SOCIAL AND TRANSMISSION CONTACT NETWORK ANALYSIS OF EPIDEMIC DYNAMICS IN AGENT-BASED MODELS

by

Jiawei Huang

Bachelor of Science, Jinan University, China, 2009

Submitted to the Graduate Faculty of

the Graduate School of Public Health in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2012

Hasan Guclu, PhD

Assistant Professor

Department of Biostatistics

Graduate School of Public Heath

University of Pittsburgh

Bruce Y. Lee, MD, MBA

Assistant Professor

Department of Medicine, Epidemiology and Biomedical Informatics

School of Medicine

Graduate School of Public Heath

University of Pittsburgh

SOCIAL AND TRANSMISSION CONTACT NETWORK ANALYSIS OF EPIDEMIC DYNAMICS IN AGENT-BASED MODELS

Jiawei Huang, M.S.

University of Pittsburgh, 2012

This thesis aims to make social network analysis on synthetic population that is used in FRED system and develop transmission network analysis tools to analyze disease epidemic dynamics in agent-based models of infectious diseases.

The social network of synthetic Allegheny County population consists 1.2M agents. The synthetic population is proved to be an integrated component with average shortest path is 6.91. The risks of being infected for age groups are positively related to the average degree of each group. Although degree distribution has a bifurcating pattern, it is still reasonable to use the synthetic population for modeling disease transmission.

Tools are developed to analyze the transmission network, which generated by FRED simulations. Three tools, **TraceAnalysis**, **StatisticalAnalysis** and **EpidemicDynamic-Plot**, were developed to calculate statistics of transmission networks, to make inference on statistics from different simulation scenarios and to plot epidemic curves. The tools are used to analyze the effectiveness of interventions. School closure and vaccination policies were chosen from FRED to be compared with the baseline FRED, which is run with no intervention policy. The results from network analysis tools indicate the dependency between agents' infection location. Special contact tracing motifs can be found by comparing the discrepancy between the number and expected number of contact tracing motifs. The results show that the schools play important roles in disease transmission.

The public health importance of network analysis tools is to find out the contact tracing motifs, to reveal the strong dependency of locations where infection events happened and to compare the effectiveness of different public intervention policies in agent-based models.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

Infectious disease has been a threat to human societies for a long time. The notorious pandemics of infectious diseases in history include black death of 1347 to 1352, which killed 25 million in Europe, and the smallpox, which killed an estimated 60 million Europeans during the 18th century[1]. The influenza Pandemic of 1918 killed 25-50 million people[2]. Today influenza kills about 250,000 to 500,000 worldwide each year.

In recent years, the emerging diseases, which is when existing parasites become pathogenic or when new pathogenic parasites enter a new host, arouse the nerve of people. Because the spread of disease is fast and it takes time for medical scientists to have vaccine or effective antiviral drugs to control emerging diseases. Under the condition that there is no effective pharmaceutical interventions, non-pharmaceutical interventions, such as school closure [2], are considered to be effective in delaying the peak and mitigate the spread of disease. Efficiency and economical cost of many mitigation strategies have already been intensively investigated by different models.

The modeling of infectious disease is a tool which has been used to study the mechanisms by which diseases spread, to predict the future course of an outbreak and to evaluate strategies to control an epidemic [3]. Compartmental models are first used to investigate epidemics [4]. They are being more and more used, because they are more flexible to incorporate heterogeneity of population in the model. Also they have more realistic assumptions. The assumption that population in subgroups are perfectly mixed in compartmental model is very strong. In computational models, the behavior and social network of every one in the population can be differentiated from each other. The compartmental models, which

[1] http://www.learnnc.org/lp/editions/nchist-twoworlds/1696

[2] http://www.history.navy.mil/library/online/influenza_main.htm

take advantage of the power of super computers, may be more powerful tools for epidemic researches.

FRED is the most recently developed computational model and it is still under development. It built up a synthetic society with synthetic populations and locations. Synthetic is generated based on the actual distribution of demographic characters. Every one of the synthetic population in the system are defined by sex, age, profession, social connection with community and behavior characteristics. Simulation s of how infectious diseases are spread among synthetic population enable us to predict the spread of disease and give the guidelines of how to protect the society.

Although there are many computational models, and many intervention strategies are already discussed using those models, the most commonly used way to indicate the severity of infectious disease are descriptive statistics, such as overall attack rate and the dynamic curves of number of infected people. More useful information of social structure or social network can be extracted by using network analysis. The social network of a person may have significant impact on one's probability of being infected. Also the transmission network, which describes all infection events, can be more informative than the overall attack rate, which is calculated by the number of infected people divided by total number of people in the simulation system. In this thesis, methods from network analysis were applied to develop tools to analyze the social structure of population and the paths of spread of disease in FRED simulation system.

In this chapter, a brief background of agent-based models(ABMs) will be given and followed by the mechanism of FRED. Literature review will cover three fields of studies: the formal studies and models of ABM, the methods of network analysis that have already been used in epidemics research and the motif finding in networks. In the end, there will be an outline of the thesis.

1.1 BACKGROUND

1.1.1 Agent-Based Models

Computer modeling is the process by which a computer is used to develop a mathematical model of a complex system or process. Computer modeling is an efficient way to take into account many different factors and to simplify and organize real-world processes[3].

Computers serve as virtual laboratories where researchers can study problems not easily examined in real life. The experiments consist of computer simulations — representations of actual communities based on demographic and transportation information. In these simulated environments, the researchers can introduce an infectious agent with certain characteristics and then watch it spread.

Agent-based models (ABMs) are dynamic models that simulate the behavior of a system over time. However, they take a bottom-up, or individual-level approach, specifying the rules that govern the behavior of individuals and allowing the overall behavior of the system to emerge from the interactions of those individuals. ABMs are often useful when the input data are focused on an individual rather than a group level or when a small change in the behavior of a few individuals could have a large impact on the system. Because of the heterogeneity of behavior of human beings, it's reasonable to use use ABM to differentiate virtual persons by assigning different demography and behavior patterns. All the individuals are divided into groups, for example, pre-school children, students, adult workers and retired people. Individuals in the same group share similar characteristics. During an epidemic, each individual has a chance of catching or spreading an infection through encounters with others at home, work, school, and elsewhere. How the disease spreads through the population depends on whether and when particular individuals encounter each other, as well as what their particular characteristics are at the time of the encounter.

Framework for Research on Epidemic Dynamics (FRED) is an agent-based model that consists large heterogeneous population. The mechanism of FRED will be introduced in the following section.

[3] https://www.epimodels.org/midasdocs/infoMaterials/MIDAS_101_Comp_Modeling_web.pdf and https://www.epimodels.org/midasdocs/infoMaterials/MIDAS_101_Model_Types_flyer_web.pdf

1.1.2 FRED System

Framework for Research on Epidemic Dynamics (FRED) is a framework designed to flexibly and rapidly allow researchers to build agent based models for simulation of disease spread through populations. FRED was developed by the Public Health Dynamics Laboratory under the direction of Dr. John Grefenstette, Dr. Shawn Brown of University of Pittsburgh and Dr. Roni Rosenfeld of Carnegie Mellon University. I was a Graduate Student Researcher (GSR) in the laboratory to do network analysis of synthetic population and the transmission among the synthetic population.

Figure 1 shows an overview of the FRED system. It is an agent based model for simulation of infectious disease. Synthetic populations which are able to carry and transmit infectious disease are created. The behavior of each synthetic person in the system is defined by demographic parameters, social contacts parameter, etc. By initially assigning agents with certain types of infectious disease, we are able to simulate the process of the spread of diseases and test effectiveness of various kinds of mitigation strategies.

In the rest of this section, I will introduce individual level model in FRED, the synthetic population, disease transmission process and mitigation strategies.

1.1.2.1 Individual Influenza Model Figure 2 shows the individual influenza model. At the beginning of each simulation run, FRED randomly pick random number of persons to be infected and all the other people are at the stage of "susceptible". Susceptible people could be infected by contact with infected people and moved to "exposed" stage. "Exposed" people are not infectious unless they enter into the "infectious" stage after incubation period. Finally, they will be in "Recovered" stage, which they will be immune to future infection.

1.1.2.2 Daily Movement of Agents Just like people in the real world, people with different occupation have different daily schedules. Students in the FRED system will go to classrooms and schools during daytime in weekdays, attend social activities in neighborhood after school and go back to household at the end of the day. Accordingly, the adult workers will go to office and workplace, then neighborhood and household. The link between school and office shows the daily routine of school teachers whose offices are in the schools. Retired and unemployed are also considered in FRED system.

1.1.2.3 Synthetic Population

The synthetic population is generated from US Census Bureau's Public Use Microdata files, and Census aggregated data. In FRED, persons are represented by a virtual agent with demographic and behavior characteristics information. The construction of the synthetic population used the processing method developed by Wheaton et al in the paper *Synthesized Population Databases: A US Geospatial Database for Agent-Based Models* [5]. The overall process includes four steps: (1) generating synthesized households and virtual residents; (2) generating synthesized virtual schools and assigning school-age agents into schools; (3) assigning adult agents to workplaces and (4) generating additional agents to live in group quarters housing. Overall, each agent has information of age, sex, employment status, occupation, household location, household membership, school assignment of students and teachers, work location assignment of employed adults, work status as employed or unemployed, and disease status.

In this thesis, the synthetic population is the population in Allegheny County, Pennsylvania, which is extracted from the U.S. synthetic population. There are totally 1,242,755 people. Figure 5 shows the distribution of age of this synthetic population.

Each individual in synthetic population is scheduled to go to several locations everyday. All the synthetic locations in Allegheny County are plotted by longitude and latitude in Figure 6. Rivers can be seen clearly. The colors are used to differentiate the neighborhood that each location belong to.

There are 657947 locations and each of them is categorized into one of the three categories (Household, Workplace and School), which end up with 537,405 households or **H**, 52,535 workplaces or **W** and 476 schools or **S**.

1.1.2.4 Social Networks and Disease Transmission Process

Agents can be infected by contacting infectious agents in the places where they go during the day. For example, a student in household have a chance to be infected by infectious family member. When he or she goes to the school building, he or she has a probability to contact anyone in the building, some of those may be infectious people. So the student have chance to be infected by them. Accordingly, during the weekends when the students do not need to come to the school building, they may attend social activities in neighborhood and contact infectious people who they may not be able to contact during weekday. Thus, the transmission can happen in different places where people have a chance to contact an infectious person.

Disease transmission probability depends on where transmission occurs, susceptibility of infectee, infectivity of infector and who infects whom [6, 7].

1.1.2.5 Intervention Strategies Public health policies are designed to efficiently and effectively control the spread of disease. Intervention strategies such as vaccination, antivirals, school closure and combination of them can be discussed under ABM models. The severity of disease will be reduced by incorporating mitigation strategies. For example, if vaccination is available to the public, and high percent of agents in the model can be vaccinated, herd immunity may prevent infectious disease to become an outbreak.

The effectiveness of various mitigation strategies have been investigated by many agent-based models, such as vaccination [8, 9, 7], antiviral drugs [10] and social distancing measures [2, 10]. And FRED has implement all of those strategies.

1.1.2.6 Results of FRED To cover a whole epidemic season, FRED needs to simulate all the movements and contacts between 1.2 million synthetic persons or agents for 240 days. It is really time consuming. The implementation of mitigation strategies will even add more calculation time. For each simulation scenario with specific intervention strategies implemented, FRED will run multiple times to evaluate the effectiveness of the strategy because of the randomness of the simulation. It takes 2 hours to do 20 runs of baseline scenario in an Mac OS operating system with two 3.06GHz Intel cores.

After running, FRED will save the records of simulation process into local files for further analysis. For each run, error file, log file, infection file, out file, trace file and vaccine file will be generated. Their contents and functions are listed below:

- Error file: empty if there is no error in the sumulation. It keeps the records of all running errors.

- Log file: size ≤1MB. It stores all output from standard output.
- Infection file: size ≤ 100MB. It stores information for all infection events. The more infection events, the larger the file would be. A typical line in this file would be
  day 2 dis 0 host 125597 age 14.751 from 753726 inf_age 14.929 at S.
  A 14.7 years old child with id 125,597 was infected at School by the agent with id 753,726 and age 14.9 on day 2.
- Out file: size ∼50 KB. It stores the number of people in each subgroup of S, E, I and R by each day and other statistics by days. A typical line would be
  Day 7 Str 0 S 1242156 E 103 I 345 I_s 244 R 151 M 0 C 74 N 1242755 AR 0.05 CI 100 V 746461 RR 0.00 NR 74 CAR 0.03 Sat 2011-01-08 Year 2011 Week 1.
  It means that at the end of day 7 after the outbreak, number of susceptibles is 1242156. 103 agents are exposed, 345 agents are infectious, 151 agents are in the recovered state. The total number of agents is 1,242,775 and the overall attack rate is 5%.
- Trace file: size ∼170 MB. It stores the information of health status of each agent. It will be described in section 3.1.1.
- Vaccine file: ∼20MB. It stores all vaccination injection events. A typical line would be
  id 74082 vaccid 0 vaccday 0 age 70 iseff 1 effday 14 currentdose 0.
  This line means the agent No. 74082 received vaccination on the beginning of outbreak season. The vaccine will be effective on two weeks later.

The typical way of seeing the severity of an outbreak is using the information in "out file" to plot the epidemic curves of exposed agents or infectious agents as Figure 8 shows. The three lines represent the number of infectious agents by days of outbreak under three running scenarios. Red is for school closure policy, green is for vaccination policy and black is for baseline, which does not include any intervention policies.

The input, output files and simulation process of FRED are introduced above. Although the comparison made in Figure 8 can explain the relative severity of infectious disease under different intervention policies, there are still many questions cannot be answered. For example, what is the risk of agents in different subgroups? Will any intervention policy may increase the risk of agents in other subgroups while reduce the risk of the target subgroup? Will there be any relationship between the places where infectors got infected and the places where infectors transmitted to others? And is there any special group that play important role in spreading disease? The transmission process can be mined by taking advantages of information stored in FRED output files, such as infection files and trace files. So that the comparison of different scenarios can go much further than merely the comparison of number of infected agents.

1.2 LITERATURE REVIEW

As computers become faster and faster, ABM have has more and more used to solve a variety of business and technology problems. Examples of the applications include consumer behavior [11], supply chain optimization [12], population dynamics [13], language choice dynamics [14] and epidemics. The modeling of infectious diseases using ABM is an emerging new method to develop computational models of the interactions between infectious agents and their hosts, disease spread and response strategies. Section 1.2.1 will give more detailed review of application of ABM in epidemic studies.

Network analysis is a method to study the relations between discrete objects. It is closely related into the study of infectious disease, since each person can be seen as an object and the relation between objects can be the social contact relationship, which represents the two objects have probability to contact each other, or the infection relationship, which means one person infect another. Network analysis can be easily integrated into epidemiological research, especially infectious disease studies. More detailed review of network analysis and epidemics studies will be given in section 1.2.2.

The more nodes and edges a network have, the higher level of complexity it will have. In order to make any sense of complexity present, a large variety of measurable properties that can be used to characterize networks [15]. Section 1.2.3 will give brief review of properties of network and their applications.

1.2.1 Agent Based Models and FRED

Before 2000, many models for the spread of infectious disease in population were analyzed mathematically and applied to different diseases. Starting from the very basic "SIR" model [4], mathematical models are becoming more and more complex and can incorporate more features. The recent models that have been developed involve aspects such as immunity, gradual loss of immunity, vertical transmission, disease vectors, quarantine, age-structured infectivity and so on. Specific models are formulated for diseases like measles, rabies and syphilis [16]. But mathematical models have their limitations to evaluate the effectiveness of the complex public intervention policies.

As the computers becoming faster and faster, the power of computer is more and more used by scientists to answer questions that are too complex for mathematical models. Modeling of infectious diseases using agent-based models is more realistic and powerful to examine how an outbreak could spread in a variety of scenarios.

An agent-based model (ABM) (also sometimes related to the term multi-agent system or multi-agent simulation) is a class of computational models for simulating the actions and interactions of autonomous agents (both individual or collective entities such as organizations or groups) with a view to assess their effects on the system as a whole. The models simulate the simultaneous operations and interactions of multiple agents, in an attempt to re-create and predict the appearance of complex phenomena. The process is an emergence from the lower (micro) level of systems to a higher (macro) level. As such, a key notion is that simple behavioral rules generate complex behavior.

ABM has been applied to the study of infectious diseases since early 2000s when one of the first large scale ABMs of infectious disease, Episims [9], was developed by Los Alamos National Laboratory. Physical contact patterns are based on actual census, land-use and population-mobility data. The Episims simulate the spread of smallpox and can be run by different mitigation strategies. Two more ABMs [17, 10] were developed to address the effective strategies to contain H5N1 in Southeast Asian. Synthetic population consists of hundreds of thousands to millions of agents in Southeast Asian was generated for simulations. The interventions strategies include targeted antiviral prophylaxis, pre-vaccination and social distance measures. The comparison of effectiveness of intervention strategeies are mainly based on the overall attack rates and epidemic dynamic curves. During the development of ABMs, more realistic assumptions are implemented into them to allow simulation to have more eloquent guidance on contaminating the real outbreak situation. Since the vaccination and antiviral drugs may not be available or sufficient for all people who need them, the prioritization of limited vaccination [8] and logistic plans of drug distribution are added into system. Multi-dose vaccinations are also implemented to be more realistic. More complex situations like multiple outbreaks and evolution of virus are also implemented or being implemented into one or more ABMs of infectious disease.

FRED is an ABM that has being developed by Public Health Dynamics Lab since 2009. It's a framework of multi-strain simulation system for epidemics that is based on the realistic U.S. population. Besides the features already present in other ABMs, FRED supports complex vaccination availability schedule, studies of human health behaviors and viral evolutions. The first utilization of FRED happened during 2009 H1N1 influenza pandemic. The problems addressed by FRED simulation on H1N1 influenza are listed below:

- **School Closure**: When strictly maintained for at least 8 weeks, school closure could delay the epidemic peak by up to 1 week, allowing additional time to develop and implement other interventions. A school closure lasting less than 2 weeks could actually facilitate a faster flu spread, because susceptible students return to their schools in the middle of an outbreak [2]. Nevertheless, because closing schools could have resulted in substantial costs to society as the potential costs of lost productivity and childcare could have far outweighted the cost savings in preventing influenza cases [18].
- **Vaccine Prioritization and Availability Schedule**: The Advisory Committee on Immunization Practice (ACIP)-recommended prioritizing at-risk individuals, rather than only high-transmitters (e.g., children) may increase attack rate, but it reduced serious disease and death, overall economic cost and resulted in the largest savings [8]. However, vaccinating all remaining workers captured additional savings and reduced healthcare worker's and critical infrastructure workers chance of infection. And once 20% compliance was achieved, marginal benefit will be less [6]. Poor counties should have the same access to influenza vaccines as wealthier ones, because they tend to have high-density

populations with more children and other higher-risk people per household, resulting in more interactions. Both increased transmission of influenza and greater risk for worse outcomes [19].

FRED is an outstanding ABM that can be easily adapted to a wide range of infectious disease outbreaks. It can give quick responses by evaluating a variety of public intervention strategies and offer evidence for policy makers. FRED is till in the development stage as of writing this thesis, more new features are being added into simulations system to make it more powerful for predicting future outbreaks and giving scientific evidence for policy maker.

1.2.2 Network Analysis and Infectious Disease Studies

Network analysis is heavily used in the epidemiological studies especially infectious disease studies. This is because the connections between persons that allow infection propagate from person to person naturally formulate a social contact network. The social contact network potentially defines transmission paths, so analyzing its structure and the effect of this structure on disease transmission process are helpful for disease control. Derived from the social contact network, a transmission network can be defined by eliminating all uninfected individuals and link all individuals by infector-infectee relationship. Understanding the structure of transmission network allows us to improve predictions of the likely distribution of infection and interactive waves of spread between subgroups.

Generally, three kinds of network are integrated into infectious disease studies in order to find out the mechanism of spread of infectious diseases and the effective way to control the spread. They are real network, simulated network, idealized network.

1.2.2.1 Idealized Networks A class of idealized networks are proposed to reveal the features of network structures that determine the epidemic dynamics. Models for random networks, lattices, small-world networks, spatial networks, scale-free networks and exponential random networks and their implication for epidemic spread were discussed [20].

Effectiveness of interventions were also discussed in idealized networks [21]. In the literature, idealized networks are built by network generating algorithms, then various interventions are tested on them. The effectiveness of antiviral treatment, reactive vaccination, school closure and their combinations were compared by overall attack rates. The idealized networks address the determinant of dynamics directly, but its generalization to the real work social network is suspicious.

1.2.2.2 Realized Contact Networks

Realized contact networks are used to delineate the transmission routes within infected populations. They are constructed by using snow-ball sampling method.

For the situations when the number of people involved in a disease outbreak is not large and the transmission routes are clear, it is possible to use snow-ball sampling to rebuild the contact networks and transmission routes. For example, the HIV and other STDs give very obvious transmission routes. The contact network of a disease can be formulated by asking every individual to name all their sexual partners over a given period. Many researchers used this method to build realized contact network to study the spread mechanism of the disease, such as HIV [22, 23, 24] and Tuberculosis [25]. The spread of obesity, even though it is a non-infectious disease, can also be investigated by friendship/kinship networks [26], as well as psychological phenomenon, such as happiness [27] and loneliness [28]. This approach has significant limitations under the condition when the infectivity of a disease is high and lots of people are infected or when the transmission routes of the disease are vague, such as air-born diseases.

1.2.2.3 Simulated networks

Given the tremendous difficulties of rebuild the whole contact network or transmission network, a variety of methods have been developed to generate synthetic populations and networks from Census or large survey of egocentric data.

Egocentric data consists of demographic and behavioral information on a number of individuals (the egos) and their contacts (the alters). The egocentric data, though cannot infer connections between egos, has valuable information on heterogeneities of the real social network. The NASTAL [29], which studies the sexual contacts, and the POLYMOD [30], which studies the social interactions within 8 European countries, gathers such egocentric data. Algorithms are then used to generate network of egos by probabilistically assigning each ego a set of contacts drawn in the information available from egocentric data. The algorithms are case-by-case based on different egocentric data that the network is formulated from [31, 32]. The synthetic population used in FRED system is in U.S. census data with algorithms developed by Geospatial Science and Technology Program in Research Triangle Institute. The details can be found in a technical report [5].

1.2.3 The Properties of the Contact Network

The contact network of the synthetic population in FRED system, which include over 1.2 million individuals, is an undirected network that is very complex. The number of edges between the individuals are over 2.6 billion. The transmission network is a directed network and have much smaller individuals and fewer edges, but it is different for each simulation run, while the contact network is static. The more people get infected, the more complex the transmission network will be. For example, an outbreak, which cause $n(n \ll 1.2M)$ individuals infected, will have number of edges less than n, or specifically $n - n_i$, where n_i is the number of people initially selected as infectious. Even though, the complexity of the transmission network prevents us from getting any clear knowledge by visualizing the network. Proper network statistics are needed to represent the characteristics of the contact network and to compare the differences between different transmission networks.

A few widely-used used statistics as well as their definitions and applications will be introduced in Table 1.2.3. They will be used to analyze the contact network and the transmission network in FRED.

1.2.3.1 Contact Tracing Motifs Finding network motifs is another powerful tool that is widely used in genomic studies.

Network motifs are connectivity-patterns (sub-graphs) that occur much more often than they do in random networks. Network studies in biology, ecology and other fields show that the network is largely composed of these network motifs. These motifs can be considered as simple building blocks from which the network is composed, so the network can be characterized by the motifs.

Motifs were first observed in gene regulatory networks in transcription of bacteria *E. coli* [33]. Later they were also found in other bacteria's transcription networks [34] such as yeast [35] and other higher organisms [36]. Many of the motifs are found to have important functions in biological mechanisms. One of the most abundant network motif in *E. coli*, for example, is negative auto-regulation in which a transcription factor(TF) represses its own transcription. These motifs are shown to have functions of response acceleration [37] and increased stability of the auto-regulated gene product concentration against stochastic noise, thus reducing variations in protein levels between cells [38].

In this thesis, study of transmission network analysis focus on developing tools to find out the connectivity-patterns (sub-graphs) are much more often than they do in random networks. They are called "contact tracing motifs". More details would be discussed in section 3.1.4.

1.3 OBJECTIVES OF THE THESIS

The objectives of this thesis are (1) to analyze the social network in the synthetic population used in the FRED agent-based based model and (2) to develop network analysis tools to analyze disease epidemic dynamics in the FRED system. The specific aims of network analysis is to

- Analyze social contact network by finding components, the distribution of degrees and the average shortest path;
- Build transmission networks by using output files of FRED;
- Perform statistical inference on the infection locations to dentify contact tracing motifs.

1.4 OUTLINE OF THE THESIS

In Chapter 2, the social network of synthetic Allegheny County population will be investigated. The format of the file that containing the information of Allegheny County's synthetic population used in FRED will be introduced first, followed by the method of building social network based on the file. Several important network attributes related to social network will then be introduced.

In Chapter 3, tools are developed to analyze the results of FRED simulation, so that we can have more information about how disease is transmitted among synthetic population and how different intervention strategies change the transmission process.

Final discussions are given in the last section.

