MODELLING COVID-19 TRANSMISSION IN *PALENGKES*: AN AGENT-BASED CASE STUDY OF BAGUIO CITY'S PUBLIC MARKET

BY

Lanchelot D. Lucero

A special problem submitted to the

Department of Mathematics and Computer Science

College of Science

The University of the Philippines

Baguio, Baguio City

as partial fulfillment of the

requirements for the degree of

Bachelor of Science in Computer Science

June 2024

This is to certify that this Special Problem entitled **"Modelling COVID-19 Transmission in *Palengkes*: An Agent-based Case Study of Baguio City's Public Market"**, prepared and submitted by **Lanchelot D. Lucero** to fulfill part of the requirements for the degree of **Bachelor of Science in Computer Science**, was successfully defended and approved on June, 2024.

Lee Javellana
Special Problem Adviser

The Department of Mathematics and Computer Science endorses the acceptance of this Special Problem as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science .

Gilbert R. Peralta, Dr. Rer. Nat.
Chair
Department of Mathematics and
Computer Science

Table of Contents

Acknowledgments

To my adviser, Sir Lee, thank you for your excellent guidance throughout the writing process of this study. Your insights and expertise have been invaluable towards the success of this paper.

To the creator of the agent-based model that this study was inspired from, Sir Fabian, thank you as well for your assistance in the initial stages of my study.

To the University of the Philippines Baguio community, thank you for honing my skills in all levels of my academic life. Further, I acknowledge the administrators, faculty, and staff of UP Baguio for their dedication in teaching and providing me the necessary resources to succeed in my chosen field. Most importantly, I acknowledge the love they have for fostering a safe and comfortable space for me to express myself freely.

To The Department of Science and Technology, thank you for the opportunity you have given me to be supported financially throughout my entire college years. This opportunity have lead towards the eventual success of this study.

To all the vendors in public markets, especially in Baguio's public market, thank you for your service to the community. As someone who buys at the public market every week, I acknowledge your efforts in providing us with fresh and quality goods all the time.

To my mom Rechel, my father Lauren, and my grandmother Mama Sally, thank you for trusting in my abilities. Your confidence in me had always been my inspiration to succeed. I offer this success to all of you after all your sacrifices and hard work throughout these years. All your efforts have not been led to waste.

To my sister Ina, who recently passed the UPCAT, thank you for being the best

sister in the world. You taught me how to be strong and not at the expense of kindness and gentleness, all of which ultimately helped me succeed in my academic and personal life. I wish you the best of luck in your new journey.

To all my friends: Nicole, Thea, Des, Leo, Chezka, Maj, Pat, Myla, and Gab, thank you for making school life full of fun and enjoyable memories. Our times together will always remain with me. Our friendships helped me move forward because you all have inspired me in your own, unique ways. I am really happy and thankful for all that we have gone through together.

To my babies here and beyond: Dash, Tobi, Maya, Ola, Mira, and Appa, I appreciate you all for being part of my life. I appreciate your smiles and greetings whenever I come home. Thank you for the relief and happiness you all offer just by being loving and cute. I am happy to give you all your favorite belly rubs.

To the Universe, thank you for all the love, protection, and guidance throughout these years. Thank you for taking care of me; for always ensuring my safety. I appreciate all your efforts, for helping me get to know myself better each day and be the best version of myself.

To myself, thank you for facing it all, for going through it, and for coming out better each time.

In sum, this success has not come into fruition only through my personal efforts but also the culmination of all the love, support, and guidance of many individuals in my life who believed in me along the way. I feel loved by all of you. I dedicate this milestone to all of you. Thank you so much.

