Erik Nelson & Jiyoung Chang

Summer 2021

Modeling COVID-19 in Small Colleges Extended Research Write-up

Table of Contents:

Introduction (Overall Design Idea)

Modeling COVID-19 spread in small colleges was a project that was held in the summer of 2020 for the purpose of creating an agent-based model on the network of a small residential college to observe and analyze the spread of COVID-19 on the campus model. Our task was to continue on with the project and furthermore add vaccines as an additional intervention in the simulation. In order to carry out this task, we decided it would be more preferable to recreate the code for the agent-based model – from the previous project – in our own way, and then add the vaccine interventions into the model. Hence, we began with designing the overall structure of the agent-based model.

Agent-based models are computational simulation models that involve the actions and interactions of many discrete "agents," which are autonomous, decision-making entities. In agent-based modeling, a system is modeled as a collection of agents and these agents can be anything that are relevant to the system – for example, they may be people, organizations, households, or companies. In the case of our agent-based model, the system is the small residential college and the agents are the faculty and students that interact with one another in the campus, which would trigger the spread of COVID-19 in the college model. The agents in an agent-based model have internal properties, or attributes, and defined behaviors – some of which are determined by environmental variables. These affect the behavior and interactions between agents, allowing us to simulate and evaluate the effects that the agents can have on the system.

To create our agent-based model, we initially designed our code so that we would have separate files for code related to agents and their attributes (CovidAgents.py), code involving spaces and environmental variables (spaces.py), and code for initializing the system and running simulations (main.py), respectively. In the process of following this initial design, we also decided to have separate files for global constants that we use throughout the simulation (global_constants.py) and for the code creating agents' schedules (Schedule.py), to avoid confusion and keep the code organized. Thus, the overall design of the code for our agent-based model consists of separate files for the global constants, agents, spaces, assigning agents' schedules, and a main file for initializing the model and running the simulation.

Main File (main.py)

The main file contains the primary functions needed to run the simulation and analyze its results. To briefly describe the execution process of the main file, it (1) receives input values from the user regarding COVID variants, interventions, and the number of simulations, (2) initializes and creates the model and agents based on the input values, (3) spreads the infections among the agents and continuously update their infection states in accordance with their behavioral pattern, and once all of the simulations are finished running, (4) it combines the data and creates graphs to visualize the results.

If we execute the main.py file, the user is asked several questions in order to collect the required input values. The questions ask the user to enter input values regarding the covid variant that will be used in the simulation, the interventions that will be applied to the model and agents, and the number of simulations the user wants to run. For the covid variant input, the user has to enter either of the three options: "A" for the Alpha variant, "D" for the Delta variant, and "O" for another variant that the user can create themselves. In terms of the inputs for interventions, the user is asked to answer either Y (yes) or N (no) regarding whether they want to add any of the vaccine, face mask, or screening test interventions; specifically for the vaccine intervention the user is also asked to enter the percentage (an integer between 0 and 100) of student and faculty agents they want vaccinated in the model. These inputs are stored in pickle files, which allow us to access the inputs - which we received through the main.py file - from other files.

Based on the information collected by the user input, the agents and spaces of the model are created and initialized by the "initialize" function in the main.py file. In this process, a certain number of agents – defined in the global_constants.py file – will be assigned to an infected state as initially infected agents that will be spreading the virus. Also, each agent is assigned a weekly schedule attribute, in which they are designated to a certain space at a specific time and day of the week, following their other properties. For example, an agent who is a student, lives on-campus, and is a STEM major will be assigned to dorm spaces and classrooms in the STEM buildings whereas a faculty who lives off-campus and is in the art department will be assigned to off-campus spaces, and offices in the academic buildings for art.

After this initialization step for the model and agents is finished, the simulation starts running, moving agents around the campus spaces following their weekly schedule attribute and updating the infection state of susceptible agents – using the "update" function in the main.py file – on the basis of whether they were in the same space as an infected agent, thus exposed to the virus; for agents that were already infected, it keeps track of the number of days they have been infected and updates their infection state to "R (recovered)" once it has been 10 days since their infection. This process is repeated until all the simulations are finished. In this stage, we made use of the multiprocessing technique, which allows us to run multiple simulations in parallel, thus saving time and increasing efficiency in terms of running the simulations.

