Infectious Disease Simulation-New Flu Strain

Harry Singh Kang, hsk697, hkangt

12/7/2024

1 Introduction

A new strain of flu has just made its way to the United States. An unnamed student was traveling abroad and got sick on way back, bringing with him the new disease. As he settles back down into his college routine, he unknowingly spread the virus to others. This paper explores how this disease could possibly spread throughout the student body through experiments gradually building up the accuracy and fidelity of a disease simulation. First, the disease's effects will be explored on the individual level, then on the small community/population level. Afterwards, the simulation will evolve from direct contraction from dorm-neighbors to random interactions with anyone within the UT student body.

2 Experiments and Results

In this section, the scope of the experiments will gradually increase from analyzing small scale impacts of the disease to how it will impact the student body.

2.1 Person

First, to see how the disease will impact an individual person (for example the person who first contracted it), simulation "sim1.cpp" was used to see how a flu lasting 3 days would affect the state of the individual as well as some other sanity tests. The setup is as follows (Figure 1):.

```
Person joe;
Disease flu(3,1.);
for ( int step = 1; step<=10; ++step ) {
  if(step==3){
    joe.infect(flu);
  };
  cout << "On day " << step << ", Joe is "<< joe.status_string();
  if(joe.status_string()=="sick"){
    cout<<"("<<joe.get_days_sick_left()<<" sick days left to go)";
  };
  cout<<"\n";
  if (joe.is_recovered())
```

```
    break;
  joe.one_more_day();
};
```

Figure 1: Snippet of "sim1.cpp" Logic Minus Comments and Sanity Tests

The Person and Disease objects create the instances and relevant information. The "infect()" function starts the sickness counter built into the Person class if a random number is below the disease rate of transmission (seen in the first two lines of Figure 8). "One_more _day()" updates the day for the person and takes the sick counter and decrements it until it is 0, thereby indicating recovery from the flu (Figure 2).

```
day=day+1;
if (daysleft==0){          /
  status="recovered";
  statuscount++;
};
if (daysleft>0){
  status="sick";
  daysleft=daysleft-1;
};
```

Figure 2: Snippet of "one_more_day()" Logic Minus Comments

The output of this experiment including the sanity tests (code not provided above) does indeed show Joe being sick for exactly three days after getting infected. Once recovered, his state/status changed accordingly. This fulfills the requirement set by the first sanity test. The next test evaluated if he could get sick again after recovering fully by infecting the same "recovered" person and updating the day. The result displayed the contrary, thereby supporting the accuracy of the experiment. The last sanity test involved a number of individuals/persons, and they were subject to the flu at 0.5 rate of transmission. The test correctly calculated around half of them to contract the flu after getting infected using a random number generator and seeing if it was less than the transmission rate.

2.2 Population

The justification behind this second experiment was to test a small population, comprising of a vector of several Person classes. In "sim2.cpp" (Figure 3), the status of the small population over several days is investigated given that there is no contact among individuals. The driving method behind this simulation

is "random_infection()", which infects random people within the population (Figure 4). Again, the flu here lasts 3 days and the simulation stops when there are no sick left. There is also another sanity test simulating a population of 100 percent vaccinated individuals.

```
Population population(15);
Disease flu(3,.9);
double initial_infect=.4;
for (int iter=1; iter<=10; ++iter){
  if (iter==3){
    population.random_infection(flu,initial_infect);
  };
  int countnuminfected=population.count_infected();
  cout<<"In step "<<iter<< " #sick:  "<<countnuminfected<<" : ";
  for (Person& indiv:population.get_people()){
    if (indiv.status_string()=="sick"){
      cout<<"+ ";
    }else if(indiv.status_string()=="susceptible"){
      cout<<"? ";
    }else{
      cout<<"- ";
    };
  };
```

Figure 3: Snippet of "sim2.cpp" Logic Minus Comments
(Outputs status of a population of 15 for a number of days)

```
srand(static_cast<unsigned>(time(0)));
set<int>indices;
while (indices.size()<numpeopleinfected){
  int index=rand() % people.size();              \
  indices.insert(index);
};
for (int i:indices){
  Person& per=people[i];
  per.infect(fever);
};
```

Figure 4: Snippet of "random_infection()" Minus Comments
(A vector of random indices is created so that random individuals within the population are infected.)

