Verifying Intervention Policies to Counter Infection Propagation over Networks: A Model Checking Approach

Ganesh Ram Santhanam, Yuly Suvorov, Samik Basu and Vasant Honavar
Department of Computer Science, Iowa State University, Ames, IA 50011, USA.
{gsanthan,ysuvorov,sbasu,honavar}@cs.iastate.edu

Abstract

Spread of infections (diseases, ideas, etc.) in a network can be modeled as the evolution of states of nodes in a graph as a function of the states of their neighbors. Given an initial configuration of a network in which a subset of the nodes have been infected, and an infection propagation function that specifies how the states of the nodes evolve over time, we show how to use model checking to identify, verify, and evaluate the effectiveness of intervention policies for containing the propagation of infection over such networks.

%vspace-0.1in

Introduction

The spread of infections such as diseases, rumors, computer viruses and fire across networks of people, computers and forests poses severe challenges to the sustainability of economic, societal and ecological systems. Designing effective intervention policies to control the spread of infection, e.g., administering vaccines, installing security patches, deploying firefighters, etc. is an important problem in computational sustainability. Of particular interest are practical algorithms and tools for finding and evaluating effective policies.

The dynamics of the spread of infections in networks can be modeled in terms of the evolution of the states of nodes in graphs. Previous work in this area (Dreyer and Roberts 2009; Finbow and MacGillivray 2009; Anshelevich et al. 2010) has examined the existence of an effective policy that prevents the dynamics of such graphs from violating some desired property. The problem of finding an effective policy to control the infection spread in a network is NP-hard even for graphs with a maximum degree three (Dreyer and Roberts 2009; Finbow and MacGillivray 2009; Anshelevich et al. 2010). Against this background, practical solutions to this problem are of significant interest.

In this paper, we present a novel solution to the problem of finding and verifying policies to control the spread of infection using the state-of-the-art *model checking* techniques (Clarke, Grumberg, and Peled 2000). We consider a simple model of spread where the state of a node in the network at each time step is a function of (a) the states of other nodes in the network; and (b) the history of previous states of the node. We encode the evolution of states of the nodes in the network succinctly using *Kripke structures*. We consider the problems of finding *intervention policies* that contain the infection in and verifying *preventive policies* that protect a subset of the nodes in the network from infection. We consider a policy to be effective if the number of nodes infected at any time is at most l. We use satisfiability of temporal formulas in the model to answer the following queries: (a) Given a network with a set of initially infected nodes, is there an effective intervention policy? (b) Given a network with a set of initially protected nodes, is the given preventive policy effective? We demonstrate how to find an intervention policy whenever such a policy exists. We also show how to exploit the ability of the model checker to find a counter example (if one exists) to a desired property of the model, to identify conditions that make a preventive policy ineffective. Our approach takes advantage of specialized algorithms used by model checkers to verify the desired reachability properties with respect to a Kripke model. We present preliminary results of experiments that demonstrate the feasibility of this approach to verifying and identifying effective intervention policies. Our approach can be used to counter the spread of diseases (WHO 2007), fire (G. MacGillivray 2003), opinion (Zanette 2002), and computer virus (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).

Models of Spread

Let $G(V, E)$ be an undirected graph, whose nodes V represent the entities (people, computers) that can potentially be infected. 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); and a set of neighbors $\rho(v_i) = \{v_j \in V : (v_i, v_j) \in E\}$ in the graph. The edges E represent the medium over which infection can be potentially transmitted from one node to another. We call the tuple of states of the nodes in the graph at any time step as the *configuration* of the graph at that time step. The set of all possible configurations of the graph is Σ^n. At each discrete time step t, each node v_i in the graph changes its state as a function of (a) the current and previous states of v_i and (b) the current states of the rest of the nodes in V; Let f and g respectively represent the *transmission* and *local update* functions (collectively called

the *infection propagation* functions), that compute the next state of a node v_i as described by (a) and (b) above respectively. The new updated state of v_i in the next time step is a composition of f and g applied on $\sigma(v_i)$. Thus, the configuration of the graph evolves with every discrete time step.

