Final Project Documentation

Ian Weih-Wadman

December 14, 2020

Algorithm Description

The algorithm is based on two main classes, the person and the environment. A person represents an individual within the city being modeled, and has a variety of states, alive/dead, infected/healthy, infectious/noninfectious, symptomatic/asymptomatic. After each 'day' within the model each person is updated. An environment consists essentially of a list of people and an update function describing how the people in that list might infect each other on a given day. The basic algorithm for the environment update is as follows. Let D be the parameter representing how closely the people in this environment interact, and I the basic infection rate of the disease. Let P_i be the probability that person i visits the environment on this day. Let U be the probability that any given healthy person is not infected at the end of the update.

For each infectious person i, multiply U by $1 - DIP_i$.

For each healthy person j, there is a probability $(1 - U)P_j$ that they become infected today.

This assumes independent random chances of infection and equal levels of interaction between all people who visit the environment on a given day.

To determine whether each living but infected person dies on a given day, each person has an integer called overallHealth. When infected, there is a fixed probability that overallHealth decreases by one on every update. If it reaches zero, the person dies. The value of overallHealth is not initialized to the same value for each person, but ranges from 3 to 10. A person with overallHealth 4 or less is considered vulnerable, which may increase their likelihood of staying home from public spaces. In the above algorithm, this corresponds to a reduction in P_i for that person. Similarly, person exhibiting symptoms will be less likely to go to a public space, also reducing P_i.

An infected person has a random probability of becoming symptomatic at each update. The infection rate, symptomatic probably, and loss of overall health probability are the three parameters of the disease.

Command Line Inputs

The program takes as inputs the names of three files, one describing the parameters of the city and the disease, one giving a list of parameter sets corresponding to different response plans to be tested, and a third empty file for output.

City and Disease File

The parameters listed in this file are, in order, population, averageFamilySize, averageWorkplaceSize, hospitalBedsPer5KPeople, incubationDays, recoveryDays, deathRate, infectionRate, symptomRate. The first five must be positive integers, the last three real numbers between 0 and 1.

Plan File

The first parameter in the files must be the the number of trials different trials that the program should run, and the second should be how many times each trial should be repeated. The rest of the files should consist of a number of parameter sets equal to the number of trials to run, and each parameter set must consist of, in order, workIfSick, workIfVulnerable, workDistance, storeIfSick, storeIfVulnerable, storeDistance, socialIfSick, socialIfVulnerable, socialFrequency. The IfSick, IfVulnerable parameters correspond to the three different environments which a person has an option not to visit, and the effects that symptomatic sickness and vulnerability on the value P_i in the algorithm above. workDistance and storeDistance determine the value of D is the algorithm above for the workplace and business environments. Finally, socialFrequency determines the likelihood of the SocialGroup environment meeting on a given day. All of these parameters must be real numbers between 0 and 1.

Output File

Lastly, an output file is needed. Each line of output will consist of all the input values for one of the plans followed by the average number of deaths resulting from that plan.

Sample Files Included

The folder Data in the submission contains a PlanData file with 25 different response plans each to be sampled 10 times, three different CityDisease files with different parameters to illustrate the various effects of different plans in different situations, and sample outputs resulting from the first two of the sample CityDisease files.

Classes Included

Plan

The Plan class stores a private vector consisting of the parameters for the current trial as read from the plan file. Its only functions are getters for the parameters of the plan and its constructor. The constructor throws an invalidInput exception if one of the parameters passed to the constructor is not between 0 and 1. Since these values cannot be changed after initialization, and throw an exception if initialized incorrectly, we can consider this a class invariant and assume that the parameters of a Plan object are always within the valid ranges.

Disease

The Disease class stores private members for the parameters of the disease, as read from the city and disease file. Its only functions are getters for the parameters of the disease and its constructor. The constructor throws an invalidInput exception if one of the parameters passed to the constructor is not in the acceptable range for that parameter. Since these values cannot be changed after initialization, and throw an exception if initialized incorrectly, we can consider this a class invariant and assume that the parameters of a Disease object are always within the valid ranges.

Person

The Person class has as private members integers representing overallHealth and days since infection and bools representing alive/dead status and symptomatic/asymptomatic status. It also holds references to objects of the Disease class and the std::mt19937 class. Prior to infection, the counter for infection days takes the value -1.

