Infectious disease simulation

Apostolos Athanasiou
aa74552
apostolos.athanasiou@utexas.edu

ABSTRACT
Simulating the spread of infectious diseases has been one of the most challenging problems faced by scientists and health officials. Although there have been proposed several modeling approaches, performing explicit simulations is the most straightforward approach. In this paper a simple model with four states for each person is developed. These four categories are: sick, susceptible, recovered, and inoculated. Five object-oriented C++ programs are developed to implement the explicit simulations in a computer environment. Moreover, a parametric study is conducted to reveal the importance of the input parameters. Finally, the observations of the parametric study are presented

Author Keywords
Epidemic disease; explicit modeling; C++;

INTRODUCTION
Over the past centuries, infectious diseases have been continuously been a threat for humans. Especially during periods with limited healthcare resources such as war time, infectious diseases have been the reason for the loss of millions of humans' lives. Even if these diseases can be cured, it is still really important to visualize how fast they can transmit in a human population. AIDS is a modern virus which may not cause death if medical treatment is received, but can be transmitted to people, following the same model as an infectious disease. Tools which can model the spread of an infectious disease have a significant value since they can predict the how fast a disease spread. Having predictions about the way a disease spreads, can be used to raise public awareness and as a result to reduce the actual rates of infections.

One characteristic application of the spread modeling is the annual flu. Every year a new virus appears, with different characteristics, and affects millions of people. Inoculation can affect how the virus spreads and reduce the number of people who are susceptible. Although the aforementioned statement is accepted by the public, not having quantifiable effectiveness outcomes, makes people less interested in having their flue shot. The tool presented in this paper can simulate simple cases of disease spread. It can be used to raise awareness about the importance of immunization. The structure of the paper is as follows: First, a quick discussion about the developed program is presented. Then a set of input parameters is used to perform 'experiments' and draw remarks about the spread of diseases. Finally, a summary of the lessons learned through this effort is provided.

THE PROGRAM
A significant effort was made to develop a modular program which can accurately model the spread of infectious diseases on arbitrary populations. To achieve that C++ programming language was the most suitable choice. Utilizing the object-oriented capabilities of the language allowed the development of an efficient program, without any repetition of blocks of code. For the shake of convenience, Visual Studio was the editor used. Although the editing was performed in a windows environment, the compilation of the scripts and the parametric study were performed in a Unix virtual machine.

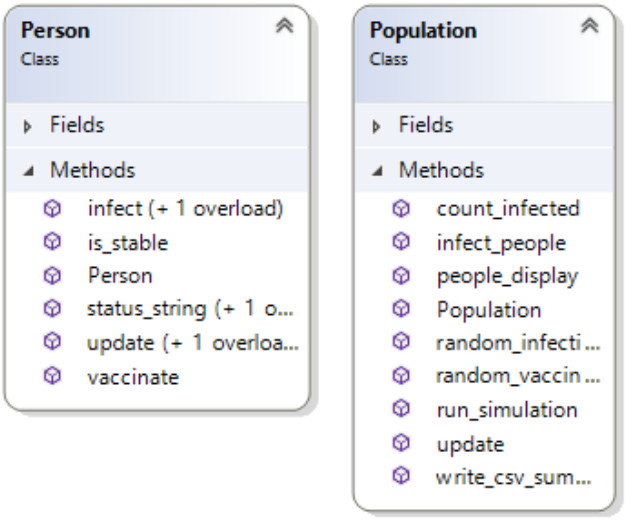

Figure 1 The classes and their corresponding methods

The architecture of the code is relatively simple. Two classes defined and used throughout the program. The first on is the Person class, which includes methods about single persons. The second one is the Population class which models how an infection propagates in a finite population. A visualization of the available classes is provided in Figure 1. In an attempt to make the program more modular, header (Project_head.cc) and implementation (Project_lib.cc) scripts were used, on top of the main program script (Ex39_1.cc to Ex39_5.cc).

By default, each person is defined as susceptible to disease at the definition of the Person class. A numeric variable is used to set the status of each person. Numbers greater than 0 indicate an infected person and the number of days until recovery. People who have recovered after being infected correspond to -1. Finally, vaccinated people correspond to -2.

Five methods are included in the Person class. The infect() method is used to change the status of a susceptible person to infected. The is_stable() method, determines whether a person has been sick in the past and now is recovered. As mentioned earlier the status of each person is stored using an integer number. The status_string() method, interprets this number to the corresponding string status. The update() method, is used to update the status of each person and the vaccinate() method is used to "protect" people from getting infected. Notice that based on the problem statement, an infected person recovers after five step cycles. All the aforementioned methods are also utilized by the Population class. Notice that right next to the name of some methods in Figure 1 is a (1+ overload) message. That message indicates that there are several definitions of the method, depending on the input data. Utilizing polymorphism was a useful tool, in order to avoid repeating any part of code.

