2011

A model checking approach for analyzing and identifying intervention policies to counter infection propagation over networks

Yuly Suvorov
Iowa State University

A model checking approach for analyzing and identifying intervention policies to counter infection propagation over networks

by

Yuly Suvorov

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:

Samik Basu, Major Professor

Andrew S. Miner

Ting Zhang

Iowa State University

Ames, Iowa

2011

TABLE OF CONTENTS

LIST OF TABLES iv

LIST OF FIGURES v

ACKNOWLEDGEMENTS vi

ABSTRACT vii

CHAPTER 1. INTRODUCTION 1

1.1 Modeling Network Propagation 2

1.1.1 Finding and Analyzing Policies 3

1.1.2 Region Computation 3

1.2 Contributions 4

CHAPTER 2. GRAPH-THEORETIC MODELS OF SPREAD 5

2.1 Local Transmission Model 6

2.1.1 Irreversible k-Threshold Process 6

2.1.2 r-Reversible k-Threshold Process 7

2.2 Policies 9

CHAPTER 3. ANALYZING POLICIES USING MODEL CHECKING . . . 14

3.1 Encoding Infection Spread in Kripke Structures 15

3.2 Finding & Verifying Policies using LTL 17

3.2.1 Q1: Finding an intervention policy 19

3.2.2 Q2: Verifying a preventive policy 21

CHAPTER 4. REGION-BASED PROPAGATION ANALYSIS 23

4.1 Region Generation for Intervention Policies 24

4.1.1 Region Generation Algorithm 25

CHAPTER 5. EXPERIMENTS & RESULTS 27

5.1 Results of Identifying Intervention Policies 27

5.1.1 Optimizations to Improve Scalability 28

5.1.2 Effectiveness of Optimizations 31

5.2 Results of Region Generation 32

CHAPTER 6. RELATED WORK 35

CHAPTER 7. CONCLUSION 37

7.1 Future Work 38

BIBLIOGRAPHY 40

LIST OF TABLES

5.1 Results for random networks with 40 nodes. Numbers in parentheses denote results with optimizations. 28

5.2 Results for scale-free networks. 31

5.3 Region Storage Requirements. 33

5.4 Average region sizes for various containment functions. 33

LIST OF FIGURES

2.1 Infection Spread in a 3 x 3 Grid 8

2.2 Infection Spread in a 4 x 4 Grid 10

2.3 Infection Spread in a ring of 7 nodes 11

ACKNOWLEDGEMENTS

I would like to thank those who have helped me with various aspects of my research and my graduate studies. First and foremost, Dr. Samik Basu, for his guidance and support throughout my graduate career. His teaching inspired me to complete my studies and my choice of research area. I would like to thank my committee members for their efforts and contributions to this work: Dr. Andrew S. Miner and Dr. Ting Zhang. Finally, I would like to thank Ganesh Ram Santhanam for his help and guidance in my research.

This work is supported in parts by NSF grant CCF 0702758.

ABSTRACT

The spread of infections (disease, ideas, fires, etc.) in a network (group of people, electronic network, forest, etc.) can be modeled by the evolution of states of nodes in a graph defined as a function of the states of the other nodes in the graph. Given an initial configuration of the graph with a subset of the nodes infected, a propagation function that specifies how the states of the nodes change over time, and a quarantine function that specifies the generation of regions centered on the infected nodes, from which the infection cannot spread; we identify and verify intervention policies designed to contain the propagation of the infection over the network. The approach can be used to determine an effective policy in such a scenario.

CHAPTER 1. INTRODUCTION

The spread of infections such as diseases, rumors, computer viruses, and fires through networks of people, computers, and forests pose significant risks in the modern sociological, technological, and environmental world. Intervention policies are essential for reacting to, and mitigating, the effects of infections.

The spread of viruses and worms through computer networks is a great concern. The danger viruses and worms pose continues to rise as more and more devices become connected to the world wide web. Many computers are vulnerable due to lax security measures (virus scanners, firewalls, etc.) and due to newly discovered vulnerabilities for which patches are not yet available. Such infections are not only widespread, but can have far reaching consequences. The Conficker worm infected an estimated 15 million computers, including computers belonging to the French military. The French military was forced to resort to using "'telephone, fax, and post"' for communication because they had ignored the threat Conficker posed and did not update their systems (UPI (2009); Willsher (2009)).

These problems are relevant in areas beyond computing. Due to the highly mobile nature of modern society, diseases pose a greater threat than ever before. In 2006 alone, there were 2.1 billion airline passengers. Such mobility allows for rapid spread of disease from the initial infection point to locations many thousands of miles away. In addition to the highly mobile society, there has been an increase in the emergence of new diseases, with almost 40 appearing in the last generation alone. The number of epidemics in the span from 2002 to 2007 is over 1100 (WHO (2007)).

> During outbreaks of these diseases [Severe Acute Respiratory Syndrome (SARS), Ebola, Marburg hemorrhagic fever, Nipah virus], rapid assessment and response, often needing international assistance, has been required to limit local spread. (WHO

(2007))

With potentially millions of people at risk from a disease, automated decision support is required for policymakers to rapidly and accurately respond to emerging diseases and epidemics.

Communication plays an essential role in the organization of people. The spread of ideas can be seen either in a positive light or a negative one. Presidential candidates use the mass media to spread their messages to garner votes. Yet, in January, the Egyptian government cut off its people from the internet in an effort to prevent a populous uprising (Cowie (2011)). To counteract such measures, the US government is funding development of "'shadow internet"' devices to counteract government shutdowns of information networks (Simao (2011)). Clearly the spread of information is vital and, as such, identifying the elements of a network with the greatest ability to spread information is esseintial. Automated decision support is necessary for identification of such elements and the analysis of how information will spread in the network.

The above are all examples of network propagation problems.

1.1 Modeling Network Propagation

The dynamics of the spread of infections in networks can be modeled in terms of the evolution of the states of nodes in graphs. In such graphs, nodes represent individuals (people, computers, parts of a forest) and edges represent possible transmission vectors. Two people coming in contact with each other would have an edge between their respective nodes. Similarly connections between computers and topographic adjacency in a forest would be represented by edges. Previous work in this area (Dreyer and Roberts (2009); Finbow and MacGillivray (2009); Anshelevich et al. (2010)) has examined the existence of effective policies that prevent the dynamics of such graphs from violating some desired property. The problem of finding an effective policy to control the spread of an infection in a network is NP-hard even for graphs with a maximum degree of three (Dreyer and Roberts (2009); Finbow and MacGillivray (2009); Anshelevich et al. (2010)). Against this background, practical solutions to the problem of infection spread in a network are of significant interest. Our approach involves two techniques: locating policies and analyzing policies in response to, or prevention of, outbreaks and subdividing the

network into smaller regions for efficient application of the aforementioned policies.

1.1.1 Finding and Analyzing Policies

We consider a simple model of infection spread representing a discrete population evolving in discrete time steps. The state of a node in the network in each time step is determined by (a) the previous states of the node and (b) the states of other nodes in the network. A node may be *open* (susceptible to infection), *infected*, or *protected* (cannot be infected nor spread the infection). The evolution of the states of nodes in the network is encoded using *Kripke structures*. We focus on two kinds of policies: *intervention policies* and *preventive policies*.

Intervention policies are designed to contain an infection by designating certain nodes as protected to stop the spread of an existing infection. Preventive policies protect a certain subset of nodes such that the spread of an infection in the network is limited. We consider a policy to be effective if in every time step, the number of infected nodes does not exceed l. Other definitions of effectiveness can be accommodated by our model. The satisfiability of temporal formulas is used to answer the following questions: (a) Given a network with a set of initially infected nodes, is there an effective intervention policy (marking of open nodes as protected)? (b) Given a network and a set of protected nodes, is the given preventive policy effective? Our approach automatically finds an intervention policy if a such a policy exists, and identifies conditions under which a preventive policy is ineffective. We first introduced the model checking approach to verifying and analyzing policies in Santhanam et al. (2011).

1.1.2 Region Computation

In practical settings, policymakers are likely to divide a network into regions. Such a division allows for independent policies for each region. If an infection struck the United States, it is likely that different policies would be applied to individual states, or even cities. A city with an international airport requires a very different strategy of containing an infection than a rural town. In the computer realm, a critical region in a network could center on a hub node servicing a large cluster of nodes. Thus, given a network graph, for every infected node a subgraph, *region*, is computed from which the infection cannot spread. Regions allow for

efficient scaling of our approach and represent the quarantining, or isolation, of the portion of the network under attack. Once the infection has been isolated to a part of the network, finer grained policies are found to further reduce the effects of the infection.

We have developed techniques for locating intervention policies within a region of a network and verifying prevention policies utilizing the specialized algorithms used by model checkers to compute reachability in a Kripke structure model. Our preliminary experimental results demonstrate the feasibility of our graph-theoretic, deterministic discrete time, approach for finding and verifying effective policies. Our approach can be used to counter the spread of diseases (WHO (2007)), fire (MacGillivray and Wang (2003)), opinions (Zanette (2002)), and computer viruses (Serazzi and Zanero (2004)), where similar deterministic, discrete-time models of infection spread have been considered (Dreyer and Roberts (2009); Finbow and MacGillivray (2009); Anshelevich et al. (2010)).

1.2 Contributions

We provide a method for verifying and analyzing policies in a graph-theoretic model of infection spread. By demonstrating the feasibility of our approach, we pave the way for the development of tools, utilizing our approach, to aid policymakers in containing infection spread in various networks. The generic nature of our approach makes it equally viable for containing disease spread in humans, opinions/rumors in social networks, computer viruses in computer networks, and fires in forests.

