The Pennsylvania State University

Graduate School

College of Information Sciences and Technology

A MODEL CHECKING APPROACH TO COUNTERING THE DYNAMICS OF INFECTION PROPAGATION OVER NETWORK

A Thesis in

Information Sciences and Technology

by

Can Zhang

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

May 2016

ABSTRACT

With the outbreak of Ebola over the past year, attention has been paid on predicting and resolving the propagation of infectious disease over network of people and animals. Model checking is a commonly used method in the field of software analysis and verification. In this thesis, we propose to use model checking to counteract the spread of foot-and-mouth disease (FMD) in networks. We abstract the FMD spread model and properties, and encode the system using a well-known model checker Spin. Our program is capable of finding intervention policies and evaluating the effectiveness of different policies. Moreover, previous works generally use simulation models to study the disease control problem which cannot provide certainty as to predict whether certain future states of the outbreak are possible under a particular control policy. Model checking, on the other hand, is guaranteed to find a path that leads to the future states as long as they are possible from a given current configuration of the contagion network under a given control policy. It is worth mentioning that the method proposed in this thesis is not limited to infectious diseases, but can also be applied to counter the spread of, for example, computer virus, forest fire, and public opinions.

TABLE OF CONTENTS

LIST OF FIGURES v

LIST OF TABLES vi

ACKNOWLEDGEMENTS vii

Chapter 1 Introduction 1

- Model Checking 2
- Linear Temporal Logic 4
- SMV and Spin 5
- Thesis Organization 5

Chapter 2 Background and Related Work 7

- FMD and Related Research 7
- Model Checking and Related Research 8

Chapter 3 Approaches and Methods 10

- Farm Network and Propagation Function 10
- Identifying a Control Policy 12
- Effectiveness Evaluation of Control Policy 13
- Checking Possible Future State 14

Chapter 4 Experiments and Results 15

- Experiment Setup 15
- Checking Possible Future States 19
- Evaluating Effectiveness of Policies. 29

Chapter 5 Summary and Conclusion 34

- Summary 34
- Limitations 34
- Future Work. 35

References 37

LIST OF FIGURES

Figure **4-1**. Simulation and verification mode algorithms of Spin.......................................18

Figure **4-2**. Days used to stop infections on different policies..29

Figure **4-3**. Number of dead nodes on different policies. 30

Figure **4-4**. Doses of vaccination used on different policies. 30

Figure **4-5**. Days used to stop infections on different latent period. 31

Figure **4-6**. Number of dead nodes on different latent period. 32

Figure **4-7**. Doses of vaccination used on different latent period. 32

LIST OF TABLES

Table **4-1**. The number of nodes in different states at each day for the counter example of property p1. 20

Table **4-2**. The number of nodes in different states at each day for the counter example of property p2. 21

Table **4-3**. The number of nodes in different states at each day for the counter example of property p3. 23

Table **4-4**. The number of nodes in different states at each day for the counter example of property p4. 24

Table **4-5**. The number of nodes in different states at each day for the counter example of property p5. 25

Table **4-6**. The results of finding the lower limit of policies that can guarantee stop the infection for both models. 26

Table **4-7**. Performance summaries of the experiments on both theoretical model and real data model... 28

Table **4-8**. Summary of comparisons between the model checking and simulation approach... 28

ACKNOWLEDGEMENTS

I am grateful to my advisor, Dr. Dinghao Wu, for his generous guidance, support, and encouragement throughout the time of my Master's study, without which this project would be impossible for me. Dr. Wu not only guided me on my research and study, but also been helpful in my life at Penn State. It was a great pleasure working with him.

I would also like to thank my committee members, Dr. John Yen and Dr. Matthew Ferrari, for serving as my committee and providing me valuable suggestions and comments on my research project. I want to thank my collaborator Dr. Will Probert for his kindly help and collaboration on this project. I would also like to thank Center for Infectious Disease Dynamics (CIDD) at Penn State for providing the foot-and-mouth disease data for this thesis.

Finally I want to thank my lab mates and my friends in IST and Penn State for making my life more enjoyable.

Chapter 1

Introduction

Foot-and-mouth disease (FMD) is an infectious viral disease that affects cloven-hoofed animals including sheep, pig, and cattle. The virus causes vesicles around hooves and mouths, high fever, weight loss, and is sometimes fatal to the infected animals. Given the hardiness of virus survival and transmissibility of FMD, which may lead to severe consequences including economic loss and trade restrictions posed to the society, it is of great importance to find effective policies to control FMD. However, since FMD is a foreign animal disease and field experiments are difficult or prohibited, mathematical and computation modeling methods have been proven to be successful ways to help control FMD.

The dynamics of the infectious disease spread can be viewed and modeled as the evolution of the states of nodes in trees, graphs, and other data structures. When given an initial configuration of a network and a propagation function defining how infectious nodes evolve over time, the model checking method can be used to find whether an intervention policy exists. We can also evaluate the effectiveness of different prevention policies. More importantly, by using model checking, we can check whether a future configuration (scenario) is possible or not. The result would provide direct insights to many important questions.

In this thesis, we propose to use model checking to resolve the problem of counteracting infection spread in networks, specifically for the control of foot-and-mouth disease (FMD). We address the following questions in this thesis. First, we construct a network and study the propagation function over the network. Second, we use model checking techniques to search for an effective intervention policy based on the initial configuration of the FMD model and propagation function. Third, we verify the effectiveness of different intervention policies. Finally, we answer some proposed questions by using model checking to examine whether a future configuration (scenario) is possible. We also compare our model checking approach with simulation on their performances. It is worth mentioning that our approach of identifying and verifying effective policies using model checking is not limited to counter-act disease across a network of animals. It can also be applied to, for example, virus across a network of computers (Serazzi and Zanero, 2004), fire across a network of forests (Finbow and MacGillivray, 2009), and rumors across a network of social media (Zanette, 2002).

1.1 Model Checking

It is of great importance to verify the correctness of computer systems. Formal verification is widely applied in both hardware and software design and system checking. Particularly it plays a major role in safety critical systems. Formal verification usually consists of three essential parts.

(1) A framework or model for the system: use a description language to model the system.

(2) A specification language: use a specification language to define the properties that will be checked for the system.

(3) A verification method: use a verification method to determine whether the model meets the specifications.

As one of the common formal verification methods, model checking is an automated property verification method originally developed to deal with the bugs in concurrent systems. Concurrency bugs are usually difficult to be found by testing due to their non-reproducible characteristics and the fact that they are not covered by test cases. Model checking plays an important role in finding bugs in the concurrent systems (Huth and Ryan, 2004).

Model checking uses temporal logic, which is propositional and predicated logic. A model based on temporal logic has several states and can be true for some states and false for other states. For those dynamic formulas, they can change their values with the evolution of the system from state to state. For traditional logic, the value may never change when the value of all its boolean variables are fixed such as $p = A \wedge \neg B \vee C$. Instead, temporal logic uses the concept of time, which is represented as the transition between different states. In this sense, the value of a formula is not fixed because of its dynamic nature (Huth and Ryan, 2004).

In model checking, we use M to represent the transition system, and ϕ to represent the properties of the system. To verify whether the property meets the model, model checking process usually consists of three general steps.

(1) Use a description language to model the system as model M.

(2) Use a specification language to encode the property ϕ.

(3) Run the model checker to see whether the property holds for the model M.

Generally speaking, the ultimate goal of model checking is to check the following entailment.

$$M, s \models \phi$$

Here *M* is the model that model checking works on, *s* is the starting state of the model and ϕ is the specifications to be checked. The output answer of running the model checker is either success, which means the property holds for the model *M*, or “error”, which means the property does not hold for the model *M*. For most model checkers, if the answer is “error”, they will also produce a counter example with traces that lead to the failure. This function is very useful in the design and debugging of the system. For our project, we can take advantage of this feature to help us find a control policy that can stop the propagation of the infectious disease (Huth and Ryan, 2004).