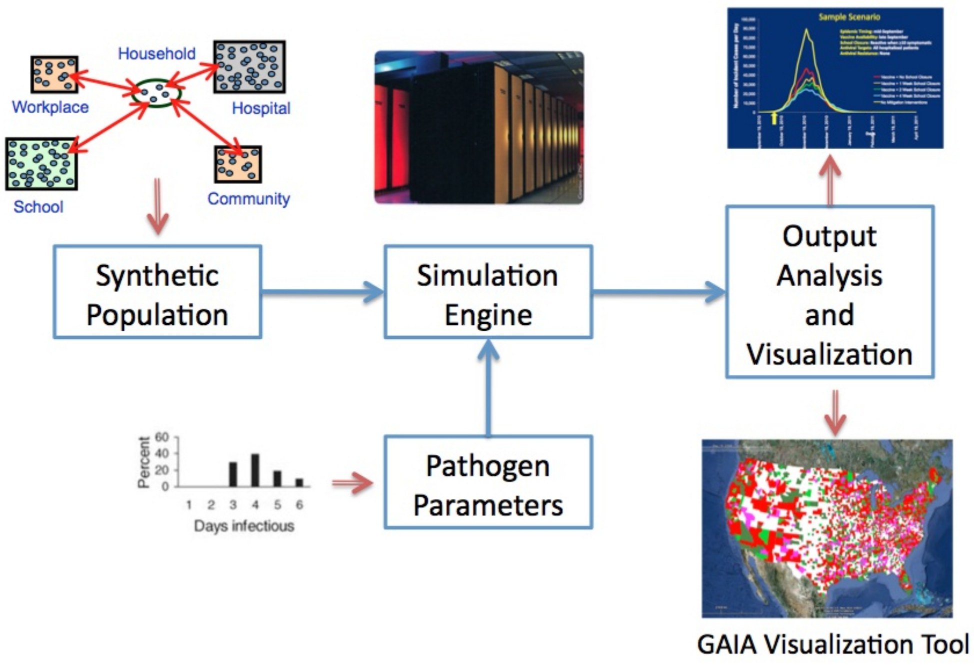


Figure 1: FRED system overview

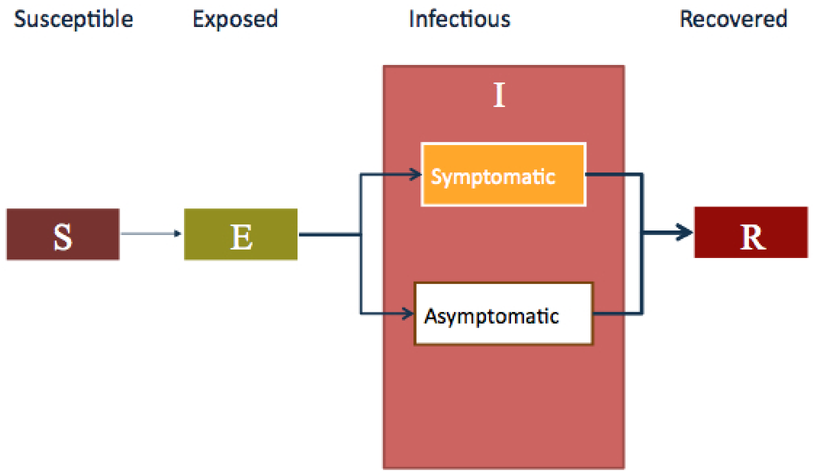


Figure 2: Individual influenza model. The raw figure is in [1] and the usage is under permission of the author.

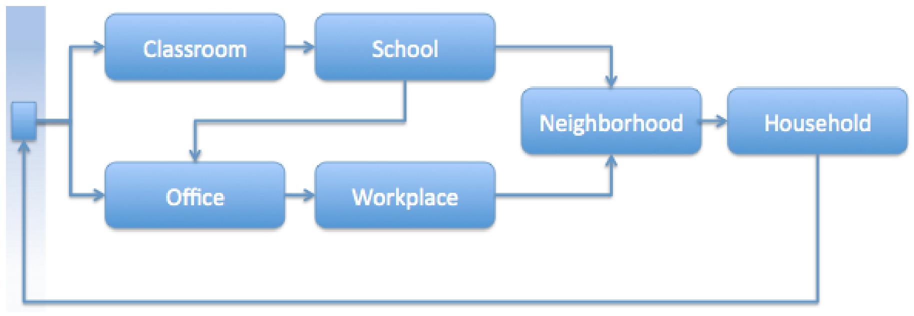


Figure 3: Daily Movement of Agents

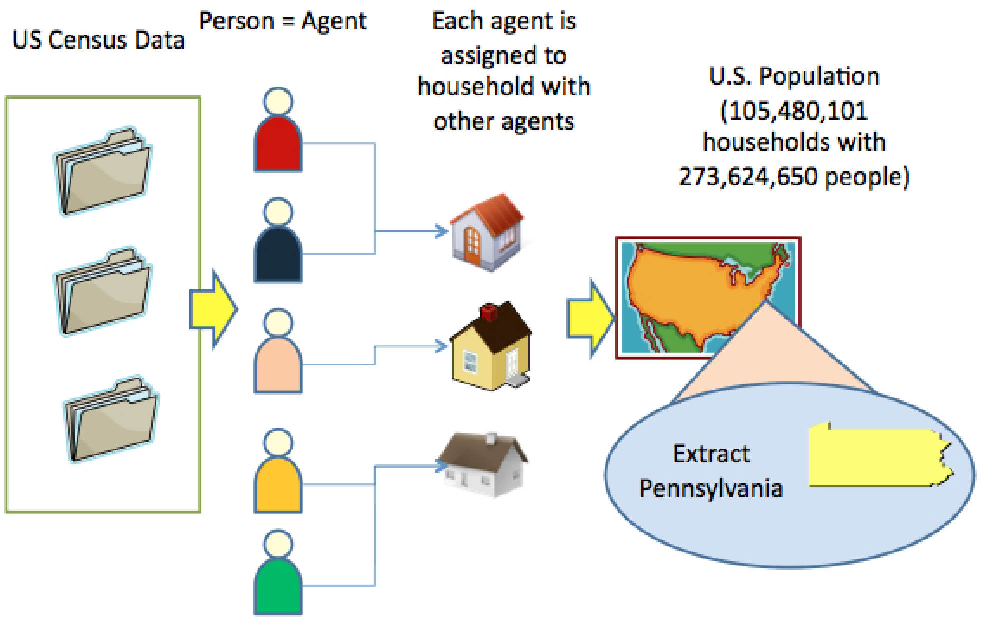

US Census Data
Person = Agent
Each agent is assigned to household with other agents
U.S. Population (105,480,101 households with 273,624,650 people)
Extract Pennsylvania

Figure 4: The construction of synthetic population. The raw figure is in [1] and the usage is under permission of the author.

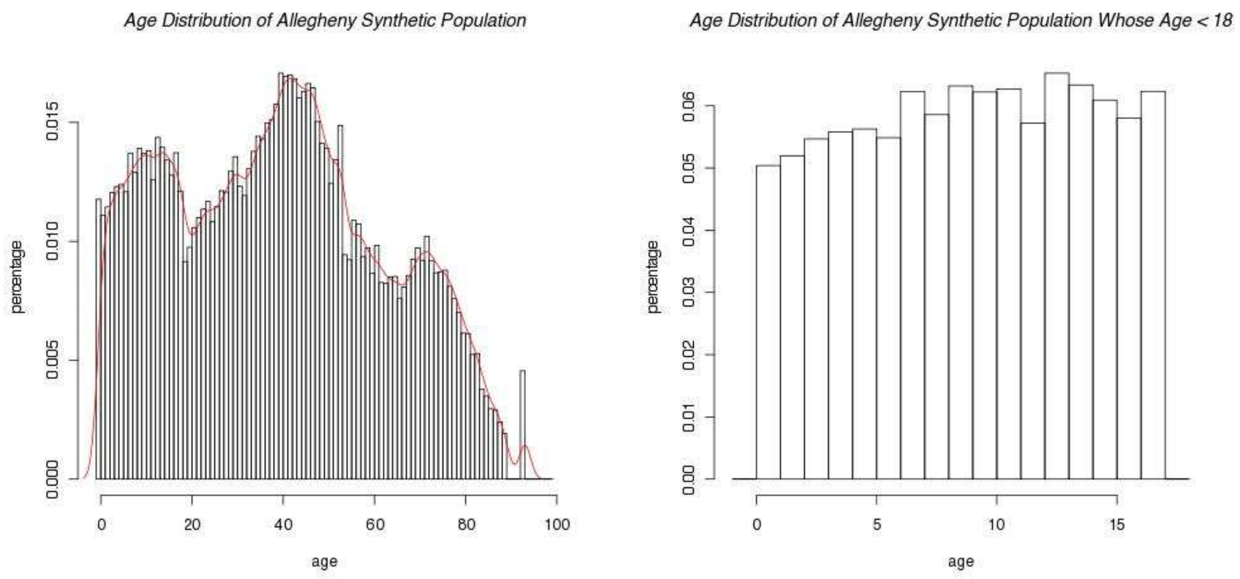


Figure 5: Age distribution of synthetic population

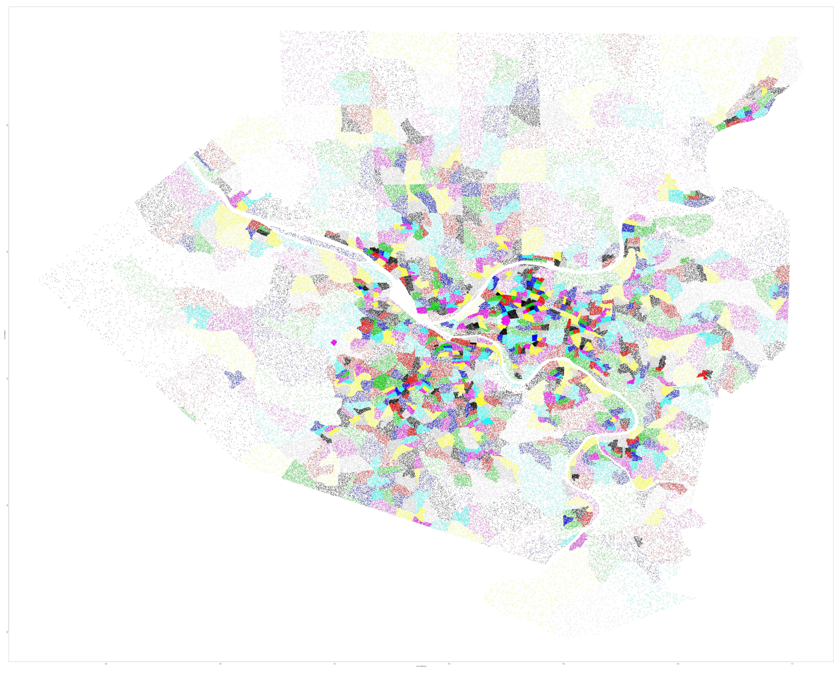


Figure 6: Geographical information of synthetic locations

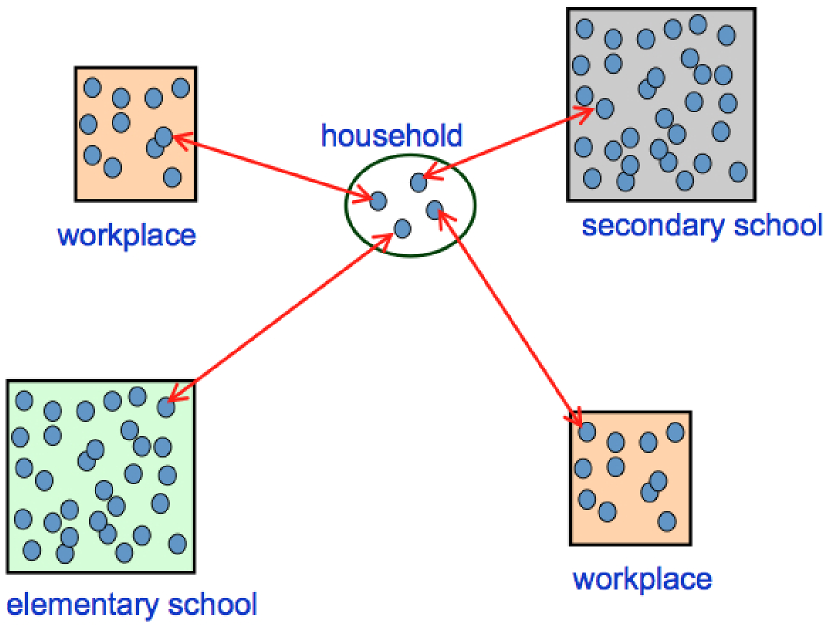


Figure 7: Social networks of synthetic population

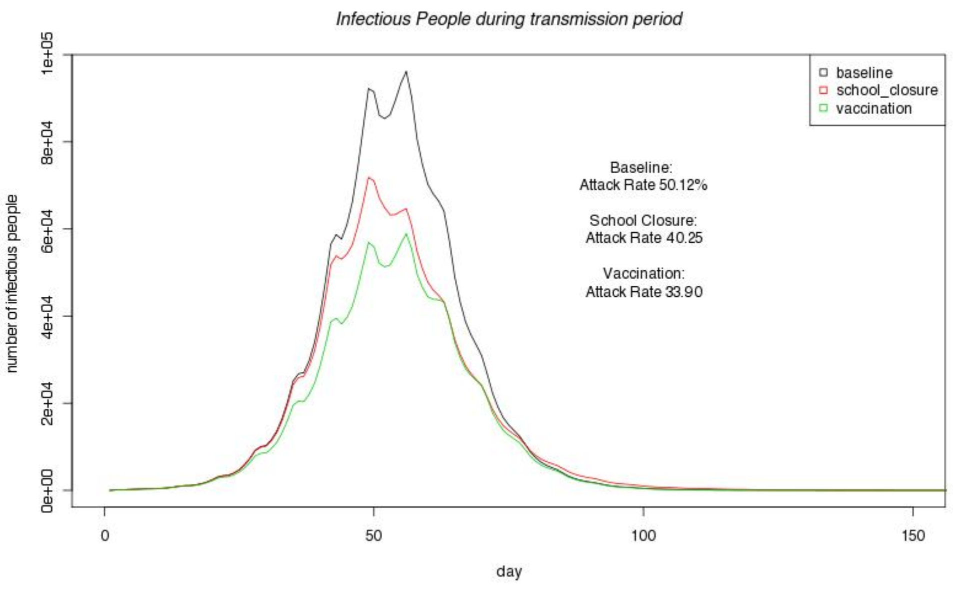


Figure 8: Epidemic curves of infectious agents under three scenarios

Table 1: Definition and application of measurements in network analysis

Properties	**Definitions**	**Applications**
Components	A component is a subset of the nodes in a network in which each node is reachable from others in the same component but not reachable from other nodes in other components.	The severity of disease spread is limited by the number of nodes in the component at the early stage of outbreak. Isolated component will not be affected by the disease if no one is infectious in the component.
Degrees	the number of neighbors that a node is connected to. In directed network (transmission network), the degree has two versions, the incoming degree k^{in} and the outgoing degree k^{out}.	It captures the heterogeneity in individuals' potential to become infected and cause further infection. Intuitively, the higher the number of contacts a node has, the more likely it is to be a neighbor of an infectious node.
Distance	The distance between any pair of nodes $d_{i,j}$ is the minimum number of steps required to reach j from i, that is, the number of steps in the shortest path.	Real networks frequently display the small-world property that is, the vast majority of nodes are reachable in a small number of steps. If it only takes a short number of steps to reach everyone in the component, diseases can rapidly spread.

2.0 SOCIAL NETWORKS FOR EPIDEMICS

In this chapter, the social network of synthetic Allegheny County population will be investigated for the purpose of validation.

In the introduction section, the format of the file that contains the information of Allegheny County's synthetic population used in FRED will be introduced first, followed by the method of building social network based on the file. Several important network attributes related to social network will then be introduced.

In the results and discussion section, the attributes of the social network of synthetic population will be presented and their implication on simulation of diseases will be discussed.

2.1 INTRODUCTION

2.1.1 Format of the Synthetic Population

The synthetic population was generated according to U.S. census data by applying the algorithms designed by RTI. The synthetic population files, namely the population and location files, are first read into FRED to initiate the agents and locations before simulation.

The first line in population file is

```
Population = 1242755
```

which indicates the number of agents in the synthetic population.

The rest of the lines are

```
420034508001_303_1 67 F 0 570 420034508001_303 -1 -1
```

```
420034508001_710_1 67 F 0 570 420034508001_710 -1 -1
420034511011_234_1 67 F 0 570 420034511011_234 -1 4200760460000055
...
```

Columns of each line represent the characteristics (e.g. ID, age, sex, marriage status, occupation code, household, school and workplace ids) of each agents. A location id of -1 indicates that the agent is not assigned to a location of that type.

The synthetic population file will be modified so that it is compatible to be read by our social network analysis tool *Network Workbench* to perform network measurements, such as number and size of components, average shortest path, degrees, etc.

2.1.2 Social Network in FRED

There are over 1.2 million agents in Allegheny County synthetic population, each of whom has several places to go routinely, such as household, neighborhood, workplaces and/or schools, depending on the profession of the agent. Consider the social contact network formed by drawing an edge between two agents if they share a place. A school-age agent may have contacts with its schoolmates, household members and neighborhood friends and a working age agent may have social contacts with its coworkers, household members and neighborhood friends. Thus, an agent would have number of connections as small as 0, if it is isolated from the community, and as big as over a thousand, if it goes to the places that many other people also go. By using this method, a large social network which consists of 1.2 million agents and millions of edges between them is formed.

2.1.3 Components of social network

Components of a graph are sub-graphs that are connected within, but disconnected between sub-graphs. If a graph contains one or more "isolates", namely uncorrected nodes, these nodes themselves are components.

Under the perspective of epidemiology, the infectious disease can only be transmitted within a component if no travel model is implemented. If none of the agents in a component is initially selected as infectious agent, the whole component would be unaffected by the disease. Thus, the knowledge of components in FRED social network would help us predict the spread of disease.

2.1.4 Average Shortest Path

Average path length is one of the measures of network topology, along with its clustering coefficient and its degree distribution. Some examples are: the average number of clicks which will lead you from one website to another, or the number of people you will have to communicate through, on average, to contact a complete stranger. Most real networks have a very short average path length leading to the concept of small world where everyone is connected to everyone else through very short paths.

The average shortest path can help predict the speed of disease spread among a group of people. The less average shortest path of a social network, the faster the disease can spread among the network, because people in the network have fewer steps to be reached by infected people.

2.1.5 Type of Network

Most social, biological, and technological networks display substantial non-trivial topological features, with patterns of connection between their elements that are neither purely regular nor purely random. Such features include a heavy tail in the degree distribution. A few well-known classes of networks are scale-free networks and small-world networks.

A scale-free network is a network whose degree distribution follows a power law, at least asymptotically. That is, the fraction $P(k)$ of nodes in the network having k connections to other nodes goes for large values of k as

$$P(k) = ck^{-\lambda}$$

where c is a normalization constant and λ is a parameter whose value is typically in the range $2 < \lambda < 3$, although occasionally it may lie outside these bounds.

A small-world network is a type of network in which most nodes are not neighbors of one another, but most nodes can be reached from every other by a small number of steps.

Specifically, a small-world network is defined to be a network where the typical distance L between two randomly chosen nodes (the number of steps required) grows proportionally to the logarithm of the number of nodes N in the network, that is:

$$L = c \cdot log(N)$$

where c is a normalization constant. Types of network implicate the way diseases are spread in the network.

2.1.6 Risks of Subgroups

The social network significantly determines the spread of infectious disease in synthetic population of FRED. It is reasonable to believe that the agent with more connections will have more possibilities to be infected, because it has a higher chance to meet an infected agent in its own social network.

Agents are categorized into subgroups by their age. Agents with age of 0 to 4 are preschool children. Agents with age of 5 to 17 are school students; 18 to 55 are adult workers; agents will gradually retire after age 55 and become retired people. Agents at different ages have different places to go, so the social network of school students and adult workers are very different from each other in terms of number of connections. So it is expected that the groups with higher average number of connections would have higher attack rate.

In the following section, the Allegheny County's synthetic population will be used to build a social contact network. The properties of the network, e.g. components, average shortest path, and the types of networks are analyzed to give a snapshot of the synthetic population.

2.2 RESULTS AND DISCUSSIONS

Components of Network

If only location types of workplaces, schools and households are considered, the social network of Allegheny County synthetic population consists of a giant component, which consists of the majority of population, and many small components, which have very few number of agents. Naturally, most of the small components consist of old retired people, who have no workplace or school to go.

In order to avoid those isolated components, neighborhoods are defined on a grid with 1 km square cells. The agents home neighborhood is the cell in which its household is located. However, an agent may visit another neighborhood in the community in a given day. The decision about where to spend the neighborhood activity period is made independently each day, with the highest probability to visit the home neighborhood, and a lesser probability to visit one of the surrounding neighborhoods, and a small probability of visiting a randomly selected neighborhood within a given community radius. After taking neighborhoods into consideration, there is only one component, which consists of all agents in the synthetic population. The neighborhood mechanism makes the simulation more realistic, because the synthetic population can only model the locations that have physical structure, and some non-physical social connection between people can be modeled by neighborhood mechanism, so that retired people can have contacts to others in the society and have possibility to get infected and spread the disease.

Average Shortest Path

The average shortest path is 6.91, which means the average shortest path between any two agents is approximately seven steps. The disease is very easy to be transmitted because it takes very few steps from initial infected agents to all other agents in the society. For a small-world network with 1,242,755 agents, the estimated average shortest path is $L = c \cdot log(1242755) = 14.03c$.

Degree Distribution

The degree distribution of social network of synthetic population can be shown in different ways. In Figure 9, the degree distribution is shown in linear-linear, log-linear and log-log scale. The figure shows a bifurcating pattern, which is not commonly seen. The reason for this special phenomenon may come from the mechanism of generation of synthetic population, which is out of scope of this thesis. The degree distribution can also be plotted as a histogram as shown in Figure 10. The average number of degree is 2137.6, which means on average an agent may possibly contact other 2,138 agents, which consist of household members, coworkers, schoolmates and neighbors. There are very few agents who have more than 7,000 contacts (2.3%), whose distribution of age is shown in Figure 11.

Risk of Subgroups

Risk of each age subgroups can be measured by the attack rate of the group, which is the number of infected agents in that age group divided by the number of all agents in the age group. Intuitively, the attack rate of age group is positively related to the degree of each age group. The degree distribution of each age group is shown in Figure 12 and descriptive statistics of degree distribution by each group is listed in Table 2.2. Youth group have highest average degree followed by adult group and pre-school children and retired agents have fewer average degree, which all make sense.

Two scenarios of baseline scenario and vaccination scenario are selected to run simulation and for calculation the attack rate of subgroups. The implementation details for two scenarios will be described in section 3.1.8 where we will investigate the transmission network under different scenarios. The attack rates of each subgroup by two scenarios are also listed in Table 2.2. The relationship between subgroups' attack rates and their mean degrees is plotted in Figure 13. The youth group suffers from the highest attack rate and the pre-school children and retired group have much lower values. In baseline scenario, the youth group suffers over 80% attack rate, because the youth group consists of school students, who have more active and intimate interaction. So, a reasonable hypothesis is that the the infection between school students are the propeller of disease transmission. As Figure 14 illustrates, the school students can bring disease back to family and infect households (step 1), e.g. other children and adults in the family. Students can again bring disease back to school and infect schoolmates(1), while adults can bring disease to workplace (step 2). Neighborhood, which consists of multiple households, allows transmissions between households(step 3). In vaccination scenario, all subgroups have lower attack rates when compared to baseline scenario, because agents from all subgroups have access to influenza vaccination, which can reduce the probability getting infected by influenza. The vaccination reduced attack rate of youth group much more than other groups.

The hypothesis of the transmission mechanism and the reason why youth group can benefit much more than other groups, however, cannot be justified based on the descriptive

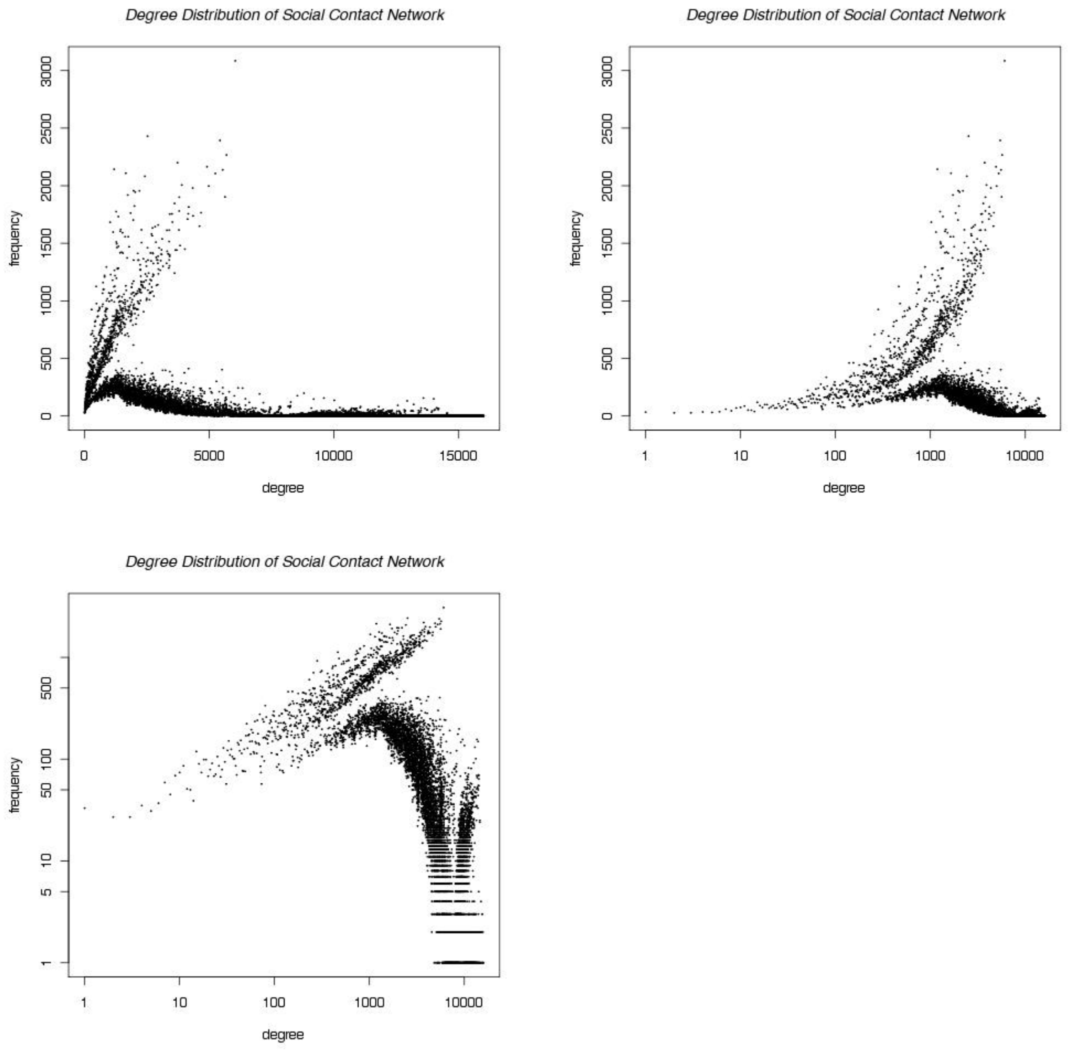


Figure 9: Degree distributions of social network in linear-linear scale (up left), log-linear scale (up right) and log-log scale (down left)

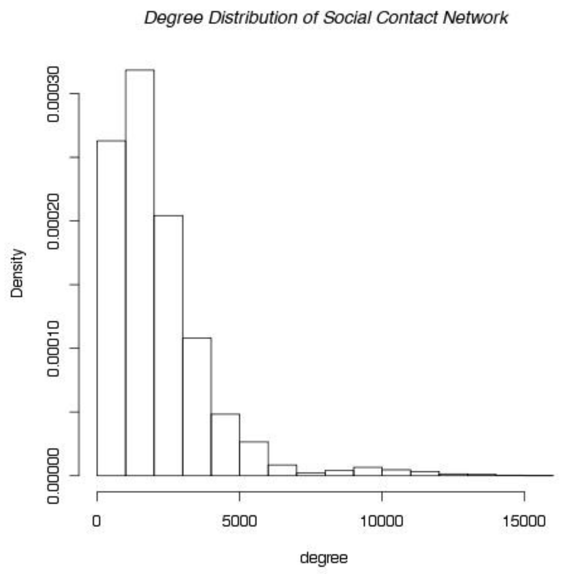


Figure 10: Density of degree of contact for whole population

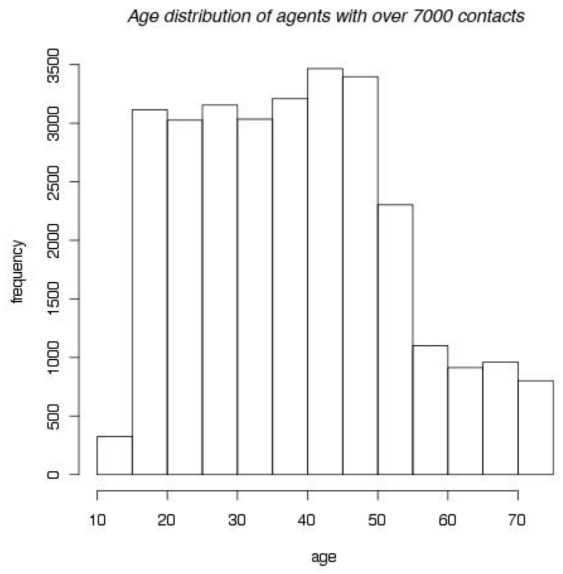


Figure 11: Age distribution for agents with more than 7000 degrees

statistics above. So more comprehensive tools are needed to inquire into the transmission path of disease among all agents, which will be discussed in the next chapter.