The rest of the thesis is as follows. In Chapter 2 we introduce the graph-theoretic model of infection spread. Following in Chapter 3, we encode the evolution of the infection spread in a graph in a Kripke structure and define policies designed to contain the spread of an infection. Chapter 4 provides a method for the division of a graph in to regions for better handling of infection spread. Chapter 5 presents our preliminary experimental results and the optimizations required in scaling our approach. In Chapter 6 we explore other approaches to modeling infection spread and future improvements to our approach. Finally, in Chapter 7 we summarize our contributions to network propagation problems.

CHAPTER 2. GRAPH-THEORETIC MODELS OF SPREAD

The following was initially presented in Santhanam et al. (2011). Let $G(V, E)$ be a directed graph, whose nodes V represent entities (people, computers, etc.) of the network that can potentially be infected. The edges E are ordered 2-tuples of nodes representing the medium over which an infection can potentially be transmitted from one node to another. Thus, if $(v_i, v_j) \in E$, then v_u can transmit the infection from itself to v_j. An undirected graph can be represented by having two edges ((v_i, v_j) and (v_j, v_i)) between any connected nodes v_i and v_j. The methods in this paper are equally valid for directed and undirected graphs, however for simplicity, unless otherwise noted, the graphs mentioned in this paper are undirected. Each node $v_i \in V$ in the graph is associated with a state $\sigma(v_i) \in \Sigma$, the domain of possible states (e.g. infected, uninfected). The set of neighbors of a node $v_i \in V$ is $\rho(v_i) = \{v_j \in V : (v_i, v_j) \in E\}$. Thus, the neighbors of the node v_i are those nodes in V that v_i has an edge to.

We denote the tuple of all states of the nodes in the graph as the *configuration* of the graph at the discrete time step t. At every discrete time step t, each node v_i in the graph changes its state as a function of (a) the current state of the other nodes in the graph and (b) the current and previous states of v_i, itself. Let f be the *transmission* function representing the state transitions described by (a). The function f determines the state of a node with respect to the states of the other nodes in the graph. Let g be the *local update* function representing the state transitions described by (b). The function g determines the state of a node with respect to the history of states of the node. Collectively, f and g are the *infection propagation* functions. The new state of node v_i in the next time step is the composition of the functions f and g applied on $\sigma(v_i)$. Thus, the configuration of the graph evolves with every discrete time step.

The above specified model of infection spread over a graph $G(V, E)$ is generic. Different

definitions of Σ, f, g, as well as the order of composition of f and g, can be used to model the spread of infections in various applications such as diseases in human populations, viruses in computer networks, opinions in social network,s or fires in forest regions. If, in a given application, the spread of an infection is independent of either (a) or (b) above, the respective function f or g becomes the identity function.

2.1 Local Transmission Model

In this thesis we focus on a model of infection spread of a graph $G(V, E)$ where $\Sigma = \{\texttt{open}, \texttt{infected}, \texttt{protected}\}$. Thus a node can be in one of three states:

`open` – the node is vulnerable to infection

`infected` – the node is infected and can spread the infection to other nodes

`protected` – the node can never be infected

The state of node v_i in the graph is denoted by $\sigma(v_i)$. In this paper we focus on the *r-reversible* and *irreversible k-threshold* processes described in (Dreyer and Roberts (2009)). These processes define the transmission functions f and g in the graph.

2.1.1 Irreversible k-Threshold Process

In this model of infection spread, a node v_i becomes infected in time step $t + 1$ if at time step t at least k of its neighbors are infected. A node in the infected state remains in the infected state throughout the evolution of the graph. Thus the local update function g is $g(\sigma(v_i)) = \sigma(v_i)$, the identity function. The transmission function f is defined as follows:

$$f(\sigma(v_i)) = \begin{cases} \texttt{infected} & \text{if } \sigma(v_i) = \texttt{open} \text{ and} \\ & \exists u_1, u_2 \dots u_k \in \rho(v_i) : \\ & \forall j \in [1, k] \\ & \sigma(u_j) = \texttt{infected} \\ \sigma(v_i) & \text{otherwise} \end{cases}$$

It can be noted that number of infected nodes forms a monotonic nondecreasing function with respect to time.

2.1.2 r-Reversible k-Threshold Process

As in the previous model, a node enters the infected state if at least k of its neighbors were in the infected state in the previous time step. A node in the infected state returns to the open state r time steps after it was infected. To accommodate this, the set of states is expanded to $\Sigma = \{\texttt{open}, \texttt{protected}, \texttt{infected}_1, \ldots, \texttt{infected}_r\}$ to track the intermediate states of the infected nodes. The infection propagation functions f and g are defined as follows:

$$f(\sigma(v_i)) = \begin{cases} \texttt{infected}_1 & \text{if } \sigma(v_i) = \texttt{open} \text{ and} \\ & \exists u_1, u_2 \ldots u_k \in \rho(v_i) : \\ & \forall j \in [1, k] \\ & \sigma(u_j) = \texttt{infected} \\ \sigma(v_i) & \text{otherwise} \end{cases}$$

$$g(\sigma(v_i)) = \begin{cases} \texttt{open} & \text{if } \sigma(v_i) = \texttt{infected}_r \\ \texttt{infected}_{q+1} & \text{if } \sigma(v_i) = \texttt{infected}_q \\ & \text{and } q < r \\ \sigma(v_i) & \text{otherwise} \end{cases}$$

Reversible processes are useful for modeling the spread of infections from which the individual (person, computer, etc.) eventually recovers. This is the case for diseases like the common cold and computer viruses when a virus scan is run.

The following example illustrates the basics of infection transmission functions.

Example 1 *Consider a 2-dimensional grid of size 3 representing a forest as show in Figure 2.1. The figure shows the evolution of the graph for three time steps under two different infection transmission functions described below.*

Suppose that nodes 2, 4, *and* 9 *are on fire/infected (colored black) initially at* $t = 0$. *In the top row the infection transmission function is the irreversible 2-threshold function. A part*

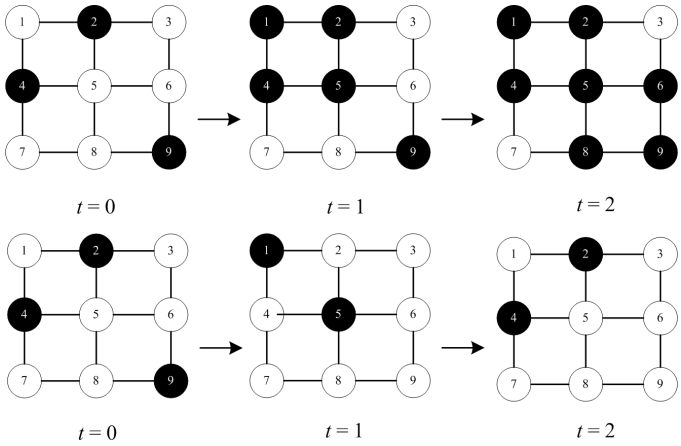

Figure 2.1 Infection Spread in a 3 x 3 Grid

of the forest remains on fire if it was on fire before, and the fire can spread to a new node if the node has two neighbors on fire. Thus the local update function g is the identity function, i.e., $g(\sigma(v_i)) = \sigma(v_i)$. The transmission function f returns infected if the node has at least two on-fire neighbors. Formally f is defined as follows:

$$f(\sigma(v_i)) = \begin{cases} \texttt{infected}_1 & \text{if } \sigma(v_i) = \texttt{open} \text{ and} \\ & \exists u_1, u_2 \in \rho(v_i) : \\ & \forall j \in \{1, 2\} \\ & \sigma(u_j) = \texttt{infected} \\ \sigma(v_i) & \text{otherwise} \end{cases}$$

The figure shows that at time step $t = 1$ the infection spreads to nodes 1 and 5, as these are the only nodes with two infected neighbors. At time step $t = 2$, the infection spreads to nodes 6 and 8. At time step $t = 3$, not shown, the fire spreads to the final two open nodes, 3 and 7. Thus, in this scenario, the entire forest is consumed by the fire.

Alternatively, the bottom row demonstrates the 1-reversible 2-threshold infection propagation function. The functions f and g are as follows:

$$f(\sigma(v_i)) = \begin{cases} \texttt{infected} & \text{if } \sigma(v_i) = \texttt{open} \text{ and} \\ & \exists v_1, v_2 \in \rho(v_i) : \\ & \forall j \in \{1, 2\} \\ & \sigma(v_j) = \texttt{infected} \\ \sigma(v_i) & \text{otherwise} \end{cases}$$

$$g(\sigma(v_i)) = \begin{cases} \texttt{open} & \text{if } \sigma(v_i) = \texttt{infected} \\ \sigma(v_i) & \text{otherwise} \end{cases}$$

Thus, a node catches fire if at least two neighboring nodes are on fire and a node on fire is extinguished in one time step.

As with the top row, initially nodes 2, 4, *and* 9 *are on fire at* $t = 0$. *At time step* $t = 1$, *by the transmission function* f, *the fire spreads to nodes* 1 *and* 5. *During the same time step, by the local update function* g, *the nodes* 2, 4, 9 *are no longer on fire and return to the open state. In time step* $t = 2$ *nodes* 2 *and* 4 *return to being on fire, while the fire dies out at nodes* 1 *and* 5. *Thus, the number of nodes on fire will remains constant at two* $\forall t > 0$. *The fire merely oscillates between two sets of nodes. This example illustrates both the irreversible and reversible k-threshold infection propagation functions.*

2.2 Policies

One of the ways of controlling the spread of an infection in a graph is by protecting (e.g., vaccinating) some of the nodes from infection with the goal of preventing the infection from spreading to some of the other open nodes. The strategy by which a subset of the open nodes is marked protected (vaccinated) is a *policy*.

Definition 1 (Policy) *A policy* π *in graph* $G(V, E)$ *is a function from time steps* t *to a set of nodes* $\subseteq V$.