1.2 Linear Temporal Logic

Linear time temporal logic (also known as LTL) is a temporal logic that treats the time as an infinite sequence of states. If we consider a sequence of states as a path, we can encode all future paths using LTL. The formal description of LTL is as follows (Huth and Ryan, 2004).

$$\phi = \mathrm{T} \mid \mathrm{F} \mid p \mid (\neg\phi_1) \mid (\phi_1 \wedge \phi_2) \mid (\phi_1 \vee \phi_2) \mid (\phi_1 \rightarrow \phi_2) \mid (\mathrm{X}\phi_1) \mid (\mathrm{F}\phi_1) \mid (\mathrm{G}\phi_1) \mid (\phi_1\ \mathrm{U}\ \phi_2) \mid (\phi_1\ \mathrm{W}\ \phi_2) \mid (\phi_1\ \mathrm{R}\ \phi_2)$$

In the definition above, p is a propositional atom. Also, T, F, $\neg\phi_1$, $\phi_1 \wedge \phi_2$ and $\phi_1 \vee \phi_2$ are all LTL formulas if ϕ_1 and ϕ_2 are LTL formulas. X, G, F, W, U, R are operators. For example, X means next, so Xϕ_1 is true if ϕ_1 is true in the next state. G means globally all future states, so Gϕ_1 is true if ϕ_1 is true for all future states. F means some future states, so Fϕ_1 is true if ϕ_1 is true for some future states. U, R, W in the formula represent Until, Release, and Weak-until.

The details of all the operators can be found at the book Logic in Computer Science (Huth and Ryan, 2004).

1.3 SMV and Spin

NuSMV (also known as SMV) that stands for “New Symbolic Model Verifier.” As an open and structured platform for model checking, SMV provides a description language for system modeling and directly checks the linear-time temporal logic on these models. SMV takes (1) a program written in a description language that describes the system, and (2) a specification written in linear temporal logic as the input. The output is either true, if the specification holds for the system, or a trace that leads to the failure showing why the specification does not hold for the system (Huth and Ryan, 2004).

Another model checker, Spin (Simple Promela INterpreter), is developed by Gerard Holzmann at Bell Labs in 1980. It is widely used in analyzing Promela programs on system design errors such as deadlocks and assertion violations. The systems that Spin works on are described in Promela (Process Meta Language), which is a modeling language used to describe concurrent and distributed systems such as network protocols. Similar to SMV, Spin verifies properties in linear temporal logic formulas. Because most of our experiments are implemented in Spin, we will discuss the use of Spin in details in the experiment sections.

1.4 Thesis Organization

The remaining of the thesis is organized as follows. Chapter 2 provides the background information and discusses the related works of literature on foot-and-mouth disease and model checking. Chapter 3 explains the approach and methods we used in details. Chapter 4 contains implementation of the experiments and discussion of the experiment results. The last chapter discusses the summary, limitation and future work for this project.

Chapter 2

Background and Related Work

2.1 FMD and Related Research

Foot-and-mouth disease (FMD) virus can cause symptoms including high fever, weight loss, and vesicles around hooves and mouths to cloven-hoofed animals such as cow, sheep and pig (Donaldson, 1987). There is no FMD in US and UK currently; however, given the hardiness of FMD virus survival and high transmissibility, it poses a severe threat to the society (Anderson, 2002). The FMD virus also has a significant impact on the economic aspect. For example, the countries that are free of FMD have trade restrictions on any products derived from FMD-susceptible animals, which causes a huge financial loss for the FMD-susceptible country (Knight-Jones and Rushton, 2013). In particular, the impact of the 2001 outbreak of FMD in the UK was estimated at £8 billion (Anderson, 2002). Given this background, it is of great importance to find effective control strategy to prevent FMD outbreaks.

FMD is a foreign animal disease, and the field experiments are difficult or prohibited. In this situation, mathematical and computation modeling methods have been proven to be extremely helpful in controlling FMD. Most of the previous researches have utilized stochastic microsimulation models to measure the performance of different control policies. For example, simulation methods have been used to study control policies for FMD in Australia (Garner and Beckett, 2005), Canada (Sanson et al., 2014), Japan (Hayama et al., 2013), North America (Harvey et al., 2007), and UK (Tildesley et al., 2006; Keeling et al., 2001). However, for stochastic simulation models, the optimality of any control action is presented as an expectation. Although other statistics may be investigated (Probert et al., 2016), in general, there is no certainty in determining whether certain future states of the outbreak are possible for a particular control action. In comparison, the model checking approach provides new insight into this problem since it gives a definite answer to the question of whether particular future states of an outbreak are possible when given current configuration of the contagion network.

2.2 Model Checking and Related Research

Clarke (2008) gives a thorough introduction of the birth and development to the model checking technique. Model checking is very useful in verifying system properties, and the combination of model checking and network modeling is novel (Clarke, 2008; Clarke et al., 2000). The verification process in model checking is automatic while the models such as networks of people or animals can be constructed manually or by using other tools. There are various model checkers, among which SMV and Spin are modern model checkers that have been widely used in both Industry and Academia. SMV is a new symbolic model checker, which provides an open and structured platform for model checking (Cimatti et al., 1999; Cimatti et al., 2002). Spin was developed at the Bell Labs in 1980. It is an open source software that is primarily used for the verification of multi-threaded applications (Spin, 1980).

Many references have commented on the difficulties of controlling the spread of infection in a network. Dreyer and Roberts studied on the irreversible k-threshold model of disease; they examined effective policies that prevent the infection propagation over network without violating desired properties (Dreyer and Roberts, 2009). Finbow and MacGillivray, Chlebíková and Chopin studied on the classic firefighter problem; they reported that it is, at least, NP-hard for the problem of identifying an effective policy to control the spread of infection in a network, even for trees or graphs of degree three (Finbow and MacGillivray, 2009; Chlebíková, and Chopin, 2014).

Anshelevich et al. presented a method to contain the infection by using targeted vaccinations, where they are allowed to vaccinate at most a fixed amount of nodes at each time step (Anshelevich et al., 2012). Newman reviewed the development of network systems and examined the infection propagation problem with real-time constraints (Newman, 2003). Santhanam et al. did a preliminary study on the infection propagation problem; he verified the effectiveness of intervention policy that counter-acts the infection spread through model checking technique (Santhanam et al., 2011). In this work, they assumed the underlying network and the propagation are already known. For our research project in this thesis, we construct both the underlying network and propagation function from data.

Chapter 3

Approaches and Methods

3.1 Farm Network and Propagation Function

The primary task of our work is to choose the best control policy for stopping an FMD outbreak. Our control policy consists of culling the infected farm once it is reported after the latent period and vaccinating the neighbor farms around the infected ones. Here the neighbor farms are those within a certain distance to the infected farms. We define this distance as the vaccination radius *RV*. The control policy can then be changed by setting the parameter *RV* in the model.

There are two important constraints relevant for control of FMD during an outbreak: a vaccination schedule constraint and a global carcass constraint. The vaccination constraint means that only a limited amount of vaccinations can be used per day. While the carcass limit means that only a limited number of carcasses can be presented throughout the landscape at any point in time. For our study models in this thesis, we will treat each farm as a whole unit regardless of the number of animals in each farm. In other words, we consider all the animals on an infected farm as infected and do not model the within-farm dynamics of disease spread. In this case, we do not need to consider the carcass limit or the max number of animals the farm can cull each day. Instead, we only consider the number of vaccinations that can be used as the main constraints of our model.

