 50 diffusion instances of 300 time steps each, in a total of 81 seconds on 72 processors. However, the design that enables this extreme scaling also limits its versatility; it can execute only a relatively small number of dynamics models. ENSIM can perform all the simulations in [36], and it also compares favorably with respect to speedup. However, ENSIM is slower because it cannot exploit problem semantics, which has ramifications for data storage, control flow, and messaging.

No one simulation environment provides functionality for all types of problems. Our work focuses on problems whose structure can be represented by graphs that describe interacting agents. Our contributions include expressiveness of multi-contagion dynamics made possible by implementing a universal model of computation. ENSIM enables fine-grained and elaborate specification of node and edge (i.e., pairwise interaction) attributes through versatile data structures. Dynamics models can be quickly combined with ENSIM to produce domain-specific simulators that scale to graphs with 100 million nodes and 2.8 billion edges.

1.7 Blocking Diffusion

Analyzing social networks has become an important topic in the data mining community [9,71,103,166,167,275,302] and remains so in the social networks community [203,319,321]. Many researchers have studied diffusion processes in social networks. Some examples are the propagation of favorite photographs in a Flickr network [70], the spread of information [133, 176] via internet communication, the effects of online purchase recommendations [196], formation of online communities [286], hashtag propagation in Twitter [277], and virus propagation among computers [259]. In some instances, models of diffusion are combined with data mining to predict social phenomena; e.g., product marketing [103,275], trust propagation [134], and epidemics through social contacts [218]. Furthermore, coupled processes of network and dynamics evolutions are studied with comparisons against experimental data [68,69]; see [323] for an overview.

We are interested in both simple and complex contagions, as defined in Section 1.4. Some of the most recognizable simple contagions are disease propagation [159,259] and computer virus transmission [158]. Complex contagions include phenomena such as diffusion of innovations, spread of rumors and worker strikes, educational attainment, fashion trends, and growth of social movements; see [67]. For example, in strikes, mob violence, and political upheavals, individuals can be reluctant to participate for fear of reprisals to themselves and their families. It is safer for one to wait for a critical mass of one's acquaintances to commit before com-

mitting oneself. A common interpretation is a safety-in-numbers argument [188, 205, 290]. Crucially, recent data mining analyses have demonstrated that a number of applications can be modeled through complex contagion dynamics on appropriate social networks; examples include online DVD purchases [196], teenage smoking initiation [139, 182], Twitter data spread of contentious issues [277], the propensity to join a mass movement in Spain 2011 and tweet about it [127], the spread of health-related information [68], and the likelihood of joining LiveJournal [172] and Facebook [316]. Another motivation for our work is from recent quantitative work [65,67] showing that simple contagions and complex contagions can differ significantly in behavior.

Devising strategies to thwart contagion propagation is important because such methods have wide applicability in several domains of network dynamics. Examples include thwarting the spread of sensitive information that has been leaked [71], disrupting communication of adversaries [17], marketing to counteract the advertising of a competing product [103, 275], calming a mob [132], or changing people's opinions [105]. Inhibiting diffusion is one aspect of a broader goal of *controlling* diffusion in complex networks as advocated in [204]. These strategies to date have focused overwhelmingly on simple contagion dynamics (an exception being [65]).

Our review centers on node schemes to block contagion propagation simply because this is the dominant area of past research, but thereafter we discuss other means of blocking contagions, such as blocking edges (so that contagion cannot be transmitted) or introducing a second contagion to counteract the effects of the first one. For an overview of these and other schemes, see [319].

1.7.1 Node Blocking

The model of Figure 1.5 illustrates the social dynamics considered in many node blocking studies. The sole state transition is shown at the top, where state 0 (respectively, 1) corresponds to a node not possessing (respectively, possessing) a contagion. A node transitions from state 0 to 1 when its threshold is met, but it cannot transition from 1 to 0. This is a progressive model [172]. Below this graphic is a 6-node graph with nodes v_1 and v_2 initially (at time $t = 0$) in state 1; these are the **seed** nodes. In the first time step, for complex contagion diffusion where $\theta = 2$, node v_3 changes to state 1 because two of its neighbors (v_1 and v_2) are in state 1. At the next time instant, $t = 2$, v_4 has two neighbors (namely, v_2 and v_3) in state 1, and therefore it transitions to state 1. No more transitions are possible, and we say the **spread size**—the final number of nodes in state 1—is 4. The system configuration $\mathcal{C} = (s_1, s_2, s_3, s_4, s_5, s_6) = (1, 1, 1, 1, 0, 0)$ is a fixed point. Now we examine the effects of a **frozen** or **critical** node, v_3, at the bottom left in Figure 1.5, represented as a blue square. A frozen or critical node does not change its state; here it is frozen at 0. Consequently, it does not change to state 1. The fraction of critical nodes is $f = 1/6$. Furthermore, it can be seen that except for the two seed nodes, none of the other nodes changes to 1, because no

node has at least two neighbors in state 1. Thus, the effect of a single critical node, in this case, is to reduce the spread size by 50%, from 4 to 2. By comparison, for the same seed nodes, the simple contagion $\theta = 1$ diffusion spread size without the critical node is 6 and the spread size with the critical node is 5. Finally, for the progressive $0 \rightarrow 1$ state transition dynamics, freezing the state of v_3 at 0 is equivalent to removing it and all of its incident edges from the graph, as shown in the bottom right graphic. Node removal produces two connected components in the graph, with a giant component (i.e., the largest component) of size 4 nodes, which is a 33% reduction from that of the original graph. Other names for frozen nodes are failed [7, 65], obstinate [338], zealots [229], and blockers [135]. The key question is: how does one select critical nodes in order to minimize the spread size?

Contagion dynamics with critical nodes have been studied in several domains, including peer influence in youth behavior, repression of social movements, opinion dynamics, and social isolation in epidemiology [4,7,17,65,162,164,182,228,229,279,291,292,334,338,338]. Newman [238] provides references to at least a dozen more earlier works. For these applications, frozen nodes have physical significance. For instance, critical nodes in technology adoption represent nodes that will stand firm with an older technology, regardless of what others do. In a social context, it is often convenient to think of critical nodes as those that oppose the spread of a contagion. Other domains that use blocking nodes include mobile networks [284] and malware propagation in computer networks [54, 267].

The problem of removing nodes from a graph to minimize the contagion spread size eminating from a group of seed nodes for threshold-1 diffusion was first addressed in [110] from a complexity standpoint. Formally, the problem is: given a graph $G(V, E)$, an initially affected set I of seed nodes, a parameter C, and cost $c(v)$ for each node v, choose a set of nodes $S \subset V \backslash I$ having cost $c(S) = \sum_{v \in S} c(v) \leq C$, such that the size of the set $A_G(I, S) = \{v | v \text{ has a path from some } w \in I \text{ in } G(V \backslash S)\}$ is minimized. The set S contains the critical nodes and $A_G(I, S)$ is the set of nodes that could get affected by the seed nodes in I. The cost is that to convert a node to a critical node. This problem was shown to be NP-hard, and a bicriteria approximation algorithm was devised.

Perhaps the most widely used blocking method is, for a prescribed budget β on the number of critical nodes, setting the β nodes with the greatest degrees (equivalently, the greatest degree centrality) as blockers [7]; this is the **high degree heuristic**. There are two approaches to specifying high degree nodes. In one case, after the removal of a node and its incident edges, the degree distribution of the new network is calculated to identify the node with the next highest degree [7, 238]. In the second case, node degrees are not recomputed; the original degrees are used to rank the importance of nodes [65]. The effectiveness of removing nodes at random and removing high degree nodes has also been studied [94]. Hubs, or high degree nodes, in scale free networks have also been investigated as critical nodes using mean field theory [98] and simulation of computer worms [54].

There are variants on the high degree hueristic. In [88, 215, 268], a random neighbor u of a randomly chosen node v is set as a blocker, the idea being that in scale-free networks,

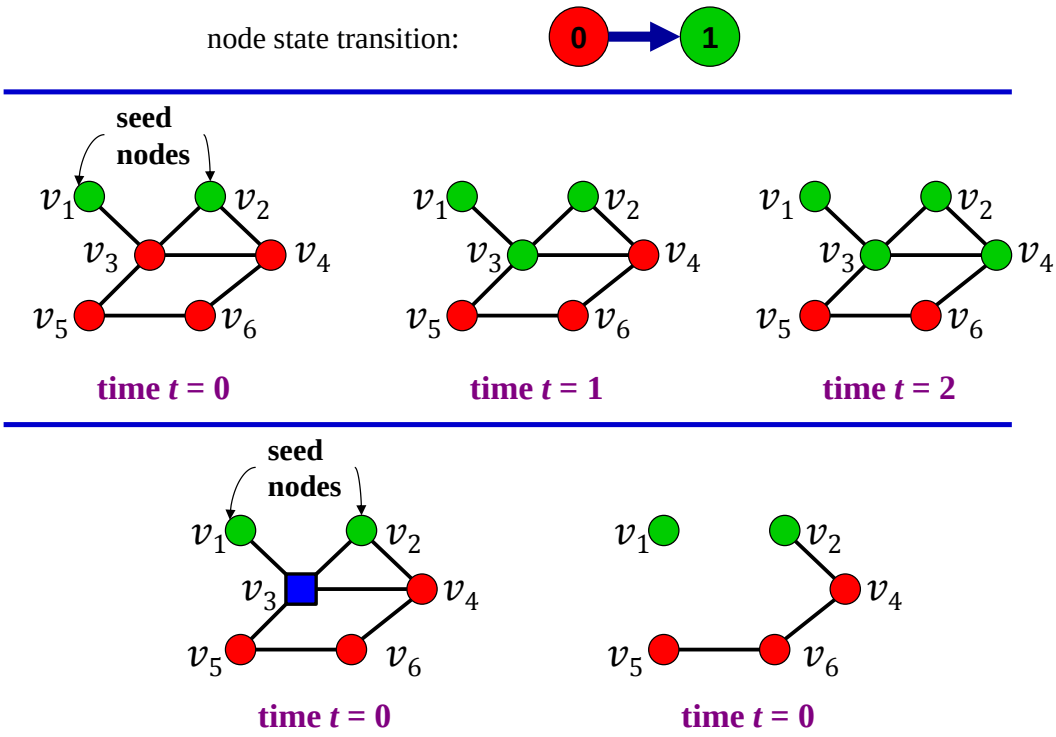


Figure 1.5: Example of contagion dynamics with node-based blocking. The figure is divided vertically into three parts. The upper graphic shows the sole state transition for a node, from state 0 to state 1, in a progressive threshold model. The middle part depicts $\theta = 2$ complex contagion diffusion on a 6-node network with two seed nodes. See the text for its description. In the bottom part, the left graphic shows the effect of designating v_3 (a square) as a critical node, meaning that it remains frozen in state 0. See the text for a description of the dynamics. The bottom right graphic illustrates that for the progressive model, freezing a critical node in state 0 is equivalent to removing it and its incident edges from the network.

randomly chosen nodes are most likely to be of low degree, and of their few neighbors, at least one is likely to be well-connected. The strategy of vaccinating one of the highest degree neighbors of a randomly chosen node in static networks is evaluated [146]. Another strategy is to pick a node v_i at random, and choose as a blocker a neighbor v_j with a maximal neighborhood that minimally overlaps that of v_i. There is an option used with both of these methods whereby instead of selecting the initial node at random, the initial node is the one vaccinated at the previous time step. Not surprisingly, these methods work better than the strategy of vaccinating a random neighbor of a randomly chosen node. One study finds that different methods work best across Erdos-Reny, small-world, and some real networks [146].

The goal of removing high degree nodes, in an ideal situation, is to fragment the graph; i.e., to break it down into different components. A variety of network-based candidate measures for identifying critical nodes under threshold 1 conditions are described in [47]. These are based on fragmenting a graph into roughly equal-sized fragments (components) in terms of numbers of nodes. One variant of the problems addressed in [47] is studied in [17]: that of minimizing the pairwise connectivity of nodes in a graph. They show the problem is NP-complete and provide a heuristic.

Betweenness centrality [115, 116, 328] or betweenness $c_b(v_i)$ of a node v_i is the fraction of shortest paths that pass through v_i and is computed as $c_b(v_i) = \sum_{v_j, v_k \in V, v_j \neq v_k} \frac{\sigma_{v_j v_k}(v_i)}{\sigma_{v_j v_k}}$ where nodes v_j and v_k are distinct, and are distinct from v_i, $\sigma_{v_j v_k}$ is the number of distinct shortest paths between v_j and v_k, and $\sigma_{v_j v_k}(v_i)$ is the number of those paths passing through v_i. Efficient algorithms for computing betweenness are given in [53,240]. [241] describes a variant called random-walk betweenness where all paths, not just shortest paths, are considered based on random walks (similar to [245]). Tunable algorithms for computing betweenness (as well as degree centrality and closeness centrality) by incorporating edge weights and/or the number of edges on a path (equivalently, the number of intermediate nodes on a path) are described in [252]. Closeness centrality of a node v, the inverse sum of shortest distances to all other nodes from v, has also been studied [116, 240].

Eigenvector centrality and its variants [44,45,252] are popular methods to identify important nodes in a graph. The goal is to rank the nodes in importance; here, importance is the ability of a node to block diffusion. Let A be the adjacency matrix for a graph G such that $A(i, j) = 1$ if there exists an undirected edge (v_i, v_j) in G and 0 otherwise. Let $A\lambda = \lambda w$, where (λ, w) is an eigenpair (eigenvalue, eigenvector) of A. Let w_1 be the dominant eigenvector of A; i.e., the eigenvector associated with the greatest eigenvalue. The ith component of w_1 is the eigenvector centrality score for v_i. Hence, by computing the dominant eigenvector (for which all components are by construction > 0), we obtain the centrality scores for all nodes. These, then, can easily be ordered. Power iteration can be used to compute the dominant eigenvalue (and eigenvector) [310]. The method also extends to weighted graphs, where the entries in matrix A are edge weights. Bonacich [44,45] describes intuition behind eigenvector centrality as a measure of popularity or numbers of friendships. Another interpretation is that a node's centrality is a (weighted) sum of the eigenvector centralities of its neighbors,

which are updated iteratively. This latter interpretation is related to those for the PageRank and HITS methods, described next.

The PageRank algorithm [55, 253] ranks nodes of a graph in order of importance, where importance of node v_i is determined by the number of directed edges into it and the importance of each node that links to it. The directionality of an edge (v_j, v_i) is from the tail node v_j to the head node v_i. Made famous by its use in the Google search engine, the pagerank (PR) of a (head) node v_i is given by $PR(v_i) = \frac{1-d}{n} + d \sum_{v_j \in In(v_i)} \frac{PR(v_j)}{d^{out}(v_j)}$ where n is the number of nodes, In is the set of tail nodes that form directed edges to v_i, $d^{out}(v_j)$ is the out-degree of tail v_j, and d is a discounting factor often set to $d = 0.85$. The $PR(v_i)$ values are the entries of the dominant eigenvector (the eigenvector associated with the largest eigenvalue), and thus the power iteration method can be used to compute pageranks iteratively. Further, analysis shows that convergence is fast [141].

The hyperlink-induced topic search (HITS) method [173] was originally used to rank web pages, as was PageRank. First, a text search is performed to identify a set of pages that best match the search text string (when the application is to find the most important web pages). Let this set of pages (nodes) be U_1. Let U_2 be the set of all nodes to which any node of U_1 points. Let U_3 be the set of nodes (or some subset of size k thereof) that point to any node of U_1. Then $U = \bigcup_{i=1}^{3} U_i$. The subgraph G' of the original graph G that is induced by the set U of nodes (called the base set) is operated on in the following. HITS ranks nodes in importance by using two scores: an authority score and a hub score. These denote, respectively, the propensity for a node to contain relevant information, and for a node to point to (i.e., form directed edges with) nodes that contain relevant content. The authority score for v_j is the number of hubs v_i that form directed edges (v_i, v_j), normalized by the 2-norm of the vector of authority scores. The hub score for v_i is the number of important (tail) nodes (i.e., authority nodes) that form edges with (head) node v_i; i.e., the number of edges (v_j, v_i), again normalized by the 2-norm of the vector of hub scores. The algorithm interates between computing authority scores and hub scores, with authority scores computed first, where each node is initially assigned authority and hub scores of 1. The algorithm is shown to converge [173]. HITS reduces to eigenvector centrality for undirected graphs [308].

The basis of another eigenvector-related blocking scheme was initially presented in [327], and later in [71,119]. Specifically, it was shown that for an SIS (susceptible-infectious-susceptible) model with birth rate β and death rate δ, the epidemic threshold τ is $\tau = 1/\lambda_1$. Here, λ_1 is the largest eigenvalue of the graph's adjacency matrix. That is, if $\beta/\delta < \tau = 1/\lambda_1$, then the epidemic will die out. If $\beta/\delta \geq \tau = 1/\lambda_1$, then the epidemic will be endemic. This motivates a blocking scheme as follows. In order to increase the epidemic threshold, to make it more difficult for the dynamics to become endemic, one should drive down the value of λ_1. Hence, a strategy is to choose the node that leads to the largest drop in the maximum eigenvalue.

The implementation is as follows. On the original graph, each node v_i is temporarily removed from the graph, one at a time, and the dominant eigenvalue associated with v_i is computed. The node v^* whose removal results in the least dominant eigenvalue signifies the node whose

removal results in the largest increase in the epidemic threshold and should be designated a blocking node. Node v_i is then permanently deleted from the graph. This process is recursively repeated to produce a ranking of node importance, in decreasing order. The method is detailed in [71]. A drawback of this method is that it is not practical for large networks, since for an n-node graph, $\sum_{i=1}^{n} i$ computations of the dominant eigenvalue are required to rank the nodes in importance for blocking. Indeed, in [268], only one network of 104 nodes was examined and only the top 10 nodes for removal were found. To overcome the problem of impractical computational times, an efficient eigenvector-based heuristic, called NetShield, was proposed to evaluate large graphs [308]. It was evaluated on three social networks, the largest of which was 2 million nodes, for speed of execution and was found to be a large improvement over two other methods. However, the effectiveness of the method remains somewhat unknown since the only data presented are for a small graph.

This line of research was extended again to characterize epidemic thresholds for a very broad range of epidemiological models that are 1-threshold [269]. They introduce an *effective strength*, s, of a model, based on birth rate, death rate, and other factors of a model and show that a contagion will not spread significantly if $s < 1$, where $s = \lambda_1 C_{VPM}$. For the SIS model, $C_{VPM} = \beta/\delta$, and the above expression is recovered. Other C_{VPM} expressions are compiled in [269]. Note how dynamics are now entering blocking methods through the epidemic threshold: up to this point in this review, all blocking approaches were based solely on network structure. We also use dynamics in our approaches.

Blocking of contagions on time-varying networks (i.e., networks where the existence of edges changes in time) is studied in [135]. They use a progressive 2-state probabilistic $\theta = 1$ diffusion model. They investigate over 15 graph metrics such as degree, diameter, pagerank, and betweenness centrality (adapted to time-varying networks) to identify blockers. The dynamics is a variant of IC, where a newly affected node will only try to affect each of its neighbors one time with a specified probability of success. However, if subsequently an edge between an affected node and an unaffected node is formed, then the affected node tries one time to affect the new unaffected node. For an affected node v_i and unaffected v_j, each time an edge is reformed, v_i will try to affect v_j, and hence there can be many trials between two nodes as the edge between them appears and disappears. One node is seeded in each diffusion instance. Simple measures such as the high degree heuristic worked as well as other more complicated measures in blocking diffusion. The epidemic threshold of time-varying graphs, where two graph structures (one for day and one for night) were alternated in diffusion processes on populations, was studied in [268].

Changing focus from simple to complex contagions, there is one complex contagion work that uses the high degree heuristic [65]. Random and high degree critical nodes were used with progressive threshold dynamics and $\theta = 2$ on two synthetic $d_{ave} = 4$ networks, one exponential-decay and one scale-free. It was found that diffusion was more difficult to block in an exponential-decay network than a scale-free one; the opposite ordering of the networks was found for simple contagion diffusion. We delve into blocking contagions in this setting in Chapter 6.

1.7.2 Edge Blocking

All of the work on edge blocking reported here is for simple contagion diffusion. The problem of removing edges from a graph to minimize the simple contagion ($\theta = 1$) spread size from a group of seed nodes (from which the contagion spreads) was first addressed from a complexity standpoint in [110]. Formally, the problem is: given a graph $G(V, E)$, an initially infected set I of seed nodes, a parameter B, and cost $c(e)$ for each edge e, choose a cut $(S, \overline{S})$ such that $I \subset S$, $|S| \leq B$, and the cost of the cut is minimized. This problem is shown to be NP-hard, and a bicriteria approximation algorithm is given.

An efficient heuristic is proposed in [341] for identifying edges to set as blocking; i.e., edges that will not transmit a contagion. For each edge, the product of the degrees of its incident nodes is computed, and edges are ranked in descending order. The k edges with the greatest degree product are removed from the network. Data on synthetic 1000-node scale-free networks show that this approach is more effective in reducing the epidemic threshold of the network than removing the k' nodes (and their k incident edges, $k' \leq k$) with greatest degree. They also show that for the SIS epidemic model, the edge deletion strategy is more effective than the node removal strategy in driving down the number of infected nodes in the long run (for the same number k of removed edges). This is not unexpected since all of the edges incident to highly connected nodes are not connected to other high-degree nodes, and thus, removing nodes entails removing some edges that are less important for containing contagion spread.

Work in [309] follows the same ideas as in [71] in decreasing the epidemic threshold, but now tries to achieve this through edge removal in a graph. The goal is to identify the edge whose removal from a graph results in the largest drop in maximum eigenvalue λ_1 of the adjacency matrix. By applying such a scheme successively, the k edges can be computed that should be removed to generate the greatest reduction in λ_1. Since brute force methods are prohibitive, an alternative scheme is required. The idea is to compute the left u and right w eigenvectors of λ_1 of the original adjacency matrix once, which gives an indication of a node's importance. Let $u(i)$ $(w(j))$ be the component of u (w) corresponding to node i (j). The importance of an edge $e_{i,j} = \{i, j\}$ is the value $u(i) \cdot w(j)$. They remove the k edges with the greatest product values. Note that this is similar in spirit to the approach used in [341]: use incident node parameter values to determine the importance of an edge. They then demonstrate, as in [341], that removing k edges is more effective than removing k' nodes (and their k incident edges).

In realistic weighted networks, the types of edges to remove to inhibit diffusion is not entirely clear. In a communication network [250], it was found that removing weak edges (versus strong edges) was more effective in breaking the network into small components, but for diffusion in weighted networks, it was the intermediate weight edges that transported the contagion that first infected the most nodes. This, again, is all under threshold-1 conditions.

1.7.3 Competing Contagions

Competing contagion methods propagate a second contagion to stymie or overturn the effects of a first contagion [13,34,57,76,142,163,184,185,244]. Most of these problems are all formally hard in the sense that no efficient algorithm exists that can provide results in reasonable time, and therefore algorithms and heuristics are devised that provide approximate solutions in acceptable times. Even some efficient approximation algorithms are not practical for large networks.

Blocking one contagion's propagation with the propagation of a second contagion (both using independent cascade models) is studied in [57]. It uses the STM of Figure 1.3(c). The system has a 3-state vertex state space, with all but one node initially in the neutral state. This one node is in the "bad" (-1) state, and is the seed for the propagation of the bad (first) contagion. At a later time, one or more nodes are seeded in the "good" state; state 1 of Figure 1.3(c). Once a node reaches state -1 or 1, it remains fixed in that state. The goal is for the good contagion to stop the bad one and it is shown that the problem of minimizing the final number of nodes in the bad state is NP-hard (this is their EIL problem). Multiple problems are investigated. In one version, the bad contagion propagates stochastically while the good one spreads deterministically (i.e, with probability 1). This EIL problem is shown to be submodular for particular sets of conditions. In another version that is not submodular, the good contagion also spreads probabilistically, but with different edge probabilities than those for the bad contagion. When the edge probabilities are the same for both contagions, the problem is again submodular. Submodularity motivates the use of a hill climbing-type algorithm [236], an approach made famous in the seminal work [166] that pervades a host of influence maximization algorithms. As in that previous work, and even to a greater extent, the basic algorithm in [57] employs simulation, which is expensive, and alternative methods are sought. Three such methods are devised and tested on networks. The problems are extended by investigating them under the condition of missing data, and it is demonstrated that the good contagion can be effective even when significant data (i.e., exactly which nodes are in the bad state and in the neutral state) are missing.

In [244], the problem of countering an undesirable contagion with a good contagion is also studied. However, they use a different STM than in [57]: they use the STM of Figure 1.3(d). That is, a node that possesses a bad contagion can be directly converted to the good contagion. They evaluate both IC and LT models. They formulate four problems, where the set of bad seed nodes can be known or unknown and the time at which a fraction β of nodes must be in the good state is either some specified finite time or infinity. They explore solutions to these problems. Clearly, one has more information in blocking a contagion if one knows the seed nodes, so this problem is easier to handle than the case where seed nodes are unknown.

In [142], a competitive LT model is used in much the same way that a competitive IC model is used in [57]. However, in [142], there are two edge weights for each edge (one for each contagion) and two thresholds for each node (one for each contagion). Hence, the conditions for propagation of each contagion are independent. The approach is interesting

because it estimates the effect of the blocking contagion—which eminates from the computed blocking seed set—using an influence maximization algorithm [79]. The work in [76] also uses the approach of [79] to develop an efficient method to estimate the most influential nodes to propagate a positive contagion under a modified IC model. Now, however, a negative contagion also propagates, and moreover, with a specified probability, a node receiving a positive contagion can be directly converted to the negative contagion. (An example is a person receiving a good recommendation about a restaurant, eats at it, but does not like it.) A differential equations-based model similar to the latter, for rumors, is provided in [163] wherein an ambivalent (state 0) person who hears a rumor from a person who believes it (state 1) may in fact believe it to be false and hence transitions to state -1.

The work of [50] for competitive LT diffusion is analogous to [76] for the IC model; i.e., in the presence of a propagating bad contagion, find a set of nodes that will maximize the spread of a good contagion. They examine different formulations, one of which is very close to that of [142], described above, but they study the problem of influence maximimization rather than blocking.

Other works include formulating the competing contagion problem as a game, with particular results for graphs represented as trees [34]. A 2-player game-based approach is formulated [177] and shows that the decision problem for each of player-1 and player-2 in selecting the best nodes to seed to maximize their information spread is NP-complete. Other game theoretic approaches include [313].

In [260], a generalized LT model is proposed that can accommodate any number of competing contagions, where a node can transition from any state to any other state across $k+1$ states, where there are k contagions and a neutral state. The analysis does not use simulation, as does many studies, but rather uses a graph coloring scheme to assign to nodes the most likely color (contagion). A system with two successive (non-competing) contagions is studied in [242].

1.8 Research Contributions

Our research contributions cover five areas of contagion dynamics.

1. We conceived, designed, and implemented a modeling environment for studying contagion processes at scale in populations that can be represented as networks. We prove that our modeling framework can simulate any dynamics that be described as a graph dynamical system. This includes multi-contagion dynamics and time-varying graphs. Interventions are also naturally handled. To date, we have implemented over 15 families of contagion models. A primary goal of our framework is human productivity: we demonstrate, informally, using metrics such as end-to-end solution time, that the framework can be used to quickly produce simulation results, often in a couple of hours

or days, which includes the time to design, construct, and verify new dynamics models that plug into the framework. Our current implementation scales to at least 100 processors (1200 cores); this exhausts our system resources. We can simulate populations with 100 million nodes and 2.8 billion edges. On such populations, execution time for 20 diffusion instances of 100 time steps each is about 2.5 hours. With some networks, we achieve a 1:1 speedup for increasing numbers of multicore processors; for other networks the speedup is only slightly less. We have demonstrated performance on large human contact networks and social media networks. All modeling work in this thesis uses our modeling environment.

2. We provide a provable bound on the maximum number of nodes to which simple and complex contagions can spread in progressive systems, which are widely used in the sociology literature. We also prove that this bound is achievable. The bound, based on k-cores, is efficiently computed even for large graphs. We show through experiments that the bound is tight for different social and synthetic networks, and identify conditions for which the bound is not tight.

3. We study problems of blocking and retarding the spread of progressive contagions. We show fundamental differences between simple and complex contagion blocking; e.g., differences in the complexities of identifying nodes to remove from a network in order to impede diffusion. Since efficient approximation algorithms with bounds on performance do not exist for complex contagion blocking, we devise and evaluate two blocking methods, and we show that they outperform five state-of-the-art methods.

4. We demonstrate through experiments that conventional wisdom on the ability of different classes of networks to support complex contagion dynamics need not always hold. We also show that dynamics on multiple graph instances from the same class can vary widely, which is a cautionary note for studies that utilize only one graph instance to draw conclusions about classes of graphs.

5. We devise and evaluate a novel back-and-forth model of contagion dynamics where nodes take one of two states (e.g., either 0 or 1) at each time, and transition back-and-forth between the two states based on threshold criteria. We provide complexity results for a number of problems related to this bithreshold system, and again show interesting differences between simple and complex contagions for a range of dynamical properties. We also devise a blocking method, evaluate it, and demonstrate that it performs much better than a well-performing technique for blocking progressive complex contagions. Finally, we explore these dynamics on several real social networks.

Chapter 2

Graph Dynamical Systems

2.1 Introduction

2.1.1 Background and Motivation

Section 1.5 describes a classic or fundamental GDS. Social science, as well as other fields, motivate the need for greater expressiveness in the model to handle some applications. This section provides a formal description of an extended GDS (EGDS).

The main extensions are (*i*) edge labels, (*ii*) local edge functions, (*iii*) non-Markovian local vertex and edge functions, and (*iv*) notions of update schemes with finer granularity than vertex and edge IDs. Other features a classic GDS admits, but that are infrequently used, become more prominant in social science applications. These include vector-valued vertex states that represent collective states over multiple contagions, and vertex and edge multi-component functions that can represent different mechanisms of state transition, including interventions, and individual/unique transition behavior for each of several contagions. Note that interventions may also alter the graph structure, and although classic GDS assume a fixed graph, vertex and edge additions and deletions are readily handled in the EGDS.

We use Figure 2.1 to describe a motivating example, that of joining a mass movement. In countries whose governments limit civil rights, joining such a movement is a nontrivial ordeal, with possible adverse ramifications to oneself and family (consider the U. S. in the Civil Rights era of Martin Luther King). Thus, in addition to the contagion of joining a movement, and possibly disengaging, there may be an emotional contagion of which fear is a component, and today, there might be an information contagion that spreads via Twitter. The contagions interact; e.g., increased fear may increase reluctance to participate; received information may alter fear state and the inclination to participate.

The six members of the system in Figure 2.1 (i.e., the six vertices) are connected in different

ways, depending on the contagion; some connections are reciprocal (i.e., undirected edges) while others are one-way (i.e., directed edges). Vertices 4 and 5, for example, only influence each other in the emotional contagion system. Vertex 2 follows vertex 1 on Twitter, but not vice versa. The total population and set of interactions is represented by the union graph; i.e., the graph that is the union of all three networks. Technically, a union graph is not required in the EGDS, but the information must exist in the EGDS in some form, and a graphical representation is intuitive. Each edge can be assigned different labels indicating the strength of an interaction or other aspects of it. In the co-location graph, for example, the strength of influence from vertex 1 to vertex 2 may be twice as great as that from vertex 2 to 1. Further, the influence of vertex 4 on 1 may be five times that of vertex 2 on 1. Moreover, these strengths may evolve in time and can be captured by edge functions; the strength of influence of vertex 4 on 1 may decrease over time while that of vertex 2 on 1 may increase. An edge function may also describe the evolution of edges; e.g., their existence in time. Edge properties and functions are not part of the classic GDS.

Continuing this example, there is evidence that a person may be more heavily influenced to join a protest if she receives reinforcing signals over short time intervals [127]. In this case, updating state based only on the previous state (i.e., a Markovian approach, used in classic GDS) will not capture this history effect.

Each vertex function may be composed of multiple component functions, one for each contagion. For example, the state of vertex v, x_v, may be a vector of length six, where the first four vector components contain the state of v for contagion c_1 and the last two elements of x_v contain the state of v for contagion c_2. The vertex function for contagion c_1—the component vertex function f_{v,c_1}—will then update the first four components of x_v and f_{v,c_2} will update the last two components of x_v. However, the entire state of v will be an input to both component vertex functions.

Sequencing the execution of vertex and edge component functions for vertices and edges across contagions may affect the system dynamics because, presumably, the local functions for a particular contagion will take as inputs the states describing the other contagions. For example, at each time t, one might execute all vertex component functions for the fear contagion simultaneously, execute the vertex component functions for the Twitter information contagion next, and execute the component vertex functions for the protest-joining contagion last. Alternatively, one might execute component vertex functions for all contagions for a vertex simultaneously, but sequence the vertices themselves as $(6, 4, 2, 1, 3, 5)$. Large differences in system behaviors induced by different sequencings of local component functions have been demonstrated in one of our studies [179]. The classic GDS does not include these more nuanced update disciplines.

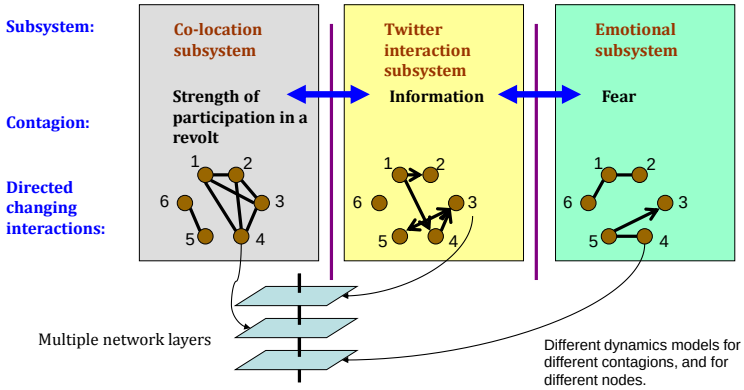


Figure 2.1: Schematic of a multi-contagion dynamics problem on time-varying networks. Here, there are three interacting contagion processes on a 6-vertex graph whose pairwise vertex interactions are predicated on the particular contagion. Multi-networks and multiple contagions have been observed in analyses of social movements and protests (see e.g., [128, 149, 220]).

2.1.2 Summary of Contributions

This section describes formal extensions of a GDS that are needed for social science applications. These extensions, in turn, are used as the theoretical underpinnings of the ENSIM modeling environment of Chapter 3.

2.2 Preliminaries

In a classic GDS, sequencing of vertices for update schemes (see Section 1.5) required a unique integer value to identify each vertex. Here, since we will add edge functions, vertex and edge component functions, and contagions to the formulation, we also need unique integer values to identify edges and component functions for vertices and edges relative to a particular contagion. The purpose of these preliminaries is simply to formalize these notions so that unique integers can be assigned to each of these constructs, so that permutations are well-defined.

We define two bijections that map edges E, into positive integers. Let $m = |E|$. Let $\Gamma\colon V \times V \longrightarrow \mathbb{N}$ be the bijection $j = \Gamma(u, v)$ that maps a directed edge from u to v $(u, v \in V)$ into a natural number $j \in \mathbb{N}$, $1 \leq j \leq m$. To begin our numbering after the number n of vertices in G (i.e., $n = |V|$), we define $\Gamma^*\colon V \times V \longrightarrow \mathbb{N}$ as the bijection $j = \Gamma^*(u, v)$ that maps a directed edge into a natural number $n+1 \leq j \leq n+m$. We also employ the bijection

$\Gamma_s^*\colon V \times V \longrightarrow \{\mathbb{N}\}$ on sets defined by $J = \Gamma_s^*(E')$, where $E' \subset E$ is a subset of graph edges and J represents the set of integer indices corresponding to this set of edges. These functions enable us to refer to a vertex or edge with a single unique index.

We add components to this development. Components are dimensions of a problem that combine with vertices and edges in the specification of sequencing of vertex and edge functions. For example, a set C of components may be a set of contagions, a set of state transition mechanisms, or a set whose elements are unique pairs formed from the Cartesian product of two sets. We define $\hat{\Gamma}\colon V \times (V \cup \{-1\}) \times C \longrightarrow \mathbb{N}$ as the bijection $j = \hat{\Gamma}(u, y, c)$. If $y \in V$, these inputs map a directed edge (u, y) and a component c into a unique integer and if $y = -1$, these inputs map a vertex u and component c into a unique integer.

2.3 Extended Graph Dynamical System

An **extended graph dynamical system** (EGDS) $\mathcal{S}$ is a five-tuple $\mathcal{S}(G, K_v, K_e, \mathbf{F}, W)$ where G is a directed graph, K_v is the set of vertex states, K_e is the set of edge states, $\mathbf{F}$ is the GDS map, and W is the update scheme that specifies the sequence in which vertices and edges, and their components, are evaluated for state transitions. All of these entities are defined below.

Let G denote a directed graph with vertex set $V = \{1, 2, \ldots, n\}$ and edge set E. An edge $(u, v) \in E$ is directed from vertex u to vertex v. An undirected edge $\{u, v\}$ simply means the two edges (u, v) and (v, u).

To each vertex v we assign a state $x_v \in K_v$ and refer to this as the **vertex state**. To each edge (u, v) we assign a state $x_{(u,v)} \in K_e$ which is the **edge state**. We also write $x_{(u,v)}$ as $x_{\langle e \rangle}$ to provide a single integer index e, and to distinguish an edge (within $\langle \cdot \rangle$) from a vertex. We also sometimes write the edge state as x_e if the context is unambiguous.

The **system state** consists of the states of all vertices and edges in G, and is written in either of the following forms

$$\begin{aligned} x &= (x_1, x_2, \ldots, x_n, x_{\langle 1 \rangle}, x_{\langle 2 \rangle}, \ldots, x_{\langle m \rangle}) \\ &= (x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+m}) \end{aligned} \tag{2.1}$$

where the n vertex states precede the m edge states.

Let $m[v]$ denote the sequence of edges (u, v) incident on v sorted in increasing order according to Γ^*. Let the i'th edge in $m[v]$ be denoted $m[v](i)$. Let $n[v]$ denote the sequence of vertices u from all directed edges (u, v) into v, also sorted in increasing order. The i'th vertex in $n[v]$ is denoted $n[v](i)$. We write the **restricted state** for vertex v as

$$x[v] = (x_{n[v](1)}, x_{n[v](2)}, \ldots, x_{n[v](d^i(v)+1)}, x_{m[v](1)}, x_{m[v](2)}, \ldots, x_{m[v](d^i(v))}) \tag{2.2}$$

for the sequence of vertex and edge states. Here $d^i(v)$ denotes the in-degree of v (i.e., the number of directed edges into v). Similarly, we express the restricted state for edge (u, v) as

$$x[(u,v)] = (x_u, x_v, x_{m[u](1)}, x_{m[u](2)}, \ldots, x_{m[u](d^i(u))}, x_{m[v](1)}, x_{m[v](2)}, \ldots, x_{m[v](d^i(v))}) \tag{2.3}$$

for its sequence of vertex and edge states. Here, the edges are those incoming to u and incoming to v. Note that the edge (u, v) itself is contained in the sequence $m[v]$. We may write $x[(u, v)]$ as either $x[\langle e \rangle]$ or $x[e]$.

The dynamics of vertex states are governed by a list of **vertex functions** $(f_v)_{v=1}^n$ (often denoted in shorthand as $(f_v)_v$). We incorporate time history by including a time window of size $T \in \mathbb{N}$, such that states at the current time t and at $T-1$ times into the past can now influence state transitions. We have $f_v \colon \left(K_v^{d(v)+1} \times K_e^{d^i(v)}\right)^T \longrightarrow K_v$ mapping as

$$x_v(t+1) = f_v\big((x(\tau)[v])_\tau\big) \quad t - T + 1 \leq \tau \leq t \; . \tag{2.4}$$

In other words, the state of vertex v at time $t+1$ is given by f_v evaluated at the restricted state $x[v]$ at times $t-T+1 \leq \tau \leq t$. We write $(x(\tau)[v])_\tau$ to denote the *sequence* of restricted states that is the input argument. However, to ease notation, we will drop the subscript τ going forward. We motivated components earlier in this section, where C denotes a set of components. Therefore, we also allow each f_v to be composed of **component vertex functions** $f_{v,c}$, so that $f_v = (f_{v,1}, f_{v,2}, \ldots, f_{v,n_c})$, where $n_c = |C|$. Then, by component, $f_{v,c} \colon \left(K_v^{d(v)+1} \times K_e^{d^i(v)}\right)^T \longrightarrow K_v$ is defined by

$$x_v(t+1) = f_{v,c}\big(x(\tau)[v]\big) \quad t - T + 1 \leq \tau \leq t \; . \tag{2.5}$$

Analogously, edge state dynamics are governed by a list of **edge functions** $(f_e)_{e=1}^m$ (denoted in shorthand as $(f_e)_e$). We may write $f_{(u,v)}$ for the edge function for (u, v). Again we incorporate time history through a time window of size T, such that states at the current time t and at $T-1$ times into the past can now influence edge state transitions. We have $f_{(u,v)} \colon \left(K_v^2 \times K_e^{d^i(u)+d^i(v)}\right)^T \longrightarrow K_e$ maps as

$$x_{(u,v)}(t+1) = f_{(u,v)}\big(x(\tau)[(u,v)]\big) \quad t - T + 1 \leq \tau \leq t \; . \tag{2.6}$$

As for vertices, we may have $f_{(u,v)}$ comprised of **component edge functions**; i.e., $f_{(u,v)} = (f_{(u,v),1}, f_{(u,v),2}, \ldots, f_{(u,v),n_c})$. A component $f_{(u,v),c} \colon \left(K_v^2 \times K_e^{d^i(u)+d^i(v)}\right)^T \longrightarrow K_e$ is defined as

$$x_{(u,v)}(t+1) = f_{(u,v),c}\big(x(\tau)[(u,v)]\big) \quad t - T + 1 \leq \tau \leq t \; . \tag{2.7}$$

An **update mechanism** governs how the list of vertex and edge functions assemble to a **graph dynamical system map** (see e.g. [211, 233])

$$\mathbf{F} \colon (K_v^n \times K_e^m)^T \longrightarrow K_v^n \times K_e^m$$

sending the system state at time t to that at time $t+1$.

For the update mechanism of sequencing vertices and edges for execution of local component functions, we describe here synchronous, sequential, and block sequential schemes. In all cases, $\hat{\Gamma}$ is used to specify a unique integer identifier for each combination of vertex and component and combination of edge and component. For n vertices, m edges, and c components, $\hat{\Gamma}$ will produce a set of integers $\{1, 2, \ldots, c(n+m)\}$.

In the **synchronous** case, all vertex and edge functions execute simultaneously

$$\begin{aligned} &\mathbf{F}(x_1(\tau), \ldots, x_n(\tau), x_{n+1}(\tau), \ldots, x_{n+m}(\tau)) \\ &= (f_1(x(\tau)[1]), \ldots, f_n(x(\tau)[n]), f_{\langle n+1\rangle}(x(\tau)[\langle n+1\rangle]), \ldots, f_{\langle n+m\rangle}(x(\tau)[\langle n+m\rangle])) \\ &= (f_1(x(\tau)[1]), \ldots, f_n(x(\tau)[n]), f_{n+1}(x(\tau)[n+1]), \ldots, f_{n+m}(x(\tau)[n+m])) \\ &\qquad t - T + 1 \leq \tau \leq t \end{aligned} \tag{2.8}$$

where the last equation drops $\langle\cdot\rangle$. We refer to this special GDS class as an **extended generalized cellular automata** (EGCA). In the synchronous case, we assume that component functions update orthogonal elements of state vectors, thus avoiding non-determinism in the update process.

In the **sequential update** case, we consider permutation update sequences (i.e., permutations of the integers produced by $\hat{\Gamma}$). For this we first introduce the notion of **G-local vertex functions**. Here, assuming for the moment only one component so that the component subscript may be omitted, the G-local vertex function $F_v\colon (K_v^n \times K_e^m)^T \longrightarrow K_v^n \times K_e^m$ is given by

$$\begin{aligned} &F_v(x_1(t), \ldots, x_n(t), x_{\langle n+1\rangle}(t), \ldots, x_{\langle n+m\rangle}(t)) = \\ &(x_1(t), x_2(t), \ldots, f_v(x(\tau)[v]), \ldots, x_n(t), x_{\langle n+1\rangle}(t), \ldots, x_{\langle n+m\rangle}(t)) \quad t - T + 1 \leq \tau \leq t\,. \end{aligned} \tag{2.9}$$

Similarly, a **G-local edge function** $F_v\colon (K_v^n \times K_e^m)^T \longrightarrow K_v^n \times K_e^m$ is given by

$$\begin{aligned} &F_e(x_1(t), \ldots, x_n(t), x_{\langle n+1\rangle}(t), \ldots, x_{\langle n+m\rangle}(t)) = \\ &(x_1(t), x_2(t), \ldots, x_n(t), x_{\langle n+1\rangle}(t), f_e(x(\tau)[e]), \ldots, x_{\langle n+m\rangle}(t)) \quad t - T + 1 \leq \tau \leq t\,. \end{aligned} \tag{2.10}$$

As with other edge-related parameters, we may write F_e as $F_{\langle e\rangle}$ or $F_{(u,v)}$.

For the case of components, the **component G-local vertex function** $F_{v,c}\colon (K_v^n \times K_e^m)^T \longrightarrow K_v^n \times K_e^m$ is given by

$$F_{v,c}(x_1(t),\ldots,x_n(t),x_{\langle n+1\rangle}(t),\ldots,x_{\langle n+m\rangle}(t)) = \\ (x_1(t),x_2(t),\ldots,f_{v,c}(x(\tau)[v]),\ldots,x_n(t),x_{\langle n+1\rangle}(t),\ldots,x_{\langle n+m\rangle}(t)) \quad t-T+1\leq\tau\leq t\ , \tag{2.11}$$

with a similar expression for a **component G-local edge function** $F_{e,c}$.

Permutations π must allow interleaving among vertex and edge component functions. For n_c components, we have the permutation update sequence

$$\pi = (\pi_1,\ldots,\pi_n,\pi_{n+1},\ldots,\pi_{n_c(n+m)}) = (\pi(1),\ldots,\pi(n),\pi(n+1),\ldots,\pi(n_c(n+m)))\ , \tag{2.12}$$

where each π_i is produced by $\hat{\Gamma}$. The corresponding sequential (or asynchronous) GDS map $\mathbf{F}_\pi\colon\ (K_v^n\times K_e^m)^T\longrightarrow K_v^n\times K_e^m$ is given by

$$\mathbf{F}_\pi = F_{\pi(n_c(n+m))}\circ F_{\pi(n_c(n+m)-1)}\circ\cdots\circ F_{\pi(n+1)}\circ F_{\pi(n)}\circ\cdots\circ F_{\pi(1)}\ . \tag{2.13}$$

The connection between Equations (2.9) and (2.13), and between Equations (2.10) and (2.13), is that F_v is $F_{\pi(j_1)}$ and F_e is $F_{\pi(j_2)}$ where v is the j_1'th entry of π and e is the j_2'th entry of π. We also refer to this class of sequential system as (permutation) **extended sequential dynamical systems** (ESDSs).

Block sequential update, the third scheme, generalizes the previous two. Vertices, edges, and components are grouped into blocks Bj and these blocks form a sequence, producing a permutation $\pi_B = (\pi_{B1},\ldots,\pi_{Bq})$, possessing q blocks, as an update sequence. Each π_{Bj} contains a unique set of integers from $\{1,2,\ldots,n_c(n+m)\}$ and each element of the set appears in one π_{Bj}. Vertex and edge (component) functions for all vertices and edges in one block are executed in parallel as in the synchronous case, with sequential ordering between consecutive blocks. As in the synchronous case, we assume that multiple component functions for the same vertex or edge update unique elements of state vectors to avoid non-determinism. We adapt Equations (2.9) and (2.10) for the general block $Bj = \{v_{a_1},v_{a_2},\ldots,v_{a_r},e_{b_1},e_{b_2},\ldots,e_{b_s}\}$ (assuming only one component) to produce the G-local function $F_{\pi_{Bj}}\colon\ (K_v^n\times K_e^m)^T\longrightarrow K_v^n\times K_e^m$ given by

$$\begin{aligned} F_{\pi_{Bj}}(x_1(t),\ldots,x_n(t),x_{\langle n+1\rangle}(t),\ldots,x_{\langle n+m\rangle}(t)) = \\ (x_1(t),x_2(t),\ldots,f_{v_{a_1}}(x(\tau)[v_{a_1}]),\ldots,f_{v_{a_r}}(x(\tau)[v_{a_r}]), \\ x_n(t),x_{\langle n+1\rangle}(t),\ldots,f_{e_{b_1}}(x(\tau)[e_{b_1}]),\ldots,f_{e_{b_s}}(x(\tau)[e_{b_s}]),\ldots,x_{\langle n+m\rangle}(t)) \\ t-T+1\leq\tau\leq t\ . \end{aligned} \tag{2.14}$$

Suitable modifications may be made to Equation (2.14) to include component functions.

The block sequential GDS map $\mathbf{F}_{\pi_B}\colon (K_v^n \times K_e^m)^T \longrightarrow K_v^n \times K_e^m$ is defined as

$$\mathbf{F}_{\pi_B} = F_{\pi_{Bq}} \circ F_{\pi_{Bq-1}} \circ \cdots \circ F_{\pi_{B1}} \ . \tag{2.15}$$

Just as for GDS, EGDS may have fair and unfair word orders and fair and unfair block word orders [232].

Example: We provide an example of a block sequential GDS map, which is a concrete representation of Equation (2.15). There are n_c components, each representing a different contagion. For each component, there is one block consisting of all edges and there is another block consisting of all vertices. Thus, there are $2n_c$ blocks. We write the block sequential permutation $\pi = (\pi_{B,1}, \ldots, \pi_{B,n_c}, \pi_{A,1}, \ldots, \pi_{A,n_c})$ where each block B, i represents all edges and contagion i and each block A, j represents all nodes and contagion j. For each component, all edges are updated synchronously. After all edges have been updated, all vertices are updated in block sequential fashion. This description is presented in the algorithm of Figure 2.2 to compute states at each time in. ■

2.4 Phase Space

For the case of deterministic vertex and edge functions, the **phase space** of the GDS map $\mathbf{F}\colon (K_v^n \times K_e^m)^T \longrightarrow K_v^n \times K_e^m$ with prescribed window size T is the directed graph G_{ps} with vertex set $K_v^n \times K_e^m$ and edge set $\{(x, \mathbf{F}(x)) \mid x \in K_v^n \times K_e^m\}$. 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** for $\mathbf{F}$. Hence, G_{ps} may have self-loops. A state that is not periodic is a **transient state**. Classically, the **omega-limit set** of x, denoted by $\omega(x)$, is the accumulation points of the sequence $\{\mathbf{F}^k(x)\}_{k\geq 0}$. In the finite case, the omega-limit set is the unique periodic orbit reached from x under $\mathbf{F}$.

A **forward trajectory** for an initial system state $x(0) = x_0$ of the GDS map $\mathbf{F}\colon (K_v^n \times K_e^m)^T \longrightarrow K_v^n \times K_e^m$ for a fixed T is a directed subgraph of G_{ps} with vertex set $(K_v'^n \times K_e'^m)$, $(K_v')^n \subset K_v^n$, and $(K_e')^m \subset K_e^m$, and edge set $\{(x, \mathbf{F}(x)) \mid x \in (K_v'^n \times K_e'^m)\}$. The first edge computed is $(x_0, \mathbf{F}(x_0))$, and subsequent edges are generated as $((x_0, \mathbf{F}(x_0)), (x(1), \mathbf{F}(x(1))), \ldots)$, where $x(k+1) = \mathbf{F}(x(k))$. The number n_e of edges computed is either specified or computations continue until a limit cycle is produced. In simulation, forward trajectories (i.e., paths through G_{ps}) are typically computed. In a modeling and simulation context, we we refer to one forward trajectory as a **diffusion instance** and a collection of diffusion instances as a **simulation**.

When vertex and edge functions are deterministic, vertices in the graph of the phase space have out-degree of exactly one. That is, given a system state x, the next state $\mathbf{F}(x)$ is uniquely computed. On the other hand, if vertex and edge functions are stochastic, then out-degree is not restricted to one. Also, any state x whose $\mathbf{F}(x)$ is stochastically computed cannot be

---

for $(j = 1$ **to** $n_c)$

for $(i = 1$ **to** $m)$

(1i) Edge i evaluates component $\pi_c(j)$, which is $f_{i,\pi_c(j)}$. Let y_i denote the value computed.

end for

for $(i = 1$ **to** $m)$

(1ii) Edge i sets its $\pi_c(j)$ component state $x_{i,\pi_c(j)}$ to y_i.

end for

end for

for $(j = 1$ **to** $n_c)$

for $(i = 1$ **to** $n)$

(2i) Vertex i evaluates component $\pi_c(j)$, which is $f_{i,\pi_c(j)}$. Let y_i denote the value computed.

end for

for $(i = 1$ **to** $n)$

(2ii) Vertex i sets its $\pi_c(j)$ component state $x_{i,\pi_c(j)}$ to y_i.

end for

end for

Figure 2.2: Consider a graph G with n vertices and m edges. This pseudo-code computes and updates edge states across all n_c contagions, and then computes and updates vertex states across all contagions. Contagions are ordered by permutation π_c. For each entry in the permutation, all edges are updated synchronously (inner **for** loops $(1i)$ and $(1ii)$); i.e., each block contains all edges. Thereafter, the contagions are iterated over again, and all vertices are updated synchronously (inner **for** loops $(2i)$ and $(2ii)$) for each permutation entry. This code provides a detailed example for a block sequential map given by Equation (2.15).

---

a fixed point because multiple transitions are possible. See [29] for further descriptions of stochastic dynamical systems. To assess stochasticity-induced variability in state transitions, multiple forward trajectories are often computed from the same initial system state x_0 and update scheme.

2.5 Summary

Extended GDS was motivated and defined. In Chapter 3, we present implementation details of our modeling environment in terms of this description.

Chapter 3

Modeling Environment

3.1 Introduction

3.1.1 Background and Motivation

In this chapter, we describe our modeling environment called ENSIM. Motivations that guide our approach and design are given, along with technical challenges in constructing such a system. We provide serial and two parallel algorithms and present time and space complexities for each. We also prove that ENSIM will simulate any GDS. We provide performance data and a case study, along with useability data. Background discussion of modeling environments, agent-based simulators, and how ENSIM compares to some of these software tools is found in Section 1.6.

All of the categories of dynamics models in Table 1.4 have embedded in them the idea of **interaction**: one node provides some stimulus to another, and this may affect the receiving node's behavior (there can also be mutual, asymmetric stimuli). However, interactions differ in different contexts. A teenager, for example, may interact differently with her parents than her peers. There are several different classifications of these interactions [99]. For example, [249] identifies communication, influence, and joint action as different kinds of interactions. We note two implications. First, different models may operate simultaneously, each describing a different contagion process where the contagions also affect each other. Even a single contagion may be described through multiple processes [179]. Second, edges in a graph representation of a population, which signify interactions, can have different meanings and hence different properties associated with them, and these differences are important to model [129]. As a result, networks may no longer be simple but rather multigraphs [331] (i.e., graphs may have multiple edges between two nodes that describe different types of interactions). A detailed example is provided in Section 2.1.1. These issues introduce complicating factors for high performance computing (HPC) different from those in many

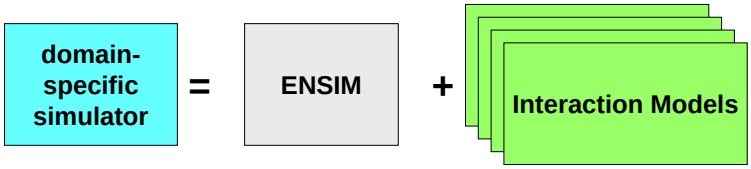


Figure 3.1: Conceptual view of combining ENSIM with dynamics models to produce a simulator. The interaction models are pluggable and ENSIM is configurable through input file parameters. Software tools such as ENSIM that enable customization of feature sets for particular user needs are called for in [49].

application domains. For example, there are many HPC simulators used in epidemiology [36], but these overwhelming use one model of behavior and simple graphs where edges have one meaning. Some simulators [258] do not even represent specific interactions, per se, and instead use compartmentalized uniform mixing models. These approaches consequently significantly reduce data storage requirements, which can be limiting for general, large-scale social simulations; but they cannot model fine-grained interactions.

The above discussion and considerations of trends in social modeling provide three critical observations that guide our work. First, many types and models of social behavior are of interest to researchers. Second, it is important to model particular, possibly multifaceted interactions between specific population entities. Third, owing to social media and online interactions, there is an ever-growing number of at-scale experiments with millions of users. To study social dynamics and extend these experimental insights, we therefore need a modeling framework that (*i*) supports a wide range of models for social dynamics and rapid integration of new models, including models that incorporate interventions; (*ii*) enables descriptions of specific interactions of different types, including time-varying interactions and those involving different *types* of nodes (e.g., humans, social media sites, airplanes); (*iii*) scales to large populations; and (*iv*) owing to the possibly large number of model inputs and permissible values for each, easily supports parametric studies involving millions of simulations.

We describe here such an environment for the simulation of contagion dynamics on populations represented as graphs, where nodes represent agents and edges represent interactions. We call this system ENSIM (Environment for Network-based Social Interaction Modeling, pronounced "enzyme"). ENSIM is a distributed, configurable **software framework**, that when composed with **interaction models** (IMs) that implement particular behaviors, becomes a tailored agent-based simulation environment (Figure 3.1). For example, all the entries in Table 1.4 can be implemented as IMs, as well as many others (see Appendix A). These IMs describe *individual* behavior, and the goal is to understand the resulting population behavior arising from local interactions. Throughout this chapter, we use *dynamics model* and *IM* somewhat interchangeably: both describe individual behavior.

As an added note, with data mining and algorithmic advances [95, 221, 246, 315], increasing numbers of user attributes are being inferred from data, yielding the possibility of models that incorporate increasing levels of individualism. Our agent-based modeling framework can handle any number of such attributes that are assigned to particular nodes and edges (computer system memory permitting).

The versatility of our modeling environment comes from its grounding in **graph dynamical systems** (GDS). A GDS is a universal model of computation; it can simulate a Turing Machine [28]. A formal description of a GDS is provided in Section 1.5 and an EGDS in Chapter 2. Here, we informally tie a GDS to the modeling environment and focus only on local vertex functions. Each node of a GDS has a state that may change with time. Further, a GDS contains a function f_i for each node v_i, whose inputs are the states (and possibly other attributes) of v_i and v_i's neighbors, and (possibly) the attributes of edges incident on v_i. At time t, all inputs are known and f_i computes the next state of v_i corresponding to time $t + 1$. The state of an entire population is updated at time $t + 1$ when all f_i have executed. There are numerous options for the sequencing of execution of the f_i. A change in the behavior of a node (e.g., a node acquires a contagion) corresponds to a change in its state. These state changes, therefore, quantify contagion dynamics. An IM implements the function f_i in software and handles the sequencing of f_i relative to other functions. These functions can also embed the meaning of edges, which is particularly relevant for social modeling as there are several taxonomies for the meanings of edges [99]. For example, in the social movement literature, [171] identifies information, identity, and exchange types of interaction mechanisms.

In general, determining dynamical properties of the kind that can be studied with ENSIM, such as whether a population will reach a state where many nodes possess a contagion, are PSPACE-complete [28]. Other dynamics-based problems, such as determining the k nodes to seed that will maximize the number of nodes that receive a contagion, is NP-hard even for relatively simple models such as independent cascade and linear threshold [166]. Furthermore, computing the spread size given a seed set in the stochastic independent cascade model is #P-hard [78]. Determining whether a node will become infected given one seed node, in a probabilistic SIR (susceptible-infectious-recovered) model is NP-hard [283]. Clearly, these types of problems are different from graph computations that determine (static) structural properties of graphs, such as clustering coefficients (e.g., [23]) or numbers of triangles [299].

3.1.2 Technical Challenges

Computing dynamics involves several factors that make these computations a challenge. Networks are typically large and irregular, thus eliminating model reduction techniques from consideration. This also makes the exploitation of data locality impossible in the general case. Many cellular automata models, for example [1], exploit this locality.

With large networks, scalability may be limited by the amount of memory needed for node- or edge-specific properties. (This is indeed the case for ENSIM.) One can reduce memory requirements by assigning the same properties to each node and edge, so that agent interactions are more homogeneous. However, such an approach precludes the study of dynamics on systems where unique interactions play an important role (e.g., [117, 129, 132, 205, 217, 220, 249, 281]), and thus is not a robust solution for social modeling. As but one example, consider a classroom of junior high school students. While they are all in the same classroom, it is obvious to anyone who attended school that not all students interact. Rather, there are groupings of students that socialize, and hence, modeling all pairwise node interactions as (*i*) existing, and (*ii*) being homogeneous (i.e., all interactions are the same) clearly does not reflect reality. Note that this approach of "homogeneity of interactions" is precisely what is done in many discrete time and discrete event simulators; e.g., [258, 261, 326] to streamline computations and reduce runtime.

This issue of generality applies more broadly: since our goal is to provide a general framework to model any GDS, optimizations for particular problems are not viable, since they would limit its use for more general problems (this is discussed further below). However, optimizations for particular domains can be implemented in IMs.

Also, large numbers of independent variables most often require ensembles consisting of many large-scale simulations so the ability to perform parametric studies with a single code execution is important. Other features that are desirable in any modeling framework for social dynamics include support for time-evolving networks, different types of initial conditions, stochasticity in behavior models, and a variety of node and edge attributes. All of these features are supported in ENSIM.

A primary goal of our work is to enable a wide range of contagion dynamics (IMs) that can be implemented and added by users. A second goal is enabling users to implement their own dynamics models and integrate them with ENSIM in a simple manner. To realize this flexibility, we require a design that brings all relevant "neighboring" data to the compute node where the dynamics for a particular graph node v are calculated. Since ENSIM is a distributed framework, each v is assigned to a process $\mathcal{E}$ that updates v's state; we say that $\mathcal{E}$ **owns** v. This makes it easier for the user to think about, design, and implement IMs using both serial and single-thread-of-execution perspectives (thereby enhancing user productivity). It is also a general approach. We make these ideas concrete with the following example.

Consider Figure 3.2, where a graph (left) has its nodes assigned to two processes $\mathcal{E}_1$ and $\mathcal{E}_2$. Hence, each process stores a subgraph of the original graph. Nodes in blue have their states updated by the respective process, so that, for example, $\mathcal{E}_1$ updates the states s_i and s_k for v_i and v_k, respectively. However, to compute s_i, the state for v_j is required (because v_j is a neighbor of v_i), so $\mathcal{E}_1$ must receive s_j from $\mathcal{E}_2$. The IM can then manipulate these data values appropriately. In this situation, v_i is the **influencee** and is neighbors are **influencers**. We thus call this approach *compute dynamics on the influencee process.*

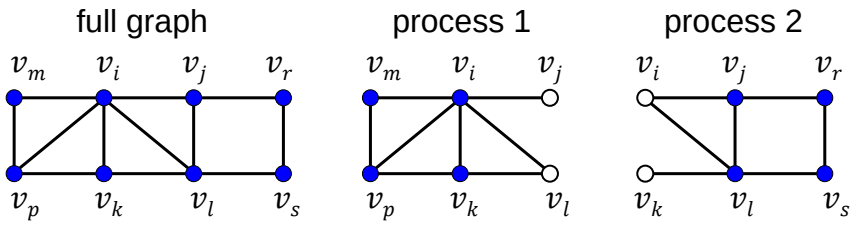


Figure 3.2: Schematic of how an entire graph (on left) may be partitioned between two processes (center and right). Processes update the states of filled nodes but require as inputs to state update computations the states of all neighbors, including those updated by other processes, shown as open symbols. The open symbols are pure influencers on the respective processes.

Later, we will need the notion of a pure influencer. A **pure influencer**, with respect to a given process, is a node that is not owned, but influences owned nodes. For example, in Figure 3.2, on $\mathcal{E}_1$, v_j and v_l are pure influencers. However, on $\mathcal{E}_2$, both of these nodes are owned. The nodes denoted by open symbols are pure influencers on the particular processes.

Other choices, such as *compute dynamics on the influencer process* and then propagate the results to the processes that own the influencee nodes, are viable for particular classes of problems (e.g., *symmetric* functions [232], as done in [36]), but will not work in general without the execution of additional dynamics rules at the influencee. This choice destroys the objectives of enabling an IM builder to use a single thread of control approach and implement all IM code in one place. For example, consider the state update for v_i in Figure 3.2. States for influencers v_m, v_p, v_k, v_j, and v_l are required as inputs. Consider a straight-forward function where each influencer v_q contributes an amount a_q to v_i's state update, where both s_q and s_i are inputs. These can be computed at the influencers (assume that q is j, so that v_j's influence on v_i is computed on $\mathcal{E}_2$), but then v_i's state must be sent to each process beforehand on which an influencer resides in order to compute a_q. Then the a_q's must be combined in some fashion, and it is reasonable to perform this computation on $\mathcal{E}_1$. In summary, to compute the new state s_i', s_i must be sent to all processes with influencer nodes, all contributions a_q are computed and sent to $\mathcal{E}_1$, and then $\mathcal{E}_1$ must perform computations.

This approach also does not handle the more general case (e.g., update functions more intricate than symmetric functions) where the a_q must be computed and combined in particular ways depending on the current states of all influencers and the influencee. Just one consequence of this latter situation is that both a_q and s_q need to be sent to $\mathcal{E}_1$. A simpler and more general approach is compute-at-the-influencee process.

We can also *compute at an arbitrary process*; i.e., compute s_i' on some process that owns neither influencers nor influencee (e.g., as done in [261], granted, for different purposes), but

this would require more communications and larger data payloads in those messages so that general state updates—of the kind we require here—can be computed.

These versatility and usability goals make our third objective of high performance computing difficult. As an example, we have the following result, based on the compute-at-the-influencee philosophy.

Proposition 1. *Consider an Erdős Renyi random graph with n nodes and probability p_e of an edge existing between each pair of nodes. Let the n nodes be distributed among the N processes in an ENSIM process group such that each process owns n/N nodes. Consider a process $\mathcal{E}_i$ that owns a node v and any other process $\mathcal{E}_j$ $(i \neq j)$. The probability that the state of v must be communicated to $\mathcal{E}_j$ is $1 - (1 - p_e)^{n/N}$.*

Proof. Consider a node v_1 on $\mathcal{E}_1$ and a node v_2 on $\mathcal{E}_2$. The probability of there not being an edge between them is $(1-p_e)$. If the nodes are distributed evenly across processing elements, then the probability that v_1 is not connected to any node on $\mathcal{E}_2$ is $(1 - p_e)^{n/N}$. Thus, the probability that v_1 is connected to at least one node on $\mathcal{E}_2$ is $1 - (1 - p_e)^{n/N}$. The result follows. □

Many social networks have average degree of roughly 30, and we are interested in larger graphs where $n \geq 1000000$. Since $p_e = d_{ave}/(n-1)$, p_e is often on the order of 3×10^{-5}. (The average degree of 30 is used as a working number; there are social networks with average degrees lesser and greater than 30. These arguments hold for other values of d_{ave}.) Given our available cluster sizes and their shared use, under normal circumstances, $N = 30$ is the maximum number of processors (and processes) that can be used on a job. The above result yields a 63% probability that the state update of every node will need to be sent to each of the other processes. Thus, with high probability, not only do we have all-to-all communications, but also we have maximum message payloads. These are precisely the conditions we investigate for weak scaling performance in Section 3.5 so that our results there are for a worst-case scenario. Consequently, it is imperative to drive down N to reduce communication costs.

3.1.3 Summary of Contributions

We first describe contributions in four areas, most of which are focused on performance, human productivity, and usability. Thereafter, we describe some technical contributions and optimizations of our framework. Thereafter, we discuss limitations of our framework. Note that related work, including many modeling frameworks and simulation issues such as scheduling, are covered in Section 1.6.

1. High performance computing. Our hybrid implementation scales to at least 100 million nodes and 2.8 billion edges, which compares very favorably with other published works.