Informally a policy dictates what nodes are to be protected at a given time step. In this paper we consider two types of policies: (1) *prevention policies* that are deployed before any nodes in the graph are infected; and (2) *intervention policies* that are deployed after an

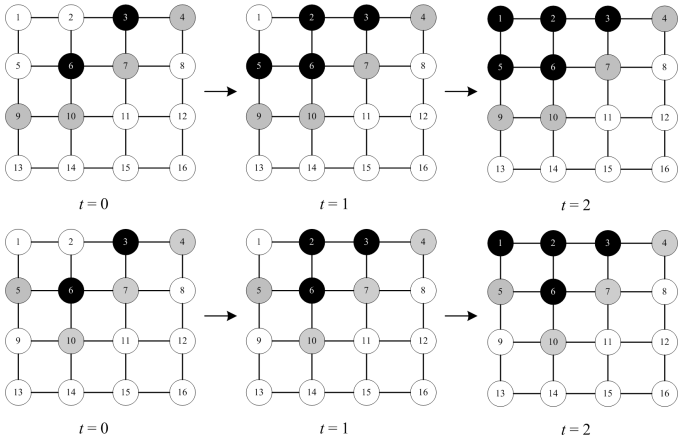

Figure 2.2 Infection Spread in a 4 x 4 Grid

infection outbreak occurs at set of nodes in the graph. Prevention policies are those that designate nodes protected only in the first time step $t = 0$. Formally, a prevention policy π is such that $\forall t > 0, \pi(t) = \emptyset$. Prevention policies seek to prevent a potential infection outbreak from becoming an epidemic. Intervention policies seek to contain the spread of an infection outbreak that is already in progress. The objective of both types of policies is to limit the spread of the infection to at most l nodes in the graph. Such policies are of particular interest to policymakers and public health officials tasked with preventing outbreaks of infections or containing infection outbreaks that occur, using prevention policies or intervention policies respectively. The following examples illustrate the use of policies in combating infections.

Example 2 *Consider a 2-dimensional grid of size 4 representing a social network as show in Figure 2.2. Suppose that, initially, at time* $t = 0$ *nodes* 3 *and* 6 *are infected. Let the infection transmission function be the irreversible 1-threshold function. Thus, once a node is infected it remains infected throughout the evolution of the graph and a node become infected if it has at least one infected neighbor.*

The figure shows the spread of the infection, i.e. the evolution of states in the graph over three time steps. The top row shows the configurations of the graph if the prevention policy π_1 *was implemented. The policy initially vaccinates the nodes (colored gray)* 4, 7, 9, *and* 10. *With the policy* π_1, *the infection is restricted to nodes* 1, 2, *and* 5, *thus protecting nodes* 8, 11−16 *from infection. The bottom row shows an alternative prevention policy* π_2 *which initially vaccinates*

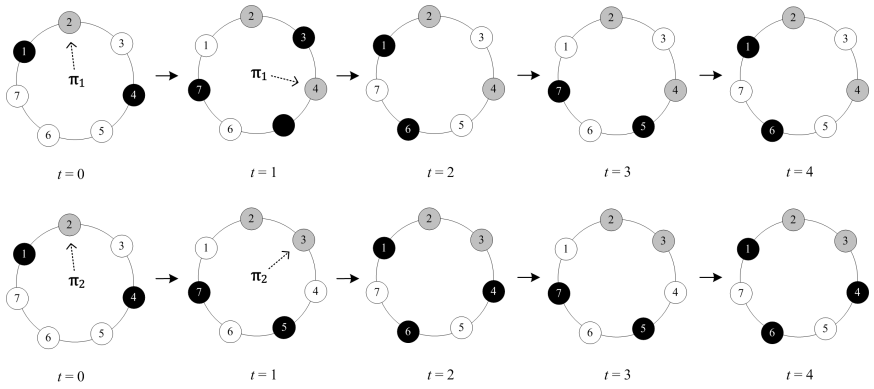

Figure 2.3 Infection Spread in a ring of 7 nodes

the nodes 4, 5, 7, *and* 10. *With policy* π_2, *the infection is restricted to nodes* 1 *and* 2 *while preventing the infection from spreading to node* 5 *as well as the nodes protected by* π_1.

Example 3 *Consider a social network of seven people in a ring topology as show in Figure 2.3. Suppose that at* $t = 0$ *nodes* 1 *and* 4 *are infected with a virus whose spread is governed by the 1-reversible 1-threshold infection transmission function. Thus a node become infected at* $t + 1$ *if at least one of its two neighbors is infected at time* t. *The transmission function* f *and the local update function* g *are as follows:*

$$f(\sigma(v_i)) = \begin{cases} \texttt{infected} & \text{if } \sigma(v_i) = \texttt{open} \text{ and} \\ & \exists v_j \in \rho(v_i) : \\ & \sigma(v_j) = \texttt{infected} \\ \sigma(v_i) & \text{otherwise} \end{cases}$$

$$g(\sigma(v_i)) = \begin{cases} \texttt{open} & \text{if } \sigma(v_i) = \texttt{infected} \\ \sigma(v_i) & \text{otherwise} \end{cases}$$

Suppose there is a medicine available that can cure a node if it is infected and prevent the node from being infected in the future (the node is protected). Due to supply constraints, only two such medicines are available to be administrated and at most one medicine can be used per time step. Figure 2.3 shows the evolution of the graph over three times steps with the application of two different intervention policies, π_1 *and* π_2. *Policy* π_1 *administers the first*

medicine to node 2 *at* $t = 0$ *and the second medicine to node* 4 *at time step* $t = 1$*. Policy* π_2 *also administers the first medicine to node* 2 *at* $t = 0$*, but administers the second medicine to node* 3 *in time step* $t = 1$*.*

The difference in policies leads to different infection behaviors. With policy π_1*,* $\forall t \geq 2$*, the infection is confined to two nodes in every time step. Specifically, at even time steps, nodes* 1 *and* 6 *are infected. At odd time steps, nodes* 5 *and* 7 *are infected. For policy* π_2*, the number of infected nodes oscillates between two and three for any time step* $t \geq 2$*. Nodes* 1*,* 4*, and* 6 *are infected at even time steps while nodes* 5 *and* 7 *are infected at odd time steps. If, in this case, the criteria for the effectiveness of policies is minimizing the number of infected nodes in each time step, we can conclude that policy* π_1 *is more effective than policy* π_2*.*

In all of the above examples, the initially infected nodes were all known. In other situations, only an approximation may be known for the total number of infected nodes, or the number of vaccines available at any time may vary at each time step. Even the total number of vaccines may be unknown as vaccine production is not a process with guaranteed yields (Falco (2009)). Thus, a policymaker must consider all possibilities for the set of initially infected nodes (and in the case of prevention policies, the set of protected nodes) in order to determine an effective intervention strategy. Moreover, among alternative policies, some may be more effective than others at controlling the spread of the infection in terms of the number of nodes in the graph that are protected from the infection. Yet measures of the effectiveness of policies to control the spread of the infection may also include such metrics as: the number of vaccines needed for the policy, the number of nodes saved per unit of vaccine administered at the beginning of the infection, the maximum number of nodes that become infected at each time step, and the number of critical nodes infected (nodes deemed to have a higher weight than others, i.e. the root DNS server versus a personal computer). All of the above considerations make it impossible for a policymaker to arrive at an effective policy without the aid of specialized computational techniques.

Given a graph $G(V, E)$, the infection propagation functions f and g, which together specify the evolution of the states in the graph G, we ask the following types of questions to control

the spread of an infection:

Q1 *Finding an intervention policy* Given an initial configuration of the graph where a set $I \subseteq V$ nodes is infected and a fixed number m, the number of nodes that can be protected initially, is there an intervention policy π, such that the infection does not spread to more than l nodes?

Q2 *Verifying a preventive policy* Given a preventive policy π specifying a set $P \subseteq V$ of protected nodes and a fixed number m, the number of initially infected nodes, does the policy π prevent the infection from spreading to more than l nodes?

Questions such as the ones above can be answered by modeling the spread of the infection over an arbitrary graph using Kripke structures and verification of the satisfiability of appropriate temporal logic properties using model checking techniques. We proceed to do so in the next chapter.

CHAPTER 3. ANALYZING POLICIES USING MODEL CHECKING

A common element of the questions at the end of the previous chapter is that they can all be answered by computing the reachability of a desired configuration of the graph or the non-reachability of any undesired configuration of the graph from all the possible initial configurations. We take advantage of the state-of-the-art approaches in model checking to verify the reachability or non-reachability of desired configurations of the graph. We first encode the transitions between configurations of the original graph, through which the infection is spreading, as input to a model checker (we use Spin (Spin (2010))). We then transform the queries regarding the existence or verification of policies in to a test of reachability in the encoded graph. We initially demonstrated this in (Santhanam et al. (2011)).

Given an initial configuration of a graph over which an infection is spreading, we construct a model in a language that the model checker accepts (Promela, in the case of Spin (Spin (2010))) as follows:

(1) The states of the nodes of the graph, over which the infection is spreading, are mapped to state variables of the model in the model checker.

(2) The allowed transitions between configurations, as dictated by the infection propagation functions f and g, are directly encoded as transitions in the model.

By (1), the configurations of the original graph correspond to states of the input model passed to the model checker. By (2), there is a one-to-one relation of the transitions between configurations in the original graph and the transitions between states of the input model passed to the model checker. This ensures that the model checker explores all possible evolutions of the graph with respect to the initial configurations. This corresponds to the simulation of all possible ways that the infection can spread over the nodes of the original graph. Such an approach

leads to a sound answer to the queries, as the model checker explorers all possible evolutions of the graph. Either an undesired configuration is reached or all reached configurations are desired.

Queries regarding the identification and verification of policies are formulated as linear temporal logic, LTL, formulas (Vardi (1996)) over the state-space of the model. We thus leverage the highly optimized algorithms for LTL model checking utilized by modern model checkers to efficiently verify the satisfiability of the corresponding LTL formulas.

3.1 Encoding Infection Spread in Kripke Structures

We use a *Kripke structure* (Clarke et al. (2000)) to model the transitions between the configurations of a graph.