In the following, we will introduce our model including the network of farms and transition rules which describe disease propagation and control action. We assume the whole landscape is a grid, and each farm is represented as a node in the grid. We use enumeration types to define the states of the farms as follows.

(1) Healthy: the farms that are healthy and open to infection.

(2) Infected: the farms that become infected and can infect their neighbors within the infection radius. However, an infection cannot be detected at this stage.

(3) Reported: the infected farms now can be detected and reported.

(4) Dead: after being detected and reported, the whole farm becomes dead (culled).

(5) Vaccinated: the farms are vaccinated and immune to the disease.

We can set the initial state so that some farms are infected and other farms are healthy. The transition takes place at each discrete timestamp t = 0, 1, 2, etc. We define the following five transition rules for our model.

(a). Each infected farm has a 50% possibility to infect its healthy neighbor farms as long as the neighbor farms are within a certain distance to the infected farm (state: healthy -> infected). Here we define this distance as infection radius *R*.

(b). The farms that are infected but not detected will become detected and reported after the latent period (state: infected -> reported).

(c). If a farm has been detected and reported, we vaccinate all the neighbor farms within the vaccination ring of the reported farm. If a neighbor farm is healthy before vaccination, the farm will become vaccinated and immune to infection in all future timestamps (state: healthy -> vaccinated). However, the vaccinations have no effects on farms that have already been infected.

(d). The infected farm will be culled and becomes dead once reported (state: reported -> dead).

(e). Except for the above cases, the farm will remain in its current state in the next timestamp.

It is worth mentioning that the radius of both infection and vaccination ring can be set as the parameters of the model. The infection radius *R* is set based on a real life scenario, and different values of vaccination radius *RV* are considered as different control policies to deal with the infectious disease. The length of the latency period (the time it takes for a farm to change from infected state to reported state) can also be changed based on the real scenarios.

3.2 Identifying a Control Policy

The primary function of Model checking is to check $M, s \models \phi$, namely to check whether some properties ϕ hold for a given model M with a given initial state s. The output of the program will either be true, if the properties hold for the system, or error. For the latter case, model checking provides traces that lead to the error showing why the properties do not hold for the system. By taking advantage of this particular design of the model checking technique, we can propose different intervention policies and use model checking to check whether each policy is successful in stopping the infection. In our experiment, we consider a policy successful if more than half of the nodes are in the healthy state (including the vaccinated state). We consider a policy failed if less than half of the nodes are in healthy state. Based on this definition, we can encode the properties in LTL as below to find a success policy.

ltl p { <> (*ltl that describes the opposite of the success policy condition*)}

The above LTL asserts that there exists a situation in which the policy fails. If the program fails to pass the property, the counterexample provided by model checking will be our expected result that there is a situation the policy succeeds. Model checking will also show the traces of how each node evolves over time that leads to the error. These results can be used as our intervention policy to stop the infection.

Another specification that we want to check is the time limit t_{max} as an expected timeframe to prevent the infection. Although t_{max} is not encoded in the program as LTL, the model checking program stops at t_{max}. It means that we fail to find an effective policy to counter the infection propagation within expected time frame.

3.3 Effectiveness Evaluation of Control Policy

The effectiveness of different policies can be studied from various aspects. In this thesis, we measure the effectiveness of each policy with the following criteria: (1) the number of days needed to stop the infection, (2) the number of vaccinations used to stop the infection, and (3) the number of dead nodes in total. We run the Spin model with different policies and compare the results in regards to those three measurements. In practice, we want to find the policy that takes less time and vaccination to stop the infection and results in less dead nodes. Based on different constraints and needs in real life scenario, we may also put more emphasize on one of the criteria.

It is worth mentioning that simulation methods are usually used for evaluating the effectiveness of control policies. In some simulation process, it will provide a random result from all the possible answers. Compared to that, model checking approach itself cannot evaluate the policy because it will always return the same result from all possible answers. However, the model checker Spin has a simulation mode that we can utilize to do the evaluation work. We will discuss this in details in the experiment and results section.

3.4 Checking Possible Future State

A significant advantage of model checking in our project is that it can provide certainty as to check whether particular future states of an outbreak are possible or not from a given current configuration of the contagion network. By taking advantage of this function, we can answer some proposed interesting questions such as whether all nodes become dead is a possible future state, or whether is it possible that we can control the infection with a given amount of vaccinations. As long as the proposed questions can be encoded in LTL formulas, model checking can provide an answer to these questions.

Chapter 4

Experiments and Results

4.1 Experiment Setup

Model and Data Description

We use the model checker Spin to perform the experiments and implement the models. Two models are applied in our experiments, the theoretical toy model, and the real data model. The theoretical toy model is a 10×10 square grid that contains 100 nodes that are used to represent farms. The real data model we examine is acquired from Center for Infectious Disease and Dynamics (CIDD) at Penn State University, which consists of 4000 farms with 500 animals in each farm. Farm locations have been randomly generated in a circle around a single infected farm consistent with the spatial distribution of farms in the county of Cumbria, UK, from the 2001 outbreak of FMD. The location data are numbers representing the x and y coordinates of each farm in unit of kilometer (km). Because of scalability issues, we cannot run Spin on a model of 4000 farms. We therefore choose the first 500 farms out of the 4000 farms to build the real data model. For both models, we store the state of each farm in a one-dimensional array named "$farm$", the location of the farms (x coordinate and y coordinate) are stored in another two one dimensional array named "x" and "y". The index of the array that stores the state is consistent with the index of the arrays that holds the location. For example, the location of $farm[0]$ is stored in $x[0]$ and $y[0]$. The distance between two different farms can be calculated based on their x and y coordinates. For example, the distance d between $farm[0]$ and $farm[1]$ is given by $d = \sqrt{(x[1] - x[0])^2 + (y[1] - y[0])^2}$.

Real Data Pre-Processing

The original location data we acquired from CIDD are decimal numbers representing the x and y coordinates in unit of *km*. In order to perform model checking approach to the real data, we need to first pre-process the raw data and turn those location numbers into integers because Spin does not support double or float type. The largest data type for the numeric numbers in Spin is 32-bit Integer. The purpose of the design without floating numbers is to facilitate the verification process. We pre-process the raw data as follows. For each location number (x and y) in the original file, we multiply them by 100 and round the number to the nearest integer. We then add 500 to each number to bring all the x and y values to a positive range. The above operations do not change the relative location of each farm but change the unit from *km* into 10*m*. Those processed data can be directly used to store the location of each farm and calculate the distance between farms.

Running Spin

Spin and iSpin (with graphical interface) are both able to do verification on a model. While iSpin is very convenient and easy to understand because it provides a full menu of choices, Spin is more adaptable and can be used in any environment. We use Spin for our experiments. The experiments for answering proposed questions are performed in a 2.90 GHz Intel Xeon processor with 131GB memory Linux server, and the experiments of evaluation part are performed with 2.7 GHz Intel Core i5 processor with 8GB memory on OS X Yosemite.

We start by building a model with Promela. In our code, the "init()" is used as the main function to start the program, the data that consists of locations and state of the farms is defined in the "readfile()" prototype, and the infection propagation function is defined in the "infect()" prototype. All the properties we want to check in this experiment are written in LTL. The details for LTL in Spin can be found on Spin official website and manual. The commonly used operators are listed in the following: [] means always, <> means eventually, and ! means negation. As we have mentioned before, the output of model checking is either success if the property (written as an LTL formula) holds for the model or error with a counter-example that violates the property. Due to this particular point, we can write the negation of the property we want to check as the LTL formula. For example, if we want to check whether a > b is a possible state or not, we can write a <= b as the LTL formula. In this way, if the property a <= b does not hold for the model, model checking will provide us a counter-example which satisfies a > b. All the LTLs in Spin are written in ltl [name] {formula} format. They all start with the "ltl" keyword, followed by the name of the LTL property, and, at last, the LTL formula written in the {} bracket.