Abstract

Modelling COVID-19 Transmission in *Palengkes*: An Agent-based Case Study of Baguio City's Public Market

Lanchelot D. Lucero
University of the Philippines, 2024

Adviser:
Lee Javellana

Public markets or *palengkes* play an integral part in many communities like the Philippines. However, since the COVID-19 pandemic, these crowded commercial spaces have also put the lives and livelihoods of residents at risk. Therefore, developing interventions to reduce the risk of transmission within public markets is crucial to protect both buyers and vendors from contracting the disease. In this study, an agent-based model is used to demonstrate how buyers move around a public market setting and quantify the risk of infection using the time spent by susceptible buyers near infected buyers. In addition, this study assessed several policies that help mitigate this risk: namely, restricting the capacity of buyers allowed inside the market, reducing the rate of buyer arrivals in the market, enforcing a mandatory face mask policy, and implementing a one-way aisle setup inside the market. To show this, the case of Baguio City's public market was applied to the model. The results of this study show that the mean daily number and chances of infection in the market is at 2.67×10^{-6} and 1.48×10^{-9}, respectively, with 89.25% of buyers having 1.2 minutes of exposure on average to infected buyers. Moreover, the results reveal the market's bottlenecks including the entrances, exits, and non-vendor aisles. Finally, using mandatory facemask policy and decreasing the rate of buyer arrivals in the market significantly reduced the number and chances of infection by 96% and 92%, respectively, whereas implementing a one-way aisle setup has shown a 27% increase in both metrics.

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, the study describes the foundational context of the study. This includes discussing the relevance of public markets and the risk of transmitting the Novel Coronavirus 2019 Disease (COVID-19) within these areas. In the following chapters, the study will assess how the disease spreads in these spaces as well as the interventions that can be made to mitigate this risk.

1.1 Background of the Study

Public markets or *palengkes* are commercial complex spaces consisting of vendor stalls that sell wet goods like meat, seafood, poultry, vegetables, and fruits and are often characterized by slippery floors and narrow aisles [3]. In the Philippines, public markets serve as one of the main providers of food and livelihood for many local residents. In fact, in the middle of the COVID-19 pandemic where the operation of multiple stores and businesses were shut down by the government, public markets were helping maintain food and job security throughout the country [23].

1.2 Statement of the Problem

Considering how public markets have remained open throughout the pandemic, it runs the risk of being crowded with many people at once. This means that there is a higher likelihood of contracting the COVID-19 disease within these spaces. Hence, it is crucial to find ways to ensure safer market operations while managing the spread of the disease. Modeling buyers' behavior and virus transmission help us to predict the probability of infection [41]. Moreover, developing interventions that help reduce the further spread of the disease also helps us in managing its effects.

1.3 Objective of the Study

In this section, the study discusses the general and the specific objective of the study. Moreover, the reasoning behind these objectives are explored.

1.3.1 General Objective of the Study

The general objective of this study is to apply a virus transmission model via respiratory droplet transmission among buyers moving around a public market, buying on stalls, and coming in close proximity with other buyers using agent-based modeling. In this study, we want to capture a detailed or disaggregated approach to how individuals interact or behave in a public market setting.

1.3.2 Specific Objective of the Study

Transmission via respiratory droplets is the primary mode of COVID-19 transmission [38][9]. Hence, we use a simple virus transmission model and ignore secondary or tertiary transmission of the virus such as fomite and airborne. In addition, this study aims to provide a quantitative assessment of the following policies:

1. Restricting the capacity of buyers allowed inside the market
2. Reducing rate of buyers entering the market
3. Enforcing a mandatory face mask policy
4. Implementing a one-way aisle setup inside the market

We use Baguio City's public market because it is among the major urban centers in the highly populated city. Not only does the market provide food security for the people, but it also supports many livelihoods which ultimately contribute to the social and economic security of the city [5]. However, the risk of contracting COVID-19 is relatively significant in this area, as there are expected to be many people in this place at once. Hence, we apply this case study to the agent-based model to understand the dynamics of virus transmission unique to the spatial aspects of the public market. Most

importantly, in using this model we will evaluate the total cumulative time spent by buyers in close proximity to an infected buyer inside the public market or as we refer to as the cumulative exposure time.

1.4 Significance of the Study

We aim that through this model, we can bridge a gap in existing literature by providing critical insights on the dynamics of virus transmission in public markets. Hence, in the event of another disease outbreak like COVID-19, this study would serve as a valuable tool in informing policymakers on decision-making regarding mitigating the risk of transmission.