Remarks

1. The model of spread over a graph $G(V, E)$ is quite generic. Different instantiations of the Σ, f and g can be used to model the spread of an epidemic in a population, the spread of opinion in a social network, the spread of fire in a forest, or the spread of a virus over a computer network.
2. The function f determines the state of a node with respect to the states of the other nodes the graph; while the function g accounts for temporal aspects in the behavior of a node based on the history of its states.
3. If in a specific application, the states of the nodes in the graph do not depend on (a) or (b) in the model of spread introduced above, the respective functions f or g become identity functions.
4. Some application scenarios may may require f and g to be composed in a specified order that needs to be respected in updating the states of the nodes in G.

In this paper, we focus on a model of infection spread over a graph $G(V, E)$ where $\Sigma = \{\texttt{open}, \texttt{infected}, \texttt{protected}\}$, i.e., each node can be in one of three states: (a) `open` – the node is vulnerable to infection, (b) `infected` – the node is infected, or (c) `protected` – the node can never be infected. We denote the state of a node v_i in the graph by $\sigma(v_i)$. We focus on the *reversible* and *irreversible* k*-threshold* processes described in (Dreyer and Roberts 2009) in this paper.

1. *Irreversible* k*-threshold process*: In this model of infection spread, an open node can become infected in the time step after at least k of its neighbors get infected, and the node remains at the infected state throughout. The local update function g is simply the identity function, and

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

2. r*-Reversible* k*-threshold process*: Here, an open node can become infected as soon as at least k of its neighbors are infected, and the node returns to the open state r time steps after getting infected. The infection propagation functions f and g are given as follows.

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

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

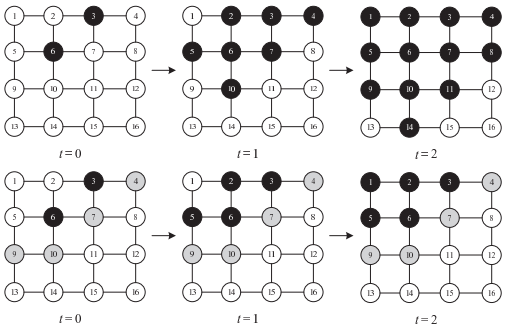

Figure 1: Infection Spread in a 4×4 Grid

Here, the set of states is expanded as $\Sigma = \{\texttt{open}, \texttt{protected}, \texttt{infected}_1 \dots \texttt{infected}_r\}$ to track the additional intermediate states of the nodes.

One way to control the spread of infection in the graph is by protecting one or more of the nodes from getting infected, e.g., by vaccinating them, which in turn prevents them from infecting some of the other open nodes.

Definition 1 (Policy). *A policy is a mapping between the configurations of a graph to sets of nodes in the graph that are to be protected.*

We consider two types of policies: (1) *prevention policies* that are deployed *before* any of the nodes in the graph are infected; and (2) *intervention policies* that are deployed *after* some of the nodes in the graph are infected. In both cases, the objective is to restrict the spread of infection to at most l nodes in the graph. Such policies are of particular interest to policymakers and public health officials who are often tasked with finding effective prevention or intervention policies in preparation or response to disease outbreak. The example below illustrates two intervention policies that prevent nodes in certain parts of the graph from getting infected.