2.3 CONCLUSION

Social network analysis of a synthetic population investigates fundamental questions about synthetic population. The synthetic population is shown to be an integrated component with average shortest path is 6.91. The risks of being infected for age groups are positively related to the average degree of each group. Although degree distribution has bifurcating pattern, it is still reasonable to use the synthetic population for modeling disease transmission.

Agents are categorized into four age subgroups and agents in different subgroups have different behavior patterns. Agents with age 0 to 4 are pre-school children and do not need to go to school. Youth group are the agents from 5 to 18 and all have to go to school. After 18, agents leave school and be one of adult worker group, which are aged 18 to 55. They have to go to workplaces if employed and will gradually retire after age 55. So the agents in different age groups have different types of locations to go and possibilities to meet different agents, which may result the different attack rates for subgroups, e.g. the youth group have more than 80% attack rate, twice as much the other groups. The vaccination also have different effect on subgroups, the youth group has the most benefit.

The Transmission mechanism among subgroups, how vaccination prevent agents from disease and why it has different effect, can not be justified based on the descriptive analysis in this chapter.

In the next chapter, transmission network, which comes from the results of disease spread simulations and includes all infected agents and infection events happened between them, will be investigated. Trace file and statistical analysis tools are developed in order to analyze the how disease is spread between schools, households and workplaces.

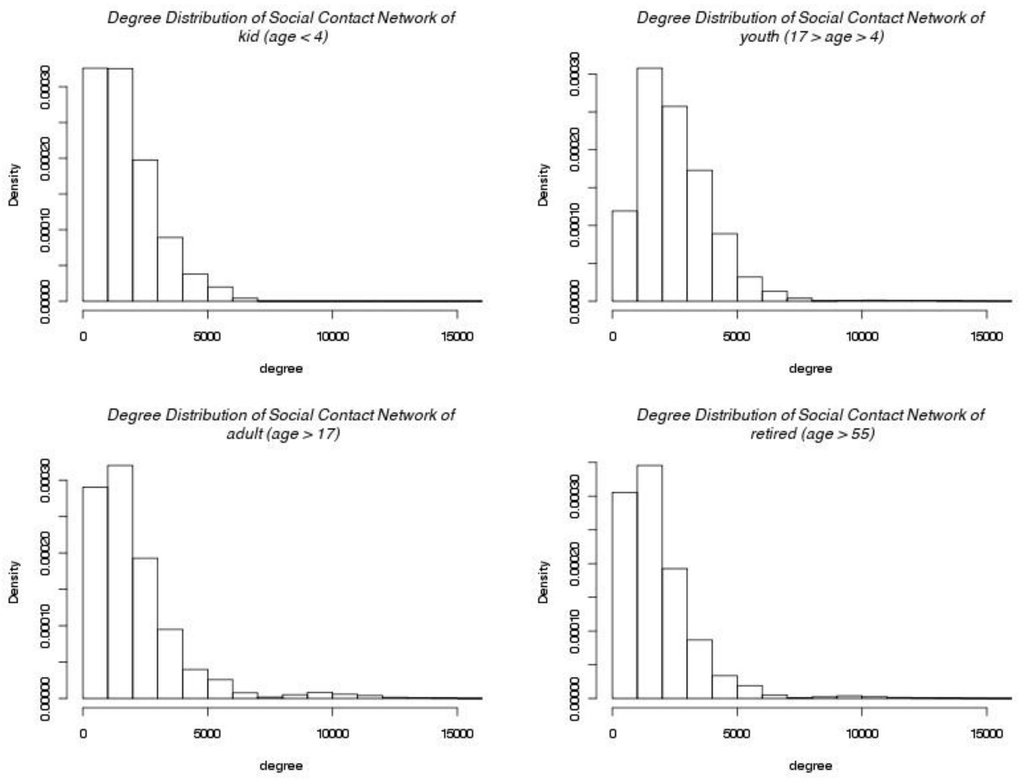


Figure 12: PDF of degree of contact for four age subgroups

Table 2: Descriptive statistics of degrees by each subgroup and the whole population (AR1: averaged attack rate for baseline scenario; AR2: averaged attack rate for vaccination scenario)

Subgroup	**Mean**	**Median**	**Standard Deviation**	**AR1**	**AR2**
Whole Population	2138	1686	1810	0.500	0.304
Pre-school Children	1752	1444	1255	0.363	0.177
Youth	2513	2248	1484	0.858	0.536
Adults	2082	1560	1898	0.430	0.261
Retired	1847	1458	1526	0.241	0.197

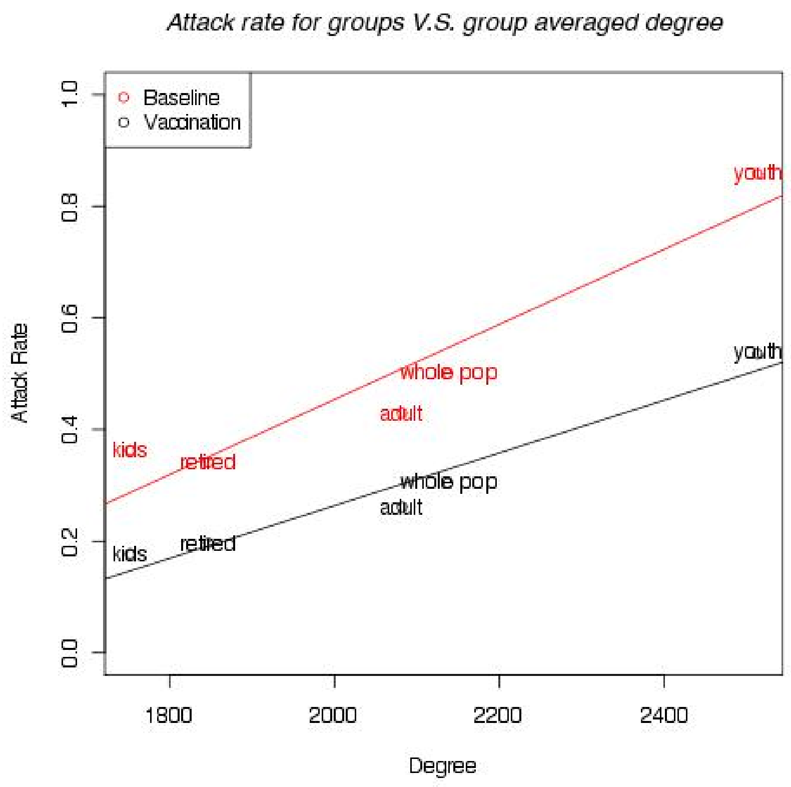


Figure 13: Attack rates for groups vs mean degrees. ”Kids” in the figure represents ”pre-school children”

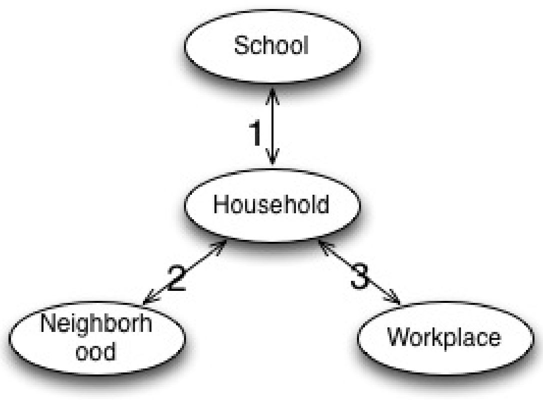


Figure 14: The interaction between different types of locations. In baseline scenario, the youth group suffers over 80% attack rate, because the youth group consists of school students, who have more active and intimate interactions. So a reasonable hypothesis is that the infection between school students are the propeller of disease transmission.

3.0 TRANSMISSION NETWORK OF EPIDEMICS

3.1 INTRODUCTION

In this chapter, tools are developed to analyze the results of FRED simulation, so that we can have more information about how disease is transmitted among synthetic population and how different intervention strategies change the transmission process. In particular, we will apply three different intervention strategies in FRED, and analyze how the transmission network change as a result of each intervention strategy.

In Introduction section, the format of files that were used as input for network analysis will be described and definitions of important terms that are commonly used will be introduced, e.g. trace file, transmission network, type of infection locations, contact tracing motifs, conditional probability, expected value of contact tracing motif, z score, and three simulation scenarios.

In methodology section, the tools for transmission analysis, which are the trace file analysis tool and the statistical analysis tools, and their functions and workflow will be introduced. The transmission analysis tools can help to know in what way the intervention strategies can impact transmission process.

The tools for transmission analysis will be used to analyze the simulation scenarios under three intervention strategies. The results and discussions will at the end of the chapter.

3.1.1 Trace File

Trace files are generated by FRED simulation runs. They contain the demographic and health status records of each agent in FRED system and all infection events for a single run simulation.

It is a text file and each line represent the status of an agent. The format of a line is as follows.

```
Line1: 0 R id 14 a 22 s M 1 exp: 62 inf: 63 rem: 67 places 6   \\
infected_at H 377 infector 16 infectees 2 antivirals: 0
Line2: 0 S id 19733 a 15 s M 5 exp: -1 inf: -1 rem: -1 places 6    \\
infected_at X     -1 infector -1 infectees 0 antivirals:  0
```

Line1 represents an agent whose id is 14; age is 22; a male. He was exposed to disease at day 62, became infectious at day 63 and got recovered at day 67. He was infected by another agent whose id is 16, at location No. 377, which is a household. During infection period, it infected two other agents. After infection period, it became recovered. So the R at the beginning of line represents the final health status of the agent.

Line2 denotes an agent who never got infected during the disease outbreak. So the exposed day, infectious day, and recovery day are all -1, and it got infected at nowhere and its infector does not exist. Therefore, the number of infectees are zero. This kind of agents are called Susceptible.

For each run of the FRED system, we will get a single well formatted trace file. Using trace file, we can build transmission network.

3.1.2 Transmission Network

The transmission network includes all infected individuals. The nodes are infected agents and each individual is connected to its infector by an arrow.

At the beginning of each simulation run, a certain number of agents are randomly selected as infected agents. So they are the only nodes without an incoming arrow in the transmission network. All the other agents who were infected during the outbreak have incoming arrows.

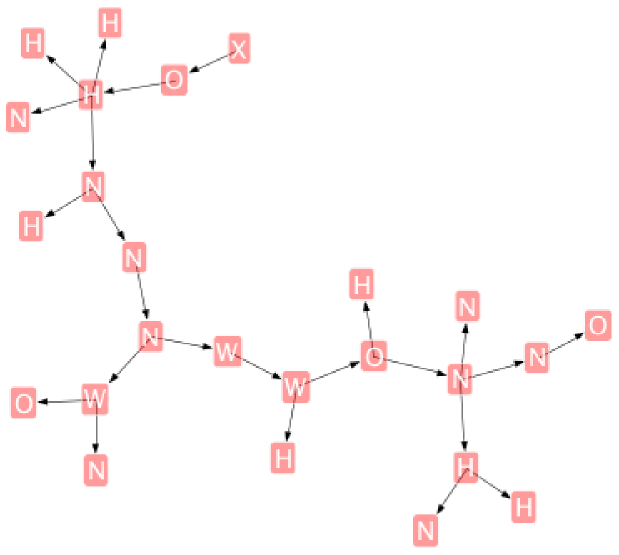


Figure 15: Social Network of a small community, agents are labeled by infection location

For those who got infected but never infected other agents, there will be no outgoing arrow from them. The number of outgoing arrows from an agent is the number of susceptibles it infected. Figure 15 shows an small transmission network. The transmission network starts from one node, and extend as a tree. It will stop expanding until nobody else was infected by the agents who are already in the tree.

3.1.3 Type of Infection Locations

The infection location is where agents got infected. For every infection event, there would be an infection location associated with it. So, for every infected agent, there must be a record to show where it was infected.

There are four regular types of infection locations and one special infection in FRED system. Regular types of infection location are Household(**H**), Neighborhood(**N**), School(**S**), Workplace(**W**). Office (**O**) is contained by workplace and Classroom(**C**) by school, so they are represented by their containers' location type. Symbols in parenthesis are abbreviations.

If an agent was infected at H, it means it was infected by its family members. The special type is X. It is used under two situations. One is for those who never infected. Another is for those who randomly selected as infectious agents at beginning of each simulation run. Figure 15 shows a small transmission network, which can help us understand the disease transmission among different types of locations. The transmission network starts from a node labelled by **X**, which means it was randomly selected infectious agent. And it infected another agent at office. Afterwards, the infectee bring disease to household and infect another one at **H**.

3.1.4 Contact Tracing Motifs

Contact tracing motifs are connectivity-patterns of transmission that occur much more often than they do in random networks. Contact tracing motifs of location types are defined to represent the relationship between the places where infectors got infected and the places where infectors transmitted to others.

- Dicon contact tracing motifs: Dicon means two contacts. If an agent was infected at a type **A** location and it transmitted to another agent at a type **B** location, then it is called a dicon contact tracing motif **AB** where **A** and **B** can be any of {**H,N,S,W**}. Number of **AB** is represented by $N(AB)$. There are 16 dicon contact tracing motifs.
- Tricon contact tracing motifs: Tricon means three contacts. It is defined the same way as dicon contact tracing motifs, represented by **ABC**, where **A, B, C** can be any of {**H,N,S,W**}. There are 64 tricon contact tracing motifs.
- Tetcon contact tracing motifs: Tetcon means four contacts. It is defined the same way as tricon contact tracing motifs, represented by **ABCD**, where **A, B, C, D** can be any of {**H,N,S,W**}. There are 256 tetcon contact tracing motifs.

For the transmission network observed in Figure 15, the number of all dicon, tricon and tetcon contact tracing motifs can be counted. For example, some of the contact tracing motifs are counted in Table 3.1.4.

Table 3: Number of selected contact tracing motif in the transmission network observed in Figure 15

dicon Motif	counts	tricon Motif	counts	tetcon Motifs	counts
WH	3	**NWW**	2	**NWWH**	1
NH	2	**WWW**	1	**HNNN**	1
WW	3	**WWH**	2	**HNNW**	1

3.1.5 Conditional Probability

Given that an infectious agent is infected at type **A** place, the conditional probability $P(B|A)$ gives the probability that this agent will transmit disease to another agent in type **B** location, **A, B** can be any one of {**H,N,S,W**}.

For example, $P(B|A)$ represents the probability of one agent being infected in neighborhood, who then infects another agent in household.

$P(H|N)$ can be calculated by formula

$$P(B|A) = \frac{N(AB)}{\sum_{i=1}^{4} N(AX_i)}, \text{where} X_i \in \{H, N, S, W\} \tag{3.1}$$

3.1.6 Expected Value of Contact Tracing Motif

The observed number of contact tracing motifs can be counted by traversing the whole observed transmission network. However, the expected value of each contact tracing motif also can be calculated by using conditional probabilities.

The infection locations for infectees are not random. In this thesis, a null model is defined in which the sequence of contact tracing motifs is Markovian, which means the infection location of an agent is assumed only conditioned on the agent's infector's infection location. Given a transmission chain $X_1, X_2, \cdots, X_n$, X_i represents the type of location,

$$Pr(X_n|X_1, X_2, \cdots, X_{n-1}) = Pr(X_n|X_{n-1}) \tag{3.2}$$

Then the expected value of a dicon contact tracing motif **AB** depends on the number of agents whose infection location is type **A**, and the conditional probability of $P(B|A)$. We define the expected value of **AB** as $EV(AB)$, which is short for $EV(N(AB))$. It can be calculated as

$$EV(AB) = N(A) \times P(B|A) \tag{3.3}$$

3.1.7 Z score of Contact Tracing Motif

By comparing the observed number and the expected number of a given contact tracing motif, we can identify contact tracing motifs with unexpected frequency in the observed transmission network.

We run n realizations of the simulations, and each run gives a transmission network. If the observed number of a contact tracing motif is more than its expected value, the contact tracing motif is up represented. Otherwise, it is down represented. Z score is used to quantitatively measure the discrepancy between observed number and its expectation.

The z score for dicon contact tracing motif **AB** is calculated as

$$z_{AB} = \frac{\hat{\mu}_{N(AB)} - \hat{\mu}_{EV(AB)}}{\sqrt{\frac{\hat{\sigma}^2_{N(AB)}}{n} + \frac{\hat{\sigma}^2_{EV(AB)}}{n}}} \tag{3.4}$$

where $\hat{\mu}_{N(AB)}$ and $\hat{\mu}_{EV(AB)}$ are the estimated mean of number and expected number of contact tracing motif **AB** in one simulation scenario. $\hat{\sigma}^2_{N(AB)}$ and $\hat{\sigma}^2_{EV(AB)}$ are the estimated standard deviation of number and expected number of Contact Tracing Motif **AB** in one scenario. n is the number of runs of one simulation scenario. Simulation scenario will be introduced in the following section.

3.1.8 Simulation Scenarios

Because FRED supports multiple mitigation strategies, it can be run with different mitigation strategies that will change the transmission process significantly. The conditional probabilities, contact tracing motifs and overall attack rates would change considerably across different mitigation strategies.

In this thesis, three scenarios with different mitigation settings were used. They are baseline scenario, school closure scenario and vaccination scenario.

- **Baseline Scenario**

  In the baseline scenarios, there will be no intervention strategy no matter how severe the outbreak is. The behavior and daily movement of agents will not be changed by outbreak severity. It is reasonable to believe that the baseline scenario should be the worst scenario with highest overall attack rate and longest outbreak length.

- **School Closure Scenario**

  School is a major place for influenza transmission, since students are more susceptible to influenza and they have intimate contacts with each other. Thus it is reasonable to close schools to reduce the severity of disease.

  In school closure scenario, if the overall attack rate of influenza is over 1 percent, the decision of closing school will be made, so that all schools will be closed for 8 weeks (56 days) after one day delay. Compared to the baseline scenario, the number of agents who got infected in school is supposed to be reduced and so as for the contact tracing motifs that include 'S'.

- **Vaccination Scenario**

  Vaccination is also proved to be an effective way to control disease spread. However, many other issues also come with vaccination. For example, the vaccine may need several doses to have full production; not all agents are willing to receive vaccination; the vaccine may not be effective for some of agents; the protection of the vaccine may be delayed for a period of time and the priority of agents are always debatable when the supply of vaccination is limited and not all can get vaccinated.

  In vaccination scenario, the simplest condition is assumed that the 60% of all agents are willing to accept vaccination; number of dose for vaccine is only one; vaccine is effective for 70% of agents; the delay of efficacy is two weeks (14 days); and it is available unlimited to all agents who are willing to receive vaccination.

3.2 METHODOLOGY

Trace file analysis tool, **TraceAnalysis**, is used to analyze the trace files generated by FRED system, which consist of all transmission information to build transmission network, and to make network analysis. Then statistical analysis tool, **StatisticalAnalysis** is used to do statistical inference on statistics from network analysis. The functions and workflow of all programs will be discussed in this section.

3.2.1 Trace File Analysis Tool

Tool #1: TraceAnalysis

The trace file analysis tools are the combination of a C++ program called **TraceAnalysis** and some running scripts called information extraction scripts. They designed to investigate the epidemic dynamics in the perspective of network analysis.

The following is the description of the trace file analysis tools' components and component interactions. Figure 16 shows step 1 to 5, using program **TraceAnalysis** to analyze single trace file. Figure 17 shows Step 6 to 10, using information extraction scripts to collect mean and standard deviation of all statistics from multiple trace files.

1. Run FRED system by setting different scenarios. For each scenario, the FRED will run multiple times and generate trace file for each run. The trace files which are generated by the same running scenario will be put under the same directory.
2. As illustrated in Figure 16, **TraceAnalysis** uses single trace file as well as the location file as input file. Each line of location file is stored as "Place" objects and each line of trace file is stored as "Agent" objects. Each agent has pointers which point to its infector and its infectees. The transmission network is built here.
3. Dicon, tricon and tetcon contact tracing motifs are found by searching the whole transmission network. The number of contact tracing motifs are stored in "Pattern" Objects.
4. Conditional probabilities and expected value of all contact tracing motifs are then calculated.

5. The results are printed out after all statistics of transmission network have been calculated in separated files. The results of distribution of infection locations, the total number of infections and overall attack rate are stored in “Distribution” file. The values of all conditional probabilities are stored in “Conditional Probability” file. The number and expected number of contact tracing motifs are stored in “Patterns” file. The “Prob” field of each contact tracing motif is the probability of the contact tracing motif. It is calculated by dividing the number of contact tracing motif by the number of all contact tracing motifs of the same type. For example, the “Prob” of contact tracing motif **HH** is calculated by dividing the number of **HH** contact tracing motif by the number of all dicon contact tracing motifs. All those files are stored in the same directory.
6. For those trace files that are generated by FRED with same scenario settings, they can be analyzed by **TraceAnalysis** individually with results being stored in different directories. Information extraction scripts are then used to extract the mean and variance of statistics.
7. The mean and standard deviation of statistics are stored in different output files. For example, the file “AR” stores the mean and standard deviation of overall attack rate of one scenario. The script will go through all directories which store the overall attack rate of each trace file, collect all values of each directory, then calculate the mean and standard deviation and write them into file “AR”.

The software **TraceAnalysis** is able to analyze each individual trace file of one scenario and gather the mean and standard deviation of each statistics across all runs of the scenario. The results of statistics can be used to make comparison between scenarios. The program “statistical analysis tools” is then developed to make comparison of the statistics of different scenarios.

3.2.2 Statistical Analysis Tools

The statistical analysis tools are written in R programming language [39], and consists of two independent programs **EpidemicDynamicPlot** and **StatisticalAnalysis**. They are dedicated to compare statistics across different scenarios and make statistical inferences.

It aims to gather the same statistic from multiple scenarios, visualize and test equality of statistics.

Tool #2: EpidemicDynamicPlot

The usual way to make comparison between scenarios is to compare the overall attack rate and the epidemic dynamic curves. **EpidemicDynamicPlot** is to plot epidemic curves of susceptible (S), exposed (E), infectious (I) and/or recovered (R) of one or several scenarios. On each day of outbreak, the numbers of S, E, I and R keeps the records of the severity of the infectious disease. FRED will keep the records of those information in a separate file for each run. For multiple runs of the same scenario, the numbers of S, E, I and R are different due to randomness. **EpidemicDynamicPlot** will plot the mean and confidence interval of S, E, I and/or R as a function of days of outbreak for selected scenarios.

Tool #3: StatisticalAnalysis

The program **StatisticalAnalysis** aims to compare different scenarios of FRED simulations. It uses the results generated from **TraceAnalysis** to make comparison of statistics across scenarios and to make figures for visualization. It gathers mean and standard deviation of all statistics from different scenarios, plots and compares them.

Figure 18 demonstrates the workflow as follows:

1. Load necessary R libraries and standard functions.
2. Load self-defined functions for reading data from files, manipulating data into specific format, customizing standard function error bar plot and etc.
3. Read global parameters from an independent file. The parameters include the paths to the scenarios' results of **TraceAnalysis**.
4. Plot error bar of mean and confidence interval of number and probability of dicon, tricon and tetcon contact tracing motifs for three scenarios. Store the figures in local directory.
5. Plot error bars of conditional probabilities of three scenarios.
6. Plot error bars of the number and expected number of dicon, tricon and tetcon contact tracing motifs for baseline scenario, school closure scenario and vaccine scenario respectively.
7. Calculate z scores for contact tracing motifs. Sort them and plot them.

The output files of **StatisticalAnalysis** are the figures of all error bar plots. All the figures will be kept in a subdirectory of **StatisticalAnalysis** for further discussion.

3.2.3 Work Flow of Transmission Network Analysis Tools

In previous sections, work flow of individual tools have be introduced. Figure 19 shows the work flow of all the tools and interdependency of components.

For each simulation scenario setting, FRED will run multiple times and get multiple trace files for each scenario. **TraceAnalysis** then analyzes each trace file, calculate statistics of transmission network, store values into files. The information extraction scripts then used to gather mean and standard deviation of all statistics for one scenario, store results for further statistical analysis. After analyzing each scenario individually, **StatisticalAnalysis** then is used to make comparison of statistics across all scenarios.

3.3 RESULTS

In previous chapter, network analysis tools and their workflow were introduced. In this chapter, I will use network analysis tools to analyze the effectiveness of public health policies in FRED system and how the transmission network is changed by implementing these policies.

FRED is able to investigate a wide and flexible range of both pharmaceutical and non pharmaceutical public policies. School closure policy and vaccination policy were chosen to run with FRED. The implementation of those policies are described in chapter 3.1.8. Running FRED with no intervention policy is baseline scenario. It is used to simulate the spread of disease without any intervention strategies. The results of baseline scenario can be used to compare to the results from other scenarios, which implementing certain interventions strategies. Running FRED with school closure policy is called school closure scenario and running with vaccination policy is vaccination scenario. In this example, each scenario is run 20 times.

After running FRED with three scenarios, the trace files for three scenarios are generated. The trace file processing and analysis is then conducted as the workflow shown in Figure 19. **TraceAnalysis** is used to analyze the 20 trace files for each scenario and store the statistics in separate file directories. Then **StatisticalAnalysis** is applied to make comparison of statistics and draw conclusions of the differences between three scenarios.