1.5 Scope and Limitation

In this study, we focus on how buyers move around the public market to buy items from vendor stalls and spend time in close proximity with other moving and potentially infectious buyers. Additionally, we treat transmission via respiratory droplets as the mode of transmission of the disease in the model. Furthermore, we assess different policies related to controlling its spread: namely, reducing the number of buyers arriving and buying inside the market, using a mandatory face mask policy, and implementing a one-way aisle setup. On the other hand, we do not extend our model in taking into account vendors in the transmission of the disease, the precise distance between buyers, and the actual floor area of the market space. Further, our model uses a predefined rate of buyer arrival and shopping trips of buyers which is constant in time despite the fact that it may change and vary in real life at any given moment. Finally, due to lack of sufficient data on the disease's transmission dynamics in public markets, our value for the transmission rate cannot be entirely taken at face value as it is measured on a different setting. Nevertheless, the exposure time still presents a highly relevant and important measurement to the relative transmission risk of the disease inside the market.

Chapter 2

Review of Related Literature

In this chapter, the review will explore scientific works that relate to the history and modelling of the transmission of COVID-19 via airborne and respiratory droplets. Additionally, the review will explore the scientific principles of agent-based modelling and its relevance in modelling infectious diseases like COVID-19.

2.1 Background on COVID-19

In December 2019, a group of patients in Hubei Province, Wuhan, China reported an illness similar to pneumonia except that it does not respond to common treatments [10]. The source is believed to be the Huanan Seafood Wholesale Market, which is subsequently closed due to the growing worries of a possible outbreak of a disease [40]. In January of 2020, the disease was investigated and the agent was identified to be a novel coronavirus causing the outbreak, which was soon to be named as the Novel Coronavirus 2019 or COVID-19 [10]. Eventually, this disease would cause the most recent epidemic in the entire world, causing catastrophic damage to people's lives and to countries' economies [20]. At this time of writing, the World Health Organization (WHO) reports the total cumulative deaths from COVID-19 to be 7 million worldwide [39]. Fortunately, several precautions against the virus have subsequently emerged and developed, such as enforcing the use of facemasks, face shields, and new vaccines. Adjacent to the emergence of the disease are scientific works that describe the mechanism of transmission of the disease, such as airborne and respiratory droplet transmissions.

2.2 Airborne Transmission Models

According to the World Health Organization, airborne transmission is where an infectious agent that spread through droplet and evaporated into microscopic aerosols ($\leq 5\mu$m) remains suspended in the air for extended periods of time over long distances, including talking or normal breathing [37]. Some models that aim to describe airborne transmissions of COVID-19 include computational fluid dynamics models, well-mixed room models, and partial differential equations.

2.2.1 Computational Fluid Dynamics

Computation Fluid Dynamics (CFD) in airborne transmission modeling of COVID-19 predicts the phenomena of the airborne virus' flow based on conservation laws governing fluid motion, usually with the help of digital computers [21]. In [2], the physics of dispersion and mixing of biofluids was used to develop a Lagrangian turbulent air-flow to demonstrate how the virus causing COVID-19 can be scattered just by an infectious person's breathing. The study also used a restaurant in Guangzhou, China as its spatial reference, concluding that the outbreak of the disease is caused by the buildup of the suspended droplets or aerosols in the air which could have been mitigated by proper, standardized ventilation. Similarly, CFD simulations were used to quantify how aerosols that have been exhaled are dispersed throughout different settings in [31], including an elevator, a small classroom, and a supermarket. The study also found how standard ventilation significantly reduces the dispersion of the particles causing COVID-19, thereby reducing the chances of being inhaled by susceptible individuals. In [36], the airborne transmission of aerosols causing COVID-19 was modeled using the principles of droplet and aerosol dispersion indoors by taking into account how aerosols are concentrated in different areas and how a susceptible person accumulates them by inhalation. It was found that the exposure time was crucial as more airborne aerosols can be inhaled or accumulated depending on the aerosols' local concentration relative to the individual. In sum, CFD models offer helpful insights in describing the airborne phenomena of the disease especially within indoor spaces. However, aside from focusing on only airborne transmission, these models tend to be theoretically complex, computationally expensive,

and hard to scale for larger simulations where many objects tend to move around the simulation space, like buyers moving around a large public market.