Example 1. *Consider a* 4×4 *grid network shown in Figure 1. Suppose the nodes* 3 *and* 6 *are infected (colored black) at time* $t = 0$*. Assume that the local update function* g *is the identity function, i.e.,* $g(\sigma(v_i)) = \sigma(v_i)$ *and the transmission function* f *models the irreversible* 1*-threshold process, i.e., a node becomes infected at time step* $T + 1$ *if any of its neighbors becomes infected at time* t*.*

The figure shows the spread of infection, i.e., the evolution of states in the graph over three time steps. The use of policy π_1 *of vaccinating the nodes* $4, 7, 9$ *and* 10 *at time* $t = 0$*, the infection is contained to nodes* $1, 2$ *and* 5*, thus protecting nodes* $8, 11 - 16$*. In contrast, the use of policy* π_2 *of vaccinating the nodes* $4, 5, 7$ *and* 10 *at time* $t = 0$*, contains the infection to nodes* 1 *and* 2*, thereby protecting the node* 5 *in addition to the nodes protected by* π_1*.*

Example 2. *Consider a ring network shown in Figure 2. Suppose that at* $t = 0$*, nodes* 1 *and* 4 *are infected with a virus that subsequently spreads according to the* 1*-reversible* 1*-threshold process. In this case, each node has exactly 2 neighbors. An open node becomes infected at* $t+1$ *if at least one of its neighbors is infected at* t*. Suppose further that a node that becomes infected at time* t *becomes open again*

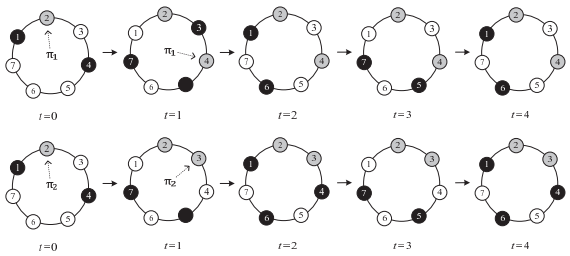

Figure 2: Infection Spread in a Ring

at time step $t+1$, i.e., the infection lasts only for one time step after which the infected node returns to its uninfected (normal) state. The infection propagation functions f and g corresponding to this model of spread 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 we have a drug that can cure a node if it is infected; and further prevent the node from contracting the infection in the future; and two such doses are available to be administered, one per time step. Figure 2 shows the evolution of the graph over three time steps according to two policies π_1 and π_2. Policy π_1 administers the drug to node 2 at $t = 0$ and to node 4 at $t = 1$, while π_2 administers the drug to node 2 at $t = 0$ and to node 3 at $t = 1$. Note that the policy π_1 controls the spread such that eventually (i.e., at time $t \geq 3$), the maximum number of infected nodes in the graph is always ≤ 2. On the other hand, when policy π_2 is applied, the maximum number of infected nodes in the graph can exceed 2 at times (e.g., at $t = 4$).

In the preceding examples, the identity of nodes that are infected at time $t = 0$ are assumed to be known. However, in many settings, one might have only an estimate of the total number of infected nodes , or the number of doses of vaccine or drug available to be administered may be variable. Thus, a policymaker must consider every possible scenario for a given set of initially infected and protected nodes in order to arrive at an effective intervention strategy. Moreover, some of the available intervention policies may be more effective than others in controlling the spread (as measured by the number of nodes in the graph that are protected from infection). Other measures of the effectiveness of policies in controlling the spread may include the number of doses of the vaccine or drug needed, the number of nodes saved per dose administered at the beginning of the spread and the maximum number of newly infected nodes at each time step. Hence, there is a need for effective computational tools for assisting policymakers to identify effective policies in specific scenarios.

Given an arbitrary graph $G(V, E)$ and the infection propagation functions f and g that specify the evolution of the states of the nodes in G, we are interested in answering questions such as the following:

Q1. *Finding an intervention policy*: Given a configuration of the graph where a set $I \subseteq V$ of nodes are infected, and a fixed number m of nodes that can be protected at the outset, is there an intervention policy π that contains the infection to at most 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 of nodes that may be infected at the outset, does the policy π prevent the infection from spreading to more than l nodes?

We next proceed to show how questions such as the above can be answered by modeling the spread of infection over an arbitrary graph using Kripke structures and verifying the satisfiability of appropriate temporal logic properties using model checking techniques.