Public functions:

void ImproveCondition: This function increases the person's overall health value by 1, and is used by the hospital environment to represent medical intervention counteracting the effects of the disease.

void InfectProbability: This function takes a double representing a probability of infection, and using the reference to a random engine determines whether or not the person became infected. If they did, sets their days since infection to 0.

bool Update: If the person is dead, returns false immediately. Increments days since infection if the person is infected. If the incubation period is over but the person is not recovered, there is a chance they will become symptomatic and there is a chance they will lose 1 overallHealth. If overallHealth is 0, this function sets their state to dead, and returns true indicating that the person has died on this day. Otherwise, returns false.

Environment

Has a list of pointers to people and a reference to a disease as protected members. Has the functions Update and Duplicate as pure virtual functions, to be implemented in the various classes derived from the environment class, a constructor which takes a reference to a disease, and a function which takes a pointer to a person and adds that pointer to the list of members. It also has a function that gets the size of the list of visitors.

Family

Derived from environment. Besides the disease passed to the environment base class, the constructor for Family takes not arguments. The update function computes the infections using the basic algorithm described in the first section of the documentation, with the distance parameter always set to 1. Distance parameters of other environments are thus defined in reference to 1 being the distance parameter of living in the same home. The duplicate function returns a pointer to a environment with the same constructor parameters as this family.

PublicSpace

Derived from environment. Besides the disease passed to the environment base class, the constructor takes parameters for the likelihood of staying home if symptomatic, the likelihood of staying home if vulnerable, and the distance parameter. Within my current city design, there are two types of publicSpace, workplaces which are slightly smaller and each person belongs to only one of, and businesses which have many visitors and each person visits multiple of. In a realistic parameter set, workplaces should have a much higher chance of infection, as reflected in the distance parameter. The duplicate function returns a pointer to a environment with the same constructor parameters as this PublicSpace.

SocialGroup

Derived from environment. Besides the disease passed to the environment base class, the constructor takes parameters for the likelihood of staying home is symptomatic, the likelihood of staying home if vulnerable, and the frequency with which the social group meets. Social groups only meet on certain days, with frequency given by the frequency parameter. If they do meet, the algorithm is identical to a publicSpace but with distance parameter set to 1 so that the infection probability is as high as a family. The duplicate function returns a pointer to a environment with the same constructor parameters as this SocialGroup.

Hospital

Derived from environment. Besides the disease passed to the environment base class, takes an integer representing the number of beds available in the hospital. Any member of the hospital environment who is symptomatic and has overallHealth sufficiently low will visit the hospital environment if there are beds available, which increases their overallHealth by 1 using the person::improveCondition function. When this happens the available beds for the day is reduced by 1. The duplicate function returns a pointer to a environment with the same constructor parameters as this Hospital.

City

This class has as private members a list of uniqueptrs to environments and a list of uniqueptrs to people, as well as a reference to an object of the class Plan and a reference to a Disease.

Public Functions:

int Update: This function calls the update function for each environment in the city's list of environments, and then calls the update function for each person in the list of people. The update function of the person class returns true if the person died that day, so during this step the city also counts the number of deaths and returns that value as an integer.

void Initialize: This function creates the list of smart pointers to people and the list of smart pointers to environments. The people are generated directly using the Person class constructor. The environments, on the other hand, start with one environment of each type, initialized using the parameters of the Plan object held by the City. These objects are then copied using the duplicate function of the environment class, to produce the list of environments. The duplicate function is used instead of a copy constructor because the duplicate function returns a pointer to the base class rather than the derived class so that can be placed in the list.

Manager

This class is responsible for reading the data from the files, initializing the cities for the trials, recording the results of the trials, and writing them to the output file. Its constructor takes the three filenames described in the inputs section. The only other function that the manager class has is RunExperiment, which initializes a city for each trial, updates each city 200 times to allow time for the outbreak to occur, then records the number of deaths resulting from each trial, as returned by the Update function of the City class. If the samples parameter is greater than one, the number of deaths is averaged over all the samples before being written to the output file. The RunExperiment function catches the exceptions thrown by the Disease and Plan classes and prints a message to the console if invalid parameters are passed.