Many of these studies use cellular automata or other problem structures much more conducive to producing high performance results than irregular graphs. For example, some use cellular automata where each compute node only communicates with nearest neighbors, as defined by cellular automata layouts, thereby significantly reducing communications. Moreover, our speedup results compare well with those of a finely tuned domain-specific epidemiological code [36]. We improve performance by using a hybrid MPI+OpenMP approach that drives down N and recoups parallelism through multithreading, and by overlapping communications and computations. This hybrid approach is well-suited for execution on supercomputers such as NCSA's Blue Waters. We achieve a 2-order-of-magnitude increase in scalable problem size for the hybrid code, compared to the MPI implementation. For most cases, the hybrid code reduces execution duration by 200% to 400% or more compared to the MPI code. We provide strong and weak scaling data, speedup results, and data on overhead (e.g., produced by communications) for six real social networks that range in size from 2 million and 100 million nodes, and synthetic networks from 1.6 million to 32 million nodes, that span Erdős Renyi (ER) random graphs, exponential-decay (ED) graphs, and scale-free (SF) graphs. We also demonstrate that by altering the IM in a simulation, performance data can change significantly.

2. Human productivity. ENSIM reduces both **total turn around time** (TTAT) and **end-to-end solution time** (ETEST). TTAT is the sum of the times to design, construct, integrate with ENSIM, and verify a new IM. ETEST is the time from problem specification to a finished simulation study; ETEST includes TTAT. We have demonstrated that we can implement a new IM for ENSIM and completely finish a simulation study well before we can modify another simulator so that it is ready to begin a study. In Section 3.7, we present TTAT times for a range of IMs. We show that many IMs that are used to study real-world behavior have TTATs on the order of one day or less. We provide a detailed and illustrative case study in Section 3.6.1 with an ETEST of about 30 hours over five calendar days. As a concrete productivity example, a new post-doctoral student, who had no knowledge of ENSIM, implemented his own IMs and generated simulation results in two days, which includes all startup costs.

3. Dynamics models. Our framework can execute any dynamics model (IM) within the class of **graph dynamical systems** (GDSs), defined formally in Section 1.5 and extended in Chapter 2. A single IM can implement any *set* of finite state machines, for multiple contagions, conforming to GDSs, and multiple IMs may be used across nodes. Throughout this chapter, we confine formal descriptions to node state updates. Our framework can handle more general classes of problems; e.g., joint action models [83], but currently not in a way that is scalable to very large networks. All classes of dynamics models in Table 1.4 can be implemented, as well as many others. IMs can also implement interventions, useful in policy decision-making. We have over 15 *families* of IMs in our **IM library** from which a user can choose. Perhaps more importantly, a user can implement her own IMs and integrate them into the modeling environment; in this sense, the framework is extensible. Existing models can also be used as starting points in the construction of new models.

4. Usability. Since the intent of ENSIM is to enable users to model a wide range of phenomena, system usability is paramount. A user-built IM must only implement a single API, no matter the application domain, and uses a well-defined set of methods to exchange data with the framework. All IM code is constructed with serial-coding and single-thread-of-execution perspectives, as distributed and concurrent programming issues are handled automatically and transparently. Multiple behaviors, through IMs, can be supplied for each *type* of node (e.g., for multicontagion dynamics), including specific dependencies among nodes through edge-specific interactions. In the extreme case, each node of a graph can have assigned to it a unique IM. The framework itself is configurable through input values, including features to assist in verification of new models. Other configurations are available. For example, node assignments to processes can be specified for load balancing (e.g., if some IMs are more compute-intensive than others) or to reduce the communications among distributed processes (by using results from graph partitioning software such as Metis [161]).

Since IMs often have several parameters, and each parameter may have multiple values, it is not uncommon to run parametric studies with a million or more diffusion instances. A **diffusion instance** is one computation of a population's dynamics from time $t = 0$ to a maximum specified time. ENSIM can run any number of diffusion instances with one execution and the user has complete control over which combinations of parameters vary, and how, across diffusion instances.

Another demonstration of usability is that ENSIM as been used to perform computations for the following works: [6,180–183,186]. Users of ENSIM include undergraduate students in electrical engineering and international studies; graduate students in computer science and public health; and physicists and computer scientists at the professional level. The framework has been used in studies of epidemiology, data flow in computer networks, adoption of farming technologies, social sciences, algorithm performance, and mathematics.

ENSIM has several limitations, as described in Section 3.1.5 below.

3.1.4 Aspects of Design and Implementation

We now turn to a few aspects of our modeling environment design and implementation that build on the previous discussion.

Our hybrid implementation is novel for agent-based simulation environments (an exception being [333]). Most rely on MPI for parallelism. We also use an adaptive threading approach wherein threads of the OpenMP group are used for both state updates and message sending operations. The asynchronous messaging system more readily exploits this dual use of threads. Each thread may spend different amounts of time in computing state updates and in messaging, depending on the non-deterministic nature of how message queues fill. Multithreading is used in five places within the code for various purposes.

Many simulation systems use finite state machines, or variants thereof, to define allowable states and transitions. ENSIM can similarly utilize these state devices. However, states and transitions can be more generally specified through rules. An example is a model of happiness of individuals [114], where different states represent different degrees of happiness. One is confronted with how many states should be modeled, and a finite state machine can be specified. An alternative is to specify rules for increasing and decreasing node happiness, without specifying a maximum happiness state. In this fashion, the maximum level of happiness of nodes in a population can be a dependent variable computed from simulation.

A **distance-r** graph, relative to a focal node v, is the subgraph induced by all nodes within distance r of v. We use distance-r graphs to define the extent of influence on each node v. Most simulators that represent populations as networks store distance-1 graphs; i.e., a node is influenced by its immediate neighbors. Distance-2 graphs are useful in social sciences to capture the effect of "friend of a friend." That is, a node that is a neighbor of v's neighbor can influence v. ENSIM can store any subgraph of distance r for the nodes of a graph. Instead of adding new edges to the network to capture the effects at distance $r > 1$, which in the worst case results in $O(m^2)$ edges, where m is the number of edges in the original graph, we preserve the original graph structure, thereby reducing graph storage requirements. This is important because graph edges generally produce the largest storage requirements in the system.

Various approaches are used to make computations more efficient in time and space. To save computational time in deterministic models, fixed points are checked for at each time step, and if one is detected, then computations stop since there will be no more state changes. In addition, for probabilistic models, if all nodes reach a final (terminal) state, then simulations are similarly stopped. To reduce space requirements, multiple graphs over which different contagions spread are stored as a single union graph, where edge labels specify which contagions can propagate over them. Similarly, edge properties use customized storage structures so that any number of properties and kinds of properties (integer, short, long, double, char) are stored in the same data structure. Input data can also be reduced if all agents are of the same type, or there are only a few different groups of properties. In these cases, global properties can be utilized by nodes and edges, rather than having the same properties redundantly stored for every node and/or edge.

3.1.5 Limitations of ENSIM

Our framework has several limitations. First, the capabilities for node mobility are very limited. Conceptually, each node can move in space; there is nothing in the framework to prevent a local vertex function from describing kinetics and kinematics (e.g., position, velocity, and the forces that act on nodes). However, analyzing such a system requires a fully connected graph as input to ENSIM, so that each node's affect on every other node can be computed. Hence, there are $O(n^2)$ edges in the graph, which makes data storage

a problem for large networks (social networks are typically sparse graphs with the average degree of a node far less than $n-1$). Hence, this approach is not scalable to large graphs.

The GDS paradigm is such that each node updates its state independently of the state updates of other nodes. For example, there may be directed edges (u, v) and (v, u) in a graph. Hence, u influences v and vice versa. Therefore, s_u is an input to f_v, and s_v is an input to f_u. However, f_v is not dependent on f_u, and vice versa. We refer to models where f_v and f_u depend on each other as **joint action models** [83]. Our framework has the capability to model joint action, but in a limited sense because the number of graph nodes must be small enough to fit in memory of one compute node. Having all joint action computations on one compute node also reduces scalability.

Currently, edge function capabilities are restricted. An edge function describes the states and transitions of a directed edge $e = (u, v)$. A function can depend on the states of e, u, v, and the states of all edges incident on u and v. However, edge state updates are not propagated through the system as are node states, so edge state updates remain local. A messaging design for edge state updates is required, akin to the existing system for nodes, and this work is planned.

So, too, implementation of time-varying existence of nodes and edges is limited. Currently, any node or edge that is introduced during a simulation must be instantiated at the outset. State variables then describe whether the node/edge is active or inactive, and in this way, nodes and edges can be "added" or "removed" from the network as a function of time. The addition and removal processes would also be controlled by vertex and edge functions.

There is no dynamic load balancing in ENSIM. Graph nodes can be assigned arbitrarily to compute nodes based on results from a user-provided offline pre-processing step. However, on-the-fly load balancing, to take into account the dynamics of contagion evolution and corresponding computational loads, does not exist.

Unlike simulators such as NetLogo, there is no user interface for ENSIM. All interactions with ENSIM are through the command line. A user typically produces scripts to generate input data files and several scripts are available to post-process results. However, new applications and new IMs often demand that new post-processing scripts be generated. At this time, consideration is being given to integrate ENSIM into CINET [2], which would expose ENSIM through a user interface for free public use.

Because ENSIM is designed to be a general-purpose modeling environment, it is not optimized or tuned for a particular problem. Simulators that are tuned may be more attractive for a particular application for several reasons. Among these are: data formats may be less complex or fewer inputs may be required; tuned simulators may run much faster because they can exploit problem sematics to utilize approximations that speed computations but yet give sufficiently accurate results.

As will be shown with forthcoming performance data, ENSIM is designed to execute on multicore compute nodes or symmetric multiprocessor nodes of a cluster. While these architec-

tures are consistent with current trends in hardware, performance will degrade significantly on single-processor compute nodes.

3.2 Graph Dynamical Systems

The basic GDS model and all the extensions discussed below are currently supported in ENSIM. (See Section 1.5 and Chapter 2.) As an example, here we focus on node state updates consistent with a basic GDS; however, we have extended this formalism so that there are analogous capabilities for edge state updates in ENSIM.

The connection between GDSs and our modeling platform is that ENSIM handles control flow and distribution of node and edge state updates (among other features; see Section 3.3) and the IMs that combine with ENSIM implement the local vertex functions, including interventions. Moreover, local functions may also represent more complicated behaviors; e.g., those in statechart-like formalisms [138].

3.2.1 Extended GDS

We now briefly discuss a number of generalizations of the GDS model that are supported in ENSIM. We consider extensions to the structure of a system as well as the types of interactions allowed.

I. Structural Extensions: ENSIM supports many extensions to the topology of the underlying graph as well as the types of state information associated with nodes and edges. These extensions are listed below.

1. The underlying graph may be undirected or directed. This allows ENSIM to support both *symmetric* and *asymmetric* interactions among entities in a system. The set of nodes and edges of the underlying graph may vary over time. Two or more contagions may diffuse through a system simultaneously. Also, the system may actually consist of multiple graphs, one for each contagion. For more efficient storage, each edge has labels denoting which contagions it can transmit.

2. Edges may have weights to model the extent to which nodes may influence their neighbors. Like nodes, edges may also have states. In general, both nodes and edges may have multi-dimensional states; e.g., the state information for a node (agent) or edge (interaction) may be specified by a vector consisting of a user-specified number of values of type `int`, type `double`, and type `char`; nodes also include `short` and `long` types. These properties can also vary in time, or be based on any other system parameter, such as time of last state change or current state.

3. Each node or edge may be associated with a separate finite state machine (FSM) to control the set of state transitions for the node/edge. Each node may have multiple FSMs; e.g., one for each contagion.

4. Nodes can be influenced by neighboring nodes at any distance r. Classically, influencing neighbors are at distance $r = 1$. However, there are models (e.g., [51, 316]) where a distance $r > 1$ formulation is required. We also use an efficient mechanism to store such graphs: we store and manage the edges on a path from v to its influencing neighbor u at distance r, rather than introduce a new edge, (v, u), which in the worst case can result in $O(m^2)$ edges.

II. Extensions to the Nature of Interactions: A number of extensions to the local transition functions are supported by ENSIM. In addition, several additional state update schemes are also supported. These extensions are discussed below.

1. Nodes need not have the same state spaces; the same holds for edges. For example, one set of nodes may represent an aircraft and another may represent humans. Assuming that the locations of humans do not change much but that those of aircraft change appreciably, the vertex state space for aircraft may include position and velocity vectors, while those for humans only contain position vectors. There is no limit on the permissible numbers of node state spaces.

2. Nodes may also have different local vertex functions and edges may have different local edge functions. Continuing with the example immediately above, the node function for an aircraft will update its position and velocity, whereas the function for human nodes will not.

3. In the GDS example considered in Section 1.5, the local transition functions are **Markovian** (or memoryless); thus, the value of a local transition function depends on just the current states of a node and its neighbors. When edges also have state values (as mentioned above), a general equation for the state of a node at time $t + 1$ under the Markovian restriction has the form

$$s_i' = s_i(t + 1) = f_i(x(t)[v_i]) \tag{3.1}$$

where $x(t)[v_i]$ represents the vector of states of the neighboring nodes of v_i at time t, and the vector of states of the edges incident on v_i.

4. When the local transition function is not Markovian but depends on the history of state values over the last T time units, the above equation takes the following more general form:

$$s_i' = s_i(t + 1) = f_i(x(\tau)[v_i]), \tag{3.2}$$

where $t - T + 1 \leq \tau \leq t$ and appropriate vector forms of the quantities $x(\cdot)[v_i]$ are used.

5. Other supported extensions include local transition functions that are *multi-valued* (i.e., *vector-valued*). There are at least three types. First, the function may describe mulitiple mechanisms of state transition, and each mechanism can be described by one or more component functions. Examples of such mechanisms can be standard behavior and interventions. Second, each component function may represent the state transitions for a separate contagion. Third, component functions may represent different types of transitions for the same contagion, one of which can be invoked based on system state. An example is probabilistic Boolean networks (PBNs) [191, 288]. It is clear that these types of vector-valued functions can be combined in arbitrary ways.

6. Local transition functions may be *stochastic* (to capture probabilistic diffusion models).

7. Three update models were discussed in Section 1.5 and Chapter 2. A number of other update schemes are supported by ENSIM. For example, a variation on sequential updates is to specify an arbitrary sequence of nodes and carry out the updates in that order. In such a sequence, a node may appear multiple times, or may not appear at all. Another possible scheme is to a pick a node randomly at each time step and update its state. The choice of a node may be uniform or it may be based on probability distributions that capture priorities of nodes. Node and edge update decisions can be made on-the-fly based on node/edge states, the last time the node/edge was updated, states of neighboring entities, etc. Another update scheme can be predicated on local transition functions that are vector valued. When a node is to be updated, the particular component function chosen (e.g., for a particular contagion) can be specified based on some ordering of the functions.

8. Contagion amounts (or doses) can be specified as the amount, or degree to which, one node influences another. These ideas are part of the notion of **generalized contagions** [101]. There are many possibilities. In the simplest case, v_i may influence all of its neighbors with contagion amount $a_{c,i} = 2$ while v_j may influence with amount $a_{c,j} = 2.5$. Models without contagion amounts have an implicit contagion amount of 1. Alternatively, a node may influence each of its neighbors differently, depending on time or the states of incident edges or nodes. Doses can also combine in nonlinear ways. For example, suppose unaffected node v_i has three neighbors v_a, v_b, and v_c. Individually, each v_j, $j \in \{a, b, c\}$, if affected, contributes a contagion amount of 1 to v_i. However, if both v_a and v_b are affected, then they combine to provide a dose of 4 to v_i. Similarly, if all three v_j are affected the contagion amount may be 7.

The above extensions point out that our framework supports a rich variety of diffusion models.

3.2.2 Formal Problem Statement

Based on the foregoing, we provide a formal statement of the simulation problem. Note that this problem is specified for *any* multi-contagion system where the inputs to each state transition function come from the neighborhood of a node.

Given: A GDS $\mathcal{S} = (G, \mathbf{F}, W)$, including graph $G(V, E)$ with vertex set V and edge set E; a set C of contagions $\{c_1, \ldots, c_q\}$ that propagate on G; vertex K_{v,c_j} and edge K_{e,c_j} state spaces for each contagion c_j; a set of vertex and edge functions $\mathbf{F}_{c_j} \in \mathbf{F}$ for each contagion c_j; a sequence W of the execution of these functions; initial states for all nodes and edges; and a time value t_{max} (positive integer).

Required: The states of all nodes and edges with respect to all contagions, for time values $t = 1, 2, \ldots, t_{max}$.

3.3 Software Descriptions

3.3.1 Overview

At a high-level, the ENSIM framework can be thought of as providing the following functionality for one diffusion instance: it accepts as input the description of a GDS and outputs the results obtained by simulating the system for a specified (maximum) number of time steps.

Throughout this section, for ease of exposition, we assume that node states evolve in time, consistent with GDS, and that the update scheme is synchronous. We address block sequential update informally. We first describe a serial algorithm and associated data structures. We then extend this to a first parallel implementation and provide time and place complexities. Thereafter, we describe the current parallel implementation.

3.3.2 Serial Implementation

Algorithm

A serial simulation algorithm is provided in Algorithm 1. Input G specifies one or more graphs of the GDS (for multiple graphs, G is the **union** graph; i.e., the union of multiple graphs). The inputs specified as node and edge properties (v_P and e_P) require further explanation. There are two types of node and edge properties that we call states and traits. The notion of state values for nodes and edges was discussed earlier; these values may change with time. Traits, on the other hand, are properties that do not change with time. (As a

simple example, a specific system may need the income of the individual represented by a node. Interactions among the entities in the system may use this value; however, it is assumed that the value itself does not change over time.) Inputs n_i and t_{max} represent respectively the number of simulation instances and the maximum number of time steps for each simulation instance. Input d_{ID} specifies the IMs associated with each node and edge and d_P specifies the relevant parameters for the models. The parameter B_0 specifies the base conditions to be used to initialize the entries of d_P and the trait and state entries of v_P and e_P. The parameter I, specifying the **seed** conditions, gives the initial values of the states of nodes and edges that particularize the initial configuration for each simulation instance. The output of the algorithm are the sequences of state values of nodes over time.

The algorithm loops over diffusion instances (line 2), time (line 5), and nodes (line 6) owned by each process. Line 8 of Algorithm 1 is required for a GDS to simulate arbitrary FSMs, including those containing cycles. This is another feature of all our algorithms.

Algorithm 1: Serial Simulation, Synchronous Update

input : Graph G, node properties v_P, edge properties e_P, number of diffusion instances n_i, maximum time per iteration t_{max}, dynamics model identifier(s) d_{ID}, dynamics model parameters d_P, base conditions B_0, seed conditions I.
output: Changes in states s_v of nodes v as a function of time t.

```
1  Read in inputs: G, v_P, e_P, d_ID, d_P, n_i, t_max and B_0.
   Instantiate an IM for each node.
   // Loop over diffusion instances.
2  for (j = 1; j ≤ n_i; ++j) do
3      Reset all node and edge properties and dynamics model parameters to their
       base conditions.
4      Read (initial) seed conditions I_j.
       // Loop over time.
5      for (t = 1; t ≤ t_max; ++t) do
           // Loop over the nodes in the graph to update state.
6          for (v = 0; v ≤ v_max; ++v) do
7              Invoke the IM (local vertex function) for node v.
               if (state of v changes) then
                   Store state updates in temporary storage s_P
8                  Reset model parameters of new state for node v.
9          Copy state changes s_P into v_P.
10         Send state changes to output.
```

Correctness

Proposition 2. *The serial Algorithm 1 correctly implements a basic synchronous GDS.*

Proof. The graph underlying the GDS is read in, as are all properties and dynamics models. Each diffusion instance is run independently and is order-independent because all properties and local function inputs are reset to their initial values (cf. line 3) at the start of each instance. Thus, we must demonstrate that for each time step within an arbitrary diffusion instance j, inputs to an arbitrary local function are the correct values and node states are updated and processed correctly. We use induction on time t.

Consider the base case $t = 1$. From lines 1 and 4, all nodes and edge properties for time $t = 0$ have been assigned to the graph nodes and edges. All nodes execute their local functions (line 7), which correspond to Equation (1.2). For arbitrary node v_l, the inputs are s_l^0 and ${s_{l_q}}^0$ $(1 \leq q \leq r)$, all of which are in v_P. Note that updated node states stored in s_P, if any, are only loaded into the node properties v_P (line 9)—and hence can only become inputs to f_k for a node v_k—after all local functions have been executed. If v_l has its state updated, then IM parameters for the state transitioned to (s_l^1) are reset, if required (line 8), in preparation for the execution of f_l at the next time step. At the end of the time step, updated states are loaded into v_P (so that all states correspond to $t = 1$) and are output to file. Thus, the simulation for $t = 1$ is correct.

Now, for the induction step, we let t be an arbitrary time t^*, and evaluate the simulation for t^*+1. From the simulation at time t^*, all states have been updated so that v_P and e_P contain properties at time t^*. For $t^* + 1$, consequently, all inputs to Equation (1.2) correspond to those at time t^*. For the same reasons as in the base step, new computed states are updated into v_P only after all local vertex functions have been executed. Thus local functions are executed properly, and all node states in v_P correspond to time $t^* + 1$, as desired. □

Remark 3. *The block sequential update scheme requires only small modifications to Algorithm 1. First, set t_{max} to the value n_{blk} t_{max}, where n_{blk} is the number of blocks in the block sequential update scheme. Assign each node a trait that indicates the time (i.e., block) within each time step at which its local function executes. Finally, let the IM contain an* ***if*** *statement such that the local function only executes when the simulation time equals the appropriate multiple of the time stored as a trait. Thus, there exists an analog to Proposition 2 for the block sequential update scheme which is omitted here.*

We immediately have the following corollary.

Corollary 4. *The analog of Algorithm 1 for block sequential loading implements any GDS.*

Time and Space Complexities

Proposition 5. *The time complexity of the serial Algorithm 1 is $O(n_i t_{max}(n + m))$.*

Proof. The time to read in all edges of the entire graph is $O(m)$. The time to read in the node properties is $O(n)$. These properties include IM IDs and properties. Similarly, the time to read in the edge properties is $O(m)$. (These states and trait values are the values B_0.) Reading in other inputs, such as number n_i of iterations and the number of time steps per simulation instance is constant time. The time to instantiate the IMs is $O(n)$. The total setup time is thus $O(n + m)$.

For each simulation instance, the times to reset properties and read in seeds are respectively $O(n + m)$ and $O(n)$. We thus have over all instances $O(n_i(m + n))$.

We now consider the time for one time step of one diffusion instance. The time to compute next states for nodes and store them in s_P is $O(n+m)$. To see this, each node's state update evaluates all of its neighbors, so over all nodes, all m edges are evaluated, giving $O(m)$. Also, the update for each node v evaluates its own state, so over all nodes, we have $O(n)$. The time to later copy the values from s_P into v_P is $O(n)$. The time to reset IM properties for the next state for nodes with state update is $O(n + m)$ because both node and edge properties may need to be reset. Thus, the total time over all simulation instances and time steps is $O(n_i t_{max}(n + m))$.

The state updates dominate the time complexity, giving the stated result. □

Proposition 6. *The space complexity of the serial Algorithm 1 is* $O(n + m)$.

Proof. Graph storage is $O(m)$. The node states and traits each require $O(n)$, and similarly, the edge states and traits each require $O(m)$. Assuming only nodes have state changes, the temporary storage for state updates requires $O(n)$. The IMs that implement the local functions require $O(n)$. Other required storage is constant space, such as the number of iterations and the maximum duration of an iteration. Thus, the total space requirement is $O(n + m)$. □

We address the data structures used in its implementation, after describing the parallel implementation of Section 3.3.3.

3.3.3 MPI Parallel Implementation

Algorithm

The parallel implementation of ENSIM based solely on MPI is now described. It follows a master-worker paradigm, where the worker is responsible for most tasks. Again, we address the synchronous update scheme; the more general block sequential scheme requires only the same small modifications for each worker process described in Remark 3.

For N processes in an MPI process group, there are $w_n = N - 1$ MPI workers and one master process. The master process coordinates some operations such as detecting fixed points for

deterministic IMs. If a fixed point is detected, for example, then simulation for the current diffusion instance can terminate because no further state changes will occur, thus saving compute and wall clock time. All workers send their state updates to the master; the master outputs data so that all results are in one file. IMs can also produce **ancillary data**, which are any combination and any number of `short`, `int`, `long`, `double`, and `char` values, which can vary arbitrarily in time, and these are also sent to the master for output. Examples of ancillary data are the nodes that contribute to the state change of a given node, or the execution duration of a IM instance for performance evaluation. No more will be said about the master process or ancillary data.

The distributed Algorithm 2 describes the behavior of each worker process. There are two major drivers for changes from the serial implementation. First, as described above, each worker is responsible for only a portion of the graph, partition G' (line 1). Graph partitions, owned nodes, influencee and influencer nodes, and pure influencers were defined in Section 3.1.2. Thus, only G' is stored and the loop over the nodes for state update (line 6) is only over the owned nodes of the partition. Second, after all state changes are are computed, MPI workers send state changes to each other, each worker stores the new states of pure influencers into v_P, and each worker sends its state updates for owned nodes to the master for output (lines 12, 14, and 15). Within the loop over time for a diffusion instance (line 4), we see that state updates are computed (**for** loop starting on line 6) and then messages with state updates are shared among workers (**for** loop starting on line 12). Hence, computation and communication are not overlapped; rather, they are executed serially at each time step. We note that the interface methods of

Correctness

We address correctness first. In light of Proposition 2, we need only show that Algorithm 2 produces the same output as Algorithm 1. The main difference between Algorithms 1 and 2 is that the latter contains code, beginning with the comment on line 11, to exchange state update messages with other workers.

Proposition 7. *The parallel Algorithm 2 correctly implements a synchronous GDS.*

Proof. We show three things for an arbitrary time step: that all required inputs for the state update for arbitrary node v on processor $\mathcal{E}$ reside on $\mathcal{E}$, that all nodes have their vertex function executed exactly once at each time, and that the node states are processed correctly.

Consider an arbitrary node v owned by an MPI process $\mathcal{E}$. The partition process ensures that all nodes that influence v remain neighbors of v in the partition. Consequently, all of the properties for v and all of its neighbors reside on $\mathcal{E}$. Hence all inputs for the local function f_i of Equation 1.2 are known locally. Thus, the local function for v can be evaluated properly on $\mathcal{E}$ at an arbitrary time $t^* + 1$.

Algorithm 2: MPI Parallel Simulation, Synchronous Update—One MPI Worker Process

input : Graph partition G', node properties v_P, edge properties e_P, number of diffusion instances n_i, maximum time per iteration t_{max}, dynamics model identifier(s) d_{ID}, dynamics model parameters d_P, base conditions B_0, seed conditions I, number of processes in MPI group n_p.

output: Changes in states s_v of nodes v as a function of time t.

```
1  Read in the partition G' of the graph G that is simulated by this MPI process.
   Read in all properties (node v_P and edge e_P) for this partition G'.
   Read in inputs: d_ID, d_P, n_i, t_max and B_0.
   Instantiate an IM for each owned node of G'.
   // Loop over diffusion instances.
2  for (j = 1; j ≤ n_i; ++j) do
3      Reset all node and edge properties and dynamics model parameters to their
       base conditions.
       Read (initial) seed conditions I_j.
       // Loop over time.
4      for (t = 1; t ≤ t_max; ++t) do
5          // Loop over the owned nodes in the graph partition G' to update state.
6          for (v = 0; v ≤ v_max; ++v) do
7              Invoke the IM (local vertex function) for node v.
               if (state of v changes) then
8                  Store state updates in temporary storage s_P
9                  Reset model parameters of new state for node v.
10         Copy state changes for owned nodes from s_P into v_P.
11         // Send state changes to all other MPI worker processes and receive remote state
           changes from all other MPI worker processes.
           Pack all state updates of owned nodes into a message.
12         for (w = 0; w < w_n; ++w) do
               if (w is the MPI rank of this worker) then
                   Send message to every other worker.
               else
13                 Receive state updates message from worker w.
14                 Update v_P with states received from worker w (these are states for the pure
                   influencers).
15         Send state changes for owned nodes to master process for output.
```

Furthmore, the partition of the graph results in the state of each node being updated by a local function on precisely one $\mathcal{E}$ because each node is owned by one worker.

Nodes are updated in the same order as in Algorithm 1. To see this, note that all state updates are first computed by each worker for their owned nodes and stored in s_P (lines 7 and 8, respectively). Only after all vertex functions have been executed in a process are state updates for owned nodes loaded into v_P (line 10). This completes all operations on all processors to update the states of owned nodes, and since each node is owned by precisely processor, state update computations for all nodes are completed.

Thereafter, starting at line 11, state updates are passed among worker processors to update the states of their pure influencers. State updates for pure influencers are received from other workers and loaded into v_P (lines 13 and 14, respectively).

Thus, for a given simulation step at time $t^* + 1$, all inputs to Equation (1.2) are those at time t^*, all states in v_P at the end of the time step correspond to $t^* + 1$, and each worker sends to the master for output updated states for its owned nodes. These actions produce the same results as those from Algorithm 1. □

We can easily transform Algorithm 2 for the block sequential update scheme, as described in Remark 3 for the serial algorithm, and therefore we also have the analog of Corollary 4 for the parallel algorithm:

Corollary 8. *The analog of Algorithm 2 for block sequential loading implements any GDS.*

Data Structures

Data structures for Algorithm 2 are summarized in Table 3.1. We make use of a variety of maps, for fast data access and for time-evolving networks (node and edge additions and removals are more efficient), even though they require more storage than some other data structures, such as arrays. The structure of a map is `map<k,v>`, where `k` is the key and `v` is the value associated with `k`. The structures of Table 3.1 are used on every MPI worker process. The graph partition is stored as an adjacency list, where `v` is the set of nodes that influence node `k` (nodes `k` are owned nodes). Node and edge traits and states are implemented with maps, but the structures and types for nodes and edges differ. For nodes, the types of states and traits are `short`, `int`, `long`, `double`, and `char`. For edges, the types are `int`, `double`, and `char`. Additional types can be added. For nodes, data for each type is stored in a separate map: `k` is a node ID and `v` is a primitive array of the specified type and user-defined size. Hence, if a simulation requires `int` node states and `double` and `char` node traits, then there are three maps. For edges, states of all types are stored with one double-keyed map. The two keys are the two nodes that form an edge (the first key is an owned node; the second key is an influencer) and the value `v` is an array of pointers to void. This enables one array element to be (cast to) pointers of each of `int`, `double`, and `char`.

This latter scheme saves storage by reducing the number of map entries. Edge traits are stored in the same way.

Table 3.1: Data structures for the MPI parallel Algorithm 2.

Entity	**Description**
Graph partition.	Adjacency list (map with keys `k` being owned nodes and values `v` being nodes that influence the key).
Node states and traits.	Template map for each kind of state or trait (keys `k` are node IDs and values `v` are arrays of primitive types).
Edge states and traits.	Double-keyed map where all types for an edge are contained in one value `v`.
IMs (interaction models).	Map with `k` being the owned node ID and `v` a IM instance.
Message payloads.	Character strings of variable length, depending on numbers of nodes transitioning state and sizes of states.

State and trait values are stored similarly for nodes and edges, but states and traits are manipulated differently. For example, if a node requires four `int` traits, then each node is provided an array of four `int`s. However, if four `int` states are required, then an array of eight `int`s is produced. The size of a state array is double the required number of values, for the following reason. ENSIM permits many diffusion instances to be run with one code execution. States (and even traits) may evolve in time. At the end of a diffusion instance, the states and traits must be reset to their initial values for the start of the next diffusion instance, and hence these values must be saved. For a number a of states, an array of size $2a$ is produced such that the first a elements—the **lower order elements**—contain the original state values and the latter a elements—called the **higher order elements**—contain the instantaneous state values. In this way, the higher order elements evolve during the diffusion process and at the start of the next diffusion instance, the values of the lower order elements are copied into the higher order elements (see line 3 of Algorithm 2).

Traits may also be stored and manipulated like states using the "double storage" scheme described above, and in fact, both states and traits can be manipulated in more general ways than those described here. For example, there may be a lower order elements and $b(\neq a)$ higher order elements. However, the most common usage is as described here.

IMs are C++ classes that implement a particular, general interface, and quantify the system dynamics (i.e., the state transitions) through the local vertex functions. Class composition can be used to produce an IM that computes state updates for multiple contagion processes and multiple mechanisms of state transfer. There are two options for implementing IMs. First, for smaller problems where memory is not an limiting issue, each owned node of a process can be assigned its own IM object. Second, to reduce memory requirements, a single

instance IM can be used where state updates for all owned nodes are computed with a single IM instance. The former is useful when the state update code is complicated, or when there are multiple *types* of nodes in the simulation (each type requires a different IM). Like all ENSIM framework configurations, such as using a particular type of IM as described here, the framework self-configures based on a simple input field.

Finally, all MPI communications are accomplished with char array payloads. There is a class containing methods that pack and unpack arbitrary message payloads.

Time and Space Complexities

We address complexities for MPI worker processes since the master requires less storage and engages in fewer tasks than workers. Thus, the worker provides the controlling (bounding) conditions. Let n be the number of nodes and m be the number of edges in the entire graph G. Let n_w be the number of nodes and m_w be the number of edges in a worker's graph partition G'. Let w_n be the number of worker processes. We provide in the proofs below contributions to asymptotic time and space in terms of n_w and m_w for clarity of exposition, but then provide the results in terms of n, m, and N using $n_w = n/N$, $m_w = m/N$, and $O(w_n) = O(N)$.

Proposition 9. *The time complexity the parallel Algorithm 2 is* $O(m+n_i t_{max}(m/N+Nn))$.

We note that reading in values often requires time proportional to n or m, even though only a fraction of nodes and edges are actually stored. We do this to reduce memory requirements and to enable any number N of processes to be used in the MPI process group. Preprocessors can partition the data, thus streamlining these operations, but such approaches often require one to specify N a priori. Thus, if N changes, this preprocessing step must be repeated. Regardless, we believe that preprocessing steps should be included in the cost of an algorithm; we do that explicitly here. This is a point for further investigation, but for most problems, read time is a small fraction of total simulation time.

Proof. The time to read in all edges of a graph to identify and store the appropriate partition is $O(m)$. The time to read in the node properties is $O(n)$. These properties include IM IDs and their properties. Similarly, the time to read in the edge properties is $O(m)$. (These states and trait values are the values B_0.) Reading in other inputs, such as number n_i of iterations and the number of time steps per simulation instance is constant time. The time to instantiate the IMs is $O(n_w)$. The total setup time is thus $O(n + m + n_w)$.

For each simulation instance, the times to reset properties and read in seeds are respectively $O(n_w + m_w)$ and $O(n)$; the entire seed file is read by each process. We thus have over all instances $O(n_i(n_w + m_w + n))$.

We now detail the time for one time step of one diffusion instance. The time to compute next states for owned nodes is $O(n_w + m_w)$ because each node's state, and those of all its

neighbors (identified by iterating through its incident edges) must be considered. Included in this time is the time to temporarily store state updates in s_P (for later copy into v_P). The time to reset IM properties for the next state for nodes with state update is $O(n_w + m_w)$ because both node and edge properties may require resetting. For messaging, the time to pack the message payload is $O(n_w)$. The cost of sending messages is as follows. The time for one worker to send its payload to every other worker is $O(w_n n_w)$. We assume transmission cost is proportional to message size ($O(n_w)$) and overhead cost is $O(w_n)$. Thus, the total time over all w_n workers is $O(w_n^2 n_w)$. The time to store the appropriate states in v_P from all other workers is $O(n)$. Thus, the total time over all simulation instances and time steps is $O(n_i t_{max}(n_w + m_w + w_n^2 n_w + n))$. Combining all of these results, keeping the dominant terms, and recasting in terms of n, m, and N gives us the stated result. □

We note in particular that there is an $O(N^2)$ contribution to the time spent by the N processes engaged in all-to-all communications. This limits scalability to larger graphs. From performing numerous studies, some involving one million or more simulation instances over ranges in graphs and dynamical conditions, we find that the implementation runs very fast if the number of graph nodes is on the order of 100,000. For graphs with roughly 2 million nodes, as the number N of MPI processes reaches about 250, the total execution time increases with increasing numbers of processors because the increasing communication times dominate the time reductions from parallel computations of state updates.

Proposition 10. *The space complexity of the parallel Algorithm 2 is $O(n + m)$ over all N MPI processes.*

Proof. We focus on an MPI worker process, since storage requirements for it dominate those for the master process. All edges of the graph partition must be stored, which is $O(m_w)$. The node states and traits each require $O(n_w)$, and similarly, the edge states and traits each require $O(m_w)$. Assuming only nodes have state changes, the temporary storage for state updates requires $O(n_w)$. The IMs that implement the local functions require $O(n_w)$, and the character array message payloads are $O(n_w)$. Thus, the total space requirement per MPI worker is $O(n_w + m_w)$, or over the entire system, $O(N(n/N + m/N))$. □

3.3.4 Hybrid Parallel Implementation

Motivation for Hybrid Approach

We note a few features of the parallel MPI implementation described in Section 3.3.3: (*i*) the algorithm does not overlap computation and communication; (*ii*) the algorithm requires all-to-all communications; and (*iii*) there is active memory management during each time step as the computation of new states causes memory to be allocated in s_P (and for ancillary data, which is not addressed here), and then deallocated after the values are copied into v_P. These all hinder performance.

Even with these shortcomings, we find that the MPI implementation more than suffices for large parametric studies (of ≥ 1 million simulation instances) on graphs with 100,000 nodes. However, we seek to study graphs with 100 million nodes (a three order of magnitude increase) and billions of edges, and for these cases, the MPI implementation is inadequate.

We therefore designed and implemented a hybrid MPI + OpenMP simulation framework to address these limitations. The parallel algorithm of this section addresses all three of the above issues by (*i*) overlapping computations and communications; (*ii*) driving down the number of MPI worker processes to reduce communications costs and recouping the reduction in parallelism by increasing on-compute-node parallelism through multithreading; and (*iii*) removing memory allocation and deallocation for state updates by allocating once all properties storage upfront during setup. Additional points are that the threads used for on-node parallelism in computing state updates are also used to advantage at three other points in the code to move data in parallel. Also, pre-allocating memory requires that s_P is fully allocated for all owned nodes and therefore the memory footprint for s_P is a significant fraction of that for v_P (v_P also contains properties for pure influencers). Finally, there is now no master node and therefore the MPI process group is a peer group of N workers.

In addition to performance-related modifications, we also improved graph representations. First, a graph may be arbitrarily partitioned; e.g., the results from software such as Metis [161] may be used to partition the graph so as to reduce communication among MPI workers, or nodes with different IMs of varying complexity can be appropriately assigned among processes for load balancing. Second, **distance** r **subgraphs** can be stored for any r. This is a subgraph (stored in an MPI process) consisting of all owned nodes and all pure influencer nodes within distance r of every owned node (and the incident edges). Storing all owned nodes and their immediate neighbors in a process, as is typically done in graph-based simulators, is a distance $r = 1$ approach. This feature is motivated by social contagion models [316] and by graph coloring algorithms [51].

We also introduce a global worker capability. This enables the entire graph to be stored on one compute node so that global quantities, such as the number of nodes in state 17, for example, can be determined. The graph is still partitioned and an MPI process group is still used to reap the performance benefits of parallelism, but selected operations are performed on the global worker. Clearly, hardware memory limitations bound the size of graph for which this can be done. This feature enables significant new functionality; e.g., unfair update schemes [232], update schemes that change per diffusion instance or even across time steps, changes in compositions of blocks in time for block sequential update schemes, determination of global properties for interventions (e.g., all people join a revolt if over 50% of individuals less than 30 years old have joined, for fear of being left out), and joint action social models [83]. Note that this feature is different from introducing an artificial node v_a that is connected to each node of a graph and dedicating a worker process $\mathcal{E}_a$ to own only v_a. In that case, interventions such as the example immediately above can be executed because v_a is connected to all other nodes, so the state of all nodes reside on a single worker process. Hence, the fraction of nodes in each state can be computed. However, connectivity

information among all nodes is missing so this precludes, for example, the use of joint action models.

Implementation Overview

We make some preliminary remarks before describing the parallel implementation. Figure 3.3 lists features provided by our parallel implementation of ENSIM. Figure 3.4 illustrates the mechanisms by which ENSIM and dynamics models interact. There is an interface to which interaction models must conform (shown in the yellow box). The interface method signatures include arguments for essentially all information about the simulation, so they are available to the IMs. The upper gray box in the center of the figure describes methods used by an IM to retrieve state and trait data from ENSIM for computation of the local vertex functions. The lower gray box describes methods used by the IM to set new states into ENSIM. These few interactions are very general.

- Distributed and thread-concurrent processing.
- Memory management.
- Configurability (e.g., GDS update schemes, output options, graph partitioning for load balancing, directed/undirected graphs).
- Data structures and many data types (e.g., for node/edge properties, any distance r subgraph partition, global peer process).
- Interface and access methods for interaction models.
- Control flow.
- Multiple MPI-based communication mechanisms.
- Performance data collection.

Figure 3.3: Features and functionality of ENSIM provided transparently.

We now turn to the parallel design and algorithm. Figure 3.5 depicts the ENSIM high-level design, and the parallel algorithm is provided in Algorithm 3. The input to the algorithm includes all of the inputs of Algorithm 2 and two other parameters, the number (n_t) of OMP threads per process and the number of messages (n_b) in each send bundle. Because there is a dedicated receiving thread that is assigned to a core, n_t is typically one less than the number of cores in a processor. The thread receives all messages for a process and stores the raw payloads in a mutexed array of character strings. This minimizes the processing required on a received message. When all messages for a time step are received, the receiving thread

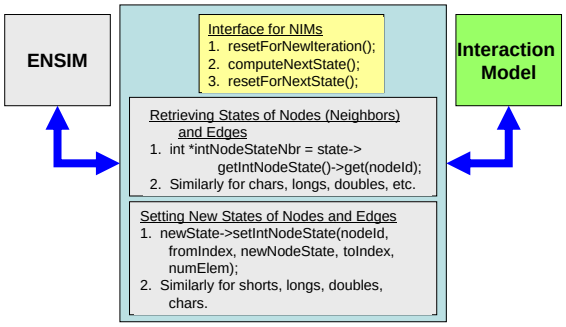


Figure 3.4: Control flow and data transfer between ENSIM and IMs take place through a well-defined interface and set of methods.

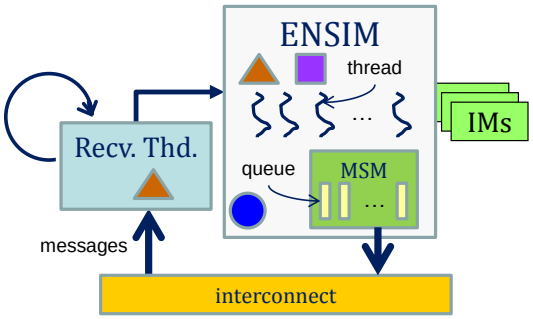


Figure 3.5: ENSIM design. Algorithm 3 describes funtionality of the main thread and OMP threads.

signals the main thread and the latter performs a parallel copy operation that unpacks the messages and puts their contents into the main properties object v_P. We now turn to the algorithm for the main thread, which also spawns OMP threads for parallel processing.

Algorithm

Algorithm 3 provides the actions of the main thread (and OMP threads) of a worker process. Many details are omitted; e.g., line 1 may have to be inside the time loop (line 3) if the graph partition changes in time; data storage for received messages in the receiving thread needs to be modified for each time step. The parallel loop over owned nodes (line 5) overlaps computation and communication, unlike Algorithm 2. Threads execute vertex functions for owned nodes, and if state is updated, a thread puts the new state on the appropriate message queues. Each process has a dedicated message queue for every other process to which it must send messages. A node v that is owned by process $\mathcal{E}$ must send v's updated state to every

other process on which v is a pure influencer. After a thread puts a message on a queue, it checks whether the queue is full. If so, it bundles the state updates into one message, flushes the queue, and sends the message. (Graph partitioning software may be used to reduce the number of required message queues.) Note that lines 2, 6, and 8 of Algorithm 3 correspond to the three interface methods of Figure 3.4.

Correctness

First we show that Algorithm 3 implements a synchronous GDS.

Proposition 11. *The parallel Algorithm 3 correctly implements a synchronous GDS.*

The proof follows the same approach as that for Algorithm 2.

Proof. Since each iteration of a simulation is independent, we focus on a single arbitrary iteration. We prove that Algorithm 3 produces the same results as Algorithm 2 for an arbitrary time step $t^* + 1$. From Proposition 2, the result follows.

We show four things for an arbitrary time step: that all required inputs for the state update for arbitrary node v on process $\mathcal{E}$ reside on $\mathcal{E}$, that all nodes have their vertex function executed exactly once, that the node states are processed correctly, and because of mutexes, that there are no deadlocks.

Let the simulation time be $t^* + 1$. Consider an arbitrary node v_i owned by an MPI process $\mathcal{E}$. The partition process ensures that all nodes that influence v_i remain neighbors of v_i in the partition. Consequently, all of the properties for v_i and all of its neighbors reside on $\mathcal{E}$. Hence all inputs for the local function f_i of Equation 1.2 for v_i are known locally. Thus, the local function for v_i can be evaluated properly on $\mathcal{E}$ at an arbitrary time $t^* + 1$.

Furthmore, the partition of the graph results in the state of each node being updated by a local function on precisely one $\mathcal{E}$ because each node is owned by exactly one process. Further, the worker threads on a process divide up the computation of the local functions such that each node's local function is executed once by one thread.

We now address the processing of state updates. For a given process, the worker threads compute the state updates of owned nodes. The inputs for the basic GDS—node states—are stored in v_P; state updates are stored in s_P (lines 6 and 7, respectively). Thus, these new states do not enter the local function computations because the values in s_P are not used as inputs to the local functions. State updates are placed on send queues that are dedicated for the other processes in the MPI process group. Messages containing these state updates are sent periodically, when the send queue is full.

Each process has a dedicated receiving thread to receive state updates from all other processes. For a given process $\mathcal{E}$, the messages it sends are precisely the states of nodes that are

Algorithm 3: Hybrid Parallel Simulation, Synchronous Update—One MPI Worker Process Main Thread

input : Graph partition G', node properties v_P, edge properties e_P, number of iterations n_i, maximum time per iteration t_{max}, base conditions B_0, dynamics model identifier(s) d_{ID}, dynamics model properties d_P, seed conditions I, sets (blocks) of nodes b_v, sets (blocks) of edges b_e, number of MPI processes n_p, number of OMP threads per process n_t, number of message per send bundle n_b.

output: Changes in states s_v of nodes v as a function of time t.

Read in all inputs and store those relevant to this process: G', v_P, e_P, d_{ID} and d_P, n_i, t_{max}, B_0, n_p, n_t.
Instantiate an IM for each owned node of G'.
Instantiate and start receiving thread (with shared object; shared with main thread).
Instantiate message sending infrastructure.

```
1  Create vector of node IDs owned by this processor (required for OMP loops).
   // Loop over diffusion instances.
   for (j = 1; j ≤ n_i; ++j) do
2      Parallel: Reset all node and edge properties and IM parameters to their
       base conditions.
       Read (initial) seed conditions I_j.
       // Loop over time.
3      for (t = 1; t ≤ t_max; ++t) do
           if (t > 1) then
4              Parallel: Copy state changes from receiving thread into main
               properties object v_P.
           // Loop over owned nodes.
5          Parallel for (v = 0; v ≤ v_max; ++v) do
6              Invoke the IM (local vertex function) for node v.
               if (state of v changes) then
7                  Store state updates in storage s_P.
8                  Reset model parameters of new state for node v.
                   Add v's state change to message queues for remote processes
                   where v is a pure influencer.
                   if (a message queue contains n_b state updates) then
                       Bundle all state updates into one message.
                       Send message.
9          Parallel: Copy owned node state changes from s_P into v_P.
           Print state changes in s_P to local file.
```

pure influencers on other processes. Hence, every receiving thread receives the state updates corresponding to time $t^* + 1$ for nodes that are pure influencers.

Only after all state updates are sent to the appropriate peer processes are the values from s_P copied into v_P. This copy operation occurs only for owned nodes on the process. Thus, at the end of time $t^* + 1$, the states of *owned* nodes in v_P correspond to time $t^* + 1$, as required. We still must update into v_P the states of pure influencer nodes for time $t^* + 1$.

Since each process prints to output the state updates for its owned nodes, and since every node is owned by exactly one process, all state updates for all nodes at time $t^* + 1$ are correctly output.

The time step for $t^* + 1$ ends with the states of all pure influencers in v_P on each process having states corresponding to time t^*. However, the first action at the start of time step $t^* + 2$ is that the main thread receives a signal from the receiving thread and states for pure influencers are copied into v_P. Thus, before any state update computations begin at time $t^* + 2$, v_P—which, again, contains the inputs to the vertex functions represented by Equation (1.2)—stores all node states corresponding to time $t^* + 1$, as required.

Lastly, we address the mutexed operations and the possibility for deadlocks. From the conditions of Coffman et al. [87], deadlocks are impossible because no thread that currently possesses a lock (mutex) attempts to acquire a second lock. There are two places where locks are acquired: (*i*) worker threads that compute state updates and must acquire locks to put updated states on send queues, and (*ii*) the receiver thread and main thread respectively write to and read from a shared object. We address each of these in turn.

A worker thread executes the vertex function for a node v. If there is no state update, the thread retrieves the next node and executes that node's vertex function. If a state update occurs for v, then the thread acquires locks, one a time, for the send queue for each process on which v is a pure influencer. When the thread acquires a lock for a single send queue, it places the node and state on the queue. If the queue is not full, it releases the queue and proceeds to acquire the lock for the next queue, if appropriate. If the queue is full, it dequeues all state updates, flushes the send queue, sends the message through MPI, and releases the lock. The thread then proceeds to the next queue, if appropriate. Hence, a worker thread only seeks to acquire one lock at a time.

The shared object used to store received messages is accessed by the receiving thread and the main thread. During each time step, the receiving thread predominantly possesses the lock for the shared object, since it is receiving messages from all other processes. After all messages are received from all other processes for a time step, which is readily determined because each message contains a flag stating whether this is the last message from a sending process for the current time step, the receiving thread sets a flag in the shared object identifying that the shared object is ready for the main thread, releases the lock, and signals the main thread. The main thread demarshalls the data and stores the pure influencer state updates into v_P, and then repeats the notification process, telling the receiving thread that

it is to begin receiving messages for the next timestep. Neither thread attempts to acquire a second lock while possessing the lock for the shared object.

□

Remark 3 and Proposition 11 provide for the following result.

Corollary 12. *The analog of Algorithm 3 for block sequential loading implement any GDS.*

We note that the MPI-based Algorithm 2 can produce deterministic results. Even if the vertex functions themselves are stochastic, running a simulation multiple times with the same number of processes will generate the same results if the same random number seeds are used. However, this is not the case for the hybrid implementation because the threads on a process execute in a non-deterministic fashion, and hence worker threads that execute stochastic vertex functions are not guaranteed to execute in the same order across simulations and hence are not guaranteed to use the same random numbers across simulations.

Data Structures

The data structures in Table 3.1 for the MPI implementation are augmented for the hybrid implementation with the structures in Table 3.2. The first structure uses two adjacency lists to store the graph partition. One list stores the $r = 1$ subgraph and the other list stores the $r > 1$ subgraph. There are efficiency reasons for separating these subgraphs; e.g., using the keys of the $r = 1$ graph enables quick identification of owned nodes. The other three groups of data structures in the table support message storage and routing. The node and edge trait and state storage structures are largely the same and are not repeated, but we instantiate duplicate storage structures to handle temporary storage associated with manipulating properties for different update schemes.

Time and Space Complexities

Proposition 13. *The time complexity the parallel Algorithm 3 is* $O(m+n_i t_{max}(m/(Nn_t)+n))$.

Proof. The complexity of the setup time is the same as that in Proposition 9, $O(n+m+n_w)$. The complexity of resetting properties and reading seed values over all diffusion instances is also the same as in Proposition 9, except that resetting properties is done by $n_t - 1$ OMP threads, and therefore $O(n_i((n_w + m_w)/n_t + n))$.

For one time step, the total complexity to compute all state updates, store all updates in s_P, and reset IM properties for all next states, each of which is done in parallel, is $O((n_w + m_w)/n_t)$. The time to store data in $N - 1$ message queues, copy state updates

Table 3.2: Data structures for the hybrid (MPI + OMP) parallel Algorithm 3. Node and edge properties structures and IM structures are reused from Table 3.1; here we provide only those structures that have been modified or are new.

Entity	**Description**
Two graph partitions.	One adjacency list (map with keys `k` being owned nodes and values `v` being nodes that influence the key). A second adjacency list for all edges where $r \geq 2$.
Messaging infrastructure.	Each MPI process p has a collection of message queues, one for each process to which p sends state updates. Additionally, MPI structures such as requests and status objects are instantiated per queue. Mutual exclusion locks are used to ensure that only one thread at a time can modify a queue.
Message Routing.	Maps to identify which processes need to transmit state updates. A send map has owned nodes v as keys and the corresponding values are the sets of MPI ranks for which v is a pure influencer. A receive map has pure influencer nodes as keys and the corresponding values are MPI ranks from which these state updates are received.
Message Receiving.	The shared object (between the receiving thread and the main thread is an array of character strings.

into character strings, and create message payloads from the strings, all of which are done in parallel, is $O(Nn_w/n_t)$. (This time does not include wait time to acquire mutexes for message queues.) Message costs are as follows. Using asynchronous communication, which is performed sequentially on the data from the message queues, we assume a cost proportional to the total message payload size, which is $O(Nn_w)$. Assuming overlap between message sending and receiving with the dedicated receiving thread, we have no time contribution from the receiving thread. The time to copy in parallel all state updates (for owned and pure influencer nodes) into v_P is $O(n/n_t)$. Thus, the total time over all time steps is $O(n_i t_{max}((n_w + m_w)/n_t + Nn_w/n_t + Nn_w + n/n_t))$. Summing all terms and retaining the dominant ones gives the result. □

Proposition 14. *The total space complexity of the hybrid parallel Algorithm 3 is* $O(nN+m)$.

Proof. All non-messaging space complexity contributions are the same as in Proposition 10, which is $O(n_w + m_w)$ per process. For messaging, the message queues contribute $O(w_n n_w)$ ($O(w_n)$ queues and $O(n_w)$ state updates per queue). The send and receive maps for routing messages are both $O(w_n n_w)$. Multiplying each contribution by the number N of processes and retaining the dominant terms yields the result. □

3.4 Networks

Details regarding the networks used in our experiments are provided in Table 3.3. The real contact networks, generated from the procedures in [31], range in size from 2M (million) nodes (Miami) to 100M nodes (the East Coast of the United States), and up to 2.8B (billion) edges. For the regional graphs, starting with North Easetern US, we begin with the generation of state networks according to [31]. To model interactions between nodes of different states, we create 10^4 random edges between nodes across states that share a border.

The Twitter network is a follower network. The mutual Twitter graph contains an undirected edge $\{a, b\}$ if and only if a follows b and b follows a [127]. In our experiments, we treat all networks as undirected to spur greater numbers of state transitions and thereby tax ENSIM's communications. Synthetic networks, which are Erdős-Renyi random graphs, range up to 8M nodes and 120M edges such that each graph has $d_{ave} = 30$.

Table 3.3: Networks studied in this work. There are seven Erdős-Renyi random graphs, ranging in size from 1.6 million to 32 million nodes, and 24 million to 486 million edges, each with average degree 30, for the weak scaling studies.

Network	**Number of Nodes**	**Number of Edges**	**Average Degree**	**Ave. Clustering Coefficient**
Miami [31]	2,092,147	52,700,258	50.4	0.4045
Washington, DC [31]	3,951,853	110,453,954	55.9	0.4066
Chicago [31]	9,038,414	268,786,768	59.5	0.3893
New York City [31]	17,876,290	480,115,815	53.7	0.4115
Twitter [190]	41,652,230	1,202,513,046	57.7	0.08330
Twitter (mutual followers)	22,580,393	265,851,838	23.5	0.1456
Erdős Renyi (multiple)	(1.6 to 32.0) $\times 10^6$	up to 486,000,000	30	7.5×10^{-5}
North Eastern US [31]	74,808,390	2,112,825,037	56.5	0.4001
Eastern US (EUS) [31]	104,149,595	2,844,381,660	54.6	0.3928

Clustering coefficient distributions for each category of graph (city, region, Twitter) are given in Figure 3.6, while degree distributions are given in Figure 3.7.

3.5 Performance Evaluation

3.5.1 Experimental Setup

In all of our performance simulations, nodes may be in one of two states denoted by 0 and 1. State 1 (respectively, 0) is sometimes called the **affected** (**unaffected**) state, mean-

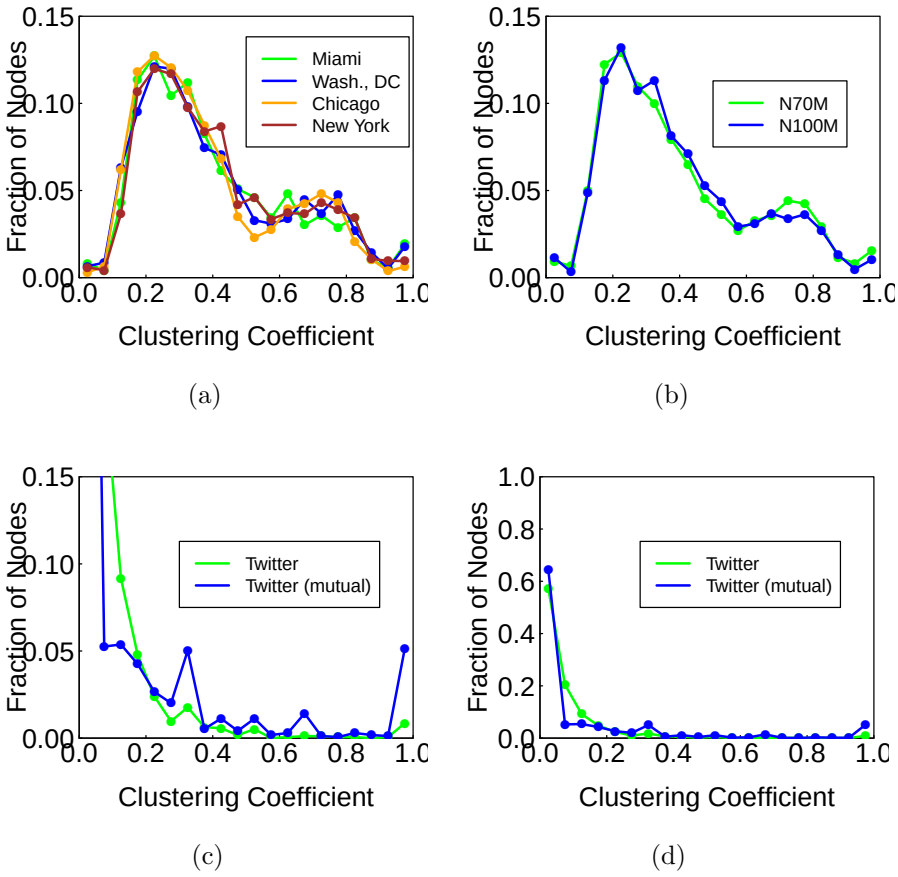


Figure 3.6: Clustering coefficient distributions binned by number of nodes with values in 0.05 increments of clustering coefficient: (a) cities; (b) regions of the US (N70M means a 70 million node network of the Northeastern US; N100M means a 100 million node network of the Eastern US); and (c) and (d) Twitter graphs (different scales).

ing a node possesses (does not possess) a contagion. In addition, the models used in the simulations represent simple, complex or generalized contagions. There are three reasons for these choices. First, these models are employed to understand human behavior in real events involving tens of thousands to tens of millions of people; e.g., the dynamics of protest spread [127], and dynamics of joining social media and using it [316]. These are also the types of models used in various studies described in Section 1.3. Second, these models yield a rich variety of behaviors that enable us to understand more readily performance issues for ENSIM. These types of models are also used in other performance evaluation studies (e.g., [91]). We use uniform thresholds for all nodes in a simulation; this is to better illustrate the effects of different thresholds. Most, but not all, results use 1-threshold models, be they deterministic or stochastic, because this maximizes the number of nodes that can transition, which increases the messaging and therefore overhead associated with ENSIM execution. In

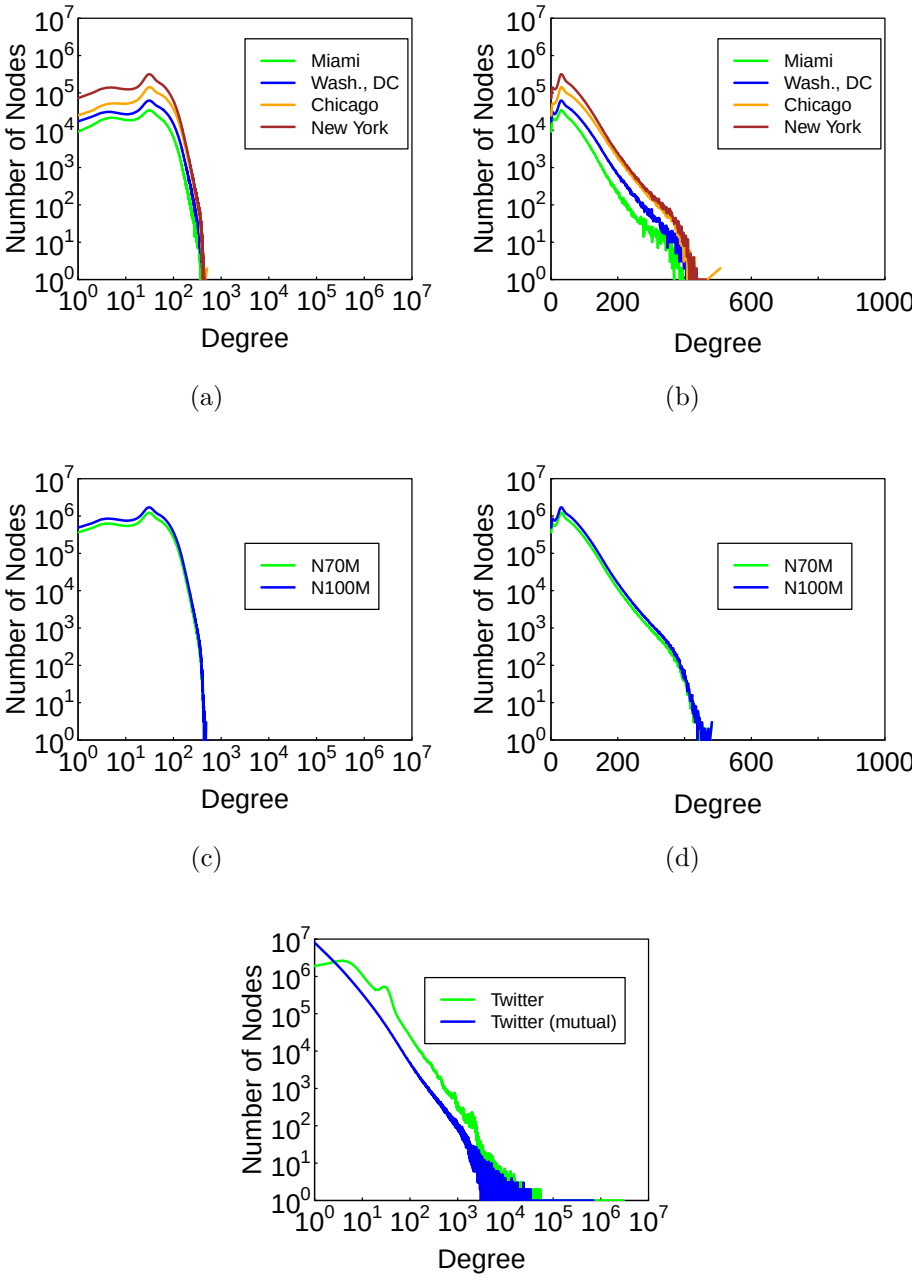


Figure 3.7: Degree distributions: (a) and (b) cities; (c) and (d) regions of the US (N70M means a 70 million node network of the Northeastern US; N100M means a 100 million node network of the Eastern US); and (e) Twitter graphs; there are nodes with degrees near 3 million. Figures (b) and (d) illustrate that the city and regional degree distributions are exponential-decay (with a knee near degree of 400), while the Twitter graphs are roughly scale-free.

this sense, our performance results are therefore types of worse cases. Third, we can use these diffusion models to control the amount of computation and thereby gain additional insight into system behavior.

A **progressive** IM allows nodes to transition from state 0 to state 1 (i.e., $0 \rightarrow 1$), but does not allow the transition $1 \rightarrow 0$ [166]. Use both deterministic models and stochastic models, and STMs of Figure 1.3(a) and 1.3(b). Since stochasticity reduces the number of state transitions, the probabilistic models induce greater computational load. A **back and forth** IM allows both $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions. We use the STM of Figure 1.3(e) for these studies. Note that this simple STM can induce extreme loading in the system; for no matter what the state of a node, it can always transition to another state. We find that, consistent with theory [187], two-cycles are often generated in these systems, meaning that a system will oscillate between two system states, and that a large fraction of nodes will transition at each time step. A **connected component threshold** IM [316] means that in order for a node v to change state, the number of connected components induced on the neighbors of v that has one or more nodes in the opposite state must be at least the given threshold θ. The connected component IM is another STC for the STM of Figure 1.3(e).

Data for both the MPI and hybrid versions of ENSIM are provided. Unless specified otherwise, every execution uses all 16 cores of a compute node. Hence, a job using 3 compute nodes uses 48 cores. We plot results using either numbers of cores or numbers of compute nodes, which ever is most useful. Differences stem from the fact that the MPI version executes by running an instance of ENSIM on each core, while the hybrid version runs one instance of ENSIM on each compute node and then uses the cores for running threads. Also, the receiving thread of each ENSIM instance is included in all core counts, even though it performs no state updates, since it is part of the framework's operations. In figures with multiple plots, legends are often the same in all plots and hence may only appear once.

Timing data are most often the time to execute one diffusion instance. That is, starting with only seed nodes possessing a contagion, we run 100 time steps, and the total time to run these 100 time steps is the execution duration presented. (If one desires a per-time-step value, the ordinates of plots can simply be divided by 100.) Because ENSIM can run any number of diffusion instances with one execution (i.e., one job submission), each execution runs 20 such diffusion instances, and the values provided are averages over these 20 instances. When we examine in more detail some diffusion models, we will use the duration of one time step to compare models, rather than the duration for the entire diffusion instance. Initial setup time to read in the graph and properties is omitted from all data. These are considered "sunk costs" and we have found that including these durations in performance data obscures the behavior of the system.

For strong scaling studies, we use 5 seed nodes (i.e., 5 nodes initially in state 1) with all remaining nodes in state 0. For weak scaling studies, we use a number of seed nodes proportional to the number of graph nodes. For most other studies, we use 10,000 seed nodes per instance, which enables us to examine a variety of contagion types with the same sets of

seed nodes.

All simulations were performed on a cluster of 48 Intel Sandy Bridge compute nodes (Dell C6220), with a Qlogic QDR Infiniband interconnect. Each node is a dual-socket Intel Sandy Bridge E5-2670 2.60GHz 8-core processor (16 cores per node) and has 64GB 1600MHz DDR3 RAM. We performed a few simulations by combining this cluster with a comparable cluster of 52 Dell 6100 12-core compute nodes and Qlogic QDR Infiniband interconnect to reach a maximum of 1200 cores.

3.5.2 Overview of Results

We summarize the experimental results of the following subsections.