2.2.2 Well-mixed Room

Well-mixed Room (WMR) models capture aerosols carrying the virus evenly dispersed across a room. In [24], the WMR airborne model is developed by the governing advection-diffusion-reaction equation whereas aerosols are advected or horizontally dispersed in the air by infected people, further diffused from turbulence and inhaled by susceptible people, and removed through ventilation. Further, the study considered the model in an airconditioned space with recirculating flow, along with facemask and no facemask setup with a single infectious individual who disperses the virus causing COVID-19 through talking or breathing. It was proved how ventilation limits the transmissibility of the disease, as well as the speed the particles are dispersed in the air. However, WRM models like [24] generally limit its spatial scope to a single room and ignore the shape or geometry of the space it is considering, such as a store or a market for instance. Furthermore, WRM models assume that everyone in the simulation space has an equal chance to be infected regardless of their relative position in the said space.

2.2.3 Partial Differentiation Equations

Partial Differentiation Equations (PDEs) in infectious diseases modeling describe and predict how infectious diseases like COVID-19 are spread through the air using numerical methods. PDEs also allow more detailed spatial scope or geometry while being more computationally manageable. In [13], the relationship between the probability of infection and rates of ventilation is estimated using the Wells-Riley equation, which is a classic model for estimating the risk involved with transmissible airborne disease like COVID-19. The study tested different ventilation scenarios including classrooms, offices, buses, aircraft cabins, and workplaces and found that the transmissibility of the virus is reduced when measures such as mask-wearing and testing in public spaces is strictly observed. In [4], a PDE model was used to estimate how much infectious particles are emitted in the mouth of an individual based on the person's activities like breathing, speaking,

standing, resting, and also the rate of inhalation. Using this novel approach, the study found high quanta emission rates when an asymptomatic person is vocalizing during simple activities such as slow-paced walking. Similarly, the Wells-Riley classic equation is used in [33] while taking into account social distancing and space ventilation. In particular, the study proposed two indices along with the Wells-Riley equation: the distance index that is theoretically analyzed through the dispersion of particles and the underlying transmission through inhalation and exhalation, and; the ventilation index that demonstrates how these particles are distributed across the space. The study found limiting the distance at least 1.6n-3.0m is enough to reduce the airborne transmission from talking under standard ventilation conditions or fresh air spaces. In sum, these studies effectively utilize PDEs to account for the transmissibility of the disease under different conditions and spatial contexts. However, these PDE models do not take close or direct person-to-person contacts as a result of a person's movement across its space.

2.3 Respiratory Droplet Transmission Models

According to WHO, respiratory droplets are generated when respiratory secretions (>5-10μm) are excreted in the air whenever an infected person coughs, sneezes, talks, or even sings and reach a susceptible person's eyes, nose, or mouth– thereby resulting to an infection [9]. Some models that describe COVID-19 via respiratory droplet transmissions include Compartmental Models and Agent-based Modelling (ABM).

2.3.1 Compartmental Models

Compartmental models aim to describe phenomena like COVID-19 by categorizing people in a population based on their disease states, progressions, and transitions. In [12], a modified Susceptible-Infected-Removed (SIR) model was used under different communities based on the data from countries like USA, Korea, and India, to show how the disease spreads and to forecast an analysis on managing the impact of the virus spread. The study extended the abilities of this compartmental model to accept an increasing

number of susceptible individuals, which means that instead of using a constant, homogeneous mix of population with decreasing number of susceptible individuals as observed in a classic SIR model, there will only be an increase in the number of these individuals to account for the relative increase in infected individuals. In [30], the Susceptible-Exposed-Infected-Recovered (SEIR) model was used in conjunction with vaccination to measure how the virus has spread in Iran, effectively finding social distancing and vaccination as key measures in curbing the outbreak. However, even with few modifications from the classic compartmental models, these models still tend to oversimplify the complex dynamics of disease transmission, such as assuming homogeneity within groups of people in terms of their behavior, susceptibility, and infectiousness which might not realistically represent an actual population in the real world.