Spin provides both the model checking (verification) mode, which does a full search of the possible states of the model, and the simulation mode, which performs a random simulation run of the model. The algorithms of both modes are shown in the figure below (Ruys, 2002). For the simulation mode, Spin uses random seeds for a random simulation run. While there is no error and deadlock, Spin will collect all current executable actions and randomly chooses one action to execute (Ruys, 2002). In our project, it will choose one path from all possible paths. For model checking mode, Spin uses depth first search to explore all the possible states. We use the exhaustive full search mode (model checking mode) to perform the major experiments. Since this mode usually takes a long time to run, we first utilize the simulation mode to do a quick debugging on the system. Besides, given a fixed model and a LTL specification, the model checking mode always returns the same result. We, therefore, use the simulation mode for verifying the effectiveness of a policy.

```
//Spin Simulation mode Algorithms
while (!error & !Blocked) {
    ActionList ActionMenu = getAllExecutableActions();
    Blocked = (ActionMenu.size() == 0);
    if (!Blocked) {
        Action act = ActionMenu.chooseRandom();
        error = act.execute();
    }
}

//Spin Verification mode Algorithms
procedure dfs(s: state) {
  if error(s)
      getError();
  foreach (successor n of s) {
      if (n not in Statespace)
          dfs(n);
}
```

Figure **4-1**: Simulation and verification mode algorithms of Spin.

The memory usage is usually the major challenge for the performance of model checking because the model checker Spin stores every state of the model during running. To overcome this issue, Spin provides both the exhaustive verification mode and supertrace verification mode. The decisions on which mode is proper is based on whether the system can afford CPU and memory. A full exhaustive verification guarantees to get an accurate answer, whose performance can be improved by either setting the memory limit or use the collapse compression. If the exhaustive mode exceeds the memory bound and becomes infeasible, supertrace verification, also known as bitstate hashing can be utilized as an alternative approach. It can provide an approximate answer to whether errors exist. We use exhaustive search for most of our experiments, but for some part of the experiments where exhaustive verification is not feasible, we apply supertrace verification. Spin also provides a wide variety of run-time options. We will not discuss those options here, and the details can be found on Spin official website and manual.

The output from the model checking program is text-based when a violation of the property occurs. If needed, structured representation and intermediate steps can be extracted. The output can be displayed directly on the console, or be written to an external file. It can be as specific as we need because we can encode the program to provide the traces that lead to the violation.

4.2 Checking Possible Future States

We answer several interesting questions in this section. Unless otherwise specified, the basic settings of the experiment parameters are as follows. We assume the initial configuration as that only one farm in the center of the landscape is infectious and all the other farms are healthy based on the data. For the theoretical model, both the infectious radius *R* and the vaccination radius *RV* are equal to 1, which is one tenth of the side length of the landscape. For the real data model, both the infectious radius *R* and the vaccination radius *RV* are set to 400 (in unit of 10*m*) based on the data, which is also approximately one tenth of the side length. For both models, the latent period equals 1 day and the maximum time limit for both models are 100 days. The details of the questions and the experiment results are listed as follows.

Q1: Check whether it is possible that all nodes become either infected or dead.

During an emergent outbreak of some infectious disease such as a catastrophic event, it is critical to check whether it is possible that all nodes are affected (become either infected or dead). We encode this property as LTL p1 to solve this problem. The LTL formula written below asserts that the sum of infectious nodes and dead nodes will always be smaller than the total number of nodes (Nfarm).

$$ltl\ p1\ \{\ [\,]\ (count\_InfectiousNode + count\_DeadNode < Nfarm)\ \}$$

For our experiments both on the 10×10 theoretical model with *R*=*RV*=1 and real data model with *R*=*RV*=400, the result of LTL p1 is failed, and the property encoded in LTL p1 is violated, which means there exists a situation that the sum of infectious nodes and dead nodes is equal to or greater than the number of total nodes. Spin also provides a counter example and full traces that lead to the end state. The results for both models are listed in Table 4-1. We find that there is no node in the vaccinated state for both models, which means the infectious nodes infect all healthy nodes within the infection radius *R* in this counter example.

Table 4-1. The number of nodes in different states at each day for the counter example of property p1.

	10 × 10 Theoretical Model				Real Data Model			
	Healthy	Infected	Vaccinated	Dead	Healthy	Infected	Vaccinated	Dead
Day 0	99	1	0	0	499	1	0	0
Day 1	95	4	0	1	480	19	0	1
Day 2	87	8	0	5	438	42	0	20
Day 3	75	12	0	13	370	68	0	62
Day 4	59	16	0	25	277	93	0	130
Day 5	41	18	0	41	146	131	0	223
Day 6	25	16	0	59	15	131	0	354
Day 7	13	12	0	75	0	15	0	485
Day 8	5	8	0	87				
Day 9	1	4	0	95				
Day 10	0	1	0	99				

Q2: Check whether it is possible that all nodes become dead.

When confronting an emergent outbreak, we may want to know how bad the situation would be and whether a worst condition could happen. We thus ask whether all nodes become dead is a possible state. We encode this property as LTL p2 in the following.

*ltl p*2 { [] (*count_DeadNode* < *Nfarm*) } .

The above LTL formula asserts that the number of dead nodes is always smaller than the total number of nodes. We perform experiments on the 10×10 theoretical model with *R*=*RV*=1 and real data model with *R*=*RV*=400. For both models, the property encoded in LTL p2 is violated, indicating the existence of a counter example in which the number of dead nodes becomes equal to or greater than the number of total nodes at a particular state. Table 4-2 shows the counter example for both models where the numbers of nodes in various states are listed for each day. We can see that the results for both models are very similar to the results in Q1, except that the violation occurs at one day later than that of Q1. There is no node in the vaccinated state throughout the disease propagation, and all nodes become dead in the end.

Table 4-2. The number of nodes in different states at each day for the counter example of property p2.

	10 × 10 Theoretical Model				Real Data Model			
	Healthy	Infected	Vaccinated	Dead	Healthy	Infected	Vaccinated	Dead
Day 0	99	1	0	0	499	1	0	0
Day 1	95	4	0	1	480	19	0	1
Day 2	87	8	0	5	438	42	0	20
Day 3	75	12	0	13	370	68	0	62
Day 4	59	16	0	25	277	93	0	130
Day 5	41	18	0	41	146	131	0	223
Day 6	25	16	0	59	15	131	0	354
Day 7	13	12	0	75	0	15	0	485
Day 8	5	8	0	87	0	0	0	500
Day 9	1	4	0	95				
Day 10	0	1	0	99				
Day 11	0	0	0	100				

It is worth mentioning that We can also use the ! in the LTL formula to represent negation. LTL p2p written below is equivalent to p2.

ltl p2p { ! (<> (*count_DeadNode* >= *Nfarm*)) }

The LTL formula p2p asserts that there does not exist a situation that the number of dead nodes is equal to or larger than the total number of nodes. In our experiment, the output of p2p is failed and spin provides a counter example that the number of dead nodes equals to the number of total nodes at the end, which is exactly the same as the counter example shown in Table 4-2.

Q3: Check whether it is possible to stop the infection with a limit amount of vaccinations.