Once all the simulations are completed, the "observe" function of the main.py file collects the data of all the simulations and creates visualizations to clarify the information in the results. There are four main results we gain from the simulation: the total number of infected agents, the number of daily exposures, the number of agents in each infection state (this is referred to as the SEIR graph), and the number of agents that were exposed in each space.

Agent File & Agent Class (CovidAgent.py)

To explain in more detail what we have in the CovidAgents.py file, we first have the Agent class, where the 'initialize' method creates a list of all the agents we need for the simulation, and each of those agents are given various attributes. Agents' attributes are either related to college, disease, or interventions for the pandemic, as our model is based on the network of a small college and is used to simulate the spread of COVID-19 and how different kinds of interventions can affect the spread.

College-related agent attributes include their type(whether an agent is an on-campus student, off-campus student, or faculty), division/major, and a weekly schedule that assigns them to certain spaces on campus at a specific time and day within the week. Some examples of attributes that are disease-related are the agent's infection state (whether they are susceptible/exposed/infected/recovered), the space where they were exposed to an infected agent, and the number of days in which the agent has been in that state. Lastly, the agent attributes related to interventions of the pandemic are the agent's vaccination status, their compliance to face mask interventions, and the list of their screening test results (either Positive/Negative/Not tested) throughout the simulation.

Agents also have the two risk multipliers as attributes for both the vaccine and face mask interventions – one that affects the possibility of the agent getting infected by an infected agent when they are in a susceptible state, and the other affecting the possibility that the agent will spread infection to other susceptible agents when they are in an infected state. Thus, each agent has four risk multiplier attributes that impact their possibility of getting infected or spreading infection depending on their vaccination state or compliance for face mask interventions.

In the 'initialize' method, agents are assigned attributes and are given default values for the attributes. After that, for certain agent attributes that need to be initialized before the simulation begins running, a randomly selected proportion of these agents will be assigned new values – that are different from the default – for those attributes. The value of the proportions can be different for each attribute and these values are either set as a global constant or given as an input by the user. For example, the proportion of the initially infected agents is a global constant whereas the value of the proportion of vaccinated agents will be an input entered by the user. When assigning agents as initially infected, we get the proportion of initially infected agents from the global_constants.py file and randomly select the following proportion of agents, changing their infection state from the default value S (susceptible) to one of the three infected states: Ia, Im, Ie (Infected asymptomatic/mildly symptomatic/extremely symptomatic). For the proportion of vaccinated agents, we receive a proportion between 0 and 1 as an input from the user and bring that input value from the main.py file to the CovidAgents.py file using the pickle library. Then again we randomly select the following proportion of agents and change their vaccination status from False to True. Such attributes that must be initialized prior to running the simulation include the agent's type (on-campus student/off-campus student/faculty), vaccination status (True if vaccinated, False if not), face mask compliance (compliant/non-compliant), and division (STEM/Humanities/Arts), etc.

Other than the 'initialize' method, there are various methods and functions for Agent class objects in the CovidAgents.py file. The methods get_division_index and get_available_hours get the information about the value of an agent's attribute – in this case, the agents' division and the empty values in their schedule attribute - and returns it in a more organized and notable way without changing the value of their attributes. On the other hand, methods and functions such as change_states, initialize_leaves, screening_test, walk_in_test, and return_screening_result changes or assigns new values to agents' attributes.

The change_states function is called every day at the end of the day in the simulation to change the infection, or SEIR, state of agents depending on how many days they have been in their current state. The initialize_leaves function assigns agents to specific leaves for certain types of spaces that have leaves (the dining hall, library, gym, social space, and office), as we assume that an agent would always go to the same leaf of the space regardless of what day and time they visit the space.

The screening_test and walk_in_test functions check the agents' SEIR state, and taking into consideration the false positive/negative rates of the test, they get each agents' result for the screening test, which will be either 'Positive' or 'Negative'. Then they append this to the agent's screening_result attribute, which is a list of all the screening test results of the agent, thus adding a new value to the agent's attribute. The return_screening_result function then checks the most recent screening test result of all the agents that were tested, and depending on whether the result is positive or negative, the agent's bedridden attribute will become True or False, indicating that the agent begins quarantine.