Analyzing Policies using Model Checking

Each of the above questions can 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 use some of the state-of-the-art approaches to model checking to verify the reachability of one configuration of the graph to another. We first encode the transitions between the configurations of the original graph over which the infection is spreading as an input graph to a model checker (we use Spin (Spin 2010)). We reduce the problem of checking the existence of, or verifying policies to the problem of testing whether a specified reachability condition holds in this graph.

Given the initial configuration of a graph over which infection is spreading, we construct a model in a language that is accepted by the model checker as follows. The nodes of the graph over which the infection is spreading are mapped to the state variables of the model in a model checker. Thus, a configuration of the original graph corresponds to a state of the model input to the model checker. The allowed transitions between the various configurations of the graph as dictated by the infection propagation functions are then directly encoded as transitions in the model. This ensures that the model checker explores all possible evolutions of the graph with respect to initial configurations, which corresponds to simulating all the possible ways in which the infection can spread over the nodes in the graph.

Queries regarding the identification and verification of policies are then formulated as linear temporal logic properties (Vardi 1996) over the state space of the model. We thus leverage the optimized algorithms for LTL model checking used by the model checker to efficiently verify the satisfiability of the corresponding properties.

Encoding Spread using Kripke Structures

We use a *Kripke structure* (Clarke, Grumberg, and Peled 2000) to model the transitions between the configurations.

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* $L : S \rightarrow 2^P$ *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 propagation functions f, g, a Kripke structure K_G that captures all the possible transitions between the configurations of the graph (with respect to f) 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. 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 $s \rightarrow s'$) 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 type Q1 is to be answered, then each initially infected node $v_i \in I$ is set to the state $\sigma(v_i) =$ `infected` and in the case of Q2, each initially protected node $v_i \in P$ is set to the state $\sigma(v_i) =$ `protected`.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.

In the above encoding of the Kripke structure, the transition rules ensure that the infection spreads exactly as specified by the infection propagation functions, i.e., the transitions in K_G correspond to the change of configurations in the graph G in a single time step. The specification of initial states S_0 identifies the set of nodes that are infected and/or the set of nodes that are protected. The model checker considers every possible instantiation of the Kripke structure corresponding to the set of initial states S_0 in order to verify a desired property. Note for the spread problems considered in this paper, given an initial configuration of the graph (a state in S_0), there is exactly one possible evolution of the graph (a path in the corresponding Kripke structure).

The Kripke structure corresponding to the graph in Example 1 is given by $K_G = \langle S, S_0, T, L \rangle$ as follows. Each state is a tuple $s = \langle \sigma(v_1) \ldots \sigma(v_n) \rangle \in S$ where $\sigma(v_i) \in \Sigma$ represents the state of the node v_i in the graph. Note that if at least one of the neighbors of v_i is currently infected, then the state of v_i **must** be infected in the target state of any transition from s in K_G, as dictated by the functions f and g (see Example 1). Hence, transitions in the Kripke structure correspond to changes in the configurations of the graph, e.g., the transition $0010010000000000 \rightarrow 01111110010000000$ (where 0 and 1 represent the states `open` and `infected` respectively), corresponds to the change of configuration from $t = 0$ to $t = 1$ in the graph when there are no protected nodes to start with.

The above Kripke structure can be encoded in the language of the model checker Spin (Spin 2010).We now show how the model checker can be used to find and verify intervention policies for containing the spread of infections.