On the other hand, the Population class, incorporates all the methods which track the effect of the infection on the whole population. Seven methods are included in the population class. The name of each method described its function. The "random" methods are used to randomly infect or vaccinate part of the population. The run_simulation() method, performs the necessary steps depending on the user input. This feature allows us to use the same class/methods for every single program developed for this project.

Five main programs are used to accommodate the Input/Output of data for the simulations. On each main program the srand(time(NULL)) statement is used to pick the seed for the pseudorandom number generator. The rand() function, when given the same starting number (seed) always generates the same sequence of numbers. That is the reason the number generator is called pseudorandom. All the scripts include extensive comments, which describe the function of each component.

EXPERIMENTS

A significant effort was made to conduct the experiments in an automated manner. Although the output on the terminal provides to the user the necessary information, it is really hard to make observations based on single experiments. To tackle this issue, an additional method was added to the implementation file, which accommodated the creation of a database in the form of a comma separated values files. The write_csv_summary() method, appends the results of the experiment into a csv file, which makes easier the preprocessing using Pandas. At this point it has to be mentioned that if the code is executed at the ISP virtual machine on TACC, the method doesn't execute appropriately. The issue is that due to some type of rights issue, the method overwrites the cvs file, instead of appending data at the last line. Even if the user modifies the permissions associated with the file (using chmod), the program still doesn't have the anticipated response. To accommodate the experiments a personal virtual machine was created. Without modifying any part of code, the program yielded the desired outcome.

Moreover, the experimental data input was automated. A shell script which included four loops was deployed, in order to perform all the experiments required. The input parameters are: (1) the size of the population, (2) the probability of transfer, (3) the immunization percentage, (4) and the number of people interacted. To reduce the computational time, educated guesses were done to estimate the ranges of the aforementioned parameters. The population ranged from 500 to 1000 people with step 100 people. The probabilities ranged between 0 and 1 with step 0.1. Finally the number of people interacted ranged from 0 to 50 with a step of 10. All the aforementioned cases were combined and yielded 4356 experiments. To compile the scripts a make file is included. The experiment.sh shell script which is included in the submission performs all the experiments automatically, without any input from the users, just by looping simple bash commands.

The results of the final step corresponding to each experiment performed are included in the summary.csv file. The last part of this presents a discussion about the questions listed in the project statement.

```
Run a number of simulations with population
sizes and contagion probabilities. Are there
cases where people escape getting sick?
```

Disease transmission probability is the main parameter which controls if the whole population will get sick. If the probability of transmission is really low, then people of the population can escape from getting sick. A characteristic example are diseases which do not transmit through contact, and as a result have a very low probability of transmission.

```
Describe the effect of vaccinated people on
the spread of the disease. Why is this model
unrealistic?
```

The model is not a good representation of the reality, since the disease is transmitted only to adjacent people around the sick person. That is not an accurate model, since humans interact with multidimensional manner, and are not stacked one next to each other.

```
Record how long the disease runs through the
population. With a fixed number of contacts
and probability of transmission, how is this
number of function of the percentage that is
vaccinated?
```

If we fix the disease transmission probability, the number of steps required for the disease to run in the population, presents a non-monotonic response. Initially for inoculation percentages less than 10% it increases, and then it starts decreasing. Again it can be observed that the disease transmission ends way faster for low transmission probability or/and high percentages of inoculated people.

```
Investigate the matter of 'herd immunity': if
enough people are vaccinated, then some people
who are not vaccinated will still never get
sick. Let's say you want to have this
probability over 95 percent. Investigate the
```

percentage of inoculation that is needed for this as a function of the contagiousness of the disease.

The experiments show that a high inoculation percentage more than 90% of the population, or a very low disease transmission probability can assure that more than 95% of the population will stay healthy throughout the duration of the experiments.

LESSONS LEARNED

Trying to model a real life phenomenon which is governed by many parameters is a tedious task. Although the program can yield an explicit result, the user should be really careful about the assumption he makes. Extracting meaningful information from databases, requires the visualization of the results. Familiarity with visualization packages such as matplotlib is required, for high quality outcomes. Working with data frames and plotting tools, is a useful skill that the author would like to develop in the following months.

Unix powerful operating system which allows to the user to automate complicated tasks. An IDE with a graphical user interface is preferable when the program developed includes a large number of files. It should be noted that the compiling scripts in a unix environment is significantly faster compared to using a windows system.

Object oriented programming is the only viable way to create error-free programs. Although there is a steep learning curve, there is a significant advantage in productivity when using object-oriented programming. After attending the Introduction to Scientific Programming, and completing this project, the way I am writing code, changed radically.