Definition 2 (Kripke Structure) *A* Kripke *structure is a tuple* $\langle S, S_0, T, L\rangle$ *where* S *is a set of states described by the valuations of a set of propositional variables* P*,* $S_0 \subseteq S$ *is a set of* initial *states,* $T \subseteq S \times S$ *is a transition relation inducing directed edges between states such that* $\forall s \in S : \exists s' \in S : (s, s') \in T$*, and* $L : S \to 2^P$ *is a labeling function such that* $\forall s \in S : L(s)$ *is set of propositions that are* true *in* s*.*

Given a graph $G(V, E)$ with a set $V = \{v_1, \ldots, v_n\}$ of nodes, a set of states Σ of the nodes of the graph, and the infection propagation functions f and g, a Kripke structure K_G that captures all the possible transitions between the configurations of the graph G (with respect to f and g) is constructed as follows:

(1) The states S of K_G are defined by the valuations of propositions $P = \{\sigma(v_i) \mid v_i \in V\}$, where each $\sigma(v_i) \in \Sigma$ indicates the state (e.g., open, infected or protected) of the corresponding node $v_i \in V$ in the graph G. A state $s \in S$ is represented by the tuple $s = \langle \sigma(v_1), \ldots, \sigma(v_n)\rangle$. Thus, each state in the Kripke structure corresponds to a unique configuration of the graph G.

(2) The transition relation T is defined as follows. For any two states $s, s' \in S$, define $(s, s') \in T$ (denoted $s \to s'$) if $s = \langle \sigma(v_1), \ldots, \sigma(v_n)\rangle$ and $s' = \langle f(g(v_1)), \ldots f(g(v_n))\rangle$.

(3) The set of start states S_0 of K_G correspond to the initial configurations of the graph G, based on the type of query that is posed to the model checker. For example, if the query is of the type Q1 is posed, then each initially infected node $v_i \in I$ is set to the state $\sigma(v_i) =$ `infected` in every state of S_0. In the case of Q2, each initially protected node $v_i \in P$ is set to the state $\sigma(v_i) =$ `protected` in every state of S_0. This restricts the state-space explored by the model checker to only consider the required initial configurations.

(4) The labeling function in this construction is simply the tuple of valuations of the state variables: $L(s) = \langle \sigma(v_1), \ldots, \sigma(v_n) \rangle$, i.e., we identify the states of the Kripke structure precisely by the states of the nodes in the graph G.

The definition of the transition function prevents the creation of "'dead-end"' states in the Kripke structure. This does not effect the use of Kripke structures for encoding infection spread in the graph because, by (2), a transition in the Kripke structure is constructed by the application of the infection propagation functions f and g to every node $v_i \in V$. Since f and g are total functions, the construction of T does not violate the definition of a Kripke structure.

In the above encoding of the Kripke structure, the transition function rule ensures that the infection spreads exactly as defined by the infection propagation functions f and g. There is a bijection between the transitions in K_G and the change of configurations in the graph G in a single time step. The set of initial states S_0 consists of the set of configurations in which the state of a set of nodes is set to `infected` (e.g., for finding intervention policies) and/or the set of nodes whose state are set to `protected` (e.g., for verifying a prevention policy). It is possible to also to set the state of certain nodes to `open` in the initial configuration to explore scenarios in which these nodes must remain free of infection but cannot themselves be in the state `protected`. The model checker checker considers every possible instantiation of the Kripke structure corresponding to the set of initial start states S_0 in order to verify a desired property. Note that for the types of infection spread problems considered in this paper the corresponding infection propagation functions are deterministic. Given a configuration of the graph G and the infection propagation functions f and g, there is a unique configuration to which the graph will transition to in a single time step. Thus, given a state in S_0 (an initial

configuration in the graph), there is exactly one possible evolution of the graph (a path in the Kripke structure). A path in a Kripke structure is defined as follows:

Definition 3 (Path) *A* path δ *in a Kripke structure* $\langle S, S_0, T, L\rangle$ *is an infinite sequence of states* $\delta = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ *such that* $\forall s_i$ *in* $\delta : \exists(s_i, s_{i+1}) \in T$

Thus a path in a Kripke structure encoding of a graph G corresponds to an evolution of the graph G from the given initial configuration.

Example 4 *The Kripke structure corresponding to the graph in Example 2 is given by* $K_G = \langle S, S_0, T, L\rangle$ *as follows. Each state in* S *is a tuple* $s = \langle\sigma(v_1), \ldots, \sigma(v_n)\rangle$ *where* $\sigma(v_i) \in \Sigma$ *represents the state of the node* v_i *in the graph* ($\Sigma = \{open, infected, protected\}$). *Note that in Example 2, the irreversible 1-threshold process is used for the infection propagation functions. Thus, if at least one of the neighbors of node* v_i *is the infected state in state* s*, in any state* s' *where* $\exists(s, s') \in T$*, the state of node* v_i ***must*** *be* `infected` ($\sigma(v_i) =$ `infected`). *Hence, transitions in the Kripke structure correspond to changes in the configuration of the graph. For the top row of Figure 2.2 the transition from the configuration in* $t = 0$ *to the configuration in* $t = 1$ *is* 0012012022000000 $\rightarrow$ 0112112022000000 *(where* 0,1,2 *represent the states* `open`, `infected`, *and* `protected` *respectively). Similarly, in the bottom row, the transition from the configuration in* $t = 1$ *to the configuration in* $t = 2$ *is* 0112212002000000 $\rightarrow$ 1112212002000000.

The above Kripke structure can be encoded in Promela, the model language of Spin (Spin (2010)). We now show how model checking algorithms, through the use of linear temporal logic, can be used to find and verify policies in response to an infection spread.

3.2 Finding & Verifying Policies using LTL

Given a Kripke structure K_G that encodes the spread of an infection in a graph $G(V, E)$, finding and verifying policies can be reduced to verifying corresponding temporal properties in linear temporal logic (LTL, see (Vardi (1996))). The syntax of LTL is defined over a set of propositions `Prop`; boolean `true`, negation ($\neg$) and OR ($\vee$); and temporal operators **X** and **U**

as follows:

$$\varphi \rightarrow \texttt{true} \mid \texttt{Prop} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi$$

The semantics of LTL are defined in terms of the set of paths that satisfy the given formula. Let δ be a path where $\delta[i]$ denotes the i-th state in the path and δ^i denotes the suffix of the path starting from $\delta[i]$ ($\delta^0 = \delta$). For example, if $\delta = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ then $\delta^1 = s_1 \rightarrow s_2 \rightarrow \ldots$. The following are the rules for the evaluation of whether a path satisfies a given LTL formula. Included are the rules for the convenience temporal operators **F** and **G**.

`true` – Any path satisfies `true`

`Prop` – A path δ satisfies `Prop` if and only if $\delta[0]$ satisfies `Prop`

$\neg\varphi$ – A path satisfies $\neg\varphi$ if the path does not satisfy φ

$\varphi_1 \vee \varphi_2$ – A path satisfies $\varphi_1 \vee \varphi_2$ if the path satisfies either φ_1 or φ_2

$\mathbf{X}\varphi$ – A path δ satisfies $\mathbf{X}\varphi$ if $\delta[1]$ satisfies φ

$\varphi_1\mathbf{U}\varphi_2$ – A path satisfies $\varphi_1\mathbf{U}\varphi_2$ if $\exists j \geq 0 : \delta[j]$ satisfies φ_2 and for all $i < j : \delta^i$ satisfies φ_1

$\mathbf{F}\varphi$ – A path δ satisfies $\mathbf{F}\varphi$ if $\exists j : \delta^j$ satisfies φ. $\mathbf{F}\varphi$ equivalent to $\texttt{true}\mathbf{U}\varphi$.

$\mathbf{G}\varphi$ – A path δ satisfies $\mathbf{G}\varphi$ if $\forall j \geq 0 : \delta^j$ satisfies φ. $\mathbf{G}\varphi = \neg\mathbf{F}(\neg\varphi)$

F and **G** are duels of each other. Specifically, $\mathbf{F}\varphi = \neg\mathbf{G}(\neg\varphi)$ and $\mathbf{G}\varphi = \neg\mathbf{F}(\neg\varphi)$.

A Kripke structure satisfies an LTL formula φ if and only if all paths starting from all its start states satisfy φ (K_G satisfies φ if $\forall\delta$ such that $\delta[0] \in S_0 : \delta$ satisfies φ).

In the next section we will consider the LTL formulas that correspond to queries of types Q1 and Q2 (see Section 2.2). For each of the query types, the initial states (e.g., `infected` or `protected`) of some of the nodes in the graph are pre-specified. In queries of type Q1, the set $I \subseteq V$ specifies the set of nodes which are in the `infected` state in all initial configurations. Similarly, queries of type Q2 define a set $P \subseteq V$ which specifies which nodes are in the `protected` state in all initial configurations. This information is encoded in the states of S_0 in the Kripke structure K_G. Thus, the model checker is guided to only consider initial states that correspond to initial configurations of the graph G. Furthermore, in some query types, exact

nodes are not specified as infected or protected. Rather, the maximum number of infected or protected nodes is given. This implies that different combinations of nodes in the graph can be infected or protected at the start of the infection spread. We encode this information by allowing the model checker to non-deterministically initialize the states of the nodes at the beginning of the infection spread. The model checker thus checks every initial configuration that respects the imposed constraints. We now look at the specifics of the query types Q1 and Q2.

3.2.1 Q1: Finding an intervention policy

In this query type, the set $I \subseteq V$ of initially infected nodes and a fixed number m of nodes that can initially be protected are pre-specified. From this we derive the following set of start states:

$$S_0 = \{\langle \sigma(v_1), \ldots, \sigma(v_n) \rangle \mid \forall v_i \in I : \sigma(v_i) = \texttt{infected} \wedge |\{v_i \mid \sigma(v_i) = \texttt{protected}\}| = m\}$$

The first term of the condition precisely defines the initially infected nodes as the ones belonging to the set I. The second term, $|\{v_i \mid \sigma(v_i) = \texttt{protected}\}| = m\}$ specifies the number of nodes to be protected rather than specifying the specific nodes. This gives rise to multiple combinations of nodes that can be set to `protected`. Specifically, if there are n nodes in the graph G and initially i nodes are designated as infected, there will be

$$\binom{n-i}{m}$$

states in the set S_0. The model checker is able non-deterministically initialize the start states in accordance with these conditions.