1. The peformance of the hybrid implementation over the MPI implementation, in terms of exeuction time, is 300% for 40 compute nodes (16 cores/node) over a range of realistic networks, and increases for increasing numbers of compute nodes. For realistic contact social networks ranging in size from 2 million to 18 million nodes, we have the following results. For small numbers of compute nodes (e.g., about 5 compute nodes), the MPI implementation is faster than the hybrid code, by about 10%. This is a result of the smaller number of ENSIM processes that must exchange messages when N is small. The hybrid code execution is slowed by critical sections. However, as N increases to 10 or more, the hybrid code execution is faster than that of the MPI code. For $N = 40$, execution time for the hybrid code is roughly one-third that of the MPI implementation, and the disparity between the two implementations grows quadratically for further increases in N. This difference is due to the fact that the MPI code does not scale beyond $N = 10$ because of the all-to-all communications, among the $16N$ ENSIM instances. The hybrid code, in constrast, scales to at least 75 compute nodes (the extent of our system).

2. The problem sizes to which the hybrid code scales are at least two orders of magnitude greater than those for the MPI code. We have demonstrated scaling to 100 million nodes for the hybrid code, and less than 1 million nodes for the MPI code, when running problems on $N = 20$ or more compute nodes.

3. We have demonstrated linear speedup, and almost a 1:1 ratio in speedup to N for the hybrid code. This is demonstrated for five realistic social networks, ranging in sizes of graphs up to 42 million nodes.

4. For STMs of the kind in Figure 1.3(b) with stochastic STCs, the fraction of total time spent in computation ranges from 74% to 90%, with the remainder of the time being overhead.

5. STMs can make a big difference in execution time. Compared to the STM of Figure 1.3(b) with a stochastic STC, the STM of Figure 1.3(e) with deterministic STCs displays an order of magnitude increase in execution duration. Speedup shows a commensurate dropoff.

6. Weak scaling results show different characters depending on STC. When the STC is deterministic, computation times remain relatively flat, while communications costs increase. However, a stochastic STC generates more computation and the computation and total time curves have a relatively constant offset for different numbers of processors.

7. Graph partitioning is much more effective in decreasing execution time for small numbers of partitions, (roughly around 4). As the number of partitions increases, the execution times on partitioned graphs can be greater than that compared to graphs whose nodes are assigned randomly to ENSIM processes.

8. The first case study illustrates two aspects of our work that are important from a human productivity standpoint (there are more data on this topic in the subsequent section). First, recall that ETEST is the end-to-end solution time and is the duration from the specification of the problem until the simulation study is complete. This includes, here, the times to design, construct, integrate, and verify two new IMs; to setup and run all experiments; and to reduce the data. This case study has an ETEST of five days. Second, the study also includes an IM with an intervention and results from it. The TTAT for all three IMs combined as slightly more than one day. We emphasize that these productivity data are informal, and formal productivity studies are complex endeavors. Nonetheless, based on our experiences in performing simulations over years in several fields (e.g., epidemiology, computer networks, social sciences), we feel that these productivity outcomes—that models can be built and integrated, and that simulations can be run and data can be reduced in five days—are significant.

9. The second case study presents what we believe are the first data that compare the ability of large contact versus social media networks to propagate complex contagions. By large, we mean networks with tens of millions of nodes. Our results show that a 44-million node Twitter network spreads complex contagions much more readily than do 40- to 100-million node contact networks. These results highlight the importance of being able to perform experiments *at scale* on real networks.

3.5.3 Strong Scaling

Strong scaling fixes the problem size and runs the same problems with different numbers of processors. The dynamics model is the following. At each time step, a node v in state 0 transitions to state 1 with probability p for each neighbor in state 1. These edge probabilities

are computed based on contact durations from detailed synthetic population generation procedures [31], and range up to about 0.4, with an average probability of about 0.054. If a node in state 1 remains affected for 3 time units, then it considers the information old and no longer passes it on (but remains in state 1). (Thus, this models a type of generalized contagion [101].) In all simulations, the contagion reached a great majority of the population (e.g., $> 95\%$ of nodes) and hence the load on the system was maximized.

MPI version. Figure 3.8 provides strong scaling results for the MPI version of ENSIM. We can see from these data that for small numbers of cores, execution duration decreases as the number of cores increases, but there is a point, roughly at about 300 cores, at which execution time levels off and then actually increases for all four networks. This is the result of the all-to-all communications, and the fact that communication durations increase faster than computation durations decrease.

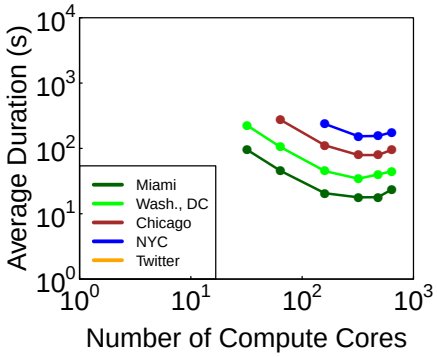


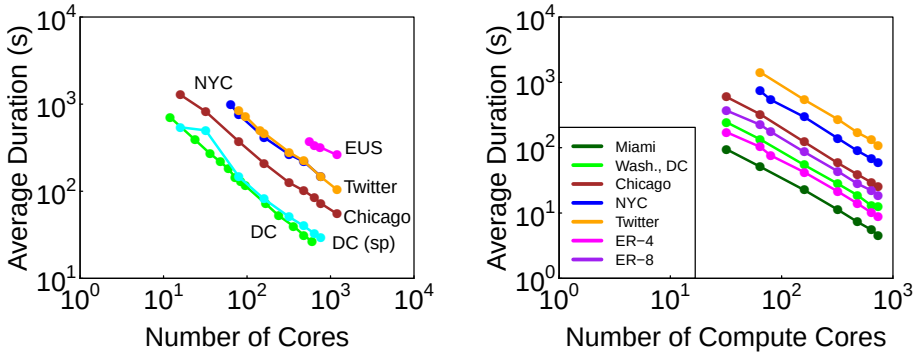
Figure 3.8: Strong scaling for several social contact networks using the MPI version of ENSIM.

Hybrid version. Figure 3.9 provides strong scaling results for several networks. The first plot contains data from a standard build of the hybrid code, and results run out to 1200 cores. This number of cores was achieved by temporarily joining to two clusters, and represents the largest jobs that we could run in this study. The Twitter and NYC data overlap in this plot because the Twitter run used a deterministic progressive threshold model, while the other runs use the stochastic model described at the outset of this section. Since nodes transition more readily in the deterministic model, the computational load is reduced and the two sets of data virtually coincide.

In the second plot, all runs use the stochastic model, and hence the Twitter runs have greater durations than the NYC runs, consistent with the fact that the Twitter network is twice as large. In addition, we also include Erdős-Renyi (ER) random graphs, so that Figure 3.9(b) includes data for ER, ED, and SF networks.

Both plots indicate that strong scaling is achieved over all numbers of compute cores for

the hybrid code, and since we have maxed-out our hardware resources, we do not know the scalability limits of the hybrid code.



(a) (b)

Figure 3.9: Strong scaling data for hybrid version of ENSIM, including ED, SF, and ER graphs. (a) Average execution time for one diffusion instance as function of number of compute cores. We used a combined job submission queue to run 100 compute nodes and 12 cores/node to reach 1200 cores. This is normal build (non-PSM). (b) PSM version of ENSIM. Average execution times for up to 46 compute nodes, 16 cores/node. ER-4 and ER-8, are, respectively, 4 million and 8 million node Erdős-Renyi random graphs. We use 1000 messages in a bundle for sending.

Speedup for hybrid version. Speedup is computed from the data in Figure 3.9(b) and is shown in Figure 3.10. There is somewhat of a trend toward greater speedup with increasing problem size (i.e., increasing graph size). The ER graphs have among the lesser speedups. We believe this is a result of the larger graphs using a larger number of compute nodes for the "base run." The larger graphs would not fit into memory, so the "base" condition is not a serial code.

Comparison of strong scaling: MPI versus hybrid version. Strong scaling results comparing MPI and hybrid versions are provided in Figure 3.11(a). These data come from Figures 3.8 and 3.9(b). The dashed curves are MPI data and the solid curves are hybrid data. For numbers of compute nodes $N < 10$, the MPI code runs about 10% faster. This improvement is because the number of cores in the all-to-all communications ($16N^2$) for the MPI version has not reached the point of diminishing returns, and the critical sections in the hybrid code causes degradation in the performance of threads. However, as $N > 20$, the deleterious effects of the MPI all-to-all communications results in poor MPI performance, as described above. Figure 3.11(b) now plots the number of compute nodes, rather than cores, and on the ordinate shows the ratio of MPI-to-hybrid total execution time for several networks and test conditions. A ratio of 3, for example, means that the hybrid code executes 3 times as fast as the MPI code. For $N = 40$, the hybrid code is 200% to 400% faster than

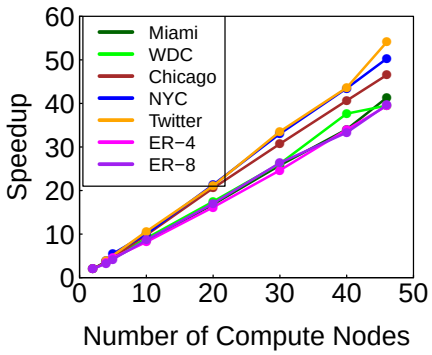


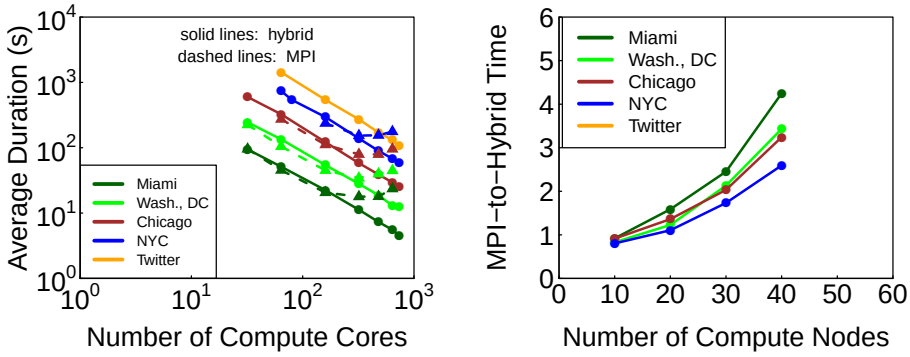
Figure 3.10: Strong scaling data for the hybrid ENSIM PSM build showing linear speedup, and almost a 1:1 ratio of speedup to compute nodes. ER graphs show lesser speedup.

the MPI code.

Ratio of computation to total time. Data illustrating the ratio of computation time to total time are provided in Figure 3.12. By **computation time**, we mean the time to execute the code for the local vertex functions. Total times are provided as solid lines and the dashed lines of the same color represent the computation time. The ratio of computation to total times is roughly 0.74 to 0.90 across all networks and conditions in Figure 3.12.

As in all plots thus far, each data point is the average of 20 measurements (i.e., 20 diffusion instances). To examine the variation among instances, Figure 3.13 shows total (solid lines) and computation (dashed lines) times for each of the 20 diffusion instances for four networks. For each network, the two curves of the same color correspond to a particular number of compute nodes. Note that ordinate scales vary among the plots. Ratios are generally in the range 0.74 to 0.90, with the majority of data points falling in the range 0.74 to 0.80. The main reason for the slight decrease in this ratio, as network size increases, is because the total time includes the time to reset the system in preparation for the next diffusion instances. Larger graphs require more reset time.

Effect of interaction model on scalability. We compare the performance of ENSIM with stochastic progressive and deterministic back-and-forth models in Figure 3.14. The solid curves, for the stochastic progressive model, are those in Figures 3.9(b) and 3.10. The dashed curves correspond to the data for the back-and-forth model. It is clear that the back-and-forth model is more computationally intensive and produces less speedup. For the same network and number of processes, the execution time for the back-and-forth model is roughly an order of magnitude greater than that for the stochastic progressive model. The model is a worse case in that a large fraction of nodes is changing state at each time step, meaning that not only do computations contribute to a high load, but also communication loads are very high and sustained over all time steps. These results clearly demonstrate that



(a) (b)

Figure 3.11: Comparisons of MPI and hybrid versions of ENSIM for PSM builds. (a) Average execution time for one diffusion instance as function of number of compute cores. The lack of scaling of the MPI version produces large differences in performance as number of compute nodes increases (hybrid code uses 16 cores/compute node). (b) Ratio of MPI-to-hybrid compute times for the same problems and numbers of cores; for example, a ratio of two means that the execution duration for the hybrid code is one-half that of the MPI duration. We use 1000 messages in a bundle for sending.

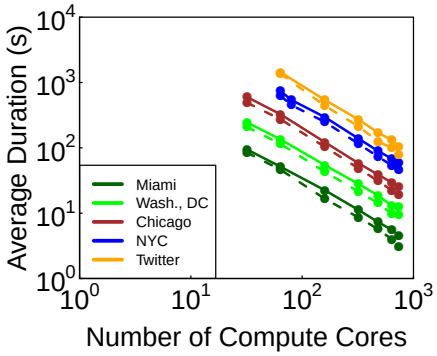


Figure 3.12: Strong scaling data for the hybrid ENSIM PSM build where solid lines represent total time and dashed lines represent compute time (i.e., compute time is the time to execute the local vertex functions).

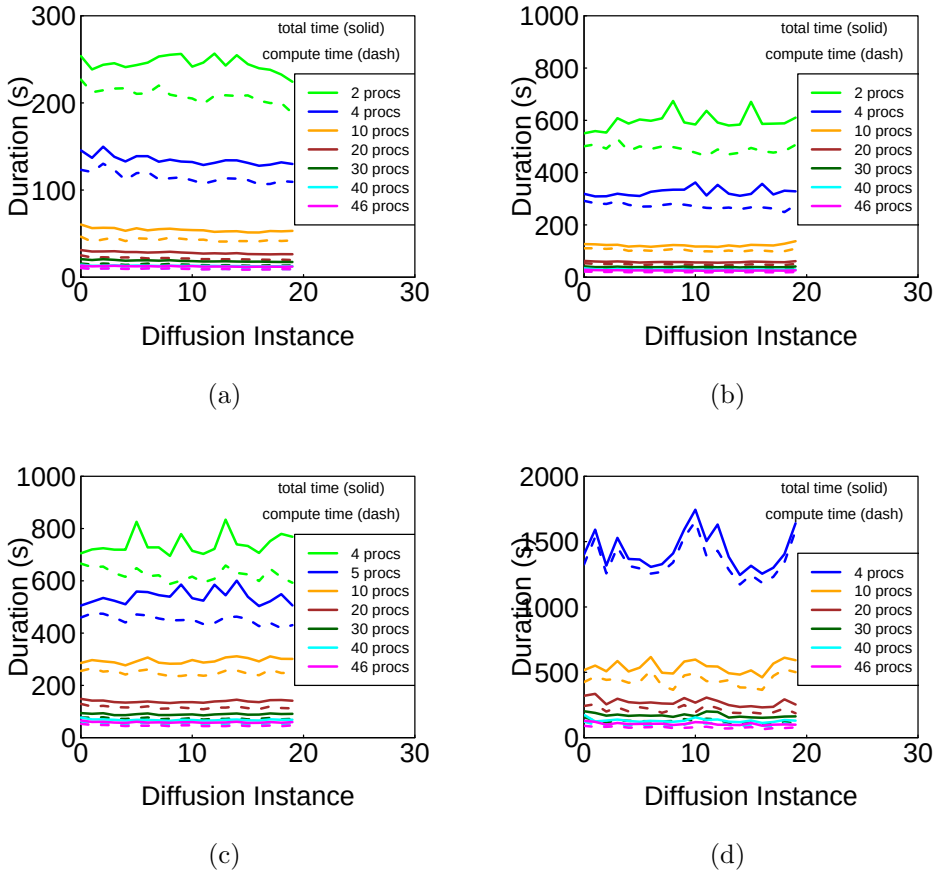
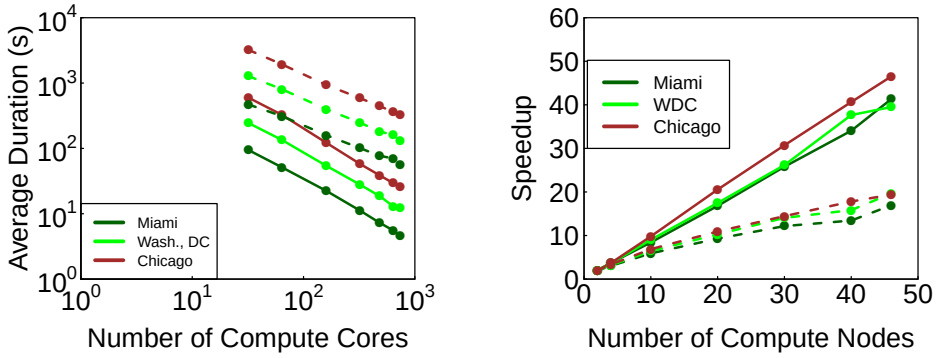


Figure 3.13: Total time (solid curves) and compute-only times (dashed curves) for each diffusion instance of a simulation. Each color corresponds to numbers of compute nodes (and equivalently, numbers of ENSIM processes) ranging from 2 to 46 (16 cores/compute node). (a) Washington, DC. (b) Chicago. (c) New York City (NYC). (d) Twitter.

the performance of ENSIM is predicated on the dynamics model employed. More detail on the performance of these and other models is given in Section 3.5.6.



(a) (b)

Figure 3.14: Comparison of stochastic progressive (solid lines) and deterministic back-and-forth (dashed lines) models: (a) strong scaling data and (b) speedup. The back-and-forth model requires far more execution time and produces less speedup.

3.5.4 Weak Scaling

Weak scaling entails fixing the problem size *per computing element* and varying the problem size and number of compute nodes in tandem such that the ratio of problem size to compute node is constant. We again use all networks. We use two models: a deterministic progressive threshold model and the stochastic model of the strong scaling studies (see Section 3.5.3). For the deterministic model, we use a constant 800000 owned graph nodes per compute node and for the stochastic model, we use 1 million owned graph nodes per compute node.

Effect of deterministic IM. Figure 3.15 provides weak scaling results for ER graphs using a deterministic progressive 1-threshold model. Computation time holds steady beyond small numbers of cores, as the number of cores increases, while the total time, and hence overhead, increases. The increasing amount of overhead is due to the all-to-all communications.

Effect of stochastic IM. To compare the effect of IM, Figure 3.16(a) provides weak scaling results for the same ER networks, but now the diffusion model is the stochastic model described in Section 3.5.3. Note that the compute and total times are much greater than those in Figure 3.5.4 and that the curves are closer together. These differences are due to the stochastic model. In this model, a vertex function may compute conditions that would give rise to a state change, but owing to stochasticity, the state change only occurs with probability 0.054. If a state change does not occur, the same computation must be repeated at the next time step. Thus, for the stochastic model, there are more executions of the

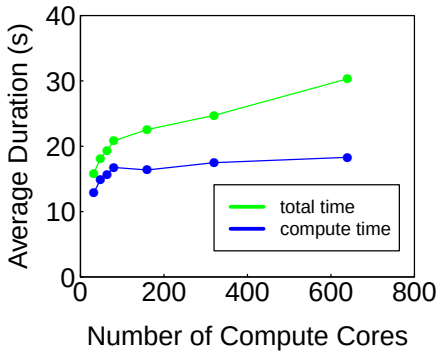


Figure 3.15: Weak scaling for Erdős-Renyi random graphs that systematically vary in size from 1.6 million nodes to 32 million nodes, with constant average degree, as the number of compute nodes range from 2 to 40 (this corresponds to 32 to 640 cores). The 32 million node graph contains 486 million edges. The diffusion model is a deterministic progressive model. These results are worst case in that the experiments require all-to-all communications and require that the state update for every node v on process $\mathcal{E}_v$ must be sent to every other process. The ordinate is the average time to complete one diffusion instance. The workload is a constant 800000 graph nodes per compute node. Numbers of seed nodes are proportional to graph size (e.g., 100 nodes for the smallest graph and 2000 for the largest). 1000 messages in a bundle for sending; message bundle sizes of 100 and 1000 showed negligible difference.

vertex functions, which increases computation time and therefore, total time. The overhead goes down, because, although the amount of computation increases, the amount of messaging does not: messages are sent only when state changes occur. Thus, the computation and total times are closer together. These data illustrate how the performance of ENSIM is affected by the diffusion model; i.e., it can be difficult to make assessments about the scalability of ENSIM without consideration of the IM.

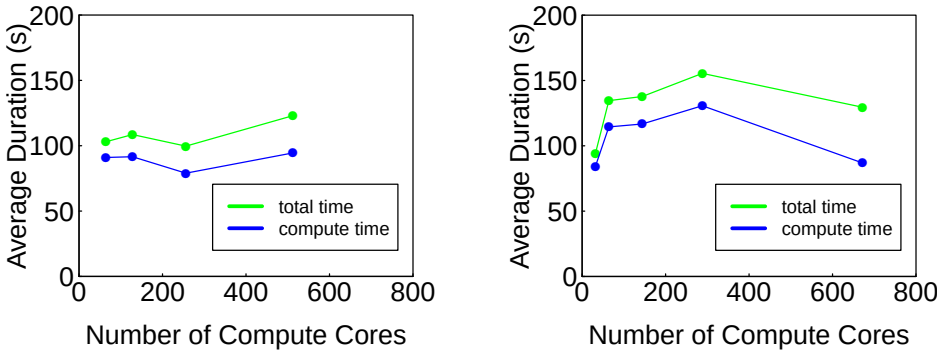


Figure 3.16: Weak scaling for realistic social and synthetic ER graphs. All data points are for a constant workload per ENSIM process at 1000000 million graph nodes per process. Each process uses 16 threads. The ordinate is the average time to complete one diffusion instance. There are 1000 messages in a bundle for sending. (a) ER networks ranging in size from 4 to 32 million nodes. (b) Five realistic social networks (four contact networks and one Twitter network) that systematically vary in size from 2 million nodes to 42 million nodes. These networks possess ED and SF degree distributions. The SF Twitter network corresponds to the largest graph (right-most data point). Hence, for example, the Miami network was run with two compute nodes and the Twitter network was run with 42 compute nodes, with other networks using intermediate numbers of compute nodes. The ratio of compute time to total time varies from 0.89 for the Miami (i.e., smallest) network to 0.84 for the NYC (i.e., largest contact) network, indicating a roughly constant ratio. The Twitter network has a ratio of 0.67.

Weak scaling data for the contact and social networks are given in Figure 3.16(b). The workload here is also constant at 1 million graph nodes per compute node. The first four data points in each curve are for the ED contact networks Miami through NYC, in increasing graph size. The last data point in each curve is for the Twitter network, which is the largest. There are interesting trends. First, the durations are greater than those for the ER graphs in Figure 3.16(a). This is most likely due to the distribution of graph nodes among compute nodes. For ER graphs, random assignment of nodes will result in a good load balance, owing to the structure of ER graphs. This is not so for the ED networks that are representative

of the contact networks studied here, so the load balance is not quite as good. The ratio of computation to communication, is comparable. Another difference is the fact that the ER graphs have average degree of 30, while the contact networks have averge degree > 50, and thus, more computation is required to iterate through all of a node's neighbors.

The last data point in each curve of Figure 3.16(b) correspond to the SF Twitter network. The average durations decrease significantly. The cause of this behavior is being investigated, but one factor may be the large-degree hub nodes in this network, and their ability to transmit contagion comparatively rapidly throughout the network.

3.5.5 Effect of Graph Partitioning

Metis is run several times for each graph, where the number of partitions varies according to the number of ENSIM processes in the distributed processing group. The goal is to minimize the number of edges across node partitions, because these edges represent required communications of state updates among ENSIM processes. (If no edges spanned two partitions, then no communications would be required among processes during simulation.) Figure 3.17 shows the reduction in execution duration when using Metis to partition the four graphs, compared to the execution times where graph nodes are randomly assigned to ENSIM processes. Positive ordinate values mean that Metis-based runs reduce execution time; negative ordinate values mean that Metis-based runs increase execution time. For $N = 2$ to 4, the reductions in execution times are between 9% and 21%. As graph size and numbers of ENSIM processes increase, the effectiveness of partitioning decreases. For larger graphs and larger numbers of processes, Metis partitioning actually increases the execution time. There are at least two reasons for these increased times due to partitioning. First, the graphs may not effectively partition for numbers of partitions in the range 20 to 40. Second, the partitioning of edges does not mean that the contagion must use all of these paths; for simple contagions, only one path is required. Thus, minimizing numbers of edges between partitions does not imply that the number of messages is minimized as contagion spreads through the network. More work remains to be done in this area.

3.5.6 Effect of Dynamics Model on Performance

We now look at combinations of graphs and dynamics models to compare population dynamics and framework performance. We demonstrate that IMs and network structure can significantly affect performance. Note that we are plotting the average duration of one time step, and plotting these data as a function of time step. These data differ from previous timing plots of this chapter, where we plotted the average time for a diffusion instance, which is the average time over all time steps. Hence, here we are looking at finer granularity in time.

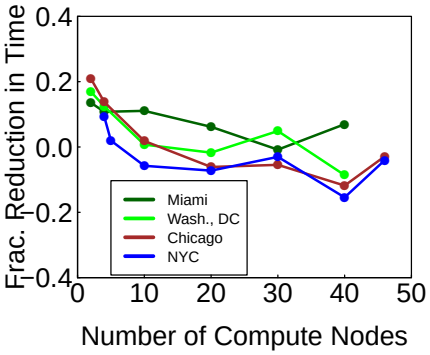
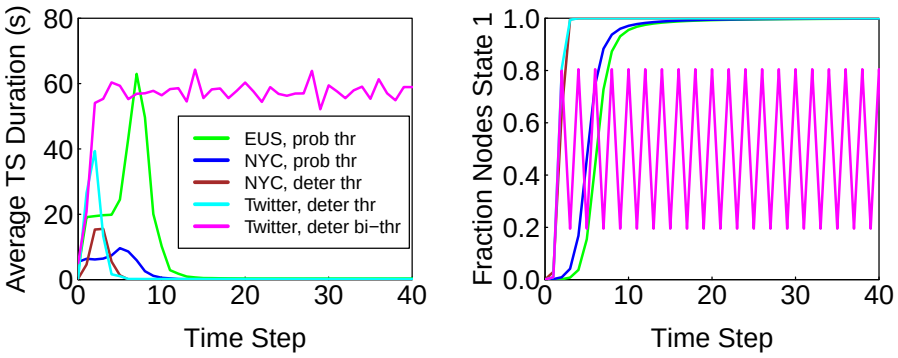


Figure 3.17: Fraction reduction in execution duration for a diffusion instance by using Metis to partition the networks. The ordinate is $(y - x)/y$, where y is the execution duration for random assignment of graph nodes to ENSIM processes and x is the execution duration when the graph nodes are partitioned using Metis.

Even simple contagion models can produce widely varying dynamics, as can be observed in Figure 3.18. We now systematically compare pairs of curves in that figure, as we move down the conditions identified in the legend. The green and blue curves (for EUS and NYC, respectively, show the effect of network size on the average time for a timestep (TS) over multiple diffusion instances, since they use the same probabilistic progressive threshold model. (With probability $p = 0.05$ an affected node passes on a contagion to an unaffected node.) The Eastern US (EUS) graph is roughly five times larger than the NYC graph; the peak computation times for these networks vary by a factor of six (10 for NYC compared to 60 for EUS). The factor of six would be even greater had we been able to run the EUS network with 480 cores, as was done for the other simulations, but EUS required more memory and hence we used more processors; see the figure caption. From Figure 3.18(b), the rates at which nodes transition to state 1 are comparable, on a fractional basis.

The brown and blue curves for NYC illustrate how the execution time is driven by the early time steps when the diffusion dynamics is deterministic (brown curve). This is due to the fact that nodes reach state 1 faster in deterministic models.

The brown and light blue curves (NYC vs. Twitter) show the effects of network size and structure. Even though the Twitter network has over twice as many nodes, the time range over which nodes change state from 0 to 1 for the two models is about the same. This is also observed in Figure 3.18(b), as both networks result in all nodes reaching state 1 by time 5. The speed with which the contagion propagates in the Twitter network is greater because it has a scale-free degree distribution compared to the exponential decay distribution for NYC (see Figure 3.7). This greater connectivity of the highest degree nodes in the Twitter network means that the Twitter contagion reaches more nodes faster, compared to a graph of comparable size but lacking high-degree hub nodes.



(a) (b)

Figure 3.18: Effects of networks and diffusion models on timing profiles and system dynamics. In all cases, 480 cores were used except for the 100M node EUS graph, which required 640 cores to fit into memory. (a) Average execution time at each timestep, per processor, for one diffusion instance. (b) Average number of nodes in state 1 versus time.

The magenta and light blue curves show the effects of dynamics models, where the magenta curve allows nodes to transition back and forth between states 0 and 1 (the bi-threshold model); all other results are for progressive $0 \rightarrow 1$ models. See Chapter 7 for a description of the bithreshold model. Hence, the number of nodes in state 1 oscillates and the computation and communications loads remain high. Oscillations in the number of nodes in state 1 in Figure 3.18(b) for the Twitter back and forth model are readily apparent. Such oscillations in user sentiment and emotions have been inferred from Twitter data [56]. It is clear from this handful of straight-forward models that ENSIM performance can vary significantly among them.

We now investigate in more detail two back and forth (baf) models. Figure 3.19 shows results for the Chicago network where two different IMs are considered. In both models, the STM is that of Figure 1.3(e), so that nodes can transition back and forth between two states. However, the STCs are different. In the basic baf model, a node transitions from state 0 to 1, and from 1 to 0, based the number of its neighbors in the opposite state, and on its up- and down-thresholds (the same legend applies for all plots). For all data in Figure 3.19, the up-threshold is set equal to the down-threshold. Figure 3.19(a) shows the system dynamics as a function of simulation time, for four sets of thresholds. For threshold $\theta = 4$, no state transitions occur. Figure 3.19(c) shows the corresponding average execution time per time step. The STC for the second model is based on connected components (cc). More precisely, for a node v, the threshold is specified on the number of connected components of the subgraph induced by the distance-1 neighbors of v that have at least one neighbor in the opposite state. The model captures the idea that each *context* in which a person operates influences her. For example, a person's family is one context, work colleagues is another, friends form another, and so on, and each of these contexts provides influence. This latter

model was developed in [316] from analyses of 10s of millions of Facebook users, and the dynamics of how people join Facebook. The results for this model are presented in Figures 3.19(b) and 3.19(d).

The state oscillations in Figures 3.19(a) and 3.19(b) continue in time, although we only show data for the first 20 time units. This behavior represents a *2-cycle*, where the system state changes back and forth between two configurations. (An example of such a cycle was also given in Section 1.5.) These two models generate, by definition, the same simple contagion dynamics. Their complex contagion behaviors, however, are quite different. For example, for up- and down-thresholds of 3, Figure 3.19(a) shows that the system state contains 40-60% of nodes in state 1, while Figure 3.19(b) shows no nodes in state 1. Important for performance, the execution times also vary significantly across models because the computation of connected components for each owned node on each time step is expensive, which is why the cc baf model has greater execution times.

3.6 Case Studies

We provide two case studies. The first examines the spread of ideology or opinion that could culminate in protests, demonstrations, or other civil unrest. It illustrates the use of multiple IMs and how results generated with these models can be used to specify policy (in the form of interventions) and their effect on population behavior. The second example explores the role of network structure in evaluating the ability of large realistic networks to propagate complex contagions. Both of these case studies demonstrate the utility of being able to study social dynamics at scale.

3.6.1 Policy Decision and Evaluation Study

We examine the spread of opinion in a population, such as if and when to join a demonstration, social movement, or revolt. It can also be used to understand marketing campaigns and how people's inclinations toward a product change based on individuals' interactions. This setup can also be applied to any behavior where peer influence is a dominant factor, such as youth smoking [144]. Here, we will focus on dissension and implications for demonstrations. Our model incorporates features called for (but not implemented) in [249]; e.g., the possibility of individuals both increasing and decreasing their commitment to unrest over time.

The basic setup is this. There is a population that initially is overwhelmingly content with the status quo. A set of seed nodes with different degrees of discontent are introduced. These comprise a small fraction of the entire population; $< 0.2\%$. The dynamics consist of pairwise interactions among people in which influence may or may not be transmitted from one person to another during an interaction. Influence may be one-way or reciprocal, and the

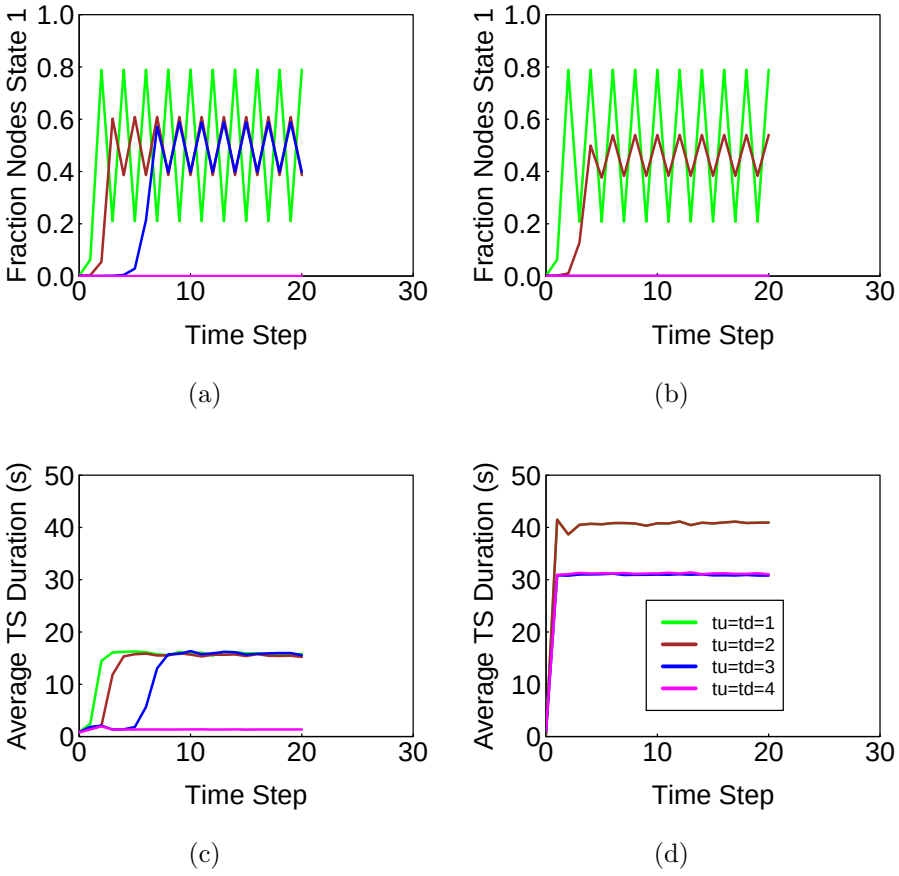


Figure 3.19: Effect of cyclic FSMs and various up- (tu) and down-thresholds (td) for node state transitions and timing profiles in the Chicago network. (a) Average fraction of nodes in state 1 as a function of simulation time for the back-and-forth (baf) model. (b) Average fraction of nodes in state 1 as a function of simulation time for the connected component (cc)-baf model. Thresholds 3 and 4 allow no diffusion. (c) Average duration of execution of time steps for the baf model. (d) Average duration of execution of time steps for the cc baf model. The connected components IM reduces the maximum threshold at which dynamics occur and drives up the duration of execution because computing connected components is costly.

degree of influence can be asymmetric. The states of individuals (nodes) in the networked population are computed in time, with the goal of determining whether and how dissension spreads through the population. We investigate the effects of several parameters, including how a government might respond in trying to deter a protest.

Recall that TTAT is the elapsed wall clock time to design, implement, integrate, and verify an IM so that it is ready for use in studies with ENSIM. The TTAT for developing the base IM used in this section was less than one day. A second IM, incorporating an intervention, was built from the base model and verified in less than 2 hours. A third model, which is an extension of the first, was built and verified in about 3 hours. We use all three models here. We find that this is typical: a study often utilizes a *family* of IMs. IMs can be produced so quickly that they can become a *parameter* of a parametric study. We illustrate this here. Simulations of 20 iterations, each with the base model and the intervention model, took about 2.7 hours to complete on $N = 20$ processors, with 12 cores each. The third model took 21 hours to execute because for each time and each owned node v, the connected components of the subgraph induced by the neighbors of v in G were computed, as in Section 3.5.6.

We built a realistic network that captures social interactions of 3.4 million nodes [31] with face-to-face interactions. To these we add five electronic interactions for each person, per day. These electronic interactions represent cell phone, Twitter, and Facebook types of communications.

Since we do not have data on electronic interactions, we randomly assign edges between nodes that represent these interactions. Two graphs are generated by using two ratios of the number of electronic edges to the number of face-to-face edges: 0.10 and 0.20. Since the average degree is 56 from face-to-face contact, this means that there are 6 or 12 electronic interactions per person, on average per day. These electronic edges are intentionally random to produce some longer-range communications that can be more readily accomplished through electronic means. They contribute to a small-world effect [330]. These numbers of electronic interactions are well within the Dunbar limit of number of interactions that a person can comprehend [126].

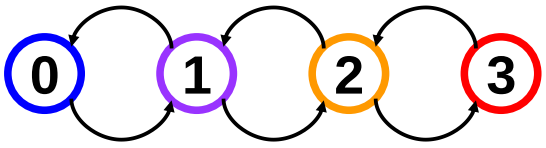


Figure 3.20: Four-state back-and-forth finite state machine.

Dynamics are as follows. States and transitions are shown in Figure 3.20. A node can be in one of four states: 0 indicates contentment with the status quo and 3 means extreme dissension, with intermediate states representing corresponding levels of dissension. Gradations in opinions and participation are well-documented [107, 212]. Each node engages in

two types of interactions, face-to-face and electronic, as described above. These interactions may exert influence from one node v_i of an edge to the other, v_j, creating a driving force for a node to *increase* its state. Since this influence process is stochastic, so that influence may not be transmitted, we call an interaction that indeed tranmits influence an **influential** interaction. (A non-influential interaction is an interaction where no influence is conveyed.) The probability of an *influential* face-to-face interaction (e.g., v_i successfully influences v_j to consider getting more involved) is a function of the time Δt of interaction, and is given by $p_{ftf} = 1 - (1 - r)^{\Delta t}$, where $r = 10^{-5}$. For 5 minute and 2 hour exchanges, the probabilities that v_i influences v_j are $p_{ftf} = 0.003$ and 0.07, respectively. This formulation has its roots in epidemiology. Since electronic communication represents a specific desire to contact another person, the probability of influence is taken as a constant $p_e = 0.1$. Note that although the probability of v_i and v_j influencing each other during an interaction is the same, represented by the edge weight, stochasticity dictates that at any given time, the interactions may be asymmetric since the Bernoulli trials for v_i influencing v_j, and vice versa, are distinct. In an influential interaction, the contagion amount a_j transmitted from v_i to v_j is equal to s_i, the state of v_i, for a face-to-face interaction. The contagion amount for electronic interaction is given by αs_i, $\alpha \in \mathbb{R}$, so that electronic interactions, and even different types of electronic communication, can produce different contagion amounts. But here, we take $\alpha = 1$ as a first step. Even for two influential interactions between two nodes at the same time, the contagion transmission is asymmetric if $s_i \neq s_j$. At each time, contagion amounts are summed by each node, and compared to the thresholds of Table 3.4. Let a_j represent the sum total contagion amount received by v_j from all of its neighbors at a particular time t. If the contagion amount $a_j \geq \theta_{up,s_j}$, where θ_{up,s_j} is the up-threshold for v_j in state s_j, then v_j increases its state (for the case $s_j < 3$). If $a_j < \theta_{down,s_j}$, then v_j reduces its state (for $s_j > 0$). The down threshold essentially drives the following process: for a given state s_j, if the continued reinforcement (via contagion) is not sufficiently sustained, then v_j will decrease its commitment. Thus, an interaction that is not influential assists in driving a state decrease.

In essence, an increase (decrease) in θ_{up} makes it more difficult (easier) to increase a node's state. Conversely, an increase (decrease) in θ_{down} makes it more difficult (easier) to sustain a node's state. Thus, state transitions are based on local neighborhoods, types of interactions, probabilities of influential interactions, contagion amounts, and thresholds.

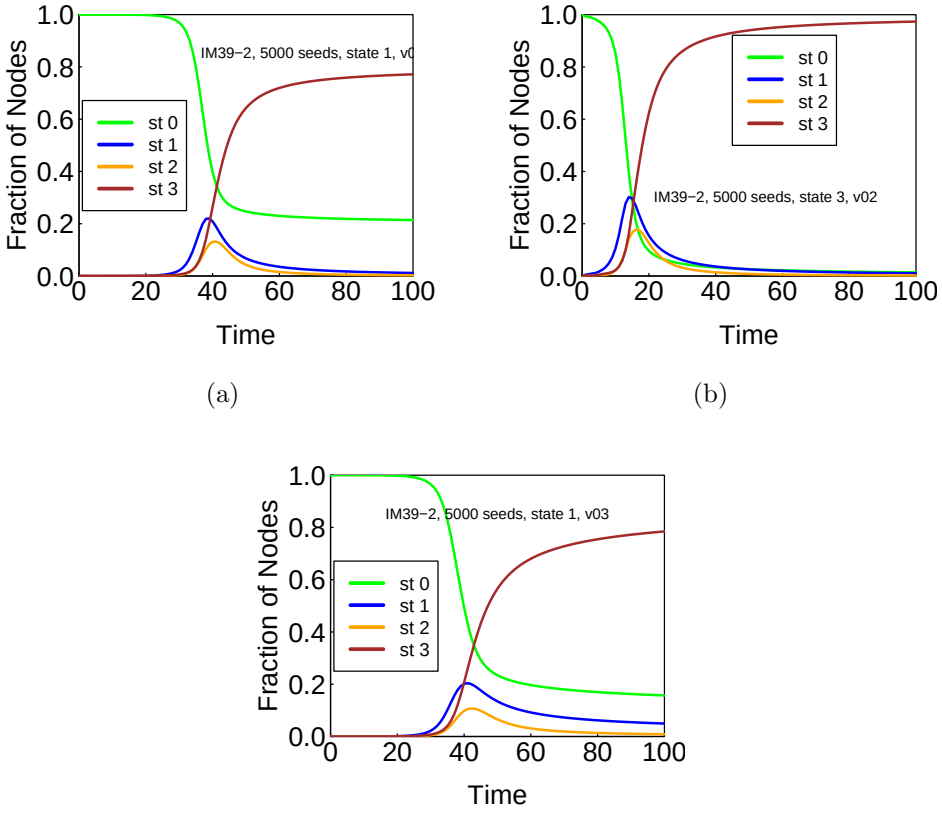
In order to understand basic system behavior, we assign all nodes the thresholds in Table 3.4. However, we can easily utilize distributions of thresholds to assign particular values to each node.

Table 3.4: Node thresholds.

State	**Up-Threshold** θ_{up}	**Down-Threshold,** θ_{down}
0	2	–
1	4	1
2	6	2
3	–	3

Initial conditions for all of our simulations are as follows. For each diffusion instance, there are 5000 randomly chosen seed nodes in the same non-zero state, with the remainder of the nodes initially in state 0 and thus are content with the status quo. This seeding represents $< 0.2\%$ of nodes and simulates spontaneous individual random"outbreaks" of discontent. Another option to explore is to simulate an initial small cadre of individuals that plot together to foster discontent; we leave this to future investigation.

We provide results for: (*i*) the effects of level of conviction of instigators; (*ii*) the effects of face-to-face and electronic communication; (*iii*) the effect of interventions by a government seeking to styme an insurrection; and (*iv*) the effects of different models on the basic dynamics. We consider a simulation timespan of 100 days to be "long term."



(a) (b)

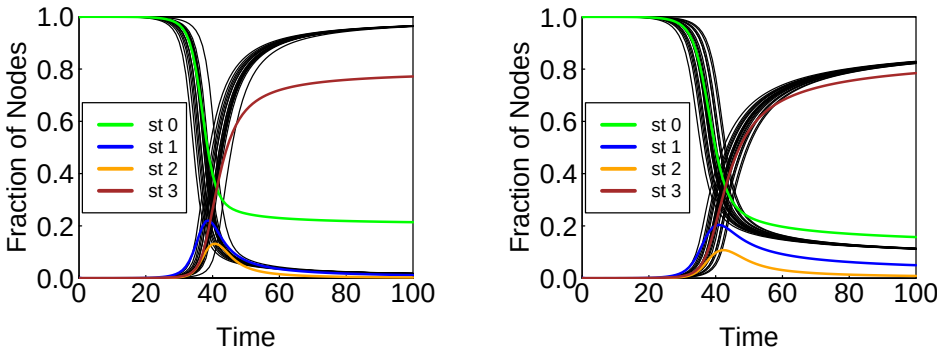
(c)

Figure 3.21: Fraction of population in each state as a function of time (in days). Initial conditions are (a) 5000 random nodes initially in state 1, and (b) 5000 random nodes initially in state 3; all other nodes initially in state 0. Results are averages over 20 simulation instances; and (c) 5000 random nodes initially in state 1; all other nodes initially in state 0 (as in (a)), but now all of the electronic communication edges are removed. Results are averages over 20 simulation instances. 5000 seeds represent $< 0.2\%$ of population.

Effect of degree of conviction of instigators. Figures 3.21(a) and 3.21(b) depict data for two different scenarios. Both plots show the fraction of the population in each of the four states as a function of time, and all curves are the averages of 20 different diffusion instances. In the first plot, 5000 randomly chosen seed nodes are in state 1, while all other nodes are in state 0. In the second plot, the 5000 nodes are initially in state 3, the most extreme state. Although state 3 eventually dominates in both, it takes twice as long for the contagion to spread when the initial nodes possess mild unrest. Also, unlike Figure 3.21(b), 20% of the population remains uncommitted in Figure 3.21(a). The reason for this is that 4 of the 20 diffusion instances do not spread the contagion at all; in these cases, the 5000 seed nodes in state 1 are insufficient to generate significant contagion spread and the dissension peters out. By time 4 in these instances, all nodes are in state 0 and the driving force for contagion spread is zero. This indicates that the initial state of 1 (or a value close to it) represents a critical point, where dissension can be either zero or widespread. Also interesting is that state 1 reaches a larger fraction of nodes than state 2 in both scenarios. It would be interesting to increase the number of states and see whether this trend continues. If so, this could have important practical applications with respect to monitoring a population for unrest, as described below.

Effect of different communication mechanisms. Figures 3.21(a) and 3.21(c) show the effect of the electronic communication edges: in the former plot they exist, in the latter they are removed. Otherwise the simulation conditions are the same. At first glance, there appears to be little difference between the two sets of results, particularly for the curves for states 0 and 3. However, there is a significant difference, as revealed in Figure 3.22. In this figure, the black curves represent the 20 individual diffusion instances, for each of states 0 and 3. In Figure 3.22(a), for instances that generate diffusion, almost all nodes end up in state 3. However, in Figure 3.22(b), where each node on average has five fewer edges (representing electronic communication that is removed), over 10% of edges remain in state 0 and there is more than a 15 percentage point decrease in the number of nodes in state 3. The four diffusion instances that result in virtually no propagation skew the average results in Figure 3.22(a) more than the single diffusion instance in Figure 3.22(b) that exhibits no diffusion. Consequently, while on average the electronic communication interactions appear to have minimal effect, they actually have a significant effect.

Effect of intervention. We investigate the effect of government intervention in trying to squelch dissension. We assume that a government monitors social sentiment [170]. When mild dissension reaches a significant fraction of the population, which we model as the condition that 15% of nodes have reached state 1, the government issues threatening warnings that make the population more apprehensive to join potests, which we model by doubling all thresholds. For our model, this means it is harder to increase one's conviction, and it is easier to decrease one's conviction. Our base cases are represented by the results in Figure 3.21(a), where all seed nodes are in initially in state 1, and in Figure 3.21(b) where all seed nodes are initially in state 3. Results incorporating the intervention are provided

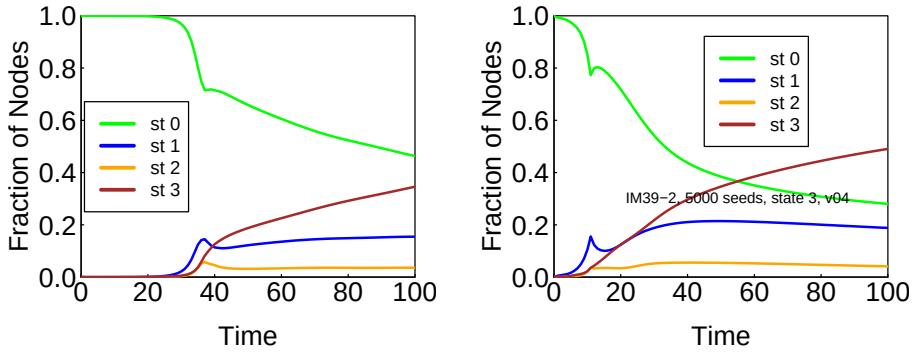


(a) (b)

Figure 3.22: Fraction of population in each state as a function of time (in days). Initial conditions for both plots are 5000 random nodes initially in state 1 with all other nodes in state 0. In (a) the electronic communication edges are present; in (b) these edges are removed. The average curves are the same as those in Figures 3.21(a) and 3.21(c). However, the black curves represent the results from individual diffusion instances. We show only the individual instances for states 0 and 3 so as to not clutter the plots. While the average curves for states 0 and 3 are similar between the two plots, the individual instances show a significant difference in the fraction of nodes in state 3: 95% of nodes in (a) and 81% in (b). The average curves in (a) are "lowered" by four diffusion instances that produce no contagion spread.

in Figure 3.23, where all seed nodes are initially in state 1 (Figure 3.23(a)) or state 3 (Figure 3.23(b)). For example, for the state 3 seed condition, the intervention causes a large disruption in population dynamics at time 12 days and in the longer term motivates 28% of the population to refrain from participation. The fraction of people in state 3, while cut in half, still represents 49% of the population. The fraction of nodes in state 1 has increased markedly. Hence, measures must be taken earlier or the action taken must be more severe, in order to halt the contagion. When the instigators possess less conviction (i.e., seed nodes in state 1), the intervention is a little more effective.

Finally, we note that many social studies implement interventions such as the one here by fixing the states of some nodes to be, in this case, state 0, and thus inhibit diffusion [7, 229, 292]. ENSIM can handle this approach, too, but here we illustrate a more subtle intervention that does not remove nodes from the graph.

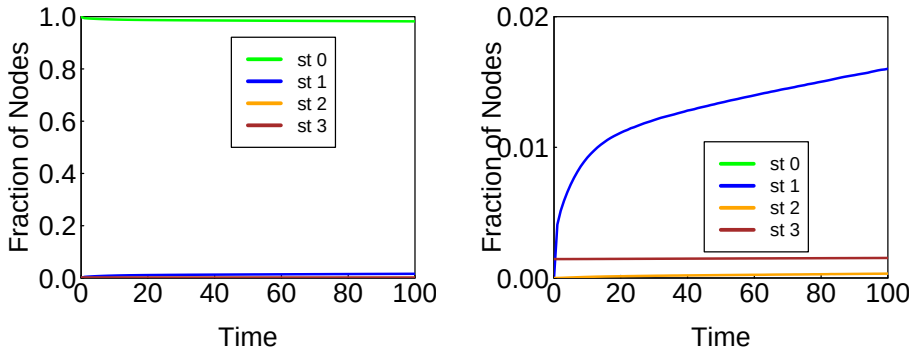


(a) (b)

Figure 3.23: Fraction of population in each state as a function of time (in days). Initial conditions are 5000 random nodes initially in (a) state 1 and (b) state 3 with all other nodes in state 0. When the government (or other opposing force) detects dissension, at the point at which 15% of the population is in state 1, it takes action that causes people to be more reluctant to participate (implemented by doubling all thresholds).

Effect of diffusion model. We now implement dynamics using the connected components model of [316], which was devised by mining Facebook data. For each node v_j, the connected components induced on the original network by the neighbors of v_j form the different contexts for influence on v_j. In effect, the connected component model, all other things being equal, reduces the influence on a node. To see this, consider a peer group of three people all in state 3, and all in the same context (i.e., connected component). In the classic threshold model, all three individuals would contribute contagion amount $a_j = 3$, for a total contagion amount of 9. However, in the connected components model, only the maximum influence from any member of the context is applied to v_j, and hence the total contagion amount supplied is 3.

Using the connected components model, and using the same properties as for the initial model, we get vastly different dynamics. First, for seed nodes in state 1, none of the 20 diffusion instances generates contagion spread. All nodes are driven to state 0 by time of 4. For the simulation with seed nodes in state 3, there is minimal diffusion. Interestingly, however, the diffusion persists, at least through 100 time steps. Data are shown in the original scale and in an expanded scale in Figure 3.24. There is a clear indication that the contagion is spreading. Whether it will eventually pervade the population is unknown, but this type of low-level unrest could be driven to a greater level with additional exogenous stimuli.



(a) (b)

Figure 3.24: Fraction of population in each state as a function of time (in days). Initial conditions are 5000 random nodes initially in state 3. This model is the connected components model with the same inputs as the other simulations. (a) normal scale and (b) zoomed ordinate scale. At time $t = 100$, there are 50000 nodes in state 1 and an additional 290 nodes in state 3, on average.

Overall sentiment. Figure 3.25 shows the average sentiment in the population as a function of time when the 5000 seed nodes are in states 1, 2, and 3, respectively (solid curves), and shows the effect of the interventions (dashed curves). Average sentiment is computed by summing the states of all nodes at each time and dividing by the population size. The data suggest that initial seeds in state 2 have the same effect as seeds in state 3, which is confirmed experimentally (i.e., the data for initial seeds in state 2 are very similar to those in Figure 3.21(b)), and is consistent with our finding for seed nodes in state 1: for the conditions of this study, if the dissent initially spreads significantly, the population will drive itself to state 3. The reduced dissension level of about 2 in the long run for seeds in state 1 is heavily influenced by the simulation instances where the unrest quickly dies out. However, the time at which civil disobedience rapidly rises varies in time with seed conditions: there is a big difference between seeds in states 1 and 2 (time 40 versus time 15), but little difference for a further increase in seed state to 3. Government intervention signficantly reduces the levels of engagement, but does not extinguish the unrest.

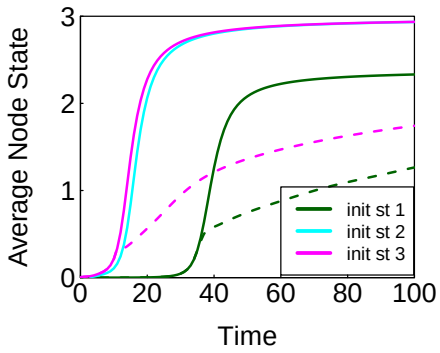


Figure 3.25: Average sentiment in the entire population as a function of time (in days) for the three cases of the initial 5000 random dissenters being mildly unhappy (state 1), moderately agitated (state 2), and extremely unrestful (state 3). These are the solid curves. The data show that intermediate levels of unrest can produce essentially the same results as high levels. Indeed, the data for 5000 nodes initially in state 2 are very similar to those in Figure 3.21(b). Results are averages over 20 simulation instances. The dashed curves are for the intervention where thresholds double when 15% of the population reaches state 1.

Clearly, all of these results are a function of the models and inputs. However, this argues for the importance of a modeling framework like ours: it enables not only changes in input values, but also arbitrary changes in the models themselves.

Practical implications. There are several interesting practical implications of these results. First, dissension can fester for a long time. Hence, if a (social) event generates resentment in a population, the lack of immediate backlash does not mean that none will occur. This is consistent with historical studies [188, 189, 205, 296]. Second, initial sentiment does not have to be extreme to cause widespread unrest, but more extreme initial commitment increases its probability of occurrence and hastens its development. Third, there may not be a smooth transition in population unrest. A critical point may exist that delineates very little dissension from massive unrest, with little room in between. Fourth, Figure 3.21 shows that one may monitor weaker sentiment and use it as a predictor of impending widespread dissension. For our conditions, state 1—relatively minor unreset—reaching 10%-20% of a population indicates that larger unrest is not far behind. The data suggest that monitoring stronger, intermediate unrest (state 2 here) may be more difficult, since the percentage of the population in that state never rises to the levels observed for state 1. Note that, in contrast, a rise in average sentiment is not as good a predictor, because by the time it begins to rise, it rises sharply (Figure 3.25). A rise in lesser unrest being a portent for surges of greater levels of unrest have been observed in many historic instances [188, 189]. Fifth, countries such as China do monitor sentiment, at least through social media [170]. These results also support their approach of tight control, for once sentiment takes hold, it can spread rapidly. Also, by driving down the strength of initial sentiment, overall population sentiment my be

better controlled. Sixth, models such as the one presented, calibrated for actual behavior, can provide insights in the dynamics of populations.

3.6.2 Network Science Study: Contact Vs. Twitter Networks Study

The question that motivated this case study is the following: which type of network structure more readily supports complex contagion propagation—face-to-face contact networks or Twitter networks? We study two networks of each type, with up to 100 million nodes, and use a 2-state ($\{0, 1\}$) dynamics model where the only permissible node state transition is $0 \rightarrow 1$. We use deterministic simulations because this choice gives the final number of nodes to reach state 1 more efficiently than running stochastic simulations. We use relative thresholds [67] θ_i/d_i for nodes v_i, where θ_i is the absolute threshold and d_i is the degree of v_i. Each simulation uses the same relative threshold for all nodes. All simulations use 10,000 seed nodes (nodes initially in state 1), except for the EUS network, where a proportionally larger number of seeds is used (58261).

Figure 3.26 shows how the final fraction of nodes in state 1 varies as the relative threshold varies for the networks. There are several interesting observations: (i) when the threshold $\theta > 0.1 d_{ave}$ (where d_{ave} is the average degree), the contact networks exhibit little spreading; (ii) in constrast, the social media networks can spread contagions significantly over a wide range of relative thresholds (even final spread fractions of 0.1 can be significant; see [46]); (iii) for very modest complex contagions ($\theta = 2, 3$), contact networks spread better, although all networks can propagate these contagions quite well; and (iv) there is a sharp transition between widespread and negligible diffusion in contact networks that is not observed in the social networks.

Since Twitter-type data spreads faster and has greater reach than face-to-face contact, one would expect Twitter to be a good mechanism for information diffusion. Our results also demonstrate that Twitter has a large number of communication channels that are better able to support the spread of complex contagion.

We close this section with two additional points. First, we know of no way to produce these types of results for arbitrary initial conditions other than through at-scale simulation experiments, thus demonstrating the utility of ENSIM. Second, to our knowledge, this is the first time at-scale complex contagion dynamics in contact and social media networks have been generated and compared.

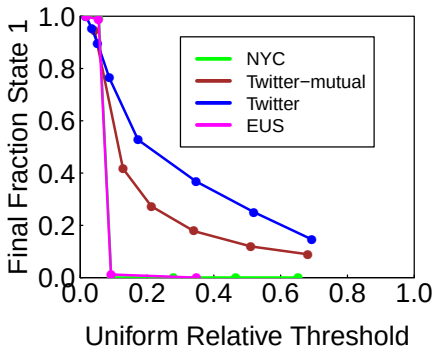


Figure 3.26: Final fraction of nodes in state 1 for progressive complex contagions as a function of relative threshold θ/d_{ave} for all nodes. The plots for EUS and NYC essentially overlap each other.

3.7 Usability

Figure 3.27 provides TTATs for the development of several models (**A** through **J**). These times were conservatively rounded up to whole 8-hour days (as these were not formal experiments). The test person is the person who devised and constructed the code and hence has intimate familiarity with ENSIM. However, a new post-doctoral researcher, with no knowledge of ENSIM, was able to implement and execute an IM given in [40] in two days, which includes the researcher's startup time.

The data in Figure 3.27 clearly show how TTAT varies with model complexity; model complexity increases from IM **A**. Nonetheless, several models from the literature can be implemented, integrated, and validated within a day. The mass movement model **J**, with a TTAT $>$ 3 weeks, is nontrivial. It is based on observations of mass movements (see e.g., [209]). It incorporates multiple node types, local and global interactions, time-varying graphs, state-dependent action decision sets for nodes, and multiple interaction and state transition types.

Our modeling framework has been used in several published works [6, 32, 180–183, 186] and has been used to perform several parametric studies involving 500000 to over a 1 million diffusion instances that involve applications in the social sciences, epidemiology, and computer networks.

3.8 Summary

We described the technical challenges in building a general purpose modeling environment. ENSIM derives its expressiveness, which is our first priority, from (extended) GDS. The second top priority is human productivity and a number of studies that we have done, albeit

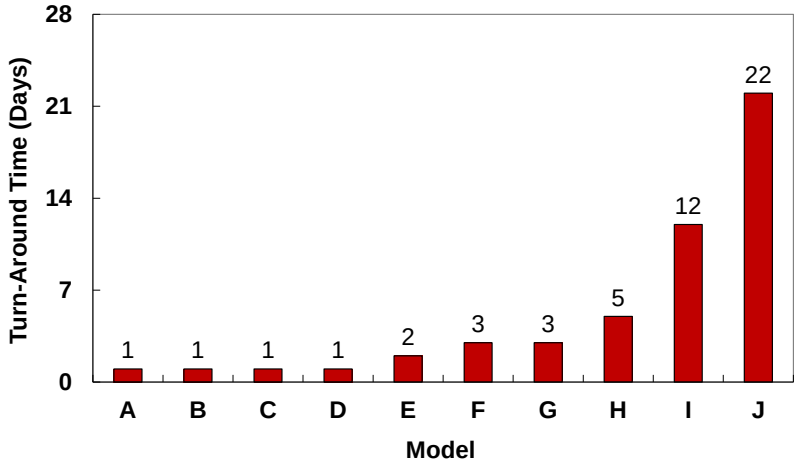


Figure 3.27: Times to design, build, and verify selected interaction models, where letters refer to models below. Times are rounded up to full days, to be conservative. These data illustrate that many models of Table 1.4 (e.g., threshold models) can be implemented within a day or two, but that complicated models can take longer. Models are (A) various simple and complex contagion models (e.g., [67,132,281]); (B) back-and-forth models (e.g., [183]); (C) SEIR models (e.g., [36]); (D) the model of the first case study; (E) Collective learning [212]; (F) multi-contagion model; (G) Trickle algorithm [201]; (H) generalized state-dependent contagion model with arbitrary FSMs; (I) multimechanism threshold models [179]; and (J) mass movement model.

informal, indicate that we achieve this goal through short duration TTAT and ETEST. The next priority is high performance computation, and our hybrid implementation is designed to meet that goal. Although more work remains, we have demonstrated scalable performance to at least 100 million nodes and 2.8 billion edges.

Chapter 4

Predicting Spread Size

4.1 Introduction

4.1.1 Background and Motivation

In reasoning about contagion dynamics, it is useful to quantify the maximum possible spread size in a population; i.e., the number of nodes that a contagion can reach. We demonstrate this utility in Chapters 5 and 7, where we investigate blocking complex contagion propagation. In those chapters, we reason about the effectiveness of a blocking scheme in terms of how many nodes still get affected, relative to the total number of nodes that could possibly be affected without the intervention.

4.1.2 Summary of Contributions

1. Efficiently computable upper bound on the maximum spread size for a progressive contagion process. We present a method for computing the maximum spread size for a contagion (i.e., a *dynamics* quantity) using a *static* graph property. We prove that this is an upper bound. The approach may also be used with the back-and-forth model of Chapter 7.

2. Experimental evaluation of its effectiveness. We evaluate the method on many realistic and synthetic networks with a wide range of graph structures, and we investigate a range of threshold values. We demonstrate that the method provides a tight or reasonably tight bound for many graphs exhibiting characteristics of social networks. We also demonstrate that for some Erdos-Renyi random graphs, the bound is poor.

4.2 Theoretical Bounds on Spread Size

Consider a social network in which every node is in one of the states in $\{0, 1\}$, and the only state transition is $0 \to 1$. Recall that the maximum spread size in a network is the total number of nodes whose state is 1 at the end of the diffusion process. We assume that the given GDS is a θ-threshold system for some $\theta \geq 1$; that is, each local transition function is the θ-threshold function. We start with a graph theoretic definition, extending [282].

Definition 15. *Let $G(V, E)$ be an undirected graph and let $\theta \geq 0$ be an integer. A θ-**core** of G is an induced subgraph $G'(V', E')$ of G such that each node in G' has a degree of at least θ. A θ-core $G'(V', E')$ of G is **maximal** if there is no* strict *superset $V'' \supset V'$ of nodes such that the subgraph of G induced on V'' is also a θ-core.*

Example: Consider the graph $G(V, E)$ shown in Figure 4.1. The induced subgraph formed on the node set $\{v_4, v_5, v_6\}$ is a 2-core of G. Likewise, the induced subgraph formed on the node set $\{v_2, v_3, v_5\}$ is also 2-core of G. The induced subgraph formed on the node set $\{v_2, v_3, v_4, v_5, v_6\}$ is the maximal 2-core of G.

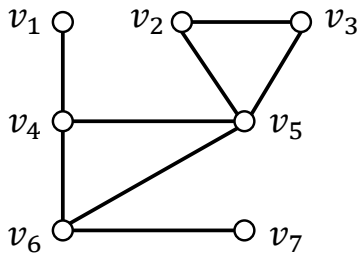


Figure 4.1: An example to illustrate the definitions of θ-core and maximal θ-core.

For any graph G and integer $\theta \geq 0$, it is easy to see that the maximal θ-core is *unique*. In fact, the maximal θ-core of G can be constructed efficiently by repeatedly removing nodes whose degree is less than θ. The removal process ends when all remaining nodes have degree $\geq \theta$ [282].

Even if G is connected, the maximal θ-core may not be connected. For a graph G and integer $\theta \geq 0$, we use $L_G(\theta)$ to denote the number of nodes in a largest connected component of the maximal θ-core of G. As will be seen from the results of this section, the parameter $L_G(\theta)$ can be used to bound the maximum spread size in θ-threshold systems. We begin with a simple lemma.

Lemma 16. *Let $G(V, E)$ be an undirected graph. For any $\theta \geq 0$, a node $v \in V$ is in the maximal θ-core of G if and only if node v has at least θ neighbors that are also in the maximal θ-core.*

Proof: Let $G'(V', E')$ denote the maximal θ-core of G. If v is in V', then by definition, its degree in G' is $\geq \theta$; that is, v has at least θ neighbors in G'. For the other direction, assume that θ of the neighbors of v are in G'. Consider the node removal process that produced G'. This process did not remove any of the θ neighbors of v which are in G'. Thus, throughout the removal process, the degree of v was at least θ. In other words, v remains in G'. □

We present a result that bounds the maximum spread size in a θ-threshold system in terms of the size of the largest connected component of its maximal θ-core. In stating this result, we assume that the seed set is of size θ. If the seed set has less than θ nodes, obviously, there will be no further diffusion in a θ-threshold system.

Theorem 17. *Let G be the underlying graph of a θ-threshold SyDS, where $\theta \geq 1$. Let $G'(V', E')$ denote the maximal θ-core of G. For any seed set $I \subseteq V'$ with $|I| = \theta$, the maximum spread size is bounded by $L_G(\theta)$, the size of a largest connected component of G'. Moreover, for each $\theta \geq 1$, this bound is achievable.*

Proof: We first prove the upper bound. For $\theta = 1$, the set I contains only one node, say v, and diffusion can lead to new affected nodes only in the connected component that contains v. Thus, the upper bound is trivial for $\theta = 1$.

So, assume that $\theta \geq 2$. First, consider the case where the nodes in I are distributed over two or more connected components of the maximal θ-core G'. Thus, no node of G' is adjacent to all the nodes in I. In this case, we can prove by contradiction no further diffusion can occur. To see this, consider a node $v \in V$ whose state changes from 0 to 1 in the first time step. Since the threshold is θ, node v must be adjacent to all the θ nodes of I. Then, by Lemma 16, v will also be in G'. However, this contradicts the previous conclusion that no node in G' is adjacent to all the nodes of I.

Thus, we may assume that all nodes of I are in the same connected component of G'. In such a case, all the newly affected nodes will also be part of that component. Therefore, the spread size is bounded by $L_G(\theta)$ when $\theta \geq 2$.