The output to "sim2.cpp" does indeed show a population of 15 to be initially infected at a .4 transmission rate after day three and staying sick for three days after. The population size was kept small to fit the output syntax within the terminal window. After day 3, as expected, 5 to 6 random individuals got sick and were represented with "+". They remained sick for the rest of the duration of the sickness (3 days) without getting anyone else sick since contacts wasn't encoded in "sim2.cpp." Those that were still susceptible had "?". The ones

that recovered after getting infected had the "-" demarcation. Every day/step had the same total number of sick, susceptible, or recovered, thereby passing the built-in sanity test that ensured no double-counting or wrongful infection in the small population. The other sanity test in "sim2.cpp", evaluating the number of individuals vaccinated with an 100 percent vaccination rate, also passed and showed that all 20 individuals were indeed vaccinated. The setup for "random_vaccination()" used in this second sanity test was similar to that of "random_infection()" from above.

2.3 Contacting Neighbors

Moving away from the contactless simulations, now it's time to add fidelity and see what would happen if the sick individuals could only infect their direct dorm neighbors each day. A snippet of the implementation is as follows (Figure 5):

```
for (int iter=0;iter<20;++iter){
    if (iter==0){
      firstperson.infect(flu);
    } else{
        for (int p=0;p<people.size();p++){
           if (people[p].status_string()=="susceptible"){
               if (p+1<people.size() && people[p+1].status_string()=="sick"){
                   people[p].infect(flu);
               };
           if (p>0 && people[p-1].status_string()=="sick"){
            people[p].infect(flu);
           };
        };
      };
    };
    ...
    population.one_more_day();
  };
```

Figure 5: Snippet of "sim3.cpp" logic minus comments
(Initially the first person is infected. Then, every day each person comes into contact with only their neighbors in front and back.)

Several sanity tests were conducted using this framework but varying the starting index or rate of transmission: 1.) Seeing if there are three sick individuals the day after the initial person is afflicted (two if the starting person is at the front or the end of the hypothetical dorm hallway). 2.) Seeing if the simulation runs for around the same number of days as the population size. 3.) The effect of changing p, the chance of disease transmission, to 0.5.

"Sim3.cpp" was tested with small populations of 10, since the neighbor algorithm forced the sanity tests to iterations the same size as the population (to reduce computation time for sanity). Sanity test 1 was affirmed, showing that if the 0th person out of 10 people is infected initially, the next day only 2 people are infected. If the person was in the middle of two neighbors, then 3 are

infected the next day. The next sanity test was also affirmed, demonstrating that infecting the 0th person (with a transmission rate of 100 percent) out of 10 leads to the simulation running for about the same number of days as the population size (it ran to 11 days in the output). The output of the last sanity test evaluated the impact of decreasing the transmission rate to 50 percent. The disease lasted for 3 days on one such run, which is to be expected because contacts are limited to just neighbors and a smaller chance of getting infected leads to less people actually getting infected.