2.3.2 Agent-based Modelling

Agent-based models (ABM) are characterized by a computer simulation that studies the behavior or interaction of agents under some set of rules or conditions. These agents could be people, places, things, time, places, or even pathogens in terms of epidemiological modeling. Moreover, ABM is also stochastic and heterogeneous; that is, certain probability distributions are used to characterize or program the behavior of these agents to account for their actual and real-life behaviors [11]. This way, ABM is different from different modeling techniques as it allows to represent heterogeneity and demonstrate an in-depth exploration of more complex and intricate systems by considering the behaviors of individual agents or actors within the system [19]. Another important feature of agent-based models is also its dynamically rich representation of environmental factors. For instance, geographic information systems (GIS) are used to represent highly detailed geographical spaces of a large-scale city in [22], including road networks, parks, tourist attractions, homes, and workplaces. By directly using these spatial attributes, ABM can be effective in demonstrating dynamics that happen across space and time [19]. In epidemiological modeling, spatial elements are useful in assessing contacts happening among individuals [19]. Therefore, by ABM, one can capture the interaction of agents or actors with each other along with their environment, making it a highly insightful tool

for informing decision-making in complex systems.

In infectious disease modelling, ABM can be used to describe respiratory droplet transmission of diseases like COVID-19 by having a more disaggregated approach to individuals' behaviors. This can be highly effective in representing actual, real world situations related to disease transmission and progression, but is highly complex in terms of computation especially when dealing with large-scale simulations. Since this type of modelling also relies on individualized data, gaps may arise due to lack of data availability or constraints in cost and exhaustive data collection. In [28], agent-based modeling was used on a supermarket with a high spatial resolution. However, the model they propose comes at a high computational cost and no code availability. In [26], an ABM model was used to estimate the rate of infection within classroom settings in the Philippines. In the study, different layouts of classrooms common in the Philippines were evaluated while still accounting for the mobility of students and teachers like seating, moving, and switching classrooms. Nevertheless, there seems to be a gap in studies considering public market spaces as its main, spatial focus.

Overall, existing studies that model virus transmission on COVID-19 mostly focus on airborne transmission, while models that describe respiratory droplet transmissions tend to lack granularity or demand high computational or theoretical complexity. Moreover, while there are extensive studies that take into account indoor spaces such as schools, supermarkets, buses, offices, and restaurants, there are currently gaps in studies that focus on respiratory droplet virus transmission specific to public markets.