Finding & Verifying Policies using LTL

Given a Kripke structure K_G that encodes the spread of infection in a graph $G(V, E)$, finding and verifying policies can be reduced to verifying corresponding temporal properties in LTL (linear temporal logic, see (Vardi 1996)). The syntax of LTL is defined over a set of propositions `Prop` 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 is defined in terms of the set of paths that satisfy the 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 $\delta[i]$ ($\delta^0 = \delta$). Any path satisfies `true`, a path δ satisfies `Prop` iff $\delta[0]$ satisfies `Prop`; a path satisfies $\neg\varphi$ iff it does not satisfy φ; and a path satisfies $\varphi_1 \vee \varphi_2$ iff it satisfies either of the disjuncts. A path δ satisfies $\mathbf{X}\varphi$ (*next* φ) iff δ^1 satisfies φ and a path δ satisfies $\varphi_1 \mathbf{U} \varphi_2$ (φ_1 *until* φ_2) iff there exists $j \geq 0 : \delta^j$ satisfies φ_2 and for all $i < j : \delta^i$ satisfies φ_1. Other temporal operators in LTL are described in terms of the above. In particular, $\mathbf{F}\varphi \equiv \texttt{true}\mathbf{U}\varphi$ (a path δ satisfies the property *eventually* iff $\exists j \geq 0 : \delta^j$ satisfies φ); $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$ (a path δ satisfies the property *globally* iff $\forall i \geq 0 : \delta^i$ satisfies φ). A Kripke structure satisfies an LTL formula φ iff all paths starting from all its start states satisfy φ.

We consider the LTL properties that correspond to queries of the type Q1 or Q2 shown below. For each of the query types, the initial states, i.e., the set of `infected` or `protected` nodes in the graph, are pre-specified. For example, in a query of type Q1, the set I gives the nodes that are initially `infected` which correspond to the set of start states S_0 in the Kripke structure K_G. This guides the model checker to only explore those paths in the Kripke structure that start from one of the states in S_0. Other queries may specify instead, the maximum number of infected or protected nodes in the graph. This implies that different combinations of nodes in the graph can be infected or protected at the start of the spread. We encode this information by allowing the model checker to non-deterministically initialize the states of the nodes in the graph while ensuring that the constraints on the number of infected or protected nodes are respected.

Q1: Finding an intervention policy In this case, the set $I \subseteq V$ of initially infected nodes, and a fixed number m of nodes that can be initially protected are pre-specified. Hence, we encode the set of start states as follows.

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

The query condition specifies a *number* m of nodes that are initially protected, rather than the precise nodes

that are protected. Thus requires considering different combinations of m nodes in the graph that could be initialized to `protected` state. The term $|\{v_i \mid \sigma(v_i) = \texttt{protected}\}| = m$ in the above definition models this condition, and allows the model checker to non-deterministically initialize the start states accordingly.

Let $total_I$ be the total number of infected nodes at any given time step. In order to find a policy π such that at most l nodes are infected, we use the LTL formula $\varphi : \mathbf{F}(total_I > l)$. The Kripke structure satisfies the LTL formula φ if and only if in every path beginning from all the possible initial states (i.e., all states in S_0), there exists a state where $total_I > l$ holds. Note that the satisfaction of this LTL formula implies that no policy exists, i.e., any combination of m nodes that are protected at the start of the spread will lead to at least $l + 1$ nodes being infected over time. On the other hand, if the LTL formula is not satisfied, it implies the existence of an initial configuration (start state in S_0) such that $total_I \leq l$ holds in each of the states along a path starting from the initial configuration (recall that there is exactly one path from each start state). In this case, the model checker returns a counter-example specifying an initial state $s = \langle \sigma(v_1) \dots \sigma(v_n) \rangle$ and a path starting from this initial state in which every state satisfies $total_I \leq l$. The desired policy π can be constructed from the counter-example returned by the model checker by assigning to the node v_i the state `protected`, if $\sigma(v_i) = \texttt{protected}$ in s.

The formula φ can be used to find policies for controlling the spread of infection when the functions f and g are monotonic, i.e., a node can never become open after it has been infected. This means that $total_I$ never decreases (see Example 1). However, in the case of r-Reversible k-threshold process a node changes state to open after it has been infected for r time steps, i.e., the transmission function g is not monotonic. Consequently, $total_I$ may increase or decrease at each time step, as seen in Figure 2 (see policy π_2 Example 2) where $total_I$ changes from 2 to 3 to 2 at $t = 1, 2, 3$.