2.4 Introducing Vaccination

Seeing the effects of smaller rates of transmission, now it's time to see how vaccinations in the population (suppose dorm hallway) can affect the spread of the flu. In this next experiment, randomly chosen individuals out of the population were vaccinated (Figure 6) to see how the flu could spread from neighbor to neighbor. "Sim4.cpp" has a similar main setup and parameters as the previous section (rate of transmission was reset to 1.0), and the output displays the results that vaccination can have on the neighbor-to-neighbor infection algorithm/experiment.

```
srand(static_cast<unsigned>(time(0)));
set<int>indices;
while (indices.size()<numpeoplevaccinated){
  int index=rand() % people.size();
  indices.insert(index);
};
for (int i:indices){
  Person& per=people[i];
  per.vaccinate();
};
```

Figure 6: Snippet of "random_vaccination()" Minus Comments

The output showed that introducing random vaccination into the small population of 10 reduced the duration of the disease spread. Compared to the first two sanity tests of the last section, introducing vaccination decreased the disease duration in the population to 6 days (compared to around 11 days previously). Evidently, if only neighbors can infect each other, vaccinating some of them prevents the spread of the flu to the people beyond that individual. This model is fairly unrealistic since people can interact with whomever in their day-to-day, not just their neighbors. This neighbor-only contact model creates a barrier/wall in which only one side of the vaccinated person could get infected.

2.5 Random Contacts and Herd Immunity

To bring the model closer to reality, random interactions were then encoded such that each person met up to 6 random people each day. The population

size was finally increased to 50,000 to represent the student body. The logic is displayed as follows (Figure 7):

```
for (int iter=0;iter<220;++iter){
  if (iter==0){
     nthperson.initial_infect(flu);
  }else{
     int indexiter=0;
     for (Person& indiv:people){
      while((indiv.get_touch_counter()<6{
         int rindex=rand()%50000;
         if (rindex!=indexiter){
                indiv.touch(people[rindex],flu);
         };
      };
      indexiter=indexiter+1;
     };
  };
  population.one_more_day();
  int countnuminfected=population.count_infected();
  if (iter>=0){
     cout<<"On iteration "<<iter<<" there are "<<countnuminfected<<"
infected."<<endl;
  };
  for (Person& indivs:people){
     indivs.reset_touch_counter();
  };
};
```

Figure 7: Snippet of "sim5.cpp" Logic Minus Comments
()

As seen above, each day, each person meets 6 randomly indexed individuals out of the population. The work-engine of the previous experiments "infect()" is still used here, but through another function called "touch()" that helps track the number of interactions each person has per day(Figure 8).

```
Disease s=d;
if (touchcounter<6){
  touchcounter=touchcounter+1;
  if (i.get_touch_counter()<6){
    i.update_counter();
  };
  if(status == "susceptible" && i.status_string()=="sick"){
    infect(s);
  };
  if(status == "recovered" && i.status_string()=="sick" &&
 statuscount<i.statuscount){
    infect(s);
  };

  if(status=="sick" && i.status_string()=="susceptible"){
    i.infect(s);
  };
```

```
  if(status=="sick" && i.status_string()=="recovered" &&
 i.statuscount<statuscount){
    i.infect(s);
  };
};
```

Figure 8: Snippet of "touch()" Minus Comments
(Each person can touch up to 6 randomly chosen people per day. Any combination of sick and susceptible will lead to a possible infection. "Statuscount" is to be ignored for this sim since it tracks mutations for the next section)

Once this was implemented, "sim5.cpp" was used to observe the spread of the disease lasting 10 days among a population of 50000 as a function of two variables (encoded as a 2-D grid of for loops): the vaccination percentage and transmission rate. The output was a matrix of logs showing how long the disease stayed in the population and how many it infected given a permutation of vaccination and transmission rates. The table below summarizes the findings:

	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.0	10	51	31	26	23	20	16	18	18	17	16
0.1	10	58	35	26	23	21	19	19	15	17	17
0.2	10	60	35	28	25	23	20	20	18	17	17
0.3	10	62	36	31	26	23	21	20	20	18	18
0.4	10	74	43	34	28	27	24	21	20	19	19
0.5	10	92	47	38	31	27	25	23	22	21	20
0.6	10	112	61	47	39	30	26	24	25	23	20
0.7	10	10	72	53	39	38	29	27	29	25	22
0.8	10	10	125	71	57	51	45	38	32	32	28
0.9	10	10	38	132	91	65	60	55	56	47	43
1.0	10	10	10	10	10	10	10	10	10	10	10

Table 1: Duration of Disease Spread in Days Depending on Transmission Rate (columns) and Vaccination Rate (rows).