Let $total_I$ be the total number of infected nodes at any given time step. Since we are looking for a policy π such that at most l nodes are infected at any given time step, we use the LTL formula $\varphi : \mathbf{F}(total_I > l)$. The Kripke structure satisfies the formula φ if in every path starting from every start state (i.e. every state in S_0) there exists a state where $(total_I > l)$ holds. Note that the satisfaction of this LTL formula implies that there does not exist a policy π which restricts the infection to no more than l nodes in any time step. Any combination of

m protected nodes leads to a state of the Kripke structure where the number of infected nodes is at least $l+1$. On the other hand, if the formula φ is not satisfied, this implies that there is an initial state $s_0 \in S_0$ such that in every state of every path beginning from state s_0, there are no more than l infected nodes. As a counterexample, the model checker will output an initial state $s_0 = \langle \sigma(v_1), \ldots, \sigma(v_n) \rangle$ and the paths from this state for which $total_I \leq l$ held in every state. The desired policy π can be constructed from this counter example by assigning to the node v_i the state `protected`, if $\sigma(v_i) =$ `protected` in s_0.

The specified formula φ can be used to find the intervention policy π if the infection propagation functions f and g are monotonic, i.e. a node cannot change its state from `infected` to `open`. Under such a constraint, the number of infected nodes, $total_I$, can never decrease (see Example 2). However, in the case of r-Reversible k-Threshold processes a node changes its state from `infected` to `open` after it had been infected for r time steps (see Example 1). This is the case because the local update function g is not monotonic. As a result, $total_I$ may increase or decrease at each time step, as seen in Figures 2.1 and 2.3. In the bottom row of Figure 2.1 the 1-reversible 2-threshold process is illustrated and the number of infected nodes decreases from the time step $t = 0$ to $t = 1$. Similarly in the bottom row of Figure 2.3 the 1-reversible 1-threshold process is illustrated and the number of infected nodes oscillates between two and three in every time step after $t = 0$.

In such a scenario the policymaker, instead of looking for a policy π in which the number of infected nodes never exceeds l, may choose to look for a policy where the number of infected nodes stabilizes. Specifically, the query "'Is there a policy π such that the number of infected nodes is always at most l, *after a certain number of time steps*?"'. To answer this query, we use the LTL formula $\varphi' = \mathbf{GF}(total_I > l)$. The Kripke structure satisfies the formula φ' if for every path beginning from a start state (all states in S_0), $total_I > l$ holds an infinite number of times. As with the formula φ above, the satisfaction of the formula φ' implies that no policy π satisfies the query. The non-satisfaction of formula φ' implies that there exists a start state $s_0 \in S_0$ whose path (recall that exactly one path from every start state) does not satisfy φ', i.e., the path satisfies the formula $\neg\varphi'$. $\neg\varphi' = \mathbf{FG}(total_I \leq l)$ meaning that the path from state s_0 eventually reaches a state after which $total_I \leq l$ holds in every state. The policy can

be obtained from the counterexample produced by the model checker as in the case of the irreversible k-threshold process.

3.2.2 Q2: Verifying a preventive policy

In this query, a policy π specifying the set $P \subseteq V$ of initially protected nodes along with a fixed number m of initially infected nodes are pre-specified. We encode the set of start state S_0 in a similar fashion to the query Q1.

$$S_0 = \{\langle \sigma(v_1), \ldots, \sigma(v_n) \rangle \mid \forall v_i \in P : \sigma(v_i) = \texttt{protected} \wedge |\{v_i \mid \sigma(v_i) = \texttt{infected}\}| = m\}$$

As in the query Q1, the condition $|\{v_i \mid \sigma(v_i) = \texttt{infected}\}| = m$ allows the model checker to non-deterministically explore all possible combinations of m initially infected nodes. Thus the model checker tests the policy against every possible infection scenario.

Verification of a given policy π is achieved through the LTL formula $\psi : \mathbf{G}(total_I \leq l)$. The formula ψ is satisfied if and only if in every path beginning from every initial state (all state in S_0), $total_I \leq l$ holds. If the formula ψ is satisfied, then the policy π is successfully able to contain any infection outbreak of m nodes, such that no more than l nodes are infected at a given time. If, on the other hand, the formula ψ is not satisfied, this implies that there is a start state $s_0 \in S_0$ in the Kripke structure, in which m nodes were initially infected, for which the policy fails. At least $l + 1$ nodes become infected in some state of the path originating from s_0. The model checker provided counterexample specifies an initial infection outbreak for which the policy is ineffective, allowing the policymaker to change the policy.

Similar to the case of Q1, queries of type Q2 posed against models in which the infection propagation functions are non-monotonic (e.g for infection spread using the r-Reversible k-threshold processes), may cause the policymaker to verify if the given policy satisfies the stability condition: the total number of infected nodes is at most l nodes, *after a certain number of time steps*. The policy can be verified against the new criteria using the LTL formula $\psi' = \mathbf{FG}(total_I \leq l)$. If ψ' is satisfied, that means the policy π is successful at containing the infection to at most l nodes after a certain amount of time steps. If ψ' is not satisfied, there exists a path beginning from a state $s_0 \in S_0$ for which the formula does not hold. The model

checker provides the initial state s_0 as a counterexample in which, with the initially infected m nodes, there is a path in which the total number of infected nodes is greater than l infinitely often (the path satisfies the formula $\neg\psi' = \mathbf{GF}(total_I > l)$). The counterexample provides the policymaker with precise information as to the reason the policy failed. This allows the policymaker to make adjustments to the policy or develop alternative policies.

CHAPTER 4. REGION-BASED PROPAGATION ANALYSIS

The model checking approach to policy identification and verification allows the policymaker to accurately and easily find and test infection prevention policies. However, the model checking approach is not feasible in all situations. For example, suppose an outbreak occurs in a network of 10,000 people. Initially 50 (0.5% of the population) people are infected with a new disease and vaccine yields allow the vaccination of up to 250 people (2.5% of the population). Let $G(V, E)$ be the graph modeling this scenario and K_G the Kripke structure encoding of G. The policymaker is tasked with implementing an intervention policy (a query of type Q1, see Section 2.2). In such a scenario there are several reasons for which the model checking approach fails.

(1) It is not feasible for the policymaker to deal with the entire network of people as a whole. Even a graph of a thousand nodes is nearly impossible to display on a computer monitor. Automated techniques such as the ones in this paper help reduce the strain on the policymaker, however even such techniques cannot completely negate the impact of large graphs.

(2) While modern checking algorithms are extremely efficient, they cannot deal with 10,000 variables. Since the exact people infected are pre-specified, there are

$$\binom{10,000 - 50}{250}$$

initial configurations of the graph G. This translates to 3.7×10^{505} initial states in S_0 of K_G! While there are techniques for reducing the number of nodes in the graph (see optimizations in Section 5.1.1), they cannot completely alleviate the problem.

(3) The model checking approach does not allow the policymaker to make a distinction between the infected nodes. Even if an intervention policy does not exist, the policymaker may be interested in at least curbing the spread of the infection around some subset of the infected nodes.

For these reasons, the policymaker is likely to subdivide the network into smaller *regions*. Division of the network in to regions resolves the aforementioned concerns. By creating regions, the policymaker is able to focus his attention on only portions of the network, thus eliminating concern (1). Model checking each individual region reduces state-space significantly. For example, if a region of the network contains 300 nodes, 2 of which are infected, and the policymaker applies 10 vaccines to the region (if, for example, the policymaker chooses to distribute vaccines based on the number of infections in the region), then there are only

$$\binom{300-2}{10} = 1.3 \times 10^{18}$$

possible initial configurations. This is over 480 *magnitudes* better than model checking the entire graph and mitigates the concern of (2). Finally the policymaker is free to allocate the portion of available vaccines as he sees fit to each region. Thus a region with more critical nodes (for example a hospital where an infection epidemic would be particularly devastating) may receive a larger portion of the vaccines. Fine grained control over vaccine distribution allows the policymaker to make better informed decisions and addresses point (3).

4.1 Region Generation for Intervention Policies

The central problem of applying regions to the control of infection spread is defining a suitable region generation algorithm. While the policymaker will always have the final say in the construction of regions, automated region generation would be an invaluable tool for simplifying the process of combating infection spread.

In the case where the query is of the type Q1 and the policymaker is seeking an intervention policy, the regions of a network take on an additional role, identifying the nodes that must be quarantined. In an outbreak of an infection, whether it is a fire, opinion, computer virus, or

disease, the first step is a quarantine of the affected entities. A quarantine guarantees that even if the localized intervention policies are ineffective, the entire network does not succumb to the infection. For example, in the case of an outbreak of a disease that poses significant risk to human health, the infected and those they have come in contact with may have travel restrictions imposed.

The effectiveness of quarantine depends on the ability to completely isolate the infection spread to a region. In practice it takes a certain amount of time to cordon off a region of a network. The more time spent in establishing a quarantine, the further the infection has time to spread. Thus the quarantine must encompass more entities. Conversely, the more time passes since the initial outbreak, the more resources a policymaker may gather in affecting the quarantine. Thus, the policymaker can protect more entities from getting infected.

Based on these observations, a region is defined to be a set of entities directly or indirectly connected to infected entities such that the infection spread can be controlled to remain within this set of entities. In other words, a region provides some insight on the entities to be quarantined.