We now argue that this bound is achievable through a simple example. For each $\theta \geq 1$, let the graph $G(V, E)$ consist of a clique on $\theta + 1$ nodes. Since the degree of each node of G is θ, $L_G(\theta) = |V|$. Further, choosing any θ nodes of the $\theta + 1$ clique as the seed nodes ensures that all the nodes in V are affected; that is, the maximum spread size is $\theta + 1 = L_G(\theta)$. □

The bound given by Theorem 17 enables us to determine whether maximum spread sizes from simulation are indeed the largest possible spread sizes for a given network and threshold. Also, our approach of computing the size of a largest connected component of a θ-core is closely aligned with determining the largest possible spread size for $\theta = 1$ diffusion by computing the largest connected component of a graph [7, 146].

There are classes of graphs for which the bound given by Theorem 17 is pessimistic. For example, consider an $r \times r$-node lattice graph $G(V, E)$, with $r \gg 2$, where each node is connected to its four neighbors (if they exist) along the four geographic directions. (Thus, nodes along the boundary have degrees less than 4.) Since all nodes have degree ≥ 2, $L_G(2) = |V|$. Yet, it can be seen that for any seed set with $|I| = \theta = 2$, the maximum possible spread size is $4 \ll L_G(2)$. Thus, it is of interest to experimentally evaluate the effectiveness of the bound for different classes of graphs and values of θ. This evaluation is carried out in the next section.

4.3 Experimental Evaluation of the Spread Size Upper Bound

In this section, we use two sets of experiments to carry out an empirical evaluation of the upper bound of Theorem 17 for various synthetic and real social networks. The first set chooses seed nodes from appropriate core subgraphs as used in the proof of Theorem 17. The second set considers a different seeding method and evaluates whether the theoretical bound remains robust. We now discuss the two sets of experiments and results.

Empirical evaluation using core construction. Our experimental procedure is as follows. We consider three real social networks: `slashdot`, `epinions` and `wikipedia`. See Table 4.1. We also evaluate three synthetic networks: two scale-free (SF) networks (`sf-01` and `sf-02`) constructed using two well-known methods (preferential attachment [25] and configuration model [238], respectively), and an Erdős-Renyi (ER) random graph. Data for the three synthetic graphs are given in Table 4.2. The purpose of using synthetic networks in this study is to evaluate whether particular construction methods yield graph structures that affect the upper bound on spread size. The average clustering coefficients for the two SF graphs differ by an order of magnitude and these values are also different from those in Table 4.1. In Chapter 5, it is experimentally demonstrated that clustering has a large impact on diffusion dynamics, motivating its investigation here. Although not shown, the degree distributions for the two SF networks are different, even though the average degree is about the same.

Table 4.1: Selected characteristics of three realistic social networks [197, 199, 274].

Network	Number of Nodes	Number of Edges	Average Degree	Average Clustering Coefficient	Number of Connected Components	Size of Largest Component
`epinions`	75879	405740	10.7	0.138	2	75877
`slashdot`	77360	469180	12.1	0.0555	1	77360
`wikipedia`	7115	100762	28.3	0.141	1	7115

We consider three different values of θ, namely 2, 3 and 5. We construct the corresponding maximal cores and use Theorem 17 to determine the upper bound on the spread sizes for various graphs. We compare these values against the maximum measured spread sizes from simulations.

In these simulations, seed nodes for the three real networks were chosen from their maximal 20-core subgraphs (which are themselves subgraphs of the corresponding 2-cores, 3-cores and 5-cores). Since the average degree of the nodes in the three synthetic graphs is around 10, simulations for the three synthetic networks were done using seed sets from the maximal 10-cores. In all cases, seed nodes induce a subgraph that is connected.

The values of numbers n_s of seed nodes explored are 2, 3, 5, 10, and 20. We provide results for $n_s = 20$ seed nodes, but since all seed sets are highly connected, the results are independent of n_s. For each value of n_s, we select 100 seed sets of that size, and run one diffusion instance for each seed set. All nodes are initially in state 0 except the seed nodes, which are initially in state 1. The only admissible state transition is $0 \to 1$. A node transitions when a threshold number of its neighbors are in state 1. We use deterministic threshold diffusion since we are interested in the final spread size, which is merely the number of nodes in state 1 at the end of a diffusion instance, which occurs when no node in state 0 can transition to 1.

Table 4.2: Selected characteristics of three synthetic networks.

Network	**Number of Nodes**	**Number of Edges**	**Average Degree**	**Average Clustering Coefficient**	**Number of Connected Components**	**Size of Largest Component**
`er`	99998	500172	10.0	0.000106	1	99998
`sf-01`	100000	499959	10.0	0.000994	1	100000
`sf-02`	100007	568277	11.4	0.0127	1032	97921

Figure 4.2 provides results for the six networks and three threshold values. There are three data points in the plot for each network, corresponding to $\theta = 2$, 3, and 5, respectively. For all networks *except* the ER network, the agreement between the experimentally measured spread size (ordinate) and the theoretical bound (abscissa) is excellent, illustrating that in real and synthetic networks, the maximum spread size can be achieved. As just one representative example, consider `wikipedia`. When $n_s = \theta = 2$, 97% of simulated diffusion instances achieve the upper bound spread size. When $n_s = \theta = 3$ and $n_s = \theta = 5$, the corresponding percentages are 77% and 35%, respectively.

The Erdős-Renyi (ER) random graph data, however, show that the spread size is much smaller than the theoretical maximum. We can explain this behavior as follows. For the 99998-node ER random graph, with an average degree of 10, the probability of having an edge between any pair of nodes is $p = d_{ave}/(n-1) = 10/(99998-1) \approx 10^{-4}$. It follows that the probability of having two seed nodes each forming an edge with a third node v (so that v will be affected for $\theta = 2$ diffusion) is roughly 10^{-8}. A simple repetition of this argument

shows that ER random graphs with tens of thousands of nodes (or more) and small to moderate degrees will propagate complex contagions with very low probability. Accordingly, the measured spread sizes are nearly zero in Figure 4.2, while the theoretical bound is large.

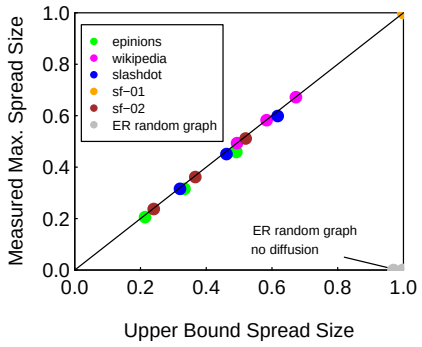


Figure 4.2: Comparison of maximum measured spread sizes from simulations (ordinate values) and upper bound spread size from Theorem 17 (abscissa values). The 45^o line of perfect agreement is also provided. The data are for the networks of Tables 4.1 and 4.2, with thresholds $\theta = 2, 3$, and 5. Note the data for `sf-01` in the upper right hand corner at $(x, y) = (1, 1)$.

The ER graph results in the last figure depend on the edge probability used to generate the graph. Moreover, for given n, there are $2^{\binom{n}{2}}$ ER graph realizations (the number of edges that can be formed from n nodes, since each edge contains two nodes, is $\binom{n}{2}$, and each edge can either exist or not exist). Which instances are more likely to occur are predicated on p, for a given n. In the preceding figure, we evaluated only one graph instance, but the point was to demonstrate that the upper bound can be arbitrarily poor.

If the edge probability for the ER graph was sufficiently large, then the experimentally-determined spread size would have agreed with the upper bound; i.e., the edge probability can be chosen so that a full cascade will result owing to the increased number of edges among the nodes. Figure 4.3 shows the full range of spread sizes as a function of edge probability for 10000-node ER graphs and 50000-node ER graphs, represented as solid and dashed lines, respectively. For example, the solid blue curve shows that for a particular 10000-node ER graph and $\theta = 3$, a probability below $p = 2 \times 10^{-3}$ will produce little contagion propagation while greater probabilities will produce a cascade. This p is the transition probability. Note that every graph used to generate the respective curves of this figure possesses a 2-, 3-, or 5-core, as appropriate, of size n, so that the upper bound spread size is n, or fractionally, 1.0. Hence, below the transition probability in the figure, for a given θ, the measured spread size and the upper bound spread size will differ greatly. However, above the transition probability, the upper bound will be tight. These data were also generated on a particular ER graph instance, for a given (p, n) pair. Nonetheless, the point is made with these data: if

the probability chosen for the ER graph in Table 4.2 was sufficiently large, the data points in the lower right corner of Figure 4.2 would have plotted in the upper right corner, indicating perfect agreement with the upper bound. Hence, a blanket statement regarding the tightness of the upper bound spread size in relation to ER graphs cannot be made. It depends on the graph instance. The likelihood is that for a given ER graph instance, the bound is going to be very good or very poor, because the transition is so steep in Figure 4.3.

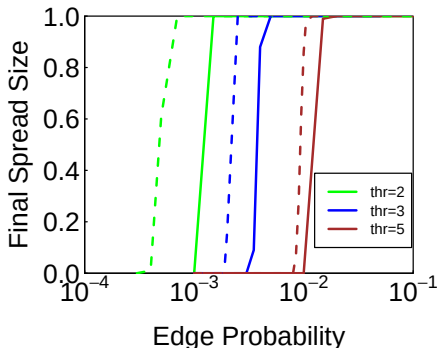


Figure 4.3: Experimentally-determined spread sizes (normalized by the number of graph nodes), for 10000-node (solid lines) and 50000-node (dashed lines) ER graphs. For each number of nodes, the edge probability was systematically varied and deterministic progressive θ-threshold diffusion was run on the graph for $\theta = 2$, 3, and 5. For a given number of nodes, the probability that causes widespread diffusion (i.e., a cascade) decreases as threshold decreases. Also, for a fixed threshold, as numbers of nodes increases the edge probability that causes a cascade decreases because the average degree d_{ave} in a graph is $d_{ave} = p(n-1)$, where p is the edge probability. Hence, more edges are formed as n increases. Each data point is the average of 100 diffusion instances.

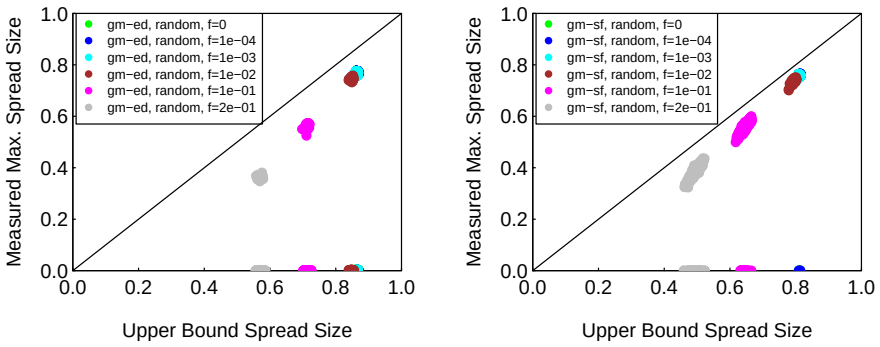
The extremes of tight and pessimistic upper bounds in Figure 4.2 motivate our next experiments.

Empirical evaluation using a random seeding and a node removal scheme. To understand whether the theoretical upper bound on spread size is robust when the assumptions used in deriving the bound are relaxed, we now consider a second method of seeding. We utilize the *random* seeding method of [65,67] where a node v is chosen uniformly at random and the seed set consists of v and all of its (distance-1) neighbors. Thus, the seed node sets are not from the maximal 2-core, as used in Theorem 17. Since these experiments involve a large number of graphs (as described momentarily), only one threshold value, namely $\theta = 2$, was used.

For this study, we also explored graph structure effects by considering two types of synthetic graphs, namely growth model scale-free networks and growth model exponential-decay networks, whose degree distributions are often found in real social networks [65]. These graphs

are generated using a combined approach from [25] and [280], producing graphs with $d_{ave} = 4$ and $C_{ave} = 0.24$; using the cited construction procedures enabled us to control both of these properties.

Thirty graph instances of each type are generated, each with 10000 nodes, and these constitute the original graphs. For each of the 30 original instances, six values (namely, 0, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1} and 2×10^{-1}) of the fraction f of removed nodes are considered. These fractions of nodes are removed at random. The removed nodes alter the graph topology and hence provide another means of evaluating graph structure effects on the theoretical upper bound spread size. For each value of f, 20 sets of nodes are removed from each original graph, in turn, resulting in 3600 ($= 30 \cdot 6 \cdot 20$) graph instances. For each graph, the size of the largest connected component of the 2-core is computed. Then, $\theta = 2$ diffusion is simulated as described above, with 20 seed sets for each graph, to experimentally estimate the largest spread size.



(a) (b)

Figure 4.4: Comparison of maximum measured spread sizes from simulations (ordinate values) and upper bound spread size from Theorem 17 (abscissa values). The 45^o line of perfect agreement is also provided. (a) Data for 3600 graph instances of `gm-ed` networks with different fractions f of nodes removed at random from the 30 original graph instances. (b) Data for 3600 graph instances of `gm-sf` networks with different fractions f of nodes removed at random from the 30 original graph instances.

Figure 4.4(a) contains data for the 3600 growth model exponential-decay (`gm-ed`) networks, and Figure 4.4(b) contains comparable data for 3600 growth model scale-free (`gm-sf`) networks. With respect to the data in Figure 4.4(a), over 20% of the 3600 graph instances produce spread sizes near the line of agreement and 80% of the graphs produce virtually no diffusion. For the graphs without any removed nodes, 27% of diffusion instances achieve close to the maximum spread size. However, even a few graphs with 20% of nodes removed generate spread sizes close to the theoretical limit. With respect to the data in Figure 4.4(b), over 98% of the 3600 graph instances produce spread sizes near the line of agreement. For

the conditions of this study, the maximum spread size is more readily achieved in `gm-sf` networks. More generally, these results suggest that the bound given by Theorem 17 provides a reasonable indication of the maximum permissible spread size even when the conditions of the theorem are relaxed.

Finally, our results also suggest that defining a cascade based on a fraction (say 95%) of $L_G(\theta)$ is a rational way to quantify wide-spread diffusion. This approach reflects the dynamics, and it is more meaningful than specifying a cascade based on a fraction of total nodes affected in a network, since complex contagions often cannot spread throughout a network.

4.4 Summary

We presented a simple method to determine an upper bound on the contagion spread size for θ-threshold diffusion in progressive two-state systems. The method uses a static network parameter, namely the size of the largest connected component of a θ-core, to determine a dynamics quantity. We proved that the result provides an upper bound, and we evaluated this bound on several networks, demonstrating that the bound is useful for graphs with structures representative of social networks.

Chapter 5

Blocking Contagions

5.1 Introduction

5.1.1 Background and Motivation

Motivation for studying contagion blocking in populations is provided at the beginning of Section 1.7. Here, we focus on progressive complex contagions, where a node can transition from state 0 to 1, but not from 1 to 0. The goal of our work is to devise methods to inhibit this $0 \rightarrow 1$ transition, thus preventing as many nodes as possible from reaching state 1. We investigate node-based blocking schemes. That is, we develop approaches to identify intransigent or **critical nodes**. Critical nodes are nodes assigned state 0 initially and remain in state 0 throughout the threshold-governed contagion propagation process. Thus, these nodes do not transition to state 1 and inhibit their neighbors from doing so. In addition, our work highlights differences between simple and complex contagions, a topic originally introduced in [67] when they demonstrated that weak links have a different effect on contagions of these types. Our approach incorporates complexity theory, algorithm development, and simulation. We are primarily intereted in complex contagions, but our blocking algorithms can be applied to simple contagions as well.

5.1.2 Summary of Contributions

Section 5.2 presents formulations of the problems studied in this chapter. These problems include SCS-MNA, SCS-MUN, and SCS-SASN; these all describe problems in specifying critical nodes to halt or reduce the number of nodes to which a contagion may propagate. We use forward references to some of these problems in order to provide a concise summary of contributions.

1. Formal hardness results for contagion blocking problems. We show that for any threshold $\theta \geq 2$ and any $\rho \geq 1$, it is **NP**-hard to obtain a ρ-approximation for either the SCS-MNA problem or the SCS-MUN problem for θ-threshold systems. (The result holds even when ρ is a function of the form n^{δ}, where $\delta < 1$ is a constant and n is the number of nodes in the underlying network.)

2. Problem for which simple and complex contagions have different complexities. We show that the problem of saving all salvageable nodes (SCS-SASN) can be solved in linear time for 1-threshold systems and that the required critical set is unique. In contrast, we show that the problem is **NP**-hard for θ-threshold systems for any $\theta \geq 2$. We present an approximation algorithm for this problem with a performance guarantee of $\rho < 1 + \ln(s)$, where s is the number of salvageable nodes in the system. We also show that the performance guarantee cannot be improved significantly, unless $\mathbf{P} = \mathbf{NP}$.

3. Effective heuristics to block contagions. We develop two intuitively appealing heuristics, designated CBH and PBH, for the SCS-MNA problem, and carry out an empirical study of their performance on three social networks, namely `epinions`, `wikipedia` and `slashdot`. We compare our schemes against five known methods for determining critical nodes (representing a range of blocking methods): random assignment, high-degree nodes, nodes of high betweenness centrality [115], nodes of high eigenvector centrality [44], which can also be computed for undirected graphs using the Hyperlink-Induced Topic Search (HITS) algorithm [173], and maximum eigenvalue drop (also called NetShield) [308]. We show that our methods are far more effective in blocking complex contagions.

4. Experimental evaluation of our heuristics. Beyond the comparisons above, we critically evaluate our methods over a range of thresholds, budgets on the number of critical nodes, numbers of seed nodes that initiate the contagion diffusion process, and networks. For example, the number β of critical nodes required to halt all contagion propagation may be more than a factor of 100 greater than the number of seed nodes. We identify limitations of our methods. We also investigate the dynamics that are produced. The two blocking methods are compared based on their effectiveness in blocking contagions and in their runtimes. We demonstrate that CBH is more effective than PBH, but that PBH runs faster for larger networks.

We use uniform or homogeneous thresholds (i.e., the same threshold) for all nodes in a simulation, to more readily identify performance dependencies on threshold. However, our heuristics can be used with heterogeneous thresholds without modification. They can also be extended for use under more general transition criteria for nodes (as in generalized contagion models [101]) and with probabilistic diffusion (where a node transitions from 0 to 1 with probability p when its threshold is met). Finally, our methods can also be extended for use in time-varying networks where the edges of the network and transition criteria change in a repeatable pattern (e.g., to reflect daytime and night-time interactions as in [268]).

5.2 Dynamical System Model and Problem Formulation

5.2.1 System Model and Associated Definitions

GDSs were introduced in Section 1.5. We focus here on one update scheme, namely synchronous update, where all nodes update their states simultaneously at each time. We refer to this type of GDS as a **Synchronous Dynamical System** (SyDS) $\mathcal{S}$. Here, nodes occupy a state from the set $\mathbb{B} = \{0, 1\}$ at each time. We use the term **affected** (**unaffected**) to refer to nodes in state 1 (0). Thus, in the case of information flow, for example, an affected node has received the information and will pass it on. It is assumed that once a node reaches the state 1, it cannot return to state 0, so the contagion dynamics are progressive.

We can now formally describe the local transition functions used in this study and the update methodology. The inputs to function f_i are the state of v_i and those of the neighbors of v_i in G; function f_i maps each combination of inputs to a value in $\mathbb{B}$. For the propagation of contagions in social networks, it is natural to model each function f_i $(1 \leq i \leq n)$ as a θ_i-**threshold function** [66, 67, 71, 105, 110, 166, 172] for an appropriate nonnegative integer θ_i. Such a threshold function (taking into account the ratcheted nature of the dynamical system) is defined as follows.

(a) If the state of v_i is 1, then the value of f_i is 1, regardless of the values of the other inputs to f_i.

(b) If the state of v_i is 0, then the value of f_i is 1 if at least θ_i of the inputs are 1; otherwise, the value of f_i is 0.

A single SyDS transition from one configuration to another can be expressed by the following pseudocode.

for each node v_i **do in parallel**

(i) Compute the value of f_i. Let s_i' denote this value.

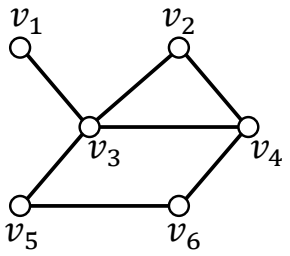
(ii) Update the state of v_i to s_i'.

end for

Thus, in an SyDS, nodes update their state *synchronously*. We provide a basic example of the dynamics, and later build on it by incorporating critical nodes.

Example: Consider the graph shown in Figure 5.1. Suppose the local interaction function at each node is the 2-threshold function. Initially, v_1 and v_2 are in state 1 and all other nodes are in state 0. During the first time step, the state of node v_3 changes to 1 since two of its

neighbors (namely v_1 and v_2) are in state 1; the states of other nodes remain the same. In the second time step, the state of node v_4 changes to 1 since two of its neighbors (namely v_2 and v_3) are in state 1; again the states of the other nodes remain the same. The resulting configuration $(1, 1, 1, 1, 0, 0)$ is a fixed point for this system.



Initial Configuration: (1,1,0,0,0,0)

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

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

Note: Each configuration has the form $(s_1, s_2, s_3, s_4, s_5, s_6)$, where s_i is the state of node v_i, $(1 \leq i \leq 6)$. The configuration at time 2 is a fixed point.

Figure 5.1: An example of a synchronous dynamical system.

The SyDS in the above example reached a fixed point. This is not a coincidence. The following simple result shows that every ratcheted dynamical system over $\mathbb{B}$ reaches a fixed point.

Proposition 18. *Every deterministic progressive discrete dynamical system over $\mathbb{B}$ reaches a fixed point in at most n transitions, where n is the number of nodes in the underlying graph.*

Proof: Consider any progressive dynamical system $\mathcal{S}$ over $\mathbb{B}$. In any transition of $\mathcal{S}$ from one configuration to another, nodes can only change from 0 to 1 (but not from 1 to 0). Thus, after at most n transitions where nodes change from 0 to 1, there can be no more state changes, i.e., $\mathcal{S}$ reaches a fixed point. □

In the context of opinion propagation, reaching a fixed point means that everyone has formed an unalterable opinion, and hence will not change their mind.

5.2.2 Problem Formulation

For simplicity, statements of problems and results in this chapter use terminology from the context of information propagation in social networks, such as that for social unrest. It is straightforward to interpret the results for other contagions.

Suppose we have a social network in which some nodes are initially affected. In the absence of any action to contain the unrest, it may spread to a large part of the population. Decision-makers must decide on suitable actions (interventions) to inhibit information spread, such as quarantining a subset of people. Usually, there are resource constraints or societal pressures to keep the number of isolated people to a minimum (e.g., quarantining too many people may fuel unrest or it may be difficult to apprehend particular individuals). Thus, the problem formulation must take into account both information containment and appropriate resource constraints.

We assume that only people who are as yet unaffected can be quarantined. Under the dynamical system model, quarantining a person is represented by removing the corresponding node (and all the edges incident on that node) from the graph. Equivalently, removing a node v corresponds to changing the local transition function at v so that v's state remains 0 for all combinations of input values. The goal of isolation is to minimize the number of new affected nodes that occur over time until the system reaches a fixed point (when no additional nodes can be affected). We use the term **critical set** to refer to the set of nodes removed from the graph to reduce the number of newly affected nodes. Resource constraints can be modeled as a budget constraint on the size of the critical set. We can now provide a precise statement of the problem of finding critical sets. (This problem was first formulated in [110] for the case where each node computes a 1-threshold function.)

Smallest Critical Set to Minimize the Number of New Affected Nodes (SCS-MNA)

Given: A social network represented by the SyDS $\mathcal{S} = (G(V, E), \mathbf{F}, W)$ over $\mathbb{B}$, with each function $f \in \mathbf{F}$ being a threshold function; the set I of nodes which are initially in state 1 (the elements of I are called **seed nodes**); an upper bound β on the size of the critical set.

Requirement: A critical set $C \subseteq V - I$ such that $|C| \leq \beta$ and among all subsets of $V - I$ of size at most β, the removal of C from G leads to the smallest number of new affected nodes.

An alternative formulation, where the objective is to maximize the number of nodes who are not affected, can also be considered. We use the name "Smallest Critical Set to Maximize Unaffected Nodes" for this problem and abbreviate it as SCS-MUN. To maintain the complementary relationship between the minimization (SCS-MNA) and maximization (SCS-MUN) versions, we assume that critical nodes are not included in the set of unaffected nodes in the formulation of SCS-MUN. Thus, any optimal solution for SCS-MUN is also an optimal solution for SCS-MNA. Our results in Section 5.3 provide an indication of the difficulties in obtaining provably good approximation algorithms for either version of the problem. So, our focus is on obtaining heuristics that work well in practice.

We also consider the problem of finding critical sets in a related context. Let $\mathcal{S} = (G(V, E), \mathbf{F}, W)$ be a SyDS and let $I \subseteq V$ denote the set of seed nodes. We say that a node $v \in V - I$ is **salvageable** if there is a critical set $C \subseteq V - (I \cup \{v\})$ whose removal ensures that v remains

in state 0 when the modified SyDS (i.e., the SyDS obtained by removing C) reaches a fixed point. Otherwise, v is called an **unsalvageable** node. Thus, in any SyDS, only salvageable nodes can possibly be saved from becoming affected.

Example: Consider the 2-threshold SyDS shown in Figure 5.1. Node v_4 in that figure is salvageable since removal of v_3 ensures that v_4 won't be affected. Nodes v_5 and v_6 are salvageable since they are not affected even when no nodes are removed from the system. However, node v_3 is not salvageable since it has two neighbors (v_1 and v_2) who are initially affected.

We now formulate a problem whose goal is to find a smallest critical set that saves all salvageable nodes.

Smallest Critical Set to Save All Salvageable Nodes (SCS-SASN):

Given: A social network represented by the SyDS $\mathcal{S} = (G(V, E), \mathbf{F}, W)$ over $\mathbb{B}$, with each function $f \in \mathbf{F}$ being a threshold function; the set I of seed nodes which are initially in state 1.

Requirement: A critical set $C \subseteq V - I$ of minimum cardinality whose removal ensures that all salvageable nodes are saved from being affected.

As will be shown in Section 5.3, the complexity of SCS-SASN problem for simple contagions is significantly different from that for complex contagions.

5.2.3 Types of Thresholds

In the above discussion, the threshold (also called the **absolute threshold**) of each node was specified as a non-negative integer. A **homogeneous threshold** SyDS is one where all the nodes of a SyDS have the same threshold θ, for some integer $\theta \geq 0$. A **heterogeneous threshold** SyDS is one where nodes may have different thresholds. Researchers [67] have also considered **relative thresholds**, where the threshold value of a node is a non-negative fraction of the number of neighbors of the node. (Here, each node is considered a neighbor of itself.) Similar to absolute thresholds, one can also consider homogeneous and heterogeneous relative thresholds. We refer to these variants as the "θ-threshold variants." Most of our theoretical results (Section 5.3) are presented in terms of absolute thresholds. Extensions of these results to the other θ-threshold variants are straightforward, as outlined in Appendix B.

5.2.4 Additional Terminology

Here, we present some terminology used in the later sections of this chapter. The term "θ-threshold system" is used to denote a SyDS with a homogeneous absolute threshold $\theta \geq 0$. (Thus, the value of θ is the same for all nodes of the system.)

We also need some terminology with respect to approximation algorithms for optimization problems [120]. For any $\rho \geq 1$, a ρ-**approximation** for an optimization problem is an efficient algorithm that produces a solution which is within a factor of ρ of the optimal value for all instances of the problem. Such an approximation algorithm is also said to provide a **performance guarantee** of ρ. Clearly, the smaller the value of ρ, the better is the performance of the approximation algorithm.

The following terms are used in describing simulation results and the behavior of the heuristics that produce critical sets. The **spread size** is the number (or fraction) of nodes in the affected state; the **final spread size** is the value at the end of a diffusion instance. A **cascade** occurs when diffusion starts from a set of seed nodes and the final fractional spread size is 0.95 or more of nodes that can possibly be affected. **Halt** means that the chosen set of critical nodes will stop the diffusion process, thus preventing a cascade. A **delay** means that the chosen set of critical nodes will increase the time at which the peak number of newly affected nodes occurs, but will not necessarily halt diffusion.

5.3 Theoretical Results for the Critical Set Problem

5.3.1 Overview and a Preliminary Lemma

In this section, we first establish complexity results for finding critical sets. We also present results that show a significant difference between 1-threshold systems and θ-threshold systems where $\theta \geq 2$. Most of the results in this section are for homogeneous thresholds; extensions of the results to heterogeneous and relative thresholds are outlined in Appendix B.

Lemma 19. *Given a SyDS $\mathcal{S} = (G(V, E), \mathbf{F}, W)$, the set $I \subseteq V$ of initially affected (i.e., seed) nodes and a critical set $C \subseteq V - I$, the number of new affected nodes in the system that results when C is removed from V can be computed in $O(|V| + |E|)$ time.*

Proof: Recall that the removal of C is equivalent to changing the local transition function f_v of each node $v \in C$ to the function that remains 0 for all inputs. Since the resulting SyDS $\mathcal{S}_1$ is also a ratcheted SyDS, by Proposition 18, it reaches a fixed point in at most $n = |V|$ steps. The fact that each local transition function is a threshold function can be exploited to find all the nodes that are affected over the time steps in $O(|V| + |E|)$ time.

The idea is to have for each node $v \in V$, a counter c_v that stores the number of neighbors of v that are currently affected. To begin with, for each unaffected node, the counter is initialized to 0. In time step 1, for each node $w \in I$, the counter for each unaffected neighbor x of w is incremented. If the count for x reaches its threshold, then x is added to a list L of nodes which will contain all the nodes that are affected at time step 1. At the next time step, the above procedure is repeated using the nodes in L (instead of the nodes in I). This method can be carried out for each subsequent time step until the system reaches a

fixed point (i.e., until the list of newly affected nodes becomes empty). It can be seen that for each node v of G, this method explores the adjacency list of v just once through all the time steps. So, the total time spent in the computation is $O(\sum_{v \in V} \text{degree}(v)) = O(|E|)$. The initialization of the counters and the final step to output the newly affected nodes take $O(|V|)$ time. Therefore, the total time is $O(|V| + |E|)$. □

5.3.2 Complexity Results

As mentioned earlier, the SCS-MNA problem was shown to be NP-complete in [110] for the case when each node has a 1-threshold function. We now extend that result to show that even obtaining a ρ-approximate solution is **NP**-hard for systems in which each node computes the θ-threshold function for any $\theta \geq 2$.

Theorem 20. *Assuming that the bound β on the size of the critical set cannot be violated, for any $\rho \geq 1$ and any $\theta \geq 2$, there is no polynomial time ρ-approximation algorithm for the SCS-MNA problem for θ-threshold systems, unless $\boldsymbol{P} = \boldsymbol{NP}$.*

Proof: Suppose $\mathcal{A}$ is a ρ-approximation algorithm for the SCS-MNA problem for θ-threshold systems for some $\rho \geq 1$ and $\theta \geq 2$. We will show that $\mathcal{A}$ can be used to efficiently solve the **Minimum Vertex Cover** (MVC) decision problem [120]: Given an undirected graph $G(V, E)$ and an integer k, is there a subset V' of V such that $|V'| \leq k$ and for each edge $\{u, v\} \in E$, at least one of u and v is in V'?

Let $G = (V, E)$ be the given graph for the vertex cover problem, with $n = |V|$ and $m = |E|$. We construct a SyDS $\mathcal{S} = (H(V_H, E_H), \mathbf{F}, W)$ as follows. The vertex set V_H consists of three pairwise disjoint groups of nodes denoted by X, Y and Z. The set $X = \{x_1, x_2, \ldots, x_t\}$ consists of θ nodes all of which are initially 1. The set $Y = \{y_1, y_2, \ldots, y_n\}$ contains a node for each member of V. Let $\alpha = \lceil \rho(n - k) \rceil + k + 1$. The set $Z = \{z_1, z_2, \ldots, z_{\alpha m}\}$ contains a total of αm nodes, with α nodes corresponding to each edge of G. All the nodes in $Y \cup Z$ are initially 0. The edges in E_H are as follows.

(a) Each node in Y is adjacent to each node in X.

(b) Each node in Z is adjacent to the first $t - 2$ nodes (i.e., nodes x_1, …, x_{t-2}) of X.

(c) Let g_j denote the group of α nodes corresponding to edge $e_j \in E$; each node of g_j is adjacent to the two nodes in Y which correspond to the end points of the edge $e_j \in E$, $1 \leq j \leq m$.

The local transition function at each node of $\mathcal{S}$ is the θ-threshold function. The value of β (the upper bound on the critical set size) is set to k. This completes the construction of the SCS-MNA instance. Obviously, the construction can be done in polynomial time.

Suppose G has a vertex cover $V' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ of size k. It can be verified that when the critical set $C = \{y_{i_1}, y_{i_2}, \ldots, y_{i_k}\}$ is removed, only the $n-k$ nodes in $Y - C$ are affected; that is, the number of new affected nodes is $n-k$. Since Algorithm $\mathcal{A}$ provides a performance guarantee of ρ, the critical set output by $\mathcal{A}$ in this case leads to at most $\rho(n-k)$ new affected nodes.

Now suppose that a minimum vertex cover for G has $k+1$ or more nodes. We claim that no matter which subset of k (or fewer) nodes from $Y \cup Z$ is chosen as the critical set, the number of newly affected nodes is at least $\rho(n-k)+1$. To see this, note that any critical set can use at most k nodes of Y. Since any minimum vertex cover for G has $k+1$ or more nodes, no matter which subset of k nodes from V is chosen, at least one edge $e_j = \{v_p, v_q\}$ remains uncovered (i.e., neither v_p nor v_q is in the chosen set). As a consequence, no matter which subset of k nodes from Y is chosen, there is at least one group g_j of α nodes in Z such that for each node $z \in g_j$, the two nodes in Y, say y_p and y_q, that are adjacent to z are not in the critical set. Thus, y_p and y_q will become affected and consequently all nodes in group g_j become affected. Since g_j contains $\alpha = \lceil \rho(n-k) \rceil + k + 1$ nodes, even if C includes k nodes from g_j, at least $\lceil \rho(n-k) \rceil + 1$ nodes of g_j will become affected. Thus, when the minimum vertex cover for G is of size $k+1$ or more, the number of newly affected nodes is *strictly greater than* $\rho(n-k)$.

Now, suppose we execute $\mathcal{A}$ on the resulting SCS-MNA instance and obtain a critical set C. From the above argument, G has a vertex cover of size at most k if and only if the number of new affected nodes that result from the removal of C is at most $\rho(n-k)$. From Lemma 19, the number of new affected nodes after the removal of a critical set can be found in polynomial time. Thus, using $\mathcal{A}$, we have a polynomial time algorithm for the MVC problem, contradicting the assumption that $\mathbf{P} \neq \mathbf{NP}$. □

We note that, in the above proof, the factor ρ need not be a constant; it may be a function of the form n^δ, where $\delta < 1$ is a constant and n is the number of nodes of the graph in the MVC instance.

We now present a result similar to that of Theorem 20 for the maximization version of the problem (SCS-MUN).

Theorem 21. *Assuming that the bound β on the size of the critical set cannot be violated, for any $\rho \geq 1$ and any $\theta \geq 2$, there is no polynomial time ρ-approximation algorithm for the SCS-MUN problem for θ-threshold systems, unless $\boldsymbol{P} = \boldsymbol{NP}$.*

Proof: Assume that $\mathcal{A}$ is a ρ-approximation algorithm for the SCS-MUN problem. We prove the result by a reduction from Minimum Vertex Cover problem, similar to the one used to prove Theorem 20. The modifications are as follows.

(a) In addition to the sets of nodes X, Y and Z, we have another set of nodes $W = \{w_1, w_2, \ldots, w_h\}$, where $h = \lceil (\rho - 1)|Z| \rceil$.

(b) Each node in W is adjacent to the first $\theta - 1$ nodes in X and all nodes in Z.

Now, if G has a vertex cover of size k, then by choosing the corresponding nodes of Y, all the nodes in $Z \cup W$ can be saved from becoming affected. (Recall from Section 5.2.2 that the chosen critical nodes are not included in the set of unaffected nodes.) Thus, in this case, the number of unaffected nodes is $|Z| + \lceil(\rho - 1)|Z|\rceil \geq \rho|Z|$. Since $\mathcal{A}$ is a ρ-approximation algorithm, it must produce a critical set of size k such that the number of nodes which are not affected is at least $|Z|$.

If every vertex cover for G has $k + 1$ or more nodes, then no matter which subset of k (or fewer) nodes is chosen from Y, at least one node of Z will become affected. Consequently, all the nodes in W (which are not in the critical set) will also become affected. Therefore, no matter which critical set of size k is chosen, the number of unaffected nodes is at most $|Z| - 1$.

Thus, using $\mathcal{A}$, we can obtain a polynomial time algorithm for the MVC problem, contradicting the assumption that $\mathbf{P} \neq \mathbf{NP}$. □

5.3.3 Critical Sets for Saving All Salvageable Nodes

Recall from Section 5.2.2 that a node v of a SyDS is salvageable if there is a critical set whose removal ensures that v will not be affected. The problem of finding optimal critical sets to save all salvageable nodes, denoted by SCS-SASN, was also formulated in that section. We now present results for SCS-SASN that show a significant difference between 1-threshold systems and θ-threshold systems where $\theta \geq 2$.

Theorem 22. *Let $\mathcal{S} = (G(V, E), \mathbf{F}, W)$ be a 1-threshold SyDS. The SCS-SASN problem for $\mathcal{S}$ can be solved in $O(|V| + |E|)$ time. Moreover, the solution is unique.*

Proof: Call a given unsalvageable node of G a **fringe** node if it has at least one neighbor that is salvageable. We now argue that the set of all fringe nodes in G is the smallest critical set whose removal ensures that all salvageable nodes are saved from being affected.

First, observe that for a 1-threshold system, a given node is salvageable iff its initial state is 0 and the initial states of all its neighbors are also 0. (Thus, determining whether a node v is salvageable can be done in time $O(\text{degree}(v))$ time, where the degree is the number of edges incident on v.) It can be seen that removing all the fringe nodes saves all salvageable nodes. Further, this is the smallest critical set since if any fringe node is not removed, all of its salvageable neighbors would become affected. Thus, there is a unique smallest critical set for the system.

As mentioned above, determining whether a node v is salvageable can be done in $O(\text{degree}(v))$ time. Therefore, the time to identify all salvageable nodes is $O(\sum_{v \in V} \text{degree}(v)) = O(|E|)$. The set of fringe nodes consists of those nodes that are initially 0, have at least one neighbor that is initially 1, and have at least one neighbor that is salvageable. Thus, once all salvage-

able nodes have been identified, determining all the fringe nodes can also be done in $O(|E|)$ time. Thus, the set of all fringe nodes can be found and output in $O(|V| + |E|)$ time. □

The next set of results concerns the SCS-SASN problem for θ-threshold systems, where $\theta \geq 2$.

Theorem 23. *(a) For any integer $\theta \geq 2$, the SCS-SASN problem is **NP**-hard for θ-threshold systems.*

*(b) There is an integer θ such that for θ-threshold systems, there is no polynomial time approximation algorithm for the SCS-SASN problem with a performance guarantee of $(1 - \epsilon)\ln(|Z|)$ for any $\epsilon > 0$, where Z is the set of salvageable nodes in the system, unless **P** = **NP**.*

Proof of Part (a): This result can be shown by a simple modification to the reduction from Minimum Vertex Cover to the SCS-MNA problem given in the proof of Theorem 20. The modification is that the set Z contains only $m = |E|$ nodes, one corresponding to each edge of G. In the resulting SyDS, the salvageable nodes are those in the set Z. Nodes in the set Y are unsalvageable. (Each of them has θ neighbors who are initially affected.) It can be verified that G has a vertex cover of size at most k iff there is a subset C of Y, with $|C| \leq k$, whose removal saves all the nodes of Z from becoming affected.

Proof of Part (b): We use a reduction from the **Minimum Set Cover** (MSC) problem [120]: Given a universal set $U = \{u_1, u_2, \ldots, u_n\}$ and a collection $C = \{C_1, C_2, \ldots, C_m\}$ of subsets of U, find a minimum cardinality subcollection C' of C such that the union of the sets in C' is equal to U. We will use the fact that the MSC problem cannot be approximated to within the factor $(1 - \epsilon)\ln(|U|)$ for any $\epsilon > 0$, unless **P** = **NP** [272].

Given an instance of the MSC problem, let q denote the maximum number of occurrences of an element of U in the sets in C. We construct an instance of the SCS-SASN problem as follows. The node set of the underlying graph of the SyDS consists of three pairwise disjoint sets X, Y and Z, where $X = \{x_1, x_2, \ldots, x_q\}$ is the set of initially affected nodes, $Y = \{y_1, y_2, \ldots, y_m\}$ is in one-to-one correspondence with the collection C and $Z = \{z_1, z_2, \ldots, z_n\}$ is in one-to-one correspondence with the set U. Thus, the initial states of the nodes in X are 1 while those of the nodes in $Y \cup Z$ are 0. The edges of the underlying graph are as follows.

(a) Each node in X is adjacent to every node in Y.

(b) Consider each element $u_i \in U$ and suppose u_i appears in sets $C_{i_1}, C_{i_2}, \ldots, C_{i_r}$, for some $r \leq q$; then, node z_i is joined to the first $q - r$ nodes of X and the r nodes $y_{i_1}, y_{i_2}, \ldots, y_{i_r}$ of Y.

The threshold for each node is set to q. Thus, the constructed SyDS is a q-threshold system.

It can be seen that none of the nodes in Y is salvageable while all the nodes in Z are salvageable. Thus, every critical set must be a subset of Y. It can also be verified that any

critical set of size α corresponds to a solution to the MSC problem with α subsets from C and vice versa. It follows that if there is an approximation algorithm for the SCS-SASN problem with a performance guarantee of $(1-\epsilon)\ln(|Z|)$ for the SCS-SASN problem, where $\epsilon > 0$, then, since $|Z| = |U|$, there is an approximation algorithm with a performance guarantee of $(1-\epsilon)\ln(|U|)$ for the MSC problem. The result of Part (b) follows with $\theta = q$. □

We now discuss an approximation algorithm for the SCS-SASN problem with a performance guarantee of H_s, where s is the number of salvageable nodes in the system and $H_s = \sum_{i=1}^{s}(1/i)$ is the s^{th} **Harmonic Number**. Since $H_s < 1 + \ln(s)$, this algorithm shows that the lower bound result of Part (b) of Theorem 23 is nearly tight. Moreover, our approximation algorithm is valid even when nodes have different thresholds (i.e., for systems with heterogeneous thresholds).

The idea is to reduce the SCS-SASN problem to a more general form of the MSC problem, called the **Set Multicover** (SMC) problem [322]. Like the MSC problem, the input to the SMC problem consists of the set U and the collection C of subsets of U. In addition, for each element $u \in U$, a coverage requirement $\sigma_u \in \mathbb{Z}^+$ is also given, and the goal of the SMC problem is to pick a minimum cardinality subcollection C' of C such that for each element $u \in U$, there are at least σ_u subsets in C' that contain u. An approximation algorithm with a performance guarantee of H_p, where $p = |U|$, is known for the SMC problem [322]. This fact is used in our approximation algorithm for the SCS-SASN problem shown in Figure 5.2. It can be seen that the approximation algorithm runs in polynomial time. We now prove its correctness and its performance guarantee.

Theorem 24. *The critical set Q produced by the algorithm in Figure 5.2 ensures that none of the salvageable nodes is affected. Also, $|Q| \leq |Q^*|H_s$, where Q^* is an optimal critical set and s is the number of salvageable nodes in the system.*

Proof: We first observe that the set Z constructed in Step 2 of the algorithm contains all the salvageable nodes. The reason is that no node in Y is salvageable since for each node $v \in Y$, the number of initially affected neighbors is at least the threshold of v. Moreover, by removing all the nodes in Y, we can ensure that none of the nodes in Z is affected.

We now argue that the Q produced by the algorithm is indeed a critical set; that is, the removal of Q ensures that none of the nodes in Z is affected. To see this, consider any node $v_i \in Z$ and let p_i and q_i denote the number of neighbors of v_i in I and Y respectively. The coverage requirement for v_i was chosen as $\sigma_i = \max\{0,\ p_i + q_i - \theta_i + 1\}$. If $\sigma_i = 0$, then $p_i + q_i \leq t_i - 1$, and therefore, v_i cannot be affected. If $\sigma_i \geq 1$, then the number of neighbors of v_i after removing all the nodes in Q is at most $p_i + q_i - \sigma_i$ which is at most $\theta_i - 1$ by the definition of σ_i. Hence, v_i will not get affected.

To prove the performance guarantee, we have the following claim.

Claim: Any critical set of size α corresponds to a solution to SMC problem with α sets and vice versa.

Proof of Claim 1: Consider any critical set $Q' = \{v_{j_1}, v_{j_2}, \ldots, v_{j_\alpha}\}$ of size α. We argue that the corresponding collection $C' = \{C_{j_1}, C_{j_2}, \ldots, C_{j_\alpha}\}$ of sets is a solution to the SMC problem. Since Q' is a critical set, for any node $v_i \in Z$, after the removal of Q', the number of neighbors of v_i in $I \cup (Y - Q)$ is at most $\theta_i - 1$. Using this fact, it can be verified that each node $v_i \in Z$ is covered at least σ_i times by C'. In other words, C' is a solution to the SMC problem. The proof that any solution to SMC problem with α sets corresponds to a critical set of size α is similar, and this establishes Claim 1.

To establish the performance guarantee, let Q^* be an optimal critical set. Thus, by the above claim, there is a solution to the SMC problem with at most $|Q^*|$ sets. Since the approximation algorithm for the SMC problem provides a performance guarantee of H_s, where $s = |U| = |Z|$, the size of the solution to the SMC problem obtained in Step 4 of the algorithm is at most $|Q^*|H_s$. Since the approximation algorithm produces a critical set whose size is exactly the number of sets in the approximate solution to the SMC problem, we have $|Q| \leq |Q^*|H_s$. □

5.4 Heuristics for Finding Small Critical Sets

5.4.1 Overview

The complexity results presented in Section 5.3 point out the difficulty of developing heuristics with provably good performance guarantees for the SCS-MNA and SCS-MUN problems. So, we focus on the development of heuristics that work well in practice for one of these problems, namely SCS-MNA. In this section, we present two such heuristics. The first heuristic uses a greedy set cover computation. The second heuristic relies on a potential function, which provides an indication of a node's ability to affect other nodes. Empirical evaluation of these heuristics on several social networks is discussed in Section 5.5.

5.4.2 Covering-Based Heuristic (CBH)

Given a SyDS $\mathcal{S} = (G(V, E), \mathbf{F}, W)$ and the set $I \subseteq V$ of nodes whose initial state is 1, one can compute the set $S_i \subseteq V$ of nodes that changes to state 1 at the i^{th} time step, $1 \leq i \leq T$, for the time $T \leq |V|$ at which the system reaches a fixed point. (This can be done efficiently as explained in the proof of Lemma 19.) The covering-based heuristic (CBH), whose details appear in Figure 5.3, chooses a critical set C as a subset of S_i for some suitable i. The intuitive reason for doing this is that each node v in S_{i+1} has at least one neighbor w in S_i. (Otherwise, v would have changed to 1 in an earlier time step.) Therefore, if a suitable subset of S_i can be chosen as critical so that none of the nodes in S_{i+1} changes to 1 during the $(i+1)^{\text{st}}$ time step, the contagion cannot spread beyond S_i. Consistent with the goal of the SCS-MNA problem, we seek to halt the diffusion process as early as possible

in Step 2 of Figure 5.3. There are two means by which this can be accomplished at each time i: Step 2(i)(a) which checks whether S_i can itself serve as a critical set (i.e., $|S_i| \leq \beta$) and Step 2(i)(e) which checks whether the covering procedure discussed below produces a suitable critical set from S_i.

In general, when nodes have thresholds ≥ 2, the problem of choosing at most β nodes from S_i to prevent a maximum number of nodes in S_{i+1} from changing to 1 corresponds to the Set Multicover (SMC) problem mentioned in Section 5.3. Step 2(i)(b) constructs an instance of SMC, where each set in the collection Γ corresponds to a node of S_i. Since SMC is **NP**-hard, a greedy approach is used (Step 2(i)(c)) for this covering problem [322]. This approach iterates over the sets of Γ; in each iteration, the chosen set from Γ corresponds to a node from S_i that contributes to saving the largest number of nodes in S_{i+1} from becoming affected. Thus, the subcollection Γ' produced by Step 2(i)(c) corresponds to a subset of S_i.

If the conditions in Steps 2(i)(a) and 2(i)(e) are not satisfied for any T, $1 \leq i \leq T-1$, then a solution that has the smallest number of nodes whose coverage requirement is not met is chosen as the output. We now establish the running time of CBH.

Proposition 25. *Let $G(V, E)$ denote the underlying graph of the given SyDS $\mathcal{S}$. The running time of CBH is $O(|V||E|)$.*

Proof: Consider the description of CBH in Figure 5.3. Using Lemma 19, Step 1 can be implemented to run in $O(|V| + |E|)$ time. In each iteration of Step 2(i), it can be seen that the dominant part of the running time is due to the greedy heuristic for SMC. This heuristic can be implemented to run in $O(\sigma)$ time, where σ is the sum of the sizes of the given sets [322]. In CBH, since the set constructed for any node v is of size at most degree(v), the sum of the sizes of all the sets is at most $\sum_{v \in V} \text{degree}(v) = |E|$. Thus, each execution of the greedy heuristic runs in $O(|E|)$ time. Since Step 2(i) runs the greedy set cover heuristic $T-1$ times, the running time of that step is $O(T|E|)$. Since $T \leq |V|$, the worst-case running time of CBH is $O(|V||E|)$. □

5.4.3 Potential-Based Heuristic

Here, we provide the details of the potential-based heuristic. The idea is to assign a potential to each node v depending on how early v is affected and how many nodes it can affect later. Nodes with larger potential values are more desirable for inclusion in the critical set. While the covering-based heuristic chooses a critical set from one of the S_i sets, the potential based approach may select nodes in a more global fashion from the whole graph.

We assume that set S_i of newly affected nodes at time i has been computed for each i, $1 \leq i \leq T$, where T is the time at which the system reaches a fixed point. For any node $x \in S_i$, let $N_{i+1}[x]$ denote the set of nodes in S_{i+1} which are adjacent to x in G. The potential $P[x]$ of a node x is computed as follows.

- For each node x in S_T, $P[x] = 0$. (**Justification:** Since there is no diffusion beyond level T, it is useless to have nodes from S_T in the critical set.)
- For each node x in level S_i, $1 \leq i \leq T-1$,

$$P[x] = (T-i)^2 \left[|N_{i+1}[x]| + \sum_{y \in N_{i+1}[x]} P[y] \right]$$

(**Justification:** The term $(T-i)^2$ decreases as i increases. Thus, the term enables us to assign higher potentials to nodes that are affected earlier. The term $|N_{i+1}[x]|$ is included in the expression for potential so that nodes which have a large number of neighbors in the next level become desirable candidates for inclusion in the critical set.)

The steps of the potential-based heuristic (PBH) are shown in Figure 5.4. Steps 1, 2 and 3 compute the potentials for all the nodes in $S_1 \cup S_2 \cup \ldots \cup S_T$ in a bottom-up fashion. Step 4 indicates how the critical set is chosen. We now establish the running time of PBH.

Proposition 26. *Let $G(V, E)$ denote the underlying graph of the given SyDS $\mathcal{S}$. The running time of PBH is $O(|V| + |E|)$.*

Proof: Consider the description of PBH in Figure 5.4. From Lemma 19, Step 1 can be implemented to run in $O(|V|+|E|)$ time. Step 2 runs in $O(|V|)$ time. In Step 3, the potential for each node v is computed by examining the neighbors of v. Thus, for any node v, the time used to compute v's potential is $O(\text{degree}(v))$. Hence, the time used to compute the potentials of all the nodes is $O(\sum_{v \in V} \text{degree}(v)) = O(|E|)$. Step 4 can be carried out in $O(|V|)$ time since the β^{th} largest potential value can be computed in $O(|V|)$ time using the well known linear time selection algorithm [92]. Hence, the overall running time of PBH is $O(|V| + |E|)$. □

The next section presents results on the performance of CBH and PBH.

5.5 Blocking Experiments and Results

5.5.1 Overview

We first describe the social networks used for testing. Next, we compare the blocking performance of our heuristics with those of five known heuristics. When then provide further results from an empirical evaluation of our methods to illustrate additional aspects of blocking. We also provide timing data for our heuristics to emphasize the tradeoff between execution speed and quality of critical sets.

5.5.2 Networks and Generation of Seed Nodes

Table 4.1 provides selected features of the three social networks used in this study. (We refer to these as "realistic social networks," since they were produced by mining real social datasets.) We assume that all edges are undirected to accentuate diffusion and thereby test the heuristics more stringently.

All our diffusion experiments use the following method of generating seed sets. For a given value of number n_s of seeds, 100 sets of seed nodes are determined from each network to provide a range of cases for testing the heuristics. Each set of seed nodes is taken from a 20-core, a subgraph in which each node has a degree of at least 20 [282]; thus, each seed node has a degree of at least 20. The 20-core provides a good compromise between selecting high-degree nodes, and having a sufficiently large pool of nodes to choose from so that there is little overlap among sets of seed nodes. Moreover, each set of seed nodes forms a connected subgraph, which fosters diffusion. Thus, the test cases utilize two means (namely, seeding of high-degree nodes and ensuring that they form a connected subgraph) to foster diffusion and hence tax the heuristics.

5.5.3 Comparisons Of CBH and PBH With Other Blocking Methods

We now turn to evaluating the heuristics in halting and delaying diffusion by first comparing our heuristics with five other heuristics: (1) randomly setting nodes critical (RCH), (2) setting high-degree nodes critical (HCH), (3) setting critical the nodes with greatest betweenness centrality (BCH), (4) setting critical nodes with greatest eigenvector centrality (ECH), which for undirected graphs is the same as the HITS method [173], and (5) the maximum eigenvalue drop method called NetShield [308]. As mentioned earlier, methods referred to in items (2) through (5) above were proposed in the literature for blocking simple contagions. The random choice method (RCH) serves as a baseline. In the remainder of this section, the above five approaches are referred to collectively as *other blocking methods*.

Our experimental procedure is as follows. We consider the three networks (`epinions`, `slashdot` and `wikipedia`) discussed in Section 5.5.2 and set the threshold value $\theta = 2$. We use two values (namely, 2 and 3) for the number n_s of seed nodes. For each value of n_s, 100 sets of seed nodes are generated from the 20-core of the corresponding social network. The budget β on the number of blocking nodes is set to 500. We chose the values of n_s and β to give the blocking methods a good opportunity to succeed since the value of β/n_s is reasonably large (250 and 166.7 respectively for $n_s = 2$ and 3). For each combination of social network, value of n_s and blocking method, we ran 100 diffusion instances (one corresponding to each seed set) and recorded the number of instances that resulted in a cascade. (Recall from Section 5.2.4 that a cascade occurs when 95% or more of the nodes that can be affected are indeed affected.)

Our experimental results for `epinions`, `slashdot`, and `wikipedia`, given in Tables 5.1, 5.2, and 5.3, respectively, can be summarized as follows.

1. For all three networks, CBH and PBH perform well in blocking diffusion, allowing almost no cascade for any diffusion instance. The random choice method RCH allows the greatest fractions of cascades for all three networks. (The reason for the poor performance by RCH is that nodes of degree 1 account for between 35% and 50% of nodes in these networks. RCH has a good chance of choosing such nodes as critical nodes even though other nodes which are better suited to block the complex contagion are available.)

2. For the `epinions` network, HCH, BCH, ECH/HITS and NetShield permit cascades to occur between 69% and 74% of the time when $n_s = 2$. For `slashdot`, the range decreases (30% to 40%), while for `wikipedia`, the range is wider (53% to 70%). For all three networks, when the number of seed nodes increases to $n_s = 3$, the fractions of cascades increase markedly for the other blocking methods. These results suggest that in the three networks, complex contagions are able to circumvent the nodes of high degree or high centrality and find alternate paths to reach many nodes.

3. It is also useful to compare the performance of the other blocking methods. The high degree heuristic performs just as well as—in some cases better than—the more sophisticated methods in blocking complex contagions. This result provides an interesting contrast with the findings in [308] for simple contagions, where NetShield was shown to have the best performance.

4. The results also show that *all* five methods are more effective in blocking complex contagions in `slashdot` than in `epinions` and `wikipedia`. We believe that this is related to the average clustering coefficients (C_{ave}) of the networks. As can be seen from Table 4.1, the `slashdot` network has the smallest C_{ave} among the three networks. For complex contagions, where multiple affected neighbors are required to propagate a contagion, the clustering coefficient is potentially an important factor in determining a network's ability to spread contagions. The C_{ave} values for `epinions` and `wikipedia` are somewhat more typical of social networks, while that for `slashdot` is smaller [243]. These results suggest that networks with smaller C_{ave} can still spread complex contagions, but blocking their progression is easier.

We end this section by noting that the other blocking methods are sensitive to the number of seed nodes: an increase in numbers n_s of seed nodes from 2 to 3 significantly decreases the effectiveness of these methods. We will see this sensitivity to n_s in CBH and PBH as well in Section 5.5.4; however, in our methods, this sensitivity is pushed further out to larger values of n_s.

Table 5.1: Performance of blocking methods for `epinions` ($\theta = 2$ and $\beta = 500$).

Blocking Method	**Fraction of Cascades** $n_s = 2$	**Fraction of Cascades** $n_s = 3$
CBH	0.00	0.00
PBH	0.00	0.01
Random (RCH)	0.94	1.00
High Degree (HCH)	0.73	0.91
Betweenness Centrality (BCH)	0.69	0.95
Eigenvector Centrality (ECH/HITS)	0.74	0.89
NetShield	0.70	0.92

Table 5.2: Performance of blocking methods for `slashdot` ($\theta = 2$ and $\beta = 500$).

Blocking Method	**Fraction of Cascades** $n_s = 2$	**Fraction of Cascades** $n_s = 3$
CBH	0.00	0.00
PBH	0.00	0.00
Random (RCH)	0.60	0.95
High Degree (HCH)	0.38	0.58
Betweenness Centrality (BCH)	0.40	0.68
Eigenvector Centrality (ECH/HITS)	0.32	0.76
NetShield	0.30	0.65

5.5.4 Empirical Evaluation of CBH and PBH

We carried out a parametric study to empirically evaluate the performance of PBH and CBH. Table 5.4 lists the parameters and values used in the parametric study.

The test plan consists of running 100 simulations of diffusion instances (one instance per seed node set) on each of the three networks for all combinations of θ, n_s, and β values shown in Table 5.4. Our simulator outputs for each node v the time at which v is affected. The heuristics use this as input data and calculate one set of β critical nodes for each iteration (i.e., diffusion instance). The simulations are then repeated, but now they include the critical nodes, so that the decrease in the total number of affected nodes caused by a critical set can be quantified. The main results from this full-factorial parametric study are outlined below. For brevity, when results for different networks and combinations of parameter values are similar, only results for one representative network and some specific combinations of parameter values are presented.

Table 5.3: Performance of blocking methods for `wikipedia` ($\theta = 2$ and $\beta = 500$).

Blocking Method	**Fraction of Cascades** $n_s = 2$	**Fraction of Cascades** $n_s = 3$
CBH	0.00	0.00
PBH	0.00	0.01
Random (RCH)	0.96	1.00
High Degree (HCH)	0.54	0.91
Betweenness Centrality (BCH)	0.70	0.95
Eigenvector Centrality (ECH/HITS)	0.54	0.87
NetShield	0.53	0.90

Table 5.4: Parameters and values of the full-factorial parametric study. (For each network and each combination of parameter values, 100 diffusion instances were run.)

Networks	**Thresholds** (θ)	**Numbers of Seeds** (n_s)	**Budgets on Critical Nodes** (β)
`epinions`, `slashdot`, `wikipedia`	2, 3, 5	2, 3, 5, 10, 20	0, 5, 10, 20, 50, 100, 500, 1000

Effect of critical set budget and number of seed nodes on cascades. Figure 5.5 compares the probability of cascades (i.e., the fraction of iterations that cascade) for CBH and PBH across all three networks for $n_s = 5$ (solid curves) and 10 (dashed curves). As one would expect, for both the methods, given a value for n_s, the probability of cascades decreases as the budget β on critical nodes increases. The two plots show how a doubling of the number of seed nodes can produce a six-fold increase in the probability of cascade; compare the `slashdot` data for $\beta = 100$, where increasing the number of seed nodes from 5 to 10 increases the probability of cascade from about 10% to 60%. The blocking performance of CBH and PBH are similar for `slashdot`; however, CBH significantly outperforms PBH for `epinions` and `wikipedia` when the number of seed nodes is increased to 10. As mentioned in Section 5.5.3, this behavior can be attributed to the significantly smaller average clustering coefficient of `slashdot` compared to the other networks.

Effect of threshold and critical set budget on cascades. The effect of increasing threshold values on the probability of cascade is shown in Figure 5.6 for the `epinions` network using CBH for four values of β; the two plots correspond to $n_s = 20$ and 50 respectively. In Figure 5.6(a), if we use $\theta = 2$ and $\beta = 1000$ as a baseline case, we see that increasing the threshold from 2 to 3 enables the number of blocking nodes to be reduced by 500 while achieving a slightly lower probability of cascade (36% vs. 27%). If the threshold is increased to 5, the number of blocking nodes can be decreased by an order of magnitude, from 1000 to 100, with only a small increase in the probability of cascade (36% vs. 42%).

Thus, increasing the threshold can significantly decrease the critical node budget needed to achieve the same probability of cascade. However, by comparing Figures 5.6(a) and 5.6(b), it is seen that increasing the number of seed nodes from 20 to 50 significantly increases the cascade probability.

Effect of critical nodes on the final number of affected nodes. For each value of β, the final fractions of nodes affected (i.e., the final spread sizes) in the `slashdot` network, arranged in increasing numerical order for the 100 iterations, are plotted in Figure 5.7(a) for $\theta = 3$. To explain this figure, consider the plot for $\beta = 10$. The curve makes a steep transition from 0 (i.e., essentially zero diffusion) to 44% at 30% of iterations, indicating that 70% of iterations produce significant diffusion. Our results in Chapter 4 can be used to determine that with $\theta = 3$, this fraction (44%) of affected nodes is indeed a cascade as defined in Section 5.2.4 (even though the number of affected nodes is far less than 100%). Thus, with $\theta = 3$ and $\beta = 10$, the probability of a cascade under CBH is 70%. As the curves in Figure 5.7(a) show, increasing the value of β decreases the probability of a cascade, and the value $\beta = 500$ halts all diffusion. We note that the maximum spread size across all values of β is a constant.

Figure 5.7(b) shows similar data for the case $\theta = 5$ with the same `slashdot` network. The two plots illustrate how larger thresholds can significantly increase the effectiveness of critical nodes; i.e., the transitions from small to large final spread sizes shift to the right, thereby decreasing the probability of cascade. Again the upper limit spread size is independent of β for a fixed θ. (The final spread size is also smaller when θ is increased from 3 to 5.)

Effect of critical set budget in delaying peak spread size. In Figure 5.8, the *average* number of newly affected nodes (as a fraction of the total number of nodes) in each time step over the 100 diffusion instances is plotted for different β values. These results are for `epinions` with $\theta = 2$ and $n_s = 10$ when PBH is used. From the figure, it can be seen that while a budget of $\beta = 500$ does not halt the diffusion, it slows down the diffusion compared to the other budget sizes, moving the time of the peak in the number of newly affected nodes from 3 to 6. In applications such as disease propagation over a population or worm propagation in computer networks, this delay provides additional time for decision-makers to devise appropriate intervention strategies [88, 267] and could be used analogously for social contagions.

Blocking performance of CBH and PBH with a small number of seed nodes. Figure 5.9 examines the regime of small numbers of seed nodes, and depicts the probability of cascades (i.e., fraction of iterations that cascade) as a function of β for `wikipedia`. A $(1/\beta)$ behavior is observed, so that the number of cascades drops off sharply with increasing budget, but to completely eliminate all cascades, $\beta = 500$ is required for both heuristics when $n_s = 5$. In most of our experiments, $\beta/n_s \geq 100$ was required to halt all diffusion.

Parameter settings that show significant differences between the blocking perfor-

mances of CBH and PBH. Figure 5.10 provides results showing the greatest differences in performance between the two heuristics in thwarting diffusion. This is done by comparing the probabilities of cascade. From Figure 5.10, it is seen that for the `epinions` network with $n_s = 10$, the probability of a cascade under CBH and PBH are respectively around 20% and 60% when β is fixed at 500. A similar difference is seen for the `wikipedia` network with $n_s = 20$.

It can be seen from Figure 5.10 that, critically, once a value of n_s produces even a small number of cascades for a fixed β, further increases in n_s will result in large increases in fractions of iterations that cascade. For both networks, we observe that $\beta/n_s = 500/3 = 167$ will halt all diffusion. However, the `epinions` data illustrate that for $n_s = 5$ and $\beta = 500$, the ratio of $\beta/n_s = 100$ does not halt all diffusion for either CBH or PBH. Hence, as n_s increases, the ratio β/n_s required to stop all diffusion can be quite large.

Timing Profiles for CBH and PBH. We now compare the execution times of CBH and PBH. To carry out this study, serial implementations of CBH and PBH were run on a single core (with 2GB of memory) of a system with a 3 GHz Intel Xeon processor. Figure 5.11 depicts the execution times for each heuristic for $\beta = 5$ as n_s is varied. The two plots are for `epinions` and `wikipedia` respectively, with different curves in the same plot corresponding to the two heuristics and three different threshold values (2, 3 and 5). For the `epinions` network, Figure 5.11(a), these times translate into a maximum of roughly 1.5 hours for CBH to determine 100 sets of critical nodes, versus less than 5 minutes for PBH. The execution times for `wikipedia` are significantly smaller than those for `epinions` since the former network is smaller in size (by a factor of 10) than the latter. In general, PBH is clearly faster than CBH, and this empirical result is in agreement with the worst-case running time estimates for the two methods obtained in Section 5.4. Since CBH is better than PBH in blocking complex contagions, we see a tradeoff between blocking performance and speed of algorithm execution.

5.6 Summary

We considered the problem of inhibiting the propagation of complex contagions in social networks. We developed analytical formulations for several versions of the problem. We showed that obtaining provably good heuristics for these problems is computationally intractable. We developed two intuitively appealing heuristics for the problem and showed that they perform well on several social networks. Both theoretical and empirical results also demonstrated ways in which simple contagions differ from complex contagions.

Input: A SyDS with underlying graph $G(V, E)$, a threshold $\theta_i \geq 0$ for each node $v_i \in V$, the set $I \subseteq V$ of initially affected nodes.

Output: A critical set $Q \subseteq V - I$ such that removal of Q ensures that none of the salvageable nodes is affected.

Steps:

1. Let $Y = \{v_i \in V \ : \ v_i$ is adjacent to θ_i or more nodes in $I\}$. (**Note:** Y is the set of nodes that *cannot* be salvaged. The algorithm returns a critical set $Q \subseteq Y$.)

2. Let $Z = V - (I \cup Y)$. (**Note:** Z is the set of salvageable nodes.)

3. Construct an instance of the Set Multicover (SMC) problem as follows.

   (a) Let $U = Z$. For each node $v_i \in Z$, suppose v_i is adjacent to p_i nodes of I and q_i nodes of Y. Then, the coverage requirement σ_i for v_i is given by $\sigma_i = \max\{0, \ p_i + q_i - \theta_i + 1\}$.

   (b) For each node $v_j \in Y$, create the set $C_j = \{v_i \in Z \ : \ \sigma_i > 0$ and $\{v_i, v_j\} \in E\}$, $1 \leq j \leq |Y|$. (**Note:** Nodes with coverage requirement ≤ 0 correspond to nodes which cannot get affected since their thresholds are too large. So, they need not be considered in creating the SMC instance.)