As shown above, the combination of high transmission rate and low vaccination rate led to the disease passing through the population the quickest (least number of days). Also, the maximum number of people infected on a particular day peaked at around 45000. Contrastingly, simulations with high vaccination rates and low transmission rates led to some of the highest durations of sickness, up until a certain point. There, the maximum number of people infected on a day was only around 1000. These findings make sense, because in the latter scenario, fewer people in total get infected but the long period of sickness (10 days) allows for transmissions late into one's sickness. If vaccination rates were low and transmission rates were high, their sickness would spread on an earlier date to more people and thus everyone would heal sooner as well.

Interestingly, the above data hints at another phenomenon called "herd immunity," seen in the bottom left of the table. At a certain vaccination level (high vaccination rate), the disease would not spread even though there were individuals who were not vaccinated within the population. This is evident in the floor number of 10 in the table even when vaccination rate is not 1.0 or 100 percent. It's also evident that the boundary for this phenomenon is diagonal, suggesting that increasing transmission rates requires more of the population to be vaccinated for the population to have "herd immunity". Thus, there is a negative relation between the two and a positive one for vaccination rates.

2.6 Implementing Mutations

A disease can mutate. Flu has a strain nearly every year that doctors continue to battle. Similarly, for these experiments, the flu introduced to campus will mutate after a certain number of transmissions. Transmissions are counted every time someone is infected, and after a certain number, the variant of the flu will update. This will impact how people will get sick, since any variant above what the individual is recovered from can still get the individual sick. As seen above with the logic in "touch()" (Figure 7), statuscount is needed to see an individual's immunity to a certain variant (it is updated once someone recovers from a strain), and variantcount is implemented to identify and update the variant of the flu. A few things change in the logic, but here are the major ones in "infect()" (Figure 9):

```
if(r<s.get_transmission() && statuscount<variantcounter){
  daysleft=s.get_days();
  transmissioncounter++;
};
if (transmissioncounter>=50000){
  if (variantcounter<3){
    variantcounter++;
  };
  transmissioncounter=0;
};
```

Figure 9: Snippet of "infect()" Minus Comments
("Statuscount", or a person's immunity, being lower than the variantcount will trigger an infection, each successful one of which will counts as a transmission. Once transmissions increase a threshold, the flu mutates and counter is reset.)

Running "sim6.cpp" several times by varying the variant threshold in "infect_lib.cc" helps determine how the introduction of mutations affects the duration of the disease spread. The results from that experiment show that increasing variants of the flu led to the disease staying in the population for a couple more days (24 days for one variant and 27 days for 4 variants at a low vaccination and transmission rate). Conducting a run with higher vaccination rates and lower transmission rates than the runs shown above enhances this distinction. With this implementation, the experiments brought the fidelity of

the model and simulation to realistic results. We can see the results on the student body an infection brought over by a student can have considering random contacts, vaccinations, mutations, etc.

3 Conclusion

Through these experiments, the model of a new flu on campus was studied. Going from a low impact study of the disease on a singular person with no contact to a full student body population with several random interactions throughout the day with infected and vaccinated individuals and a possibility of multiple variants. Several interesting takeaways were drawn from the experiment such as the relation between vaccination and transmission rate and the disease spread duration and how herd immunity can protect those that are vaccinated. High vaccination and low transmission rates were found to lead to long lasting diseases but less people affect, while the opposite was true for the other combination. Herd immunity was observed once vaccination rates got high enough but still under 100 percent and seemingly increased in appearance with low transmission rates. There are several ways to improve the model such as implementing more realistic phenomenon like incubation periods, more robust people-interaction models, geographic considerations, etc. For now, this behaves well to simulate local campus epidemics.

4 References

N/A