In this case, instead of finding a policy π such that at most l nodes are infected at any time step, the policymaker might seek to stabilize the the total number of infected nodes, i.e., "Is there a policy π such that the total number of infected nodes is at most l always, *after a certain number of time steps*?" In this case, we can use the LTL formula $\varphi' : \mathbf{GF}(total_I > l)$. The Kripke structure satisfies the LTL formula φ' if and only if in every path beginning from each possible initial state (i.e., all states in S_0), $total_I > l$ holds infinitely often. As in the previous query, the satisfaction of this LTL formula implies that no policy such exists; and its non-satisfaction implies that the desired policy exists. Specifically, non-satisfiability of φ' implies the existence of a start state such that the path from that state satisfies $\mathbf{FG}(total_I \leq l)$ (recall that there is exactly one path from each start state). The policy can be obtained by from the counter example returned by the model checker as in the case of the irreversible k-threshold process.

Q2: Verifying a preventive policy In this query, a policy π specifying a set $P \subseteq V$ of initially protected nodes, and a fixed number m of nodes that may be initially infected are pre-specified. The set of start states is encoded similar to the case of Q1 as follows.

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

As in Q1, the condition $|\{v_i \mid \sigma(v_i) = \texttt{protected}\}| = m$ allows the model checker to non-deterministically explore all possible combinations of m nodes to be initially infected. Thus the policy is modeled along with all possible scenarios of initial infections.

Verification of a given policy π can be achieved using the LTL formula $\psi : \mathbf{G}(total_I \leq l)$. ψ is satisfied if and only if in every path beginning from all the possible initial states (i.e., all states in S_0), $total_I \leq l$ always holds. This implies that protecting the nodes initially according to the given policy can successfully control the infection from spreading to more than l nodes, regardless of which combination of m nodes are infected at the start of the spread. On the other hand, if ψ is not satisfied, then the model checker returns a counter example specifying a path starting from a state $s \in S_0$ in the Kripke structure where a combination of m nodes are initially infected, for which the policy fails, i.e., the infection spreads to at least $l + 1$ nodes. The counter example specifies a scenario of initial infection outbreak in the graph, which may be used by the policymakers to change the policy.

As in the case of Q1, queries of type Q2 posed against models where the transmission functions are non-monotonic (e.g., for spread using r-Reversible k-threshold processes), the policymaker may look for verifying if the given policy will satisfy the stability condition that' the total number of infected nodes is at most l always, *after a certain number of time steps*. The policy can be verified against this new criteria using the formula $\psi' : \mathbf{FG}(total_I \leq l)$. If ψ' is satisfied, it means that the policy is successful in containing the infection to at most l nodes eventually, after a certain number of time steps. On the other hand, if ψ' is not satisfied, the model checker provides a counter-example specifying an initial configuration (start state in S_0) having m infected nodes resulting in a a path in which the total number of infected nodes is $> l$ infinitely often. The counter example may be used by the policymakers to examine the cause of the policy's failure and develop alternative policies.

Implementation and Experiments

We now proceed to describe results of preliminary experiments that demonstrate the feasibility of the proposed model checking approach to finding intervention policies and verifying prevention policies. We have implemented a preprocessor (in Java) that accepts as input, the network, the initial configuration, and optionally the policy to be verified, and provides a Kripke model encoding of the problem in Promela, the input language for the Spin model checker (Spin 2010). The preprocessor generates the input model such that the model checker only explores states where the query condition $total_I \leq l$ is satisfied; this is sufficient for the model checker to produce correct solutions for the model checking problem because reachability of a state where the query condition is not satisfied implies the non-existence of intervention policy or unsatisfiability of prevention policy along that path.