At the beginning, the commonly used methods such as overall attack rate and dynamic curves are applied to make comparison of scenarios. The overall attack rates as well as their standard deviations and coefficient of variation(CV) of three scenarios are listed in Table 4. Then the results generated by **StatisticalAnalysis** are shown as figures in Figure 22 to Figure 27.

EpidemicDynamicPlot plots the epidemic curves of susceptible(S), exposed(E), infectious(I) and recovered(R) agents in Figure 20 and Figure 21. The axis represents the days of outbreak. The day 0 is the first day of simulation when 100 agents was chosen to be infected by the infectious disease. The numbers of S, E, I and R are calculated at the end of each simulation day. The dot for each day is the estimated mean $\hat{\mu}$ of 20 runs. The error bars are the confident intervals $[\hat{\mu} - 1.96 * \hat{\sigma}, \hat{\mu} + 1.96 * \hat{\sigma}]$. Baseline scenarios are all plotted in black, school closure scenario in red and vaccination scenario in green.

The comparison of statistics from network analysis are shown in the following figures. Figure 22 compares the distribution of types of location and conditional probabilities. Comparison of observed numbers of dicon, tricon and tetcon contact tracing motifs are shown in Figure 23. The tetcon contact tracing motifs are too many to be visualized, so only 20 percent of them with largest differences between scenarios are plotted.

The comparisons above are made between three different scenarios. Figure 24 and 25 investigate the probabilities and the expected probabilities of contact tracing motifs within each of three scenarios. The z scores of dicon and tricon contact tracing motifs are sorted and plotted in Figure 27 and Figure 27.

3.4 DISCUSSIONS

The traditional way to compare simulation scenarios are to compare the overall attack rates and dynamics of infection.

The school closure reduce the overall attack rate by $9.86\%([9.62\% - 10.11\%])$. Vaccination reduce the rate by $20.95\%([20.60\% - 21.3\%])$. And vaccination was more effectiveness than school closure ($p < 0.001$). The variances of overall attack rates are small related to mean according to small values of coefficient of variance in Table 4. So the overall attack rates for each scenario is quite stable.

The epidemic curves of the exposed and infectious agents of school closure and vaccination are significantly lower than the baseline scenario. At the beginning of dynamic curves for baseline and school closure scenarios are almost overlapping, because the trigger for the school closure policy is that the overall attack rate must be higher than 1 percent of all population. So the policy won't be executed until about 30 days of outbreak, when the number of infected agents are in considerably large. The periodicity of curves for infectious agents and exposed agents are due to the behavior difference of weekdays and weekends. The number of newly exposed agents in weekends is lower than that in weekdays, because agents would stay at household or neighborhood during the weekends, the infectivity for the contacts happened in household and neighborhood are lower than the contacts in classroom and workplace.

Thus, the traditional comparison shows that both mitigation strategies significantly mitigate disease outbreak. The results from **TraceAnalysis** enable us to know the difference of contact tracing motifs.

In Figure 22, the distribution of infection locations are similar for baseline scenario and vaccination scenario, since in vaccination scenario, the vaccine is available to all groups of agents. However, the school closure scenario is different from the other two scenarios. The probability that an infection happened in school for school closure scenario is about 8% of all infections, which is much lower than that of baseline scenario (14.6%) and vaccination scenario (12.2%), because school closure policy impacts schools the most significantly. The schools are the main source of infection and they are shut down for eight weeks. So it's reasonable that the probability of being infected in school for school closure scenario is nearly half of that in baseline scenario.

Figure 22 also verify the assumption that the conditional probabilities of infection probability depends on the infector's infection location. For example, if an agent is infected at a household (**H**), it is more likely to transmit another agent in Neighborhood(**N**, $P(N|H) \in [0.38, 0.42]$) and less likely in School(**S**, $P(S|H) \in [0.08, 0.16]$). The conditional probabilities of $P(S|H), P(S|N), P(S|S)$ and $P(S|W)$ for school closure scenario are much lower than that of other two scenarios. However, the conditional probability of $P(H|H), P(N|H)$ and $P(H|N)$ for school closure scenario are higher than that of other scenarios. Because the activities in household and neighborhood are more active during school closure period.

Figure 23 shows the observed number of all contact tracing motifs. It's clear that the vaccination policy can reduce the observed number of all contact tracing motifs, but the school closure policy only reduce the contact tracing motifs with 'S', such as HS, NS, SH, SN, while observed number of the other contact tracing motifs are almost the same as baseline scenario. So as for tricon and tetcon contact tracing motifs. So school closure policy blocks the transmission paths only involve **S**. So it is effective in controlling disease transmission in schools, while the disease spread in other locations are nearly as bad as baseline scenario. The vaccination policy is able to reduce the transmission happened in all locations, reduce observed number of all contact tracing motifs.

Another phenomenon worths mention is that the observed numbers of some tetcon contact tracing motifs related to Workplace **W** and Neighborhood **N** for school closure scenario are higher than baseline scenario. Although the school closure policy reduce overall attack rates, it does not necessarily reduce all contact tracing motifs. An hypothesis for this phenomenon is that the closure of schools increase the interaction between students outside the school. Students can get infected through interactions in neighborhood and make the adults in neighborhood more vulnerable and easier to bring disease to workplace. The verification of this hypothesis is out of the scope of this thesis which requires more deliberately designed tools.

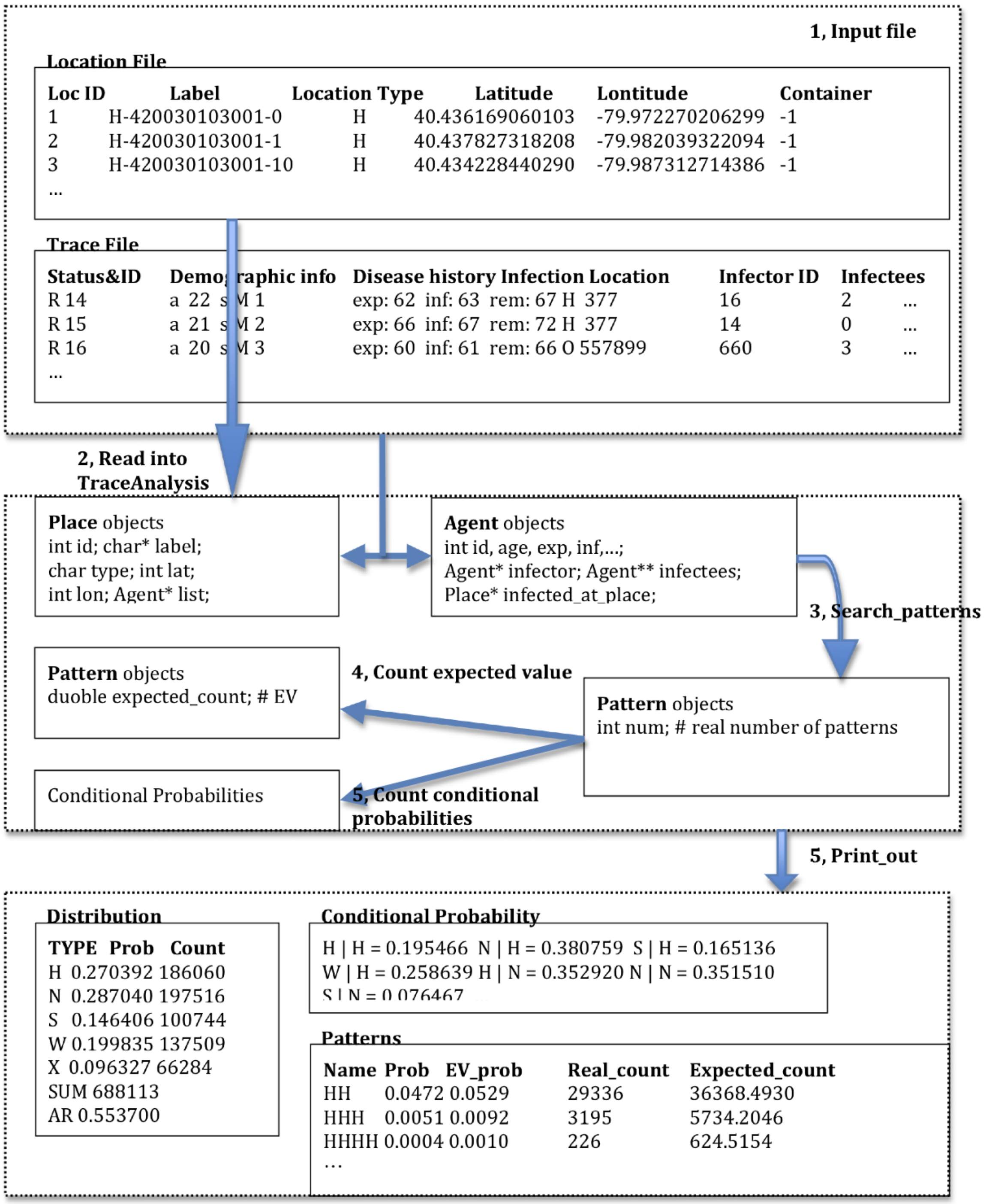
Figure 24 and 25 display the observed number and expected number of each dicon and tricon contact tracing motifs for each scenario. The black dots are observed numbers and red dots are expected numbers. When black dot is higher than corresponding red dot, such as dicon contact tracing motif **SS** in baseline scenario (upper figure in Figure 24), it means the dicon contact tracing motif **SS** is over represented than its expected number. For dicon contact tracing motif **HW** in baseline scenario, the observed number is under represented than its expected number.

Z score is used to quantify the discrepancy (Chapter 3.2.2) between observed number and expected number of contact tracing motifs. The contact tracing motifs with positive z scores are those over represented in transmission process. And those with negative z scores are under represented.

The z score of dicon and tricon contact tracing motifs are sorted and plotted in Figure 27 and 27. The top contact tracing motifs with largest z scores for dicon contact tracing motif and tricon contact tracing motif are listed in Table 5 and 6. Those contact tracing motifs are most over represented contact tracing motifs that may have greatest impact on disease transmission. By observing the figures and lists, a few rules can be concluded as follows:

- Contact tracing motif **SS**, **SN**, **SH** and **SW** are all most over represented contact tracing motifs in baseline and vaccination scenarios. Thus schools play very important role in spreading disease to other locations such as household, neighborhood and workplace, which support the hypothesis in the end of last chapter that the school students are the propeller of disease transmission.
- Contact tracing motif **SS** is not over represented in school closure. Its observed number is very close to its expected number. The school closure policy significantly reduce the interaction between students which results reduction of infection between students.
- All over represented tricon contact tracing motifs in baselines include at least one **S**, which confirms that the schools are the center of disease transmission. So both school closure and vaccination policy can both significantly reduce the overall attack rates and the influence of schools on the disease spread process.

In conclusion, the tools developed in this chapter help to understand how disease is transmitted among synthetic population and how intervention strategies change the transmission process. Three different intervention strategies in FRED, e.g. baseline, school closure and vaccination, are analyzed by tools. Contact tracing motifs of all scenarios, especially those with largest z scores, are examined. The results confirm the hypothesis that the schools play important roles in disease transmission.



Location File

```
Loc ID      Label              Location Type      Latitude          Lontitude          Container
1       H-420030103001-0          H        40.436169060103   -79.972270206299   -1
2       H-420030103001-1          H        40.437827318208   -79.982039322094   -1
3       H-420030103001-10         H        40.434228440290   -79.987312714386   -1
...
```

Trace File

```
Status&ID      Demographic info    Disease history Infection Location       Infector ID    Infectees
R 14           a  22  s M 1        exp: 62  inf: 63  rem: 67 H  377         16             2        ...
R 15           a  21  s M 2        exp: 66  inf: 67  rem: 72 H  377         14             0        ...
R 16           a  20  s M 3        exp: 60  inf: 61  rem: 66 O 557899       660            3        ...
...
```

```
Place objects
int id; char* label;
char type; int lat;
int lon; Agent* list;
```

```
Agent objects
int id, age, exp, inf,...;
Agent* infector; Agent** infectees;
Place* infected_at_place;
```

```
Pattern objects
duoble expected_count; # EV
```

```
Pattern objects
int num; # real number of patterns
```

```
Conditional Probabilities
```

Distribution

```
TYPE  Prob   Count
H  0.270392 186060
N  0.287040 197516
S  0.146406 100744
W 0.199835 137509
X  0.096327 66284
SUM 688113
AR 0.553700
```

Conditional Probability

```
H | H = 0.195466  N | H = 0.380759  S | H = 0.165136
W | H = 0.258639 H | N = 0.352920 N | N = 0.351510
S | N = 0.076467
```

Patterns

```
Name  Prob   EV_prob         Real_count    Expected_count
HH      0.0472 0.0529        29336         36368.4930
HHH     0.0051 0.0092        3195          5734.2046
HHHH 0.0004 0.0010           226           624.5154
…
```

Figure 16: Work flow of **TraceAnalysis**

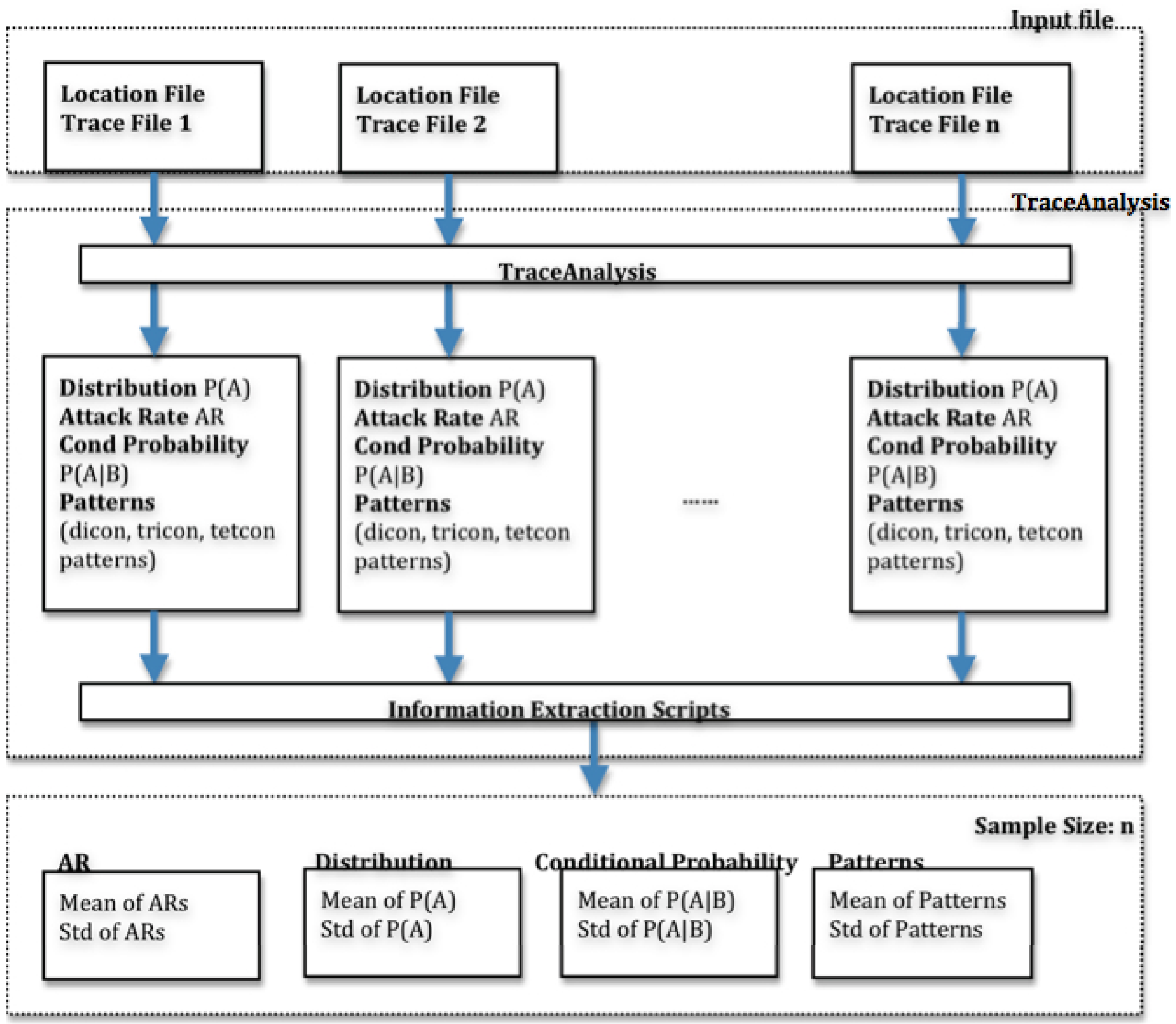
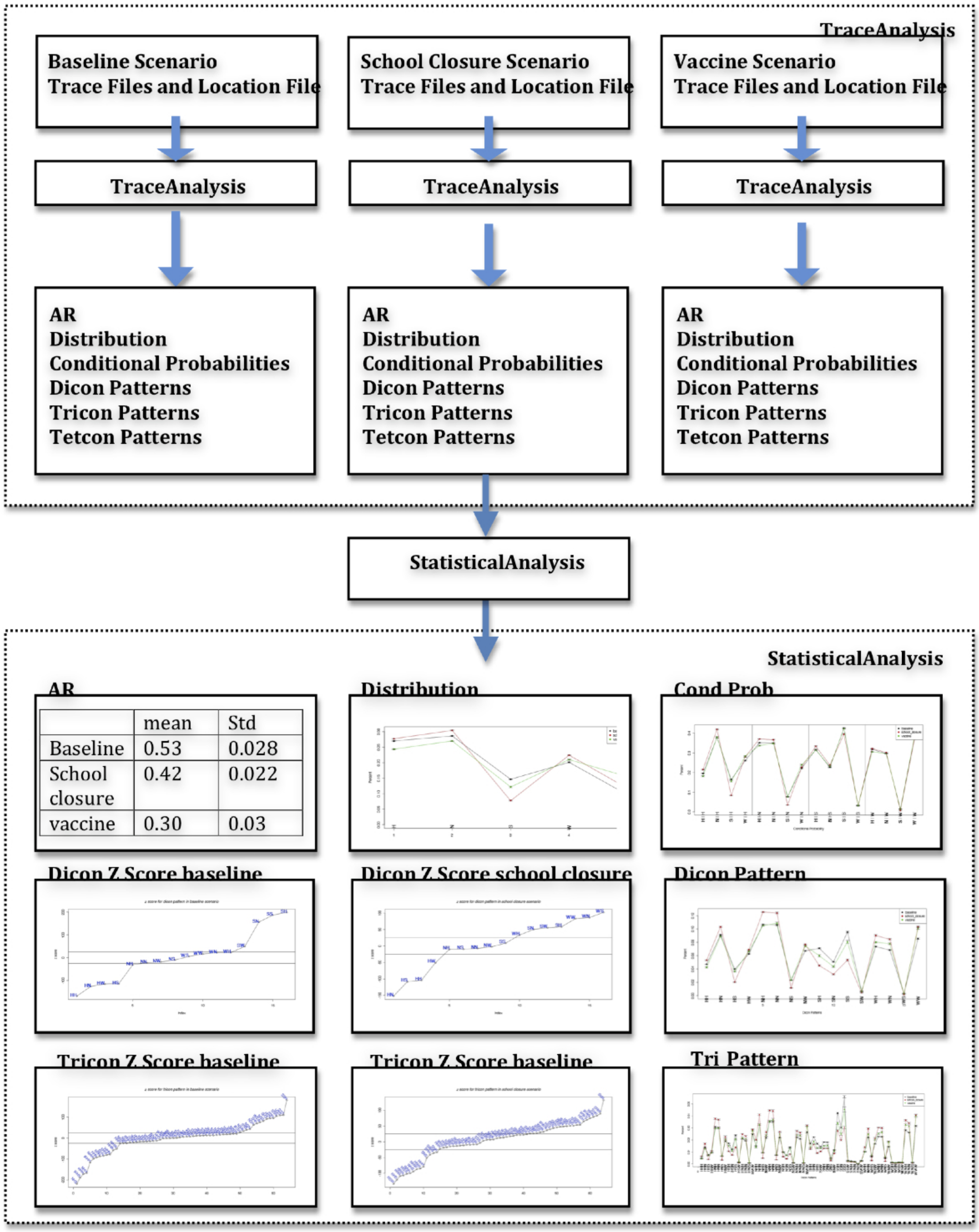


Figure 17: workflow of Information Extraction Scripts



	mean	Std
Baseline	0.53	0.028
School closure	0.42	0.022
vaccine	0.30	0.03

Figure 18: workflow of **StatisticalAnalysis**

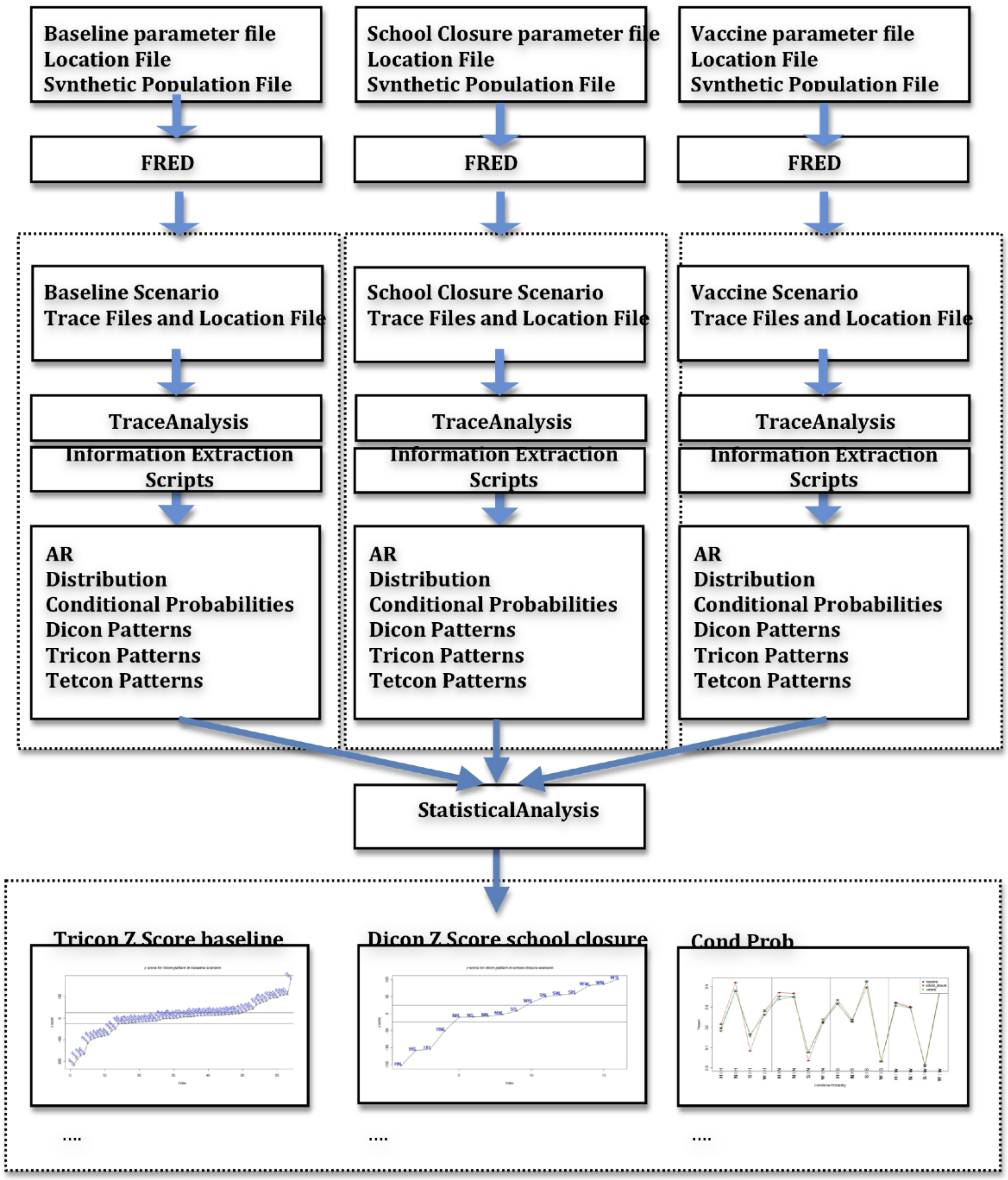
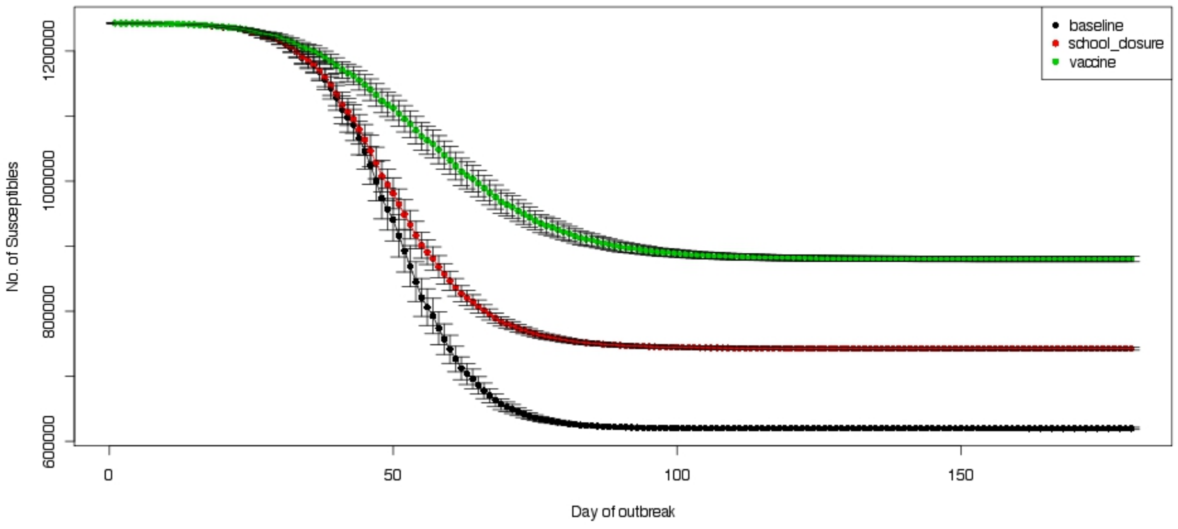


Figure 19: Workflow of All Network Analysis Tools

Table 4: Overall attack rates as well as their standard deviation and coefficient of variation of three scenarios

Scenario	$\hat{\mu}$	$\hat{\sigma}$	CV
baseline	0.53	0.028	0.053
school closure	0.42	0.022	0.052
vaccination	0.30	0.01	0.03