Chapter 3

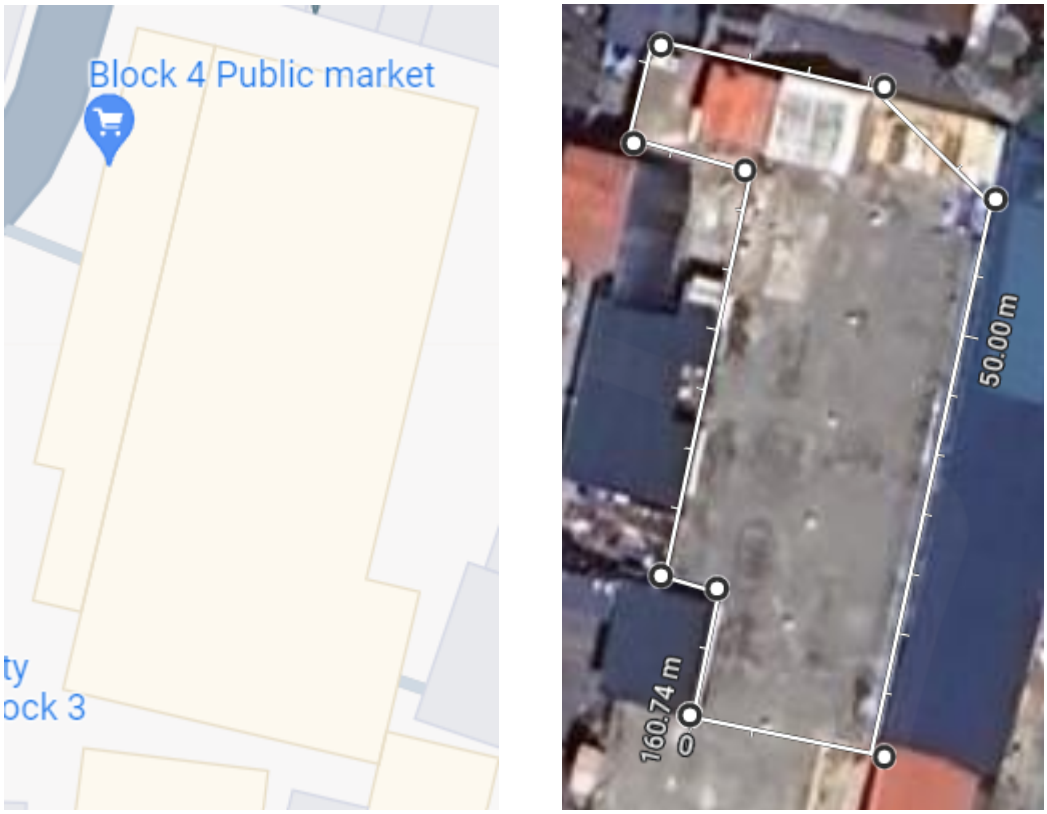
Methodology

In this chapter, the methodological framework of the study is demonstrated. The framework is drawn from the agent-based model based on the previous work of Ying and O'Cleary [41], who studied the transmission of COVID-19 disease infection in supermarkets. We modify the existing model to account for the unique attributes of Baguio City's Public Market environment, such as market layout and shopping dynamics of buyers.

3.1 Spatial and Temporal Scopes

We define the spatial and temporal scopes of our model. We consider the spatial resolution according to the movement of buyers along the passageways in the public market. These are areas where buyers can enter, browse, make purchases, and exit. For this study, we choose Baguio City's public market as our spatial reference. Particularly, we choose the market's fourth block's lower ground covering the fruits and vegetables section. The spatial reference is shown in Figure 3.1. Due to lack of data availability for the block's floor plan, we manually map the spatial resolution using our personal visits throughout the entire block. This is to ensure that we can still realistically track the movement of buyers around the market even with restraints in cost and data.

For the temporal resolution, we set the time step to a minute that extends throughout the opening hours of the public market. This ensures a high level of granularity in terms of how many buyers are there arriving, moving, and interacting inside the public market at any given time.



(a) Map of Block 4 (b) Fruits and vegetables section

Figure 3.1: Block 4, Public Market in Baguio City [16]

3.2 Agents

The agents in the simulation are referred to as buyers. Buyers are simulated local residents of Baguio City that shop in the public market. We ignore vendors in the model because our primary focus in this study is to study COVID-19 virus transmission unique to buyer's dynamics, which involves them moving around the market while browsing, shopping, and coming in contact with other moving buyers.

3.3 Initialization

Before starting the simulation, we begin by generating the network containing the buyer pathways within the market. Each node in the network corresponds to a specific location within the market: entry or exit points, shopping locations at vendor stalls, and non-shopping locations at some intersections. Links connect these nodes together to represent how buyers can move through the public market. Due to lack of available data, we create the network representation of the market using personal observations during visits to the market. We illustrate the generated network in Figure 3.2.