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 1: Results for random networks with 40 nodes.

Table 1[1] shows the results of our implementation for networks having 40 nodes and $40, 50, 60, 70$ and 80 edges (E) randomly generated such that the maximum degree of each node is ≤ 5. We tested each network with query Q1 $\mathbf{F}(total_I > l)$ with $l = 10$ and $l = 20$. The results show that the model checker is able to identify intervention policies, when one exists, within a minute for many of the test cases. The number of states explored by the model checker increases with increase in the number of edges (for a fixed value of l). This is because increasing number of edges leads to increase in the number of possible ways the infection can spread in the network. On the other hand, the number of states explored by the model checker decreases with increase in l (for a fixed value of edges). This is because increasing l increases the number of intervention policies that are effective and the model checker terminates exploration of the model as soon as it identifies one such policy. We observe similar trends with respect to memory usage.

Optimizations to Improve Scalability

We developed several optimizations that can be incorporated into the preprocessing step. These optimizations are specifically designed to allow faster search for intervention policies in settings where the spread of infection is irreversible.

Iterative Bounded Search: The main idea here is to iteratively search for an intervention policy, each time choosing the nodes to protect in a specific order. In iteration i, we consider all the non-infected nodes that are $\leq i$ edges away from the infected nodes as *candidates* to be protected (rather than considering *all* non-infected nodes). This strategy yields a trivial intervention policy if we are able to protect all the candidate nodes, i.e., when number of candidates $\leq$ the number of nodes that can be protected at the outset (Protecting each of the neighbors of each infected node ensures that the infection cannot spread to any as yet uninfected nodes). The iteration is continued (for $i = 1, 2, \ldots$) until an intervention policy is obtained or the i is larger than the maximum distance between the nodes in the network being considered (i.e., non existence of intervention policy is proved). This strategy guides model exploration in a manner that ensures that an intervention policy (if one exists) is obtained (faster) with minimal exploration of the parts of the model state-space that do not contribute to finding an intervention policy.

Single-Step Search to Detect Non-existence of Policy: If the number of nodes that can get infected in one step exceeds the number of nodes that can be protected at the outset by more than the specified threshold for the intervention policy, then we can infer the non existence of an intervention policy (without having to deploy the model checker).

Node Merging: This optimization is based on a simple observation that when two adjacent nodes n_1 and n_2 in G are are uninfected and unprotected, and one of the nodes (say n_2) has no neighbors other than n_1, then n_2 can be merged with n_1 without affecting the answer to the query. When i nodes adjacent to n_1 are thus merged with n_1, we annotate the node n_1 with i to denote the fact that if n_1 gets infected at (discrete) time step t, then each of the i nodes adjacent to n_1 in G are infected at at time step $t+1$. While this optimization reduces the number of states in the model, solutions obtained for the optimized model remain valid for the original model. Note that this strategy cannot be applied in settings where nodes can be protected after the spread of infection is already underway.

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 1. The new results (in terms of size of the state-space, time and memory usage) are presented in parenthesis along with the results obtained without the optimizations. The optimizations resulted in about 50% reduction in the number of states explored by the model checker, the time taken, and the amount of memory used. The optimizations made it possible to find intervention policies for random networks with 40 to 100 nodes, which was not possible without the optimizations.

It should be noted that real world networks (e.g., social networks, the Internet, the power grid), tend to exhibit scale-free topologies and and hierarchical modularity (Ravasz and Barabási 2003). Due to the existence of a few *hub* nodes that have a relatively high degree of connectivity, the degree distribution of the nodes in such networks generally follow a power law distribution (Ravasz and Barabási 2003).

We have conducted 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, 20 protected nodes; and 20 infected, 10 protected nodes), and threshold values of 30 and 40. The corresponding results are presented in Table 2.