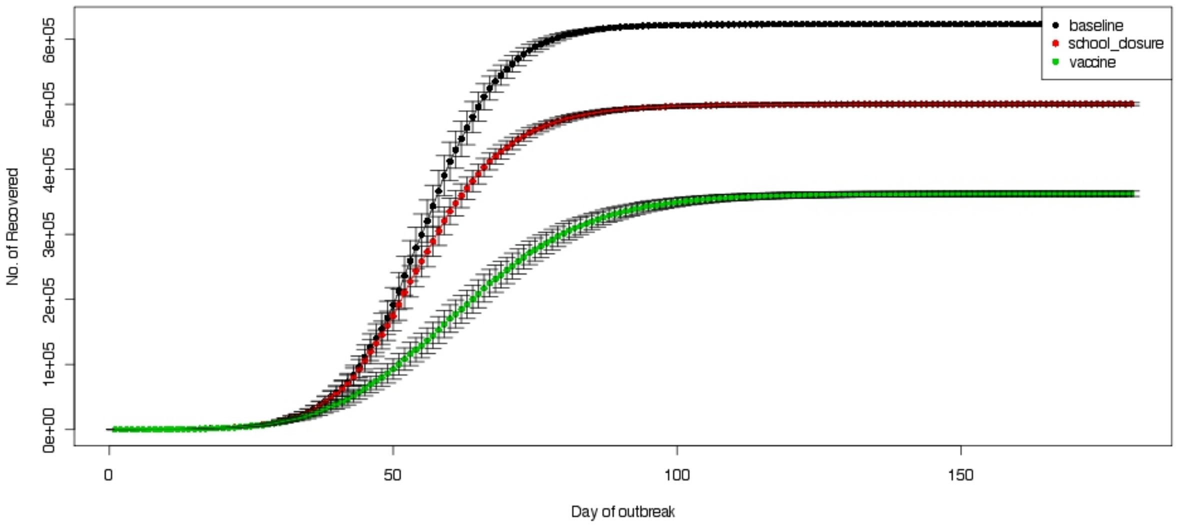
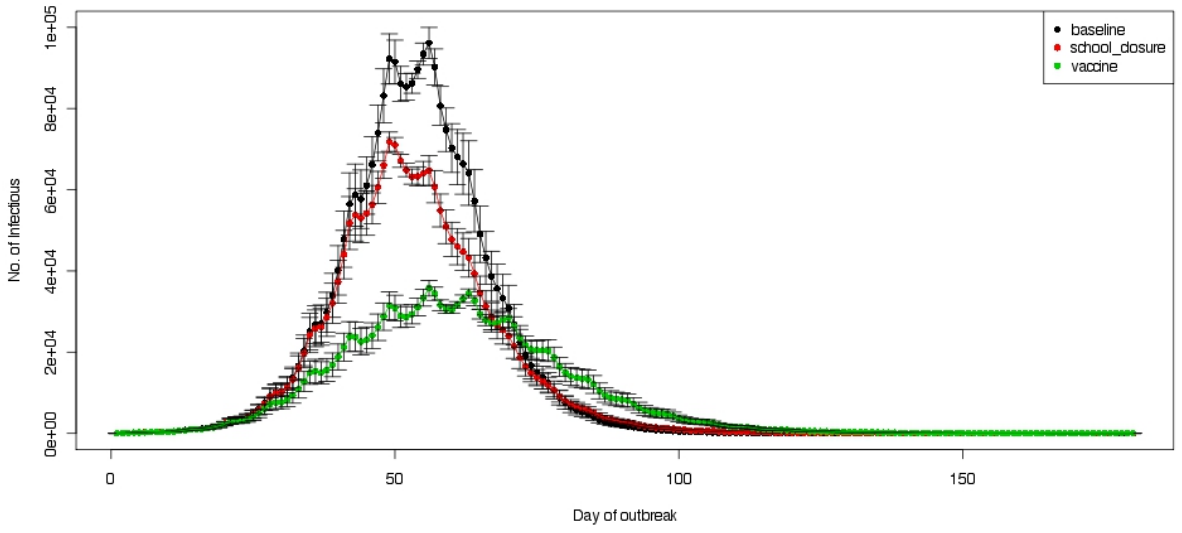


Figure 20: The dynamic of susceptible and recovered agents for three scenarios as well as their standard deviation



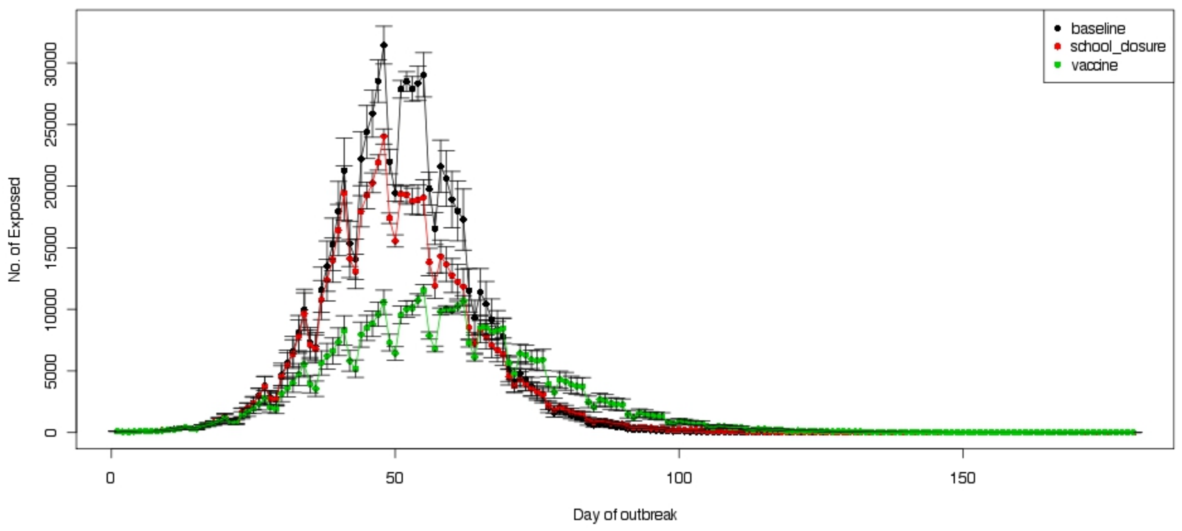
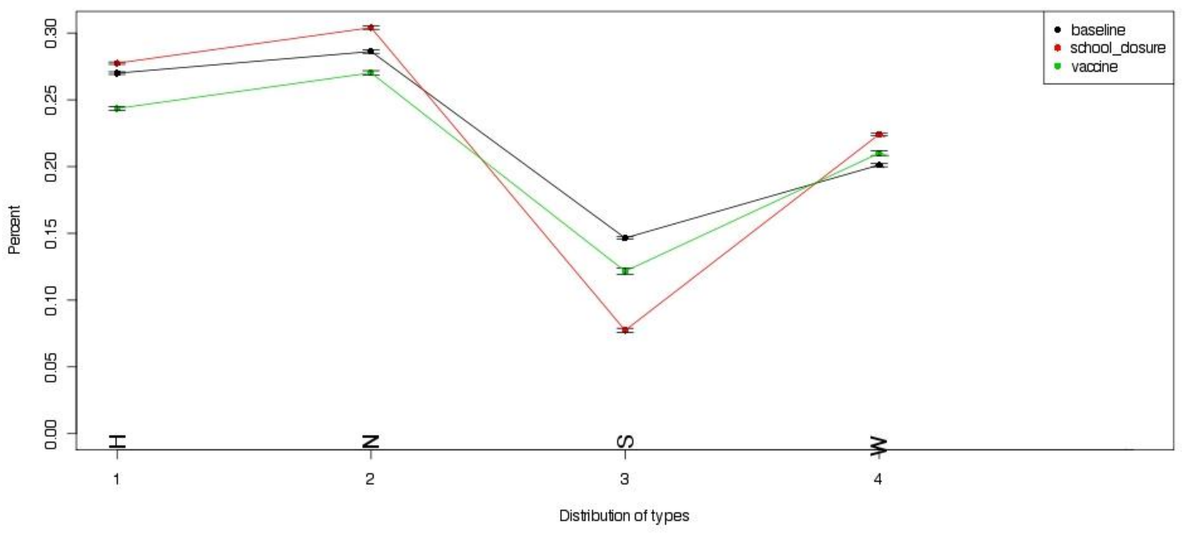


Figure 21: The dynamic of infectious and exposed agents for three scenarios as well as their standard deviation



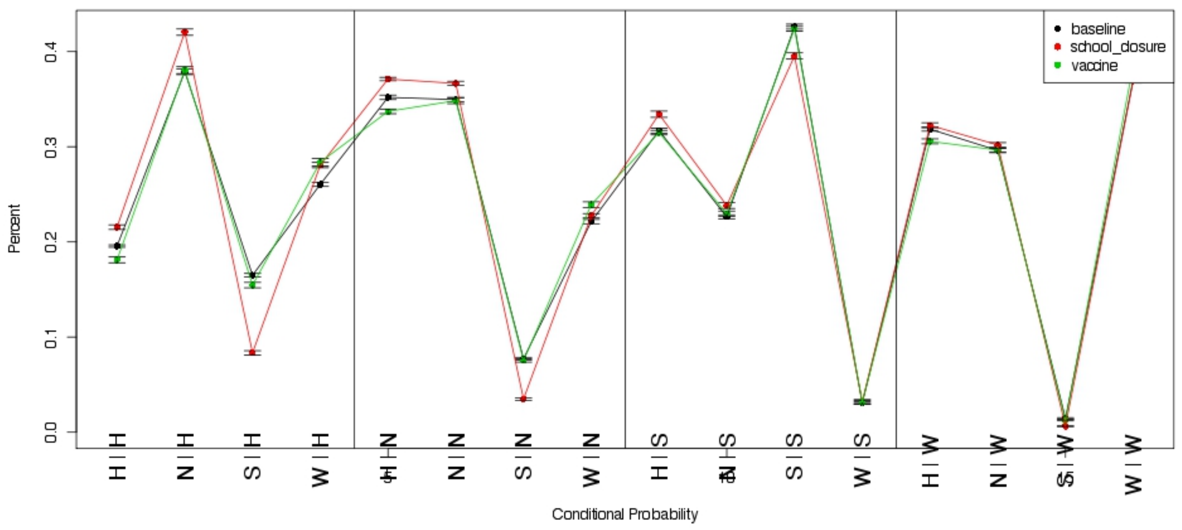


Figure 22: The upper figure is the comparison of distribution of types of location under three scenarios. The black line represent baseline scenario, red school closure scenario and green vaccination scenario. The dots of the line are the percent of agent infected in corresponding type of location.

The lower figure is conditional probabilities of three scenarios. It verifies the assumption that agents are not randomly infected in locations. For example, if an agent is infected in Household, it is more likely to infect another one in Neighborhood and less likely in School.

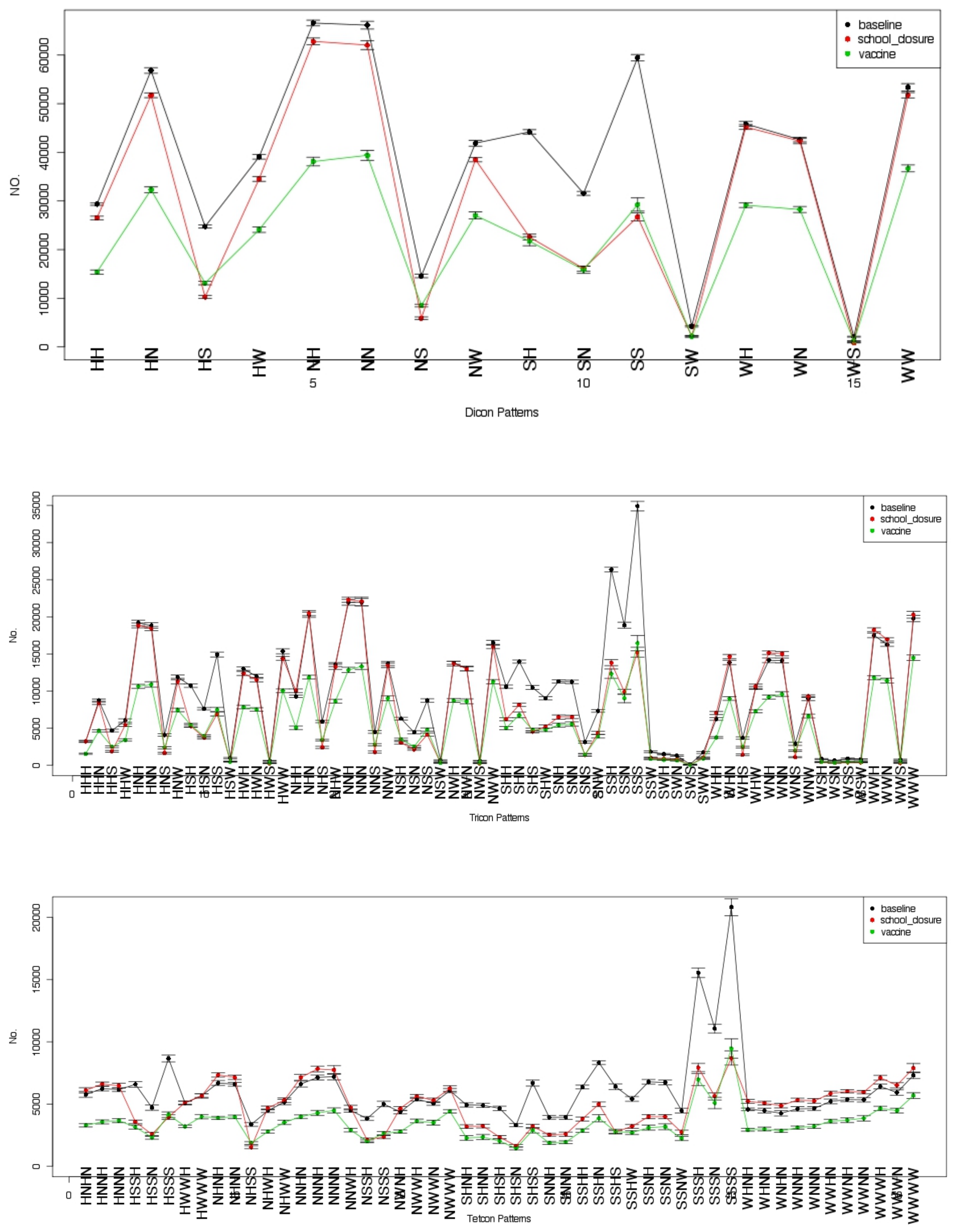


Figure 23: It shows the observed number of all contact tracing motifs. Vaccination policy can reduce the observed number of all contact tracing motifs, but the school closure policy only reduce the contact tracing motifs with 'S'.

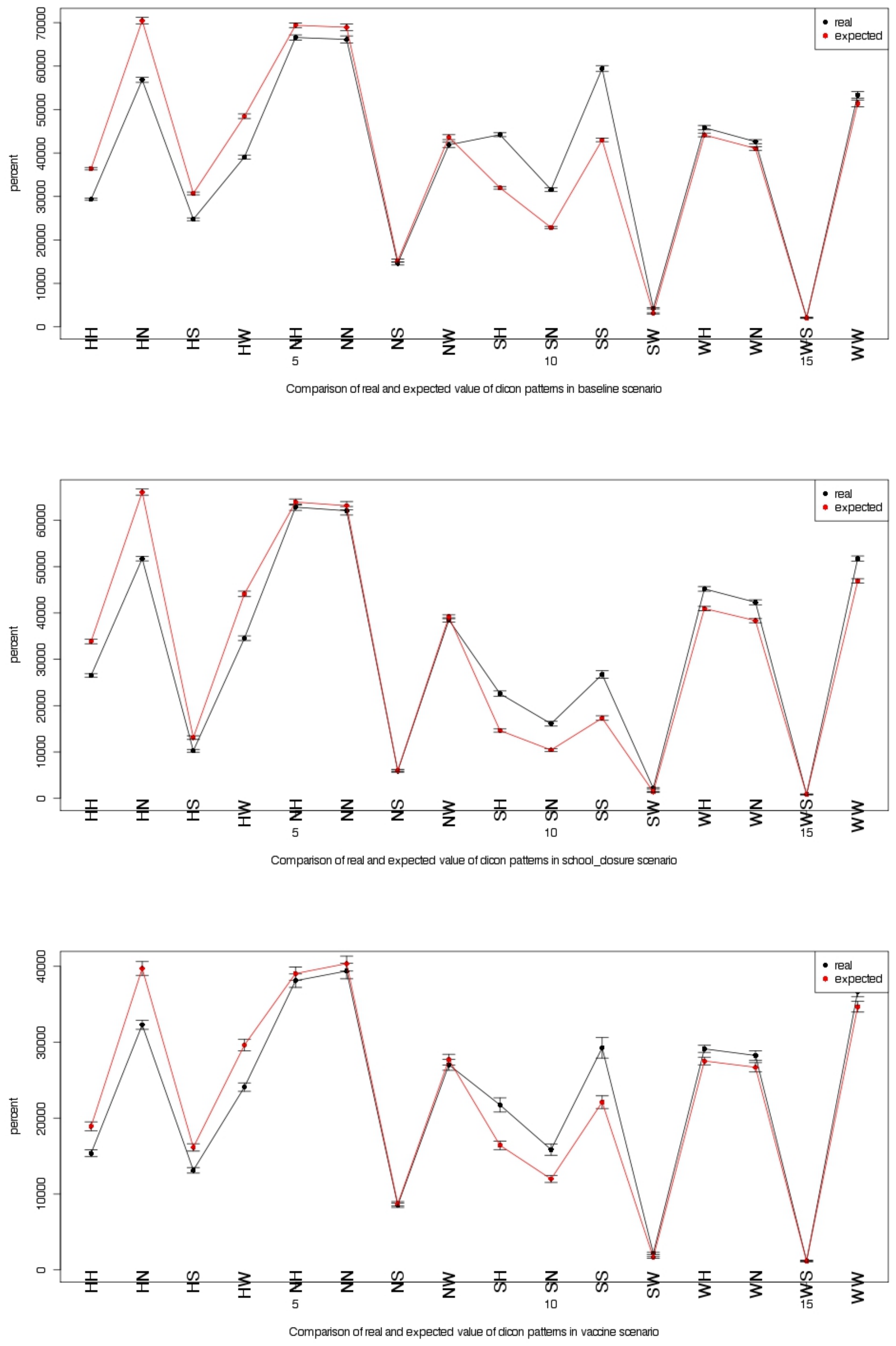


Figure 24: The comparison between observed number and expected number of dicon contact tracing motifs of three scenarios. The black dots are observed numbers and red dots are expected numbers.

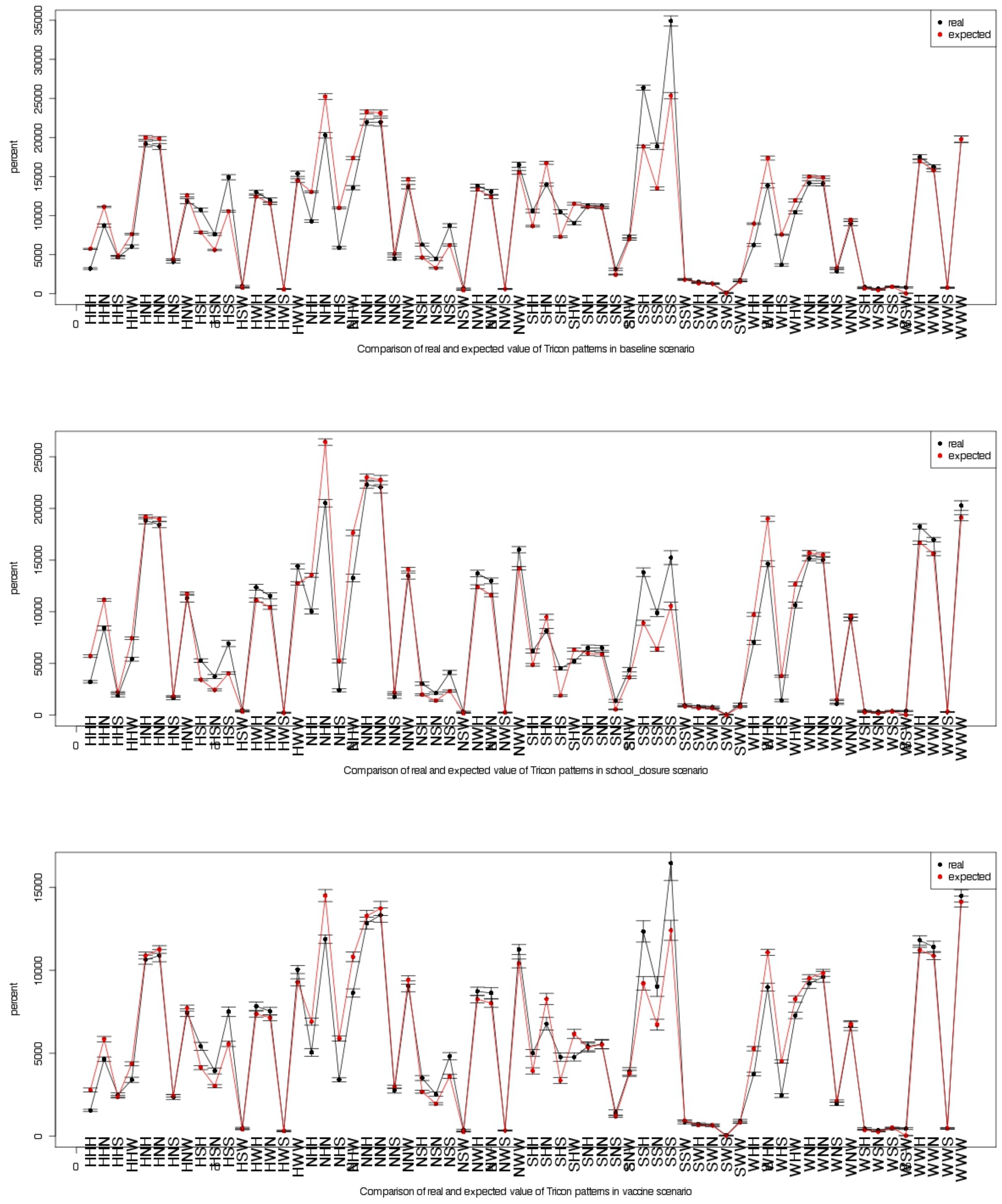


Figure 25: The comparison between observed number and expected number of tricon contact tracing motifs of three scenarios

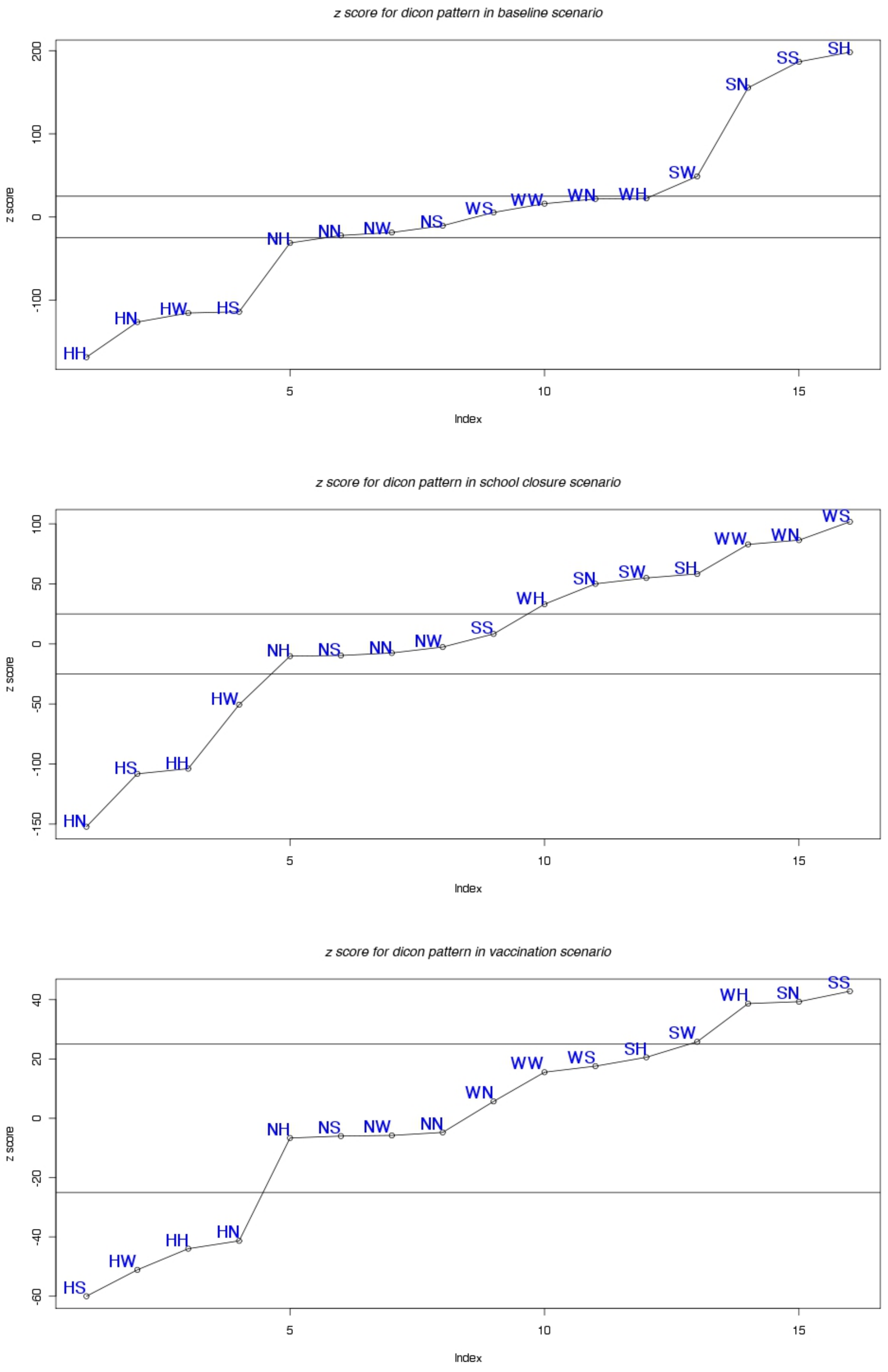


Figure 26: Z scores for dicon contact tracing motifs in three scenarios

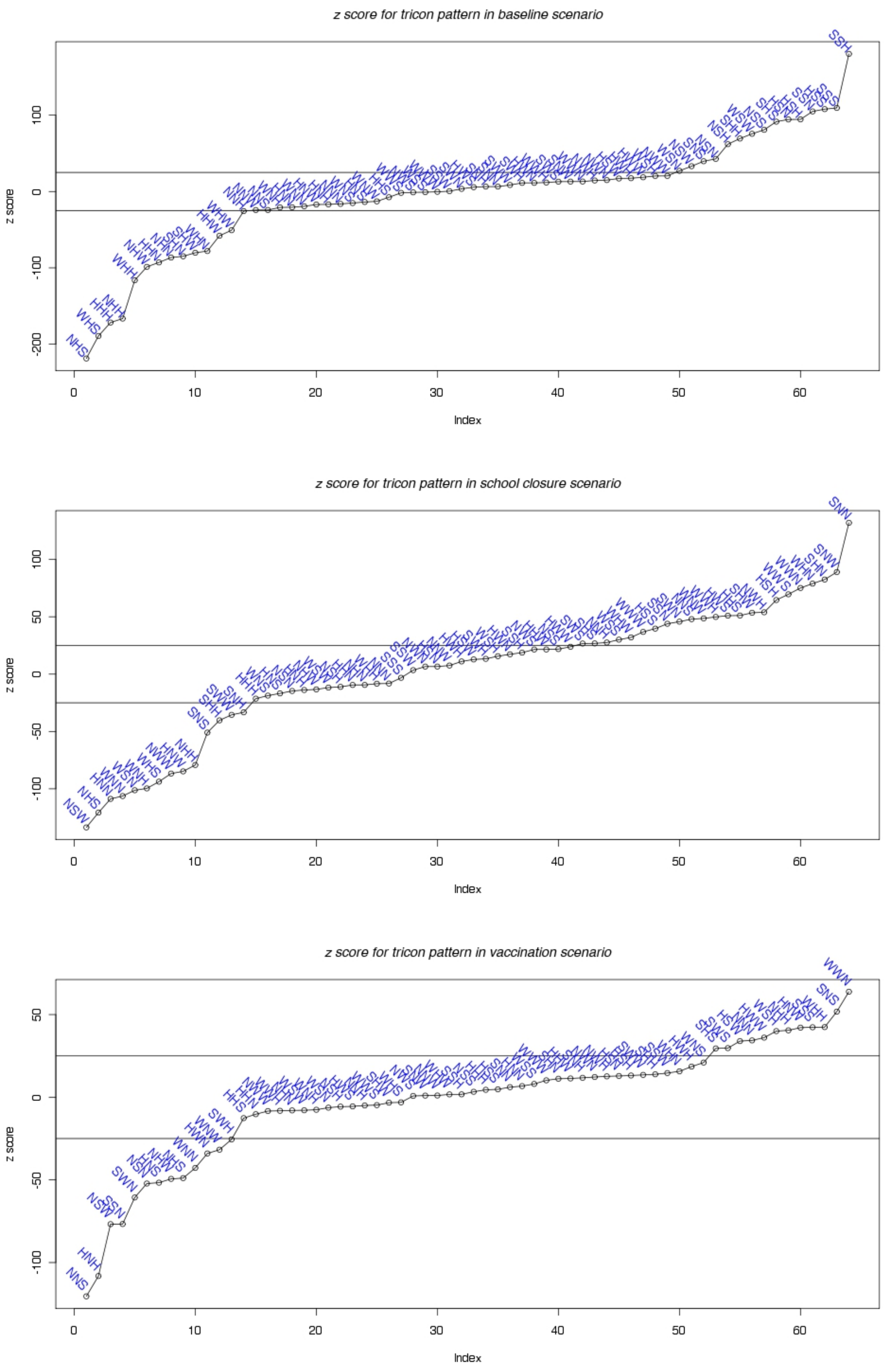


Figure 27: Z scores for tricon contact tracing motifs in three scenarios

Table 5: Dicon contact tracing motifs with largest z scores

	Baseline	School Closure	Vaccination
1	SH	WS	SS
2	SS	WN	SN
3	SN	WW	WH
4	SW	SH	SW
5	WH	SW	SH
6	WN	SN	WS

