Epidemiological Agent-based Model Optimization

DS-GA 1019 Spring 2023
Group 11: Xu Han, Chenxi Ning, Jason Wang, Philip Xing, Chloe Zheng

Introduction

Background

The COVID-19 pandemic posed unprecedented challenges to global public health, economy, and society. Understanding the patterns of an epidemic and the impact of preventive measures is crucial in epidemiology. Mathematical and computational models may provide epidemiologists with useful tools, such as valuable visualizations and demos that model the spread of infectious diseases, the efficacy of interventions, and the overall trajectory of epidemics [5].

Our group aims to develop an Agent-based model (ABM) to simulate an epidemic and generate visual predictions of susceptible, infected, and recovered (SIR) populations over time. An ABM is an appropriate approach because it can explicitly model individual agent statuses within a population and their interactions. This stochastic model also takes into account the uncertainties and variability present in disease transmission and recovery processes, providing a more realistic representation of the epidemic dynamics [6].

However, ABMs can be computationally expensive, particularly when simulating large populations over long time periods. We intend to optimize the runtime of the ABM, ensuring efficient simulation and analysis by comparing and combining various optimization strategies to enhance the model's performance.

Research Question

Our project's research question is to investigate how different vaccination rollout strategies can impact the SIR populations of an epidemic. Specifically, we are interested in exploring the effects of varying vaccination start date, distribution rate, and the targeted population size when vaccinating a more infectable immunodeficient population first. By examining these variables, we hope to visualize and gain insight into how vaccination campaigns can be optimized to more effectively control and mitigate the spread of infectious diseases. As a part of our discovery process, we also wanted to optimize our simulation so that we can run more simulations in less time.

Methodology

Simulation

We first assign each agent in our ABM a random location within a unit square, a status (**S**usceptible, **I**nfected or **R**ecovered), days with that status, and vaccine-specific attributes. At the beginning of the simulation, one agent starts as infected with the disease (i.e. patient zero), while all others start as susceptible.

Our simulation runs for a specified number of iterations, where each iteration represents a day. For each day, every agent's location is updated using a random angle and a random distance from their existing location. If the new location is outside the unit square bounds, the agent gets snapped back to the edge. Every day, an agent's status can also be (1) changed from susceptible to infected, (2) changed from infected to recovered, or (3) remain the same. Transition (1) requires the agent to be within a certain distance from an infected agent and transition (2) requires the agent to have been infected for a minimum number of days. Both transitions are then stochastic in nature after meeting their respective criteria.

To answer our research question, we create three types of simulations - a basic simulation, a simulation with random vaccination distribution, and a simulation with targeted vaccine distribution. The basic simulation demonstrates how an epidemic may unfold without vaccines. The other two simulations show how various vaccination roll out methods can affect the probability of getting infected for each agent and therefore influence the infected population curves over the course of the epidemic. See the full code repository in [1].

Optimization Methods

Optimization is a vital area of study because it involves improving the efficiency of programs using large datasets and executing many computations. Inefficient code can lead to slower execution times and ultimately hinder performance. Employing optimization techniques may help to improve the efficiency of code and as a result, reduce the execution time and enhance productivity.

Two optimization methods we explore are (1) using Cython and (2) Vectorization with NumPy. Cython translates Python code into optimized C code and uses this compiled C code as extension modules for Python. Cython provides easy access to C libraries and data types for Python users, while also harnessing faster execution times with C compilation [3, 4]. Vectorization with NumPy allows for parallel computing operations on arrays and matrices. NumPy is also implemented in a low-level language (C) that operates quickly on large data. These optimization techniques can significantly improve the performance of data-intensive programs.

We also use Line Profiler, a Python performance tuning tool, to diagnose the efficiency of our simulations [2]. Line Profiler allows for a line-by-line analysis of code execution time, measuring the time taken to execute each line of code in a program and producing a detailed report.

Results

Varying Vaccination Speed, Start, and Targets

We set the ability to adjust the simulation's parameters through the command-line interface each time we ran a simulation. The parameters we kept constant were simulation duration (100 days), number of agents (10,000), infection distance (0.03), infection probability (0.3), minimum infection duration (7), recovery probability (0.3), initial vaccine efficacy (0.95), vaccinated recovery reduction (2), and immunodeficient infection probability increase (0.3). We kept these parameters constant and varied the experiment variables one at a time so that we could fairly observe the effect of each variation.

To observe the effects of vaccination start time, we varied the vaccine availability day between day 1, 10, and 25. Our results were captured in plots and gif animations, which are available in our public git repository [1] and project presentation. We see that the later the vaccine is available and begins distribution, the more pronounced the infected population curve becomes. For example, when the vaccine began distribution on day 1, the infected population reached a peak of about 2000 infected at once around day 20. However, when the vaccine began distribution on day 25, the infected population reached a peak of about 3000 infected at once around 30.