In real life scenario of dealing with the infectious disease, the number of vaccines available is often a significant challenge. It is crucial to check whether it is possible to stop the disease propagation with a limited amount of vaccines. In our experiment, we define the limit number to be 50% of the total nodes, and set the vaccination radius *RV* to be twice of the infection radius *R*. Now we write down a LTL p3 in the following asserting that it is always true that either the number of vaccinated nodes is equal to, or smaller than 0, or the number of vaccinated nodes is equal to or larger than 50% of the total number of nodes, or the number of infectious nodes is nonzero.

*ltl p*3 { [] ((*count_VaccineDose* <= 0) || (*count_VaccinatedDose* >= *Nfarm**5/10) || (*count_InfectiousNode* != 0))}

Note that we can use “and” and "or" operators to combine multiple logics in the LTL formula. For our experiments both on the 10×10 theoretical model with *R*=1 and *RV*=2, and real data model with *R*=400 and *RV*=800, the result of LTL p3 is failed. The violation of LTL p3 means that there exists a state that the number of vaccinated nodes is larger than 0 and smaller than 50% of the total number of nodes, and the number of infectious nodes is 0. This particular state is just the successful situation that we are looking for in which the infection is stopped with the limited number of vaccines. We show the counter example provided by spin that leads to this state for both theoretical and real data models in Table 4-3. We can see that the infection stops at the second day with vaccine doses used less than 50% of the number of the total nodes.

Table 4-3. The number of nodes in different states at each day for the counter example of property p3.

	10×10 Theoretical Model				Real Data Model			
	Healthy	Infected	Vaccine	Dead	Healthy	Infected	Vaccine	Dead
Day 0	99	1	0	0	499	1	0	0
Day 1	87	4	12	1	416	19	83	1
Day 2	87	0	12	5	416	0	83	20

We talk about identifying successful or failed policies in previous sections. Another significant aspect of model checking approach is that instead of checking whether a future configuration is possible given a policy, it can check whether a policy can succeed or fail with a definition of success. In the next steps, we define a policy as successful if the number of vaccinated nodes plus healthy nodes is equal to or larger than half of the total nodes when the infection stops, and a policy is failing otherwise. After we define what a successful or failing policy is, we can also use model checking to answer the following questions Q4 and Q5.

Q4: Check whether it is possible that a policy can succeed.

When we talk about stopping the infectious disease, the first important thing comes to mind is to find a successful policy. In this case, it is extremely important to verify whether it is possible a policy can succeed. The LTL formula written below describes a failing situation, which asserts that there exists a state that the number of vaccinated nodes plus healthy nodes is smaller than half of the total number of nodes.

*ltl p*4 { <> *(count_VaccinatedNode* + *count_HealthyNode* < *Nfarm*/2) }

Our goal is to find a counter example in which the total number of vaccinated and healthy nodes is always larger than or equal to half of the total node number so that the policy is successful in this example. For our experiments both on the 10×10 theoretical model with *R*=*RV*=1 and real data model with *R*=*RV*=400, the result of LTL p4 is violated, indicating that there exists a path that the control policies taken in our experiments are successful. The counter example provided by Spin for both models are listed below in Table 4-4. For both models, the results are the same as the results in Q1 for the first 4 days. On the fifth day, the infectious nodes only infect some of the healthy nodes within the infection radius and other healthy nodes are then vaccinated. On the sixth day, no healthy nodes become infected and all the healthy nodes within *RV* are vaccinated.

Table 4-4. The number of nodes in different states at each day for the counter example of property p4.

	10 × 10 Theoretical Model				Real Data Model			
	Healthy	Infected	Vaccinated	Dead	Healthy	Infected	Vaccinated	Dead
Day 0	99	1	0	0	499	1	0	0
Day 1	95	4	0	1	480	19	0	1
Day 2	87	8	0	5	438	42	0	20
Day 3	75	12	0	13	370	68	0	62
Day 4	59	16	0	25	277	93	0	130
Day 5	41	9	9	41	146	27	104	223
Day 6	32	0	18	50	109	0	141	250

Q5: Check whether it is possible a policy can fail.

Similar to what we discussed in Q4, it is equally important to verify whether it is possible that a policy would fail. The LTL formula written below asserts a successful situation that the total number of vaccinated and healthy nodes is always greater than or equal to half of the total number of the nodes. If violated, the counter example gives the path that the number of vaccinated nodes plus healthy nodes becomes smaller than half of the total number of nodes at some time so that the policy fails in this example.

$$ltl\ p5\ \{\ []\ (count\_VaccinatedNode + count\_HealthyNode >= Nfarm/2)\}$$

We still perform experiments on the 10 × 10 theoretical model with *R*=*RV*=1 and on the real data model with *R*=*RV*=400, and the result of above LTL (p5) is violated. This indicates the existence of the failure example, which means the policy taken is possible to be failing. We show this counter example in detail for both models in Table 4-5. The results of infection steps here are the same as the infection steps in Q1 for the first 4 days. On the fifth day, the violation occurs during the infection step once the total number of vaccinated and healthy nodes becomes less than half of the total number of nodes.

Table 4-5. The number of nodes in different states at each day for the counter example of property p5.

	10 × 10 Theoretical Model				Real Data Model			
	Healthy	Infected	Vaccinated	Dead	Healthy	Infected	Vaccinated	Dead
Day 0	99	1	0	0	499	1	0	0
Day 1	95	4	0	1	480	19	0	1
Day 2	87	8	0	5	438	42	0	20
Day 3	75	12	0	13	370	68	0	62
Day 4	59	16	0	25	277	93	0	130
Day 5	49	19	0	32	249	121	0	130

Q6. Finding the lower limit for a policy that can guarantee to stop the infection.

Some infectious disease poses severe damage to the economy and human life. We want to find effective policies that can guarantee success. We can use the LTL in Q5 to find the lower limit for a policy that guarantees to stop the infection successfully. When given a fixed model (with a fixed model size, infection radius *R* and latent period *L*), we can find the lower limit of the vaccination radius *RV* that guarantees to stop the disease propagation.

The setup of this experiment is different from previous questions. We adapt different control policies with vaccination radius *RV* increasing from the infection radius *R*. Specifically, for the theoretical model, we set *R*=1 and increase the *RV* from 1 to 2 with interval of 0.2; for the real data model, we set *R*=400 and increase the *RV* from 400 to 800 with interval of 100. We use the supertrace verification modes for the real data model to afford the memory usage.

For the theoretical model, when *RV* < 2, the output of the program is failed (LTL p5 is violated) with counter examples that lead to failure of the policy. When we increase the *RV* to 2, the output of the program becomes a success, which means that the policy with *RV*=2 will be successful in all possible future states. For the real data model, a similar behavior is seen. When *RV*<600 the output of the program is a failure but when we increase the *RV* to 600, the output of the program becomes a success, which means that the policy of *RV*=600 guarantees success in all possible future states. The detailed experiment results are shown below.

Table 4-6. The results of finding the lower limit of policies that can guarantee stop the infection for both models.

Theoretical Model (R = 1)					
RV = 1	RV = 1.2	RV = 1.4	RV = 1.6	RV = 1.8	RV = 2.0
Fail	Fail	Fail	Fail	Fail	Pass
Real Data Model (R = 400)					
RV = 400	RV = 450	RV = 500	RV = 550	RV = 600	
Fail	Fail	Fail	Fail	Pass	

Q7. Evaluate the effectiveness of different policies.

As we discussed in the previous sections, the effectiveness of a policy is based on the following criteria: (1) the number of days to stop an infection, (2) the doses of vaccinations needed to stop an infection, and (3) the number of dead nodes when infection ends. As we can see from the previous experiment results, when fails to pass on a property, model checking always returns the same result by giving the same counter example. In this case, we cannot evaluate the average performance of an individual policy by using model checking. For this kind of study, people usually run a lot of random simulations, and look at the average performance. Spin itself provides a simulation mode, and we can use this mode to evaluate the effectiveness of each policy. We will study on this in the next section.