4. Use the known approximation algorithm with a performance guarantee of H_p, where $p = |U|$, to find a solution to the SMC problem instance constructed in Step 3.

5. If the solution obtained in Step 4 is the set $C' = \{C_{j_1}, C_{j_2}, \ldots, C_{j_r}\}$, then output $Q = \{v_{j_1}, v_{j_2}, \ldots, v_{j_r}\}$ as the critical set.

Figure 5.2: An approximation algorithm for the SCS-SASN problem.

Input: A SyDS $\mathcal{S} = (G(V, E), \mathbf{F}, W)$, the set $I \subseteq V$ of nodes whose initial state is 1, the upper bound β on the size of critical set and the number of simulation steps $T \leq |V|$.

Output: A critical set $C \subseteq V - I$ whose removal leads to a small number of new affected nodes.

Steps of the heuristic:

1. Simulate the system for T time steps and determine sets S_1, S_2, ..., S_T, where S_i is the set of newly affected nodes at time i, $1 \leq i \leq T$.

2. **Comment:** This step uses a greedy covering procedure.

   (i) **for** $i = 1$ **to** $T - 1$ **do**

   (a) **if** $|S_i| \leq \beta$, **then** output critical set $C = S_i$ and **stop**.

   (b) For each node $v_j \in S_i$, construct the set Γ_j which consists of all the neighbors of v_j in S_{i+1}. Let Γ denote the collection of all the sets constructed. (We have $|\Gamma| > \beta$.) The coverage requirement for each node v_k of S_{i+1} is $n_k - \theta_k + 1$ where n_k is the number of affected neighbors of v_k at times $\leq i$, and θ_k is the threshold of v_k.

   (c) Use a greedy approach to find a subcollection Γ' of Γ containing at most β sets that satisfies the coverage requirement for as many elements of S_{i+1} as possible.

   (d) Let the critical set C_i consist of the nodes of S_i corresponding to the elements of Γ'.

   (e) **if** all nodes in S_{i+1} are covered appropriately **then** output $C = C_i$ and **stop**.

   (f) Let $R_i \subseteq S_{i+1}$ be the set of nodes whose coverage requirement is not met by Γ'.

   (ii) Among all the critical sets C_i constructed in Step 2(i)(c), output the earliest C_i such that $|R_i|$ is minimum.

Figure 5.3: Details of the covering-based heuristic.

Input: A SyDS $\mathcal{S} = (G(V, E), \mathbf{F}, W)$, the set $I \subseteq V$ of nodes whose initial state is 1, the upper bound β on the size of critical set.

Output: A critical set $C \subseteq V - I$ whose removal leads to a small number of new affected nodes.

Steps of the heuristic: (The description below uses some notation presented in the main text.)

1. Simulate the system $\mathcal{S}$ and determine sets S_1, S_2, ..., S_T, where T is the time step at which $\mathcal{S}$ reaches a fixed point and S_i is the set of newly affected nodes at time i, $1 \leq i \leq T$.

2. **for** each node $x \in S_T$ **do**

   $P[x] = 0$.

3. **for** $i = T - 1$ **downto** 1 **do**

   for each node $x \in S_i$ **do**

   (a) Find $N_{i+1}[x]$ and let $P[x] = |N_{i+1}[x]|$.

   (b) **for** each node $y \in N_{i+1}[x]$ **do**

   $P[x] = P[x] + P[y]$

   (d) Set $P[x] = (T - i)^2 P[x]$.

4. Let the critical set C contain β nodes with the highest potentials among all the nodes. (Break ties arbitrarily.) Output C.

Figure 5.4: Details of the potential-based heuristic.

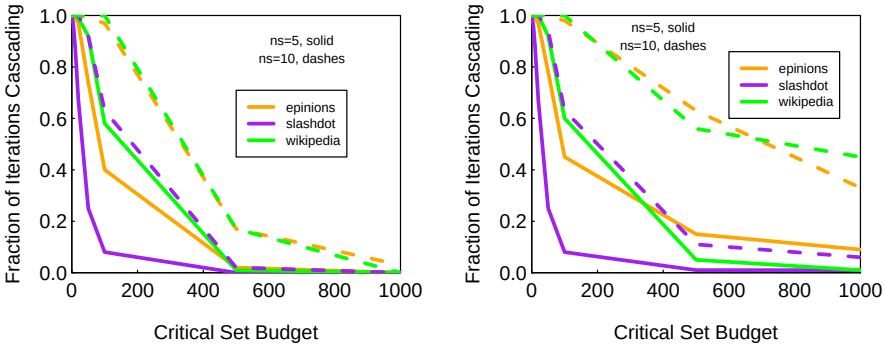


Figure 5.5: Comparisons of the (a) CBH and (b) PBH for inhibiting diffusion in all networks, for two seed set sizes and $\theta = 2$.

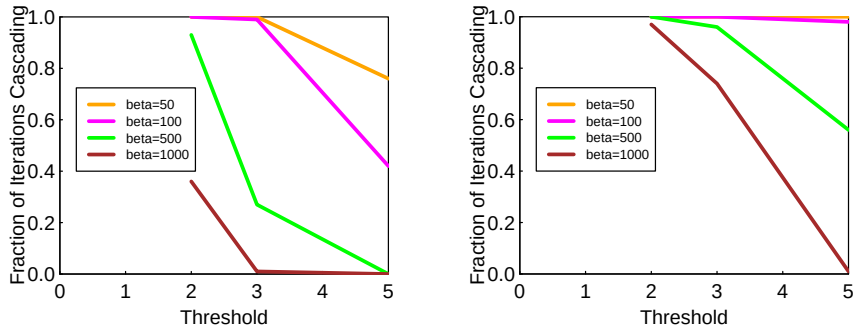
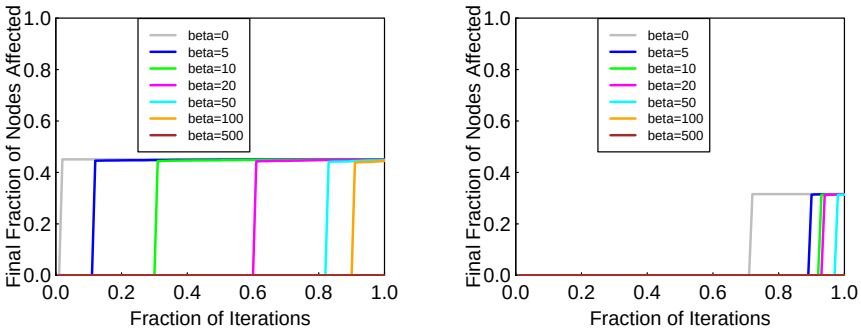


Figure 5.6: Effect of threshold on the probability of generating a cascade in the `epinions` network, with $\beta = 50$, 100, 500 and 1000. CBH is used in both plots, with $n_s = 20$ in (a) and $n_s = 50$ in (b).



(a) (b)

Figure 5.7: (a) Final number of affected nodes (arranged in increasing order for each of the 100 iterations) for the `slashdot` network and CBH heuristic for $n_s = 10$ and (a) $\theta = 3$ and (b) $\theta = 5$.

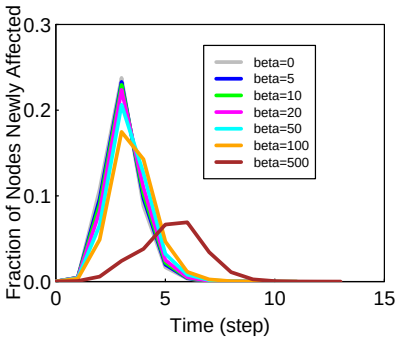


Figure 5.8: Average curves of newly affected nodes for PBH for the case $\theta = 2$, $n_s = 10$, and different values of β with the `epinions` network.

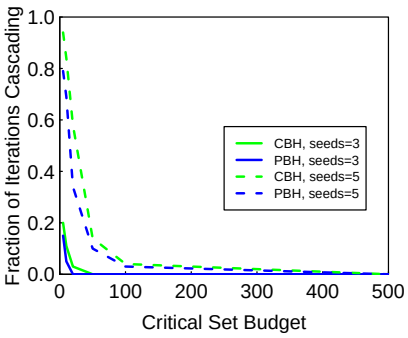


Figure 5.9: Comparisons of the CBH and PBH in inhibiting diffusion in the `wikipedia` network for $\theta = 3$.

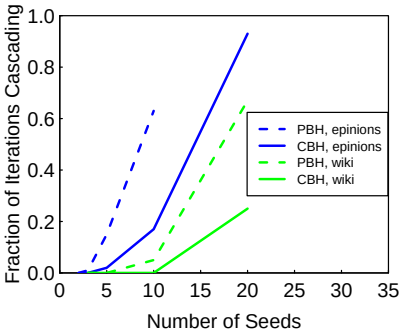
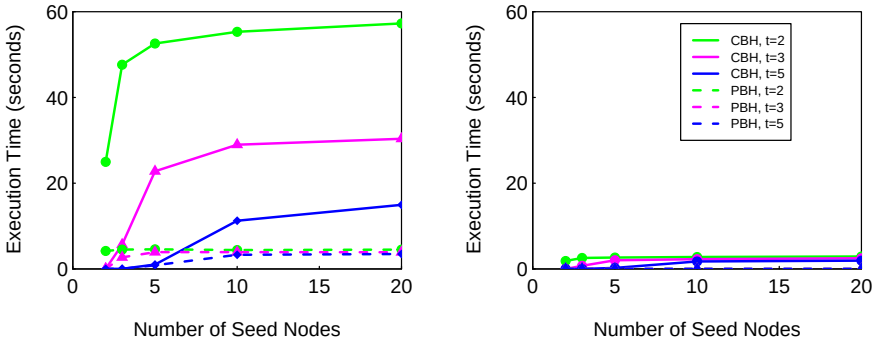


Figure 5.10: Comparisons of the CBH and PBH in inhibiting diffusion in the `epinions` with $\theta = 2$ and `wikipedia` (`wiki`) with $\theta = 3$. The number β of critical nodes is 500.



(a) (b)

Figure 5.11: Times for CBH and PBH to compute one set of critical nodes for $\beta = 5$ as a function of number of seed nodes for three threshold values. Part (a) is for the `epinions` network and Part (b) is for the `wikipedia` network. Times are averages over 100 iterations. The legend is the same for both plots, where "t" represents threshold.

Chapter 6

Effects of Network Structure

6.1 Introduction

6.1.1 Background and Motivation

We explore the effect of graph structure on contagion dynamics, quantified in terms of spread size, in multiple networks from two well-accepted families of graphs, scale-free (SF) and exponential-decay (ED). We evaluate simple and complex contagions with and without frozen nodes. We use the frozen or critical node model described in Section 1.7.1, where frozen nodes are selected either randomly (**random frozen nodes**) or as the highest degree nodes (**targeted frozen nodes**). Since SF and ED graphs capture many social networks arising in practice (e.g., see [65,81,84,237,238]), many researchers have studied the robustness of these two classes of graphs. (Although the term **robustness** is used differently in multiple contexts, here we define it as the ability of a network to produce significant spread sizes when a specified fraction of nodes is designated as frozen, consistent with [7,65].) For example, [7] observes that for random frozen nodes (which they call *failed* nodes), SF networks produce larger giant connected components than do ED networks. When all seed nodes are from the same giant component C, the simple contagion spread size equals the number of nodes in C. For this reason, it is concluded [7] that for simple contagions, SF graphs are more robust than ED networks to random frozen nodes. There are well-recognized differences between simple and complex contagion diffusion, as demonstrated in [67]. Hence, [65] built on the work of [7] by using similar networks (specifically, one SF network and one ED network) to model complex diffusion. They produced the opposite result for complex contagions: SF networks are less robust than ED networks under both frozen node models. In particular, the following statement appears in the abstract of [65]:

> "I show that scale-free networks are much less robust than exponential networks for the spread of complex contagions, which highlights the value of more homo-

> geneously distributed social networks for the robust transmission of collective behavior."

This point is reiterated in the text. By "homogeneously distributed social networks," he refers to ED networks whose node degrees have less variation than those in SF networks.

Our work builds on that of [7] and [65]. Our goals are to determine whether structure varies across graph instances and if so, how these differences affect simple and complex contagions.

Work incorporating multiple graph instances typically combines (averages) dynamics results across all graph instances with no consideration of dynamics produced by individual graphs. Several studies ignore the effect of clustering coefficient. Others use a different graph instance for each diffusion simulation and again average results so that the effect of graph structure is not explicitly explored. See [80,129,219,290,291]. Other studies use only one graph instance per class and graph size; see for example, [7, 65, 295] [1]. Our results demonstrate that these approaches can be problematic.

Also, [113, 290] show that changes in selected properties (e.g., average clustering coefficient, average node degree) can affect diffusion dynamics. We, in contrast, show large differences in dynamics across graph instances with the *same* properties, with and without frozen nodes. Put another way, [290] states that his "contribution is to provide a general description of how network structure and individual motivations interact to determine participation in collective behaviors [i.e., in spread size], in a manner not reliant on detailed knowledge of said network structure." However, we demonstrate, at least for the network classes and dynamics addressed here, that detailed knowledge beyond typically-cited properties (which is often just degree distribution) is required to explain dynamics. We provide data and a simple explanatory model in Section 6.3.

We believe that our results are broadly applicable to other models of network dynamics, beyond progressive thresholds, where local interactions (represented by graph edges) drive diffusion. For example, our results also hold for the following probabilistic threshold-based diffusion model: a node with threshold θ remains in state 0 if fewer than θ of its neighbors are in state 1; its state changes from 0 to 1 with probability p_{high} when θ or more of its neighbors are in state 1. Among other models, we expect that the collective action models in [290, 291] and various types of threshold models (e.g., [35, 83, 166, 329]) will also produce different dynamics across graph instances, as they are based on local interactions defined by graph structure [2].

Our approach uses a large simulation study in conjunction with statistical tests. This work extends the experiments of [65] on ED and SF networks with average degree $d_{ave} = 4$ and

[1] [7] evaluates one graph instance for each $n = 1000$, 5000, and 10000, for SF and ED networks, where n is the number of graph nodes. [65] utilizes one 10000-node ED network and one 10000-node SF network.

[2] Our work has limitations. Our findings are confined to the conditions of this study. Also, we are not claiming that degree distribution, clustering coefficient, and giant component size are the only network parameters of interest; see discussions of other properties in [122, 231, 239].

average clustering coefficient $CC_{ave} = 0.25$ in the following ways: (i) we utilize two well-known graph construction methods called growth models [25] to generate 30 graphs from each of the SF and ED graph classes; (ii) we study another well-known graph construction method, the configuration model [230,238]; (iii) we investigate a broader range in the fraction f of frozen nodes; (iv) we compare giant component sizes of graphs with and without frozen nodes against those of [7]; and (v) we perform simulations on all of these graphs for simple and complex diffusion while systematically varying the frozen node and seed node sets. Our extension of [7] is the investigation of multiple graph instances. As in [7, 65], we use both random and targeted frozen nodes.

6.1.2 Summary of Contributions

All of our results are statistically significant.

1. Specifying the form of degree distribution (e.g., SF), average degree, and average clustering coefficient are not sufficient to characterize complex contagion network dynamics, with and without frozen nodes. The following three points support this conclusion.

(a) Wide variations in contagion spread for the same graph structure. We observe spread sizes for complex contagions that differ by as much as an order of magnitude from those of [65], over a wide range of conditions including targeted and random frozen nodes, *for graphs with the same structural characteristics.*

(b) Ranking of network classes that differ from previous results. For complex diffusion, contrary to [65], we find that SF networks are more robust than ED networks. Hence, it need not be the case that ED graphs are more robust than SF networks.

(c) Mechanism driving complex diffusion is different from that described in previous work. We find that it is the high-degree nodes in SF networks that drive complex diffusion. We find, contrary to other work, that the homogeneous degree distributions are not conducive to propagating complex contagions. (Compare with quote above.)

2. In the presence of frozen nodes, mean results for giant component and spread sizes across graphs produced by a single construction method can be misleading. In a set of graphs produced by a *single* construction method where networks have the same degree distribution (and therefore the same average degree) and same average clustering coefficient, giant component and spread sizes with frozen nodes may vary by more than a factor of three across graphs.

6.2 Networks and Experimental Parameters

Much of the network construction details are presented in Appendix C. Graph construction methods are not specified in [65], so we cannot replicate them. However, we demonstrate that our graphs match all of the properties specified in [65]. We also show that our graphs possess giant components produced by frozen nodes that agree with those of another study [7].

We use a well-known random attachment model to produce ED networks and a well-known preferential attachment model to generate SF networks [25]. We demonstrate with thousands of graph instances that the basic procedures cannot provide the desired $CC_{ave} = 0.25$; some CC_{ave} values are orders of magnitude less. Hence, we augment these procedures with another well-known method [280] that achieves the requisite CC_{ave} while maintaining the degree distribution property. A similar approach is described in [147]. We refer to these construction methods as **growth models**, wherein a small set of nodes is initially specified and the graph is grown by successive additions of a new node and one or more edges between the new node and pre-existing ones. Random attachment of new edges produces ED networks denoted by `gm-ed` (growth model, exponential-decay), while preferential attachment of new edges produces SF networks, denoted by `gm-sf`. Thirty graph instances of each type were generated by executing the network generation algorithms 30 times. Details regarding constants used in the construction methods and characteristics of the graphs are provided in Appendix C. To match the characteristics of graphs used in [65], our graphs have $d_{ave} = 4$, $CC_{ave} = 0.24$, and the requisite ED and SF degree distributions. These distributions are confirmed in Figure C.3 using the criteria given in [25].

We also evaluate the dynamics of SF networks generated by another well-known construction method called the **configuration model** [230, 238]. We refer to the graphs as type `cm-sf`. As the number of nodes in a graph increases, the configuration model approaches the Chung-Lu model [82], yet another popular construction method. This method is explained in detail in Appendix C. We generate 30 graph instances that possess the same degree distribution and average degree as the SF networks used in the growth model. As with the growth model, we cannot achieve the desired $CC_{ave} = 0.25$ with only the basic procedure and so we augment it with a method of adding edges to complete triangles uniformly at random. The generated graphs have $CC_{ave} = 0.12$, which is below the value used in [65], but still well within the range of CC_{ave} values for social networks [156, 200, 238].

The parameters used in our full factorial experimental simulation study are provided in Table 6.1. The frozen node models (random and targeted) were defined in Section 1.7. Since each graph has $n = 10000$ nodes, the frozen node fraction $f = 10^{-5}$ represents the base case of zero frozen nodes (since $\lfloor nf \rfloor = \lfloor 10^4 \cdot 10^{-5} \rfloor = 0$); this conveniently enables us to plot zero frozen nodes on a logarithmic scale. See Appendix C for more details of the experimental test matrix.

Recall from the example in Figure 1.5 that frozen nodes can produce multiple components in a graph. From the initial graphs and those that result from removing frozen nodes, we

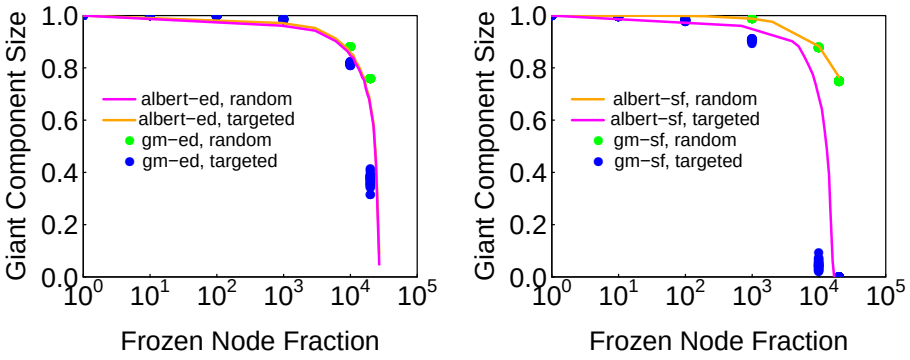
Table 6.1: Test parameters for giant component determinations and diffusive simulations. These conditions produce 90 graph instances, 21,600 giant component determinations, and 864,000 simulated diffusion instances.

Graph Types	**Number of Instances Per Class**	**Types of Frozen Nodes**	**Fraction f of Frozen Nodes**	**Number of Frozen Node Sets Per f**	**Number of Seed Node Sets Per Frozen Node Set**	**Thresholds**
`gm-ed`, `gm-sf`, `cm-sf`	30	random, targeted	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 0.1, 0.2	20	20	1, 2

compute the normalized average sizes of giant components GC_{ave} as a function of frozen node fraction f for the growth model graphs whose dynamics we compare with the corresponding results from [65]. The normalized giant component size is the number of nodes in a giant component divided by the number of nodes in the original network ($0 \leq GC \leq 1$). Figure 6.1 compares our average giant component sizes (data points) with those from [7] (lines). Each curve corresponds to data from a single graph instance, while for each value of f, our data are represented by 30 green and 30 blue circles, one for each network. Each of our data points is an average over 20 GC values, one for each set of frozen nodes per graph in Table 6.1. We always use logarithmic scales for f because there are times when the effect of one frozen node is substantial and the logarithmic scale enables us to clearly see these effects. Although for SF networks and targeted frozen nodes in Figure 6.1(b), our average giant component sizes $GC_{ave} \to 0$ at smaller values of f compared to the results of [7], it is also the case that the gradient is steep in this portion of the curve, and the greater scatter in the blue data points at $f = 0.1$ is indicative of this gradient. Overall, it is clear that our giant component sizes agree well with those from an independent source. Hence, the growth model graphs used in the dynamics experiments match those used in [65] with respect to degree distribution, average degree and average clustering coefficient; moreover, the sizes of the giant components of these graphs agree well with those in [7].

6.3 Experimental Results on Contagion Dynamics

We begin with a brief description of our simulation methodology for studying contagion dynamics. For each graph instance (including frozen node type, fraction f of frozen nodes, and frozen node set in Table 6.1), 20 seed node sets are generated. Seed nodes, which are



(a) (b)

Figure 6.1: Normalized average giant component size GC_{ave} versus frozen node fraction f for graph classes (a) ED and (b) SF. In each plot, the curves represent data from [7]. There is one curve each for random and targeted frozen nodes representing one graph instance. Our data are represented by points. At each value of $f = 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 2 \times 10^{-1}$, we plot 30 data points for each of random and targeted frozen nodes, one for each graph instance. (The green points at smaller f do not appear in the plots because the values of GC_{ave} are near 1, and the points are hidden by the blue points.) Our results are in general agreement with those of [7].

in state 1 at the beginning of a diffusion instance, are chosen using the method of [65, 67]. First, one node v, called the **anchor** node, is selected uniformly at random from the nodes remaining after the frozen nodes (if any) are identified. Node v and all of its (distance-1) neighbors are the seed nodes. If a neighbor of v is a frozen node, then this candidate set of seed nodes is discarded to avoid conflicts (a node cannot be both a seed and frozen node), and a new node v is selected. Seed nodes are selected from the giant component of a graph (the component before the frozen nodes are introduced) so that the effects of frozen nodes can be evaluated. Thus, it is possible to select seed nodes that are part of a small component that is produced by removing frozen nodes, even though the seed nodes are part of the original giant component. As an aside, this is why the spread size for simple contagions need not be the same as the size of the giant component produced after deleting frozen nodes.

Contagion propagation is simulated on each graph as follows. All nodes are initially in state 0, except the seed nodes. Frozen nodes, if present, are fixed in state 0 for the entire diffusion process. All nodes are evaluated **synchronously** at each time step. That is, for each node u in state 0 at time t, the state of u at time $t + 1$ is 1 if at least θ neighbors are in state 1 at time t; otherwise, the node remains in state 0. A node in state 1 remains in state 1 for the duration of a diffusion instance and an instance ends when no node can transition to state 1. This approach produces the same spread size as the procedure used by [65]. In this manner, we generate 400 diffusion instances (20 frozen node sets by 20 seed node sets) for

each combination of `gm-ed` or `gm-sf` graph instance, frozen node type, fraction f of frozen nodes, and threshold, resulting in over 800,000 diffusion instances. The number of nodes in state 1 at the end of a diffusion instance, divided by the number of nodes in the original graph, is the **normalized spread size** S ($0 \leq S \leq 1$). The average normalized spread size over the 400 diffusion instances is denoted by S_{ave}. We plot S_{ave} versus f for both frozen node types and all graphs. Data for each of the 30 graphs are plotted separately to display variability. We select this representation in order to compare our results with those of [65].

Average spread size data for complex diffusion from [65] and from our growth model graphs `gm-ed` and `gm-sf` are plotted in Figure 6.2. Figure 6.2(a) provides ED data while Figure 6.2(b) depicts SF data. [3] Robustness for a given f is determined by comparing S_{ave} from the plots for different graphs. The graph with the greater S_{ave} for a given f is more robust. One could plot these data in different ways. We choose this approach because we want to emphasize the differences in results across studies.

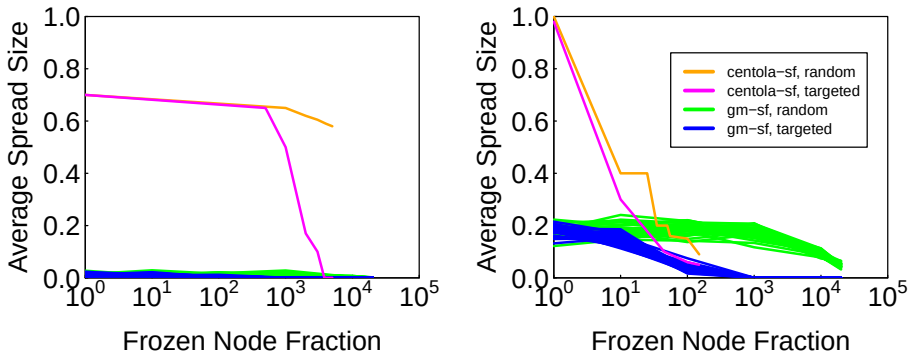
It is clear that our results are significantly different from those in [65]. For SF networks and $f = 0$, Centola's average spread sizes are a factor of 5 greater than ours. For ED networks over $0 \leq f \leq 0.01$, Centola's S_{ave} values are an order of magnitude greater than ours (see Finding 1a of Section 6.1.1). These differences in results provide concrete support for Contribution 1 (stated in Section 6.1.1).

A conclusion in [65] is that ED networks are more robust than SF networks for complex contagion, since for even one frozen node ($f = 10^{-4}$), the dashed curves in Figure 6.2(a) have greater ordinates than those in Figure 6.2(b). In contrast, our data, which show that SF networks have greater S_{ave} than ED networks for all values of f, suggest that SF networks are more robust (except when S_{ave} is close to 0 and the comparison is not meaningful); see Finding 1b (Section 6.1.1). For ED networks, our S_{ave} values are about 0.02 to 0.03; the values are not zero except at larger f.

We now evaluate the statistical significance of our results. Analysis of variance (ANOVA) [271] shows that graph class (`gm-ed` versus `gm-sf`) was significant in producing different S_{ave} for complex contagion over all other parameters ($p < 0.001$). Frozen node type, f value, and threshold value were significant in producing different S_{ave} ($p < 0.00001$).

We first give an intuitive explanation of our complex contagion results; a more detailed treatment is provided subsequently. These analyses support Finding 1c. Consider our results for complex contagion with zero frozen nodes, where we have $S_{ave} < 0.03$ for ED networks and $S_{ave} < 0.3$ in SF networks. For the SF networks and no frozen nodes, about 18% of nodes, per graph, have degree-1, and about 43% of nodes, per graph, have degree-2. See Figure C.3 for degree distributions. This means that 61% of nodes have degree-2 or less. Since the anchor node v of a seed set is determined uniformly at random, the odds favor selecting a small-degree node. If a degree-1 node is selected, then complex contagion propagation is not

[3] Each data point from [65] represents the average of 1000 simulation results. Each data point for our curves represents 400 simulation results; over 30 graphs, this is 12,000 data points per f value.



(a) (b)

Figure 6.2: Average spread size S_{ave} for complex contagion versus frozen node fraction f for graph classes (a) ED and (b) SF. In each plot, [65] data (red and orange lines) for random and targeted frozen nodes are plotted along with the data generated in this study (green and blue lines) for the growth model networks `gm-ed` and `gm-sf`. Since we generated 30 graph instances per class, we plot the results for each graph separately to convey variability. For the case of no frozen nodes ($f = 10^{-5}$ in the plots) the data show that Centola's average spread sizes are far greater than those generated herein, for both graph classes; i.e., Centola's graphs show more network robustness than do our graphs. This is also generally the case with frozen nodes (i.e., $f > 0$).

possible because only the anchor's single neighbor can be adjacent to a state 0 node, and therefore no nodes can transition to state 1. Similarly, for a degree-2 anchor node u with neighbors u_1 and u_2, the only way propagation can occur is when some node u_3 has both u_1 and u_2 as neighbors. Clearly, there will not always be such a u_3 when the average degree is only 4. Thus, even for the case of $f = 0$, we do not expect a large average spread size as indicated by our results. Similarly for ED networks, 13% and 28% of nodes, per graph, have respectively degree-1 and degree-2. This means that 41% of nodes have degree-2 or less. While this total is less than the 61% found for SF networks, meaning that there are more higher-degree nodes to select from as anchor nodes in ED graphs, we will show below that the greater maximum degrees in SF networks play a key role in fostering diffusion. With the compounding factor of frozen nodes, one would not expect spread sizes to increase as appreciable numbers of frozen nodes are introduced. The point is that our results of lower average spread sizes for both SF and ED networks for complex contagion make intuitive sense based solely on our graph structures.

We now provide a detailed explanation for the observed diffusion behaviors of ED and SF networks for complex contagion using a simple parameter that accounts for both graph structure and initial diffusion conditions. In particular, we are now evaluating S for each of the 288,000 complex contagion propagation instances, rather than S_{ave} over multiple diffusion instances per graph as displayed in Figure 6.2. Let n_e denote the total number of edges between seed nodes (in state 1) and nodes in state 0 at time 0 (i.e., before the diffusion process begins) for one diffusion instance. We also choose a normalized limiting spread size $S_l = 0.05$ so that a normalized spread size above (below) S_l is considered significant (small); this value is driven by the data in Figure 6.2. Intuitively, as n_e increases, one would expect a greater probability of generating a larger S. We seek a critical value of n_e, denoted by n_e^*, such that if $n_e < n_e^*$, the measured spread size S is small (i.e., $S < S_l$) and if $n_e \geq n_e^*$, S is significant (i.e., $S \geq S_l$). There are four cases where we use *positive* to indicate a prediction $S \geq S_l$ and *negative* to indicate a prediction $S < S_l$. For example, a false negative for a diffusion instance occurs when $n_e < n_e^*$, so we predict $S < S_l$, but we measure through simulation that $S \geq S_l$.

Figure 6.3 depicts the fraction of predictions that are true-positives, false-positives, true-negatives, and false-negatives for the complex contagion instances on the growth model graphs `gm-ed` and `gm-sf`. The abscissa is n_e, where nodes in state 0 are called **unaffected** nodes. We see from the figure that $n_e^* = 55$ identifies the intersection of true-positives and true-negatives at an ordinate value of 0.91. Hence, from this purely empirical approach, if $n_e < 55$ (≥ 55), then with 91% accuracy, the final spread size for a diffusion instance will be small (significant). Owing to the intricate nature of complex contagion diffusion, it is unclear whether other simple static measures can provide this level of accuracy. For example, if we use the total number of seed nodes as the measure to predict small and significant spread sizes, the accuracy is only 69%. More to the point, the chosen measure (n_e^*) explains our observed differences in behavior between ED and SF networks. Because the greatest degrees in SF networks are an order of magnitude greater than those in ED networks (see Figure

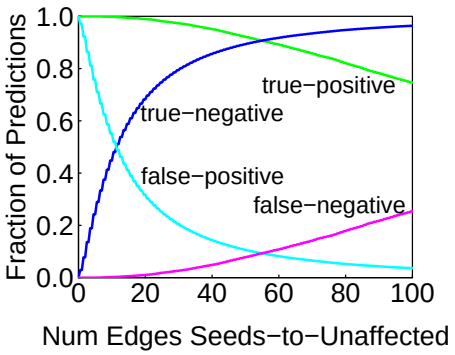


Figure 6.3: The fractions of true-positives and false-negatives for significant spread sizes (these sum to 1.0) and fractions of true-negatives and false-positives for small spread sizes (these sum to 1.0) for the n_e parameter in predicting small and significant spread sizes over 288,000 diffusion instances on all growth model graphs for complex contagion. These results show that a critical value of n_e, namely, $n_e^* = 55$, can predict with 91% accuracy whether a spread size will be significant or small.

C.3), larger values of n_e are more likely to be generated by SF networks. We have confirmed this by computing *all* possible seed sets in all growth model graphs with and without frozen nodes. These additional edges help propagate complex contagions.

We have the following additional points. First, the fractions of true-positive and true-negative (i.e., correct) predictions are each an order of magnitude greater than their incorrect counterparts. Second, 92% of measured spread sizes are small ($S < S_l$). This means that most of the SF diffusion instances do not produce appreciable spreading, consistent with Figure 6.2(b). In fact, detailed inspection of the data shows that $n_e \approx 500$ may result in little or no diffusion in SF networks. This is easy to see: the anchor seed node v can be a degree-1 node, whose only neighbor is a high-degree node w. Thus, w generates a large n_e, but each of these state-0 neighbors of w has only w in state 1, and hence no complex contagion propagation takes place.

Just as interesting as the differences in results across studies in Figure 6.2 are the reasons for them. The quote from [65] in Section 6.1.1 indicates that the reason ED networks are more robust is the homogeneous degrees of nodes in these networks. Our reason for SF networks being more robust is the opposite: the heterogeneous node degrees—and more precisely, the high degree nodes—of SF networks drive diffusion. That is, the number n_e of edges between seed nodes and their neighboring unaffected nodes increases by having high-degree nodes as seeds (Finding 1c).

There are other, more sophisticated, measures that may be employed, such as graph cuts at every discrete time during a simulation, to evaluate complex contagion in greater detail (n_e is the number of graph edges cut to isolate the seed nodes). However, our goal is to explain

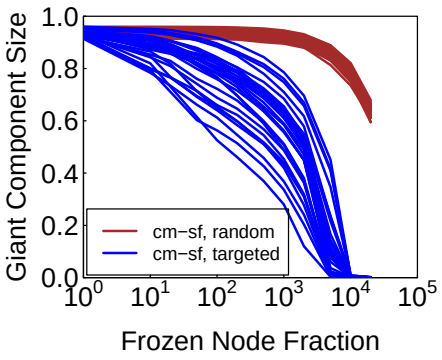


Figure 6.4: Average giant component size GC_{ave} as a function of frozen node fraction f. There are 30 curves, one for each graph instance for `cm-sf`. Each data point is the average of 20 giant component values, each corresponding to one frozen node set.

an observed behavior using an intuitive graph parameter which can be quickly computed. Further, it accounts for both graph structure and initial diffusion conditions; it also captures a large majority of the observed results.

We have demonstrated that by matching all graph characteristics specified in [65] and giant component sizes in [7], one can produce graphs that generate vastly different spread sizes, resulting in a different ranking of robustness compared to that obtained in [65]. One can naturally ask a more narrowly focused question: whether large variations in spread size can be produced across graphs from a *single* construction method. We demonstrate that this is possible, even when these graphs have the same degree distribution (and average degree) and average clustering coefficient. Our results focus on the effects of frozen nodes. For this, we use the configuration model SF graphs (`cm-sf`) and conditions of Table 6.1, as well as the same simulation steps defined above. [4]

Figure 6.4 provides average giant component size as a function of frozen node fraction for 30 graph instances. There are 12 values of f per graph and frozen node type, and each data point of each curve represents the average of 20 frozen node sets. These data thus represent averages over 14,400 different graphs. We focus on the data from targeted frozen nodes. We conduct Welch t-tests on each of the $\binom{30}{2} = 435$ pairs of targeted frozen node data. [5] Of these pairs, 361 (83%) are significantly different at the 95% confidence level ($p < 0.01$). [6] Thus, most of the GC_{ave} curves are significantly different, meaning the graphs themselves are significantly different. Therefore, averaging these data values into a single curve is of questionable value, and can be misleading.

[4] We actually use the six f values in Table 6.1 and six more values, namely $2\times10^{-4}, 5\times10^{-4}, 2\times10^{-3}, 5\times10^{-3}, 2\times10^{-2}$ and 5×10^{-2}.

[5] In these tests, each curve has twelve (f, GC_{ave}) data points.

[6] 214 of the 435 pairs (49%) are significant at $p < 0.001$.

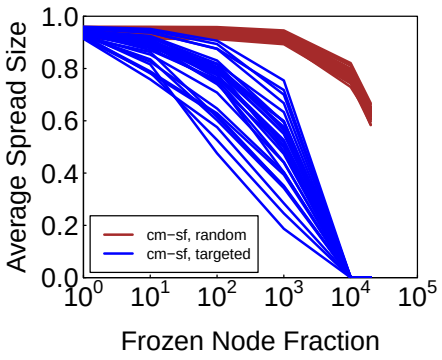


Figure 6.5: Average spread size S_{ave} for simple contagion as a function of frozen node fraction f for both random and targeted frozen nodes in 30 graph instances of the `cm-sf` graph class.

Figure 6.5 depicts the simple contagion results for the graphs of Figure 6.4, using only the f values in Table 6.1. It is evident that the giant component size is governing diffusion, with commensurate variability in S_{ave} across the 30 graphs. [7] [8] For example, at $f = 10^{-2}$ and targeted frozen nodes, S_{ave} ranges by more than a factor of 3, thus supporting Finding 2 (Section 6.1.1). Thus, a single construction method can produce graphs whose structure and dynamics can vary widely, rendering average behavior across network instances meaningless or misleading (Contribution 2).

It is important to recall that frozen nodes can equivalently be thought of as nodes removed from a graph or nodes whose state does not change. This is the case for the progressive model of Figure 1.5 and related models; e.g., [67, 166, 329]. The first interpretation affects graph structure while the second is a description of the dynamics model. Thus, while we often describe results in terms of connected components of a graph, our results do not pertain only to graphs for which a group of nodes are removed. Rather, they also apply to graphs whose structure is unaltered but whose dynamics involve frozen nodes. This is important for models of the kind considered in [35, 229] where nodes can transition back and forth between states. In these types of models, removing nodes and freezing their states are not equivalent. Our results are relevant for both types of models.

[7] The curves in Figure 6.5 look slightly different from those in Figure 6.4, particularly between $f = 10^{-2}$ and 10^{-1}, because the GC_{ave} curves include data at $f = 2 \times 10^{-2}$ and 5×10^{-2}.

[8] We also ran the complex contagion experiments with the `cm-sf` networks and confirmed that the model associated with the results in Figure 6.3 still holds for all growth and configuration model graphs combined; e.g., n_e^* changed from 55 to 54 and the prediction probability decreased slightly from 91% to 89%.

6.4 Summary

Many studies of contagion dynamics utilize SF and ED networks because they represent social networks found in practice. These graphs, which are characterized by a particular degree distribution, can be produced in many ways including growth models [25], configuration models [230, 238], and Chung-Lu models [82]. Yet, degree distribution alone does not define a graph [231]. Nonetheless, degree distribution is often used as a discriminator in evaluating dynamics of different classes of graphs.

We have shown that graphs within the same class of degree distribution (e.g., SF, ED), with the same average degree and average clustering coefficient can produce spread sizes that vary by a factor of ten from those in published works. These differences result in different robustness rankings of SF versus ED networks compared to previous work. In addition, graphs with the same degree distribution and average clustering coefficient, produced with the *same* construction method, can exhibit a factor of three variation in giant component and spread sizes with frozen nodes. In these cases, averaging results over multiple graph instances has little or no meaning, but yet is routinely done.

Chapter 7

Bithreshold Systems

7.1 Introduction

7.1.1 Background and Motivation

Social behaviors that are characterized by **back and forth** decision processes were described in Section 1.3.1. Briefly, in these systems, a node occupies one of two possible states (often labeled 0 and 1), but can change to the other state (and back again) with sufficient influence from its neighborhood. The ability for a node to repeatedly change between two states is fundamentally different from the progressive model of Chapters 5 and 6, where once a node reaches the affected state (typically state 1), it remains in that state.

We are interested in back and forth systems where each transition is controlled by a threshold. Thus, there are two thresholds, and we call the system a **bithreshold** system. That is, for a two-state $\{0, 1\}$ system where both up-transitions (i.e., $0 \to 1$) and down transitions (i.e., $1 \to 0$) are permitted, there is a threshold that governs the up-transition and a second, possibly distinct, threshold that governs the down-transition. The STM is given by Figure 1.3(e). Bithreshold systems capture such behaviors as starting and stopping dieting or excessive drinking. This model is fundamentally different from an SIS (susceptible-infectious-susceptible) epidemiological model because the transition from I back to S is typically based on time (i.e., there is an infectious duration). In the bithreshold social model, in contrast, a node may remain in either state indefinitely, and changes state when a sufficient driving force supplied by its neighborhood compels it to do so.

There are a few models beyond SIS that permit transitions back and forth between two states. In the voter model each agent changes state (to 0 or 1) by assuming the state of one randomly selected neighbor; see [111] and references therein. Thus, all neighbors of an agent are not utilized as in the bithreshold model. In majority models [105], an agent switches to the state of the majority of its neighbors. In contrast, state change in the bithreshold model

may be induced by more or less than one-half of its neighbors, so it is a generalization of a majority model. Perhaps the model closest to the bithreshold model is that of [35] because it has up and down threshold behavior. However, that uniform mixing model assumes a completely connected population network, where every agent influences every other agent in the same fashion, and every agent has complete knowledge of the system. The model makes no distinction among agents. Our model accounts for local structure within the population, where in general, each agent is only influenced by a subset of the population. Among other differences, our agents may have heterogeneous, history-dependent up-thresholds and down-thresholds. Our approach also incorporates the notion that a node may contribute unequal influence to each of its neighbors; this is an aspect of generalized contagion models [101].

In this chapter, we study a back and forth bithreshold systems. We study a number of formal problems that are grounded in GDS in order to understand system dynamics. We also study the blocking problem: how to stop the spread of a back and forth contagion from driving nodes to state 1. We do this by identifying critical or blocking nodes whose state is frozen at 0, thereby inhibiting the transition to state 1 and assisting nodes to transition to state 0. Investigations into the feasibility of using peer influence (which is captured by contagion processes) to dissuade undesirable behaviors is specifically called for in [206].

7.1.2 Summary of Contributions

In this summary, to keep it concise, we use forward references to some notions of discrete dynamical systems.

1. Computational complexity results for bithreshold system dynamics that, among other things, illustrate differences between simple and complex contagions. We show that the decision problem of determining whether a bithreshold system has a fixed point and its corresponding counting problem (i.e., determining the number of fixed points) can be solved efficiently for simple contagion bithreshold systems (i.e., simple bithreshold systems), but the respective problems when nodes can have thresholds of 2 are NP-complete and #P-complete. Additional results are provide for fixed point, as well as predecessor existence and reachability problems. All of these results can be found in Appendix D.

2. Complexity results for inhibiting diffusion of bithreshold contagions. We show that the minimum cost critical set problem is NP-hard for both simple and complex contagions. For simple contagions, we show that the problem can be approximated to within a factor of $O(\log n)$, where n is the number of nodes in the network and that no asymptotic improvement in performance guarantee is possible unless P = NP. For complex contagions, we show that a restricted version of the problem can be approximated to within a factor of $O(\log n)$. Even for this restricted version, we show that no asymptotic improvement in performance guarantee is possible unless P = NP.

3. A blocking algorithm to impeded or halt the spread of back and forth complex

contagions. Based on the approach used in the above approximation algorithms, we present a heuristic, called Maximum Contributor Heuristic (MCH), for finding critical nodes to prevent nodes from switching to a harmful or undesirable state 1. We show that MCH outperforms the High Degree Heuristic (i.e., setting highest degree nodes as critical) by multiple orders of magnitude. In Chapter 5, the HDH was shown to perform as well as several other state of the art methods in blocking complex contagions in progressive systems.

4. Critical evaluation of our blocking method. Over a large parameter input space, we show that MCH will halt diffusion and that the required number of critical nodes is dependent on network structure, thresholds, and seeding in complicated ways. For example, different graphs structural features can be important when the budget on blocking nodes is small versus large. Also, the number of blocking nodes required to halt transitions of nodes to state 1 can be a factor of 100 times greater than the number n_s of seed nodes, even for small seed sets. We evaluate both deterministic and stochastic diffusion, compare these and a hybrid method of blocking, and present limitations of MCH. We find over the parameter regime of this study, that the stochastic and hybrid blocking approaches perform comparably in halting stochastic diffusion, but the hybrid approach is faster to execute.

As this is the first work of its kind, we study fundamental behaviors to understand bithreshold systems and control of system dynamics under various conditions.

7.2 Bithreshold Synchronous Dynamical Systems

7.2.1 Basic Model Definitions

We model the propagation of contagions over a social network using a synchronous dynamical system (SyDS). A GDS was described in Section 1.5 and the specialization of GDS to SyDS was described in Section 5.2. We use here an SyDS and the same vertex state space $\mathbb{B}$ and the same synchronous update scheme as in Section 5.2. However, we now use a different local function f_i for nodes v_i ($1 \leq i \leq n$) that reflects bithreshold dynamics. Moreover, the bithreshold function studied here is different from the one defined in Section 1.5.

In this chapter, function f_i is a **bithreshold function**, characterized by two non-negative integer values denoted by $\theta_{\text{up},i}$ and $\theta_{\text{down},i}$. A precise definition of the function f_i is as follows.

(a) If the state of v_i is 0, then f_i is 1 if at least $\theta_{\text{up},i}$ of the neighbors of v_i are in state 1; otherwise, the value of f_i is 0.

(b) If the state of v_i is 1, then f_i is 0 if at least $\theta_{\text{down},i}$ of the neighbors of v_i are in state 0; otherwise, the value of f_i is 1.

Thus, $\theta_{\text{up},i}$, called the **up threshold** of v_i, represents the minimum number of neighbors of v_i that must be in state 1 for v_i to change from 0 to 1. Likewise, $\theta_{\text{down},i}$, called the **down**

threshold of v_i, represents the minimum number of neighbors of v_i that must be in state 0 for v_i to change from 1 to 0. A SyDS in which each node has a bithreshold transition function is called a **bithreshold** SyDS, denoted by BT-SyDS.

A single SyDS transition from one configuration to another can be expressed by the following pseudocode.

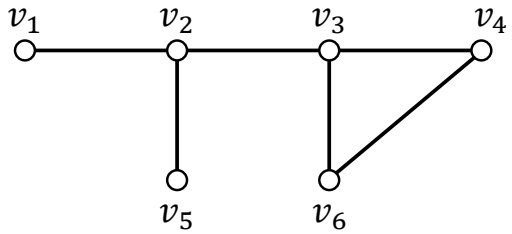
for each node v_i **do in parallel**

(i) Compute the value of f_i. Let s_i' denote this value.

(ii) Update the state of v_i to s_i'.

end for

Thus, in a SyDS, nodes update their state *synchronously*.



Initial Configuration:	(1,0,0,0,0,0)
Configuration at time 1:	(0,1,0,0,0,0)
Configuration at time 2:	(1,0,1,0,1,0)
Configuration at time 3:	(0,1,0,1,0,1)

Note: For each node, the up and down threshold values are 1. Each configuration has he form $(s_1, s_2, s_3, s_4, s_5, s_6)$, where s_i is the state of node v_i, $(1 \leq i \leq 6)$. From time step 3 onwards, the system cycles between the two configurations at times 3 and 2.

Figure 7.1: An example of a bithreshold synchronous dynamical system.

Example: Consider the BT-SyDS whose underlying graph is shown in Figure 7.1. Suppose for each node v_i, the function f_i is the bithreshold function with $\theta_{\text{up},i} = \theta_{\text{down},i} = 1$, $1 \leq i \leq 6$. In the initial configuration, v_1 is in state 1 and all other nodes are in state 0. During the first time step, the state of node v_1 changes to 0 (since v_1 has a neighbor whose state is 0) and that of v_2 changes to 1 (since v_2 has a neighbor whose state is 0); the other nodes remain in state 0. In the second time step, it can be verified that v_1 changes to 1, v_2 changes to 0, v_3 changes to 1, v_5 changes to 1 and the other two nodes remain in state 0. Continuing this process, it can be seen that from time step 3 onwards, the BT-SyDS cycles between the two configurations $(0, 1, 0, 1, 0, 1)$ and $(1, 0, 1, 0, 1, 0)$.

As will be shown later, any BT-SyDS in which all up and down threshold values are 1 either reaches a fixed point or cycles between two configurations after a number of steps that is bounded by the diameter[1] of the underlying graph.

7.2.2 Additional Definitions Related to the Model

We adopt the notions of simple and complex contagions per Section 1.4. A BT-SyDS in which $\theta_{\text{up},i} = \theta_{\text{down},i} = 1$ for each node v_i is called a **simple** BT-SyDS. If at least one of the threshold values in $\mathcal{S}$ is *greater than* 1, then $\mathcal{S}$ is referred to as a **complex** BT-SyDS.

If $\theta_{\text{up},i} = 0$ for some node v_i, then the state of v changes from 0 to 1 even when none of the neighbors of v_i is 1. We call such a node v_i an **uncontrolled up node**. Likewise, if $\theta_{\text{down},i} = 0$ for some node v_i, then node v_i will be referred to as an **uncontrolled down node**. Given a configuration $\mathcal{C}$, the state s_i of a given node v_i in $\mathcal{C}$ is denoted by $\mathcal{C}(v_i)$. In this chapter, we also use a slightly relaxed notion of fixed points, namely *pseudo* fixed points, as defined below.

Definition 27. *Given a SyDS $\mathcal{S}$, a configuration $\mathcal{C}$ of $\mathcal{S}$ is a **pseudo fixed point** if in the sequence of configurations obtained from $\mathcal{C}$, no node changes from 0 to 1.*

Thus, starting from a pseudo fixed point, nodes may change from 1 to 0; however, no node may change from 0 to 1. From the above definition, it can be seen that each fixed point is also a pseudo fixed point. As will be seen later, the problem of determining whether a given configuration is a pseudo fixed point can be solved efficiently for any BT-SyDS which does not have any uncontrolled up nodes. One can also define a pseudo fixed point from which the only allowed change is from 0 to 1. In the applications that motivated the BT-SyDS model, it is more natural to forbid 0 to 1 transitions rather than 1 to 0 transitions.

7.3 Minimum Cost Critical Set Problem

7.3.1 Problem Definition

While the above the problems studied in Appendix D deal directly with the phase space of a BT-SyDS, the problem of this section arises in the context of reconfiguring social networks through interventions. In particular, the problem deals with the conversion of a given configuration into a pseudo fixed point by modifying some nodes whose current state is 1. Here, "modifying" a node involves two things: (*i*) the state of the node is changed from 1 to 0 and (*ii*) the up threshold of the node is changed to a high value (e.g. degree of the node plus 1)

[1]The definition of diameter is given in Section D.1.2.

so that the node can never change to 1. We will say that such a node is **frozen at 0**. It should be noted that the modified node is *not* removed from the network; it remains in the network to influence neighboring nodes to change from 1 to 0 in subsequent time steps. The set of nodes chosen for modification is referred to as the **critical set**. It should be noted that only those nodes whose current state is 1 can be part of the critical set. In the social network context, modifying a node involves a cost. So, it is important to find a critical set of minimum cost. A precise formulation of the corresponding problem is as follows.

Minimum Cost Critical Set (MCCS)

Instance: A BT-SyDS $\mathcal{S}$, a configuration $\mathcal{C}$ of $\mathcal{S}$, for each node $v_i \in V$, the cost $c(v_i) \geq 0$ for modifying v_i and a budget β on the modification cost.

Question: Is there a critical set B such that the cost of B is at most β and B modifies $\mathcal{C}$ into a pseudo fixed point?

7.3.2 Preliminaries

The goal of the minimum cost critical set (MCCS) problem is to convert a given configuration into a pseudo fixed point by changing the state values of some nodes from 1 to 0 and making sure that those nodes are frozen at 0. Further, the cost of modification must be minimized. In this section, we show that the MCCS problem remains NP-complete for simple and complex BT-SyDSs. For simple BT-SyDSs, we show that unless P= NP, no polynomial time algorithm can provide a performance guarantee of $o(\log n)$, and we present an approximation algorithm with a performance guarantee of $O(\log n)$. For complex BT-SyDSs, we show that unless P= NP, no polynomial time algorithm can provide a performance guarantee of $o(\log n)$, even for a special case of the problem. For that special case, we present an approximation algorithm with a performance guarantee of $O(\log n)$.

Throughout Section 7.3, we make two assumptions. The first assumption is that the underlying graph of the given BT-SyDS is connected. (Otherwise, each connected component can be considered separately.) The second assumption is that the given BT-SyDS does not have any node whose up threshold is 0. The reason is that such uncontrolled up nodes can change from 0 to 1 regardless of the state values of their neighbors. We begin with a lemma which will be used throughout Section 7.3.

Lemma 28. *Let $\mathcal{S}$ be a BT-SyDS where the up threshold of each node is at least 1. Let $\mathcal{C}$ be a given configuration. The problem of determining whether $\mathcal{C}$ is a pseudo fixed point can be solved in polynomial time.*

Proof: Let n denote the number of nodes of $\mathcal{S}$. Consider running $\mathcal{S}$ with $\mathcal{C}$ as the initial configuration. If $\mathcal{C}$ is a pseudo fixed point, then the system must reach a fixed point after

at most n transitions, since the number of 1's in $\mathcal{C}$ that can change to 0 is at most n. We may need one more step to verify that a configuration is a fixed point. Thus, the algorithm to determine whether $\mathcal{C}$ is a pseudo fixed point is the following.

1. Run the BT-SyDS $\mathcal{S}$ starting from $\mathcal{C}$ until it reaches a fixed point or one of the transitions causes a node to change from 0 to 1. (As mentioned above, this needs at most $n+1$ transitions.)
2. During the execution of Step 1, if some node changes from 0 to 1, then $\mathcal{C}$ is not a pseudo fixed point; otherwise, $\mathcal{C}$ is a pseudo fixed point.

Since the node functions can be evaluated in polynomial time, the overall time to run $\mathcal{S}$ for $n+1$ steps is a polynomial. □

7.3.3 Critical Set Problem for Simple BT-SyDSs

Throughout this section, we will assume that in a simple BT-SyDS, the up and down thresholds for each node are 1. We begin with some observations concerning pseudo fixed points for such systems.

By definition, for any BT-SyDS, every fixed point is also a pseudo fixed point. The following lemma points out that for simple BT-SyDSs, every pseudo fixed point is also a fixed point. Thus, for simple BT-SyDSs, the notions of "pseudo fixed points" and "fixed points" coincide.

Lemma 29. *Let $\mathcal{S}$ be a simple BT-SyDS (i.e., all threshold values are 1). Every pseudo fixed point of $\mathcal{S}$ is also a fixed point.*

Proof: Suppose $\mathcal{C}$ is a pseudo fixed point. Assume for the sake of contradiction that $\mathcal{C}$ is not a fixed point. Let $\mathcal{C}_1$ be the successor of $\mathcal{C}$. Since $\mathcal{C}$ is a pseudo fixed point, no node changes from 0 to 1 during the transition from $\mathcal{C}$ to $\mathcal{C}_1$. However, since $\mathcal{C}$ is not a fixed point, some node v which was 1 in $\mathcal{C}$ must have changed to 0 in $\mathcal{C}_1$. Since the down threshold of v is 1, v must have a neighbor u such that $\mathcal{C}(u) = 0$. Now, since the up threshold of u is 1 and u has a neighbor v with value 1, u will change to 1 during the transition from $\mathcal{C}$ to $\mathcal{C}_1$. This contradicts the assumption that $\mathcal{C}$ is a pseudo fixed point. In other words, $\mathcal{C}$ is indeed a fixed point of $\mathcal{S}$. □

Before the next lemma, we introduce some terminology.

Definition 30. *Suppose $\mathcal{S}$ is a simple BT-SyDS. Suppose a nonempty subset $F \subseteq V$ of the nodes $\mathcal{S}$ is frozen at 0. Let $\mathcal{C}$ be a configuration of $\mathcal{S}$ such that for each node v in F, the state of v is 0 in $\mathcal{C}$.*

*(a) Each node $u \in F$ is a **frozen node** with respect to $\mathcal{C}$.*

(b) Each node v in $V - F$ whose state is 0 in $\mathcal{C}$ is a ***normal 0-node*** *with respect to $\mathcal{C}$.*

(c) Each node w in $V - F$ whose state is 1 in $\mathcal{C}$ is a ***normal 1-node*** *with respect to $\mathcal{C}$.*

When the configuration $\mathcal{C}$ is clear from the context, we will use the above terminology without the qualifier "with respect to $\mathcal{C}$".

The following lemma characterizes the configurations that are pseudo fixed points for simple BT-SyDSs.

Lemma 31. *Let $\mathcal{S}$ be a simple BT-SyDS. Suppose a nonempty subset $F \subseteq V$ of the nodes $\mathcal{S}$ is frozen at 0. Let $\mathcal{C}$ be a configuration of $\mathcal{S}$ such that for each node v in F, the state of v is 0 in $\mathcal{C}$. Then, $\mathcal{C}$ is a pseudo fixed point if and only if the following two conditions hold.*

1. *No normal 0-node is adjacent to a normal 1-node.*
2. *Each normal 1-node is adjacent to at least one frozen node.*

Proof:

If part: Suppose the two conditions hold. To prove that $\mathcal{C}$ is a pseudo fixed point, we will show that the successor of $\mathcal{C}$ is the configuration in which all nodes are in state 0. (Since all the up thresholds are all 1, the configuration of all 0's is a fixed point.)

By definition, each frozen node remains in state 0 in all future transitions. Consider any normal 0-node v. By Condition 1 of the lemma, no neighbor of v is in state 1 in $\mathcal{C}$. Since the up threshold of v is 1, the state of v cannot change to 1 during the transition from $\mathcal{C}$. Consider any normal 1-node w. By Condition 2 of the lemma, at least one neighbor of w is a frozen node (which is in state 0). Since the down threshold of w is 1, w will change to 0 in the transition from $\mathcal{C}$. Thus, all the nodes reach the state 0 in the successor of $\mathcal{C}$. In other words, $\mathcal{C}$ is a pseudo fixed point.

Only If part: Suppose $\mathcal{C}$ is a pseudo fixed point. Thus, in the sequence of transitions from $\mathcal{C}$, no node can change from 0 to 1. We now prove that the two conditions of the lemma hold.

To prove Condition 1 of the lemma, suppose there is a normal 0-node v that is adjacent to a normal 1-node. Since the up threshold of v is 1, v will change to 1 during the transition from $\mathcal{C}$, contradicting the assumption that $\mathcal{C}$ is a pseudo fixed point. Thus, Condition 1 must hold.

We now prove Condition 2 of the lemmas. For the sake of contradiction. suppose there is a normal 1-node w that is not adjacent to any frozen node. Since F is nonempty and the underlying graph G is connected, there is a path in G from w to each of the frozen nodes.

Among all such paths, consider a *shortest* path (i.e., a path with the smallest number of edges) from w to a frozen node, say z. Let $P = \langle w, w_1, w_2, \ldots, w_r, z\rangle$ denote this path. Since w is not adjacent to z, nodes w and w_1 are distinct.

Claim 1: Each of the nodes w_i, $1 \leq i \leq r$, is a normal 1-node in $\mathcal{C}$.

Proof of Claim 1: We first note that node w_i $(1 \leq i \leq r)$ cannot be a frozen node since P is a shortest path from w to a frozen node. Suppose some node w_j along P is a normal 0-node. Observe that $j \neq 1$ since w_1 is a normal 1-node. Thus, P contains a normal 1-node (namely, w_1) and a normal 0-node (namely, w_j). Hence, when we traverse P starting from w_1, at some point we must come across two successive nodes w_p and w_{p+1} such that w_p is a normal 1-node and w_{p+1} is a normal 0-node. Now, consider the transition from $\mathcal{C}$. Since w_{p+1} is a normal 0-node, it is adjacent to a normal 1-node (w_p) and the up threshold of w_{p+1} is 1, node w_{p+1} will change from 0 to 1 during the transition. This contradicts the assumption that $\mathcal{C}$ is a pseudo fixed point and establishes the claim.

We now continue with the proof of Condition 2. In view of Claim 1, consider the last two nodes w_{r-1} and w_r of the path P. Since we have already shown that Condition 1 of the lemma holds, neither w_{r-1} nor w_r (which are normal 1-nodes) can be adjacent to any normal 0-nodes. Thus, in particular, all neighbors of w_{r-1} in $\mathcal{C}$ must be normal 1-nodes, and all all neighbors of w_r, except for z, are normal 1-nodes in $\mathcal{C}$. Now, during the transition from $\mathcal{C}$, w_r changes to 0 (since it is adjacent to the frozen node z) while w_{r-1} remains at 1 (since all its neighbors are normal 1-nodes). Let $\mathcal{C}'$ be the successor configuration of $\mathcal{C}$. In $\mathcal{C}'$, the states of w_{r-1} and w_r are 1 and 0 respectively and the up threshold of w_r is 1. Therefore, during the transition from $\mathcal{C}'$, the state of w_r will change from 0 to 1, again contradicting the assumption that $\mathcal{C}$ is a pseudo fixed point. This completes the proof of Condition 2 as well as that of the lemma. □

Note that for simple BT-SyDSs, Lemma 31 leads to an alternative algorithm to determine whether a given configuration is a pseudo fixed point. We now present results that address the complexity and approximability of the MCCS problem for simple BT-SyDSs.

Theorem 32. *(a) The* MCCS *problem is* NP*-complete for simple BT-SyDSs. (b) There is a constant $c' > 0$ such that the* MCCS *problem for simple BT-SyDSs with n nodes cannot be approximated to within a factor less than $c' \ln n$ in polynomial time, unless* $\mathsf{P} = \mathsf{NP}$.

Proof of Part (a): It is easy to see that the MCCS problem for BT-SyDSs is in NP. We prove NP-hardness by a reduction from the Minimum Dominating Set (MDS) problem (see Section D.1.3 for a definition).

Let an instance of the MDS problem be given by the undirected graph $G(V, E)$ and integer K. We construct an instance of the MCCS problem for simple BT-SyDS $\mathcal{S}$ as follows. The underlying graph $G'(V', E')$ of $\mathcal{S}$ consists of a copy of G plus three new nodes denoted by x, y and z. There is an edge joining node x to one node, say v_1, of V. Two other edges

$\{x, y\}$ and $\{y, z\}$ are also added. So, E' consists of all the edges of E plus the three new edges added. The up and down threshold values for all the nodes in V' are 1. In the chosen configuration $\mathcal{C}$ (which needs to be converted into a pseudo fixed point), node z is in state 0 and all the other nodes are in state 1. The modification cost for each node of G' is chosen as 1. This completes the construction of the MCCS instance, and it is easy to see that the construction can be done in polynomial time. We now prove that the graph G has a dominating set of size at most K if and only if the resulting MCCS instance has a solution of cost at most $K + 1$.

Suppose G has a dominating set V' of size at most K. The set F of frozen nodes for $\mathcal{S}$ is given by $F = V' \cup \{y\}$. Thus, in the resulting configuration $\mathcal{C}'$, the states of node z and all the nodes in F are 0; the states of the other nodes are 1. Note that the modification cost is at most $K + 1$. We argue that $\mathcal{C}'$ is a pseudo fixed point by showing that it satisfies the two conditions of Lemma 31. First, note that z is the only normal 0-node of $\mathcal{C}'$, and its only neighbor y is a frozen node. Now consider the normal 1-nodes of $\mathcal{C}'$. We need to show that each such node is adjacent to at least one frozen node. The normal 1-node x is adjacent to the frozen node y. The other normal 1-nodes are the nodes in $V - V'$. Consider any such node $v \in V - V'$. Since V' is a dominating set for G, v is adjacent to some node in V'. Since each node in V' is also a frozen node, v is adjacent to at least one frozen node. Thus, both conditions of Lemma 31 are satisfied, and $\mathcal{C}'$ is a pseudo fixed point.

For the converse, suppose there is a solution to the MCCS instance with cost at most $K+1$. Let F be the set of frozen nodes in the solution and let $\mathcal{C}'$ be the resulting pseudo fixed point. Thus, $|F| \leq K + 1$. We observe that F must contain the node y; otherwise, the normal 0-node z will be adjacent to a normal 1-node (y), and $\mathcal{C}'$ cannot be a pseudo fixed point (by Lemma 31). Thus, F contains at most K nodes from V. We have two cases to consider.

Case 1: F does not contain node x.

In this case, we claim that $V' = F - \{y\}$ is a dominating set of size at most K for G. To prove this, we first note that $V' \subseteq V$ and $|V'| \leq K$. Also, each node of $V - V'$ is a normal 1-node with respect to the pseudo fixed point $\mathcal{C}'$. Thus, by Lemma 31, each node in $V - V'$ must be adjacent to some node in F. None of the nodes of V is adjacent to y, and F does not include x. Thus, each node of $V - V'$ must be adjacent to some node in $V' = F - \{y\}$. In other words, V' is a dominating set of size at most K for G.

Case 2: F contains node x.

In this case, let $V' = (F - \{x, y\}) \cup \{v_1\}$, where v_1 is the node of V adjacent to x in G'. Here also, $V' \subseteq V$ and $|V'| \leq K$. To see that V' is a dominating set, consider any node $v_i \in V - V'$. Note that $i \neq 1$ since $v_1 \in V'$. Since v_i is a normal 1-node with respect to $\mathcal{C}'$, by Lemma 31, it must be adjacent to some node in F. However, since $i \neq 1$, in G', v_i is not adjacent to either x or y. In other words, v_i is adjacent to some node in $F - \{x, y\} \subseteq V'$. Thus, V' is a dominating set for G and this completes the NP-hardness proof.

Proof of Part (b): We use the following result from [272]: There is a constant $c > 0$ such that the MDS problem cannot be approximated to within a factor less than $c \ln n$ for n node graphs, unless P= NP. We will show that if there is an approximation algorithm for the MCCS problem with a performance guarantee less than $(c/4) \ln n$, then there is an approximation algorithm for the MDS problem with a performance guarantee less than $c \ln n$.

Suppose there is an approximation algorithm $\mathcal{A}$ for the MCCS problem with a performance guarantee less than $(c/4) \ln N$ for any graph with N nodes. Consider an instance of the MDS optimization problem consisting of a graph G with n nodes. Let $\text{OPT}_{DS} \geq 1$ denote the size of an optimal dominating set for G. We use the construction presented in the proof of Part (a) to construct from G a simple BT-SyDS $\mathcal{S}$ with underlying graph G' and the configuration $\mathcal{C}$. Let OPT_{CS} denote the cost of the optimal solution for the MCCS instance for $\mathcal{S}$. From the reduction presented in the proof of Part (a), we note that

$$\text{OPT}_{CS} \ \leq \ \text{OPT}_{DS} + 1. \tag{7.1}$$

We run $\mathcal{A}$ on the resulting instance of the MCCS problem and let $\text{COST}(\mathcal{A})$ denote the cost of the approximate solution produced by $\mathcal{A}$. Since G' has $N = n + 3$ nodes and $\mathcal{A}$ has a performance guarantee of less than $(c/4) \ln N$, we have

$$\text{COST}(\mathcal{A}) \ < \ (c/4) \ln (n + 3) \, \text{OPT}_{CS}. \tag{7.2}$$

For large enough n, we have $\ln (n + 3) < 2 \ln n$. Therefore, from Equation (7.2),

$$\text{COST}(\mathcal{A}) \ < \ (c/2) \ln n \, \text{OPT}_{CS}. \tag{7.3}$$

Using the construction presented in the proof of Part (a), we can produce a dominating set of size $\text{COST}(\mathcal{A}) - 1 \ < \ \text{COST}(\mathcal{A})$ for G. Letting HEU_{DS} denote the size the resulting approximate dominating set for G, we have

$$\begin{array}{rcll} \text{HEU}_{DS} & < & \text{COST}(\mathcal{A}) & \\ & < & (c/2) \ln n \, \text{OPT}_{CS} & \text{(From Equation (7.3))} \\ & \leq & (c/2) \ln n \, (\text{OPT}_{DS} + 1) & \text{(From Equation (7.1))} \\ & \leq & (c/2) 2 \ln n \, \text{OPT}_{DS} & \text{(since } \text{OPT}_{DS} \geq 1\text{)} \\ & = & c \ln n \, \text{OPT}_{DS}. & \end{array}$$

Thus, we have constructed a polynomial time approximation algorithm with a performance guarantee less than $c \ln n$ for the MDS optimization problem. This in conjunction with the result of [272] mentioned above completes the proof of Part (b). □

We now present an approximation algorithm that provides a performance guarantee of $O(\log n)$ for the MCCS problem for simple BT-SyDSs. Given a simple BT-SyDS $\mathcal{S}$ with underlying graph $G(V, E)$ and a configuration $\mathcal{C}$, this approximation algorithm (called Approx-Simple-BT) converts $\mathcal{C}$ into a pseudo fixed point $\mathcal{C}'$. The algorithm relies on the known

Input: A simple BT-SyDS $\mathcal{S}$ with underlying graph $G(V, E)$, a configuration $\mathcal{C}$ and the cost $c(v)$ for freezing each node v whose state is 1 in $\mathcal{C}$.