Spaces File & Space Class and Subclasses (spaces.py)

The researchers of the original research paper came up with a list of extremely common spaces for liberal arts college campuses. These spaces were dorm buildings, academic spaces, social spaces, office spaces, a library, a gym, and a dining hall. Additionally, they came up with a "large gatherings" space to represent various large gatherings between students that occur on campus and an "off campus" space to represent the interactions that agents who live and spend significant time off-campus will have.

In order to respect spaces in general, we created a Space class for attributes or methods that all spaces in the model share. All spaces should be able to be closed, return a list of agents in the space with a specific SEIR state, return the probability of getting infected in that space (core), spread COVID-19 in the leaves of the space, spread COVID-19 in the core of the space, and spread COVID-19 in both the core and the leaves of the space. These functions are universal to all spaces, and thus they are in the Space class.

However, spaces have different properties that differentiate them. For example, spaces have different amounts of leaves and different subdivisions of the space (like how there is a STEM office space, a humanities office space, and an arts office space). Furthermore, we made a design decision that the agents in a space class should be the agents in that space *at a given day*[1] *and a time*, based on the agent's schedule. To accomplish this, we made each space class also take in a day and time attribute when initialized[2], so that each instance of a space class is really a space at a given day and time. This initialize function also is used to set up the c_v, the capacity of a space and the r_v, the risk multiplier of a space. These values are largely gathered from global_constants.py, but it is class-specific.

Additionally, most spaces have a leaves field, which is a list that contains every leaf that is underneath that space. Each leaf is an instance of the Subspace class, which is initialized with a parent space, the subspace's c_v, and the subspace's r_v. Additionally, all subspaces can be closed, return a list of agents active in the subspace, return a list of agents in a certain SEIR state, return the infection probability for the subspace, and spread infection in the subspace.

Finally, each individual space class has a method, *assign_agent* or *assign_agents*, that gives us a way to assign an agent of the model to a space at a given day or time. Luckily, the assignment of that agent to the appropriate subspace is done through other means, particularly the **agent file**, so we do not need to do much work in this file to determine which leaves agents should be placed into.

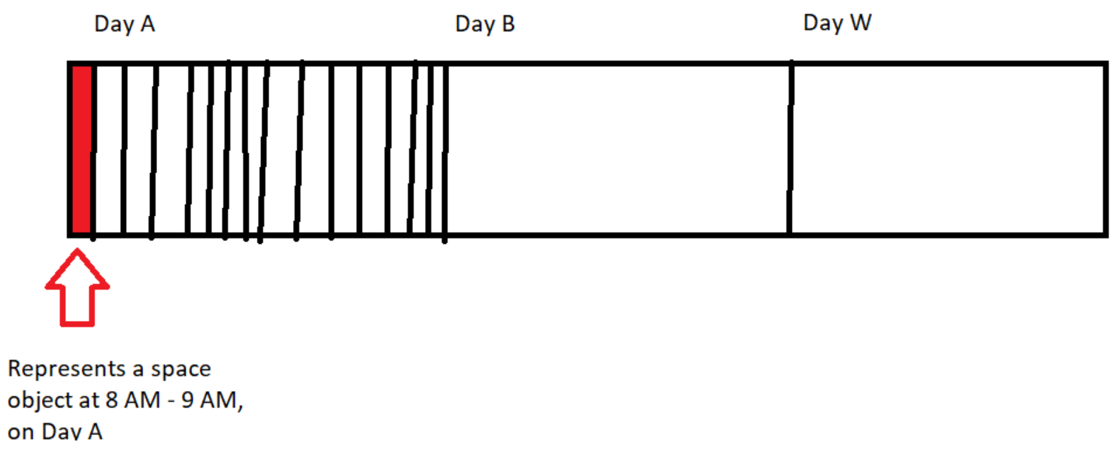
More details on the spaces file can be determined by looking at the relevant documentation in the codebase.