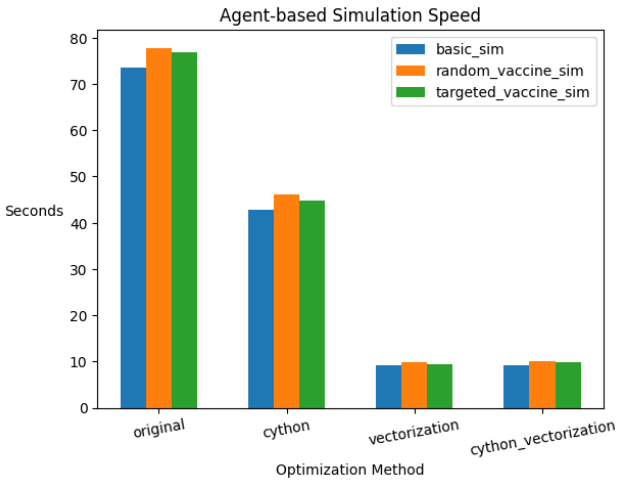
To observe the effects of vaccination distribution rate, we vary the daily vaccination distribution counts between 100, 200, and 500 vaccines randomly distributed a day. We observe that the faster we distribute the vaccine, the more we flatten the infected population curve.

To observe the effects of targeting an immunodeficient population first, we vary the immunodeficient population proportion between 0, 0.2, and 0.4 of the total population. These immunodeficient agents have 0.3 flat increase in their infection probability, meaning in our simulations, they have a total of 0.6 chance of getting infected when near an infected agent. Our targeted vaccine simulation then distributes 200 vaccines a day to the immunodeficient population first before distributing to the rest of the population. We observe that targeting the immunodeficient population first does little to change the infection population curve overall. Our hypothesis is that given roughly 10% of the population gets infected after 10 days, which is when the vaccine begins rolling out, vaccinating the immunodeficient population first does little to change the overall population curves given the relatively high infectiousness of our simulated epidemic.

Line Profiler, Cython, and Vectorization

To improve the efficiency of our simulations, we first use Line Profiler to produce a detailed report for our basic simulation. We find that our infect function in our transition module takes about 80-98% of the execution time of our original code, depending on the user parameters. Therefore, we focus on improving this function with each of our optimization methods.

In our Cython implementation, we declare types for every relevant integer and float. In our final version, we compiled the agent.pyx, transition.pyx, and location.pyx with Cython. In our vectorized implementation, we added NumPy arrays to save information about Agents. We expect this to speed up execution because we access these arrays many times, and it is more efficient to access and edit arrays than lists. We also remove the nested for loop in the infect function in the transition module - NumPy matrices are used to parallelize the distance computations. The bar graph shows that the Cython implementation is almost 2 times faster, while the vectorization implementation is about 8 times faster than all three of our simulations. The combination of Cython and vectorization does not significantly increase the speed of the simulations compared to vectorization alone.



Note that for the bar graph, we use the following parameters for each simulation:

- basic_sim 100 10000 0.03 0.3 7 0.3
- random_vaccine_sim 100 10000 0.03 0.3 7 0.3 10 200 0.95 2
- targeted_vaccine_sim 100 10000 0.03 0.3 7 0.3 10 200 0.95 2 0.4 0.3 30

To evaluate our best optimization method, we use Line Profiler again to check if we successfully decreased the time for the function, infect. The table shows that the infect function takes significantly less execution time in the vectorized code compared to the original code. Note: the same basic_sim parameters used for the bar graph were used for the following results:

Simulation	% Time	Line Contents
basic_sim - Original	98.0	infect(...)
basic_sim - Vectorization	55.6	infect(...)

Conclusions

Conclusion: Our project on optimizing epidemiological ABMs has provided valuable insights into the spread of infectious diseases and the efficacy of various vaccination strategies. Even though our results suggest that targeting immunodeficient individuals in vaccination campaigns may not significantly impact the overall infection curve, given the high infectiousness, we have demonstrated the importance of early vaccination distribution in mitigating the spread of infectious diseases.

Optimization: Vectorization fits our model best, offering the most improvement in computational performance. While some techniques may be easier to implement, there is a trade-off between optimization efficiency and simplicity which indicates that selecting the most suitable optimization method is crucial for improving the efficiency of the model.

Future Directions: In the future we can study how social distances between agents can affect the spread of disease, incorporate central locations to simulate essential places such as grocery stores and hospitals, consider the possibility of reinfection, and divide the population into age groups to account for age-related vulnerability to infection. By refining our model, we can further enhance our understanding of infectious disease dynamics and contribute to the development of more effective pandemic mitigation strategies for the future.

References

1. https://github.com/xh852/Epidemiological-Agent-based-Model-Optimization
2. https://nyu-cds.github.io/python-performance-tuning/03-line_profiler/
3. https://cython.readthedocs.io/en/latest/src/quickstart/overview.html
4. https://nyu-cds.github.io/python-cython/
5. The Institute for Disease Modeling: https://idmod.org/
6. 3Blue1Brown's "Simulating an epidemic": https://www.youtube.com/watch?v=gxAaO2rsdIs