Result and Performance Summary

For all the questions above, we summarize the performance (running time and memory usage) of the above experiments in the table below. With the increase of the input size, model checking takes more time to run and uses more memory. Compared to the relatively short running time, memory usage is the primary concern for model checking. We can see that Q6 takes more time and memory than the other questions. As mentioned in the experiment part, we find that the property holds for the model in Q6. In this case, model checking does a full exhaustive search that covers all the possible states before it produces the result that the property holds for the model. For other questions, the running time and memory usage mainly depends on the number of violations existing for this model. In general, the fewer violations exist, the more time and memory it takes for the model checker to find the error.

Table 4-7. Performance summaries of the experiments on both theoretical model and real data model.

Time /Memory (Second /GB)	Q1	Q2	Q3	Q4	Q5	Q6
Theoretical Model	0.31/0.23	0.32/0.24	0.01/0.18	0.18/0.21	0.10/0.20	0.11/0.24
Real Data Model	12.8/5.22	13.1s/5.37	0.36/0.31	6.76/3.07	3.58/1.47	73.4/2.51

Table 4-8. Summary of comparisons between the model checking and simulation approach.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Model Checking	Yes	Yes	Yes	Yes	Yes	Yes	No
Simulation	Not Guaranteed	Not Guaranteed	Not Guaranteed	Not Guaranteed	Not Guaranteed	Not Guaranteed	Yes

We also compare model the checking approach and simulation approach with regards to whether each method can answer those seven questions we proposed. We summarize the results in the table above. Model checking is capable of providing a definitive answer to Q1 to Q6 with a single run of Spin. As long as a question can be written in LTL, it can be solved with a single run. Simulation might also be able to answer these questions with a single run or a large number of runs, but it is not guaranteed because of its random nature. In general, model checking can be used to automatically verify whether some future configurations of a system are possible or not while simulations cannot. Simulations are mainly used to get the average results of an individual policy by running a large number of runs while model checking cannot achieve this. However, using Spin's simulation mode one can also achieve this goal. In the following section, we use Spin's simulation mode to evaluate the effectiveness of different policies.

4.3 Evaluating Effectiveness of Policies.

In this section, we conduct two groups of experiments: first, we want to evaluate the effectiveness of different control policies; second, for a given policy, we want to study how different latent period would affect the infection result. For both experiments, we record three variables: (1) the numbers of days to stop the infection, (2) the numbers of vaccinations needed, and (3) the numbers of dead nodes as criteria for evaluation. We run the spin program with the simulation mode on the real data model with 4000 farms. For the first experiment on evaluating the effectiveness of different policies, we set the infectious radius *R* as 100, and the vaccination radius *RV* as from 150, 160, 170, 180, 190, and 200, respectively. Each *RV* is considered as a different control policy. We run each policy 100 times, and the results of the experiments are shown below in the boxplots.

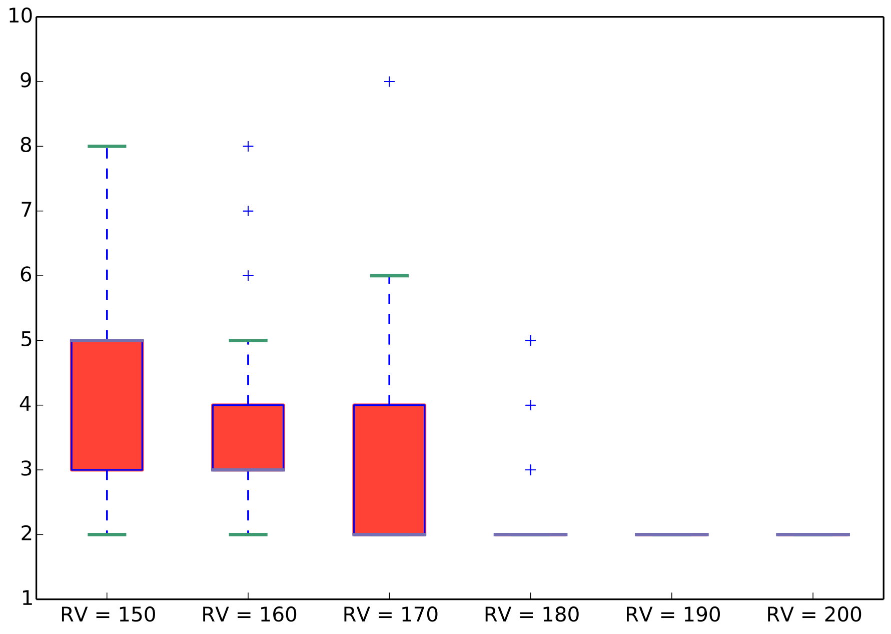

Figure **4-2**. Days used to stop infections on different policies.

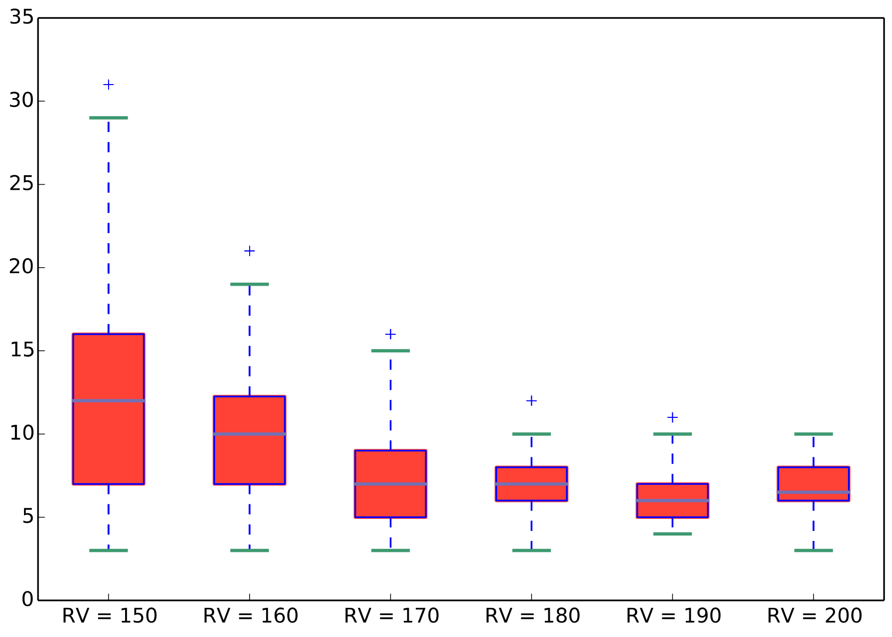

Figure **4-3**. Number of dead nodes on different policies.