Output: A set $F \subseteq V$ of frozen nodes and a new configuration $\mathcal{C}'$ which is a pseudo fixed point.

Steps of the Algorithm:

1. Let $F_1 = F_2 = \varnothing$. (At the end, the set F of frozen nodes is given by $F = F_1 \cup F_2$.)
2. **if** $\mathcal{C}$ is the all 0's configuration or the all 1's configuration, **then** output $F = \varnothing$ and $\mathcal{C}' = \mathcal{C}$ and **stop**.
3. **for** each normal 0-node v in $\mathcal{C}$ **do**
   (a) Find the set $S_v = \{w : \{v, w\} \in E$ and w is a normal 1-node in $\mathcal{C}\}$.
   (b) $F_1 = F_1 \cup S_v$.
4. Construct an instance of the Minimum Set Cover (MSC) problem as follows.
   (a) The set U of elements to be covered is defined by
   $$U = \{u : u \text{ is a normal 1-node in } V - F_1 \text{ and } u \text{ is not adjacent to a node in } F_1\}.$$
   (b) For each normal 1-node w in $V - F_1$, create a set S_w as follows:
   $$S_w = \{w\} \cup \{u : u \in U \text{ and } \{u, w\} \text{ is in } E\}.$$
   The cost of S_w is the modification cost $c(w)$ of node w.
5. Using the known $O(\log n)$ approximation algorithm for MSC, construct an approximate solution to the set cover instance constructed in Step 4. Let S_{w_1}, S_{w_2}, ..., S_{w_r} denote the sets in this approximate set cover.
6. $F_2 = \{w_1, w_2, \ldots, w_r\}$.
7. Let $F = F_1 \cup F_2$. Construct $\mathcal{C}'$ from $\mathcal{C}$ by freezing all the nodes in F to 0. (For each node v in $V - F$, its state value in $\mathcal{C}'$ is the same as that in $\mathcal{C}$.)
8. Output the set F and the new configuration $\mathcal{C}'$.

Figure 7.2: Steps of Algorithm Approx-Simple-BT.

$O(\log n)$ approximation for the Minimum Set Cover (MSC) problem (see Section D.1.3 for a definition).

The steps of the algorithm are shown in Figure 7.2. We note that Step 2 of the algorithm checks whether the given configuration $\mathcal{C}$ is a configuration of all 0's or all 1's since those are the only fixed points for any simple BT-SyDS whose underlying graph is connected. When $\mathcal{C}$ is a fixed point, there is no need to freeze any nodes and the algorithm returns $\mathcal{C}$ itself. Otherwise, the algorithm freezes nodes in two stages. In the first stage (Step 3 of the algorithm), every normal 1-node that is adjacent to a normal 0-node is added to F_1. This is necessary by Condition 1 of Lemma 31. In the second stage (Steps 4 through 6 of the algorithm), the subset F_2 of frozen nodes is chosen so as to satisfy Condition 2 of Lemma 31. That is, for each normal 1-node v which does is not adjacent to at least one frozen node in F_1, either v or a normal 1-node adjacent to v is added to F_2. This is done by using an approximation algorithm for the Minimum Set Cover (MSC) problem so that the cost of the nodes chosen in F_2 can be suitably bounded.

We now prove the correctness and performance guarantee of the algorithm.

Theorem 33. *(a) The configuration $\mathcal{C}'$ produced by Algorithm Approx-Simple-BT (Figure 7.2) is a pseudo fixed point. (b) The modification cost of the algorithm is within a factor $O(\log n)$ of the optimal modification cost, where n is the number of nodes in the underlying graph.*

Proof: We prove each part separately. Throughout the proof, we refer to the steps of the algorithm shown in Figure 7.2.

Proof of Part (a): When the input configuration $\mathcal{C}$ is the all 0's configuration or the all 1's configuration, it is a fixed point for $\mathcal{S}$ and the algorithm correctly returns $\mathcal{C}$ itself as the output. Thus, we need to consider only the case where $\mathcal{C}$ has at least one node in state 0 and at least one in state 1. Thus, $\mathcal{C}$ is not a pseudo fixed point, and one or more nodes must be frozen to create a pseudo fixed point. We will prove that the configuration $\mathcal{C}'$ output by the algorithm is a pseudo fixed point by showing that it satisfies both the conditions of Lemma 31.

To prove Condition 1 of Lemma 31, consider any normal 0-node v in $\mathcal{C}'$. We need to show that v is not adjacent to any normal 1-node in $\mathcal{C}'$. Since we only freeze nodes which are normal 1-nodes in $\mathcal{C}$, v must also be a normal 0-node in $\mathcal{C}$. Step 3 of the algorithm ensures that for each normal 0-node v in $\mathcal{C}$, if v is adjacent to a normal 1-node w in $\mathcal{C}$, then w is chosen as a frozen node (i.e., added to F_1). Therefore, $\mathcal{C}'$ satisfies Condition 1 of the lemma.

To prove Condition 2 of Lemma 31, consider any normal 1-node v in $\mathcal{C}'$. (Thus, v is not a frozen node.) We need to show that v is adjacent to at least one frozen node. If v is not adjacent to a frozen node at the end of Step 3 of the algorithm, then v becomes a member of the set U of elements to be covered in Step 4. The approximate set cover chosen in Step 5

ensures that v is covered by some set S_{w_j} that corresponds to the frozen node w_j in F_2. In other words, in the solution produced by the algorithm, v is adjacent to the frozen node w_j. Therefore, $\mathcal{C}'$ also satisfies Condition 2 of the lemma and hence is a pseudo fixed point. This completes the proof of correctness of the algorithm.

Proof of Part (b): We prove that the algorithm provides a performance guarantee of $O(\log n)$. Let F^* denote the set of frozen nodes in an optimal solution and let $\mathcal{C}^*$ denote the corresponding pseudo fixed point. Thus, the cost of the optimal solution, denoted by $\text{COST}(F^*)$, is the sum of the modification costs of the nodes in F^*. The set F of frozen nodes chosen by the approximation algorithm is the union of two disjoint subsets F_1 (constructed in Step 3) and F_2 (constructed in Step 6). The cost of the solution produced by the approximation algorithm, denoted by $\text{COST}(F)$, is the sum of the modification costs of the nodes in F. Since $F = F_1 \cup F_2$, we have,

$$\text{COST}(F) \;=\; \text{COST}(F_1) \;+\; \text{COST}(F_2). \tag{7.4}$$

We will bound $\text{COST}(F_1)$ and $\text{COST}(F_2)$ separately and combine the bounds using Equation (7.4) to establish the performance guarantee provided by the algorithm. We do this through a sequence of claims.

Claim 1: $\text{COST}(F_1) \leq \text{COST}(F^*)$.

Proof of Claim 1: We prove the claim by showing that $F_1 \subseteq F^*$; that is, every node chosen in F_1 should also be chosen in the optimal solution F^*. To see this, suppose v is node in F_1 but not in F^*. Thus, v is a normal 1-node that is adjacent to some normal 0-node w in the configuration $\mathcal{C}$. If v is not frozen, then since the up threshold of w is 1, w will change to 1 in the transition from the configuration $\mathcal{C}^*$ produced by the optimal solution. This contradicts the assumption that $\mathcal{C}^*$ is a pseudo fixed point and completes the proof of Claim 1.

Claim 2: The set $F^* - F_1$ is a solution to the MSC instance constructed in Step 4 of the algorithm.

Proof of Claim 2: We prove the claim by contradiction. Suppose $F^* - F_1$ is not a solution to the MSC instance. Thus, there is some node u such that u is a normal 1-node in $\mathcal{C}$, u is not in $F^* - F_1$ and u is not adjacent to any node in $F^* - F_1$. Note that u is not adjacent to any node in F_1. Therefore, u is not adjacent to any node in F^*. Thus, u is also a normal 1-node in $\mathcal{C}^*$ and u is not adjacent to any frozen node in F^*. This violates Condition 2 of Lemma 31, and leads to the conclusion that $\mathcal{C}^*$ is not pseudo fixed point. This contradiction completes the proof of Claim 2.

Claim 3: $\text{COST}(F_2) \leq O(\log n)\,\text{COST}(F^*)$.

Proof of Claim 3: By Claim 2, $F^* - F_1$ is a solution to the MSC instance constructed in Step 4 of the algorithm. Thus, the cost of an optimal solution to the MSC instance is at

most $\text{COST}(F^* - F_1) \leq \text{COST}(F^*)$. Step 5 constructs an approximate solution whose cost is within a factor $O(\log n)$ of the optimal cost of the MSC instance. Claim 3 follows.

We can now bound $\text{COST}(F)$ as follows.

$$\begin{array}{rcll} \text{COST}(F) & = & \text{COST}(F_1) \ + \ \text{COST}(F_2) & \text{(Equation(7.4)} \\ & \leq & \text{COST}(F^*) \ + \ O(\log n)\,\text{COST}(F^*) & \text{(By Claims 1 and 3)} \\ & = & O(\log n)\,\text{COST}(F^*). & \end{array}$$

This completes the proof of the theorem. □

7.3.4 Hardness and Approximation Results for Complex BT-SyDSs

We now show that the MCCS problem is computationally intractable for complex BT-SyDSs. In fact, this result holds for BT-SyDSs where the maximum threshold value is 2 and all nodes have the same modification cost, as shown below.

Theorem 34. *The* MCCS *problem is* NP*-complete for BT-SyDSs in which the maximum threshold value is 2 and all nodes have the same modification cost.*

Proof: We begin by showing that the MCCS problem is in NP. Let $\mathcal{S}$ be the given BT-SyDS and let $\mathcal{C}$ be the given configuration. We first guess the set V' of nodes to be modified and obtain the configuration $\mathcal{C}'$ from $\mathcal{C}$ by changing the states of nodes in V' to 0 (and freezing those nodes at 0). It is easy to verify that the cost of all the nodes in V' is at most the given cost bound. By Lemma 28, we can efficiently verify that configuration $\mathcal{C}'$ is a pseudo fixed point. This establishes the membership of MCCS in NP.

To prove NP-hardness, we use a reduction from the Minimum Vertex Cover (MVC) problem. Let the given instance I of the MVC problem consist of a connected graph $G_1(V_1, E_1)$ and integer K. Let $V_1 = \{v_1, v_2, \ldots, v_n\}$ and $E_1 = \{e_1, e_2, \ldots, e_m\}$. We construct a BT-SyDS $\mathcal{S}$ as follows. The underlying graph $G(V, E)$ of $\mathcal{S}$ has one node p_i for each node v_i of G_1 $(1 \leq i \leq n)$ and one node q_j for each edge e_j of G_1 $(1 \leq j \leq m)$. Let $P = \{p_1, p_2, \ldots, p_n\}$ and $Q = \{q_1, q_2, \ldots, q_m\}$. If edge e_j joins nodes v_x and v_y in G_1, then node q_j is adjacent to nodes p_x and p_y in G. The up and down thresholds for each node p_i are 1. The up threshold for each node q_j is 2 and the down threshold is 1. The cost of modifying any node is chosen as 1. In the chosen configuration $\mathcal{C}$, each node p_i is set to 1 and each node q_j is set to 0. The upper bound on the cost of modification is set to K. This completes the construction of the MCCS instance. Obviously, the construction can be carried out in polynomial time.

Suppose G_1 has a vertex cover V' of size at most K. The modified configuration $\mathcal{C}_1$ for the MCCS instance is obtained as follows: for each node $v_x \in V'$, choose p_x as a frozen node. Other nodes have the same state value in $\mathcal{C}$ and $\mathcal{C}_1$. Since $|V'| \leq K$, the modification cost

is at most K. To show that $\mathcal{C}_1$ is a pseudo fixed point, we argue that the successor of $\mathcal{C}_1$ is the configuration consisting of all 0's. Since the modified nodes are frozen at 0, they cannot change to 1. Consider any node p_i which was not frozen. By our construction, all neighbors of p_i are nodes in Q, whose state value is 0. Since the down threshold of p_i is 1, p_i will change to 0 during the transition from $\mathcal{C}_1$. Now, consider any node $q_j \in Q$. Let $e_j = \{v_x, v_y\}$ be the edge in G_1 corresponding to node q_j of $\mathcal{S}$. Since V' is a vertex cover for G_1, V' contains at least one of the nodes v_x and v_y. Therefore, at least one of p_x and p_y has been frozen at 0. Since the degree and up threshold of q_j are both 2 and at least one neighbor of q_j is frozen at 0, q_j remains in state 0 during the transition from $\mathcal{C}_1$. Thus, all nodes are in state 0 in the successor of $\mathcal{C}_1$. In other words, $\mathcal{C}_1$ is indeed a pseudo fixed point. Therefore, the MCCS instance has a solution.

Now, suppose the MCCS instance has a solution. Let $\mathcal{C}_1$ denote the pseudo fixed point constructed. Define the set $V' \subseteq V_1$ as follows:

$$V' = \{v_x \ : \ p_x \text{ is a frozen node}\}.$$

Since the only nodes which are 1 in $\mathcal{C}$ are nodes p_i that correspond to the nodes of G_1, only those nodes can be frozen in constructing $\mathcal{C}_1$. Further, since the cost of modification is at most K, we have $|V'| \leq K$. We show by contradiction that V' is a vertex cover for G_1. Suppose for some edge $e_j = \{v_x, v_y\}$, neither v_x nor v_y appears in V'. Thus, neither p_x nor p_y was frozen; that is, $\mathcal{C}_1(p_x) = \mathcal{C}_1(p_y) = 1$. Now, in $\mathcal{C}_1$, node q_j of $\mathcal{S}$ (corresponding to the edge e_j of G_1) has value 0 but has two neighbors (namely p_x and p_y) that have value 1 in $\mathcal{C}_1$. Since the up threshold of q_j is 2, q_j will change from 0 to 1 in the transition from $\mathcal{C}_1$. This contradicts the assumption that $\mathcal{C}_1$ is a pseudo fixed point. In other words, V' is a vertex cover for G_1, and this completes the proof. ■

We presented the above reduction to the MCCS problem from MVC to show that the former problem is NP-complete even for BT-SyDSs with a maximum threshold value of 2. One can carry out a similar reduction from the Minimum Set Cover (MSC) problem to the MCCS problem. In such a reduction, sets and elements of MSC play the roles of nodes and edges respectively of the MVC instance. The reduction can also be seen to preserve approximations; that is, any solution to the MSC problem with cost α corresponds to a solution to the MCCS problem with the same cost α. Since MSC problem cannot be approximated to within a factor $o(\log n)$ unless P = NP [322], the reduction from MSC leads to a similar negative result for the MCCS problem.

Corollary 35. *Unless P = NP, the* MCCS *problem for complex BT-SyDSs cannot be approximated to within* $o(\log n)$*, where* n *is the number of nodes in the given BT-SyDS* $\mathcal{S}$*.*

An examination of the reductions used in the proofs of Theorem 34 and Corollary 35 shows that the initial configuration $\mathcal{C}$ produced in those reductions has the following property: The nodes which are in state 1 in $\mathcal{C}$ form an *independent set* in the underlying graph; that is, for

any pair of nodes u and v which are in state 1 in $\mathcal{C}$, the edge $\{u, v\}$ is not in the underlying graph. So, Corollary 35 points out that even such a restricted version of the MCCS problem cannot be approximated to within a factor $o(\log n)$ for complex BT-SyDSs. Our next result shows that this restricted version can indeed be approximated to within $O \log n)$. This is done by reducing the problem in an approximation preserving manner to the Minimum Set Multi-Cover problem (MSMC) defined in Section D.1.3. The details are given in the proof of the following result.

Theorem 36. *The* MCCS *problem can be approximated to within a factor* $O(\log n)$ *for BT-SyDSs with* n *nodes when the following conditions hold: (a) The up threshold of each node is at least 1. (b) Nodes which are 1 in the initial configuration* $\mathcal{C}$ *(which must be converted into a pseudo fixed point) form an independent set in the underlying graph.*

Proof: Consider an instance of MCCS problem given by a BT-SyDS $\mathcal{S}$ and a configuration $\mathcal{C}$ such that the nodes which are in state 1 in $\mathcal{C}$ form an independent set in the underlying graph of $\mathcal{S}$. We show how to reduce this instance to an instance of the the MSMC problem such that any solution of cost α to the MCCS problem corresponds to a solution to the MSMC instance with the same cost and vice versa.

Let $G(V, E)$ denote the underlying graph of $\mathcal{S}$. Let $V_1 \subseteq V$ denote the subset of nodes whose state values is 1 in $\mathcal{C}$, and let $V_2 = V - V_1$. The set V_2 represents the set of elements to be covered in the MSMC instance. For each node $u \in V_1$, we construct a set S_u consisting of the neighbors of u from V_2. The cost of set S_u is the modification cost $c(u)$ of u. For each node $v \in V_2$, let $\theta_{\text{up}}(v)$ denote its up threshold and let d_v denote the number of neighbors of v from V_1. The coverage requirement r_v for v is set to $\max\{d_v - \theta_{\text{up}}(v) + 1,\ 0\}$. (Elements with coverage requirement of 0 can be deleted.) We thus have an instance of the MSMC problem.

Suppose there is a solution of cost α for the MCCS problem. Let the set of frozen nodes in this solution be denoted by V'. Consider the solution to the MSMC problem by choosing the set S_u for each node $u \in V'$. It can be verified that this collection satisfies the multi-cover requirement and that its cost is α.

Now, suppose there is a solution of cost α for the MSMC problem. Construct the set of nodes frozen V' by adding each node u such that the set S_u is in the solution to the MSMC problem. Consider the configuration $\mathcal{C}_1$ obtained by freezing the nodes in V' to 0. To show that $\mathcal{C}_1$ is a pseudo fixed point, we argue that the successor of $\mathcal{C}_1$ is the configuration of all 0's. First, the frozen nodes stay at 0 by definition. Consider any node $v \in V_1$ which was not frozen. Since the nodes in state 1 in $\mathcal{C}$ form an independent set in the underlying graph, each neighbor of v is a node$w$ in V_2, and the state value of w in $\mathcal{C}_1$ is 0. Since the down threshold of v is 1, v will change to 0 during the transition from $\mathcal{C}_1$. Now, consider any node $x \in V_2$. In the constructed MSMC problem instance, the coverage requirement chosen for x ensures that the number of neighbors of x which are in state 1 in $\mathcal{C}_1$ is less than the up threshold of x. Therefore, x remains in state 0 during the transition from $\mathcal{C}_1$. In other words, all nodes

are in state 0 in the successor of $\mathcal{C}_1$. Since the up threshold for each node is at least 1, the configuration of all 0's is a fixed point. Thus, $\mathcal{C}_1$ is a pseudo fixed point. Moreover, the modification cost needed to produce $\mathcal{C}_1$ is also α.

Thus, the required approximation algorithm for the version of the MCCS problem that satisfies the condition of the theorem is the following: construct an instance of the MSMC problem and use the known $O(\log n)$ approximation for the latter problem. ■

7.4 An Algorithm for Finding Critical Nodes in Complex BT-SyDS

We reiterate that a critical set, for our purposes, is a set of agents that behave in a desired way (corresponding to state 0) and that do not deviate from this behavior. They thus influence their neighbors to refrain from undesirable behaviors (state 1) such as excessive drinking, smoking, or joining a protest.

The approximation algorithm for the MCCS problem mentioned in the proof of Theorem 36 is based on a greedy approach which proceeds as follows [322]. In each iteration, it chooses a set for which the unit cost of covering an element (i.e., the ratio of the cost of a set to the number of elements covered by the set) is a minimum among the remaining sets. The algorithm terminates when the coverage requirement for all the elements has been satisfied. In this section, we use this approach to develop an efficient heuristic (called **Maximum Contributor Heuristic** or MCH) to compute a set B of critical nodes that attempts to eliminate all $0 \to 1$ state transitions. An important advantage of MCH is that it can be employed for both deterministic and probabilistic diffusion. Also, our method increases the candidate set of nodes from which critical sets are obtained, and thus will perform at least as well and often better than a method such as CBH of Chapter 5. For simplicity, it is assumed all the nodes have the same modification cost. With this assumption, the greedy approach has a simple interpretation: in each iteration, choose a set that covers the maximum number of elements. The algorithm can be readily changed to allow nodes to have different modification costs.

First we introduce a few definitions and notation, and describe the central aspect of the algorithm's operation, followed by the algorithm itself. We say a node v is **affectable** at time i if at this time it is in state 0 and and at time $i+1$ it transitions to state 1 (i.e., v has a number $nbr_1(v)$ of neighbors in state 1 such that $nbr_1(v) \geq \theta_{up}(v)$). Let A_i be the set of all affectable nodes at time i. If a neighbor $y \in nbr_1(v)$ is chosen for the critical set B, so that y is frozen at 0, then v may cease to be affectable because $nbr_1(v)$ is reduced. If so, v is removed from A_i. A node u_i^{mc} is a **maximum contributor** if it is in state 1 at time i and has the greatest number of neighbors that are affectable at time i. By definition, $u_i^{mc} \in Q_i$, the set of all nodes that are in state 1 at time i. Our goal is to select a minimum set of

critical nodes from Q_i that reduces A_i to the empty set. We compute a critical node set B for a diffusion instance using Algorithm 4. The **while** loop is executed at each time i, and selects a node u_i^{mc} and updates A_i and Q_i as just described.

Algorithm 4: Compute Critical Set For One Iteration

input : sequence of sets $(Q_i)_{i=1}^{i_{max}}$ of all nodes in state 1 at time i; network G.
output: set of critical nodes B.

Determine from $(Q_i)_{i=1}^{i_{max}}$ the sequence of sets $(A_i)_{i=1}^{i_{max}}$.
Set r =MAX_INT.
for *($i = 1$* **to** *i_{max})* **do**
  // If the number of nodes in state 1 at time i is less than or equal to β, then this set is the critical set.
  if *($|Q_i| \leq \beta$)* **then** Set $B = Q_i$; Return B.

  // Execute greedy covering strategy.
  Set $B_i = \varnothing$.
  if *(A_i is empty)* **then**
    Set B as the set of β nodes randomly chosen from Q_i; Return B.
  while *((A_i not empty)* **and** *($|B_i| \leq \beta$))* **do**
    Identify and remove u_i^{mc} from Q_i; add u_i^{mc} to B_i.
    Remove all nodes $v \in A_i$ where $nbr_1(v) < \theta_{up}(v)$.
  if *(A_i is empty)* **then** Return B_i.

  else if *($|A_i| < r$)* **then** Set $i_{low} = i$ and $r = |A_i|$.
Return $B_{i_{low}}$.

We now address time complexity of the algorithm.

Proposition 37. *Let $G(V, E)$ be the underlying graph of a the given SyDS. The time complexity for Algorithm 4 is $O(|V||E|)$.*

Proof. The time to run the simulations that provide input data for this algorithm can be implemented to run in $O(|V| + |E|)$ time, using Lemma 19. Algorithm 4 follows the same sequence of steps as in the algorithm of Figure 5.3, except that the collection of nodes from which critical nodes are chosen is the set of all nodes currently in state 1, rather than only those affected at time i. In both cases, the sum of the sizes of the sets is at most $|E|$. Hence, one execution of the covering step Algorithm 4 requires $O(|E|)$ time. Over all i_{max} time steps, since $i_{max} \leq |V|$, the worst-case running time is $O(|V||E|)$ for one diffusion instance. □

7.5 Networks and Experimental Parameters

The three networks evaluated experimentally are listed in Table 7.1. The first two are social networks and the third is a social collaboration network. Because a goal is to evaluate the influence of network structure on blocking behavior, the three networks were chosen for study due to their ranges in properties such as numbers of nodes and edges, average degree d_{ave}, and average clustering coefficients cc_{ave}. Variations from a factor of 2 (on numbers of edges and in d_{ave}) to an order of magnitude (in cc_{ave}) were evaluated.

The `epinions` network contains an edge from a member of the Epinions.com web site to another member that it trusts. `slashdot` contains edges between friend and foe contributors (contributors designate themselves as such) to Slashdot, an online news outlet where users supply content. The `astroph` network contains edges between pairs of people that coauthored a paper together; the papers were submitted to *arXiv* for the Astro Physics category. We take all networks to be undirected so diffusion can occur between each pair of nodes in either direction, making blocking diffusion more difficult. Directed edges in the first two networks were converted to undirected edges. Each network has a dominant connected component that was used in this study.

Table 7.1: Networks ([198, 199, 274], respectively) and selected characteristics. "Comp." in the last two columns means connected components.

Network	**Number of Nodes**	**Number of Edges**	**Average Degree**	**Average Clustering Coefficient**	**Num Comp.**	**Size Largest Comp.**
`epinions`	75879	405740	10.7	0.138	2	75877
`slashdot`	77360	469180	12.1	0.0555	1	77360
`astroph`	18771	198050	21.1	0.631	289	17903

Figure 7.3 provides degree distributions, ordered clustering coefficients for individual nodes, and distributions of sizes of k-cores for the three networks. Nodes of the 20-cores were used to compute the seed node sets for diffusion simulations, as described below, and k-cores are used in the results section to interpret findings. In particular, they can be used to determine the maximum possible spread size for a given θ_{up} value, as described in Chapter 4.

Table 7.2 lists the main parameters and values used in the experiments of Section 7.6. Simulations encompassing the cartesian product of all parameters were performed. In total, over 470000 diffusion instances were run and over 390000 critical sets were computed for this parametric study. An **iteration** is a diffusion instance in which initial states for all nodes are specified and dynamics are simulated for some specified maximum number t_{max} of time steps. Simulation durations are $t_{max} = 60$ time units for deterministic iterations and 120 time units for stochastic iterations. A **simulation** is a set of iterations. Each iteration of a simulation only varies in the composition of the seed node set. For deterministic simulations,

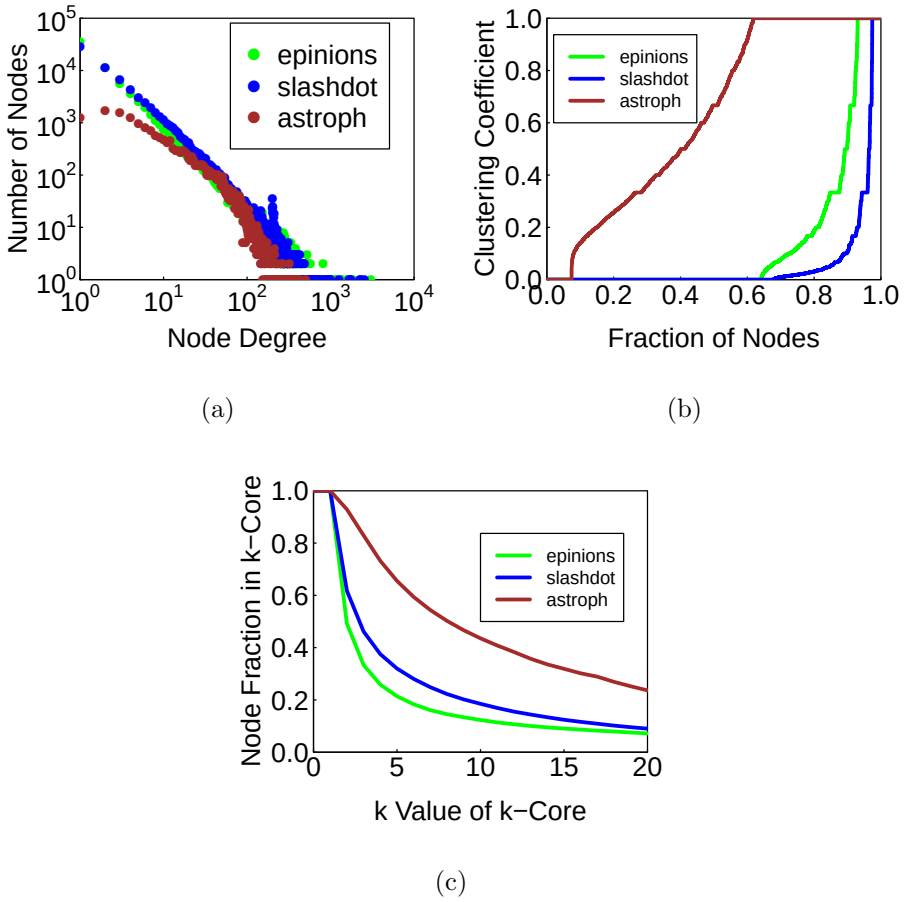


Figure 7.3: For each of the three networks: (a) degree distributions, (b) ordered values of clustering coefficient for each node, and (c) k-core distributions.

50 seed sets were used with each value of n_s. For probabilistic diffusion, 20 instances were completed for each seed set of n_s nodes to capture stochastic propagation, so 1000 iterations (20×50) comprise one simulation. Each set of 20 iterations is called an **iteration group**.

Each seed node was taken from a 20-core. The 20-core is a good compromise between selecting high-degree, well-connected nodes, and having a sufficiently large pool of nodes to choose from in order to minimize the overlap of seed sets. For example, our largest seed size is 20 and for 50 iterations, this requires 1000 nodes. The 20-core provides a pool of 5456 `epinions` nodes, 4439 `astroph` nodes, and 6974 `slashdot` nodes for seeding. Further, the seed sets form a connected subgraph of the original network. Scoping studies indicated that adjacency of seed nodes leads to significantly greater diffusion than allowing the seeds to be disconnected. Thus, the test cases utilized two means, seeding high degree nodes that are connected to each other, to foster diffusion and hence tax the critical node heuristic.

Table 7.2: Parameters and primary values used in the simulation parametric study. Additional parameter values were used for particular parameter sets. This study consisted of over 470000 diffusion instances and 390000 critical node set computations.

Networks	**Thresholds Sets,** $(\theta_{up}, \theta_{down})$	**Numbers of Seeds,** n_s	**Budgets of Critical Nodes,** β	**Transition Probability,** p
`epinions`, `slashdot`, `astroph`	(1,1), (1,2), (2,1), (2,2), (3,3)	2, 3, 5, 10, 20	0, 5, 10, 20, 50, 100	0.5, 1

The state transition probabilities p_{up} and p_{down} were taken as equal in all studies, and are denoted by p. For $p = 1.0$, the parameters of Table 7.2 were augmented by also examining $\beta = 500, 1000$, and 2000, and θ_{down} values as large as 50.

The reason that the probabilistic computations were limited to 20 iterations within an iteration group is output file size, not run time. One simulation of 1000 iterations for $p = 0.5$ and $\beta = 0$ for the two larger networks generates between 40GB and 50GB of output. This is because we need to know the times at which every node changes states for the $\beta = 0$ cases because these results are needed for the critical node determinations. Disk space limitations prevent more iterations from being run. (Note that the way test matrices typically explode in size, it is difficult to imagine a system that has the requisite disk space to permit generation of these large files for all simulations.) For other simulations (where $\beta \neq 0$), we post-process the results on-the-fly, outputting only summary data, and hence can signficantly reduce the output file sizes from simulations.

The test plan consists of running simulations on the three networks for all combinations of threshold sets (θ_{set}), n_s, p, and $\beta = 0$. All nodes except seed nodes are initially in the unaffected state (state 0). The critical node heuristic (Algorithm 4) is executed next. It uses the simulation outputs (for each iteration, the times at which each node changes states) as input data and calculates one set of β critical nodes for each iteration; these critical node

calculations are repeated for each value of β in Table 7.2. Simulations are then repeated, but the critical nodes are included, and are frozen at state 0.

For simulations where $p = 1$, the critical node sets computed for an iteration (i.e., one seed node set) are used in subsequent simulations with $\beta > 0$ for that specific iteration. For simulations where $p = 0.5$, the critical nodes for each iteration group (where all iterations in the group utilize the same seed node set) are aggregated and the β nodes that appear most frequently across the 20 iterations (ties broken arbitrarily) are used as the sole set of critical nodes for simulations of the iteration group for $\beta > 0$. This process (simulations and critical node determinations) requires significantly more time to complete when $p = 0.5$ because of the factor of 20 on the number of iterations per simulation.

This reduced number of 20 iterations for probabilistic diffusion, for one seed set, is also supported by the variation in critical node sets that were determined, as follows. For each of the 20 iterations for one seed node set, the dynamics can be different (i.e., the time histories of the states of the nodes can be different). Hence, we determine separately a different critical node set for each iteration. We compared the variation in the composition of each critical node set with the average critical node set computed over all 20 iterations, for a particular set of seed nodes. We computed the coefficient of variation (COV) ($COV = \sqrt{\sigma}/\mu$, where σ is the standard deviation in the number of node IDs in each of the 20 critical node sets from the node IDs in the mean critical node set, and μ is the mean number of nodes; i.e., $\mu = \beta$) across all 20 instances. We repeated this process for each set of seed nodes. We found that over all of these sets of seed nodes, the average COV was small (0.099) and the maximum was 0.22. That is, the compositions of the 20 different critical node sets, across all 20 iterations, did not vary much. Hence, it was not necessary to simulate more than 20 diffusion instances. This justified the use of 20 iterations per seed node set. In essence, we chose breadth in the parametric study over depth, and the small average COV justified this choice.

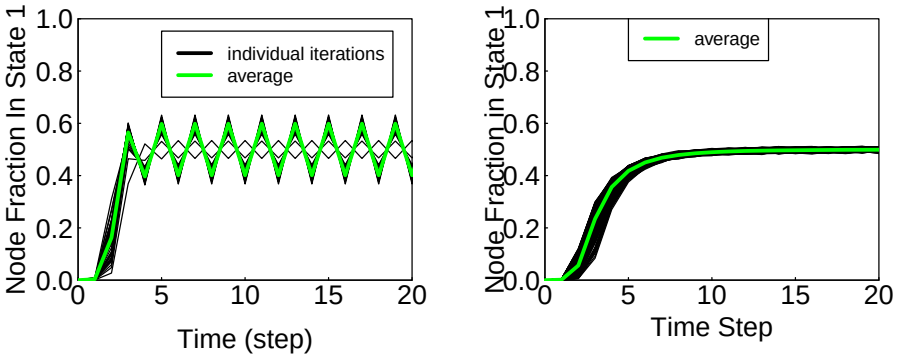
7.6 Simulation Results

We present empirical results from the simulation study. We demonstrate fundamental differences between deterministic and stochastic diffusion dynamics, but show that both systems will not reach a pseudo fixed point without an external forcing mechanism. The mechanism we use is the introduction of critical nodes. With increasing numbers β of critical nodes, transitions from state 0 to state 1 can be reduced or eliminated. We implicitly assume that state 1 is undesirable and hence we seek to minimize the numbers of nodes in this state, and to ever reach state 1. Network structure plays a large role in the dynamics: for small β, the connectedness of a network as measured by the largest component of a k-core governs behavior, while for large β, the number of nodes with large degrees controls behavior. We also compare the efficacy of deterministic, probabilistic, and hybrid probabilistic critical node schemes and show that neither probabilistic scheme always outperforms the other. All

data shown represent the average of 50 and 1000 instances for deterministic and probabilistic diffusion, respectively, except for where we explicitly examine variability across diffusion instances. Also, since $p_{up} = p_{down}$ for all results, we use p for both.

Baseline behaviors for deterministic and stochastic diffusion. Figure 7.4 shows the variability in the number of nodes in state 1 as a function of time, for the `epinions` network, for each of $p = 1$ and $p = 0.5$. The $p = 1$ data are for each iteration of a 50-iteration simulation, where the independent variable is the seed node set. The $p = 0.5$ data are for each of 1000 iterations. Iterations are shown in black; averages in green. Thresholds of 1 were chosen to accentuate the state transitions. Figures 7.5 and 7.6 show analogous results for `slashdot` and `astroph`, respectively.

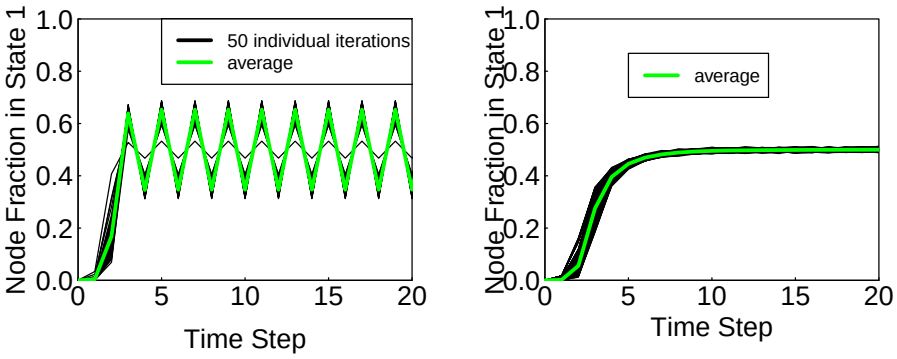
Overall, the data in both plots per figure (deterministic vs. stochastic) show more variation in the initial transient portion of the curves than in the quasi-steady state regime. From the deterministic results, we see some iterations out of phase with other iterations (i.e., some iterations reach a peak in fraction of nodes in state 1 while others reach a minimum). The steady state regime consists a period-2 cycle in the system dynamics for deterministic simulations. In contrast, the probabilistic results show no such oscillations, and the initial transient lasts longer, as expected. For deterministic simulations, in the quasi-steady state regime, all nodes are changing state at each time, and hence Figures 7.4(a), 7.5(a), and 7.6(a) also display the fraction of nodes changing to state 1. In contrast, the fractions of nodes changing states in the probabilistic simulations are about 1/2 the nodes in state 1; that is, the global number of nodes in state 1 holding steady in Figures 7.4(b), 7.5(b), and 7.6(b) hides the fact that about 1/2 of those nodes are actually changing state. (This is why we write *quasi*-steady state.)



(a) (b)

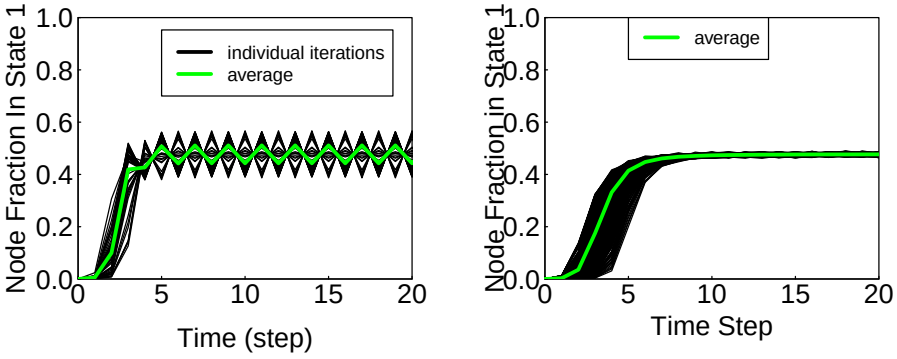
Figure 7.4: For the `epinions` network, the fraction of nodes currently in state 1 (a) for each of 50 iterations and the average for $p = 1$, and (b) for each of 1000 iterations and the average for $p = 0.5$. In both plots, $\theta_{up} = \theta_{down} = 1$, $n_s = 2$, and $\beta = 0$.

In Figure 7.7, we have the same deterministic conditions as described above, except that



(a) (b)

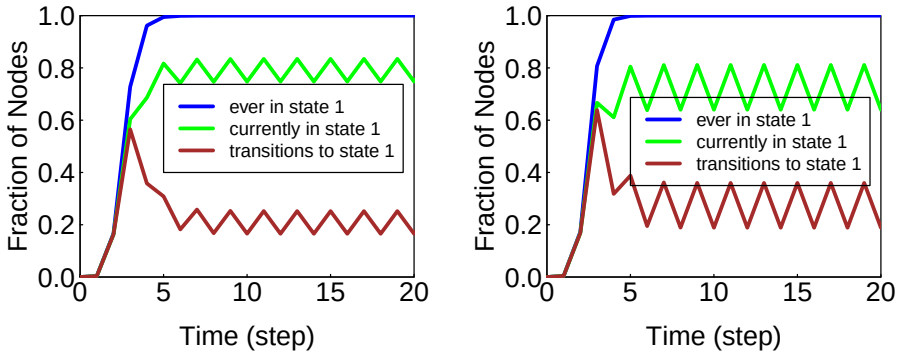
Figure 7.5: For the `slashdot` network, the fraction of nodes currently in state 1 (a) for each of 50 iterations and the average for $p = 1$, and (b) for each of 1000 iterations and the average for $p = 0.5$. In both plots, $\theta_{up} = \theta_{down} = 1$, $n_s = 2$, and $\beta = 0$.



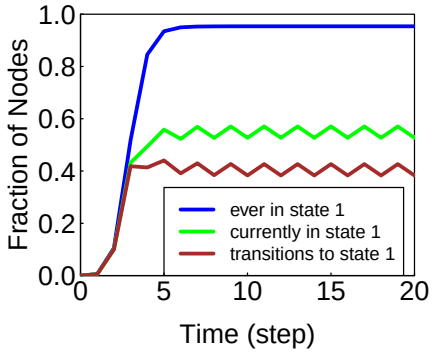
(a) (b)

Figure 7.6: For the `astroph` network, the fraction of nodes currently in state 1 (a) for each of 50 iterations and the average for $p = 1$, and (b) for each of 1000 iterations and the average for $p = 0.5$. In both plots, $\theta_{up} = \theta_{down} = 1$, $n_s = 2$, and $\beta = 0$.

now $\theta_{down} = 2$. The green curves depict the number of nodes in state 1, as before, while the brown curves indicate the number of nodes transitioning to state 1. Now, for `epinions` and `slashdot`, there are large fractions of nodes in state 1 that remain in state 1 (the brown curves are significantly below the green curves). However, for `astroph`, the data are closer to the previous results where all nodes are changing state.



(a) (b)



(c)

Figure 7.7: Number of nodes transitioning to state 1, number of nodes in state 1, and total number of nodes ever reaching state 1 as a function of time for (a) `epinions`, (b) `slashdot`, and (c) `astroph`. For all plots, $\theta_{up} = 1$, $\theta_{down} = 2$, $n_s = 2$, $p = 1$ and $\beta = 0$. The overshoot in nodes transitioning to state 1 is barely visible for `astroph`.

The above results indicate that once a non-zero set of nodes reaches state 1, a system will not drive itself to zero nodes in state 1. This holds for deterministic and probabilistic diffusion (for probabilistic diffusion, at least not in the short-term). The only way to achieve the objective of very few or zero nodes in state 1 is to prevent nodes from ever reaching state 1. Controlling diffusion with critical nodes can accomplish this goal. For networks of large maximum degree (herein, maximum degrees can be as high as 3000), we see will that the

needed number β of critical nodes can be larger than $500n_s$.

From a practical standpoint, this suggests that absent some intervention, in realistic populations where $\theta_{up}, \theta_{down} \approx 1 - 3$, undesirable behaviors will not be naturally driven from a system. Peer influence or some other factor such as an advertising campaign must be used to drive the unwanted behavior from the population. Here we use peer influence, through the MCH algorithm, in the form of individuals who exhibit a desired behavior, cannot be persuaded to adopt an undesirable behavior, and hence who influence others to adopt the wanted behavior.

Effects of critical nodes. Figure 7.8 shows the fraction of nodes in state 1 for different numbers of critical nodes in the `slashdot` network. As β increases from 0 to 1000, the number of nodes in state 1 decreases and the oscillation amplitude attenuates.

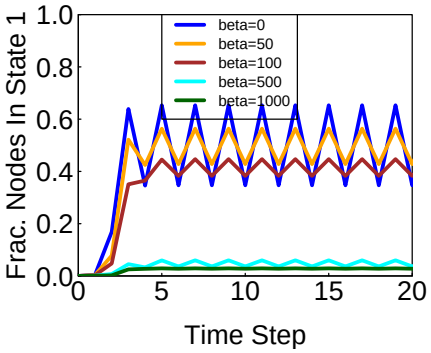


Figure 7.8: Deterministic dynamics for the `slashdot` network with $\theta_{up} = \theta_{down} = 1$ and $n_s = 2$, showing how the fraction of nodes currently in state 1 decreases with increasing numbers of critical nodes.

Comparison of the MCH blocking algorithm with one for simple contagions. We compare the efficacy of MCH for selecting critical nodes with a strategy that works well in determining critical nodes for simple contagions where the only state transition is $0 \to 1$: setting high degree nodes as critical [135]. We refer to this approach as the high degree heuristic (HDH). Table 7.3 summarizes the average fraction of nodes ever reaching state 1 for the three networks and three threshold sets. All results are for deterministic diffusion. Numbers of critical nodes are also given as a fraction of network nodes in column 3. The number of seed nodes is 2; a smaller number means the driving force for diffusion is smaller and hence easier to block, thereby enhancing the effectiveness of HDH. The table shows, e.g., that for HDH and threshold sets (1,1) and (1,2), 88% of `astroph` nodes on average reach state 1; for MCH, no nodes reach state 1.

At first glance, it appears that the HDH does a noticably better job for `epinions` and `slashdot` when $\theta_{up} = 2$ (0.25 and 0.15, respectively). However, k-core analyses show that the fractions of nodes in the 2-cores are 0.49 and 0.62, respectively. Thus, from the results of Chapter 4, HDH allows one-half of the `epinions` nodes that can possibly reach state 1

to actually attain it, while it allows one-quarter of the possible nodes in `slashdot`. Moreover, these results illustrate the important effects of clustering. `astroph` (`slashdot`) has the largest (smallest) average clustering coefficient and the largest (smallest) fraction of nodes reaching state 1. As clustering increases, threshold-2 contagions are better able to propagate; see Section 5.5.3 for similar observations concerning clustering coefficient and complex diffusion. Hence, it is the lack of clustering in `slashdot` that reduces the fraction of affected nodes, rather than the effectiveness of HDH. Average clustering coefficients of 0.1, 0.2, and higher are routinely observed in social networks, meaning that MCH will outperform HDH to a greater extent as clustering increases.

Table 7.3: Comparison of High Degree Heuristic and Maximum Contributor Heuristic. The last two columns show the fraction of network nodes that reach state 1. MCH is far more effective in blocking diffusion. Lesser values are better.

Network	**Threshold Sets,** $(\theta_{up}, \theta_{down})$	**Num. (& %) Critical Nodes**	**Fraction of Affected Nodes, MCH**	**Fraction of Affected Nodes, HDH**
`astroph`	(1,1), (1,2)	500 (3%)	0.00007	0.88
`astroph`	(2,2)	50 (0.3%)	0.0001	0.64
`slashdot`	(1,1), (1,2)	2000 (3%)	0.018	0.63
`slashdot`	(2,2)	50 (0.1%)	0.00003	0.15
`epinions`	(1,1), (1,2)	1000 (1%)	0.00002	0.76
`epinions`	(2,2)	50 (0.1%)	0.00002	0.25

Performance of the MCH blocking algorithm. Fractions of nodes ever reaching state 1 where $n_s = 20$ and $\theta_{down} = 1$ are displayed in Figure 7.9. The first and second plots are for $\theta_{up} = 2$ and 3, respectively. Both plots show "crossover" across the networks because of graph structure effects. For small β, `astroph` has the greatest fraction of nodes reaching state 1, and the largest θ-core size for θ-threshold diffusion, while for larger β, the larger network with the greatest degrees (i.e., `epinions`) enables diffusion to circumvent the critical nodes using these "hub" nodes. These results illustrate that as θ_{up} increases, the number of critical nodes required to halt all diffusion decreases; e.g., the `slashdot` network requires 1500 critical nodes when $\theta_{up} = 2$ and 500 when $\theta_{up} = 3$. In contrast, decreasing θ_{down} (to promote transitions to state 0) has little effect on critical set sizes (results not shown; see figure caption). These results are consistent with the intuitive notion that it may be easier for society to attack problems by increasing the threshold (or cost) for adoption rather than making it "easier" to give up a behavior; these results help to quantify this qualitative observation. We return to the effect of θ_{down} at the end of this section.

Interactions among seed set sizes, critical nodes, and deterministic and probabilistic diffusion are displayed in Figure 7.10 for the `epinions` network where the fraction of nodes ever to reach state 1 are displayed. The first plot shows deterministic diffusion results with $\theta_{up} = \theta_{down} = 2$. The curves transition from concave to convex as β increases from 100 to

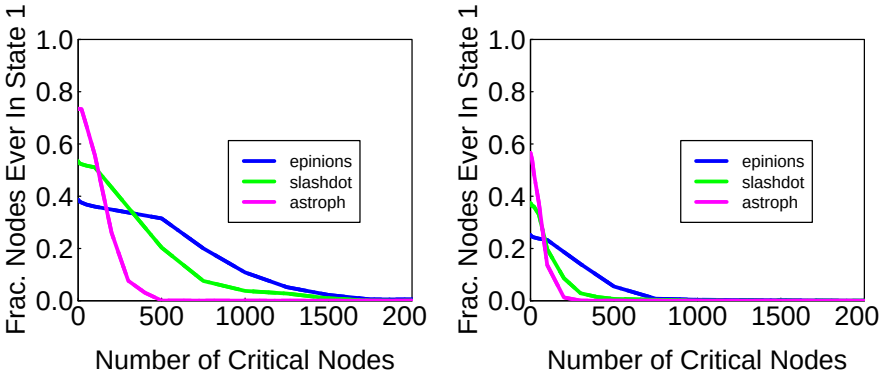


Figure 7.9: Fraction of nodes ever to reach state 1 as a function of β for the three networks where $p = 1$, $\theta_{down} = 1$, and $n_s = 20$; (a) $\theta_{up} = 2$; and (b) $\theta_{up} = 3$. Results are virtually identical for $\theta_{down} = 2, 3$ and are not shown.

500. For $n_s = 20$, roughly $\beta = 100n_s = 2000$ critical nodes are required to stop diffusion.

We compare deterministic and probabilistic diffusion and blocking implementations in Figure 7.10(b). In the absence of critical nodes, the discriminating factor in diffusion is deterministic versus probabilistic state transitions; the seed sets are the same in both sets of simulations. For deterministic simulations, a critical node set is computed for each seed set. For stochastic simulations, a critical set is determined for each of the 20 instances that have the same seed node set. The β nodes with the greatest frequency of occurrence within the 20 sets are taken as the sole critical set and applied to all 20 iterations to assess their blocking capability. Figure 7.10(b) illustrates differences between probabilistic and deterministic diffusion (long dash vs. solid curves) for the `epinions` network, for 3 critical set sizes. For $\beta = 10$, probabilistic diffusion gives greater fractions of nodes reaching state 1, while for $\beta = 20$ and 100 the deterministic and probabilistic curves intersect: as the number of critical nodes increases, the deterministic results show progressively less effective blocking compared to the stochastic results. These behaviors are caused by two competing mechanisms. On one hand, stochastic diffusion has a driving force that is less than that for $p = 1$. On the other hand, critical nodes are determined over 20 instances and represent an averaging process. The critical node sets for deterministic diffusion, in contrast, are for a particular seed set. Which factors dominate are a problem-dependent function of network and diffusion process.

We also used the critical node sets from deterministic diffusion in simulations of stochastic diffusion (short dashes Figure 7.10(b)). The hybrid results are better than the stochastically-determined critical nodes for small β, but are worse for larger β. As the number of nodes ever reaching state 1 is driven to zero, the two methods converge because all diffusion will be halted at time step 1. Less computational effort is required to produce the critical nodes

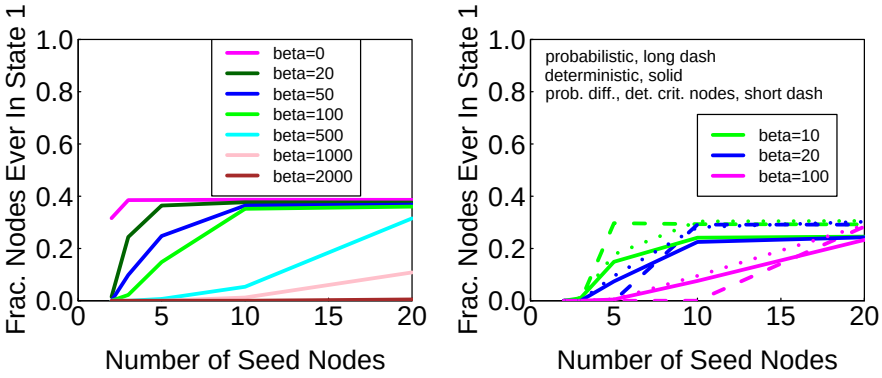


Figure 7.10: For the `epinions` network, the fractions of nodes ever in state 1 for different numbers of seed and critical nodes: (a) for $p = 1$, $\theta_{up} = \theta_{down} = 2$; and (b) for $p = 0.5$ and 1, $\theta_{up} = \theta_{down} = 3$.

for deterministic diffusion.

We combine deterministic blocking results for all three networks and for three β values and $\theta_{up} = \theta_{down} = 2$ in Figure 7.11. The $\beta = 0$ data again follows the 2-core rankings: `astroph` has the largest 2-core and thus has the largest fraction of nodes in state 1. However, for smaller seed sets, $\beta = 50$ nodes generates comparable results among all networks. The curves diverge for larger n_s. For $\beta = 500$ blocking nodes, the curves completely reverse: now the 500 critical nodes more effectively block `astroph` and do the poorest job on `epinions`.

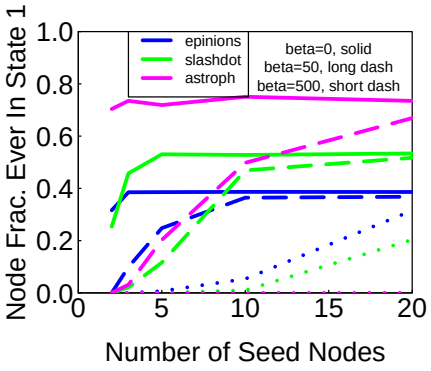
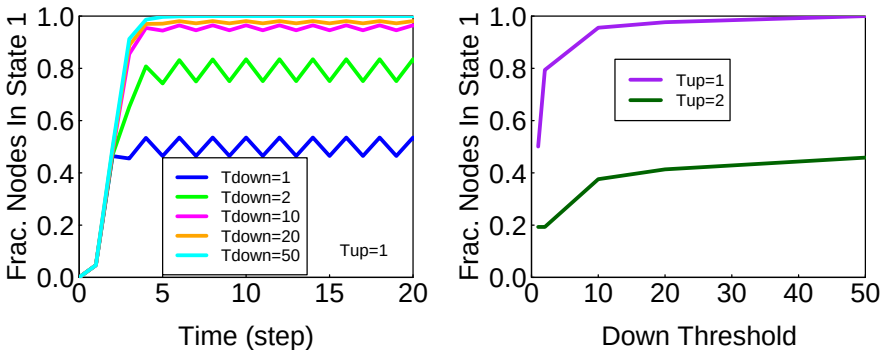


Figure 7.11: Fraction of nodes ever to reach state 1 for $\theta_{up} = \theta_{down} = 2$, $p = 1$ for all three networks.

Effect of θ_{down} in perpetuating an unwanted behavior. A natural question is how large must θ_{down} be to prohibit any node from transitioning from state 1 to state 0, thereby reducing the bi-threshold system to a progressive system, and "locking-in" the unwanted

behavior. Clearly $\theta_{down} = d_{max} + 1$, where d_{max} is the maximum degree of any node in a network, will result in a system where the unwanted behavior is prevalent because no node has $d_{max} + 1$ neighbors. It turns out that much smaller values of θ_{down} can suffice.

Figure 7.12(a) shows the fraction of nodes in state 1 for θ_{down} values ranging from 1 to 50 for `epinions`. As θ_{down} increases, the difficulty in transitioning to state 0 increases. For $\theta_{down} = 50$, all nodes move to and remain in state 1. Since $\theta_{up} = 1$, a connected graph will produce a cascade where all nodes are in state 1, and hence the system will reach a fixed point. If we take the mean of the steady state value of each curve in Figure 7.12(a) and plot them against θ_{down}, we obtain the purple curve in Figure 7.12(b). The other curve is generated from analogous data where $\theta_{up} = 2$. The theoretical maximum in spread size can be computed based on k-cores and is 49% of nodes for the `epinions` network and $\theta_{up} = 2$, which provides an upper bound to the measured result of 40%. From these data, we make two observations. First, since $d_{max} = 3044$ for `epinions`, the system behaves as a progressive system for $\theta_{down} \ll (d_{max} + 1)$. Second, by the time θ_{down} increases to 10 (< 50), these systems are asymptotically approaching progressive systems. From a practical standpoint, these results tell us that even moderate θ_{down} must be reduced to low values in order for people to revert to state 0. This may be the case, for example, for drug use, where reducing addiction may reduce θ_{down}. Thus, interventions may also be needed to reduce the impediments to giving up undesirable behavior.

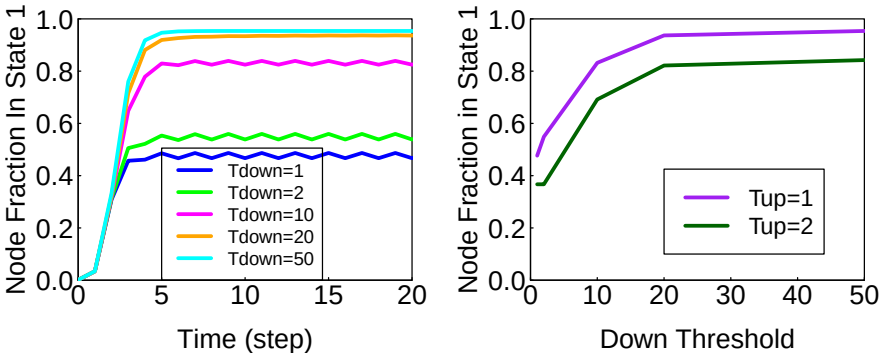


(a) (b)

Figure 7.12: For the `epinions` network: (a) the fraction of nodes currently in state 1 for $\theta_{up} = 1$, $p = 1$ and $n_s = 20$; (b) mean fraction of nodes in state 1 at steady state as a function of θ_{down} for $\theta_{up} = 1$ and $\theta_{up} = 2$.

Data for `slashdot` are very similar to those for `epinions`. However, `astroph` data in Figure 7.13 shows a different response. First, the difference between $\theta_{up} = 1$ and $\theta_{up} = 2$ is much smaller, and this can be observed from the k-core distribution for this graph: there is a much greater 2-core in the `astroph` network than in the `epinions` and `slashdot` networks, and hence the increase in up-threshold has a smaller retarding effect. Second, the point at which θ_{down} yields asymptotic behavior roughly doubles from that for `epinions` and

`slashdot`, from about 10 to about 20. This is also consistent with the greater 2-core and average clustering coefficient for `astroph`.



(a) (b)

Figure 7.13: For the `astroph` network: (a) the fraction of nodes currently in state 1 for $\theta_{up} = 1$, $p = 1$ and $n_s = 20$; (b) mean fraction of nodes in state 1 at steady state as a function of θ_{down} for $\theta_{up} = 1$ and $\theta_{up} = 2$.

7.7 Summary

We motivated and introduced a new bithreshold contagion model where nodes can transition back and forth between states 0 and 1. We identified several differences between simple and complex contagions in this setting. The blocking problem for contagion spreading was formulated and evaluated. We compared our implemented heuristic with the high degree heuristic, which has been shown to work as well as many more sophisticated methods for progressive complex contagions. The blocking method applied to both deterministic and stochastic diffusion have been evaluated on three social networks.

Chapter 8

Concluding Remarks

Using a combination of theory, algorithms, and modeling, we have studied contagion dynamics. Modeling environments such as the one presented here are needed to understand population behaviors at scale. With the increasing use of social media, datasets involving tens of millions of users exist and larger ones are probably in the offing. Hence, modeling capabilities must be able to scale with these sizes. Interesting, too, are the possibilities to develop new models of behavior from these data, as has been done by others. Combining these types of social networks with other types of networks, such as human contact networks, will probably receive increasing attention.

The ability to control contagion dynamics has potentially huge benefits, such disuading a group from unhealthy behaviors. Other applications include product marketing. We have examined approaches for blocking diffusion in this work, but there are other objectives, such as influence maximization. Ultimately, a goal is to inform practical and useful policies for improved public welfare.

Understanding population structure and its impact on dynamics has received less attention to date than understanding network structures themselves, although this, too, has been changing. Identifying static network properties to infer dynamical quantities, such as that done here in bounding the maximum possible contagion spread size, will always be useful.

New models of (human) interaction, such as the bithreshold model introduced here, is another dimension of contagion modeling. It is important to understand basic dynamical properties of such systems, as we have done with the bithreshold model.

Bibliography

[1] Brandon G. Aaby, Kalyan S. Perumalla, and Sudip K. Seal. Efficient Simulation of Agent-Based Models on Multi-GPU and Mult-Core Clusters. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010)*, 2010.

[2] Sherif Elmeligy Abdelhamid, Richard Alo, S. M. Arifuzzaman, Pete Beckman, Md Hasanuzzaman Bhuiyan, Keith Bisset, Edward A. Fox, Geoffrey C. Fox, Kevin Hall, S.M.Shamimul Hasan, Anurodh Joshi, Maleq Khan, Chris J. Kuhlman, Spencer Lee, Jonathan P. Leidig, Hemanth Makkapati, Madhav V. Marathe, Henning S. Mortveit, Judy Qiu, S.S. Ravi, Zalia Shams, Ongard Sirisaengtaksin, Rajesh Subbiah, Samarth Swarup, Nick Trebon, Anil Vullikanti, and Zhao Zhao. CINET: A CyberInfrastructure for Network Science. In *8th IEEE International Conference on eScience*, 2012.

[3] D. Acemoglu, G. Como, F. Fagnani, and A. E. Ozdaglar. Opinion fluctuations and disagreement in social networks. *CoRR*, abs/1009.2653, 2010.

[4] D. Acemoglu and A. Ozdaglar. Opinion dynamics and learning in social networks. *Dynamic Games and Applications*, 1:3–49, March 2011.

[5] S. Adali, R. Escriva, M. Hayvanovych, M. Magdon-Ismail, B. Szymanski, W. Wallace, and G. Williams. Measuring Behavioral Trust in Social Networks. In *IEEE International Conference on Intelligence and Security Informatics (ISI 2010)*, 2010.

[6] Abhijin Adiga, Chris J. Kuhlman, Henning S. Mortveit, and V. S. Anil Kumar. Sensitivity of Diffusion Dynamics to Network Uncertainty. In *The Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*, 2013.

[7] R. Albert, H. Jeong, and A. Barabasi. Error and Attack Tolerance of Complex Networks. *Nature*, 406:378–381, 2000.

[8] Reka Albert and Albert-Laszlo Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[9] Ashton Anderson, Daniel Huttenlocher, Jon Kleinberg, and Jure Leskovec. Effects of User Similarity in Social Media. In *Proceedings of the 5th ACM Symposium on Web Search and Data Mining (WSDM 2012)*, pages 703–712, 2012.

[10] Steffen Angenendt, Muriel Asseburg, Andre Bank, Asiem El Difraoui, Ulrike Freitag, Iris Glosemeyer, Wofram Lacher, Katja Niethammer, Volker Perthes, Walter Posch, Stephan Roll, Johannes Thimm, and Kirsten Westphal. Protest, revolt, and regime change in the arab world 2012. Technical report, German Institute for International and Security Affairs, 2012.

[11] Anonymous. Egypt Braces for Nationwide Protests. In *The HeraldOnline*. 2011.

[12] Anonymous. Egypt Protests a Ticking Time Bomb: Analysts. In *The New Age*. 2011.

[13] Elliot Anshelevich, Deeparnab Chakrabarty, Ameya Hate, and Chaitanya Swamy. Approximation Algorithms for the Firefighter Problem: Cuts over Time and Submodularity. In *Proceedings of the Conference of the International Society for Augmentative and Alternative Communication (ISAAC 2009)*, pages 974–983, 2009.

[14] Sinan Aral, Lev Muchnik, and Arun Sundararajan. Distinguishing Influence-Based Contagion from Homophily-Driven Diffusion in Dynamic Networks. *Proceedings of the National Academy of Sciences*, 106:21544–21549, 2009.

[15] Sinan Aral and Dylan Walker. Identifying Influential and Susceptible Members of Social Networks. *Science*, 337:337–341, 2012.

[16] Dustin Arendt and Yang Cao. Efective gpu acceleration of large scale, asynchronous simulations on graphs. *Advances in Complex Systems*, 15(4):1250035–1–1250035–20, 2012.

[17] A. Arulselvan, C. W. Commander, L. Elefteriadou, and P. M. Pardalos. Detecting Critical Nodes in Sparse Graphs. *Comput. Oper. Res.*, 36(7):2193–2200, 2009.

[18] Luca Dall Asta and Claudio Castellano. Effective Surface-ension in the Noise-Reduced Voter Model. *Europhys. Lett.*, 77(12):60005p1–60005p6, 2011.

[19] K. Atkins, J. Chen, V.S.A. Kumar, M. Macauley, and A. Marathe. Locational market power in network constrained markets. *Journal of Economic Behavior and Organization*, 70:416–430, 2009.

[20] R. Atkinson, W. Dietz, J. Foreyt, N. Goodwin, J. Hill, J. Hirsch, F. Pi-Sunyer, R. Weinsier, R. Wing, J. Hoofnagle, J. Everhart, V. Hubbard, and S. Yanovski. Weight Cycling. *Journal of the American Medical Association*, 272(15):1196–1202, 1994.

[21] Robert Axelrod. The Dissemination of Culture: A Model with Local Convergence and Global Polarization. *Journal of Conflict Resolution*, 41:203–226, 1997.

[22] Nils A. Baas and Torbjorn Helvik. Higher Order Cellular Automata. *Advances in Complex Systems*, 8:169–192, 2005.

[23] David Bader. Analyzing massive social networks using multicore and multithreaded architectures. In *Facing the Multicore-Challenge*, volume 6310 of *Lecture Notes in Computer Science*, pages 1–1. 2011.

[24] R. Bagrodia, K. M. Chandy, and Wen Toh Liao. A Unified Framework for Distributed Simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS 1991)*, 1:348–385, 1991.

[25] A. Barabasi and R. Albert. Emergence of Scaling in Random Networks. *Nature*, 286:509–512, 1999.

[26] Peter D. Barnes, Christopher D. Carothers, David R. Jefferson, and Justin M. LaPre. Warp Speed: Executing Time Warp on 1,966,080 Cores. In *Proceedings of the 27th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2013)*, 2013.

[27] C. Barrett, K. Bisset, S. Eubank, X. Feng, and M. Marathe. 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 (SC 2008)*, 2008.

[28] C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. Complexity of Reachability Problems for Finite Discrete Dynamical Systems. *J. Comput. Syst. Sci.*, 72(8):1317–1345, 2006.

[29] C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. Modeling and analyzing social network dynamics using stochastic discrete graphical dynamical systems. *Theoretical Computer Science*, 412:3932–3946, 2011.

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

[31] Christopher Barrett, Richard Beckman, Maleq Khan, V.S. Anil Kumar, Madhav Marathe, Paula Stretz, Tridib Dutta, and Bryan Lewis. Generation and Analysis of Large Synthetic Social Contact Networks. In *Proceedings of the 2009 Winter Simulation Conference (WSC 09)*, December 2009.