4.1.1 Region Generation Algorithm

Our approach to region generation is an iterative one. As input we take a node v, for which the region is computed grown. The containment function h represents the resources that a policymaker has to enforce a containment of the region. Thus, with every iteration we increase the the size of the region as long as the region fails to satisfy the containment function.

Given a graph $G(V, E)$, a containment function h, and a node v, the region centered at v is generated as follows:

At every iteration, the algorithm computes the set of nodes R'. R' is the set of all neighbors of the nodes in R, excluding themselves. If the size of R' is less than or equal to the value of the containment function at this iteration, the algorithm terminates and returns the set R, the region centered at node v. Otherwise, the nodes in R' are added to R and the counter variable for the containment function is incremented.

In each iteration, the region effectively grows to encompass all its neighbors. From the

```
1: R ← {v}
2: count ← 1
3: while true do
4:    R′ ← ∅
5:    for all v1 ∈ R do
6:       for all v2 neighbors of v1 do
7:          R′ ← R′ ∪ {v2}
8:       end for
9:    end for
10:   R′ ← R′ − R
11:   if |R′| ≤ h(count) then
12:      return R
13:   else
14:      R ← R ∪ R′
15:      count ← count + 1
16:   end if
17: end while
```

aspect of a containment / quarantine policy, this is the maximum achievable spread of the disease in a single time unit. A quarantine must encompass at least all these nodes, thus preventing the infection from escaping the quarantine. Effectively, the region growth assumes worst case infection spread. With each time step, the number of outside connections the containment function can handle varies. For monotonically increasing functions, with each time step the containment function is able to handle more outside connections. The choice of containment function has a large influence on the resulting region. A faster growing function, such as $h(x) = 2^{2^x}$, will generate a smaller region around the node v than a slower growing function such as $h(x) = x^2$. A fast growing function represents a higher quarantine priority and higher resource allocation for the infected node than a slower growing containment function.

The correct choice of the containment function is crucial for the creation of manageable regions. A highly connected network, such as a social network, requires a rapid response to contain an infection spread. On the other hand, a road network where nodes represent intersections of roads and edges are the roads connecting intersections, achieves manageable region sizes with a slower growing containment function. This is from the simple fact that most intersections have four or less outgoing roads.

CHAPTER 5. EXPERIMENTS & RESULTS

We now describe the results of our preliminary experiments for identifying intervention policies and the computation of regions. We first present the results for the identification of intervention policies, as well as the optimizations that make this approach feasible.

5.1 Results of Identifying Intervention Policies

We have developed a Java preprocessor that takes as input the network, the initial configuration, and optionally the policy to be verified. Outputted is a Kripke structure encoding of the model in Promela, the language of Spin (Spin (2010)). The model is generated such that the model checker explores only those states where the condition $total_I \leq l$ holds. This is sufficient for the model checker to output a correct solution because the reachability of a state where $total_I \leq l$ does not hold along a path, indicates the non-existence of an intervention policy.

Table 5.1[1] shows the results of our implementation for networks of 40 nodes with $40, 60, 70$ and 80 edges (E) randomly generated such that the degree of each node is ≤ 5. In each network, 10 nodes were randomly selected and set to the `infected` state. We tested each network with a query of type Q1 (see Section 2.2): $\mathbf{F}(total_I > l)$ with $l = 10$ and $l = 20$. The number of `protected` nodes, m, was 20 in all cases. The experiment was repeated ten times for each combination of inputs and the results were averaged.

The results show that the model checker is able to identify intervention policies, if they exist, within a minute for most test cases. A longer time indicates the traversal of more initial states and paths of the Kripke structure in search of an intervention policy. For fixed l, the number of states explored by the model checker increases as the number of edges in the network

[1] All experiments were conducted using Intel i7 3.528 GHz processor with 6GB memory on 64 bit Kubuntu 10.10 OS.

E	States ($\times 10^6$)		Time (secs.)		Memory (GB)	
	10	20	10	20	10	20
40	0.82 (10^{-5})	10^{-4} (10^{-5})	2.54 (10^{-3})	10^{-3} (10^{-3})	0.07 (10^{-3})	10^{-3} (10^{-3})
50	12.24 (10^{-3})	10^{-3} (10^{-2})	52.20 (10^{-3})	0.02 (0.02)	1.08 (10^{-3})	10^{-3} (10^{-3})
60	42.01 (6.14)	0.10 (0.12)	201.98 (12.76)	0.43 (0.23)	3.68 (0.51)	0.01 (0.01)
70	59.17 (30.73)	0.62 (0.02)	270.20 (87.38)	2.75 (0.05)	5.18 (2.53)	0.05 (10^{-3})
80	56.33 (25.93)	45.66 (23.46)	272.00 (65.00)	193.04 (49.47)	4.99 (2.14)	4.05 (1.93)

Table 5.1 Results for random networks with 40 nodes. Numbers in parentheses denote results with optimizations.

increases. This is because increasing the number of edges in a network increases the ways an infection can spread. The more connected a network, the faster an infection is able to spread to new nodes. On the other hand, as l increases from 10 to 20, for fixed number of edges, the number of traversed states decreases. This is because, with an increased value for l, more intervention policies become valid. Since the model checker terminates once it locates a single valid intervention policy, an increase in valid policies leads to a decrease in the number of states searched.

The amount of time spent and the amount of memory used is proportional to the number of states traversed. This is the case for time because the search for an intervention policy is a search over the state-space of the model. In the case of memory, it is because Spin stores every traversed state without discarding any states (Spinroot (2011)). For most test cases memory use remains under one gigabyte.

5.1.1 Optimizations to Improve Scalability

We developed several optimizations that can be incorporated into the preprocessing step. These optimizations are specifically designed to allow a faster search for intervention policies in scenarios where infections are irreversible, such as the irreversible k-threshold process.

Single-Step Search to Detect Non-existence of Policy: If the number of nodes that can be be in the `infected` state in the first time step ($t = 1$) exceeds the number of nodes that can be protected at the outset by more than a specified threshold for the intervention policy, then we can infer the non-existence of any intervention policy (without deploying the model checker). Formally, if the number of initially infected nodes is i, the number of nodes infected in the first time step if there are no protected nodes is I_{max}, the number of initially protected

nodes m, and the threshold for the intervention policy l; then no intervention policy exists if $i + I_{max} - m > l$. This is the case because in a single time step, a node in the `protected` state cannot influence the state of any other node. Thus in the first time step there would be, at minimum, $i + I_{max} - m$ infected nodes.

Iterative Bounded Search: The main idea with this optimization is to guide the model checker's traversal of the initial states in the search of an intervention policy. We do this by restricting, in each iteration, the nodes which the model checker can set to `protected`. In iteration i, we consider all the non-infected nodes that are $\leq i$ edges away from any infected node as *candidates* to be protected (as opposed to *all* nodes in the network). This strategy yields a trivial policy in the first iteration, if the number of candidate nodes is less than or equal to the number of nodes that can be protected at the outset (Protecting every neighbor of every infected nodes guarantees the infection cannot spread to any new node). The iteration is continued (for $i = 1, 2, \dots$) until an intervention policy is obtained or the i is larger than the maximum distance between any two nodes in the network (if at this point no intervention policy has been found, the non-existence of the intervention policy is proven). This strategy guides the model checker's model exploration in a manner that ensures an intervention policy, if one exists, is obtained (faster) with minimum exploration of the parts of the model state-space that do not contribute to the finding of an intervention policy. Note that in the worst case scenario, when the maximum iteration is reached, the model checker treats all non-infected nodes as candidates. Thus this strategy is guaranteed to find an intervention policy if one exists.

Node Merging: This optimization is based on the simple observation that when two adjacent nodes v_1 and v_2 in G are uninfected and unprotected, and of the nodes (say v_2) has no neighbors other than v_1, then v_2 can be merged with v_1 without affecting the answer to the query. When i nodes adjacent to v_1 are thus merged, we annotate the node v_1 with i indicating that if v_1 become infected in time step t, then each of the i nodes adjacent to v_1 in G are infected at time step $t+1$. If v_1 or one of the i adjacent nodes is to be protected, the optimal strategy is marking v_1 as protected as opposed to one of its neighbors. This prevents the spread to the adjacent i nodes since they only have one neighbor. By treating v_1 and its neighbors as one state in the model checker, we insure that the model checker chooses the most efficient intervention

policy (in such situations) and reduce the number of states the model checker must explore (the model checker no longer needs to traverse states where it designates the neighbors of v_1 as protected). While the optimization reduces the number of states in the model, any solution obtained for the optimized model remains a valid for the original model. Note this strategy cannot be applied in settings where nodes can be protected after the spread of the infection is already underway.

Removal of Non-Infectable Nodes: This optimization is valid for any k-threshold process. If a non-infected node has less than k neighbors, the node can be removed from the model since the node cannot be infected. A non-infectable has no effect on the spread of an infection since it can neither be infected nor spread the infection. Note that any node with no neighbors can be removed from the model.

Removla of Distant Nodes: This optimization is derived from the observation that when dealing with irreversible infections, if the number of infected nodes does not change between two time steps on a given path, the number of infected nodes will remain constant for the rest of the states in the path. Thus, the slowest infection must spread to one non-infected node in each time step. If the infection threshold of the intervention policy is l, then any non-infected node whose distance from any infected node $> l + 1$ need not be considered by the model checker. This is because the model checker terminates looking at a path if at any state the number of infected nodes exceeds l, thus it will never look at a node v which requires at least $l + 1$ nodes to be infected before v can become infected. More accurately, the maximum distance to be considered is not $l + 1$, but $l - i + 1$ if i is the number of initially infected nodes. By removing any nodes whose minimum distance to an infected node exceeds $l - i + 1$, we are able to reduce the state-space without affecting the validity of the solution.

Removal of Infected Nodes: When dealing with a model of infection spread where the infection is irreversible, it is not necessary to maintain state variables for initially infected nodes in the model. That information is encoded in the transition rules of the neighbors of the infected nodes. This reduces the number of state variables, thus leading to better memory performance in the model checker.