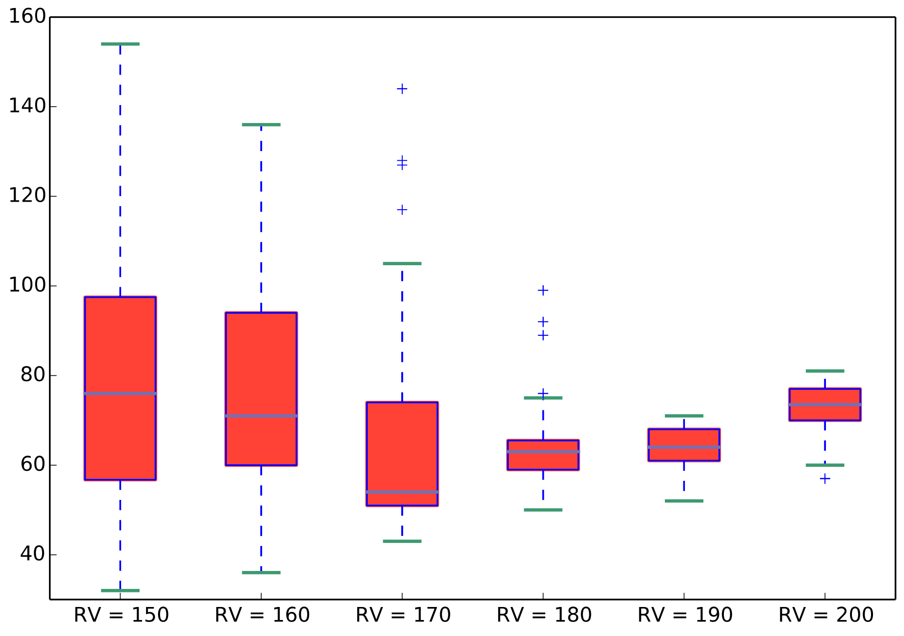

Figure **4-4**. Doses of vaccination used on different policies.

In general, all the three criteria decrease with the increasing of the vaccination radius. In specific, with the increasing of the vaccination radius, the number of days needed to stop the infection decreases. The number of dead nodes and the doses of vaccine used also decrease but do not change much between *RV*=160 and *RV*=200. In general, we would consider the policies from *RV*=160 to *RV*=200 as good policies, and which policy is the best depends on what criterion is more important for us. For example, if we care more about the number of dead nodes, the policy *RV*=190 is the best given the least number of dead nodes. If we only have limited number of vaccinations, *RV*=170 might be a better solution given it only needs tiny amount of vaccinations to stop the infection.

For the second experiment on finding how latent period would affect the infection result, we set the infectious radius *R* as 50 and the vaccination radius *RV* as 100, the latent period *L* as from 1, 2, 3, 4, and 5, respectively. We run Spin with each latent period 100 times, and the results of the experiments are shown below in the boxplots.

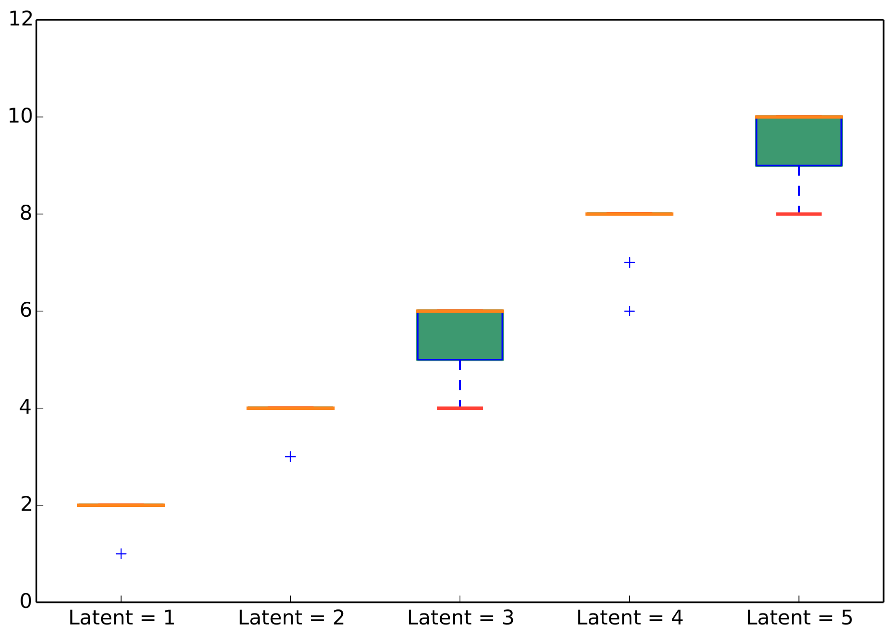

Figure **4-5**. Days used to stop infections on different latent period.

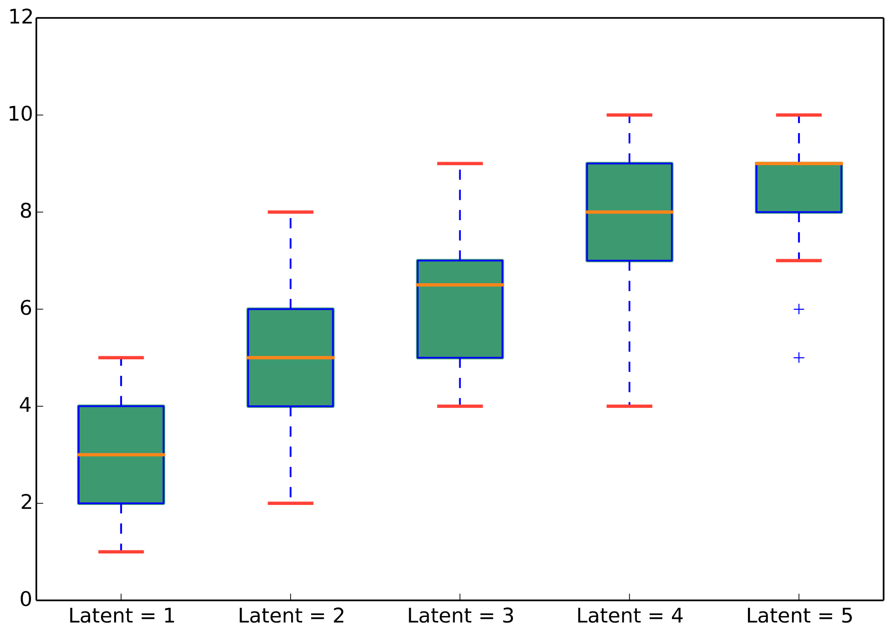

Figure **4-6**. Number of dead nodes on different latent period.

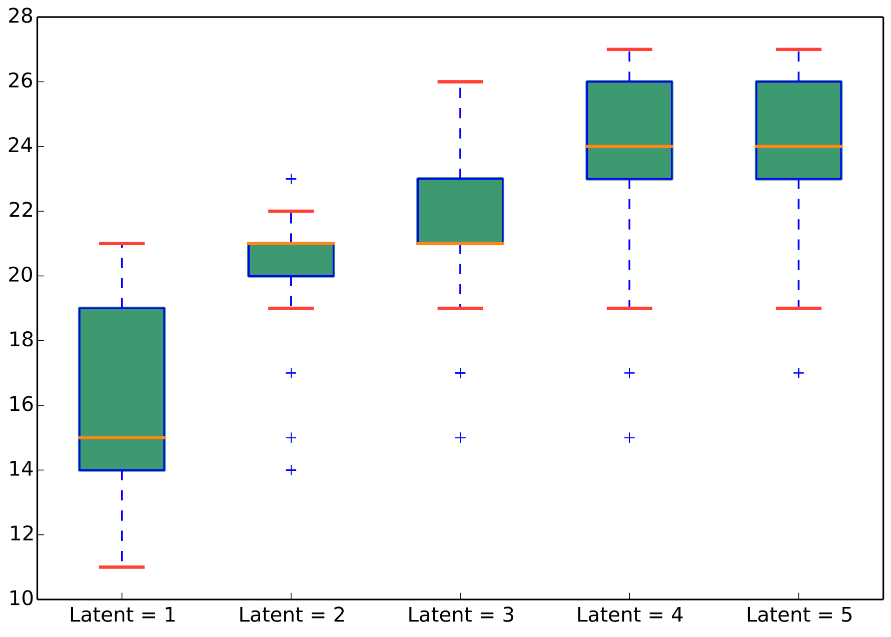

Figure **4-7**. Doses of vaccination used on different latent period.