[1] Note that day is referencing the day types ("A", "B", "W", "S")
[2] The **Dorm class**, since students will have a consistent dorm room for the entire semester, does not follow this pattern. Instead, the dorm class is simply initialized by the size of the dorm hall (small, medium or large). Similarly, the **Large Gatherings class** does not need to be initialized with day and time since simulations of large gatherings will always happen at the end of each week.

Schedule File (Schedule.py)

The schedule file is used to give every agent a schedule, by assigning agents to spaces for each time and day that they are active on-campus. The specifics of which days and times certain agents are active on-campus are described in the research paper that this codebase is designed after, so we will not go into detail on that information here.

The main function in the schedule file is the create_spaces function. This function takes a space as a string (for example, "Library") and then creates an object of that class for each hour, day, and division (if applicable). The spaces are then returned in a list that is structured like the following image:



However, the dorms and the academic spaces have special qualities that require separate creation functions for them. Since dorms do not depend on time, we just create each dorm space for the entire semester and then return all of the dorm spaces in a single list. On the other hand, academic spaces require a more complex list structure than the one pictured above. First, the list is split up by division (Index 0 is STEM spaces, index 1 is humanities spaces, and index 2 is arts spaces). Then, the list is split up by just Day A and Day B, because classes are not held on the weekend. The rest follows the diagram, but this distinction required a separate function than the generic create_spaces function used for all other spaces.

Now, the rest of the functions in this file are for assigning agents to spaces. For information on the order that agents should be assigned in, please refer to the original research paper.

For assigning agents to dorms, we first create a list of on-campus students and a list of off-campus students. Then, for the on-campus students, we randomly selected a dorm building for each student and try to place them in it. If we cannot place the student in that dorm building, we select another dorm building until we find one that has an open room, either an open single or an open double, to place the student. For off-campus agents, we simply block off time in their schedule attribute that they will not be on campus.

The most complicated part of this code is the process by which students and faculty are assigned classes. First, we start with faculty and go through every academic building for each hour and day. For each building, we calculate the number of classrooms and assign a unique faculty teaching in the same division as the building (STEM, humanities, arts) to each one, as long as there are no scheduling conflicts. Then, we look for faculty who have been assigned to two classrooms already and remove them from future consideration. For students, we first randomly select four unique hour and day combinations that their classes will be set in. Then, we first assign each student to their two division classes to ensure that each student can be in classes for their division. Then, we assign each student to classes in two randomly select divisions to give each student a schedule of four total classes.

We also wrote a assign_dining_times function which relies heavily on a secondary assign_meal function. assign_meal works by taking in the hours the dining hall is open for a meal (breakfast, lunch, or dinner) and then randomly selecting an hour in that period for the agent to eat. It also makes use of the agent's get_available_hours function, which looks to make sure that the scheduled meal time is not a time that is occupied with anything else.

Finally, there are two more functions for assigning agents to spaces: assign_gym and assign_remaining_time, which cover the rest of the possible spaces an agent can be in. The code for these spaces is pretty straightforward with the original research paper, so we will not go into further detail here.

Extraneous Information (global_constants.py and Python packages used)

The global_constants.py file holds all the constants that we may want to use in other places in the code. Many of these constants are described in the original research paper, so more detail on specifics can be found there. Thus, the global constants in the file are all constants that were decided when the paper is written. We, the programmers, have not decided any of these ourselves. The constants are also broken out by type to make navigation of the code easier.

We also used several different Python packages in this program. Many of them have been described above, but here is some additional information about the packages:

- We used the **pickle** package to save "constants" that were set by user input. These include COVID variant information, interventions, and percentage of the population vaccinated. This data is stored in a pickle file in the pickle directory, and then retrieved elsewhere in the code from the directory.