Table 6: Tricon contact tracing motifs with largest z scores

	Baseline	School Closure	Vaccination
1	SSH	SNN	WWN
2	SSS	SNW	SNS
3	HSS	SHN	WHH
4	SSN	WHN	SSS
5	HSH	WWN	HNW
6	HSN	WWS	NHH
7	SHS	HSH	WSH
8	NSS	NWH	HWW
9	WSW	SHW	NNN
10	SHH	HHS	HSW
11	NSH	HWS	SWS
12	NSN	WWH	SHS
13	SNS	WNW	WNS

4.0 FINAL DISCUSSIONS

Network analysis can be easily applied into the epidemiological studies, especially the spread of infectious disease. Because the connections between individuals that make disease spread happen naturally define a network. The network that is generated by contact tracing or infection relationship allows us to investigate the epidemiological dynamics. Particularly, an understanding of the properties of transmission networks can be used to devise intervention strategies to control disease spread. For example, identification of likely Contact Tracing Motifs and hence devising intervention strategies reduce the spread of infection.

The most significant problem for epidemiological studies to incorporate network analysis is the lack of information about tracing all infection events and rebuild the transmission network. Gardy and et al [40] investigate a Turberculosis outbreak in Canada, which involved 36 cases. The transmission network was constructed by means of interviews with all patients and complete genome sequencing of patients. It is costly and time consuming to build a transmission network even for a small network with tens of agents. When the number of individuals involved in network grows, the dense social network among individuals will make it harder to correctly identify the transmission relationships.

The simulations of infectious disease using ABMs offer opportunities to incorporate network analysis into infectious disease studies. The computational simulation will keep all records of infection events, so it is much easier to trace the infection relationships and build the transmission networks. The infected individuals in the transmission network is theoretically unlimited. There are lots of network analysis tools that have been developed and widely used, for example, the Network Workbench and Pajek[1]. They are able to calculate

[1]Many tools and download links can be found in webpage http://en.wikipedia.org/wiki/Social_network_analysis_software

the metrics in the networks such as betweenness, centrality, closeness, degree, average shortest path and etc[2]. But there are not many articles about network analysis of transmission networks and few transmission network analysis tools are available for ABMs.

The first part of the thesis investigated the social contact network of synthetic population and its validity. The synthetic population is reasonable after examining the components of the social network, average shortest path, degree distribution, and risks of subgroups. However, the descriptive analysis in this part can not answer the question of how disease is transmitted among different subgroups and different types of locations and how mitigation interventions changed the transmission of disease, which gives the reasons to develop the tools to analyze the transmission process in the second part.

In the second part of the thesis, network analysis tools for analyzing transmission processes in ABMs of infectious disease as well as information extraction scripts and tools were explained.

The analysis focused on the types of infection locations aims at finding special Contact Tracing Motifs and comparing Contact Tracing Motifs between simulation scenarios. **TraceAnalysis** is used to build transmission network from text-based trace files, then to calculate various statistics of infection locations from transmission networks. **EpidemicDynamicPlot** is developed to plot the epidemic curves of susceptible, exposed, infectious and recovered agents during transmission. **StatisticalAnalysis** performs statistical inference on the statistics of infection locations and contact tracing motifs. The results from tools allow us to have a better understanding of the transmission processes. They can provide guidance on making intervention policies.

The current tools are able to investigate the infection locations of transmission network and find out special contact tracing motifs with large z scores in FRED system. For future work, more work need to be done to investigate those special contact tracing motifs, their roles in disease spread process and how to reduce number of contact tracing motifs. Statistical analysis of other attributes of agents other than infection locations can be easily performed using the framework of **TraceAnalysis**. Analysis of attributes like age, sex, day of being

[2]The definitions and formulas for metrics can be found in webpage http://en.wikipedia.org/wiki/Social_network#Social_network_analysis

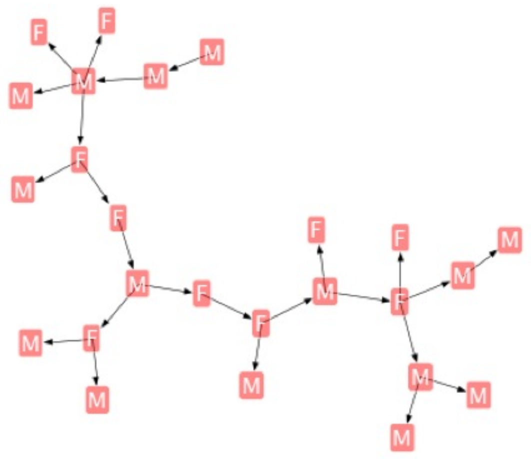


Figure 28: Social Network of a small community, agents are labeled by sex

exposed, becoming infectious and getting recover may reveal more contact tracing motifs. They will be useful for calibration of the model, comparison of simulation scenarios and mitigation policies. For example, Figure 28 gives the same infection network as in Figure 15, but labels are changed to show sex instead of place of infection. contact tracing motifs like **MF** and **MMM** can be defined accordingly and the same analysis methods (e.g. computing z-score and finding motifs) could apply to the age analysis of transmission networks as well.

APPENDIX

A.1 SOURCE CODE OF TRACEANALYSIS

A.1.1 Place Object

```
/*
 * Place.h
 *
 *  Created on: Aug 16, 2010
 *      Author: jih49
 */

#ifndef PLACE_H_
#define PLACE_H_
#include <iostream>
#include <vector>
//#include "Agent.h"
class Agent;
using namespace std;

class Place {
public:
 Place(){ in = 0; out = 0; }
 virtual ~Place(){};
 void Setup(int ID, char* LABEL, char TYPE, double LAT, double LON, int CONTAINER);
 void Add(Agent* agent);
 void Add_containee(Place* place);
 void Set_container();
 void Update();
 void Print_out();
 void Print_socialaction();
 //access functions
 int get_id() { return id;}
 char * get_label() { return label; }
 char get_type() { return type; }
 double get_latitude() { return latitude; }
 double get_longitude() { return longitude; }
 int get_num() { return list.size(); }
 Place* get_container() { return container; }
 int get_in() { return in; }
 int get_out() { return out; }

private:
 int id;      // place id
 char label[32];     // external id
 char type;     // HOME, WORK, SCHOOL, COMMUNITY
 int container_id;     // id of container place
 Place* container;
 vector<Place*> containee;
 double latitude;    // geo location
 double longitude;
 vector<Agent*> list;

//statistics
 int in;
 int out;
};

#endif /* PLACE_H_ */
```

```
/*
 * Place.cpp
 *
 *  Created on: Aug 16, 2010
 *      Author: jih49
 */

#include "Place.h"
#include "Global.h"


void Place::Setup(int ID, char* LABEL, char TYPE, double LAT, double LON, int CONTAINER) {
   id = ID;      // place id
   strcpy(label, LABEL);     // external id
   type = TYPE;     // HOME, WORK, SCHOOL, COMMUNITY
   latitude = LAT;     // geo location
   longitude = LON;
   container_id = CONTAINER;     // id of container place
}

void Place::Set_container(){
 if (-1 == container_id){
  container = this;
 }else{
  container = &pla[container_id];
  if (NULL == container){
   printf("Place[%d]_cannot_find_its_container_Place[%d]\n",id, container_id);
   abort();
  }
  pla[container_id].Add_containee(this);
 }
}

void Place::Add(Agent* agent){
 list.push_back(agent);

 if (get_container() != this){
  get_container()->Add(agent);
 }

```

```
}

void Place::Add_containee(Place * place){
 containee.push_back(place);
}

void Place::Update(){
 vector <Agent *>::iterator itr;
 for (itr = list.begin(); itr < list.end(); itr++){
  Agent* infector = (*itr)->get_Infector();
  if (is_in(infector, list)){
   in++;
   //infector->print_out();
  }else{
   out++;
   //infector->print_out();
  }
 }
}

void Place::Print_out(){
 //Statusfp = fopen("SchoolRecords","w");// if debugging use stdout
 Statusfp = stdout;
 if (0 == containee.size()){
  fprintf(Statusfp,"Place[%d] has no containee\n", id);
 }else{
  if (V){
   if (out != 0){
   fprintf(Statusfp,"Place[%d] has %d containees %d infected",id,
     (int)containee.size(), (int)list.size());
   fprintf(Statusfp, " In %3d, Out %3d, In/Out %5.4f\n", in, out, (double)in/(double)out);
   }
  }

  if (V > 2){
   fprintf(Statusfp,"Place[%d] has containees:\n", id);
   vector<Place*>::iterator itr;
   for (itr = containee.begin(); itr < containee.end(); itr++){
    fprintf(Statusfp,"\tPlace[%d]\n", (*itr)->get_id());
   }
  }

  if (V > 1){
   vector <Place *>::iterator itr;
   for (itr = containee.begin(); itr < containee.end(); itr++){
    fprintf(Statusfp,"\tcontainee %d, %d infected\n", (*itr)->get_id(),
      (*itr)->get_num());
   }
  }

  //print infected's infomation
  if (V > 1){
   vector<Agent*>::iterator itr;
   for (itr = list.begin(); itr < list.end(); itr++){
    (*itr)->print_out();
   }
  }

 }
 //fclose(Statusfp);
}

void Place::Print_socialaction(){
 char nodefile[50];
 sprintf(nodefile, "./socialaction/SocialAction-Node-Place-%d.txt", id);
 Outfp = fopen(nodefile,"w");//if debugging use stdout
 if (NULL == Outfp){
  printf("Help! Cannot open %s\n", nodefile);
  abort();
 }
 //Outfp = stdout;
 vector<Agent*>::iterator itr;
 fprintf(Outfp, "id\tinfector\tinfectee\tinfected_at\tinfected_at_number\texp\n");
 fprintf(Outfp,"INTEGER\tINTEGER\tINTEGER\tSTRING\tINTEGER\tINTEGER\n");
 for (itr = list.begin(); itr < list.end(); itr++){
  //(*itr)->print_socialaction();
  fprintf(Outfp,"%d\t%d\t%d\t%c\t%d\t%d\n", (*itr)->get_id(),
    (*itr)->get_infector(), (*itr)->get_infectees(), (*itr)->get_infected_at(), (*itr)->ge
 }
 fclose(Outfp);

 char edgefile[50];
 sprintf(edgefile,"./socialaction/SocialAction-Edge-Place-%d.txt", id);
 Outfp = fopen(edgefile,"w");
 if (NULL == Outfp){
  printf("Help! Cannot open %s\n", edgefile);
  abort();
 }

 fprintf(Outfp,"infector\tinfectee\n");
 fprintf(Outfp,"INTEGER\tINTEGER\n");
 for (itr = list.begin(); itr < list.end(); itr++){
  Agent* tor = (*itr)->get_Infector();
  if (is_in(tor, list)){
   fprintf(Outfp, "%d\t%d\n", tor->get_id(), (*itr)->get_id());
  }
 }
 fclose(Outfp);
}
```

A.1.2 Agent Object

```
/*
 * Agent.h
 *
 *  Created on: Jun 23, 2010
 *      Author: jih49
 */

#ifndef AGENT_H_
#define AGENT_H_

using namespace std;

class Place;

class Agent {
public:
  Agent();
  ~Agent();

  void setup_agent(int id_, int age_, char sex_, int profession, int exp_,
       int inf_, int rem_, char infected_at_,
       int infected_at_number_, int infecotr_, int infectees_, int antivirals_);
  void setup_place();
  void print_out();
  void print_infectee();
  void print_socialaction();
  void add_infectee(Agent *Pop);
  void setup_infector();
  int Count_des();

  int get_id() {return id;}
  int get_exp() { return exp;}
  int get_infector() {return infector;}
  int get_infected_at_number() {return infected_at_number;}
  int get_infectees() {return infectees;}
  int get_infectee_r() {return infectee_r;}
  int get_des() {return descendents;}
  char get_infected_at() {return infected_at;}
  Place * get_infected_at_place() { return infected_at_place;}
  Agent* get_Infector() {return Infector;}

private:
  int id,age,profession,exp,inf,rem,infected_at_number,infector,infectees,antivirals;
  char sex,infected_at;
  Place * infected_at_place;
  Agent ** infectee; //reference to each infectees
  Agent * Infector;
  int infectee_r; //read number of infectees
  int descendents;
};

#endif /* AGENT_H_ */
```

```
/*
 * Agent.cpp
 *
 *  Created on: Jun 23, 2010
 *      Author: jih49
 */

#include "Agent.h"
#include "Place.h"
#include <stdio.h>

extern Place* pla;
extern Agent** pop_ptr;
extern int place_size;
Agent::Agent(){
 id = -1;
 infector = -1;
 infected_at_number = -1;
 infectees = -1;
 infected_at = 'U';
 age = -1;
 infectee_r = -1;
 infectee = NULL;
 infector = NULL;
 exp = -1;
 inf = -1;
 rem = -1;
 antivirals = -1;
}

Agent::~Agent() {
 delete infectee;
}

void Agent::setup_agent(int id_, int age_, char sex_, int profession_,
      int exp_, int inf_, int rem_, char infected_at_,
  int infected_at_number_, int infector_, int infectees_, int antivirals_){
 id = id_;
 age = age_;
 sex = sex_;
 profession = profession_;
 exp = exp_;
 inf = inf_;
 rem = rem_;
 infected_at = infected_at_;
 infected_at_number = infected_at_number_;
 infector = infector_;
 infectees = infectees_;
 antivirals = antivirals_;
 infectee_r = 0;
 infectee = new Agent *[infectees];

 infected_at_place = &pla[infected_at_number];
 if (NULL == infected_at_place){
  printf("Cannot find Place[%d]\n",infected_at_number);
  abort();
 }
}

```

```
void Agent::setup_place(){
 if (infected_at_number > 0){
  if (infected_at_number < place_size){
   pla[infected_at_number].Add(this);
  }else{
   printf("Place[%d] cannot be found\n",infected_at_number);
  }
 }
}

void Agent::print_out() {
 int container_id = infected_at_place->get_container()->get_id();
 printf("Agent %7d age %3d profession %d infector is %7d infectee is %d
      infected at %c %6d %6d\n",
     id, age, profession, infector, infectees, infected_at,
     infected_at_number, container_id);
}

void Agent::print_socialaction(){
// printf("%d %7d %2d %c %3d\n", id, infector, infectee, infected_at, exp);
 printf("%d %d %d %c %d\n", id, infector, infectees, infected_at, exp);
}

void Agent::print_infectee(){
 if (infectees == 0)
  return;
 if (infectee_r != infectees)
  printf("Not identical\n");

 printf("Agent %d, num = %d\n", id, infectees);
 for (int i = 0; i < infectee_r; i++){
  printf("\tinfectee %d is No. %d\n", i, infectee[i]->get_id());
 }
}

void Agent::add_infectee(Agent *Pop){
 infectee[infectee_r] = Pop;
 infectee_r++;
}

void Agent::setup_infector(){
 Infector = pop_ptr[infector];
 pop_ptr[infector]->add_infectee(this);

}

int Agent::Count_des(){
 if (infectee_r == 0){
  return 1;
 }else {
  int des = 0;
  for (int p = 0; p < infectee_r; p++){
   des += infectee[p]->Count_des();
  }
```

```
    return des+1;
  }
}
```

A.1.3 Function_Location_Pattern Object

```
/*
 * Functions.h
 *
 *  Created on: Jul 26, 2010
 *      Author: jih49
 */

#ifndef FUNCTIONS_H_
#define FUNCTIONS_H_

void ppt(void);
void Place_preparation();
void Agent_preparation(char * filename);
void Search_patterns();
void Count_initial_distribution();
void Set_pattern_prob();
void Count_pattern_expected_prob();
void Chi_square_test();
void Print_out();
void inout();
#endif /* FUNCTIONS_H_ */
```

```
/* Functions.cpp
 *
 *  Created on: Jul 26, 2010
 *
 *      Author: jih49
 */
#include <stdio.h>

#include "Functions_Location_Pattern.h"
#include "Global.h"
#include "iostream"
#include "fstream"
#include "string"

extern Pattern2 pattern2[cat][cat];
extern Pattern3 pattern3[cat][cat][cat];
extern Pattern4 pattern4[cat][cat][cat][cat];


void Place_preparation(){
 FILE *placefile;

  placefile = fopen("alleg_loc.txt","r");

  if (placefile == NULL){
   printf("file alleg loc not found\n");
   exit(1);
  }

  place_size = 0;
  fscanf(placefile, "Locations = %d", &place_size);

  if (V){
   printf("Place_size is %d\n", place_size);
  }
  pla = new (nothrow) Place [place_size];

  for (int loc = 0; loc < place_size; loc++){
   int id;
   char s[32];
   char loctype;
   double lon, lat;
   int container;
   if (fscanf(placefile, "%d %s %c %lf %lf %d",
              &id, s, &loctype, &lat, &lon, &container) != 6) {
    fprintf(stdout, "Help! Read failure for location %d\n", loc);
    abort();
   }
   if (id != loc) {
    fprintf(stdout, "Help! Read index %d for location %d\n", id, loc);
    abort();
   }

   pla[loc].Setup(id,s,loctype,lat,lon,container);
  }

  for (int loc = 0; loc < place_size; loc++){
   pla[loc].Set_container();
  }
  printf("Place all set\n");
}

void Agent_preparation(char * filename){

  ifstream inData;
  inData.open(filename,ios::in);

  if(!inData.is_open()){
   cout << "File don't exist" << "\n";
   abort();
  }

  pop_size = 0;
  string line;
  while (!inData.eof()){
   getline(inData,line);
   if (line != "" && line.substr(2, 1) == "R"){
    pop_size++;
    }
  }
  inData.close();

  FILE *tracefile;

  tracefile = fopen(filename,"r");

  if (tracefile == NULL){
   printf("file %s not found\n", filename);
   exit(1);
  }

  if (V){
   printf("pop-size is %d\n", pop_size);
  }

  pop = new (nothrow) Agent[pop_size];
  if (pop == NULL){
   printf("Help! Pop allocation failure\n");
   abort();
  }

  int max_id = 0;
  for (int p = 0; p < pop_size;){
   int id,age,profession,exp,inf,rem,infected_at_number,infector,
   infector_2,infectees,antivirals;
   char sex,infected_at,status;
   //Input format, 12 variables
   if(fscanf(tracefile, "0 %c id      %d  a  %d  s %c %d exp: %d
      inf: %d  rem: %d places 6 infected_at %c %d infector %d
      infector %d infectees %d antivirals:  %d ",
         &status, &id, &age, &sex, &profession, &exp,
         &inf, &rem, &infected_at, &infected_at_number,
         &infector, &infector_2, &infectees, &antivirals) != 14){
     printf("Help! Read failure for person id = %d\n", id);
     printf("0 %c id %d a %d s %c %d exp: %d  inf: %d  rem: %d places 6
       infected_at %c %d infector %d infector %d infectees %d antivirals:  %d",
         status, id, age, sex, profession, exp, inf, rem, infected_at,
        infected_at_number,infector, infector_2, infectees, antivirals);
    abort();
   }

   if (status == 'R'){
    pop[p].setup_agent(id, age, sex, profession, exp, inf, rem, infected_at,
           infected_at_number, infector, infectees, antivirals);
    p++;
   }

   if (id > max_id){
    max_id = id;//count the maximum id
   }
  }

```

```
 fclose(tracefile);
 if (V){
  printf("max_id = %d\n", max_id);
 }
 //assign an array of pointer
 pop_ptr = new Agent*[max_id];
 for (int p = 0; p < pop_size; p++){
  pop_ptr[pop[p].get_id()] = &pop[p];
 }

 printf("==============================\nIntegrity\n");

 for (int p = 0; p < pop_size; p++){
  if (pop_ptr[pop[p].get_infector()] != NULL){
   pop[p].setup_infector();
  }else{
   if (V && (pop[p].get_infector() != -1)){
     printf("%d cannot find infector %d\n", pop[p].get_id(),pop[p].get_infector());
   }
  }
  pop[p].setup_place();
 }
}


void Search_patterns(){
 //    printf("==============================\nSearching Pattern2\n");
 Agent *temp;

 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
   pattern2[i][j].setup(Categ[i],Categ[j]);
  }
 }

 for (int p = 0; p < pop_size; p++){
  temp = pop[p].get_Infector();
  if (NULL != pop[p].get_Infector()){
   char infected_type = pop[p].get_infected_at();
   char infector_type = temp->get_infected_at();
pattern2[get_type_number(infector_type)][get_type_number(infected_type)].add_pattern(temp,
  }
 }

 //    printf("==============================\nSearching Pattern3\n");

 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
   for (int k = 0; k < cat; k++){
    pattern3[i][j][k].setup(Categ[i],Categ[j],Categ[k]);
   }
  }
 }