[32] Richard Beckman, Chris J. Kuhlman, Achla Marathe, Elaine Nsoesie, and Samarth Swarup. Modeling the Spread of Smoking in Adolescent Social Networks. In *The Fall Research Conference of the Association for Public Policy Analysis and Management, Advanced Simulation Approaches for Community Sustainability*, 2011.

[33] I. J. Benczik, S. Z. Benczik, B. Schmittmann, and R. K. P. Zia. Opinion dynamics on an adaptive random network. *Physical Review E*, 79:046104–1–046104–10, Apr 2009.

[34] Shishir Bharathi, David Kempe, and Mahyar Salek. Competitive Influence Maximization in Social Networks. In *WINE*, pages 306–311, 2007.

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

[36] K. Bisset, J. Chen, X. Feng, A. Kumar, and M. Marathe. EpiFast: A Fast Algorithm for Large Scale Realistic Epidemic Simulations on Distributed Memory Systems. In *Proceedings of the 23rd International Conference on Supercomputing*, pages 430–439, 2009-b.

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

[38] K. Bisset, X. Feng, M. Marathe, and S. Yardi. Modeling Interaction Between Individuals, Social Networks, and Public Policy to Support Public Health Epidemiology. In *Proceedings of the 2009 Winter Simulation Conference*, pages 2020–2031, 2009.

[39] Keith Bisset, J. Chen, Chris J. Kuhlman, V. S. Anil Kumar, and Madhav V. Marathe. Interaction-Based HPC Modeling of Social, Biological, and Economic Contagions Over Large Networks. In *Proceedings of the 2011 Winter Simulation Conference (WSC 2011) [invited paper]*, December 2011.

[40] Lawrence Blume, David Easley, Jon Kleinberg, Robert Kleinberg, and Eva Tardos. Which Networks Are Least Susceptible to Cascading Failures. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS 2011)*, pages 393–402, 2011.

[41] S. Boccaletti, V. Latora, Y. Moreno, M. Chavezf, and D.-U. Hwanga. Complex networks: Structure and dynamics. *Physics Reports*, 422:175–308, 2006.

[42] G. V. Bodenhausen, L. Sheppard, and G. Kramer. Negative Affect and Social Judgment: The Differential Impact of of Anger and Sadness. *European Journal of Social Psychology*, 24:45–62, 1994.

[43] H. Bodlaender. Treewidth: Algorithmic Techniques and Results. In *Proc. 22nd Symp. Mathematical Foundations of Computer Science (MFCS 1997)*, pages 29–36, 1997.

[44] Phillip Bonacich. Factoring and weighting approaches to status scores and clique indentification. *Journal of Mathematical Sociology*, 2:113–120, 1972.

[45] Phillip Bonacich. Some unique properties of eigenvector centrality. *Social Networks*, 29:555–564, 2007.

[46] Robert M. Bond, Christopher J. Fariss, Jason J. Jones, Adam D. I. Kramer, Cameron Marlow, Jaime E. Settle, and James H. Fowler. A 61-million-person experiment in a social influence and political mobilization. *Nature*, 489:295–298, 2012.

[47] S. Borgatti. Identifying sets of key players in a social network. *Comput Math Organiz Theor*, 12:21–34, 2006.

[48] Stephen P. Borgatti, Ajay Mehra, Daniel J. Brass, and Giuseppe Labianca. Network analysis in the social sciences. *Science*, 323:892–895, 2000.

[49] Katy Borner. Plug-and-Play Macroscopes. *Communications of the ACM*, 54:60–68, 2011.

[50] Allan Borodin, Yuval Filmus, and Joel Oren. Threshold Models for Competitive Influence in Social Networks. In *Proceedings of the 6th international conference on Internet and network economics (WINE'10)*, pages 1–15, 2010.

[51] Doruk Bozdag, Umit V. Catalyurek, Assefaw H. Gebremedhin, Fredrik Manne, Erik G. Boman, and Fusun Ozguner. Distributed-Memory Parallel Algorithms for Distance-2 Coloring and Related Problems in Derivative Computation. *SIAM Journal of Scientific Computing*, 32:2418–2446, 2010.

[52] Melvyn Bragg. *The Adventure of English.* Arcade Publishing, 2003.

[53] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

[54] Linda Briesemeister, Patrick Lincoln, and Phillip Porras. Epidemic profiles and defense of scale-free networks. In *Proceedings of the 2003 ACM CCS Workshop on Rapid Malcode (WORM 03)*, pages 67–75, 2003.

[55] Sergey Brin and Larry Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the Seventh International World Wide Web Conference (WWW 1998)*, 1998.

[56] Christopher Brown, Joey Frazee, David Beaver, Xiong Liu, Fred Hoyt, and Jeff Hancock. Evolution of Sentiment in the Libyan Revolution, 2011. Working Paper for the NSF Minerva Project.

[57] Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. Limiting the Spread of Misinformation in Social Networks. In *Proceedings of the 20th International Conference of World Wide Web Conference (WWW 2011)*, 2011.

[58] N. Bulger, A. DeLongis, R. Kessler, and E. Wethington. The Contagion of Stress Across Multiple Roles. *Journal of Marriage and the Family*, 51:175–183, 1989.

[59] Colin Campbell, Suann Yang, Rka Albert, and Katriona Sheab. A network model for plantpollinator community assembly. *Proceedings of the National Academy of Sciences*, 108(1):197–202, 2011.

[60] A. Cangelosi and D. Parisi. Computer Simulation: A New Scientific Approach to the Study of Language Evolution. In A. Cangelosi and D. Parisi, editors, *Simulating the Evolution of Language*. Springer, 2001.

[61] Christopher D. Carothers and Kalyan S. Perumalla. On Deciding Between Conservative and Optimistic Approaches on Massively Parallel Platforms. In *Proceedings of the 2010 Winter Simulation Conference (WSC 2010)*, 2010.

[62] Christopher D. Carothers, Kalyan S. Perumalla, and Richard M. Fujimoto. Efficient Optimistic Parallel Simulations Using Reverse Computation. *ACM Transactions on Modeling and Computer Simulation (TOMACS 1999)*, 9:224–253, 1999.

[63] C. Castellano, S. Fortunato, and V. Loreto. Statistical physics of social dynamics. *Rev. Mod. Phys.*, 81(2):591–646, 2009.

[64] C. Castillo-Chavez and B. Song. Models for the transmission dynamics of fanatic behaviors. In H. Banks and C. Castill-Chavez, editors, *Bioterrorism: Mathematical Modeling Applications in Homeland Security; in SIAM Frontiers in Applied Mathematics*, pages 155–172. SIAM, 2003.

[65] D. Centola. Failure in Complex Social Networks. *J. Math. Sociology*, 33:64–68, 2009.

[66] D. Centola, V. Eguiluz, and M. Macy. Cascade Dynamics of Complex Propagation. *Physica A*, 374:449–456, 2006.

[67] D. Centola and M. Macy. Complex Contagions and the Weakness of Long Ties. *American Journal of Sociology*, 113(3):702–734, 2007.

[68] Damon Centola. The spread of behavior in an online social network experiment. *Science*, 329:1194–1197, 2010.

[69] Damon Centola, Juan Carlos Gonzalez-Avella, Victor M. Eguiluz, and Maxi San Miguel. Homophily, cultural drift, and the co-evolution of cultural groups. *Journal of Conflict Resolution*, 51:905–929, 2007.

[70] M. Cha, A. Mislove, B. Adams, and K. Gummadi. Characterizing Social Cascades in Flickr. In *Proc. of the First Workshop on Online Social Networks (WOSN 08)*, pages 13–18, 2008.

[71] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos. Epidemic Thresholds in Real Networks. *ACM Trans. Inf. Syst. Secur.*, 10(4):13–1 – 13–26, 2008.

[72] Wai Kin Victor Chan. Agent-Based Simulation Tutorial–Simulation of Emergent Behavior and Differences Between Agent-Based Simulation and Discrete Event Simulation. In *Proceedings of the 2010 Winter Simulation Conference (WSC 2010)*, pages 135–150, 2010.

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

[74] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Commun. ACM*, 24(4):198–206, April 1981.

[75] Karthik Channakeshava, Keith Bisset, Madhav V. Marathe, V. S. Anil Kumar, and Shrirang Yardi. High performance scalable and expressive modeling environment to study mobile malware in large dynamic networks. In *Proceedings of 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2011)*, pages 770–781, 2011.

[76] Wei Chen, Alex Collins, Rachel Cummings, Te Ke, Zhenming Liu, David Rincón, Xiaorui Sun, Yajun Wang, Wei Wei, and Yifei Yuan. Influence Maximization in Social Networks When Negative Opinions May Emerge and Propagate. In *SDM*, pages 379–390, 2011.

[77] Wei Chen, Alex Collins, Rachel Cummings, Te Ke, Zhenming Liu, David Rincón, Xiaorui Sun, Yajun Wang, Wei Wei, and Yifei Yuan. Influence maximization in social networks when negative opinions may emerge and propagate. In *SIAM International Conference on Data Mining (SDM 2011)*, pages 379–390, 2011.

[78] Wei Chen, Chi Wang, and Yajun Wang. Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks. In *Proc. ACM Intl. Conf. on Data Mining and Knowledge Discovery (KDD 2010)*, pages 1029–1038, 2010.

[79] Wei Chen, Yifei Yuan, and Li Zhang. Scalable Influence Maximization in Social Networks Under the Linear Threshold Model. In *Proceedings of the 10th IEEE Conference on Data Mining (ICDM 2010)*, pages 88–97, 2010.

[80] H. Choi, S. Kim, and J. Lee. Role of network structure and network effects in diffusion of innovations. *Industrial Marketing Management*, 39:170–177, 2010.

[81] F. Chung, L. Lu, and V. Vu. The Spectra of Random Graphs with Given Expected Degrees. *Internet Mathematics*, 1(3):257–275, 2003.

[82] Fan R. K. Chung and Linyuan Lu. The volume of the giant component of a random graph with given expected degrees. *SIAM J. Discrete Math.*, 20(2):395–411, 2006.

[83] Michael Suk-Young Chwe. Structure and strategy in collective action. *American Journal of Sociology*, 105:128–156, 1999.

[84] A. Clauset, C. R. Shalizi, and M. Newman. Power-Law Distributions In Empirical Data. *SIAM Review*, 51:661–703, 2009.

[85] P. Clifford and Aidan Sudbury. A model for spatial conflict. *Biometrika*, 60(3):581–588, 1973.

[86] Simon Coakley, Mairan Gheorghe, Mike Holcombe, Shawn Chin, David Worth, and Chris Greenough. Exploitation of High Performance Computing in the FLAME Agent-Based Simulation Framework. In *2012 IEEE 14th International Conference on High Performance Computing and Communications (HPCC 2012)*, pages 538–545, 2012.

[87] Edward Coffman, Michael Elphick, and Ari Shoshani. System deadlocks. *Computing Surveys*, 3:67–78, 1971.

[88] R. Cohen, , S. Havlin, and D. ben Avraham. Efficient Immunization Strategies for Computer Networks and Populations. *Physical Review Letters*, 91:247901–1–247901–4, 2003.

[89] J. Coleman. Social Theory, Social Research, and a Theory of Action. *American Journal of Sociology*, 91(6):1309–1335, 1986.

[90] Nicholson Collier and Michael North. Repast HPC: A Platform for Large-scale Agent-based Modeling. In Werner Dubitzky, Krzysztof Kurowski, and Bernard Schott, editors, *Large-Scale Computing Techniques for Complex System Simulations*, chapter 5, pages 81–110. Wiley–IEEE Computer Society, 2011.

[91] Nicholson Collier and Michael North. Parallel agent-based simulation with Repast for High Performance Computing. *Simulation*, 2012.

[92] T. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.

[93] Nicole Crepaz, Khiya J. Marshall, Latrina W. Aupont, Elizabeth D. Jacobs, Yuko Mizuno, Linda S. Kay, Patricia Jones, Donna Hubbard McCree, and Ann O'Leary. The efficacy of hiv/sti behavioral interventions for african american females in the united states: A meta-analysis. *American Journal of Public Health*, 99:2069–2078, 2009.

[94] Paolo Crucitti, Vito Latora, Massimo Marchiori, and Andrea Rapisarda. Error and attack tolerance of complex networks. *Physica A*, 340:388–394, 2004.

[95] Sanjiv R. Das and Mike Y. Chen. Yahoo! for amazon: Sentiment extraction from small talk on the web. *Management Science*, 53:1375–1388, 2007.

[96] Sanmay Das and Malik Magdon-Ismail. A model for information growth in collective wisdom processes. *ACM Transactions on Knowledge Discovery from Data*, 2012. to appear.

[97] Christophe Deissenberg, Sander van der Hoog, and Herbert Dawid. EURACE: A massively parallel agent-based model of the European economy. *Applied Mathematics and Computation*, 204:541–552, 2008.

[98] Z. Dezso and A. Barabasi. Halting Viruses In Scale-Free Networks. *Physical Review E*, 65:055103–1–055103–4, 2002.

[99] Mario Diani. Social Movements and Collective Action. In John Scott and Peter J. Carrington, editors, *The SAGE Handbook of Social Network Analysis*, pages 223–235. SAGE, 2011.

[100] Daniel Diermeier and Jan A. Van Mieghem. Spontaneous collective action, 2001.

[101] P. Dodds and D. Watts. A Generalized Model of Social and Biological Contagion. *Journal of Theoretical Biology*, 232(4):587–604, 2005.

[102] R. William Doherty. The Emotional Contagion Scale: A Measure of Individual Differences. *Journal of Nonverbal Behavior*, 21:131–154, 1997.

[103] P. Domingos and M. Richardson. Mining the Network Value of Customers. In *Proc. ACM Intl. Conf. on Data Mining and Knowledge Discovery (KDD 2001)*, pages 57–61, 2001.

[104] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes. Critical phenomena in complex networks. *Reviews of Modern Physics*, 80:1275–1335, 2008.

[105] P. Dreyer and F. Roberts. Irreversible k-Threshold Processes: Graph-Theoretical Threshold Models of the Spread of Disease and Opinion. *Discrete Applied Mathematics*, 157:1615–1627, 2009.

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

[107] Jon Elster. *The Cement of Society*. Cambridge University Press, 1989.

[108] J. Epstein. Modeling civil violence: An agent-based computational approach. *PNAS*, 99:7243–7250, 2002.

[109] Joshua M. Epstein, Jon Parker, Derek Cummings, and Ross A. Hammond. Coupled Contagion Dynamics of Fear and Disease: Mathematical and Computational Explorations. *PLoS ONE*, 3:e3955–1–e3955–11, 2008.

[110] S. Eubank, V. S. Anil Kumar, M. V. Marathe, A. Srinivasan, and N. Wang. Structure of Social Contact Networks and Their Impact on Epidemics. In J. Abello and G. Cormode, editors, *Discrete Methods in Epidemiology*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 179–200. American Mathematical Society, Providence, RI, 2006.

[111] E. Even-Dar and A. Shapira. A Note on Maximizing the Spread of Influence in Social Networks. In *Workshop on Internet and Network Economics (WINE 2007), LNCS 4858*, volume 4858, pages 281–286, 2007.

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

[113] J. Fowler. Turnout in a Small World. In *Social Logistics of Politics*, pages 269–287. Temple University Press, 2005.

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

[115] Linton C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40:35–41, 1977.

[116] Linton C. Freeman. Centrality in social networks: conceptual clarification. *Social Networks*, 1:215–239, 1978.

[117] Noah E. Friedkin and Eugene C. Johnson. Social Influence Networks and Opinion Change. *Advances in Group Processes*, 16:1–29, 1999.

[118] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley-Interscience, 2000.

[119] A. Ganesh, L. Massoulie, and D. Towsley. On the Vulnerability of Large Graphs. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, volume 2, pages 1455–1466, 2005.

[120] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Co., San Francisco, CA, 1979.

[121] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *Proceedings of the International Conference on Advanced Robotics (ICAR 2003)*, pages 317–323, 2003.

[122] Michelle Girvan and Mark E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences (PNAS)*, 99:78217826, 2002.

[123] Misha Glenny. *McMafia*. Alfred A. Knopf, New York, New York, 2008.

[124] Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the Network: A Complex Systems Look at the Underlying Process of Word of Mouth. *Marketing Letters*, 12:211–223, 2001.

[125] Eric Goles and Servet Martinez. *Neural and Automata Networks*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.

[126] Bruno Goncalves, Nicola Perra, and Alessandro Vespignani. Modeling Users' Activity on Twitter Networks: Validation of Dunbar's Number. *PLoS ONE*, 6:e22656–1–e22656–5, 2011.

[127] Sandra Gonzalez-Bailon, Javier Borge-Holthoefer, Alejandro Rivero, and Yamir Moreno. The Dynamics of Protest Recruitment Through an Online Network. *Nature Scientific Reports*, pages 1–7, 2011. DOI: 10.1038/srep00197.

[128] Roger V. Gould. Multiple networks and mobilization in the paris commune, 1871. *American Sociological Review*, 56:716–729, 1991.

[129] Roger V. Gould. Collective action and network structure. *American Sociological Review*, 58:182–196, 1993.

[130] Przemyslaw A. Grabowicz, Jose J. Ramosco, Esteban Moro, Josep M. Pujol, and Victor M. Eguiluz. Social features of online networks: The strength of intermediary ties in online social media. *PLoS ONE*, 7:e29358–1–e29358–9, 2012.

[131] M. Granovetter. The Strength of Weak Ties. *American Journal of Sociology*, 78(6):1360–1380, 1973.

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

[133] D. Gruhl, R. Guha, D. Liben-Nowell, and A. Tomkins. Information Diffusion Through Blogspace. In *Proc. of the 13th International World Wide Web Conference (WWW 2004)*, pages 491–501, 2004.

[134] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of Trust and Distrust. In *Proc. of the 13th International World Wide Web Conference (WWW 2004)*, pages 403–412, 2004.

[135] Habiba, Y. Yu, T. Berger-Wolf, and J. Saia. Finding Spread Blockers in Dynamic Networks. In *The 2nd SNA-KDD Workshop '08 (SNA-KDD 2008)*, 2008.

[136] M. Halloran, N. Ferguson, I. Longini S. Eubank, D. Cummings, B. Lewis, S Xu, C. Fraser, A. Vullikanti, T. Germann, D. Wagener, R. Beckman, K. Kadau, C. Barrett, C. Macken, D. Burke, and P. Cooley. Modeling targeted layered containment of an influenza pandemic in the united states. *PNAS*, 105(12):4639–4644, 2008.

[137] Baris Hancioglu, David Swigon, and Gilles Clermont. A dynamical model of human immune response to influenza A virus infection. *Journal of Theoretical Biology*, 246(1):70–86, 2007.

[138] David Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[139] K. Harris. The National Longitudinal Study of Adolescent Health (Add Health), Waves I and II, 1994-1996; Wave III, 2001-2002 [machine-readable data file and documentation], 2008. Chapel Hill, NC: Carolina Population Center, University of North Carolina at Chapel Hill 2008.

[140] E. Hatfield, J. T. Cacioppo, and R. L. Rapson. *Emotional Contagion*. Cambridge University Press, 1994.

[141] Taher Haveliwala and Sepandar Kamvar. The second eigenvalue of the google matrix. Technical Report 7056, Stanford University, 2003.

[142] Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. Influence Blocking Maximization in Social Networks Under the Competitive Linear Threshold Model. In *Technical Report*, 2011. Appears as arXiv:1110.4723v1.

[143] Eric Hoffer. *The True Believer*. Harper Perennial Modern Classics, 2002. Originally published in 1951.

[144] Beth R. Hoffman, Steve Sussman, Jennifer B. Unger, and Thomas W. Valente. Peer influences on adolescent cigarette smoking: A theoretical review of the literature. *Substance Use and Misuse*, 41:103–155, 2006.

[145] R. Holley and Thomas Liggett. Ergodic Theorems for Weakly Interacting Infinite Systems and the Voter Model. *Annals of Probability*, 3(4):643–663, 1975.

[146] P. Holme. Efficient local strategies for vaccination and network attack. *Europhysics Letters*, 68:908–914, 2004.

[147] P. Holme, B. Kim, C. Yoon, and S. Han. Attack vulnerability of complex networks. *Physical Review E*, 65:056109–1–056109–14, 2002.

[148] Petter Holme and M. E. J. Newman. Nonequilibrium phase transition in the coevolution of networks and opinions. *Physical Review E*, 74:056108–1–056108–5, 2006.

[149] M. Hoogendoorn, J. Treur, C. N van der Wal, and A. van Wissen. An agent-based model for the interplay of information and emotion in social diffusion. In *Proceedings of the 10th IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2010)*, pages 439–444, 2010.

[150] Carol Huang. Facebook and Twitter key to Arab Spring uprisings: report. In *The National*. 2011.

[151] Cindy Hui, Mark Goldberg, Malik Magdon-Ismail, and William A. Wallace. Agent-Based Simulation of the Diffusion of Warnings. In *Agent-Directed Simulation Symposium (ADS '10) as part of the 2010 Spring Simulation MultiConference (SpringSim '10)*, 2010.

[152] B. W. K. Hung, S. E. Kolitz, and A. Ozdaglar. Optimization-Based Influencing of Village Social Networks in Counterinsurgency. In *Proceedings of the 4th International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction (SBP 2011)*, pages 10–17, 2011.

[153] Harry B. Hunt III, Madhav V. Marathe, Venkatesh Radhakrishnan, and Richard Edwin Stearns. The Complexity of Planar Counting Problems. *SIAM J. Comput.*, 27(4):1142–1167, 1998.

[154] M. Hybinette, E. Kraemer, Y. Xiong, G. Matthews, and J. Ahmed. SASSY: A Design for Scalable Agent-Basd Simulation System Using a Distributed Discrete Event Infrastructure. In *Proceedings of the 2006 Winter Simulation Conference*, pages 926–933, 2006.

[155] D. J. Irons. Logical analysis of the budding yeast cell cycle. *Journal of Theoretical Biology*, 257:543–559, 2009.

[156] J. Leskovec website, 2011. `http://cs.stanford.edu/people/jure/`.

[157] David Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7:404–425, 1985.

[158] Cong Jin, Jun Liu, and Qinghua Deng. Network virus propagation model based on effects of removing time and user vigilance. *International Journal of Network Security*, 9:156–163, 2009.

[159] Ira M. Longini Jr., Azhar Nizam, Shufu Xu, Kumnuan Ungchusak, Wanna Hanshaoworakul, Derek A. T. Cummings, and M. Elizabeth Halloran. Containing pandemic influenza at the source. *Science*, 309:1083–1087, 2005.

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

[161] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998.

[162] K. Kawachi. Deterministic models for rumor transmission. *Nonlinear Analysis: Real World Applications*, 9:1989–2028, 2008.

[163] Kazuki Kawachi, Motohide Seki, Hiraku Yoshida, Yohei Otake, Katsuhide Warashina, and Hiroshi Ueda. A rumor transmission model with various contact interactions. *Journal of Theoretical Biology*, 253:55–60, 2008.

[164] Jared P. Keller, Kevin C. Desouza, and Yuan Lin. Dismantling terrorist networks: Evaluating strategic options using agent-based modeling. *Technological Forecasting & Social Change*, 77:1014–1036, 2010.

[165] Jeffrey A. Kelly, Yuri A. Amirkhanian, Elena Kabakchieva, Sylvia Vassileva, Timothy L. McAuliffe, Wayne J. DiFranceisco, Radostina Antonova, Elena Petrova, Boyan Vassilev, Roman A. Khoursine, and Borislav Dimitrov. Prevention of hiv and sexually transmitted diseases in high risk social networks of young roma (gypsy) men in bulgaria:randomised controlled trial. *BMJ*, 333:1098–1101, 2006.

[166] David Kempe, Jon Kleinberg, and Eva Tardos. Maximizing the Spread of Influence Through a Social Network. In *Proc. ACM Intl. Conf. on Data Mining and Knowledge Discovery (KDD 2003)*, pages 137–146, 2003.

[167] David Kempe, Jon Kleinberg, and Eva Tardos. Influential Nodes in a Diffusion Model for Social Networks. In *Proc. Intl. Conf. on Automata, Languages and Programming (ICALP 2005)*, pages 1127–1138, 2005.

[168] Bernice Roberts Kennedy and Chalice C. Jenkins. Promoting african american women and sexual assertiveness in reducing hiv/aids: An analytical review of the research literature. *Journal of Cultural Diversity*, 18:142–149, 2011.

[169] Masahiro Kimura, Kazumi Saito, Kouzou Ohara, and Hiroshi Motoda. Opinion Formation by Voter Model with Temporal Decay Dynamics. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (PKDD 2012)*, pages 565–580, 2012.

[170] Gary King, Jennifer Pan, and Margaret Roberts. How censorship in china allows government criticism but silences collective expression. *Working Paper*, pages 1–36, 2012.

[171] J. Kitts. Mobilizing in Black Boxes: Social Networks and SMO Participation. *Mobilization*, 5:241–257, 2000.

[172] J. Kleinberg. Cascading Behavior in Networks: Algorithmic and Economic Issues. In N. Nissan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 24, pages 613–632. Cambridge University Press, New York, NY, 2007.

[173] John M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46:604–632, 1999.

[174] Jon Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2002)*, pages 91–101, 2002.

[175] Jon Kleinberg. Computing: The Wireless Epidemic. *Nature*, 449:287–288, 2007.

[176] G. Kossinets, J. Kleinberg, and D. Watts. The Structure of Information Pathways in a Social Communication Network. In *Proc. ACM Intl. Conf. on Data Mining and Knowledge Discovery (KDD 2008)*, 2008.

[177] Jan Kostka, Yvonne Anne Oswald, and Roger Wattenhofer. Word of Mouth: Rumor Dissemination in Social Networks. In *Structural Information and Communication Complexi,ty Lecture Notes in Computer Science Volume 5058 (SIROCCO 2008)*, pages 185–196, 2008.

[178] P. L. Krapivsky, S. Redner, and D. Volovik. Reinforcement-Driven Spread of Innovations and Fads. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(12):P12003–1–P12003–12, 2011.

[179] Chris J. Kuhlman, V. S. Anil Kumar, Madhav V. Marathe, Henning S. Mortveit, Samarth Swarup, Gaurav Tuli, S. S. Ravi, and Daniel J. Rosenkrantz. A General-Purpose Graph Dynamical System Modeling Framework. In *Proceedings of the 2011 Winter Simulation Conference (WSC 2011)*, December 2011.

[180] Chris J. Kuhlman, V. S. Anil Kumar, Madhav V. Marathe, S. S. Ravi, and Daniel J. Rosenkrantz. Finding Critical Nodes for Inhibiting Diffusion of Complex Contagions in Social Networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (PKDD 2010)*, pages 111–127, 2010.

[181] Chris J. Kuhlman, V. S. Anil Kumar, Madhav V. Marathe, S. S. Ravi, and Daniel J. Rosenkrantz. Effects of Opposition on the Diffusion of Complex Contagions in Social Networks: An Empirical Study. In *Proceedings of the 2011 International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction (SBP11)*, pages 188–196, 2011.

[182] Chris J. Kuhlman, V. S. Anil Kumar, Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, Samarth Swarup, and Gaurav Tuli. A Bithreshold Model of Complex Contagion and its Application to the Spread of Smoking Behavior. In *Proceedings of the Workshop on Social Network Mining and Analysis (SNA-KDD 2011)*, August 2011.

[183] Chris J. Kuhlman, V. S. Anil Kumar, Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, Samarth Swarup, and Gaurav Tuli. 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)*, pages 91–100, November 2011.

[184] Chris J. Kuhlman, V. S. Anil Kumar, and S. S. Ravi. Critical sets to control bias in discrete opinion dynamics. In *Proceedings of the 2012 ACM Web Science Conference (WebSci 2012)*, pages 255–264, 2012.

[185] Chris J. Kuhlman, V. S. Anil Kumar, and S. S. Ravi. Controlling Opinion Propagation in Online Networks. *Journal of Computer Networks*, 2013.

[186] Chris J. Kuhlman, Madhav V. Marathe, S. S. Ravi, and Daniel J. Rosenkrantz. Exploiting Network Structure in Enhancing Diffusion of Complex Contagions. In *Proceedings of the Analysis of Complex Networks (ACNE) Workshop of ECML PKDD 2010*, pages 20–34, September 2010.

[187] Chris J. Kuhlman, Henning S. Mortveit, David Murrugarra, and V. S. Anil Kumar. Bifurcations in Boolean Networks. In *Discrete Mathematics and Theoretical Computer Science proceedings of the 17th International Workshop on Cellular Automata and Discrete Complex Systems (DMTCS Automata 2011)*, pages 29–46, November 2011.

[188] Timur Kuran. Sparks and prairie fires: A theory of unanticipated political revolution. *Public Choice*, 61:41–74, 1989.

[189] Timur Kuran. The inevitability of future revolutionary surprises. *American Journal of Sociology*, 61:1528–1551, 1995.

[190] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a Social Network or a News Media. In *Proc. of the 19th International World Wide Web Conference (WWW 2010)*, 2010.

[191] Harri Lahdesmaki, Sampsa Hautaniemi, Ilya Shmulevich, and Olli Yli-Harja. Relationships between probabilistic Boolean networks and dynamic Bayesian networks as models of gene regulatory networks. *Signal Processing*, 86:814–834, 2006.

[192] Mayank Lahiri and Manuel Cebrian. The genetic algorithm as a general diffusion model for social networks. In *In Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, 2010.

[193] Carl A. Latkin. Outreach in natural settings: The use of peer leaders for hiv prevention among injecting drug users' networks (supplement). *Public Health Reports*, 113:151–159, 1998.

[194] R. Laubenbacher, A. Jarrah, H. Mortveit, and S. Ravi. A Mathematical Formalism for Agent-Based Modeling. In R. Meyers, editor, *Encyclopedia of Complexity and System Science*, pages 160–176. Springer, 2009.

[195] Hsien-Yi Lee. Contagion in International Stock Markets during the Sub Prime Mortgage Crisis. *International Journal of Economics and Financial Issues*, 2:41–53, 2012.

[196] J. Leskovec, L. Adamic, and B. Huberman. The Dynamics of Viral Marketing. *ACM Transactions on the Web*, 1(1), 2007.

[197] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting Positive and Negative Links in Online Social Networks. In *Proc. of the 19th International World Wide Web Conference (WWW 2010)*, 2010.

[198] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM Transactions on Knowledge Discovery from Data (ACM TKDD)*, 1(1), 2007.

[199] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters, 2008. Appears as arXiv.org:0810.1355.

[200] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Statistical Properties of Community Structure in Large Social and Information Networks. In *Proc. of the 17th International World Wide Web Conference (WWW 2008)*, 2008.

[201] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–28, 2004.

[202] Kevin Lewis, Marco Gonzalez, and Jason Kaufman. Social Selection and Peer Influence in an Online Social Network. *Proceedings of the National Academy of Sciences (PNAS)*, 109(1):68–72, 2012.

[203] Jianghong Li, Margaret R. Weeks, Stephen P. Borgatti, Scott Clair, and Julia Dickson-Gomez. A Social Network Approach to Demonstrate the Diffusion and Change Process of Intervention From Peer Health Advocates to the Drug Using Community. *Substance Use & Misuse*, 47:474–490, 2012.

[204] Yang-Yu Liu, Jean-Jacques Slotine, and Albert-Laszlo Barabasi. Controllability of complex networks. *Nature*, 473:167–173, 2011.

[205] Susanne Lohmann. The Dynamics of Informational Cascades: The Monday Demonstrations in Leipzig, East Germany, 1989-91. *World Politics*, 47:502–534, 1994.

[206] R. Lopez. Is Obesity Contagious? *Expert Rev. Endocrinol. Metab.*, 3(1):21—22, 2008.

[207] Volodymyr V. Lysenko and Kevin C. Desouza. Cyberprotest in Contemporary Russia: The Cases of Ingushetiya.ru and Bakhmina.ru. *Technological Forecasting & Social Change*, 77:1179–1193, 2010.

[208] Volodymyr V. Lysenko and Kevin C. Desouza. The Role of Internet-Based Information Flows and Technologies in Electoral Revolutions: The Case of Ukraine's Orange Revolution. *First Monday*, 15, 2010.

[209] Volodymyr V. Lysenko and Kevin C. Desouza. Moldova's Internet Revolution: Analyzing the Role of Techologies in Various Phases of the Confrontation. *Technological Forecasting & Social Change*, 79:341–361, 2012.

[210] C. Macal and M. North. Tutorial on Agent-Based Modelling and Simulation. *J. of Simulation*, 4:151–162, 2010.

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

[212] Michael Macy. Chains of Cooperation: Threshold Effects in Collective Action. *American Sociological Review*, 56(6):730–747, 1991.

[213] Michael W. Macy. Backward-Looking Social Control. *American Sociological Review*, 58:819–836, 1993.

[214] Michael W. Macy. Social Learning and the Structure of Collective Action. In Edward J. Lawler, Barry Markovsky, Karen Heimer, and Jodi O'Brien, editors, *Advances in Group Processes*, pages 1–35. JAI Press, 1993.

[215] N. Madar, T. Kalisky, R. Cohen, D. Ben-Avraham, and S. Havlin. Immunization and Epidemic Dynamics in Complex Networks. *The European Physical Journal B*, 38:269–276, 2004.

[216] Thijs Markwat, Erik Kole, and Dick van Dijk. Contagion as a domino effect in global stock markets. *Journal of Banking & Finance*, 33:1996–2012, 2009.

[217] Peter V. Marsden and Noah E. Friedkin. Network Studies of Social Influence. *Sociological Methods & Research*, 22:127–151, 1993.

[218] Gonzalo Martin, Maria-Cristina Marinescu, David E Singh, and Jesus Carretero. Leveraging social networks for understanding the evolution of epidemics. *BioMedCentral Systems Biology*, 5:1–16, 2011.

[219] G. Marwell, P. Oliver, and R. Prahl. Social Networks and Collective Action: A Theory of the Critical Mass. III. *American Journal of Sociology*, 94:502–534, 1988.

[220] Winter A. Mason, Frederica R. Conrey, and Eliot R. Smith. Situating Social Influence Processes: Dynamic, Multidirectional Flows of Influence Within Social Networks. *Personality and Social Psychology Review*, 11:279–300, 2007.

[221] Michael Mathioudakis, Nilesh Bansal, and Nick Koudas. Identifying, Atributing, and Describing Spatial Bursts. In *Proceedings of the VLDB Endowment*, pages 1091–1102, 2010.

[222] Joseph B. McCormick and Susan Fischer-Hoch. *Level 4: Virus Hunters of the CDC*. Turner Publishing, 1996.

[223] Miller McPherson, Lynn Smith-Lovin, and James M. Cook. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, 27:415–444, 2001.

[224] Sergey Melnik, Jonathan A. Ward, James P. Gleeson, and Mason A. Porter. Multi-Stage Complex Contagions, 2011. appears as ArXiv arXiv:1111.1596.

[225] Mercken, Tom Snijders, Christian Steglich, Vartiainen, and de Vries. Dynamics of Adolescent Friendship Networks and Smoking Behavior. *Social Networks*, 32:72–81, 2010.

[226] Konstantin Mertsalov, Malik Magdon-Ismail, and Mark Goldberg. Models of Communication Dynamics for Simulation of Information Diffusion. In *Proceedings of Advances in Social Networks Analysis and Mining (ASONAM 2009)*, pages 44–49, 2008.

[227] T. Miller and D. Hendrie. Substance Abuse Prevention Dollars and Cents: A Cost-Benefit Analysis. Technical Report DHHS Pub. No. (SMA) 07-4298, Center for Substance Abuse Prevention, Substance Abuse and Mental Health Services Administration, Rockville, MD, 2008.

[228] M. Mobilia. Does a single zealot affect an infinite group of voters? *Physical Review Letters*, 91(2):028701–1–028701–4, 2003.

[229] M. Mobilia, A. Petersen, and S. Redner. On the role of zealotry in the voter model. *J. Statistical Mechanics: Theory and Experiment*, P08029:1–17, 2007.

[230] M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Journal Random Structures and Algorithms*, 6:161–179, 1995.

[231] J. Moody. Network Structure and Diffusion, 2009.

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

[233] E. Mossel and S. Roch. On the Submodularity of Influence in Social Networks. In *Proc. ACM STOC*, pages 128–134, 2007.

[234] David Murrugarra, Alan Veliz-Cuba, Boris Aguilar, Seda Arat, and Reinhard Laubenbacher. Modeling stochasticity and variability in gene regulatory networks. *EURASIP Journal on Bioinformatics and Systems Biology*, 2012:5, 2012.

[235] Robin L. Nabi. Anger, Fear, Uncertainty, and Attitude: A Test of the Cognitive-Functional Model. *Communication Monographs*, 69:204–216, 2002.

[236] G. L. Nemhauser, L. A. Wosley, and M. L. Fisher. An Analysis of Approximations for Maximizing Submodular Functions. *Mathematical Programming Study*, 14:265–294, 1978.

[237] M. Newman. The Structure of Scientific Collaboration Networks. *Proceedings of the Naitonal Academy of Sciences (PNAS)*, 98(2):404–409, 2001.

[238] M. Newman. The Structure and Function of Complex Networks. *SIAM Review*, 45:167–256, 2003.

[239] M. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences (PNAS)*, 103:85778582, 2006.

[240] M. E. J. Newman. Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. *Physical Review E*, 64:016132–1–016132–7, 2001.

[241] M. E. J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27:39–54, 2005.

[242] M. E. J. Newman. Threshold effects for two pathogens spreading on a network. *Physical Review Letters*, 95:108701–1–108701–5, 2005.

[243] M. E. J. Newman and Juyong Park. Why social networks are different from other types of networks. *Physical Review E*, 68:036122–1–036122–8, 2003.

[244] Nam P. Nguyen, Guanhua Yan, My T. Thai, and Stephan Eidenbenz. Containment of Viral Spread in Online Social Networks. In *Proceedings of the ACM Web Science Conference (WebSci 2012)*, June 2012.

[245] Jae Dong Noh and Heiko Rieger. Random walks on complex networks. *Physical Review Letters*, 92:118701–1–118701–4, 2004.

[246] Brendan O'Connor, Ramnath Balasubramanyan, Bryan R. Routledge, and Noah A. Smith. From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. In *Proceedings of the International AAAI Conference on Weblogs and Social Media*, pages 1915–1918, 2010.

[247] Office of National Drug Control Policy. The Economic Costs of Drug Abuse in the United States, 1992-2002. Technical Report Publication No. 207303, Executive Office of the President, Washington, DC, 2004.

[248] P. Oliver, G. Marwell, and R. Teixeira. A Theory of the Critical Mass. I. Interdependence, Group Heterogeneity, and the Production of Collective Action. *American Journal of Sociology*, 91(3):522–556, 1985.

[249] Pamela E. Oliver and Daniel J. Myers. Networks, Diffusion, and Cycles of Collective Action. In Mario Diani and Doug McAdam, editors, *Comparative Politics: Social Movements and Networks*, pages 173–203. Oxford University Press, 2003.

[250] J.-P. Onnela, J. Saramaki, J. Hyvonen, G. Szabo, D. Lazer, K. Kaski, J. Kertesz, and A.-L. Barabasi. Structure and tie strengths in mobile communication networks. *Proceedings of the National Academy of Sciences (PNAS)*, 104:7332–7336, 2007.

[251] Jukka-Pekka Onnela and Felix Reed-Tsochas. Spontaneous Emergence of Social Influence in Online Systems. *Proceedings of the National Academy of Sciences (PNAS)*, 107(43):18375–18380, 2010.

[252] Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, 32:245–251, 2010.

[253] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[254] David Scott Palmer. Introduction: History, Politics, and Shining Path in Peru. In David Scott Palmer, editor, *The Shining Path of Peru*, pages 1–32. St. Martin's Press, 1994.

[255] Keyao Pan. Understanding Original Antigenic Sin in Influenza with a Dynamical System. *PLoS ONE*, 6:e23910–1–e23910–13, 2011.

[256] Alfred Park and Richard Fujimoto. A scalable framework for parallel discrete event simulations on desktop grids. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, pages 185–192, Washington, DC, USA, 2007. IEEE Computer Society.

[257] Alfred Park and Richard M. Fujimoto. Efficient Master/Worker Parallel Discrete Event Simulation. In *Proceedings of the 23th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2009)*, pages 145–152, 2009.

[258] Jon Parker and Joshua Epstein. A Distributed Platform for Global-Scale Agent-Based Models of Disease Transmission. *ACM Transactions on Modeling and Computer Simulation (TOMACS 2011)*, 22, 2011.

[259] Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic spreading in scale-free networks. *Physical Review Letters*, 86:3200–3203, 2001.

[260] Nishith Pathak, Arindam Banerjee, and Jaideep Srivastava. A Generalized Linear Threshold Model for Multiple Contagions. In *Proceedings of the 10th IEEE Conference on Data Mining (ICDM 2010)*, pages 965–970, 2010.

[261] Kalyan Perumalla and Sudip Seal. 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 (PADS 2010)*, 2010.

[262] Kalyan S. Perumalla. μsik–A Micro-Kernel for Parallel/Distributed Simulation Systems. In *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation (PADS 2005)*, 2005.

[263] Kalyan S. Perumalla. Parallel and Distributed Simulation: Traditional Techniques and Recent Advances. In *Proceedings of the 2006 Winter Simulation Conference (WSC 2006)*, 2006.

[264] Kalyan S. Perumalla and Brandon G. Aaby. Data Parallel Execution Challenges and Runtime Performance of Agent Simulations on GPUs. In *2008 Spring Simulation MultiConference (SpringSim 2008)*, 2008.

[265] Kalyan S. Perumalla and Sudip K. Seal. Discrete Event Modeling and Massively Parallel Execution of Epidemic Outbreak Phenomena. *Simulation*, 88:768–783, 2012.

[266] Ricardo Pinho, Elhanan Borenstein, and Marcus W. Feldman. Most networks in wagner's model are cycling. *PLoS ONE*, 7:e34285–1–e34285–8, 2012.

[267] Phillip Porras, Linda Briesemeister, Keith Skinner, Karl Levitt, Jeff Rowe, and Yu-Cheng Allen Ting. A hybrid quarantine defense. In *Proceedings of the 2004 ACM CCS Workshop on Rapid Malcode (WORM 04)*, pages 73–82, 2004.

[268] B. Prakash, H. Tong, N. Valler, M. Faloutsos, and C. Faloutsos. Virus Propagation on Time-Varying Networks: Theory and Immunization Algorithms. In *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2010)*, pages 99–114, 2010.

[269] B. Aditya Prakash, Deepayan Chakrabarti, Michalis Faloutsos, Nicholas Valler, and Christos Faloutsos. Threshold Conditions for Arbitrary Cascade Models on Arbitrary Graphs. In *Proceedings of the 11th IEEE Conference on Data Mining (ICDM 2011)*, pages 537–546, 2011.

[270] Rami Puzis, Meytal Tubi, Yuval Elovici, Chanan Glezer, and Shlomi Dolev. A Decision Support System for Placement of Intrusion Detection and Prevention Devices in Large-Scale Networks. *ACM Transactions on Modeling and Computer Simulation (TOMACS 2011)*, 22:1–2, 2011.

[271] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. ISBN 3-900051-07-0.

[272] R. Raz and S. Safra. A Sub-Constant Error-Probability Low-Degree Test, and a Sub-Constant Error-Probability Characterization of NP. In *Proc. ACM Symp. on Theory of Computing (STOC 1997)*, pages 475–484, 1997.

[273] Timothy C. Reluga, Jan Medlock, and Alan S. Perelson. Backward bifurcations and multiple equilibria in epidemic models with structured immunity. *Journal of Theoretical Biology*, 252:155–165, 2008.

[274] M. Richardson, R. Agrawal, and P. Domingos. Trust Management for the Semantic Web. In *Proc. of the Second International Semantic Web Conference (ISWC 2003)*, pages 351–368, 2003.

[275] M. Richardson and P. Domingos. Mining Knowledge-Sharing Sites for Viral Marketing. In *Proc. ACM Intl. Conf. on Data Mining and Knowledge Discovery (KDD 2002)*, pages 61–70, 2002.

[276] Patrick F. Riley and George F. Riley. SPADES–A Distributed Agent Simulation Environment with Software-in-the-Loop Execution. In *Proceedings of the 2003 Winter Simulation Conference (WSC 2003)*, pages 817–825, 2003.

[277] Daniel Romero, Brendan Meeder, and Jon Kleinberg. Differences in the Mechanics of Information Diffusion Across Topics: Idioms, Political Hashtags, and Complex Contagion on Twitter. In *Proceedings of the 20th International World Wide Web Conference (WWW 2011)*, 2011.

[278] I. Rosenstock, V. Strecher, and M. Becker. Social Learning Theory and the Health Belief Model. *Health Education Quarterly*, 15(2):175–183, 1988.

[279] M. Salathe and J. Jones. Dynamics and Control of Diseases in Networks with Community Structure. *PLoS Computational Biology*, 6:e1000736–1–e1000736–11, 2010.

[280] T. Schank and D. Wagner. Approximating Clustering Coefficients and Transitivity. *Journal of Graph Algorithms and Applications*, 9(2):265–275, 2005.

[281] T. Schelling. *Micromotives and Macrobehavior*. W. W. Norton and Company, 1978.

[282] S. Seidman. Network Structure and Minimum Degree. *Social Networks*, 5:269–287, 1983.

[283] Michael Shapiro and Edgar Delgado-Eckert. Find the probability of infection in an SIR network is NP-Hard. *Mathematical Biosciences*, 240:77–84, 2012.

[284] M. Sheng, J. Li, and Y. Shi. Critical Nodes Detection in Mobile Ad Hoc Network. In *Proc. of the 20th International Conference on Advanced Information Networking and Applications (AINA 06)*, 2006.

[285] Susan G. Sherman, Donald S. Ganna, Karin E. Tobin, Carl A. Latkin, Christopher Welsh, and Peter Bielenson. The life they save may be mine: Diffusion of overdose prevention information from a city sponsored programme. *International Journal of Drug Policy*, 20:137–142, 2009.

[286] Xiaolin Shi, Jun Zhu, Rui Cai, and Lei Zhang. User grouping behavior in online forums. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2009)*, pages 777–786, 2009.

[287] Robert J. Shiller and John Pound. Survey Evidence on Diffusion of Interest and Information Among Investors. *Journal of Economic Behavior and Organization*, 12:47–66, 1989.

[288] Ilya Shmulevich, Edward R. Dougherty, Seungchan Kim, and Wei Zhang. Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18:261–274, 2002.

[289] Haroon Siddique, Paul Owen, and Adam Gabbatt. Protests in Egypt and Unrest in the Middle East–As It Happened. In *The Guardian*. 2011.

[290] D. Siegel. Social networks and collective action. *Americal Journal of Political Science*, 53:122–138, 2009.

[291] D. Siegel. When does repression work?: Collective action and social networks. *Journal of Politics*, 73:993–1010, 2011.

[292] David A. Siegel. Non-disruptive tactics of suppression are superior in countering terrorism, insurgency, and financial panics. *PLoS ONE*, 6:e18545–1–e18545–6, 2011.

[293] Tom Snijders, Christian Steglich, and Gerhard van de Bunt. Introduction to Actor-Based Models for Network Dynamics. *Social Networks*, 32:44–60, 2010.

[294] Erick Stattner, Martine Collard, and Nicolas Vidot. D2SNet: Dynamics of diffusion and dynamic human behavior in social networks. *Computers in Human Behavior*, 29:496–509, 2013.

[295] D. Stauffer and M. Sahimi. Can a Few Fanatics Influence the Opinion of a Large Segment of a Society? *European Phys. J. B*, 57:146–152, 2007.

[296] P. Stearns. *1848: The Revolutionary Tide in Europe*. W. W. Norton & Company, New York, NY, 1974.

[297] Christian Steglich, Tom Snijders, and Michael Pearson. Dynamic Networks and Behavior: Separating Selection from Influence. *Sociological Methodology*, 40:239–392, 2010.

[298] R. Sturm. The Effects Of Obesity, Smoking, And Drinking On Medical Problems And Costs. *Health Affairs*, 21(2):245–253, 2002.

[299] Siddharth Suri and Sergei Vassilvitskii. Counting Triangles and the Curse of the Last Reducer. In *Proceedings of the 20th International World Wide Web Conference (WWW 2011)*, pages 607–614, 2011.

[300] Taro Takaguchi, Naoki Masuda, and Petter Holme. Bursty communication patterns facilitate speading in a threshold-based epidemic dynamics, 2012. Appears as arXiv.org:1206.2097v.

[301] Yarong Tang, Kalyan Perumalla Richard Fujimoto, Homa Karimabadi, Jonathan Driscoll, and uri Omelchenko. Optimistic Parallel Discrete Event Simulation of Physical Systems Using Reverse Computations. In *Proceedings of the 19th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2005)*, 2005.

[302] C. Tantipathananandh, T. Y. Berger-Wolf, and D. Kempe. A Framework for Community Identification in Dynamic Social Networks. In *Proc. ACM Intl. Conf. on Data Mining and Knowledge Discovery (KDD 2007)*, pages 717–726, 2007.

[303] M. Taylor. Rationality and Revolutionary Collective Action. In M. Taylor, editor, *Rationality and Revolution*, pages 63–97. Cambridge University Press, 1988.

[304] M. Taylor. Structure, Culture and Action in the Explanation of Social Change. *Politics & Society*, 17(2):115–162, 1989.

[305] Leonard Thompson. *A History of South Africa*. Yale University Press, New Haven, CT, 3 edition, 2001.

[306] Seth Tisue and Uri Wilenski. NetLogo: A Simple Environment for Modeling Complexity. In *International Conference on Complex Systems*. Appears as http://ccl.northwestern.edu/papers/netlogo-iccs2004.pdf., 2004.

[307] Karin Elizabeth Tobin, Satoko Janet Kuramoto, Melissa Ann Davey-Rothwell, and Carl Asher Latkin. The step into action study: a peer-based, personal risk network-focused hiv prevention intervention with injection drug users in baltimore, maryland. *Addiction*, 106:366–375, 2011.

[308] H. Tong, B. Prakash, C. Tsourakakis, T. Eliassi-Rad, C. Faloutsos, and D. Chau. On the Vulnerability of Large Graphs. In *Proceedings of the 10th IEEE Conference on Data Mining (ICDM 2010)*, pages 1091–1096, 2010.

[309] Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. Gelling, and Melting, Large Graphs by Edge Manipulation. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM 2012)*, pages 245–254, 2012.

[310] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

[311] Daniel Trpevski, Wallace K. S. Tang, and Ljupco Korcarev. Model for rumor spreading over networks. *Physical Review E*, 81:056102–1–056102–11, 2010.

[312] Jason Tsai, Emma Bowring, Stacy Marsella, and Milind Tambe. Empirical evaluation of computational emotional contagion models. In *Proceedings of the 11th International Conference on Intelligent Virtual Agents (IVA 2011)*, 2011.

[313] Jason Tsai, Thanh H. Nguyen, and Milind Tambe. Security games for controlling contagion. In *Proceedings of the National Conference on Artificial Intelligence (AAAI 2012)*, 2012.

[314] Gaurav Tuli, Madhav Marathe, S. S. Ravi, and Samarth Swarup. Addiction Dynamics May Explain the Slow Decline of Smoking Prevalence. In *Proceedings of the 2012 International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction (SBP12)*, 2012.

[315] Andranik Tumasjan, Timm O. Sprenger, Philipp G. Sanders, and Isabell M. Welpe. Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment. In *Proceedings of the International AAAI Conference on Weblogs and Social Media*, pages 178–185, 2010.

[316] Johan Ugander, Lars Backstrom, Cameron Marlow, and Jon Kleinberg. Structural Diversity in Social Contagion. *Proceedings of the Naitonal Academy of Sciences (PNAS 2012)*, 109(9):5962–5966, 2012.

[317] S. Vadhan. The Complexity of Counting in Sparse, Regular and Planar Graphs. *SIAM J. Comput.*, 31(2):398–427, 2001.

[318] Thomas W. Valente. *Social Networks and Health*. OxFord University Press, 2010.

[319] Thomas W. Valente. Network Interventions. *Science*, 337:49–53, 2012.

[320] Thomas W. Valente, Beth R. Hoffman, Annamara Ritt-Olson, Kara Lichtman, and C. Anderson Johnson. Effects of a social-network method for group assignment strategies on peer-led tobacco prevention programs in schools. *American Journal of Public Health*, 93:1837–1843, 2003.

[321] Thomas W. Valente and David Vlahov. Selective risk taking among needle exchange participants: Implications for supplemental interventions. *American Journal of Public Health*, 91:406–411, 2001.

[322] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[323] Alessandro Vespignani. Modelling dynamical processes in complex socio-technical systems. *Nature Physics*, 8:32–39, 2012.

[324] D. Volovik and S. Redner. Dynamics of Confident Voting. *arXiv:1111.3883v1 [physics.soc-ph]*, 2011.

[325] Andreas Wagner. Evolution of Gene Networks by Gene Duplications: A Mathematical Model and its Implications on Genome Organization. *Proceedings of the National Academy of Sciences (PNAS)*, 91:4387–4391, 1994.

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

[327] Yang Wang, D. Chakrabarti, Chenxi Wang, and C. Faloutsos. On the Vulnerability of Large Graphs. In *Proceedings of the 22nd International Symposium on Reliable Distributed Systems (RDS 2003)*, pages 25–34, 2003.

[328] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[329] D. Watts. A Simple Model of Global Cascades on Random Networks. *PNAS*, 99:5766–5771, 2002.

[330] Duncan J. Watts and Steven H. Strogatz. Collective Dynamics of 'Small-World' Networks. *Nature*, 393:440–442, 1998.

[331] Douglas B. West. *An Introduction to Graph Theory*. Prentice Hall, Upper Saddle River, NJ, 2001.

[332] Stan Wilson. Riot anniversary tour surveys progress and economic challenges in Los Angeles. In *Cable New Network (CNN)*. 2012.

[333] Peter Wittek and Xavier Rubio-Campillo. Scalable agent-based modelling with cloud HPC resources for social simulations. In *4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2012)*, pages 355–362, 2012.

[334] J. Xie, J. Emenheiser, M. Kirby, S. Sreenivasan, B. Szymanski, and G. Korniss. Evolution of Opinions on Social Networks in the Presence of Competing Committed Groups. *PLoS One*, 7:e33215–1–e33215–9, 2012.

[335] Amos Yadlin, Oded Eran, Shimon Stein, Zvi Magen, Yoram Schweitzer, Shlomo Brom, Ephraim Kam, Yeol Guzansky, Benedetta Berti, Gallia Lindenstrauss, David Friedman, Ephraim Asculai, Emily B. Landau, Anat Kurz, Gabi Siboni, and Mark Heller. One year of the arab spring: Global and regional implications. Technical report, The Institute for National Security Studies, 2012.

[336] Jian Yang and David A. Bessler. Contagion Around the October 1987 Stock Market Crash. *European Journal of Operation Research*, 184:291–310, 2008.

[337] Tracy Yang and Jamus Jerome Lim. Crisis, Contagion, and East Asian Stock Markets. *Review of Pacific Basin Financial Markets and Policies*, 7:119–151, 2004.

[338] E. Yildiz, D. Acemoglou, A. Ozdaglar, A. Saberi, and A. Scaglione. Discrete opinion dynamics with stubborn agents. *OPRE-2011-01-026*, 2011.

[339] C. Yin. Equilibria of collective action in different distributions of protest threshold. *Public Choice*, 97:536–567, 1999.

[340] Teruhiko Yoneyama, Sanmay Das, and Mukkai Krishnamoorthy. A hybrid model for disease spread and an application to the sars pandemic. *Journal of Artificial Societies and Social Simulation*, 2011. to appear.

[341] Hai-Feng Zhang, Ke-Zan Li, Xin-Chu Fu, and Bing-Hong Wang. An efficient control strategy of epidemic spreading on scale-free networks. *Chinese Physical Review Letters*, 26:068901–1–068901–4, 2009.

[342] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 6 edition, 2005.

Appendix A

A Selection of Models Implementable With ENSIM

In this appendix, we provide classes of models that can be implemented with ENSIM. These supplement the contagion categories (and models) in Table 1.4. This suggests the versatility provided by GDS. We have attempted to list each reference only once, unless the paper has multiple models. In the latter case, each model is listed only once. Many models could be placed in multiple categories, but we choose only one. ***Many of these models are used in multiple works, but we cite only one paper, so as not to bloat the references.*** Thus, by surveying the references, one gets a sense of the wide range of models that can be implemented with ENSIM.

Table A.1: Social sciences models.

Model	**Description**
Collective action, ratcheted thresholds.	Economics (e.g., purchasing high-cost items); crowd behavior (e.g., strikers, leaving a social occasion); civil disobedience; politics [67, 105, 108, 290].
Mass movements.	Revolutions come about owing to subtle, latent changes in a population; social movements; spread of influence and protest in Spain, May 2011 [127, 188].
Independent cascades.	Social behavior (e.g., adoption of fads, peer pressure) [166].
Threshold.	Diffusion of innovations [80, 224, 290, 329].
Independent cascades, linear threshold.	Social diffusion (e.g., fads, rumors) [166, 167, 172, 277].
Diffusion.	How alarming warnings are spread through a population [151]. Effect of (information) diffusion for dynamic networks [226]. Spread of information via internet communication [133, 176]. Diffusion of trust [5, 134].
Blocking diffusion.	Blocking the spread of social contagions such as fads or marketing campaigns; time-varying networks [135], thresholds [180], and blocking nodes [7, 65, 183, 229, 338].
Learning models of collective action.	Past experience and observable outcomes guide future actions [212–214].
Information growth on internet.	Model of how information grows for a topic in Wikipedia [96].
Repression.	Models of form $b_i + c_{i,t} > 0$. This particular model also incorporates fear and anger [291].
Transitions based soley on probabilities of transition.	Probability changes are based on local structure [249].
State of a vertex is a real number and is the average of states of neighbors.	An example of a non-threshold state transition [129].
Multi-state (i.e., $>$ 2 states).	Agents can be in more states than just 0 and 1 [178, 224].
History-dependent models.	Models based on observations; bursty behavior [127].
Emotion-based models.	These models can incorporate time histories [149, 312].
Ideologies.	Multimechanism models [179].
Voter models.	Nodes update their state by choosing a state (or averaging or taking the majority of states) among neighbors [18, 184, 185, 224, 311, 324]. Time-dependent voter model [169].
Competing contagions.	Propagate a second contagion to intercept an initial contagion [50, 57, 76, 77, 142, 163, 177, 244].
Vector-valued vertex functions.	Can be used for multi-mechanism state transitions [179,191].

Table A.2: Models of computer network behaviors.

Model	**Description**
Spread of malware.	Propagation of malware through a network of wireless devices [75].
Intrusion detection systems (IDS).	Modeling computer network intrusion detection systems [270].
Blocking malware.	Use of blocking nodes to stop malware [54, 267].
Messaging algorithms.	Trickle algorithm for energy-efficient messaging [201].

Table A.3: Biological models.

Model	**Description**
Biology.	Identifying biological functions of genes [160].
Ecology.	Ecological community assembly in plant-pollinator models [59].
Cell-based immunology.	Cells transition through states defined by dynamical system and new cells are produced and existing cells may die. Differential equation model [137]. Cells-based model including memory. Differential equation model [255].
Regulatory networks.	Gene regulatory networks [234].
Cell cycle.	Model of budding yeast cell cycle modeling using Boolean networks [155].
Gene network mutations.	Wagner model [266, 325].

Table A.4: Epidemiological models.

Model	**Description**
SEIR and SIR types.	Virus diffusion in humans and other animal populations [27,136,258]. These include probabilistic timed transition systems (PTTS) [38].
Multiple levels of infection.	Models buildup of immunity after contracting and then recovering from a virus [273].
Modeling disease and fear.	This is one FSM, not two different contagions; models flight caused by fear (not sequestration) [109].
Hybrid epidemiological model.	The model uses an agent-based model to simulate disease transmission between countries; countries are represented as nodes. Within country disease transmission at each node is modeled as a differential-equation based SEIR model [340].

Table A.5: Language, linguistics models.

Model	**Description**
Language evolution.	Spread and evolution of languages [60].

Table A.6: Economic and financial models.

Model	**Description**
Economics and electric power.	Market power of suppliers of electricity [19].

Table A.7: Mathematical models and behavior of graph dynamical systems.

Model	**Dscription**
Cellular automata.	Higher order cellular autamata [22].
Sequential dynamical systems.	Sequential dynamical systems with various update schemes such as sequential, synchronous, block sequential, fair and unfair word orderings, fair and unfair block orderings [233].
Bithreshold systems.	Thresholds for up and down state transitions [187].
Multithreshold model.	Any number of states with threshold transitions [125].
Genetic algorithm diffusion model.	Combines genetic algorithms and hyperplane-defined functions [192].
Dimer automaton.	Pair-wise vertex update functions applied to vertices of a (dyadic) edge [16].
Probabilistic Boolean networks.	[191].

Appendix B

Blocking Results Beyond Homogeneous Thresholds

Many of the results presented in Section 5.3 were for the case of homogeneous absolute thresholds. We now briefly indicate how the results can also be extended to other forms of thresholds defined in Section 5.2.3.

We note that heterogeneous thresholds (absolute and relative) are a special case of the corresponding form of homogeneous thresholds. Therefore, any hardness result that holds for homogeneous thresholds also holds for heterogeneous thresholds. We use this fact several times in the following description. The statements and proofs of all the lemmas and theorems referenced below appear in Section 5.3.

1. Lemma 19 holds for both homogeneous and heterogeneous absolute thresholds. With a minor modification to the proof, the result can be shown to hold for homogeneous and heterogeneous relative thresholds as well. The modification is that when the counter value for a node indicates that the node has reached its relative threshold, the node is added to the list of affected nodes.

2. Theorem 20 holds for homogeneous absolute thresholds and hence for heterogeneous absolute thresholds. To see that it holds for relative thresholds as well, we note that the absolute threshold value t for a node v can be thought of as the relative threshold value of $t/\deg(v)$, where $\deg(v)$ is the number of neighbors of v (including v itself). The same approach can be used to show that Theorems 21, 22 and 23 hold for heterogeneous absolute thresholds and both forms of relative thresholds.

3. It was pointed out in Section 5.3 that the approximation algorithm for the SCS-SASN problem actually works for heterogeneous absolute thresholds. Hence, it works for homogeneous absolute thresholds as well. To see that it also works for relative thresholds, we first convert a relative threshold value of τ for a node v to the absolute threshold value $\lceil \tau * \deg(v) \rceil$

and then use the given approximation algorithm.

Appendix C

Network Structure Study

The organization of this appendix is as follows. First, we motivate the graph construction methods used in our work by pointing out that standard methods that do not control for clustering coefficient will yield average clustering coefficient (CC_{ave}) values that are far smaller than the 0.25 value that we desire, in order to match the values used in [65]. Next, we describe well-known graph construction procedures for generating networks with the required average degree (d_{ave}) and CC_{ave} values. We provide values for the structural parameters of the networks used in our simulations. Finally, we elaborate on the experimental test matrix of Table 6.1. Strictly speaking, we present in this paper results for the exponential-decay and scale-free growth model graphs and the scale-free configuration model graphs. However, we also address here the configuration model exponential-decay graphs because it might be beneficial for any group wishing to reproduce our results or use these techniques in their work.

To understand the limitations of known graph generation methods in controlling the value of CC_{ave}, we generated a total of 4000 networks using a preferential attachment growth model for scale-free graphs, a random attachment growth model for exponential-decay graphs, a configuration model for scale-free graphs, and a configuration model for exponential-decay graphs. For the set of 1000 graphs in each of the four categories where $d_{ave} = 4$, the average clustering coefficients were as follows: 0.00458 for preferential attachment graphs, 0.000529 for random attachment graphs, 0.101 for scale-free configuration model graphs, and 0.000667 for exponential-decay configuration model graphs. Clearly, these values are too small, and thus motivate methods for augmenting the graph construction methods that control only for d_{ave}.

We generated the growth-model graphs first using an augmented approach and then generated the graphs based on the configuration model. The reason is that the configuration model allows for better control of the degree distribution of a graph. Once the growth-model networks were generated, curve fits to the degree distributions gave us the functional forms of the produced distributions. We then used these parameters in the configuration model

scheme. Even with this method, it was necessary to iterate on the form for the scale-free distribution to arrive at the desired average degree. This iterative procedure enabled us to produce configuration model graphs that are close in degree distribution to those generated with growth models.

C.1 Graph Generation Methods

C.1.1 Preferential Attachment Growth Model to Produce Scale-Free Graphs

A graph is constructed by combining the approaches in [25] and [280]. The procedure is provided in Figure C.1. (A similar method is described in [147].) In describing this procedure, we assume that the parameters n_0 (the number of initial nodes) and n (the total number of nodes) are given.

1. Let $S = \{v_1, \ldots, v_{n_0}\}$. For each node in S, add a self-loop. The resulting graph has n_0 nodes, edges, and components.
2. For $j = n_0 + 1,\ n_0 + 2,\ \ldots,\ n$ do
   (a) Add node v_j to S.
   (b) Choose m_j, the number of edges incident on v_j. (**Note:** The choice of values for m_j is discussed in the text.)
   (c) Each of these m_j edges is attached to a pre-existing node u from among $v_1, \ldots, v_{j-1} \in S$ according to a *scheme* discussed in the text. (When a node u is chosen, it is removed from S so that the graph remains simple.)
   (d) If $m_j \geq 2$, so that v_j has at least two neighbors, then add e_j edges between neighbors of v_j uniformly at random, where $0 \leq e_j \leq \binom{m_j}{2}$.
3. Remove the n_0 self-loop edges introduced in Step (1).

Figure C.1: Procedure for generating a scale-free degree distribution using a growth model.

We now explain the use of the procedure in Figure C.1 for our particular networks. We start out with $n_0 = 2$ nodes, denoted by v_1 and v_2. Each node has a self loop. We add a new node v_j and either one edge or two edges incident on v_j (i.e., $m_j \in \{1, 2\}$). The probability $p_a(u)$ of attaching v_j to a node $u \in S$ is proportional to the current degree of u. That is, the scheme mentioned in Step 2(c) of Figure C.1 for scale-free networks is based on the probability value $p_a(u)$ given by

$$p_a(u) = d_u / \left(\sum_{k=1}^{j-1} d_k \right). \tag{C.1}$$

Here, $j-1$ is the number of nodes not counting v_j, and d_u is the current degree of node u. Thus, nodes with larger degrees have a greater chance to be joined to the new node v_j, generating a power law degree distribution. This is the procedure described in [25], except that as part of our goal to achieve a desired clustering coefficient, we do not always add the same number of new edges for each v_j. Determining when to add one edge or two edges is a trial-and-error process. We now indicate how this procedure can be further augmented with that in [280] to increase the average clustering coefficient of the graph.

Suppose two edges are added from a new node v_j to pre-existing nodes u_k and u_l. Then the addition of the edge $\{u_k, u_l\}$ completes the triangle with v_j and therefore increases the clustering coefficient for node v_j. (Clustering coefficients for u_k and u_l will also change, in general.) Determining when to add edges that complete triangles is also a trial-and-error process. Note also that by adding the edge $\{u_k, u_l\}$, we alter the degree distribution, so there is interaction between obtaining a desired average degree and average clustering coefficient, adding further to the trial-and-error nature of the graph construction. The description thus far has been for adding one node v_j to the current graph. We repeat this procedure, for each value of v_j from 3 to 10000, resulting in a 10000-node graph. Each added node represents one iteration of graph construction.

C.1.2 Random Attachment Growth Model to Produce Exponential-Decay Graphs

This method is very similar to the one in Section C.1.1, following [25], except that instead of using Equation (C.1) to identify the node(s) u to which new node v_j will have edge(s), u is determined uniformly at random from all nodes that exist just before v_j is added. As before, the clustering coefficient is altered with the trial-and-error process of [280].