V	E	States ($\times 10^6$)		Time (secs.)		Memory (GB)	
		30	40	30	40	30	40
10 Infected nodes, 20 protected nodes							
80	119.5	0.07	10^{-4}	0.18	10^{-3}	10^{-2}	10^{-3}
90	138.0	3.46	6.76	0.83	27.98	10^{-2}	0.63
100	149.4	5.46	10^{-3}	29.00	10^{-3}	0.51	10^{-3}
20 Infected nodes, 10 protected nodes							
80	120.9	12.29	29.33	26.59	112.66	1.01	2.46
90	133.6	6.14	30.04	13.61	149.10	0.51	2.53
100	151.5	10^{-6}	27.99	0.00	146.67	10^{-3}	2.53

Table 5.2 Results for scale-free networks.

5.1.2 Effectiveness of Optimizations

We performed a new set of experiments by applying the above optimizations on random graphs with the same parameters as shown in Table 5.1. The new results (for states explored, time spent, and memory used) are shown in parenthesis in Table 5.1. The optimizations led to a decrease in the states explored, time spent, and memory used by the model checker of about 50%. The optimizations made it possible to scale our approach to random networks with 40 to 100 nodes, which was not possible without the optimizations.

It should be noted that networks in the real world (e.g., social networks, the Internet, the power grid), tend to exhibit scale-free topologies and hierarchical modularity (Ravasz and Barabási (2003)). Scale-free networks are characterized by having a few highly connected *hub* nodes and the rest of the nodes with a lower degree of connectivity. The degree distributions in such networks generally follow a power law distribution (Ravasz and Barabási (2003)).

We have done some preliminary experiments to assess the effectiveness of our approach to finding policies in scale-free networks. In our experiments we used randomly generated scale-free networks with 80 to 100 nodes (V), with two combinations of initial configurations (10 infected nodes, 20 protected nodes; and 20 infected nodes, 10 protected nodes), and threshold values, l, of 30 and 40. We used the freely available Java library, JGraphT for scale-free network generation (Naveh (2005)). As with the previous experiment, each initial configuration was run ten times and the average of the results taken. The corresponding results are presented in Table 5.2. The reason the number of edges (E) in each graph is not a whole number is because randomly generated scale-free networks do not necessarily have the same number of edges. Thus, the number of edges shown, is the average number of edges over the ten runs for

each graph size.

We note that the time and space use for some of the experiments is extremely low (see Table 5.2, 100 nodes with 20 initially infected nodes and a threshold l of 30). This is the result of the aforementioned *Single-Step Search to Detect Non-existence of Policy* optimization. The Java preprocessor detected the non-existence of an intervention policy and thus terminated the experiment run without invoking the model checker. Also, it can be seen that in the case of 20 infected nodes (and 10 protected nodes) that for any given network size, more resources are utilized when the threshold is $l = 40$ than when it is $l = 30$. Since the threshold is higher, it takes more time steps for the infection to spread beyond the maximum threshold. This causes the model checker to travele further along each path from an initial state in the model before discarding the path. The above results were originally published in Santhanam et al. (2011). We have expanded on the list optimizations we implemented to achieve these results.

5.2 Results of Region Generation

We now proceed to describe the results of our preliminary experiments on region generation. The purpose of our experiments was to observe the properties of the regions generated by various containment functions. To this end, we implemented the algorithm in Section 4.1.1. A loader program was written to take a graph as input from a file where each line contained two nodes separated by a tab character denoting a single edge of the graph. The region algorithm was then run on every node of the graph and aggregate results taken. To facilitate region computation on every node in a graph, the neighbors of every node $v \in V$ were precomputed and stored in a map (where the key is a node v and the corresponding value is the set of neighbors of v), thus eliminating a costly search over all the edges of the graph in every iteration of the region generation algorithm.

Four real word networks were analyzed in our experiment. Networks *CA-GrQc* and *CA-CondMat* represent the author collaborations in two categories of the arVix publication. In both networks authors represent nodes, and undirected edges are formed if an author a co-authored a paper with author b. Network *email-Enron* contains nodes which represent email addresses of the Enron corporation and the undirected edges represent email communication

Graph	Nodes	Edges	Memory without Map (MB)	Memory with Map (MB)
CA-GrQc	5242	28980	32.79	39.82
CA-CondMat	23133	186936	101.44	112.49
email-Enron	36692	367662	155.81	171.83
roadNet-CA	1965206	5533214	1777.81	2678.42

Table 5.3 Region Storage Requirements.

Graph	Nodes	Edges	Average Region Size					
			x^2	x^3	2^x	3^x	4^x	2^{2^x}
CA-GrQc	5242	28980	2718	2097	2638	1675	828	76
CA-CondMat	23133	186936	18425	16883	18342	15860	12894	903
email-Enron	36692	367662	29725	28869	29717	28467	27129	8509
roadNet-CA	1965206	5533214	104	4.4	18	4.0	3.8	3.8

Table 5.4 Average region sizes for various containment functions.

between email addresses. Finally, *roadNet-CA* is a network of the California road map with nodes representing intersections (or road end-points) and undirected edges representing roads connecting the intersections. All networks were obtained through the Standford Network Analysis Project (Leskovec (2011)).

Table 5.3[2] shows the memory storage requirements of the four networks. To distinguish between the amount of memory required to load the network in to memory (for the computation of a single region) and the amount of memory required to store the network plus the map of neighbors of each node (when computation of many regions is desired), two columns are provided. Memory use data was collected by pausing the execution of the Java program at the point the entire network was loaded in to memory and observing *java.exe* memory usage in the Windows Task Manager. Memory usage may vary based on the JVM and operating system used. As expected, the amount of memory required to store a network increases with the increase in the number of nodes and edges in the network. Similarly, the amount of additional memory required to store the mapping between nodes and their neighbors is directly related to the number of edges in the network. The memory usage results demonstrate that region generation is feasible on modern computers. Even a network with almost two million nodes and five and a half million edges requires about 1.8 gigabytes of memory for the computation of individual regions.

We chose six different containment functions to explore: $x^2, x^3, 2^x, 3^x, 4^x$ and 2^{2^x}. The

[2]All region experiments were conducted using Intel i7 3.528 GHz processor with 6GB memory on Windows XP 64 OS.

results of the mean region sizes are shown in Table 5.4. As expected, the faster growing containment functions generate regions of progressively smaller sizes. A special note on the results of the functions x^3 and 2^x needs to be made. The results for all networks show larger regions generated with the function 2^x than with the function x^3. This follows from the fact that it is not until $x = 10$ that $2^x > x^3$.

From the region sizes, it is possible to infer information about the underlying network, specifically the level of connectivity amongst nodes. The more connected the network is the faster the region grows. It is not until the containment function is able to catch up the region growth that the region algorithm terminates. Thus networks that demonstrate large region sizes despite rapidly growing containment functions have a higher degree of connectivity than other networks. For example, the connectivity of the California road map network is much smaller than the other networks. This stems from the fact electronic networks allow for much larger connectivity than physical networks and the fact most intersections do not connect more than four roads. The more connected a network is, the faster a quarantine response must be to restrict the infection spread to a small region.

The amount of time region computation takes is dependent on the size of the network and the final size of the region. However, in all cases, the computation of individual regions did not take more than one second. Thus, a policymaker can try multiple containment functions for various region growth within a reasonable amount of time.

CHAPTER 6. RELATED WORK

In this chapter we explore related work in the field of modeling infection propagation. We use a discrete model of infection spread in the network with k-threshold processes to model the spread of infections in networks of computers, people, and trees (Dreyer and Roberts (2009)).

Gary MacGillivray and Ping Wang modeled fire spread in a graph with a tree topology utilizing the irreversible 1-threshold process for the infection propagation functions (MacGillivray and Wang (2003)). Similarly, Ping Wang and Stephanie A. Moeller looked at the spread of a fire in two dimensional and three dimensional grid topologies (Wang and Moeller (2002)). In both cases, the policy considered is a variation of the intervention policy, such that only one node could be marked *protected* in each time step. The papers do not provide experimental results for the location of an effective intervention policy. However, with only a small modification to our approach, allowing the placement of vaccines at any time step, we can provide experimental results for their work.

Much work has been done on the spread of diseases in human networks (Dreyer and Roberts (2009), Newman (2002), Arino and van den Driessche (2003), Hethcote and Driessche (1995)). The main focus of this work is usually on continuous model pioneered by Kermack and McKendrick (1927). In such models, the population is divided in to categories such as *susceptible*, *infected*, and *recovered* and functions are defined to model the transfer of people from one group to another. Unlike our approach, this model does not simulate interactions at the individual node level, for example "'10% of the members move from *susceptible* to *infected* in a time step"' as opposed to "'persons $1, 7, 21$ move to the *infected* state in this time step"'.

Barrett et al. (2009a) have developed techniques for generating networks based on real life social interaction networks. The networks thus generated treat individuals as nodes and interactions as edges, however the edges are labeled with the time of the interaction. Thus the

network generated has a temporal element. These synthetic networks will be good inputs for testing our methods, once we modify our approach to account for temporal changes in networks (see the Future Work Section 7.1).

The *SimDemics* simulator described Barrett et al. (2008) and Barrett et al. (2009b) allows the simulation of probabilistic infection spread (a node is infected with a certain probability p) in large networks containing temporal data. While Barrett et al. (2008) provides a game-theoretical approach to locating policies, the game itself is not implemented within *SimDemics*. Thus, *SimDemics* is able to simulate the spread of an infection and the effectiveness of a policy, but is unable to itself locate an effective policy.