```

```

for (int i = 0; i < cat; i++){
 for (int j = 0; j < cat; j++){
  vector<AddressType2*>* relation = pattern2[i][j].get_relation();
  for (vector<AddressType2*>::iterator itr = (*relation).begin();
    itr < (*relation).end();
    itr++){
   temp = (*itr)->get_infector()->get_Infector();
   if (NULL != temp){
    char first_type = temp->get_infected_at();
    Agent* first = temp;
    Agent* second = (*itr)->get_infector();
    Agent* third = (*itr)->get_infectee();
    pattern3[get_type_number(first_type)][i][j].add_pattern(first, second, third);
   }
  }
 }
}

//    printf("==============================\nSearching Pattern4\n");

for (int i = 0; i < cat; i++){
 for (int j = 0; j < cat; j++){
  for (int k = 0; k < cat; k++){
   for (int l = 0; l < cat; l++){
    pattern4[i][j][k][l].setup(Categ[i],Categ[j],Categ[k],Categ[l]);
   }
  }
 }
}


for (int i = 0; i < cat; i++){
 for (int j = 0; j < cat; j++){
  for (int k = 0; k < cat; k++){
   vector<AddressType3*>* relation = pattern3[i][j][k].get_relation();
   for (vector<AddressType3*>::iterator itr = (*relation).begin();
     itr < (*relation).end();
     itr++){
    temp = (*itr)->get_first()->get_Infector();
    if (NULL != temp){
     char first_type = temp->get_infected_at();
     Agent* first = temp;
     Agent* second = (*itr)->get_first();
     Agent* third = (*itr)->get_second();
     Agent* forth = (*itr)->get_third();
     pattern4[get_type_number(first_type)][i][j][k].add_pattern(first,
                      second,
                      third,
                      forth);
    }
   }
  }
 }
```

```
  }
}


void Count_initial_distribution(){
 printf("==============================\nInitial Distribution\n");
 int count[cat];
 for (int i = 0; i < cat; i++){
  count[i] = 0;
 }

 for (int p = 0; p < pop_size; p++){
  count[get_type_number(pop[p].get_infected_at())]++;
 }

 int sum = 0;
 for (int i = 0; i < cat; i++){
  sum += count[i];
 }
 for (int i = 0; i < cat; i++){
  initial_distribution[i] = (double)count[i]/(double)sum;
  if (V){
   printf("%c %f %d\n",Categ[i], initial_distribution[i], count[i]);
  }
 }
 printf("SUM %d\n",sum);
}

void Set_pattern_prob(){
 //    printf("==============================\nSet Pattern Prob\n");
 sum_2 = 0;
 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
   sum_2 += pattern2[i][j].get_num();
  }
 }
 if (V > 1){
  printf("Order 2 has %d members\n", sum_2);
 }
 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
   pattern2[i][j].set_real_prob(sum_2);
  }
 }

 sum_3 = 0;
 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
   for (int k = 0; k < cat; k++){
    sum_3 += pattern3[i][j][k].get_num();
   }
  }
 }
 if (V > 1){
  printf("Order 3 has %d members\n", sum_3);
 }
 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
   for (int k = 0; k < cat; k++){
    pattern3[i][j][k].set_real_prob(sum_3);
   }
  }
 }

 sum_4 = 0;
 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
   for (int k = 0; k < cat; k++){
    for (int l = 0; l < cat; l++){
     sum_4 += pattern4[i][j][k][l].get_num();
    }
   }
  }
 }
 if (V > 1){
  printf("Order 4 has %d members\n", sum_4);
 }
 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
   for (int k = 0; k < cat; k++){
    for (int l = 0; l < cat; l++){
     pattern4[i][j][k][l].set_real_prob(sum_4);
    }
   }
  }
 }
}


void Count_pattern_expected_prob(){
 printf("================================\nConditional Probability\n");
 for (int i = 0; i < cat; i++){
  int sum = 0;
  for (int j = 0; j < cat; j++){
   Conditional_prob[i][j] = 0;
   sum += pattern2[i][j].get_num();
  }
  for (int j = 0; j < cat; j++){
   Conditional_prob[i][j] = (double)pattern2[i][j].get_num()/(double)sum;
   printf("%c | %c = %f\n", Categ[j],Categ[i],Conditional_prob[i][j]);
  }
 }

 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
    pattern2[i][j].calculate_expected_value(new Pattern, initial_distribution);
  }
 }

 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
    for (int k = 0; k < cat; k++){
     pattern3[i][j][k].calculate_expected_value(&(Pattern)pattern2[i][j],
                initial_distribution);
    }
  }
 }

 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
    for (int k = 0; k < cat; k++){
     for (int l = 0; l < cat; l++){
      pattern4[i][j][k][l].calculate_expected_value(&(Pattern)pattern3[i][j][k],
                initial_distribution);
     }
    }
  }
 }
}

void Chi_square_test(){
 for (int i = 0; i < cat; i++){
   for (int j = 0; j < cat; j++){
    pattern2[i][j].chi_square_test();
   }
 }

 for (int i = 0; i < cat; i++){
   for (int j = 0; j < cat; j++){
    for (int k = 0; k < cat; k++){
     pattern3[i][j][k].chi_square_test();
    }
   }
 }
 for (int i = 0; i < cat; i++){
  for (int j = 0; j < cat; j++){
   for (int k = 0; k < cat; k++){
    for (int l = 0; l < cat; l++){
     pattern4[i][j][k][l].chi_square_test();
    }
   }
  }
 }
}


void Print_out(){
 printf("==============================\nPattern2\n");
 for (int i = 0; i < cat -1; i++){
   for (int j = 0; j < cat - 1; j++){
    pattern2[i][j].print_out();
   }
```

```
    }

    printf("=================================\nPattern3\n");
    for (int i = 0; i < cat - 1; i++){
     for (int j = 0; j < cat - 1; j++){
      for (int k = 0; k < cat - 1; k++){
       pattern3[j][k][i].print_out();
      }
     }
    }

    if (V){
     printf("=================================\nPattern4\n");
     for (int i = 0; i < cat - 1; i++){
      for (int j = 0; j < cat - 1; j++){
       for (int k = 0; k < cat - 1; k++){
        for (int l = 0; l < cat - 1; l++){
         //if (pattern4[k][l][j][i].get_num()){
         pattern4[k][l][j][i].print_out();
         //}
        }
       }
      }
     }
    }
   }

   void inout(){
    printf("=================================\nInOutRatio\n");
    int size = sizeof(Type)/sizeof(char);
    for (int i = 0; i < size - 1; i++){
     int in = 0;
     int out = 0;
     int num = 0;

     for (int loc = 0; loc < place_size; loc++){
      if (pla[loc].get_type() == Type[i]){
        in += pla[loc].get_in();
        out += pla[loc].get_out();
        //pla[loc].Print_out();
        num++;
      }
     }
     fprintf(stdout,"Type %c TOTAL is %6d In %6d Out %5.4d In/Out %f\n",
       Type[i], num, in, out, (double)in/(double)out);
    }
   }
```

A.1.4 Global Object

```
/*
 * funciontest.h
 *
 *  Created on: Jul 7, 2010
 *      Author: jih49
 */

#ifndef FUNCIONTEST_H_
#define FUNCIONTEST_H_
#include "Pattern2.h"
#include "Pattern3.h"
#include "Pattern4.h"
#include "Place.h"


#include "Params.h"

const int cat = 5;
extern int tess;
extern int pop_size;
extern int place_size;
extern int V;
extern int D;
extern int sum_2, sum_3, sum_4;
extern double threshold;
extern double initial_distribution[cat];
extern char Categ[cat];
extern char Type[7];
extern char* filename;
extern double Conditional_prob[cat][cat];
extern Agent* pop;
extern Agent** pop_ptr;
extern Place* pla;
extern FILE* Statusfp;
extern FILE* Outfp;
/*
extern Pattern2 pattern2[cat][cat];

extern Pattern3 pattern3[cat][cat][cat];

extern Pattern4 pattern4[cat][cat][cat][cat];
*/
void get_global_parameters();
int get_type_number(char type);
bool is_in(Agent* ag, vector<Agent*> list);
#endif /* FUNCIONTEST_H_ */
```

```
/*
 * funciontest.cpp
 *
 *  Created on: Jul 7, 2010
 *      Author: jih49
 */

#include "Global.h"

```

```
int tess = 5;
int pop_size;
int place_size;
int V;
int D;

Agent* pop;
Agent** pop_ptr;
Place* pla;

Pattern2 pattern2[cat][cat];
Pattern3 pattern3[cat][cat][cat];
Pattern4 pattern4[cat][cat][cat][cat];

int sum_2, sum_3, sum_4;
double threshold;
double initial_distribution[cat];
char Categ[cat] = {'H','N','S','W','X'};
char Type[7] = {'H', 'N', 'S', 'C', 'W', 'O', 'X'};

char* filename;

double Conditional_prob[cat][cat];

FILE* Statusfp;
FILE* Outfp;

void get_global_parameters() {
 read_parameters("_");
 get_param((char *) "V", &V);
 get_param((char *) "threshold", &threshold);
}

int get_type_number(char type){
 if (type == 'H'){
  return 0;
 }else if (type == 'N'){
  return 1;
 }else if (type == 'S'||type == 'C'){
  return 2;
 }else if (type == 'W'||type == 'O'){
  return 3;
 }else if (type == 'X'){
  return 4;
 }else{
  printf("not_a_good_type\n");
  return -1;
 }
}

bool is_in(Agent* ag, vector<Agent*> list){
 for (unsigned int i = 0; i < list.size(); i++){
  if (ag == list[i]){
   return true;
```

```
    }
  }
  return false;
}
```

A.1.5 Params Object

```
/*
   Copyright 2009 by the University of Pittsburgh
   Licensed under the Academic Free License version 3.0
   See the file "LICENSE" for more information
*/

//
//
// File Params.h
//
#ifndef _FRED_PARAMS_H
#define _FRED_PARAMS_H

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <string>
#include <vector>

using namespace std;

int get_param(char *s, int *p);
int get_param(char *s, unsigned long *p);
int get_param(char *s, double *p);
int get_param(char *s, float *p);
int get_param(char *s, char *p);
int get_param(char *s, string &p);
int read_parameters(char *paramfile);
int get_param_vector(char *s, vector < int > &p);
int get_param_vector(char *s, vector < double > &p);
int get_param_vector(char *s, double *p);
int get_param_matrix(char *s, double ***p);
bool does_param_exist(char *s);
#endif // _FRED_PARAMS_H
```

```
/*
   Copyright 2009 by the University of Pittsburgh
   Licensed under the Academic Free License version 3.0
   See the file "LICENSE" for more information
*/

//
//
// File Params.cc
//
```

```

#include "Params.h"
#include <math.h>
#include <string>
#include <sstream>

using namespace std;

#define MAX_PARAMS 1000
#define MAX_PARAM_SIZE 1024

char Param_name[MAX_PARAMS][MAX_PARAM_SIZE];
char Param_value[MAX_PARAMS][MAX_PARAM_SIZE];
int Params;
int Param_verbose = 1;

int read_parameters(char *paramfile) {
  FILE *fp;
  char name[MAX_PARAM_SIZE];
  Params = 0;

  fp = fopen("params.def", "r");
  if (fp != NULL) {
    while (fscanf(fp, "%s", name) == 1) {
      if (name[0] == '#') {
        int ch = 1;
        while (ch != '\n')
          ch = fgetc(fp);
        continue;
      } else {
        if (fscanf(fp, " = %[^\n]", Param_value[Params]) == 1) {

          //Remove end of line comments if they are there
          string temp_str(Param_value[Params]);
          size_t pos;
          string whitespaces(" \t\f\v\n\r");

          pos = temp_str.find("#");
          if (pos != string::npos)
            temp_str = temp_str.substr(0, pos);

          //trim trailing whitespace
          pos = temp_str.find_last_not_of(whitespaces);
          if (pos != string::npos) {
            if(pos != (temp_str.length() - 1))
              temp_str.erase(pos + 1);
          }
          else
            temp_str.clear(); //str is all whitespace

          strcpy(Param_value[Params], temp_str.c_str());

          strcpy(Param_name[Params], name);
          if (Param_verbose > 2) {
            printf("READ_PARAMS:␣%s␣=␣%s\n", Param_name[Params],
                Param_value[Params]);
          }
          Params++;
        } else {
          printf(
              "Help!␣Bad␣format␣in␣params.def␣file␣on␣line␣starting␣with␣%s\n",
              name);
          abort();
        }
      }
    }
  } else {
    printf("Help!␣␣Can't␣read␣paramfile␣%s\n", "params.def");
    abort();
  }
  fclose(fp);

  if (Param_verbose > 1) {
     for (int i = 0; i < Params; i++) {
       printf("READ_PARAMS:␣%s␣=␣%s\n", Param_name[i], Param_value[i]);
     }
   }

  return Params;
}

int get_param(char *s, int *p) {
  int found = 0;
  for (int i = 0; i < Params; i++) {
    if (strcmp(Param_name[i], s) == 0) {
      if (sscanf(Param_value[i], "%d", p)) {
        found = 1;
      }
    }
  }
  if (found) {
    if (Param_verbose) {
      printf("PARAMS:␣%s␣=␣%d\n", s, *p);
      fflush( stdout);
    }
    return 1;
  } else {
    if (Param_verbose) {
      printf("PARAMS:␣%s␣not␣found\n", s);
      fflush( stdout);
    }
    abort();
  }
  return 0;
}

int get_param(char *s, unsigned long *p) {
  int found = 0;
  for (int i = 0; i < Params; i++) {
    if (strcmp(Param_name[i], s) == 0) {
      if (sscanf(Param_value[i], "%lu", p)) {
        found = 1;
      }
    }
  }
  if (found) {
    if (Param_verbose) {
      printf("PARAMS: %s = %lu\n", s, *p);
      fflush( stdout);
    }
    return 1;
  } else {
    if (Param_verbose) {
      printf("PARAMS: %s not found\n", s);
      fflush( stdout);
    }
    abort();
  }
  return 0;
}

int get_param(char *s, double *p) {
  int found = 0;
  for (int i = 0; i < Params; i++) {
    if (strcmp(Param_name[i], s) == 0) {
      if (sscanf(Param_value[i], "%lf", p)) {
        found = 1;
      }
    }
  }
  if (found) {
    if (Param_verbose) {
      printf("PARAMS: %s = %f\n", s, *p);
      fflush( stdout);
    }
    return 1;
  } else {
    if (Param_verbose) {
      printf("PARAMS: %s not found\n", s);
      fflush( stdout);
    }
    abort();
  }
  return 0;
}

int get_param(char *s, float *p) {
  int found = 0;
  for (int i = 0; i < Params; i++) {
    if (strcmp(Param_name[i], s) == 0) {
      if (sscanf(Param_value[i], "%f", p)) {
        found = 1;
      }
    }
  }
  if (found) {
    if (Param_verbose) {
      printf("PARAMS: %s = %f\n", s, *p);
      fflush( stdout);
    }
    return 1;
  } else {
    if (Param_verbose) {
      printf("PARAMS: %s not found\n", s);
      fflush( stdout);
    }
    abort();
  }
  return 0;
}

int get_param(char *s, string &p){
  int found = 0;
  for (int i = 0; i < Params; i++) {
    if (strcmp(Param_name[i], s) == 0) {
      stringstream ss;
      ss << Param_value[i];
      if(ss.str().size() > 0){
      p = ss.str();
      found = 1;
      }
    }
  }
  if (found) {
    if (Param_verbose) {
      printf("PARAMS: %s = %s\n", s, p.c_str());
      fflush( stdout);
    }
    return 1;
  } else {
    if (Param_verbose) {
      printf("PARAMS: %s not found\n", s);
      fflush( stdout);
    }
    abort();
  }
  return 0;
}

int get_param(char *s, char *p) {
  int found = 0;
  for (int i = 0; i < Params; i++) {
    if (strcmp(Param_name[i], s) == 0) {
      if (strcpy(p, Param_value[i])) {
        found = 1;
      }
    }
  }
  if (found) {
    if (Param_verbose) {
      printf("PARAMS: %s = %s\n", s, p);
      fflush( stdout);
    }
    return 1;
  } else {
    if (Param_verbose) {
      printf("PARAMS: %s not found\n", s);
      fflush( stdout);
    }
    abort();
  }
  return 0;
}

int get_param_vector(char *s, vector < int > &p){
  char str[1024];
  int n;
  char *pch;
  int v;
  get_param(s,str);
  pch = strtok(str," ");
  if(sscanf(pch,"%d",&n) == 1){
    for (int i=0;i<n;i++){
      pch = strtok(NULL," ");
      if(pch == NULL) {
        printf("Help! bad param vector: %s\n", s);
        abort();
      }
      sscanf(pch,"%d",&v);
      p.push_back(v);
    }
  }
  else {
    printf("Incorrect format for vector %s\n",s);
    abort();
  }
  return n;
}

int get_param_vector(char *s, vector < double > &p){
  char str[1024];
  int n;
  char *pch;
  double v;
  get_param(s, str);
  pch = strtok(str," ");
  if (sscanf(pch, "%d", &n) == 1) {
    for (int i = 0; i < n; i++) {
      pch = strtok (NULL, " ");
      if (pch == NULL) {
        printf("Help! bad param vector: %s\n", s);
        abort();
      }
      sscanf(pch, "%lf", &v);
      p.push_back(v);
    }
  }
  else {
    printf("Incorrect format for vector %s\n",s);
    abort();
  }
  return n;
}

int get_param_vector(char *s, double *p) {
  char str[1024];
  int n;
  char *pch;
  get_param(s, str);
  pch = strtok(str, " ");
  if (sscanf(pch, "%d", &n) == 1) {
    for (int i = 0; i < n; i++) {
      pch = strtok(NULL, " ");
      if (pch == NULL) {
        printf("Help! bad param vector: %s\n", s);
        abort();
      }
      sscanf(pch, "%lf", &p[i]);
    }
  } else {
    abort();
  }
  return n;
}

int get_param_matrix(char *s, double ***p) {
  int n = 0;
  get_param((char *) s, &n);
  if (n) {
    double *tmp;
    tmp = new double[n];
    get_param_vector((char *) s, tmp);
    int temp_n = (int) sqrt((double) n);
    if (n != temp_n * temp_n) {
      printf("Improper matrix dimensions: matricies must be square,"
        " found dimension %i\n", n);
      abort();
    }
    n = temp_n;
    (*p) = new double *[n];
    for (int i = 0; i < n; i++)
      (*p)[i] = new double[n];
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        (*p)[i][j] = tmp[i * n + j];
      }
    }
    delete[] tmp;
    return n;
  }
  return -1;
}

bool does_param_exist(char *s) {

  bool found = false;
  for (int i = 0; i < Params && !found; i++) {
    if (strcmp(Param_name[i], s) == 0) {
      found = true;
    }
  }

  return found;
}
```

A.1.6 Pattern/Pattern2/Pattern3/Pattern4 Objects

```
/*
 * Pattern.h
 *
 *  Created on: Jun 25, 2010
 *      Author: jih49
 */

#ifndef PATTERN_H_
#define PATTERN_H_

#include <vector>
#include <math.h>
#include "Agent.h"

class Pattern {
public:
 Pattern() { num = 0; expected_prob = 0.0; expected_prob_ind = 0; real_prob = 0; }
 ~Pattern(){}
 virtual void calculate_expected_value(Pattern* ,double* initial_distribution)
   { expected_prob = 0; }

 int get_num(void) { return num; }
 char* get_pattern() { return transmission_pattern; }
 double get_expected_prob() { return expected_prob; }
 double get_real_prob() { return real_prob; }

 void set_real_prob(int Sum);
 void chi_square_test();
 void print_out();
protected:
 int num;
 double real_prob;
 double expected_prob;
 double expected_prob_ind;
 char* transmission_pattern;
 double chi_test,chi_test_2;
 char index,index2;
 int sum;
};

#endif /* PATTERN_H_ */
```

```
/*
 * Pattern.cpp
 *
 *  Created on: Jun 25, 2010
 *      Author: jih49
 */

#include "Pattern.h"

extern int V;
extern double threshold;
void Pattern::set_real_prob(int Sum){
 sum = Sum;
 real_prob = (double)num/(double)sum;
 if (V > 1)
  printf("real_prob_of_%s_is_%f\n", transmission_pattern, real_prob);
}

void Pattern::chi_square_test(){
 chi_test = pow((real_prob-expected_prob),2)
    *sum/expected_prob/(1-expected_prob);
 if (chi_test < threshold){
  index = 'N';
 }else if(num < sum*expected_prob){
  index = 'D';
 }else if(num > sum*expected_prob){
  index = 'U';
 }else{
  index = 'X';
 }

 chi_test_2 = pow((real_prob-expected_prob_ind),2)
     *sum/expected_prob_ind/(1-expected_prob_ind);
 if (chi_test_2 < threshold){
  index2 = 'N';
 }else if(num < sum*expected_prob_ind){
  index2 = 'D';
 }else if(num > sum*expected_prob_ind){
  index2 = 'U';
 }else{
  index2 = 'X';
	}
}

void Pattern::print_out(){

	//printf("Pattern %s: %5d %5.4f\n",transmission_pattern, num, real_prob);

	printf("%s real %6.4f expected %6.4f expected_ind
     %6.4f Chi1 %8.2f %c Chi2 %8.2f %c\n",
		transmission_pattern, real_prob, expected_prob,
			expected_prob_ind, chi_test, index, chi_test_2, index2);

}
```

```
/*
 * Pattern2.h
 *
 *  Created on: Jun 29, 2010
 *      Author: jih49
 */

#ifndef PATTERN2_H_
#define PATTERN2_H_
#include "AddressType2.h"
#include "Pattern.h"
#include "Global.h"

class Pattern2 : public Pattern{
public:
	Pattern2();
	~Pattern2(){}

	void setup(char infector, char infectee);
	void add_pattern(Agent* infector, Agent * infectee);
	void calculate_expected_value(Pattern* pattern, double* initial_distribution);
	vector<AddressType2*>* get_relation() { return &Relation; }
private:
	vector<AddressType2*> Relation;
};

#endif /* PATTERN2_H_ */
```

```
/*
 * Pattern2.cpp
 *
 *  Created on: Jun 29, 2010
 *      Author: jih49
 */

#include "Pattern2.h"

Pattern2::Pattern2() : Pattern(){
	transmission_pattern = new char[2];
```

```
}

void Pattern2::setup(char infector, char infectee){
 num = 0; expected_prob = 0.0; real_prob = 0;
 Relation.clear();
 transmission_pattern[0] = infector;
 transmission_pattern[1] = infectee;
}

void Pattern2::add_pattern(Agent* infector, Agent * infectee){
 num++;
 AddressType2 *add = new AddressType2(infector, infectee);
 Relation.push_back(add);
}

/*
void Pattern2::print_out(){
 printf("Pattern %s: real %6d (%10.9f), expected %8.2f (%10.9f), Chi %8.2f, %c\n",
   transmission_pattern, num, real_prob, sum_2*expected_prob, expected_prob, chi_test, ind
 if (V > 2){
  vector<AddressType2*>::iterator itr;
  for (itr = Relation.begin(); itr != Relation.end(); itr++){
   printf("infector %8d -> infectee %8d\n", (*itr)->get_infector()->get_id(), (*itr)->get_
  }
 }
}
*/

void Pattern2::calculate_expected_value(Pattern* pattern, double* initial_distribution){
 expected_prob = initial_distribution[get_type_number(transmission_pattern[0])]
*Conditional_prob[get_type_number(transmission_pattern[0])][get_type_number(transmission_pa
 expected_prob_ind = initial_distribution[get_type_number(transmission_pattern[0])]
      *initial_distribution[get_type_number(transmission_pattern[1])];
}
```

```
/*
 * Pattern3.h
 *
 *  Created on: Jun 28, 2010
 *      Author: jih49
 */

#ifndef PATTERN3_H_
#define PATTERN3_H_

#include "AddressType3.h"
#include "Pattern.h"
#include "Global.h"
class Pattern3 : public Pattern{
public:
 Pattern3();
 ~Pattern3(){}
 void setup(char first, char secon, char third);
 void add_pattern(Agent* first, Agent * second, Agent* third);
 void calculate_expected_value(Pattern* pattern, double* initial_distribution);
 vector<AddressType3*>* get_relation() { return &Relation;}
private:
 vector<AddressType3*> Relation;
};

#endif /* PATTERN3_H_ */
```

```
/*
 * Pattern3.cpp
 *
 *  Created on: Jun 28, 2010
 *      Author: jih49
 */

#include "Pattern3.h"

Pattern3::Pattern3() : Pattern() {
 transmission_pattern = new char[3];
}

void Pattern3::setup(char first, char second, char third){
 num = 0; expected_prob = 0.0; real_prob = 0;
 Relation.clear();
 transmission_pattern[0] = first;
 transmission_pattern[1] = second;
 transmission_pattern[2] = third;
}

void Pattern3::add_pattern(Agent* first, Agent* second, Agent* third){
 num++;
 AddressType3 *add = new AddressType3(first, second, third);
 Relation.push_back(add);
}
/*
void Pattern3::print_out(){
  printf("Pattern %s: real %6d (%10.9f), expected %8.2f (%10.9f), Chi %8.2f, %c\n",
    transmission_pattern, num, real_prob, sum_3*expected_prob, expected_prob, chi_test, ind
  if (V > 2){
    vector<AddressType3*>::iterator itr;
    for (itr = Relation.begin(); itr != Relation.end(); itr++){
      printf("first %7d %c  ->  second %7d %c  ->  third %7d %c\n",
        (*itr)->get_first()->get_id(), (*itr)->get_first()->get_infected_at(),
        (*itr)->get_second()->get_id(), (*itr)->get_second()->get_infected_at(),
        (*itr)->get_third()->get_id(), (*itr)->get_third()->get_infected_at());
    }
  }
}
*/
void Pattern3::calculate_expected_value(Pattern* pattern, double *initial_distribution){
 expected_prob = pattern->get_real_prob()
*Conditional_prob[get_type_number(transmission_pattern[1])][get_type_number(transmission_p
  expected_prob_ind = pattern->get_real_prob()
      *initial_distribution[get_type_number(transmission_pattern[2])];

}
```

```
/*
 * Pattern4.h
 *
 *  Created on: Jun 28, 2010
 *      Author: jih49
 */

#ifndef PATTERN4_H_
#define PATTERN4_H_

#include "AddressType4.h"
#include "Pattern.h"
#include "Global.h"
class Pattern4 : public Pattern{
public:
  Pattern4();
  ~Pattern4(){}
  void setup(char first , char second, char third , char forth);
  void add_pattern(Agent* first , Agent * second, Agent* third, Agent* forth);
  void calculate_expected_value(Pattern* pattern, double* initial_distribution);
private:
  vector<AddressType4*> Relation;
};

#endif /* PATTERN4_H_ */
```

```
/*
 * Pattern4.cpp
 *
 *  Created on: Jun 28, 2010
 *      Author: jih49
 */

#include "Pattern4.h"

Pattern4::Pattern4(): Pattern() {
  transmission_pattern = new char[4];
}
void Pattern4::setup(char first , char second, char third , char forth){
 num = 0; expected_prob = 0.0; real_prob = 0;
  Relation.clear();
  transmission_pattern[0] = first;
  transmission_pattern[1] = second;
  transmission_pattern[2] = third;
  transmission_pattern[3] = forth;
}

void Pattern4::add_pattern(Agent* first , Agent* second, Agent* third, Agent* forth){
 num++;
 AddressType4 *add = new AddressType4(first, second, third, forth);
 Relation.push_back(add);
}

/*
void Pattern4::print_out(){
 printf("Total : %d\n", num);
 if (V > 2){
  vector<AddressType4*>::iterator itr;
  for (itr = Relation.begin(); itr != Relation.end(); itr++){
   printf("first %7d %c -> second %7d %c -> third %7d %c -> forth %7d %c\n",
       (*itr)->get_first()->get_id(), (*itr)->get_first()->get_infected_at(),
       (*itr)->get_second()->get_id(), (*itr)->get_second()->get_infected_at(),
       (*itr)->get_third()->get_id(), (*itr)->get_third()->get_infected_at(),
       (*itr)->get_forth()->get_id(), (*itr)->get_forth()->get_infected_at());
  }
 }
}
*/

void Pattern4::calculate_expected_value(Pattern *pattern, double *initial_distribution){
 expected_prob = pattern->get_real_prob()
*Conditional_prob[get_type_number(transmission_pattern[2])][get_type_number(transmission_pa
 expected_prob_ind = pattern->get_real_prob()
      *initial_distribution[get_type_number(transmission_pattern[3])];

}
```