C.1.3 Configuration Model Graphs with Scale-Free Degree Distributions

The configuration model [230, 238] uses a specified degree distribution in an otherwise predetermined set of steps to generate a graph with the specified distribution. An interesting feature of the configuration model is that each possible graph instance is equally likely to occur for one execution of the model, and hence there is no bias in the produced graphs [238]. For a scale-free network, we seek a power-law degree distribution of the form

$$p(d) = c_1 d^{-c_2} \tag{C.2}$$

where $p(d)$ is the probability of generating a node with degree d, and c_1 and c_2 are constants. The configuration model procedure is shown in Figure C.2, where n denotes the number of

nodes.

At this point, we have a scale-free network, but it may not have the requisite average clustering coefficient. Hence, we perform the following steps, which essentially add triangles at random to the graph to increase the clustering coefficient. Doing so in the fashion described in Figure C.2 preserves the property that every graph instance is equally likely to occur.

1. Start with the graph G from the configuration model procedure.
2. Compute the clustering coefficient for each node of G, and the average clustering coefficient CC_{ave} of G.
3. Given n nodes in a graph, let $n_t = \lfloor n/3 \rfloor$. Construct a set T with n_t triangles by selecting 3 nodes uniformly at random and forming a triangle with them, and repeating this process a total of n_t times.
4. While (CC_{ave} of G is less than the required value) do
   (a) Select a triangle uniformly at random from T and add it to G. (Edges of the triangle that are already in G are not added.)
   (b) If the previous step added at least one new edge to G, then recompute CC_{ave} of G.

Figure C.2: Procedure for modifying a configuration model-generated graph to achieve a prescribed average clustering coefficient.

We make one final comment. In adding the triangles at random (Step 4(a) of Figure C.2), we found that we could not consistently achieve an average clustering coefficient of 0.25; that is why we reduced the value to 0.12. By *selectively* forming triangles, it is possible to achieve a value of 0.25; however, we wanted to preserve the property of the configuration model that each graph instance is equally likely to occur, so we accept a smaller CC_{ave} that is still representative of social networks [238].

C.1.4 Configuration Model Graphs with Exponential-Decay Degree Distributions

The steps in the process of generating graphs with the configuration model are the same as those of the preceding section, except the form of the degree distribution changes to that for an exponential-decay graph. The probability of a node having degree d is now given by

$$p(d) = c_3 \exp(-c_4 d) \tag{C.3}$$

where $p(d)$ is the probability of generating a node with degree d, and c_3 and c_4 are constants.

C.2 Networks With Controlled Average Degree and Clustering Coefficient

C.2.1 Preferential Attachment Scale-Free Graphs.

For the procedures of Section C.1.1, the variables in Table C.1 describe how we specify the number of new edges m_j for each node v_j and how we specify whether a new edge ($\{u_k, u_l\}$) is added to complete the triangle for v_j. The parameter o (taken from [280]) determines whether we generate edge $\{u_k, u_l\}$; here, $o = 1$ ($o = 0$) means that the edge is (is not) generated. The parameter ξ used in Table C.1 corresponds to the parameter d in [280][1]; it determines the number of new edges for each node v_j. So, for the first 8 iterations, we add only one new edge, for the next 20 iterations, 2 edges are added per node, and the process repeats. At the end of this process, we remove the two self-loops specified at the start of the process, so that we obtain simple graphs (no loops and at most one edge between any pair of nodes). We generate multiple graph instances simply by rerunning the scheme, allowing the stochasticity inherent in Equation (C.1) to generate new graph instances. As will be seen, the graphs generated are significantly different from each other.

Table C.1: Parameters for generating preferential attachment scale-free networks with prescribed d_{ave} and CC_{ave}.

Parameter	**Value**
o parameter low in algorithm	0
o parameter high in algorithm	1
Number of consecutive iterations o parameter is low	25
Number of consecutive iterations o parameter is high	20
The number of new edges per node ξ low	1
The number of new edges per node ξ high	2
Number of consecutive iterations for which ξ is low	8
Number of consecutive iterations for which ξ is high	20

C.2.2 Random Attachment Exponential-Decay Graphs.

For the procedures of Section C.1.2, the parameters for the model are given in Table C.2. Once again, the 30 graph instances are generated by fixing all values as in Table C.2, with

[1]In this paper, we use ξ instead of d, since d is used for node degrees.

stochasticity entering through the random selection of nodes with which node v_j forms edges.

Table C.2: Parameters for generating random attachment exponential-decay networks with prescribed d_{ave} and CC_{ave}.

Parameter	**Value**
o parameter low in algorithm	0
o parameter high in algorithm	1
Number of consecutive iterations o parameter is low	24
Number of consecutive iterations o parameter is high	35
The number of new edges per node ξ low	1
The number of new edges per node ξ high	2
Number of consecutive iterations for which ξ is low	2
Number of consecutive iterations for which ξ is high	2

C.2.3 Configuration Model Scale-Free Graphs.

Using the procedures of Section C.1.3, the parameters c_1 and c_2, shown in Table C.3, were selected based on curve fits to the scale-free networks produced from Section C.1.1.

Table C.3: Constants for generating scale-free degree distributions with the configuration model.

Parameter	**Value**
c_1	2.823
c_2	2.085

C.2.4 Configuration Model Exponential-Decay Graphs.

Using the procedures from Section C.1.4, the parameters c_3 and c_4, shown in Table C.4, were selected based on curve fits to the exponential-decay networks produced from Section C.1.2.

Table C.4: Constants for generating exponential-decay degree distributions with the configuration model.

Parameter	Value
c_3	0.3335
c_4	0.4500

C.3 Differences Among Generated Graphs Within One Class

For each graph class, our goal is to generate 30 representative instances. To quantify the difference between pairs of graphs, we use the *normalized symmetric difference* measure, Δ, which can be defined as follows. Suppose $G_A(V, A)$ and $G_B(V, B)$ are two graphs on the same node set V but with different edge sets A and B. Then,

$$\Delta(A, B) = \frac{|A \cup B| - |A \cap B|}{|A| + |B|}. \tag{C.4}$$

Here, the numerator is the size of the symmetric difference of the edge sets A and B (i.e., the number of edges which are either in A or in B but not in both). Thus, the normalized symmetric difference is the ratio of the size of the symmetric difference to the total number of edges in both graphs. When $\Delta = 1$, the two graphs G_A and G_B have no common edges and hence are different, and when $\Delta = 0$, the two graphs are identical. Table C.5 provides the minimum value of Δ for any pair of graphs from the 200 instances of each graph class. As the minimum Δ values are near 1, the graphs are dissimilar.

Table C.5: Minimum value of normalized symmetric difference for each graph class. These results show that each instance within a set of 200 instances is significantly different from all other instances.

Preferential Attachment Scale-Free	Random Attachment Exponential-Decay	Configuration Model Scale-Free	Configuration Model Exponential-Decay
0.990	0.996	0.969	0.999

C.4 Selected Properties of 30 Graph Instances for Each Graph Class

There are differences in the sizes of the largest connected component of each graph. For the growth models, the construction methods produce graphs with only one connected component with high probability. (This has been confirmed by analyzing the graphs. Only one scale-free preferential attachment network had two components, one with 9,999 nodes and the other with one node. In this case, due to the stochastic nature of the construction, node 1 did not form an edge with any of the other 9,999 nodes.) For the configuration model graphs, however, one cannot guarantee connectedness because pairs of nodes are selected randomly to form edges. In fact, none of the generated configuration model graphs contains a giant component with all 10,000 nodes. Table C.6 shows that the average number of components for the exponential-decay graphs is around 34, and that there is an order of magnitude more components, on average, for the scale-free graphs. However, even though there are many components, each graph has a giant component, as seen in the table (the remaining connected components are very small). The minimum number of nodes in the giant component of the 30 exponential-decay graphs is 9,898 nodes; for the 30 scale-free graphs, the minimum size of the giant component is 9,141 nodes.

Table C.6: Average and standard deviation of the number of connected components and size of the giant component over 30 instances for each of the configuration model graph construction methods.

Graph Type	**Number of Connected Components**		**Size of Giant Component**	
	Average	**Std. Dev.**	**Average**	**Std. Dev.**
scale-free	308.3	55.29	9365.6	114.3
exponential-decay	34.37	7.289	9930.4	15.65

In our simulations, we treat the giant component as the graph itself. That is, all seed nodes and all failed nodes are chosen from the giant component. In effect, the giant component *becomes* the graph. This eliminates the possibility of selecting seed nodes and failed nodes from a smaller connected component; thus, differences in behaviors among graphs are due to differences in structures within the giant components. Note that the giant component is the component produced by the graph construction method and does not include the effects of the failed nodes. In particular, failed nodes can cause a giant component to be broken up into smaller components, but this is again a function of the structure of the giant component itself.

To evaluate the variability in graph structure on the simulation results, 30 graph instances of each type were randomly chosen (from the ensemble of 200 instances) for our simulations. The average degree and average clustering coefficient, as well as the corresponding standard deviations σ, are given for each graph class in Table C.7. The standard deviations, for the

most part, are greater for the configuration model.

Table C.7: Average and standard deviation of average degree d_{ave} and clustering coefficient CC_{ave} over 30 instances for the graphs generated using growth models and configuration models. The first table contains degree data and the second contains clustering coefficient data. All graphs have a nominal d_{ave} value of 4. Nominal CC_{ave} values 0.24 and 0.12 were used with the growth models and the configuration models respectively, as described in the text.

Graph Type	**Growth Model**		**Configuration Model**	
	d_{ave}	σ_d	d_{ave}	σ_d
scale-free	4.13	0.00239	4.00	0.177
exponential-decay	4.13	0.000458	4.05	0.0313

Graph Type	**Growth Model**		**Configuration Model**	
	CC_{ave}	σ_{CC}	CC_{ave}	σ_{CC}
scale-free	0.239	0.00220	0.122	8.05×10^{-3}
exponential-decay	0.241	0.00163	0.120	5.30×10^{-5}

Degree distributions for each class of graph are given in Figure C.3. Each plot contains degree distributions for 30 graph instances. The scale-free or power-law degree distributions have thicker tails, which is often observed in these types of networks; see, for example the `epinions`, `slashdot08-11`, and `wikipedia` networks provided at [156]; see also [134].

C.5 Test Plan

Simulation parameters of the full factorial design of this study are given in Table 6.1. Here, using Figure C.4, we make explicit the sequence of steps taken to generate the graphs and simulation conditions.

Moving left to right in Figure C.4, the two graph classes and two construction methods combine to generate four graph types, with 30 instances per type. (Note that numbers inside ovals signify replicates. It is not possible to show every branch in our process owing to the number of parameters and replicates, and hence we collapse many branches in this diagram via ovals.) Each graph instance has $n = 10,000$ nodes. To each graph instance, we apply random and targeted failed node approaches. For each approach we specify six values of failed node fraction f, from $f = 10^{-5}$ (which means zero failed nodes) to $f = 0.2$. For each f, 20 sets of failed nodes are produced and applied to each graph instance, thereby multiplying the number of instances. This process produces all of the graph instances. For each resulting graph instance, 20 sets of seed nodes are produced (see the beginning of Section **??** for a description of the scheme), thereby multiplying the number of sets of initial conditions for simulation instances. Further, for each set of seed nodes, two diffusion

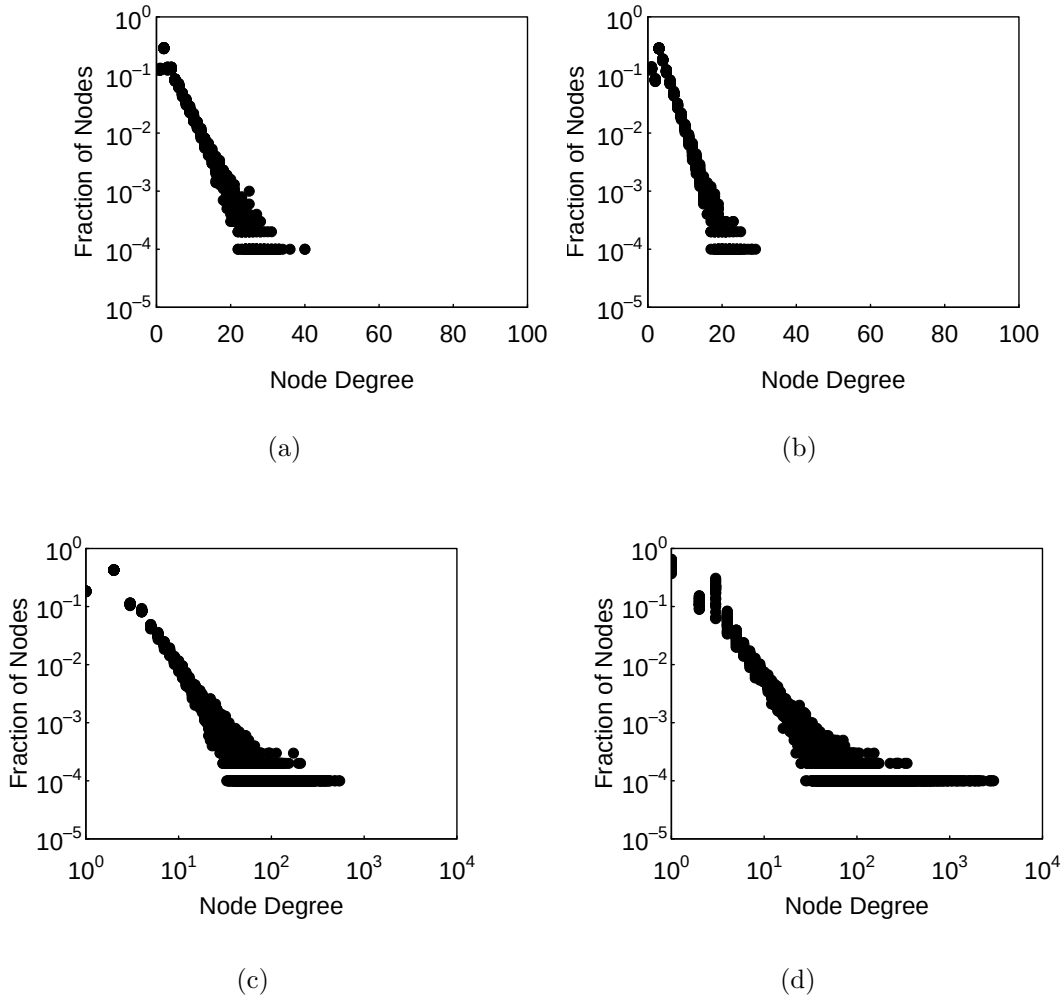


Figure C.3: Degree distributions for the four network classes. Each plot contains distributions for 30 graphs. (a) Growth model, random attachment, exponential-decay. (b) Configuration model, exponential decay. (c) Growth model, preferential attachment, scale-free. (d) Configuration model, scale-free. The plots for exponential-decay networks appear as straight lines on linear-logarithmic axes and those for scale-free graphs appear as straight lines on log-log axes.

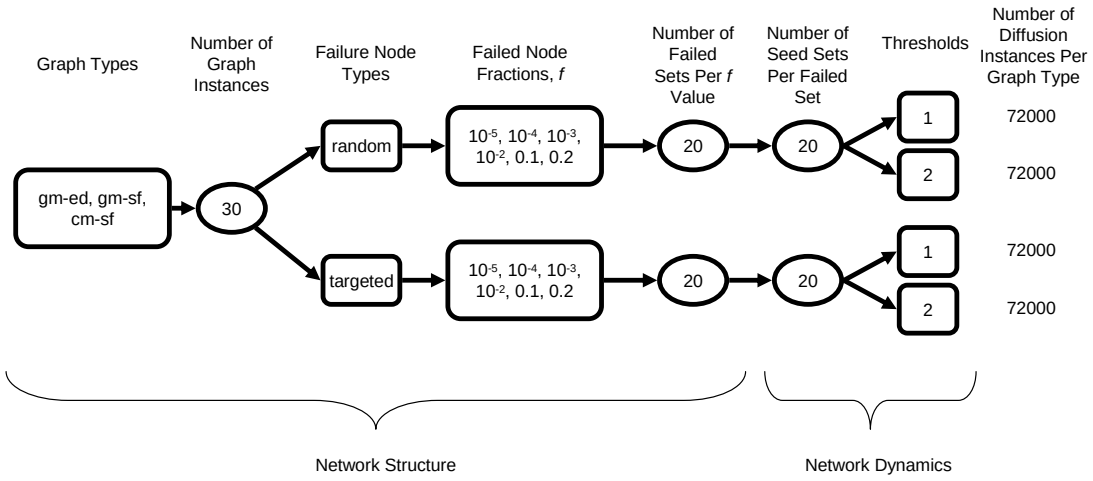


Figure C.4: Sequence of operations for generating graphs and initial conditions for the 21,600 giant component determinations and 864,000 simulation instances.

instances are produced using $t = 1$ (simple diffusion, SD) and $t = 2$ (complex diffusion, CD). Hence, for one graph type, we have 72,000 diffusion instances across one type of failed node, all values of f, multiple sets of failed and seed nodes, and one threshold. For both types of failed nodes and both thresholds, there are 288,000 diffusion instances. Multiplying by the four graph types results in 864,000 diffusion instances for the study.

Appendix D

Bithreshold Synchronous Dynamical Systems

In this appendix, complexity issues for bithreshold synchronous dynamical systems are studied. These results are part of the work of Chapter 7 in characterizing these systems.

D.1 Problem Formulations and Other Definitions

D.1.1 Computational Problems for BT-SyDSs

We study a number of different computational problems for BT-SyDSs. Our results show a number of interesting differences between the complexities of these problems for simple and complex BT-SyDSs. Formal definitions of the problems studied in this chapter are given below. In the literature, some of these problems have been considered for other dynamical system models [28, 30].

Fixed Point Existence (FPE)

Instance: A BT-SyDS $\mathcal{S}$ specified by the underlying graph $G(V, E)$ and a bithreshold function f_i for each node $v_i \in V$.

Question: Does $\mathcal{S}$ have a fixed point?

The corresponding counting problem (i.e., finding the number of fixed points of $\mathcal{S}$) will be denoted by #FPE.

Predecessor Existence (PRE)

Instance: A BT-SyDS $\mathcal{S}$ specified by the underlying graph $G(V, E)$ and a bithreshold function f_i for each node $v_i \in V$; a configuration $\mathcal{C}$ for $\mathcal{S}$.

Question: Does $\mathcal{C}$ have a predecessor, that is, is there a configuration $\mathcal{C}'$ such that the system has a one step transition from $\mathcal{C}'$ to $\mathcal{C}$?

The corresponding counting problem (i.e., finding the number of predecessors of $\mathcal{C}$) will be denoted by #PRE.

Configuration Reachability (REACH)

Instance: A BT-SyDS $\mathcal{S}$ specified by the underlying graph $G(V, E)$, a bithreshold function f_i for each node $v_i \in V$, and two configurations $\mathcal{C}_1$ and $\mathcal{C}_2$.

Question: Starting from configuration $\mathcal{C}_1$, does $\mathcal{S}$ reach configuration $\mathcal{C}_2$?

D.1.2 Other Definitions

Several definitions concerning Boolean functions and graphs are given below.

Definition 38. ***[Symmetric and r-Symmetric Boolean functions]***

*(a) A q-input Boolean function $f : \{0,1\}^q \to \{0,1\}$ is **symmetric** if the value of the function is determined by how many of the q input values are 1 (rather than which input values are 1).*

*(b) A q-input Boolean function $f : \{0,1\}^q \to \{0,1\}$ is **r-symmetric** for some positive integer r, if the q inputs can be partitioned into at most r subsets such that the value of the function is determined by how many inputs in each of the r groups are 1.*

We note that a symmetric Boolean function is 1-symmetric. A q-input symmetric Boolean function can be specified using an array of size $q+1$, where the Boolean value stored in the ith element of the array gives the value of the function when exactly i input values are 1, $0 \leq i \leq q$. In general, a q-input r-symmetric Boolean function can be specified using a table with at most q^{r+1} rows [30].

Given a connected graph $G(V, E)$, one can compute the shortest path length (in terms of the number of edges) between each pair of nodes. The **diameter** of G is the maximum of these shortest path lengths.

The following are standard definitions of *tree decomposition* and *treewidth* [43].

Definition 39. *Given an undirected graph $G(V, E)$, a* **tree-decomposition** *of G is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a family of subsets of V, and $T = (I, F)$ is an undirected tree with the following properties:*

1. $\bigcup_{i \in I} X_i = V$.
2. *For every edge* $e = \{v, w\} \in E$, *there is a subset* X_i, $i \in I$, *with* $v \in X_i$ *and* $w \in X_i$.
3. *For all* $i, j, k \in I$, *if* j *lies on the path from* i *to* k *in* T, *then* $X_i \bigcap X_k \subseteq X_j$.

The **treewidth** *of a tree-decomposition* $(\{X_i \mid i \in I\}, T)$ *is* $\max_{i \in I}\{|X_i| - 1\}$. *The treewidth of a graph is the minimum over the treewidths of all its tree decompositions. A class of graphs is* **treewidth bounded** *if there is a constant* k *such that the treewidth of every graph in the class is at most* k.

D.1.3 Definitions of Some Known Problems

The following problems, which are well known to be hard for appropriate complexity classes, will be used in the reductions presented in subsequent sections.

Restricted 3SAT (R3SAT)

Instance: A set $X = \{x_1, x_2, \ldots, x_n\}$ of Boolean variables, a set $C = \{C_1, C_2, \ldots, C_m\}$ of clauses, where each clause contains two or three literals and each literal occurs in one or two clauses.

Question: Is there an assignment of Boolean values to the variables in X so that each clause is satisfied?

It is known that R3SAT is NP-complete and the corresponding counting version #R3SAT is #P-complete [153].

#Monotone 2-CNF-SAT (#M2-CNF-SAT)

Instance: A set $X = \{x_1, x_2, \ldots, x_n\}$ of Boolean variables, a set $C = \{C_1, C_2, \ldots, C_m\}$ of clauses, where each clause contains exactly two unnegated literals and each variable appears in at least one clause.

Required: Find the number of assignments that satisfy all the clauses in C.

The above problem is shown to be #P-complete in [317].

Minimum Vertex Cover (MVC)

Instance: An undirected graph $G(V, E)$ and an integer $K \leq |V|$.

Question: Does G have a vertex cover of size at most K (i.e., is there a subset $V' \subseteq V$ such that $|V'| \leq K$ and for every edge $\{u, v\} \in E$, at least one of u and v is in V')?

MVC is known to be NP-complete [120].

Minimum Dominating Set (MDS)

Instance: An undirected graph $G(V, E)$ and an integer $K \leq |V|$.

Question: Does G have a dominating set of size at most K (i.e., is there a subset $V' \subseteq V$ such that $|V'| \leq K$ and every node $v \in V - V'$ is adjacent to some node in V'?

MDS is known to be NP-complete [120]. It can be approximated to within a factor of $O(\log n)$; however, unless P= NP, no better approximation can be obtained using a polynomial time algorithm [322].

Minimum Set Cover (MSC)

Instance: A base set $U = \{u_1, u_2, \ldots, u_n\}$, and a collection $S = \{S_1, S_2, \ldots, S_m\}$ of nonempty subsets of U, a cost c_j for each set $S_j \in S$ and a cost bound Γ.

Question: Is there a set cover of cost at most Γ (i.e., is there a subcollection $S' \subseteq S$ such that the sum of the costs of the sets in S' is at most Γ and every element of U appears in some set of S')?

MSC is known to be NP-complete [120]. It can be approximated to within a factor of $O(\log n)$; however, unless P= NP, no better be approximation can be obtained using a polynomial time algorithm [322].

Minimum Set Multi-Cover (MSMC)

Instance: A base set $U = \{u_1, u_2, \ldots, u_n\}$, a positive integer coverage requirement r_i for each element u_i, $1 \leq i \leq n$, a collection $S = \{S_1, S_2, \ldots, S_m\}$ of nonempty subsets of U, a nonnegative cost c_j for each set S_j, $1 \leq j \leq m$, a cost bound β.

Question: Is there a set multi-cover of cost at most β (i.e., is there a subcollection $S' \subseteq S$ such that the cost of S' is at most β and every element u_i of U is covered by at least r_i sets in S'?

The NP-completeness of MSMC follows from that of MSC. It is known that MSMC can be efficiently approximated to within a factor of $O(\log n)$ [322].

D.2 Results for Fixed Point Existence and Counting Problems

In this section, we present our results for the FPE and #FPE problems for BT-SyDSs. Our results for this problem show interesting differences in the complexity of these problems for

simple and complex BT-SyDSs.

If a BT-SyDS does not have any node with up-threshold = 0, then the configuration of all 0's is a fixed point. Likewise, if a BT-SyDS does not have any node with down-threshold = 0, then the configuration of all 1's is a fixed point. Therefore, the hardness results established in this section for the FPE problem are for BT-SyDSs which contain nodes with up-threshold = 0 as well as those with down-threshold = 0.

We start with a result which shows that the FPE and #FPE problems can be solved efficiently for a class of BT-SyDSs that properly contains simple BT-SyDSs.

Theorem 40. *For BT-SyDSs in which all the threshold values are from* $\{0, 1\}$*, the* FPE *and* #FPE *problems can be solved efficiently.*

Proof: We show how the #FPE problem can be solved efficiently for BT-SyDSs in which all threshold values are from {0,1}. The efficient solvability of FPE follows from that result.

Let $\mathcal{S}$ be any BT-SyDS in which all the threshold values are from {0,1}. Suppose the underlying graph G of $\mathcal{S}$ has k connected components (CCs). We can consider each CC separately since the number of fixed points for $\mathcal{S}$ is the product of the corresponding number for each CC. We have the following claim.

Claim 1: Let H be a connected component of G. In any fixed point of H, all the nodes in H must have the same state value.

Proof Claim 1: Suppose there is a fixed point $\mathcal{C}_H$ of H such that $\mathcal{C}_H(u) = 1$ and $\mathcal{C}_H(v) = 0$ for two nodes u and v of H. In this case, it can be seen that H has two adjacent nodes x and y with complementary state values. Since the threshold values for x and y are at most 1, the two nodes will change states in the next step, contradicting the assumption that $\mathcal{C}_H$ is a fixed point.

Thus, for any CC H of G, there are at most two fixed points, namely the configuration in which all nodes in H are 1 and the one where all the nodes in H are 0. It is easy to test whether these two special configurations are fixed points of H, and hence the number of fixed points for H can be computed efficiently. The total number of fixed points for G can then be computed as the product of the numbers for the different CCs. □

In contrast to Theorem 40, the next result shows that the FPE and #FPE problems are computationally intractable for complex BT-SyDSs; in particular, these intractability results hold for BT-SyDSs where all the threshold values are from {0,1,2}.

Theorem 41. *For BT-SyDSs in which all the threshold values are from* {*0,1,2*}*, the* FPE *problem is* NP*-complete and the* #FPE *problem is* #P*-complete.*

Proof: It is easy to see that the FPE and #FPE problems for BT-SyDSs are in NP and #P respectively. We prove the NP-hardness by a parsimonious reduction from the R3SAT problem defined in Section D.1.3.

Consider an instance I of R3SAT given by the set $X = \{x_1, x_2, \ldots, x_n\}$ of n variables and the set $C = \{C_1, C_2, \ldots, C_m\}$ of m clauses. We construct a BT-SyDS $\mathcal{S}$ as follows. For each variable x_i, the underlying graph $G(V, E)$ of $\mathcal{S}$ contains two nodes denoted by y_i and z_i, $1 \leq i \leq n$. For each node y_i, the up threshold is 2 and the down threshold is 0; for each node z_i, the up threshold is 0 and the down threshold is 2 ($1 \leq i \leq n$). For each literal x_i or $\overline{x_i}$, we construct a tree, which we call the **literal tree**. The leaves of any literal tree are called **attachers** and the interior nodes of a tree are called **unifiers**. For a given literal, the number of attachers equals two plus the number of two-literal clauses containing the literal, plus twice the number of three-literal clauses containing that literal. Each attacher has its up and down thresholds equal to 2. Further, each attacher has one edge connecting it to a unifier for that literal and one edge connecting it to some other node in the graph of $\mathcal{S}$. Each unifier has up and down threshold values equal to 1. Further, each unifier has 2 or 3 incident edges, connecting it to other nodes for its literal. The edges between unifiers for a given literal form a tree. Node y_i has two incident edges, one to an attacher node for the literal x_i, and one to an attacher node for literal $\overline{x_i}$; similarly, node z_i has two incident edges, one to an attacher node for the literal x_i, and one to an attacher node for literal $\overline{x_i}$, $1 \leq i \leq n$.

For each clause C_j, the graph $G(V, E)$ of $\mathcal{S}$ has a node c_j with an up threshold of 0 and down threshold of 2, $1 \leq j \leq m$. If there are two literals in the clause C_j, then node c_j has two incident edges, each connecting it to an attacher node of the literal tree for the two literals. If C_j has three literals, then G has three additional nodes from C_j, which we call **merging nodes** for the clause. Each merging node has its up and down threshold values as 2. Also, there is an edge between c_j and each of its three merging nodes. Each merging node has three incident edges, one to its clause node c_j, and two to attacher nodes for two of the literals in the clause. No two merging nodes for the clause have edges to attacher nodes for the same pair of literals. For instance, suppose a given clause C_j contains x_5, $\overline{x_7}$ and $\overline{x_9}$. Then one merging node has edges to c_j, an attacher node for x_5, and an attacher node for $\overline{x_7}$. A second merging node has edges to c_j, an attacher node for $\overline{x_7}$ and an attacher node for $\overline{x_9}$. The final merging node has edges to c_j, an attacher node for $\overline{x_9}$ and an attacher node for x_5. This completes the construction of $\mathcal{S}$. Obviously, this construction can be done in polynomial time.

Consider a satisfying assignment α to the R3SAT instance I. Construct a configuration $\mathcal{C}(\alpha)$ of $\mathcal{S}$ as follows. Each node c_j is in state 1 and so is each merging node and each node z_i. Each node y_i is in state 0. Each node in a given literal tree has the truth value assigned to that literal by α. It can be verified that $\mathcal{C}(\alpha)$ is a fixed point of $\mathcal{S}$.

Now consider an arbitrary fixed point $\mathcal{C}$ of $\mathcal{S}$. We will show how a satisfying assignment to R3SAT instance I can be constructed.

Consider the literal tree for a given literal. Since each unifier node in the tree has both its thresholds equal to 1, in $\mathcal{C}$, all its neighbors have the same value as the unifier node. Thus, in $\mathcal{C}$, all the nodes in the literal tree for the given literal have the same value.

Now consider a given variable x_i. Since y_i has a down threshold of 0, the value of y_i in $\mathcal{C}$ is 0. Also, since y_i has an up threshold of 2, at most one of its two neighbors has value 1 in $\mathcal{C}$. Since one neighbor is an attacher node for literal x_i and the other neighbor is an attacher node for literal $\overline{x_i}$, the literal tree for x_i and that for $\overline{x_i}$ cannot both have the value 1 in $\mathcal{C}$. Similarly, since z_i has an up threshold of 0, the value of z_i in $\mathcal{C}$ is 1. Also, since z_i has a down threshold of 2, at most one of its two neighbors has value 0 in $\mathcal{C}$. One neighbor of z_i is an attacher node for literal x_i and the other neighbor is an attacher node for literal $\overline{x_i}$. So, the literal tree for x_i and that for $\overline{x_i}$ cannot both have the value 0 in $\mathcal{C}$. Combining the effects of y_i and z_i, it follows that one of the literal trees has value 0 and the other literal tree has value 1 in $\mathcal{C}$.

Now consider a given clause node c_j. Since the up threshold of c_j is 0, c_j has value 1 in $\mathcal{C}$. Since the down threshold of c_j is 2, at most one of its neighbors can have the value 0 in $\mathcal{C}$. Suppose clause C_j contains two literals. Each of the two edges incident on c_j connect it to an attacher node for a literal occurring in clause C_j. So, at least one of these literal trees must have value 1 in $\mathcal{C}$.

Now suppose clause C_j contains three literals. At most one of the three merging nodes connected to c_j can have value 0 in $\mathcal{C}$. So, at least two of these merging nodes must have value 1 in $\mathcal{C}$. Now, consider a given merging node. Since the merging node has its down threshold equal to 2, if both attacher nodes connected to the merging node have value 0 in $\mathcal{C}$, the merging node must have value 0 in $\mathcal{C}$. Since one of the neighbors of the merging node is a clause node with value 1 in $\mathcal{C}$, if at least one of the attacher nodes connected to the merging node has value 1 in $\mathcal{C}$, then the attacher node must have value 1 in $\mathcal{C}$. So, at least one of the literal trees with attacher nodes adjacent to the merging nodes for clause C_j must have value 1 in $\mathcal{C}$.

Given the fixed point $\mathcal{C}$ for $\mathcal{S}$, consider the following assignment $\alpha(\mathcal{C})$ to the Boolean variables of R3SAT instance I: variable x_i is set to 1 iff the literal tree for x_i has value 1 in $\mathcal{C}$. This is a satisfying assignment, since for each clause C_j at least one of the literals in C_j equals 1. This completes the proof of NP-hardness.

We have seen that for each satisfying assignment α, there is a fixed point $\mathcal{C}(\alpha)$, where the values of the literal trees in $\mathcal{C}(\alpha)$ match the values of the variables in α. Similarly, for each fixed point $\mathcal{C}$, there is a satisfying assignment $\alpha(\mathcal{C})$ where the values of the variables match the values of the literal trees. Moreover, no two fixed points have the same combination of values for the literal trees. Therefore, the number of satisfying assignments to the R3SAT instance I equals the number of fixed points of $\mathcal{S}$. Thus, the construction is parsimonious and this completes the proof of #P-hardness. ■

It is also of interest to consider variants of the FPE problem such as the following: Given a BT-SyDS $\mathcal{S}$ and an integer $q \leq n$, does $\mathcal{S}$ have a fixed point in which at most q nodes are in state 1? This variant, which we denoted by MIN-1-FPE, is motivated by the problem of

determining whether a given BT-SyDS has a stable configuration with only a small number of nodes in state 1. The following results hold for MIN-1-FPE.

Theorem 42. *(i) The* MIN-1-FPE *problem can be solved efficiently for BT-SyDSs with a maximum threshold of 1.*

(ii) The MIN-1-FPE *problem is NP-complete for BT-SyDSs where the maximum threshold is 2.*

Proof – Part (i): When all threshold values for the given BT-SyDS $\mathcal{S}$ are at most 1, we show how to construct in polynomial time a configuration $\mathcal{C}$ for $\mathcal{S}$ such that the only nodes in state 1 in $\mathcal{C}$ are those which must be in state 1 in any fixed point.

First, consider each node v such that $\theta_{\text{up}}(v) = 0$ in $\mathcal{S}$. Clearly, in any fixed point, the state of v must be 1. Thus, in $\mathcal{C}$, we initially set the state value for each such node to 1 and the state values of the other nodes to 0. Now, we check each node w in state 0, and if the number of w's neighbors in state 1 is at least the up threshold of w, we change the state of w to 1. We repeat this process until there are no more nodes whose states must be changed from 0 to 1. Since there can be at most n changes from 0 to 1, this process runs in polynomial time.

A straightforward inductive proof shows that each of the nodes whose state was changed to 1 must be in state 1 in any fixed point of $\mathcal{S}$.

Now, the answer to the MIN-1-FPE instance is "yes" if and only if if $\mathcal{C}$ is a fixed point and the number of nodes in state 1 in $\mathcal{C}$ is at most q. Since these checks can be done in polynomial time, the result of Part (i) follows.

Proof – Part (ii): This follows immediately from the NP-completeness of the FPE problems for BT-SyDSs in which the maximum threshold value is 2 (Theorem 41) by setting $q = n$. ■

Theorems 40, 41, and 42 together provide a clear delineation between the complexities of fixed point problems for simple and complex contagions.

D.3 Results for the Predecessor Existence and Counting Problems

D.3.1 Overview

In this section, we consider the predecessor existence (PRE) problem for BT-SyDSs. We again show that the problem is efficiently solvable for a class of BT-SyDSs that properly contains simple BT-SyDSs. However, even for this restricted class, we show that the corresponding counting problem is #P-complete. We also show the PRE problem is NP-complete

for complex BT-SyDSs. However, when the treewidth of the underlying graph is bounded, we show that the PRE problem can be efficiently solved even for complex BT-SyDSs.

D.3.2 An Efficient Algorithm for Predecessor Existence When Thresholds are {0,1}

Here we will show that the PRE problem is efficiently solvable for BT-SyDSs when all the threshold values are from {0,1}. The algorithm is developed through a sequence of lemmas that point out several properties of such BT-SyDSs. Recall that given a configuration $\mathcal{C}$, the state of a node v in $\mathcal{C}$ is denoted by $\mathcal{C}(v)$.

Lemma 43. *Suppose that in a given configuration $\mathcal{C}$ of a BT-SyDS $\mathcal{S}$, a given node v has the property that the threshold for transitioning to $\overline{\mathcal{C}(v)}$ is zero. Then, in every predecessor $\mathcal{C}'$ of $\mathcal{C}$, $\mathcal{C}'(v) = \overline{\mathcal{C}(v)}$.*

Proof: If $\mathcal{C}'(v)$ were equal to $\mathcal{C}(v)$, then the transition from $\mathcal{C}'$ to $\mathcal{C}$ would change the value of v to $\overline{\mathcal{C}(v)}$, a contradiction. □

Lemma 44. *Let $\mathcal{S}$ be a BT-SyDS where all threshold values are from {0,1} and let $\mathcal{C}$ be a configuration of $\mathcal{S}$. Suppose a given node v has the property that all of its neighbors (if any) have value $\mathcal{C}(v)$, and the threshold of v for transitioning to $\mathcal{C}(v)$ is 1. Then, in every predecessor $\mathcal{C}'$ of $\mathcal{C}$, $\mathcal{C}'(v) = \mathcal{C}(v)$.*

Proof: If v has no neighbors, then for any configuration in which v has the value $\overline{\mathcal{C}(v)}$, the (system) transition would not change the value of v.

So, assume that v has at least one neighbor. Suppose there is a predecessor configuration $\mathcal{C}'$ such that $\mathcal{C}'(v) = \overline{\mathcal{C}(v)}$. For each neighbor u of v, $\mathcal{C}(u) = \mathcal{C}(v)$. Since the value $\mathcal{C}'(v)$ is the complement of $\mathcal{C}(u)$ and u and v are neighbors, it must be the case that $\mathcal{C}'(u) = \overline{\mathcal{C}(v)}$. Since $\mathcal{C}'(v)$ and $\mathcal{C}'(u)$ for every neighbor u of v are equal to $\overline{\mathcal{C}(v)}$, and the threshold for v to change its value to $\mathcal{C}(v)$ is 1, the transition from configuration $\mathcal{C}'$ will give v the value $\overline{\mathcal{C}(v)}$, a contradiction. Therefore, $\mathcal{C}'(v) = \mathcal{C}(v)$. □

Lemma 45. *Let $\mathcal{S}$ be a BT-SyDS where all threshold values are from {0,1}. Suppose that for two successive configurations $\mathcal{C}'$ and $\mathcal{C}$ of $\mathcal{S}$, a given node v has the property that $\mathcal{C}'(v) = \mathcal{C}(v)$. Then for every neighbor u of v, $\mathcal{C}'(u)$ also equals $\mathcal{C}(v)$.*

Proof: Suppose $\mathcal{C}'(u) = \overline{\mathcal{C}(v)}$. Then, since node v has a neighbor with a complementary value in $\mathcal{C}'$, the transition from $\mathcal{C}'$ to $\mathcal{C}$ would change the value of v, contradicting the assumption that $\mathcal{C}'(v) = \mathcal{C}(v)$. □

Lemma 46. *Let $\mathcal{S}$ be a BT-SyDS where all threshold values are from $\{0,1\}$. Suppose that for two successive configurations $\mathcal{C}'$ and $\mathcal{C}$ of $\mathcal{S}$, a given node v has the property that $\mathcal{C}'(v) = \overline{\mathcal{C}(v)}$. Then for every neighbor u of v such that $\mathcal{C}(u) = \mathcal{C}(v)$, $\mathcal{C}'(u) = \overline{\mathcal{C}(v)}$.*

Proof: Suppose for some neighbor u of v, $\mathcal{C}'(u) = \mathcal{C}(v)$. Since v is a neighbor of u and $\mathcal{C}'(v)$ has the complementary value to $\mathcal{C}'(u)$, the transition from $\mathcal{C}'$ to $\mathcal{C}$ would change the value of u, contradicting the assumption that $\mathcal{C}(u) = \mathcal{C}(v)$. Thus, $\mathcal{C}'(u) = \overline{\mathcal{C}(v)}$. □

We are now ready to prove the main result of this subsection.

Theorem 47. *The* PRE *problem for BT-SyDSs in which all the threshold vales are from $\{0,1\}$ is efficiently solvable.*

Proof: Suppose that the underlying graph G of the given BT-SyDS $\mathcal{S}$ has more than one connected component (CC). Then, a given configuration has a predecessor iff it has a predecessor when projected onto each CC. So, without loss of generality, we may assume that G consists of a single CC. Let $\mathcal{C}$ denote the given configuration for which a predecessor needs to be found.

Let $\mathcal{C}'$ denote a predecessor (if it exists) for $\mathcal{C}$. The algorithm discussed below tries to construct such a $\mathcal{C}'$.

From Lemmas 43 and 44, some (possibly zero) nodes are *forced* to have a certain value in every predecessor of $\mathcal{C}$. From Lemmas 45 and 46, some other nodes are also *forced* to certain values in every predecessor of $\mathcal{C}$. Let $\widehat{\mathcal{C}}$ be the partial configuration in which all those nodes that are forced to have specific values by Lemmas 43 through 46 are assigned those values. If any node is forced to have both value 0 and value 1, then $\widehat{\mathcal{C}}$ does not exist and $\mathcal{C}$ has no predecessor. So, assume that $\widehat{\mathcal{C}}$ exists. For any node that is assigned a value by $\widehat{\mathcal{C}}$ such that all of its neighbors are also assigned a value by $\widehat{\mathcal{C}}$, these values determine the value that the local transition function f_v assigns to node v. If this value is not equal to $\mathcal{C}(v)$, then $\mathcal{C}$ has no predecessor. So, assume that for all nodes v that are forced and all of whose neighbors are forced, the transition produced by $\widehat{\mathcal{C}}$ assigns the value $\mathcal{C}(v)$. We claim that in this case, $\mathcal{C}$ has a predecessor.

Let $\mathcal{C}'$ be the extension of $\widehat{\mathcal{C}}$ where each node that is not assigned a value in $\widehat{\mathcal{C}}$ is given the complement of its value in $\mathcal{C}$ (i.e., if a given node v is not assigned a value by $\widehat{\mathcal{C}}$, then in $\mathcal{C}'$, node v is assigned the value $\overline{\mathcal{C}(v)}$). We now show that $\mathcal{C}'$ is a predecessor of $\mathcal{C}$.

Let $\mathcal{C}''$ denote the successor of $\mathcal{C}'$. We show that $\mathcal{C}'' = \mathcal{C}$ by proving that for every node v that is either unforced (i.e., not assigned a value in $\widehat{\mathcal{C}}$) or is forced but has at least one neighbor that is unforced, the values assigned to v and its neighbors by $\mathcal{C}'$ ensure that in the transition from $\mathcal{C}'$ to $\mathcal{C}''$, node v receives the value $\mathcal{C}(v)$.

First, suppose v is unforced. In this case, our construction sets $\mathcal{C}'(v) = \overline{\mathcal{C}(v)}$. If the threshold of v for value $\mathcal{C}(v)$ is zero, then in the transition from $\mathcal{C}'$, node v changes to $\mathcal{C}(v)$, so $\mathcal{C}''(v)$

$= \mathcal{C}(v)$. So, assume that the threshold of v for value $\mathcal{C}(v)$ is 1. If all the neighbors of v have the value $\mathcal{C}(v)$ in $\mathcal{C}$, then Lemma 44 would force node v. Since v is unforced, at least one neighbor, say u, has the complementary value in $\mathcal{C}$; that is, $\mathcal{C}(u) = \overline{\mathcal{C}(v)}$. Suppose first that u is unforced. Then, in the extension of $\widehat{\mathcal{C}}$ to $\mathcal{C}'$, u is assigned the complement of its value in $\mathcal{C}$; that is, $\mathcal{C}'(u) = \mathcal{C}(v)$. Since $\mathcal{C}'(v) = \overline{\mathcal{C}(v)}$, the transition from $\mathcal{C}'$ flips the value of v, and so $\mathcal{C}''(v) = \mathcal{C}(v)$. Now suppose that u is forced. If u is forced to $\overline{\mathcal{C}(v)}$, then Lemma 45 is applicable to the transition of u from the value $\overline{\mathcal{C}(v)}$ to $\overline{\mathcal{C}(v)}$, and this forces node v, contradicting the assumption that v is unforced.

Now suppose that node v is forced and has an unforced neighbor, say u. Suppose v is forced to the value $\mathcal{C}(v)$. Then Lemma 45 applies to node v, forcing all neighbors, a contradiction. So, v is forced to the value $\overline{\mathcal{C}(v)}$; that is, $\widehat{\mathcal{C}}(v) = \overline{\mathcal{C}(v)}$. If $\mathcal{C}(u)$ were to equal $\mathcal{C}(v)$, then Lemma 46 would apply to node v and its neighbor would be forced, a contradiction. Thus, $\mathcal{C}(u) = \overline{\mathcal{C}(v)}$. Since u is unforced, we set $\mathcal{C}'(u) = \overline{\mathcal{C}(u)}$, that is, we set $\mathcal{C}'(u) = \mathcal{C}(v)$. Since $\mathcal{C}'(v) = \overline{\mathcal{C}(v)}$ and v has a neighbor u with $\mathcal{C}'(u) = \mathcal{C}(v)$, the transition from $\mathcal{C}'$ flips the value of v, setting $\mathcal{C}''(v) = \mathcal{C}(v)$. ■

D.3.3 Hardness of the Counting Version for {0,1} Thresholds

Here, we show that the problem (#PRE) of counting the number of predecessors of a given configuration is #P-complete even for BT-SyDSs where all threshold values are from {0,1}.

Theorem 48. *The #PRE problem is #P-complete even for BT-SyDSs in which all threshold values are from {0,1}.*

Proof: Membership in #P is obvious. We prove #P-hardness through a parsimonious reduction from #M2-CNF-SAT.

Let I denote an instance of #M2-CNF-SAT with Boolean variable set $X = \{x_1, x_2, \ldots, x_n\}$ and clause set $C = \{C_1, C_2, \ldots, C_m\}$. We construct a BT-SyDS $\mathcal{S}$ as follows. The underlying graph has one node a_i for each variable x_i $(1 \leq i \leq n)$ and one node b_j for each clause C_j $(1 \leq j \leq m)$. Each clause node b_j is joined to the nodes corresponding to variables that appear in clause C_j. The up and down thresholds for each variable node a_i are 1. The up and down thresholds for each clause node b_j are 1 and 0 respectively. The given configuration $\mathcal{C}$ for which we want to count the number of predecessors sets each variable node a_i to 0 and each clause node b_j to 0. This completes the construction, which can obviously be carried out in polynomial time.

Consider any satisfying assignment to the #M2-CNF-SAT instance I. Let $\mathcal{C}'$ be the configuration of $\mathcal{S}$ where each clause node is set to 0 and each variable node is given the truth value in the chosen satisfying assignment. It can be verified that $\mathcal{C}'$ is a predecessor of $\mathcal{C}$. Moreover, each satisfying assignment to I leads to a distinct predecessor of $\mathcal{C}$.

Now consider any predecessor $\mathcal{C}'$ of $\mathcal{C}$. If any clause node b_j is set to 1 in $\mathcal{C}'$, then since the down threshold of b_j is 0, b_j will change to 0 in the transition from $\mathcal{C}'$. However, in $\mathcal{C}$, the value of b_j to be 1. Thus, each clause node must be set to 0 in $\mathcal{C}'$. Further, at least one variable node adjacent to each clause node must have the value 1 in $\mathcal{C}'$, since the clause node has an up threshold of 1. Thus, the truth assignment obtained by setting the variable x_i to the value assigned to a_i by $\mathcal{C}'$ is a satisfying assignment to I. Moreover, each predecessor $\mathcal{C}'$ of $\mathcal{C}$ leads to a distinct satisfying assignment of I.

Hence the number of predecessors of $\mathcal{C}$ is equal to the number of satisfying assignments to the #M2-CNF-SAT instance I and this completes the proof of Theorem 48. ■

D.3.4 NP-Hardness of Predecessor Existence for Complex BT-SyDSs

In this section, we show that the PRE problem is NP-complete for complex BT-SyDSs. In fact, our result shows the problem is computationally intractable even when all the threshold values are 2 and the underlying graph has a maximum degree of 4.

Theorem 49. *The* PRE *problem is NP-complete for BT-SyDSs in which all threshold values are 2 and the maximum node degree in the underlying graph is 4.*

Proof: Membership in NP is obvious. We prove NP-hardness by a reduction from R3SAT. Let I denote an instance of R3SAT with Boolean variable set $X = \{x_1, x_2, \ldots, x_n\}$ and clause set $C = \{C_1, C_2, \ldots, C_m\}$. We construct a BT-SyDS $\mathcal{S}$ as follows.

For each variable x_i, the underlying graph $G(V, E)$ of $\mathcal{S}$ has three nodes denoted by x_i, $\overline{x_i}$ and y_i. For each clause C_j, graph G has four nodes denoted by a_j, b_j, c_j and d_j. For each variable x_i, the three corresponding nodes x_i, $\overline{x_i}$ and y_i are connected together into a triangle (i.e., a 3-clique). For each clause C_j, there are four edges, namely $\{a_j, b_j\}$, $\{b_j, d_j\}$, $\{a_j, d_j\}$ and $\{c_j, d_j\}$. For each clause C_j, there are also edges between c_j and the nodes corresponding to the literals in C_j. All thresholds are set to 2. The given configuration $\mathcal{C}$ has the value 1 for each node c_j and the value 0 for all other nodes. This completes the construction and it is obvious that it can be carried out in polynomial time.

Suppose the R3SAT instance I has a solution. Consider the configuration $\mathcal{C}'$ for $\mathcal{S}$ where each literal node has its truth value in the satisfying assignment, each node d_j has the value 1 and all other nodes have the value 0. It can be verified that $\mathcal{C}'$ is a predecessor of $\mathcal{C}$. (In particular, since each node c_j has two neighbors that are 1, it changes to 1 in the transition from $\mathcal{C}'$ to $\mathcal{C}$.)

Now suppose that $\mathcal{C}$ has a predecessor $\mathcal{C}'$. Suppose for some variable x_i, both x_i and $\overline{x_i}$ are 1. In such a case, no matter what value y_i has in $\mathcal{C}'$, it would change to 1. However, since $\mathcal{C}(y_i) = 0$, at most one of x_i and $\overline{x_i}$ is 1 in $\mathcal{C}'$.

Now suppose that for some c_j, all its literal neighbors are 0 in $\mathcal{C}'$. If c_j was 1 in $\mathcal{C}'$, since it has at least two literal neighbors with value 0, c_j would change to 0. If c_j was 0 in $\mathcal{C}'$, then at most one neighbor (d_j) is 1 in $\mathcal{C}'$; so, c_j would remain 0 in the transition from $\mathcal{C}'$. However, since $\mathcal{C}(c_j) = 1$, at least one of the literal neighbors of c_j is 1 in $\mathcal{C}'$.

Given predecessor $\mathcal{C}'$, consider the assignment to the Boolean variables where variable x_i is set to 1 iff node x_i has the value 1 in $\mathcal{C}'$. Since each node c_j has at least one neighboring literal with value 1 in $\mathcal{C}'$, this is a solution to the R3SAT instance I. ■

D.3.5 An Efficient Algorithm for Treewidth-Bounded Graphs

In this section, we prove that the PRE problem is efficiently solvable for BT-SyDSs when the underlying graph has bounded treewidth. Our result is based on the following result from [30].

Theorem 50. *The* PRE *problem can be solved efficiently for SyDSs when the underlying graph has bounded treewidth and each local transition function is r-symmetric for some fixed integer r.* ■

In view of Theorem 50, the efficient solvability of the PRE problem for treewidth bounded BT-SyDSs would follow by showing that each local transition function for a BT-SyDS is r-symmetric for some fixed integer r. The following lemma establishes that fact.

Lemma 51. *Every bithreshold local transition function is 2-symmetric.*

Proof: Suppose f_i is the bithreshold function at node v_i with $\theta_{\text{up},i} = \tau_1$ and $\theta_{\text{down},i} = \tau_2$. Let d_i denote the degree of v_i. If $d_i = 0$, then the function f_i has only one input (namely, the state of v_i); therefore, f_i is trivially symmetric and hence also 2-symmetric. So, we may assume that $d_i \geq 1$.

Partition the inputs to function f_i into two groups, with the first group consisting of just v_i and the second group consisting of all the neighbors of v_i. We use the notation (n_1, n_2) to denote the collection of inputs to f_i where the number of 1's in the first and second groups are n_1 and n_2 respectively. With a slight abuse of notation, we also let $f_i(n_1, n_2)$ denote the value of the function f_i for the group of inputs represented by the pair (n_1, n_2). Since the first group has one node and the second has d_i nodes, we have $n_1 \in \{0, 1\}$ and $n_2 \in \{0, 1, \ldots, d_i\}$. We specify f_i as a 2-symmetric function using the following three cases.

Case 1: $1 \leq \tau_1 \leq d_i$ and $1 \leq \tau_2 \leq d_i$. Here, f_i can be specified as follows.

$$\begin{aligned} f_i(0,j) &= 0, \quad 0 \le j \le \tau_1 - 1 \quad \text{and} \\ &= 1, \quad \tau_1 \le j \le d_i. \end{aligned}$$

$$\begin{aligned} f_i(1,j) &= 0, \quad 0 \le j \le d_i - \tau_2 - 1 \quad \text{and} \\ &= 1, \quad d_i - \tau_2 \le j \le d_i. \end{aligned}$$

Case 2: $\tau_1 = 0$ or $\tau_2 = 0$. If $\tau_1 = 0$, then $f_i(0,j)$ is given by

$$f_i(0,j) = 1, \quad 0 \le j \le d_i,$$

and the values of $f_i(1,j)$, $0 \le j \le d_i$ are specified by one of the three cases (depending on the value of τ_2). The case where $\tau_2 = 0$ is handled in a similar manner.

Case 3: $\tau_1 > d_i$ or $\tau_2 > d_i$.

If $\tau_1 > d_i$, then once v_i reaches the state 0, it remains in that state for ever. Thus, in that case,

$$f_i(0,j) = 0, \quad 0 \le j \le d_i,$$

and the values of $f_i(1,j)$ are specified by one of the cases. The case where $\tau_2 > d_i$ is handled in a similar manner. This completes the proof. □

The following result is an immediate consequence of Theorem 50 and Lemma 51.

Theorem 52. *The* PRE *problem can be solved efficiently for BT-SyDSs whose underlying graphs are treewidth-bounded.* ■

D.4 Results for the Reachability Problem

In this section, we show that for the class of BT-SyDSs where all threshold values are from {0,1}, the REACH problem can be solved efficiently. Without loss of generality, we assume that the underlying graph of the given BT-SyDS is connected. (If the graph has two or more connected components, the reachability problem can be considered for each component separately.) In devising this algorithm, we establish several properties of the phase space of BT-SyDSs where the threshold values are from {0,1}. We begin with a definition.

Definition 53. *Let $\mathcal{S}$ be a BT-SyDS where the underlying graph is connected and all threshold values are from {0,1}. Let $\mathcal{C}$ be a configuration of $\mathcal{S}$. A given node v of $\mathcal{S}$ is **strong** in $\mathcal{C}$ if either both thresholds for v are 0 or there is a neighbor u of v such that $\mathcal{C}(u) \neq \mathcal{C}(v)$. Otherwise v is **weak** in $\mathcal{C}$. A given weak node is **stable** if the threshold for the node to change its value is 1.*

*A given configuration $\mathcal{C}$ of $\mathcal{S}$ is is **strong** if all the nodes of $\mathcal{S}$ are strong in $\mathcal{C}$, is **stable weak** if all the nodes are weak and stable in $\mathcal{C}$, and is **transitional** if there is at least one strong node, at least one weak node and all weak nodes are stable weak.*

The following lemmas establish some properties of strong and weak nodes.

Lemma 54. *Let $\mathcal{S}$ be a BT-SyDS where the underlying graph G is connected and all threshold values are from $\{0,1\}$. A given configuration $\mathcal{C}$ of $\mathcal{S}$ is a fixed point iff $\mathcal{C}$ is stable weak.*

Proof: First, suppose $\mathcal{C}$ is a fixed point. If any node is strong in $\mathcal{C}$ or has a threshold of 0 to change, that node will change its value in the transition from $\mathcal{C}$. So, all nodes are stable weak in $\mathcal{C}$.

Now suppose $\mathcal{C}$ is a stable weak configuration. Since all the nodes are weak, they all have the same value. Since they are all stable weak, they require a neighbor with the opposite value to change. So, in the transition from $\mathcal{C}$, no node changes value. In other words, $\mathcal{C}$ is a fixed point. □

Lemma 55. *Let $\mathcal{S}$ be a BT-SyDS where all threshold values are from $\{0,1\}$. If two adjacent nodes in a given configuration $\mathcal{C}$ have complementary values, then they will have complementary values in the successor of $\mathcal{C}$.*

Proof: Obvious. □

Lemma 56. *Let $\mathcal{S}$ be a BT-SyDS where the underlying graph is connected and all threshold values are from $\{0,1\}$. If a given node v is strong in a given configuration $\mathcal{C}$ of $\mathcal{S}$, then v is also strong in the successor of $\mathcal{C}$, and has the complementary value in the successor.*

Proof: If v is strong because its up and down threshold values are 0, then v is strong in every configuration, and its value will flip in the transition from $\mathcal{C}$. If v is strong because v has a neighbor u such that $\mathcal{C}(u) = \overline{\mathcal{C}(v)}$, then the transition from $\mathcal{C}$ will flip the values of both u and v, and v will also be strong in the successor configuration. □

Lemma 57. *Let $\mathcal{S}$ be a BT-SyDS where the underlying graph is connected and all threshold values are from $\{0,1\}$. If a given configuration $\mathcal{C}$ of $\mathcal{S}$ is strong then $\mathcal{C}$ is part of a 2-cycle in the phase space of $\mathcal{S}$.*

Proof: This is a direct consequence of Lemma 56. □

Lemma 58. *Let $\mathcal{S}$ be a BT-SyDS where the underlying graph is connected and all threshold values are from $\{0,1\}$. After one transition of $\mathcal{S}$, every node is either a strong node or a stable weak node.*

Proof: Consider a transition of $\mathcal{S}$ from configuration $\mathcal{C}'$ to $\mathcal{C}$, and consider an arbitrary node v. If v is strong in $\mathcal{C}'$, then from Lemma 56, v is also strong in $\mathcal{C}$. So, assume that v is weak in $\mathcal{C}'$. Thus, for all neighbors u of v, $\mathcal{C}'(u) = \mathcal{C}'(v)$. First, suppose that $\mathcal{C}(v) = \mathcal{C}'(v)$. Since v does not change in the transition from $\mathcal{C}'$ to $\mathcal{C}$, the threshold for v to change value is 1. Hence, in $\mathcal{C}$, v is either strong or stable weak. Now suppose that $\mathcal{C}(v) = \overline{\mathcal{C}'(v)}$. Since v is weak in $\mathcal{C}'$, all its neighbors have the same value, namely $\mathcal{C}'(v)$, in $\mathcal{C}'$. Hence, for v to change value, its threshold to change from $\mathcal{C}'(v)$ to $\overline{\mathcal{C}'(v)}$ must be 0. Since v is not strong in $\mathcal{C}'$, the threshold for v to change from $\overline{\mathcal{C}'(v)}$ to $\mathcal{C}'(v)$ must be 1. So, v is either strong in $\mathcal{C}$ or stable weak. □

Lemma 59. *Let $\mathcal{S}$ be a BT-SyDS where the underlying graph G is connected and all threshold values are from $\{0,1\}$. After one transition of $\mathcal{S}$, the graph G is strong or stable weak or transitional.*

Proof: Follows from Lemma 58. □

We need one more definition before stating our next lemma.

Definition 60. *Let $\mathcal{S}$ be a BT-SyDS where the underlying graph G is connected and all threshold values are from $\{0,1\}$. Let $\mathcal{C}$ be a transitional configuration of $\mathcal{S}$ and let v be a weak node in $\mathcal{C}$. The* ***critical distance*** *of v is the distance (i.e., the number of edges) from v to the nearest strong node.*

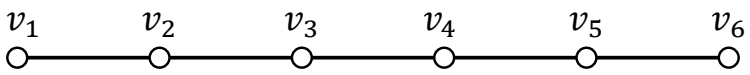
Lemma 61. *Let $\mathcal{S}$ be a BT-SyDS where the underlying graph G is connected and all threshold values are from $\{0,1\}$. Let $\mathcal{C}'$ be a transitional configuration of $\mathcal{S}$ and let $\mathcal{C}$ be the successor of $\mathcal{C}'$. Then, $\mathcal{C}$ is either a strong configuration or a transitional configuration. Moreover, all weak nodes whose critical distance is 1 in $\mathcal{C}'$ are strong in $\mathcal{C}$, and all other weak nodes have their critical distance reduced by 1.*

Proof: Suppose the critical distance in $\mathcal{C}'$ for a weak node v is 1. Thus, v has a neighbor u such that u is strong and $\mathcal{C}'(u) = \mathcal{C}'(v)$. From Lemma 56, $\mathcal{C}(u) = \overline{\mathcal{C}(u)}$. Since v is a weak node in a transitional configuration, v is a stable weak node and so $\mathcal{C}(v) = \mathcal{C}'(v)$. Thus, v is strong in $\mathcal{C}$. Any weak node v in $\mathcal{C}'$ whose critical distance is greater than 1 is surrounded by other weak nodes u with $\mathcal{C}'(u) = \mathcal{C}'(v)$, none of which change during the transition from $\mathcal{C}'$ to $\mathcal{C}$. Therefore, v remains weak in $\mathcal{C}$. By simple induction on the value of the critical distance, it can be seen that each weak node in $\mathcal{C}'$ that remains weak in $\mathcal{C}$ has its critical distance reduced by 1. □

Theorem 62. *Let $\mathcal{S}$ be a BT-SyDS where the underlying graph G is connected and all threshold values are from $\{0,1\}$. Let d be the diameter of G. Any path in the phase space of $\mathcal{S}$ consists of either a transient path of at most one configuration leading to a fixed point configuration or a transient path of length at most $d+1$ configurations leading to a cycle of length 2.*

Proof: From Lemma 59, after one transition, the new configuration $\mathcal{C}$ is either strong, stable weak or transitional. If $\mathcal{C}$ is strong, then from Lemma 57, it is part of a 2-cycle. If $\mathcal{C}$ is stable weak, then from Lemma 54, it is a fixed point. So, assume that $\mathcal{C}$ is transitional. Thus, $\mathcal{C}$ has at least one strong node. Since G has a diameter of d, the maximum possible value of critical distance for any weak node is d. From Lemma 61, each subsequent transition reduces the maximum critical distance by 1. So, after $d-1$ transitions from $\mathcal{C}$, there are no more weak nodes; that is, a configuration of all strong nodes has been reached. ■

The following example illustrates the bound established in the above theorem.



Note: The up and down thresholds for v_1 are 0. For each of the other nodes, the up threshold is 1 and the down threshold is 0.

Figure D.1: An example to illustrate the bound established in Theorem 62.

Example: Consider the BT-SyDS $\mathcal{S}$ shown in Figure D.1. The diameter of the underlying graph is 5. The phase space of $\mathcal{S}$ contains the following path:

$$\begin{aligned}
\mathcal{C}_1 &= (1, 1, 1, 1, 1, 1) \\
\mathcal{C}_2 &= (0, 0, 0, 0, 0, 0) \\
\mathcal{C}_3 &= (1, 0, 0, 0, 0, 0) \\
\mathcal{C}_4 &= (0, 1, 0, 0, 0, 0) \\
\mathcal{C}_5 &= (1, 0, 1, 0, 0, 0) \\
\mathcal{C}_6 &= (0, 1, 0, 1, 0, 0) \\
\mathcal{C}_7 &= (1, 0, 1, 0, 1, 0) \\
\mathcal{C}_8 &= (0, 1, 0, 1, 0, 1)
\end{aligned}$$

After the above sequence, $\mathcal{S}$ cycles through the two configurations $\mathcal{C}_7$ and $\mathcal{C}_8$.

In $\mathcal{C}_1$, node v_1 is strong, and the other nodes are weak but not stable weak. Configuration $\mathcal{C}_2$ is transitional since v_1 is strong and the other nodes are stable weak. The maximum critical distance of 5 (which is the diameter of the underlying graph) occurs for v_6. In $\mathcal{C}_3$, v_1 and v_2 are strong and the maximum critical distance is 4. In $\mathcal{C}_4$, v_1, v_2 and v_3 are strong and the maximum critical distance is 3. In $\mathcal{C}_5$, v_1 through v_4 are strong and the maximum critical distance is 2. In $\mathcal{C}_6$, v_1 through v_5 are strong and the maximum critical distance is 1. Configuration $\mathcal{C}_7$ is strong, and it is part of the 2-cycle with $\mathcal{C}_8$. □

The following result is a direct consequence of Theorem 62.

Corollary 63. *The* REACH *problem for BT-SyDSs where all the threshold values are from* $\{0,1\}$ *can be solved in polynomial time.* ■

When all threshold values are 1, we can provide a more precise characterization of the phase space as indicated in the following result.

Theorem 64. *Let* $\mathcal{S}$ *be a BT-SyDS where the underlying graph* G *is connected and all threshold values are 1. Let* d *be the diameter of* G*. Any path in the phase space of* $\mathcal{S}$ *consists of either a cycle of length 1 or a transient path of length at most* $d-1$ *configurations leading to a cycle of length 2.*

Proof: Let $\mathcal{C}$ be the initial configuration. Since all threshold values are 1, all weak nodes in $\mathcal{C}$ are stable weak. So, $\mathcal{C}$ is strong or stable weak or transitional. If $\mathcal{C}$ is strong, then by Lemma 57, it is part of a 2-cycle. So, assume that $\mathcal{C}$ is transitional. Thus, $\mathcal{C}$ has at least one strong node. Since all thresholds are 1, if a given node is strong, then at least one of its neighbors has a complementary value; thus, that neighbor is also strong. Thus, in $\mathcal{C}$, there is at least one strong node with value 1 and at least one strong node with value 0. Consider a given weak node v. Since G has a diameter of d, the distance between v and the closest strong node u such that $\mathcal{C}(u) \neq \mathcal{C}(v)$ is at most d. The first node in any path from u to v is a strong node w such that $\mathcal{C}(w) = \mathcal{C}(v)$. Thus, the maximum possible value of the critical distance for v is $d-1$. This holds for every weak node in $\mathcal{C}$. From Lemma 61, each transition starting from $\mathcal{C}$ reduces the maximum critical distance by 1. So, after $d-2$ transitions, there are no more weak nodes, and a configuration of all strong nodes has been reached. ■

Like the FPE problem, one may also consider variants of the REACH problem. One such variant, which we denote as MIN-1-REACH is the following: Given a BT-SyDS $\mathcal{S}$ an initial configuration $\mathcal{C}$ and an integer $q \leq n$, does the system reach a configuration in which at most q nodes are in state 1? If nodes in state 1 correspond to smokers, this question asks whether a social network will reach a configuration with a small number of smokers. The next result points out that this variant of reachability can also be solved efficiently for BT-SyDSs in which the maximum threshold value is 1.

Proposition 65. *The* MIN-1-REACH *problem can be solved efficiently for BT-SyDSs in which the maximum threshold value is 1.*

Proof: It was shown in the proof of Theorem 62 that a BT-SyDS $\mathcal{S}$ in which each threshold value is at most 1 reaches either a fixed point or a cycle between two configurations in a number of steps which is bounded by the diameter of the underlying graph. It is easy to check whether the number of 1's in the configuration reached after each transition is bounded by q. Since the diameter of any undirected graph is at most $n-1$, the number of configurations to be checked is at $n+1$. The proposition follows. ■

The complexity of the REACH problem for complex BT-SyDSs is open.