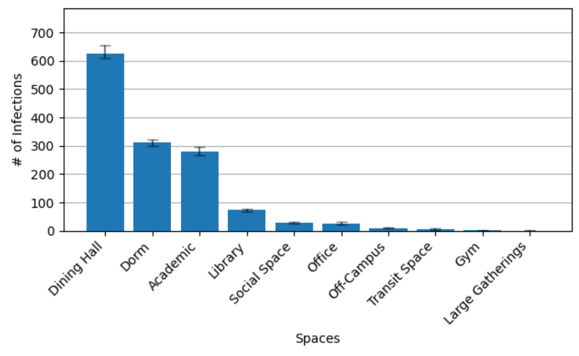
- We used the **random** library for all of the randomness required with infection spread.
- We used the **matplotlib** library for a graphing tool. There was not a specific reason why we chose this tool, except that it was the easiest to learn and get started with.
- Finally, we also used the **multiprocessing** library to enable the simulations to be run at the same time. We achieve this by creating a Pool object and then asynchronously applying the update function in main.py to a Manager dictionary. The Manager dictionary is essential so that the processes can edit the same dictionary whenever they happen to finish. Then we close and join the pool at the end and we can use the data dictionary in the observe function. More information about the multiprocessing library, including Manager and Pools, can be found on this hyperlink.

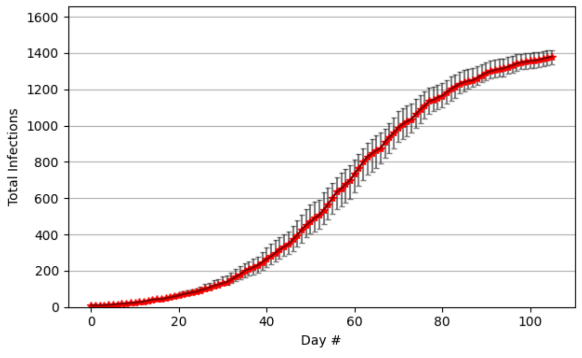
Conclusion

Running simulations using different variants and interventions, we were able to observe and analyze the spread of COVID-19 in the small college campus model and how the variants and interventions can affect the infection spread. Though we did encounter some results that seemed rather strange – such as the relatively excessive amount of infections in the dining hall compared to other spaces – we look forward to continue working on improving the model, confirming that there are no errors in the simulation process or conditions and making additions regarding variants and vaccines following the current pandemic situation. Below are the graphs resulting from running the simulation with our model.

Simulation Results

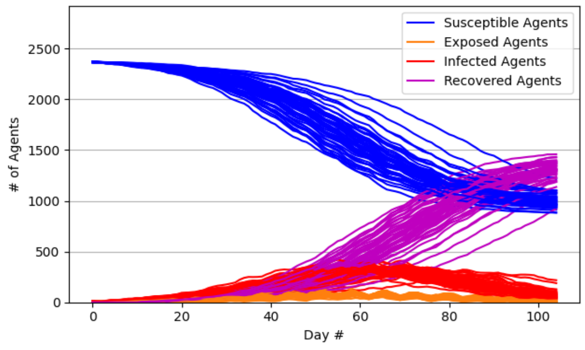
Base Results - ran simulations with alpha variant and no interventions