Our approach provides a framework by which we are able to locate intervention policies and verify prevention policies in discrete networks instead of treating populations as continuous variables (Kermack and McKendrick (1927)). Our approach for modeling infection propagation in discrete networks can be used to model fire outbreaks as described in works by Wang et al ((MacGillivray and Wang (2003)), Wang and Moeller (2002)). Furthermore, by augmenting Kripke model for infection spread with real-time constraints and probabilistic choices, our approach can be extended to incorporate both temporal discrete networks (Barrett et al. (2009a)) and probabilistic infection propagation. As part of future work, we plan to investigating such extensions and their applicability in practice.

CHAPTER 7. CONCLUSION

We have presented a practical solution to the problem of finding and verifying policies for controlling the spread of infections (diseases, computer viruses, opinions, fires) in networks. Our approach encodes the spread of an infection in a network. The spread of infection is encoded in a Kripke structure where each change in the configuration (a tuple of the states of every node in the network) corresponds to a transition in the Kripke structure. This allows us to reduce the problem of identifying intervention policies and verifying prevention polices to the problem of model checking temporal properties, such as the number of infections at each time step is less than the threshold, on a Kripke structure. Furthermore, by taking advantage of the model checker's ability to identify counterexamples that demonstrate exactly why the given temporal property is not satisfied by the model, we are able to derive the desired policies. This is achieved by verifying a temporal property for which our desired condition, that the number of infected entities is always less than the threshold, is violated. The counterexample to this property, is in fact the policy we are seeking.

We have used the LTL model checker Spin (Spin (2010)), we (a) find an intervention policy (if one exists) for containing an infection spread; and (b) verify a prevention policy (i.e., a strategy to contain the outbreak of any infection within a specified threshold regardless of the location of the outbreak) where policies are required to contain the spread of the infection to at most l nodes in the network. The model of spread we considered is very general. The details of how the infection spreads between nodes of the network is specified through the infection propagation functions f and g. The functions f and g can be appropriately defined to model the spread of disease in humans, the spread of opinions in a social network, the spread of computer viruses in a computer network, or the spread of a fire in a forest.

Finally, we have introduced a method for subdividing a network for easier management of

the spread of the infection. The central theme of our method is to identify regions around infection outbreaks such that the policymaker has the resources to contain the infection to the regions. We are thus able to use the above mentioned model checking techniques to identify and verify policies in networks with tens of thousands of nodes. Additionally, individually tailored policies may be applied to each region, thus improving the flexibility of our approach.

7.1 Future Work

Our work opens several avenues for further research. We have developed several optimizations such as single-step search to detect non-existence of policies, iterative bounded searches, node merging, and removal of non-infectable nodes. However, further optimizations to the algorithms in this thesis can be developed. Alternate definitions for the construction of regions can be considered. Such constructions could provide the policymaker with further flexibility in responding to an infection outbreak. The policymaker would be able to automate the construction of regions based on criteria such as the sum of the degrees of the nodes in the region. Further optimizations are also possible to the model checking algorithms. For example, it may be possible to further reduce the size of the graph explored by the model checker by collapsing triangles (strongly connected components consisting of three nodes). By reducing the graph size, it will be possible to model check larger graphs, and increase the speed of model checking current graphs. A similar approach is the identification of clusters within the graph. Leveraging the work done in this area could lead to further improvements (Moody (2001); Clauset et al. (2004)). Additionally, other propagation functions beyond the ones addressed in this thesis can be addressed, along with optimizations for them. For example, propagation functions to model the spread of an infection where an infected entity is first a carrier before developing symptoms of the infection.

Due to the large state-space a model checker must explore, there is a limit to the size of models that can be verified. However, it is possible alternatives approaches can be used to avoid the limitations of model checkers. The identification and verification of policies can be done by encoding the original graph as a set of relations in a logical programming language such as Teyjus or XSB (λProlog (2011); XSB (2011)). With correctly formulated queries, logical

programming can perform the same exhaustive searches of the state-space as model checkers. Thus, logical programming languages can be explored as alternatives to model checkers.

Most infections are not spread between individuals at a one hundred percent probability. Thus a deterministic model may not always provide the most accurate model for the spread of infections. Thus, another avenue of research is extending the framework to handle probabilistic infection propagation functions. In such models, infection propagation is expressed as a probability that a node n will become infected in the next time step. Such a model would require a different formulation for the effectiveness of a policy, such as: A policy is effective if at every time step there is a 95% probability that no more than l nodes are infected.

Modifications can also be made to the definition of the graph. In our work, the graph $G(V, E)$ has been held constant, neither V nor E vary with time. An alternative is to introduce variance in the structure of the graph with respect to the flow of time. Computer networks are not static. Computers are added and removed, and the connections between computer are made and lost. To more accurately simulate such a network, the graph G must evolve with time. Similarly, outside of the computing world, people come in contact with others in a specific order. If in the course of my day I go to the store, then the barber (where I am infected), and finally the gym, I cannot spread the infection to the shoppers I interacted with in the store. Thus in the graph representation of this network, the list of my neighboring nodes will vary as a function of time. A model for such interactions requires that edges be added and deleted from the model over the course of the simulation. Our framework is capable of handling such graphs, requiring only changes to the Java preprocessor model generation code to output different node adjacency matrices for each time step.

Finally, we plan to develop a framework that will allow users to perform (semi-)automatic propagation analysis by considering different types of propagation functions and network models (as described above). The primary challenge in developing such a framework is to provide an intuitive user-interface that will allow easy input of propagation functions and network models by the user.

BIBLIOGRAPHY

Anshelevich, E., Chakrabarty, D., Hate, A., and Swamy, C. (2010). Approximability of the firefighter problem. *Algorithmica*, pages 1–17. 10.1007/s00453-010-9469-y.

Arino, J. and van den Driessche, P. (2003). A multi-city epidemic model. *Mathematical Population Studies*, 10(3):175–193.

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

Barrett, C. L., Bisset, K., Chen, J., Eubank, S., Lewis, B., Kumar, V. S. A., Marathe, M. V., and Mortveit, H. S. (2009b). *Interactions among human behavior, social networks, and societal infrastructures: A Case Study in Computational Epidemiology*, pages 477–+.

Barrett, C. L., Eubank, S., and Marathe, M. V. (2008). An interaction-based approach to computational epidemiology. In *AAAI'08*, pages 1590–1593.

Clarke, E., Grumberg, O., and Peled, D. (2000). *Model Checking*. MIT Press.

Clauset, A., Newman, M. E. J., and Moore, C. (2004). Finding community structure in very large networks. *Physical Review E*, 70(6):066111–+.

Cowie, J. (2011). Egypt leaves the internet. http://www.renesys.com/blog/2011/01/egypt-leaves-the-internet.shtml.

Dreyer, P. A. and Roberts, F. S. (2009). Irreversible k-threshold processes: Graph-theoretical threshold models of the spread of disease and of opinion. *Discrete Applied Mathematics*, 157(7):1615–1627.

Falco, M. (2009). Cdc: Production of h1n1 flu vaccine lagging. http://articles.cnn.com/2009-10-16/health/h1n1.vaccine.delay_1_flu-vaccine-h1n1-schuchat?_s=PM:HEALTH.

Finbow, S. and MacGillivray, G. (2009). The firefighter problem: A survey of results, directions and questions. *Australas. J. Combin.*, 43:57–77.

Hethcote, H. W. and Driessche, P. (1995). An sis epidemic model with variable population size and a delay. *Journal of Mathematical Biology*, 34:177–194. 10.1007/BF00178772.

Kermack, W. O. and McKendrick, A. G. (1927). A Contribution to the Mathematical Theory of Epidemics. *Royal Society of London Proceedings Series A*, 115:700–721.

λProlog (2011). λprolog: Programming with higher-order logic. http://www.lix.polytechnique.fr/~dale/lProlog/.

Leskovec, J. (2011). Stanford large network dataset collection. http://snap.stanford.edu/data/index.html.

MacGillivray, G. and Wang, P. (2003). On the firefighter problem. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 47:83–96.

Moody, J. (2001). Peer influence groups: identifying dense clusters in large networks. *Social Networks*, 23(4):261–283.

Naveh, B. (2005). Jgrapht. http://www.jgrapht.org/.

Newman, M. E. J. (2002). Spread of epidemic disease on networks. *Phys. Rev. E*, 66(1):016128.

Ravasz, E. and Barabási, A.-L. (2003). Hierarchical organization in complex networks. *Phys. Rev. E*, 67(2):026112.

Santhanam, G. R., Suvorov, Y., Basu, S., and Honavar, V. (2011). Verifying intervention policies to counter infection propagation over networks: A model checking approach.

Serazzi, G. and Zanero, S. (2004). Computer virus propagation models. In Calzarossa, M. C. and Gelenbe, E., editors, *Performance Tools and Applications to Networked Systems*, volume 2965 of *Lecture Notes in Computer Science*, pages 26–50. Springer Berlin Heidelberg.

Simao, P. (2011). Shadow internet: Secret u.s. effort reportedly aims to help dissidents. http://www.huffingtonpost.com/2011/06/12/shadow-internet-secret-us_n_875577.html.

Spin (2010). On-the-fly, ltl model checking with spin. http://spinroot.com/spin/whatispin.html.

Spinroot (2011). Spin memory usage issue. http://spinroot.com/fluxbb/viewtopic.php?id=209.

UPI (2009). Virus strikes 15 million pcs. http://www.upi.com/Top_News/2009/01/25/Virus_strikes_15_million_PCs/UPI-19421232924206.

Vardi, M. Y. (1996). An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata, volume 1043 of Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag.

Wang, P. and Moeller, S. A. (2002). Fire control on graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 41:19–34.

WHO (2007). The world health report 2007 - a safer future: global public health security in the 21st century.

Willsher, K. (2009). French fighter planes grounded by computer virus. http://www.telegraph.co.uk/news/worldnews/europe/france/4547649/French-fighter-planes-grounded-by-computer-virus.html.

XSB (2011). Welcome to the home of xsb! http://xsb.sourceforge.net/index.html.

Zanette, D. H. (2002). Dynamics of rumor propagation on small-world networks. *Phys. Rev. E*, 65(4):041908.