From the results shown above, all the three quantities increase with increasing latent period. The result is very straightforward given the fact that the longer the latent period, the longer time it takes for the infectious node to become reported. It will have a longer time to infect other nodes before we cull the infected nodes and vaccinate other healthy nodes.

Chapter 5

Summary and Conclusion

This chapter summarizes the experiment results and the achievement of this project. We also discuss the limitation and possible future directions.

Summary

In this thesis, we propose to use model checking to resolve the propagation of foot-and-mouth disease (FMD) over a network of farms on a landscape. We conducted experiments on both the small-scale theoretical model and large-scale real data model based on the data acquired from the Center for Infectious Disease Dynamics (CIDD) at Penn State. The experiment result shows that when given a model with an initial state and a propagation function, model checking approach is able to identify an intervention policy and verify the effectiveness of different policies. More importantly, as there is no certainty for simulation methods to find whether certain future states of the outbreak are possible or not under a particular control action, model checking has the advantage that it can provide certainty in checking whether a future configuration is possible. This is especially useful for checking rare future events, and helps provide valuable insights during an emergent outbreak such as a catastrophic event.

Limitations

As we have mentioned in the experiment section, memory issue is still the main challenge for the scalability of Spin. A Spin model itself does provide different verification modes to deal with the exceeding memory issue. Collapse compression and bitstate hashing help reduce the memory usage to some extent. But still, if we want an accurate result from the exhaustive search instead of an approximate result from the memory saving supertrace verification, we need to afford a large memory usage. One possible solution to overcome this difficulty could be optimizing the way the model is encoded. Given the fact that Spin stores all the variables for each state, using fewer amounts of variables and smaller size data type to store those variables could help to reduce the memory usage for each state, and result in a less memory usage in the full verification process.

Although Spin supports the simulation mode, it is not as convenient as other simulation programs for performing experiments on simulation purposes. Spin does not support float or double data type. Random function is also not included in Spin. Nevertheless, the primary feature of Spin is to check possible future states; thus it is not directly comparable to other programs on simulations purposes. In this sense, this is not a serious limitation.

Future Work

First of all, we would like to solve the memory usage issue to make the current model more scalable. The possible solutions include either optimizing the model itself or exploring more Spin running options to find more suitable options for the current model.

Second, in our experiments, we use a grid structure for the theoretical model and an array to hold the location for the real data model. For both models, the relations between two nodes are based on their distance. In real life scenario, the “distance” could be changing all the time, and the relation between two nodes could be based on their connectivity and relationship instead of their distance. In this case, we want to build a model with an arbitrary network as input, where any node can be connected to any node. This kind of model could be a useful supplement to our current model.

Third, in our current model, the infection probability does not make a difference for the model checking results. As long as an infection is possible, the associate state is stored and searched regardless of the value of the possibility. In practice, however, it is important to assign different weight to an event or a state based on its possibility of occurrence. We plan to use probabilistic model checking to deal with this issue in the future.

At last, the approach we used in this thesis could be applied to other fields including public opinion and computer virus. As long as the data and propagation function can be modeled, and the properties we want to check can be encoded as LTL, model checking is capable of solving the problem. We may also apply model checking to other fields in the future.

References

Tildesley, M. J., Savill, N. J., Shaw, D. J., Deardon, R., Brooks, S. P., Woolhouse, M. E. J., Grenfell, B. T., and Keeling, M. J. (2006). Optimal reactive vaccination strategies for a foot-and-mouth outbreak in the UK. *Nature*, *440*(7080), 83–86.

Knight-Jones, T. J. D., and Rushton, J. (2013). The economic impacts of foot and mouth disease - What are they, how big are they and where do they occur? *Preventive Veterinary Medicine*. *112*(3), 161-173.

Keeling, M. J., Woolhouse, M. E. J., Shaw, D. J., Matthews, L., Chase-Topping, M., Haydon, D. T., Cornell, S. J., Kappey, J., Wilesmith, J., and Grenfell, B. T. (2001). Dynamics of the 2001 UK foot and mouth epidemic: stochastic dispersal in a heterogeneous landscape. *Science, 294*(5543), 813–817.

Hayama, Y., Yamamoto, T., Kobayashi, S., Muroga, N., and Tsutsui, T. (2013). Mathematical model of the 2010 foot-and-mouth disease epidemic in Japan and evaluation of control measures. *Preventive Veterinary Medicine*, *112*(3-4), 183–193.

Harvey, N., Reeves, A., Schoenbaum, M. A., Zagmutt-Vergara, F. J., Dube, C., Hill, A. E., Corso, B. A., McNab, W. B., Cartwright, C. I., and Salman, M. D. (2007). The North American animal disease spread model: A simulation model to assist decision making in evaluating animal disease incursions. *Preventive Veterinary Medicine*, *82*(3-4), 176–197.

Garner, M. G., and Beckett, S. D. (2005). Modelling the spread of foot-and-mouth disease in Australia. *Australian Veterinary Journal*, *83*(12), 758–766.

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

Serazzi, G., and Zanero, S. (2004). Computer virus propagation models. *Lecture Notes in Computer Science*, *2965*, 26–50.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Rovere, M., Sebastiani, R., and Tacchella, A. (2002). NuSMV 2: An opensource tool for symbolic model checking. In *Proceedings of Computer Aided Verification*, *2404*, 359–364.

Newman, M. E. J. (2003). The structure of complex networks. *SIAM Review*, *45*(2), 167–256.

Chleb ́kov á, J., & Chopin, M. (2014). The firefighter problem: A structural analysis. *Parameterized and Exact Computation*, *Lecture Notes in Computer Science*, 172-183.

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

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.

Santhanam, G. R., Suvorov, Y., Basu, S., and Honavar, V. (2011). Verifying intervention policies to counter infection propagation over networks: A model checking approach. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 1408–1414.

Anshelevich, E., Chakrabarty, D., Hate, A., and Swamy, C. (2012). Approximability of the firefighter problem : Computing cuts over time. *Algorithmica*, *62*(1-2), 520–536.

Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M. (1999). NuSMV : A new symbolic model verifier. In *Computer Aided Verification, Lecture Notes Computer Science*, 495–499.

Clarke, E. M. (2008). The birth of model checking. *Lecture Notes in Computer Science*, *5000 LNCS*, 1–26.

Sanson, R., Dub é, C., Cork, S., Frederickson, R., and Morley, C. (2014). Simulation modelling of a hypothetical introduction of foot-and-mouth disease into Alberta. *Preventive Veterinary Medicine, 114*(3-4), 151-163.

Probert, W. J. M., Shea, K., Fonnesbeck, C. J., Runge, M. C., Carpenter, T. E., D ürr, S., Garner, M.G., Harvey, N., Stevenson, M. A., Webb, C. T., Marleen, Werkman., Tildesley, M. J., and Ferrari, M. J. (2016). Decision-making for foot-and-mouth disease control: Objectives matter. *Epidemics, 15*, 10-19.

Garner, M., and Beckett, S. (2005). Modelling the spread of foot-and-mouth disease in Australia. *Australian Vet J Australian Veterinary Journal*, *83*(12), 758-766.

Anderson, I. (2002). Foot and mouth disease: Lessons to be learned inquiry report HC888. *London: The Stationary Office*.

Donaldson, A. I. (1987). Foot-and-Mouth disease: the principal features. *Irish Veterinary Journal*, *41*(9), 325-327.

Ruys, T. C. (2002). SPIN beginners' tutorial. *SPIN 2002 Workshop*, 11–13.

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

Spin - Formal Verification. (2016). Retrieved March 31, 2016, from http://spinroot.com/spin/whatispin.html