As compared to random networks of similar size (results not shown), the performance of our approach (as measured by the size of the state space, time, and memory used) is at least an order of magnitude (and sometimes two orders of magnitude) better in the case of scale-free networks. We note that the space and time usage is extremely low for some of the generated problem instances (e.g., see Table 2, 100 node networks with 20 initially infected, 10 protected nodes and a threshold of 30). This suggests that in the case of these instances the non-existence of a policy was determined in the

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

V	E	States ($\times 10^6$)		Time (secs.)		Memory (GB)	
		30	40	30	40	30	40
10 Infected nodes, 20 protected nodes							
80	119.53	0.07	10^{-4}	0.18	10^{-3}	10^{-2}	10^{-3}
90	138.00	3.46	6.76	0.83	27.98	10^{-2}	0.63
100	149.43	5.46	10^{-3}	29.00	10^{-3}	0.51	10^{-3}
20 Infected nodes, 10 protected nodes							
80	120.97	12.29	29.33	26.59	112.66	1.01	2.46
90	133.67	6.14	30.04	13.61	149.10	0.51	2.53
100	151.53	10^{-6}	27.99	0.00	146.67	10^{-3}	2.53

Table 2: Results for scale-free networks.

preprocessor stage, without the need for invoking the model checker to explore to the state space.

The scalability of our approach can also be further improved by taking advantage of the advances in model checking such as abstraction, symmetry reduction and bounded model checking (Clarke 2008; Biere et al. 1999; Cook and Sharygina 2005). Alternative techniques based on SAT solvers (Biere et al. 2006) would be interesting to explore and compare with our approach.

Conclusion

We have presented a practical solution to the problem of finding and verifying policies for controlling the spread of infections (diseases, ideas, etc.) in networks. Our approach encodes the spread of infection in a network using a Kripke structure, where each change in the configuration (the set of infected or protected nodes in the network) corresponds to a transition in the Kripke structure. This allows us to reduce the problem of verifying the effectiveness of prevention policies to the problem of model checking temporal properties of the Kripke structure. Furthermore, we we can take advantage of the ability of model checkers to identify counter-examples that demonstrate that a given temporal property is not satisfied by the model to derive the desired policies from the counter-example by verifying temporal properties that violate the conditions that need to be satisfied by the policies.

Using the Spin model checker, we have used LTL model checking techniques to (a) find an intervention policy (if one exists) for containing an infection break; and (b) verify a prevention policy (i.e., a strategy to protect a subset of the nodes before the onset of an outbreak) where the policies are required to control the spread to at most l nodes in the network. The model of spread we considered is quite general: The details of how the nodes in the network respond to the spread of infection are specified by the infection propagation functions f and g. Hence, the techniques introduced here are equally applicable to settings that involve the spread of infections in a population, the spread of viruses or malware in a computer network, or the spread of opinions or rumors in a social network. As part of future work, we plan to investigate strategies to model and analyze spread problems with probabilistic and real time constraints (Newman 2003).

Acknowledgments

This work is supported in parts by NSF grant CCF0702758. The work of Vasant Honavar was supported by the National Science Foundation, while working at the Foundation. Any opinion, finding, and conclusions contained in this article are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

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

Biere, A.; Cimatti, A.; Clarke, E. M.; and Zhu, Y. 1999. Symbolic model checking without bdds. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS 1579*, 193–207. Springer.

Biere, A.; Heljanko, K.; Junttila, T. A.; Latvala, T.; and Schuppan, V. 2006. Linear encodings of bounded ltl model checking. *CoRR* abs/cs/0611029.

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

Clarke, E. 2008. The birth of model checking.

Cook, B., and Sharygina, N. 2005. Symbolic model checking for asynchronous boolean programs. In *SPIN*, 75–90.

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.

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

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

Newman, M. E. J. 2003. The structure and function of complex networks. *SIAM REVIEW* 45:167–256.

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

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

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

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*, 238–266. Springer-Verlag.

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

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