A.1.7 AddressType2/AddressType3/AddressType4 Objects

```
/*
 * AddressType2.h
 *
 *  Created on: Jun 28, 2010
 *      Author: jih49
 */

#ifndef ADDRESSTYPE2_H_
#define ADDRESSTYPE2_H_
#include "Agent.h"

class AddressType2 {
public:
 AddressType2(Agent* Infector, Agent* Infectee);
 ~AddressType2();

 Agent* get_infector(void) { return infector;}
 Agent* get_infectee(void) { return infectee;}

private:
 Agent* infector;
 Agent* infectee;
};

#endif /* ADDRESSTYPE2_H_ */
```

```
/*
 * AddressType2.cpp
 *
 *  Created on: Jun 28, 2010
 *      Author: jih49
 */

#include "AddressType2.h"

AddressType2::AddressType2(Agent* Infector, Agent* Infectee) {
 infector = Infector;
 infectee = Infectee;
}

AddressType2::~AddressType2() {
 delete infector;
 delete infectee;
}
```

```
/*
 * AddressType3.h
 *
 *  Created on: Jun 28, 2010
 *      Author: jih49
 */

#ifndef ADDRESSTYPE3_H_
#define ADDRESSTYPE3_H_

#include "Agent.h"

class AddressType3 {
public:
 AddressType3(Agent* First, Agent* Second, Agent* Third);
 ~AddressType3();
 Agent* get_first(void) { return first; }
 Agent* get_second(void) { return second; }
 Agent* get_third(void) { return third; }
private:
 Agent* first;
 Agent* second;
 Agent* third;
};

#endif /* ADDRESSTYPE3_H_ */
```

```
/*
 * AddressType3.cpp
 *
 *  Created on: Jun 28, 2010
 *      Author: jih49
 */

#include "AddressType3.h"

AddressType3::AddressType3(Agent* First, Agent* Second, Agent* Third) {
	first = First;
	second = Second;
	third = Third;
}

AddressType3::~AddressType3() {
	delete first;
	delete second;
	delete third;
}
```

```
/*
 * AddressType4.h
 *
 *  Created on: Jun 28, 2010
 *      Author: jih49
 */

#ifndef ADDRESSTYPE4_H_
#define ADDRESSTYPE4_H_

#include "Agent.h"

class AddressType4 {
public:
	AddressType4(Agent* First, Agent* Second, Agent* Third, Agent* Forth);
	~AddressType4();
	Agent* get_first(void) { return first; }
	Agent* get_second(void) { return second; }
	Agent* get_third(void) { return third; }
	Agent* get_forth(void) { return forth; }
private:
	Agent* first;
	Agent* second;
	Agent* third;
	Agent* forth;
};

#endif /* ADDRESSTYPE4_H_ */
```

```
/*
 * AddressType4.cpp
 *
 *  Created on: Jun 28, 2010
 *      Author: jih49
 */

#include "AddressType4.h"

AddressType4::AddressType4(Agent* First, Agent* Second, Agent* Third, Agent* Forth) {
  first = First;
  second = Second;
  third = Third;
  forth = Forth;
}

AddressType4::~AddressType4() {
  delete first;
  delete second;
  delete third;
  delete forth;
}
```

A.1.8 fred_main Object

```
/*
 * fred_main.h
 *
 *   Created on: May 13, 2010
 *       Author: jih49
 */

#ifndef FRED_MAIN_H_
#define FRED_MAIN_H_


#endif /* FRED_MAIN_H_ */
```

```
/*
 *   fred_main.cpp
 *
 *   Created on: May 13, 2010
 *       Author: jih49
 */
#include <err.h>
#include <errno.h>
#include <sys/stat.h>
#include <vector>
#include "fred_main.h"
#include "Global.h"
#include "Functions_Location_Pattern.h"
#include "Place.h"

extern Pattern2 pattern2[cat][cat];
extern Pattern3 pattern3[cat][cat][cat];
extern Pattern4 pattern4[cat][cat][cat][cat];
//const char Type [7] = {'H', 'N', 'S', 'C', 'W', 'O', 'X'};
```

```

using namespace std;

int main(int argc, char *argv[])
{
  time_t clock;// current date
  fprintf(stdout, "FRED started  ");
  time(&clock);
  fprintf(stdout, "%s", ctime(&clock));

  Statusfp = stdout;

  get_global_parameters();

  for (int i = 1; i < argc; i++) {
   filename = argv[i];
   Place_preparation();
   Agent_preparation(filename);



   /*    char fname[80];
   sprintf(fname, "DesAnalysis%s",filename);
   FILE *Outfp = fopen(fname,"w");
   if (NULL == Outfp){
    printf("Help! Cannot open %s\n", fname);
    abort();
   }
   fclose(Outfp);
*/

    Search_patterns();
   Count_initial_distribution();
   Set_pattern_prob();
   Count_pattern_expected_prob();
   Chi_square_test();
   Print_out();


////////Place Analysis
   //update all places
   for (int loc = 0; loc < place_size; loc++){
    pla[loc].Update();//Update statistics , e.g. in out
   }
   //print in/out ratio
   inout();
/*
   mode_t mask;          // the user's current umask
   mode_t mode = 0777; // as a start

   // create the "OUT" directory, if it does not already exist
   mask = umask(0); // get the current mask, which reads and sets...
   umask(mask);      // so now we have to put it back
   mode ^= mask;     // apply the user's existing umask
```

```
    if (0!=mkdir("socialaction", mode) && EEXIST!=errno) // make it
      err(errno, "mkdir(\"socialaction\") failed");        // or die


    Place * tmp;
    for (int loc = 0; loc < place_size; loc++){
     if (pla[loc].get_type() == 'S'){
      tmp = &pla[loc];
      tmp->Print_socialaction();
     }
    }
   */
   }

   time(&clock);
   fprintf(stdout, "==================================\nrun_successfully\n");
   fprintf(stdout, "%s", ctime(&clock));
   return 0;
}
```

A.2 SOURCE CODE OF STATISTICALANALYSIS

```
# author : Jiawei
# dat e : Aug 1 st 2011
# functions : read command line
arguments <    commandArgs( ) ;
options <    arguments [ grep(    =    , arguments ) ] ;
options <    strsplit ( options ,    =    ) ;
print ( length ( options ) ) ;
getOption <    function (name ) {

if ( length ( options)==0) {
 return(NULL) ;
}
for ( i in 1 : length ( options ) ) {
 i f ( options [ [ i ] ] [ 1 ] == name ) {
  return( options [ [ i ] ] [ 2 ] ) ;
  }
 }
}
```

```
# author : Jiawei
# date : Aug 1st 2011
# functions: load library , read data from files
# plot figures of comparisons, writefiles , and etc.

library (Hmisc);

read_statistic <    function (statistic , scenarios_directory) {
for ( i in 1 : length ( scenarios_directory ) ) {
TraceAnalysis <    paste(scenarios_directory[i] ,    /TraceAnalysis   , sep=      );
```

```
setwd(TraceAnalysis);
tmp <     read.table(statistic, sep=   \t   );

if ( i == 1) {
stat_data <     tmp;
} else {
stat_data <     cbind ( stat_data , tmp ) ;
}
}
return( stat_data ) ;
}

# error bar plot
errbarplot <     function(data, colnames, scenarios, xxlab=         , yylab=         ) { #plo

amplify <     1.96;
if (nrow(data) !=length( colnames )) {
colnames = NULL;
}

for ( i in seq ( 1 , ncol (data) ,by=2)) {
if ( i == 1) {
errbar(1:nrow(data),data[,i],data[,i]   data[,i+1]*amplify,data[,i]+data[,i+1]*amplify,
text (1:nrow(data),0,srt=90,adj=1,label=colnames,xpd=T,cex=1.5);
} else {
errbar(1:nrow(data),data[,i],data[,i]   data[,i+1]*amplify,data[,i]+data[,i+1]*amplify,
}
}

for ( i in seq ( 1 , ncol (data ) ,by=2)) {
lines(1 :nrow(data ) , data [ , i ] , col = ( i+1)/ 2 ) ;
}

legend (   topright      ,scenarios,pch=16,col=seq(1,(ncol(data)/2)));
}


generate_cp_names <     function ( type ) {
for ( i in 1 : length ( type ) ) {
for ( j in 1 : length ( type ) ) {
if ( i == 1&&j == 1) {
names <     c ( paste ( type [ j ] ,     |     , type [ i ] ) ) ;
} else {
names <     c (names , c ( paste ( type [ j ] ,     |     , type [ i ] ) ) ) ;
}
}
}
return(names ) ;
}

generatepattern2_names <     function ( type ) {
for ( i in 1 : length ( type ) ) {
for ( j in 1 : length ( type ) ) {
if ( i == 1&&j == 1) {
```

```
names <    c ( paste ( type [ i ] , type [ j ] , sep=       ) ) ;
} else {
names <    c (names , c ( paste ( type [ i ] , type [ j ] , sep=       ) ) ) ;
}
}
}
return(names ) ;
}

generate_pattern3_names <    function ( type ) {
for ( i in 1 : length ( type ) ) {
for ( j in 1 : length ( type ) ) {
for ( k in 1 : length ( type ) ) {
if ( i == 1&&j == 1&&k==1) {
names <    c ( paste ( type [ i ] , type [ j ] , type [ k ] , sep=       ) ) ;
} else {
names <    c (names , c ( paste ( type [ i ] , type [ j ] , type [ k ] , sep=       ) ) )
}
}
}
}
return(names ) ;
}

generate_pattern4_names <    function ( type ) {
for ( i in 1 : length ( type ) ) {
for ( j in 1 : length ( type ) ) {
for ( k in 1 : length ( type ) ) {
for ( l in 1 : length ( type ) ) {
if ( i == 1&&j == 1&&k==1&&l==1) {
names <    c ( paste ( type [ i ] , type [ j ] , type [ k ] , type [ l ] , sep=       ) )
} else {
names<   c (names,c(paste(type[i], type[j], type[k], type[l],sep=      )));
}
}
}
}
}
return(names);
}

# return array of rows with top 20%largest difference between expectedvalue and real value
#filtertop <    function ( data , ratio =0.2){
# difference <    abs ( data [ ,1]    data [ , 3 ] ) ;
# return (c(rank(  difference)<length(difference)*ratio));
#}

filter_top <    function (data , ratio =0.2) {
data <    data [ , seq ( 1 , ncol (data ) ,by=2 ) ] ;
variance <    apply (data , 1 , var ) ;
return(c(rank(  variance) < length(variance)*ratio) ) ;
}
```

```
# EpidemicDynamicPlot .R
# date : 6/30/2011
# author : Jiawei Huang

# function: plot epidemic dynamics by days.
# parameter s : group = S |E| I |R
# command line:
# R CMD BATCH     a r g s     g r o u p =   I        EpidemicDynamicPlot.R
#

library(Hmisc) ;
wd <    getwd();
source (paste (wd,    /CommandReader. R   , sep=      ));
source (paste (wd,    /params. d e f   , sep=       ));

############################## Parametersetting
#[final constant] parameters , donnot change
S <    6 ;
E <    8 ;
I <    10 ;
R <    14 ;

# read from command line , which group to plot
group <    getOption(       g r o u p    ) ;

if(is.null(group)){
 OPT <      E x p o s e d  ;
 group <    E;
} else if ( group ==    I    ) {
 OPT <      I n f e c t i o u s  ;
 group <    I ;
} else if(group ==    S   ){
 OPT <      S u s c e p t i b l e s  ;
 group <    S;
} else if (group==   R   ){
 OPT <      R e c o v e r e d  ;
 group <    R;
} else {
 OPT <      E x p o s e d   ;
 group <    E;
}
OPT;

# output directory: arbitrarily assigned output directory.Defaultis./StatisticalAnalysis
StatisticalAnalysis <    paste (wd,    /StatisticalAnalysis   , sep=      );
if(!file.exists (StatisticalAnalysis)){
 dir.create(StatisticalAnalysis);
}
outfile <    paste (wd,    /StatisticalAnalysis/    ,OPT,     .jpg  , sep=      );

##############################
jpeg ( outfile ,width = 960, height = 480, quality =100);

for ( i in 1 :length (scenarios_directory)) {
  files <
   list.files(scenarios_directory[i])[grep(  out  ,list.files(scenarios_directory[i]))];
  out.all<  c();
  for (j in 1:length(files)){
   out <   read.table(paste(scenarios_directory[i],  /  ,files[j],sep=    ))
   if ( j == 1) {
    out.all <   out[, group];
   } else{
    out . al l <   cbind ( out . al l , out [ , group ] ) ;
   }
  }
  out.summary <   cbind(apply(out.all,1,mean),apply(out.all,1,sd));
  out.summary <   out.summary[1:180,];
  if ( i==1) {
   errbar(1:nrow(out.summary), out.summary[,1],out.summary[,1]    out.summary[,2],
    out.summary[,1]+out.summary[,2],col=i,
    xlab =  Day of outbreak  ,ylab = paste (  No.of  , OPT, sep =     ));
   lines (out.summary[,1],col=i);
  } else {
   errbar(1:nrow(out.summary),out.summary[,1], out.summary[  ,1]    out.summary[,2],
    out.summary[,1]+out.summary[,2]  , col=i  , add = TRUE);
    lines (out.summary[,1], col=i);
  }
 }
legend (  topright  , c(scenarios),pch=16,col=seq(1,length(scenarios_directory)));

dev.off();
```

```
# author : Jiawei
# date : Aug 1st 2011
#functions : main function. control analysis rocess.
###load functions
wd <   getwd( ) ;
source (paste (wd,   /StatisticalAnalysisFunctions.R  ,sep=    ));
source ( paste (wd,   /CommandReader.R   , sep=    ));
source ( paste (wd,   /params.def   , sep=    ));

### Set parameters
# output directory: arbitrarily assigned output directory.Defaultis./StatisticalAnalysis
StatisticalAnalysis <   paste (wd,   /StatisticalAnalysis  , sep=    );
if(!file.exists(StatisticalAnalysis)){
 dir.create(StatisticalAnalysis);
}

### pattern2 ,3 ,4 percent
statistic <    Pattern2real  ;
pattern2_real <   read_statistic(statistic  , scenarios_directory);
type <   c(  H  ,  N  ,  S  ,  W  );
names <   generate_pattern2_names(type);
file_name< paste(StatisticalAnalysis,  /DiconComparison.jpeg  ,sep=    );
jpeg ( filename = file_name , width = 960, height = 480, quality=100);
errbarplot (pattern2_real ,names , scenarios,
    xxlab=  Dicon Patterns  , yylab =  Percent  );
```

```
dev.off();

statistic <     Pattern3_real  ;
pattern3_real <    read_statistic(statistic,scenarios_directory);
type <    c(  H   ,  N   ,  S   ,  W   );
names <    generate_pattern3_names(type);
file_name <    paste(StatisticalAnalysis,   /TriComparison.jpeg   ,sep=      );

jpeg (filename = file_name, width=1200, height = 480,quality=100);
errbarplot( pattern3_real,names , scenarios,
    xxlab= Tricon Patterns  ,yylab = Percent  );
dev.off();

statistic <     Pattern4_real  ;
pattern4_real <    read_statistic(statistic,scenarios_directory);
type <    c (   H   ,   N   ,   S   ,   W   );
names <    generate_pattern4_names(type);
file_name <    paste (StatisticalAnalysis,    /TetComparison.jpeg   ,sep=      );
#filter
filter <  filter_top(pattern4_real);
pattern4_real <    pattern4_real[filter,];
names <    names[filter];

jpeg(filename = file_name , width = 1200 , height = 480 , quality =100);
errbarplot (pattern4_real ,names , scenarios ,
    xxlab= Tetcon Patterns   , yylab = Percent  )
dev.off();

### pattern2 ,3 ,4 count number
statistic <     Pattern2_countreal  ;
pattern2_sum <    read_statistic(statistic,scenarios_directory);
type <    c(  H   ,  N   ,  S   ,  W   );
names <    generate_pattern2_names(type);
file_name <    paste(StatisticalAnalysis,    /DiconComparisonSUM.jpeg   , sep=      );
jpeg(filename =file_name , width = 960, height = 480 , quality =100);
errbar_plot(pattern2_sum,names,scenarios,
    xxlab= Dicon Patterns   , yylab = NO.   );
dev.off();

statistic<   Pattern3_countreal  ;
pattern3_sum <    read_statistic(statistic,scenarios_directory);
type <    c (   H   ,   N   ,   S   ,   W   );
names <    generatepattern3_names(type);
file_name <    paste(StatisticalAnalysis,    /TriComparisonSUM.jpeg   ,sep=      );
jpeg ( filename = file_name , width = 1200 , height = 480 , quality =100);
errbar_plot ( pattern3_sum,names , scenarios ,
    xxlab= Tricon Patterns  ,yylab = No .   );
dev.off();

statistic <     Pattern4_countreal  ;
pattern4_sum <    read_statistic(statistic,scenarios_directory);
type <    c (   H   ,   N   ,   S   ,   W   );
names < generate_pattern4_names( type );
filename <    paste(StatisticalAnalysis,    /TetComparisonSUM.jpeg   ,sep=      );
```

```
#filter
filter <  filter_top(pattern4_sum);
pattern4_sum_filtered <     pattern4_sum[filter ,];
names <     names[filter];

jpeg ( filename = file_name , width =1200 , height = 480, quality=100);
e r r ba r plot ( pattern4_sum_filtered ,names , scenarios ,
    xxlab= Tetcon Patterns  , yylab = No .   )
dev.off();
```

BIBLIOGRAPHY

[1] B.Y. Lee, P.C. Cooley, S. Brown, M.A. Potter, and W Wheaton. A computer simulation model of the state of pennsylvania., May 2009.

[2] Bruce Y. Lee and et al. Simulating school closure strategies to mitigate an influenza epidemic. *Journal of Public Health Management and Practice*, 16(3):252–261, May/Jun 2010.

[3] D. J. Daley and J. Gani. *Epidemic Modeling: An Introduction.* NY: Cambridge University Press, 2005.

[4] W. O. Kermack and A. G. McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A*, 115(772):700–721, August 1927.

[5] W. D. Wheaton, J. C. Cajka, B. M. Chasteen, D. K. Wagener, P. C. Cooley, L. Ganapathi, D. J. Roberts, and J. L. Allpress. Synthesized Population Databases: A US Geospatial Database for Agent-Based Models. *Methods Rep RTI Press*, 2009:905, May 2009.

[6] Bruce Y. Lee, Shawn T. Brown, Philip C. Cooley, Richard K. Zimmerman, William D. Wheaton, Shanta M. Zimmer, John J. Grefenstette, Tina-Marie Assi, Timothy J. Furphy, Diane K. Wagener, and Donald S. Burke. A computer simulation of employee vaccination to mitigate an influenza epidemic. *American Journal of Preventive Medicine*, 38(3):247 – 257, 2010.

[7] Neil M. Ferguson, Derek A. T. Cummings, Christophe Fraser, James C. Cajka, Philip C. Cooley, and Donald S. Burke. Strategies for mitigating an influenza pandemic. *Nature*, 442(7101):448–452, April 2006.

[8] B. Y. Lee, S. T. Brown, G. W. Korch, P. C. Cooley, R. K. Zimmerman, W. D. Wheaton, S. M. Zimmer, J. J. Grefenstette, R. R. Bailey, T. M. Assi, and D. S. Burke. A computer simulation of vaccine prioritization, allocation, and rationing during the 2009 H1N1 influenza pandemic. *Vaccine*, 28:4875–4879, Jul 2010.

[9] Stephen Eubank, Hasan Guclu, V. S. Anil Kumar, Madhav V. Marathe, Aravind Srinivasan, Zoltan Toroczkai, and Nan Wang. Modelling disease outbreaks in realistic urban social networks. *Nature*, 429(6988):180–184, May 2004.

[10] Neil M. Ferguson, Derek A. T. Cummings, Simon Cauchemez, Christophe Fraser, Steven Riley, Aronrag Meeyai, Sopon Iamsirithaworn, and Donald S. Burke. Strategies for containing an emerging influenza pandemic in Southeast Asia. *Nature*, 437(7056):209–214, August 2005.

[11] M. J. North, C. M. Macal, J. S. Aubin, P. Thimmapuram, M. Bragen, J. Hahn, J. Karr, N. Brigham, M. E. Lacy, and D. Hampton. Multiscale agent-based consumer market modeling. *Complexity*, 15(5):37–47, 2010.

[12] Thierry Moyaux, Brahim Chaib-Draa, and Sophie D'Amours. *Supply Chain Management and Multiagent Systems: An Overview*. Multiagent based Supply Chain Management, 2006.

[13] Paul Caplat, Madhur Anand, and Chris Bauch. Symmetric competition causes population oscillations in an individual-based model of forest dynamics. *Ecological Modelling*, 211(3-4):491 – 500, 2008.

[14] Tarik Hadzibeganovic, Dietrich Stauffer, and Christian Schulze. Agent-based computer simulations of language choice dynamics. *Annals of the New York Academy of Sciences*, 1167(1):221–229, 2009.

[15] Mark E. J. Newman, Albert L. Barabási, and Duncan J. Watts, editors. *The Structure and Dynamics of Networks*. Princeton University Press, 2006.

[16] Herbert W. Hethcote. The mathematics of infectious diseases. *SIAM Review*, 42(4):pp. 599–653, 2000.

[17] Ira M. Longini, Azhar Nizam, Shufu Xu, Kumnuan Ungchusak, Wanna Hanshaoworakul, Derek A. T. Cummings, and M. Elizabeth Halloran. Containing pandemic influenza at the source. *Science*, 309(5737):1083–1087, 2005.

[18] Shawn Brown, Julie Tai, Rachel Bailey, Philip Cooley, William Wheaton, Margaret Potter, Ronald Voorhees, Megan LeJeune, John Grefenstette, Donald Burke, Sarah McGlone, and Bruce Lee. Would school closure for the 2009 h1n1 influenza epidemic have been worth the cost?: a computational simulation of pennsylvania. *BMC Public Health*, 11(1):353, 2011.

[19] Bruce Y. Lee, Shawn T. Brown, Rachel R. Bailey, Richard K. Zimmerman, Margaret A. Potter, Sarah M. McGlone, Philip C. Cooley, John J. Grefenstette, Shanta M. Zimmer, William D. Wheaton, Sandra Crouse Quinn, Ronald E. Voorhees, and Donald S. Burke. The benefits to all of ensuring equal and timely access to influenza vaccines in poor communities. *Health Affairs*, 30(6):1141–1150, 2011.

[20] Matt J. Keeling and Ken T. D. Eames. Networks and epidemic models. *Journal of The Royal Society Interface*, 2(4):295–307, September 2005.

[21] Fabrice Carrat, Julie Luong, Herve Lao, Anne-Violaine Salle, Christian Lajaunie, and Hans Wackernagel. A 'small-world-like' model for comparing interventions aimed at preventing and controlling influenza pandemics. *BMC Medicine*, 4(1):26+, October 2006.

[22] Alden S. Klovdahl. Social networks and the spread of infectious diseases: The aids example. *Social Science & Medicine*, 21(11):1203 – 1216, 1985.

[23] A.S. Klovdahl, J.J. Potterat, D.E. Woodhouse, J.B. Muth, S.Q. Muth, and W.W. Darrow. Social networks and infectious disease: The colorado springs study. *Social Science & Medicine*, 38(1):79 – 88, 1994.

[24] A. M. Jolly and J. L. Wylie. Gonorrhoea and chlamydia core groups and sexual networks in manitoba. *Sexually Transmitted Infections*, 78(suppl 1):i145–i151, April 2002.

[25] A. S. Klovdahl, E. A. Graviss, A. Yaganehdoost, M. W. Ross, A. Wanger, G. J. Adams, and J. M. Musser. Networks and tuberculosis: an undetected community outbreak involving public places. *Soc Sci Med*, 52(5):681–694, March 2001.

[26] Nicholas A. Christakis and James H. Fowler. The spread of obesity in a large social network over 32 years. *The New England journal of medicine*, 357(4):370–379, July 2007.

[27] James H. Fowler and Nicholas A. Christakis. Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the framingham heart study. *BMJ*, 337:a2338, December 2008.

[28] John T. Cacioppo, James H. Fowler, and Nicholas A. Christakis. Alone in the crowd: the structure and spread of loneliness in a large social network. *Journal of personality and social psychology*, 97(6):977–991, December 2009.

[29] Anne M. Johnson, Jane Wadsworth, Kaye Wellings, Sally Bradshaw, and Julia Field. Sexual lifestyles and HIV risk. *Nature*, 360(6403):410–412, December 1992.

[30] Joël Mossong, Niel Hens, Mark Jit, Philippe Beutels, Kari Auranen, Rafael Mikolajczyk, Marco Massari, Stefania Salmaso, Gianpaolo Scalia Tomba, Jacco Wallinga, Janneke Heijne, Malgorzata Sadkowska-Todys, Magdalena Rosinska, and W. John Edmunds. Social contacts and mixing patterns relevant to the spread of infectious diseases. *PLoS Med*, 5(3):e74, 03 2008.

[31] Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random Struct. Algorithms*, 6:161–179, March 1995.

[32] A. C. Ghani, J. Swinton, and G. P. Garnett. The role of sexual partnership networks in the epidemiology of gonorrhea. *Sex Transm Dis*, 24:45–56, Jan 1997.

[33] Shai S. Shen-Orr, Ron Milo, Shmoolik Mangan, and Uri Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature Genetics*, 31(1):64–68, April 2002.

[34] Patrick Eichenberger, Masaya Fujita, Shane T Jensen, Erin M Conlon, David Z Rudner, Stephanie T Wang, Caitlin Ferguson, Koki Haga, Tsutomu Sato, Jun S Liu, and Richard Losick. The program of gene transcription for a single differentiating cell type during sporulation in bacillus subtilis. *PLoS Biol*, 2(10):e328, 09 2004.

[35] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[36] Duncan T. Odom, Nora Zizlsperger, D. Benjamin Gordon, George W. Bell, Nicola J. Rinaldi, Heather L. Murray, Tom L. Volkert, Jrg Schreiber, P. Alexander Rolfe, David K. Gifford, Ernest Fraenkel, Graeme I. Bell, and Richard A. Young. Control of pancreas and liver gene expression by hnf transcription factors. *Science*, 303(5662):1378–1381, 2004.

[37] Nitzan Rosenfeld, Michael B Elowitz, and Uri Alon. Negative autoregulation speeds the response times of transcription networks. *Journal of Molecular Biology*, 323(5):785 – 793, 2002.

[38] A. Becskei and L. Serrano. Engineering stability in gene networks by autoregulation. *Nature*, 405(6786):590–593, June 2000.

[39] R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2005. ISBN 3-900051-07-0.

[40] Jennifer L. Gardy, James C. Johnston, Shannan J. Ho Sui, Victoria J. Cook, Lena Shah, Elizabeth Brodkin, Shirley Rempel, Richard Moore, Yongjun Zhao, Robert Holt, Richard Varhol, Inanc Birol, Marcus Lem, Meenu K. Sharma, Kevin Elwood, Steven J.M. Jones, Fiona S.L. Brinkman, Robert C. Brunham, and Patrick Tang. Whole-genome sequencing and social-network analysis of a tuberculosis outbreak. *New England Journal of Medicine*, 364(8):730–739, 2011.