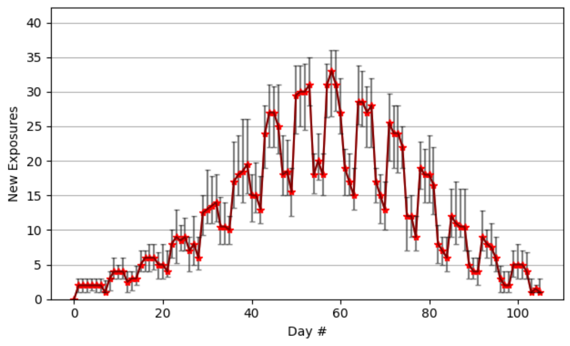




<u>**Number of agents exposed in different campus spaces**</u>

<u>**Number of total infections based on number of school day**</u>

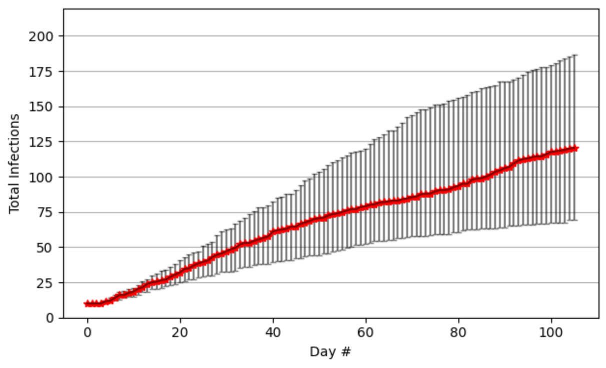




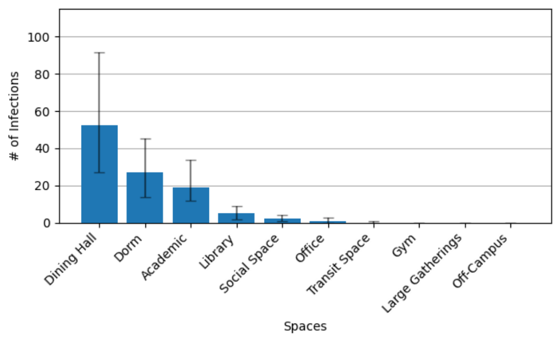
<u>**Number of total infections based on number of school day**</u>

<u>**Number of daily exposures based on number of school days**</u>

Vaccine Intervention Results - ran simulations with alpha variant and vaccine intervention applied to model

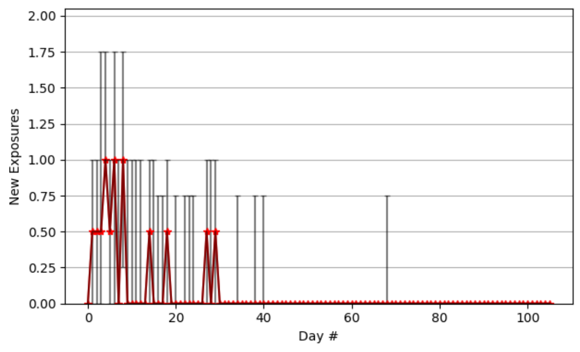


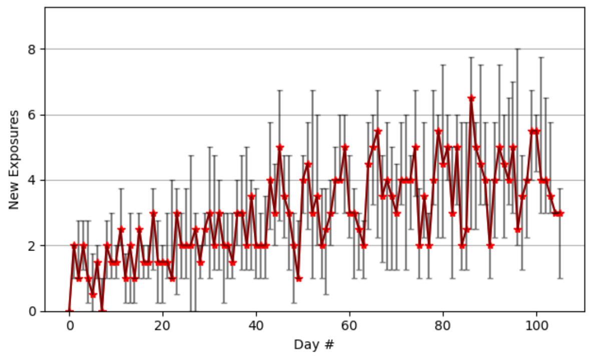
Number of total infections based on number of school day



Number of agents exposed in different campus spaces

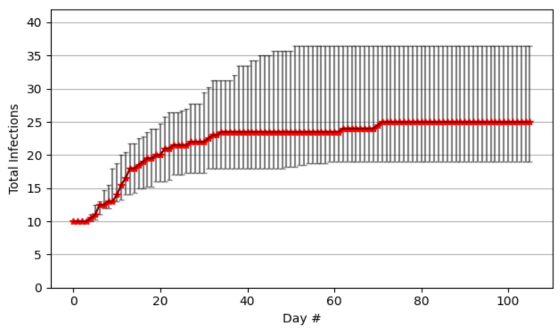
Different Variants Results - ran simulations, once with the alpha variant (left) and another with the delta variant (right), to compare results

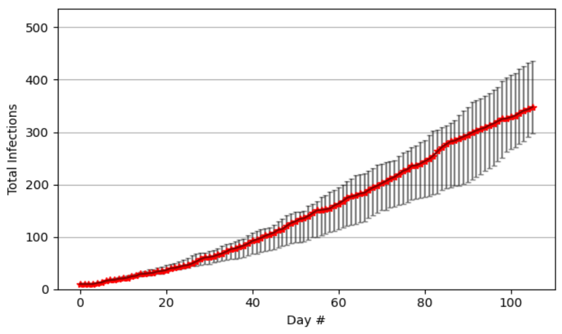




Number of daily exposures (alpha variant)

Number of daily exposures (delta variant)





Number of total infections (alpha variant)

Number of total infections (delta variant)

Citation

Bahl R, Eikmeier N, Fraser A, Junge M, Keesing F, Nakahata K, et al. (2021) Modeling COVID-19 spread in small colleges. PLoS ONE 16(8): e0255654. https://doi.org/10.1371/journal.pone.0255654