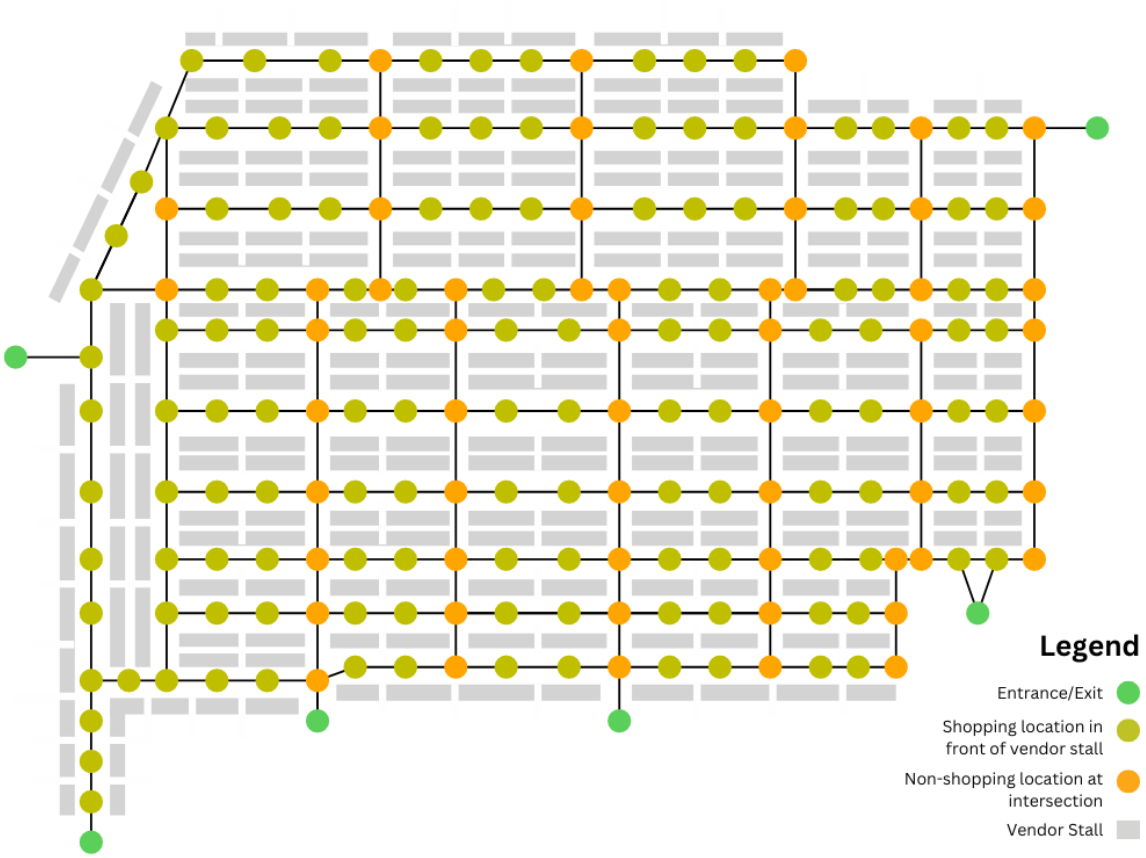


Figure 3.2: Network representation of the market.

After creating the network, we proceed by generating a list of shopping trips. We define a shopping trip as a sequence of connected nodes in the network representing the series of locations inside the market where a buyer enters, browses or makes purchases,

and leaves. Each buyer has a random shopping trip assigned to them upon arrival at the market.

To initialize a shopping trip, we first generate the number of items a certain buyer will purchase during their visit, which we denote as K. Using a log-normal distribution, we draw a random integer value for K. Log-Normal distribution is a probability distribution where a dataset produces a normal distribution when evaluated with the natural logarithm function. We use this distribution because samples drawn from it best represent shopper behavior, that is many buyers have shorter trips for fewer items in their basket while less buyers have longer trips with more items in their basket [32]. We parameterize the distribution with mean μ standard deviation σ from the underlying distribution and use numpy's `lognormal` function. We choose $\mu = 0.07$ and $\sigma = 0.76$ based on the purchased number of products for small supermarkets and convenience stores in [32] due to the lack of consumer behavior data for public markets.

After sampling K, we create the shopping trips as follows. First, we choose a random entrance node in the network. This corresponds to an actual entrance in the market. Second, we randomly select a number of K nodes in the network with replacement, but excluding the entrance and exit nodes. These nodes represent the sequence of vendor stall locations where a buyer makes their purchases. Third, we select a random exit node in the network. This will represent the location in the market where a buyer leaves after completing their shopping. Finally, we find the shortest path between each chosen node using the Dijkstra Algorithm. We show an example shopping trip in Figure 3.3A, where the buyer purchases K=7 items at the stalls marked 2, 3, 4, 5, 6, 7, and 8 along with a random entrance node marked as 1 and random exit node marked as 9. After that, we find the shortest path between each chosen location to generate the full shopping trip as shown in green arrows. Prior to the simulation, we pre-generated 1,000,000 shopping trips using this method.

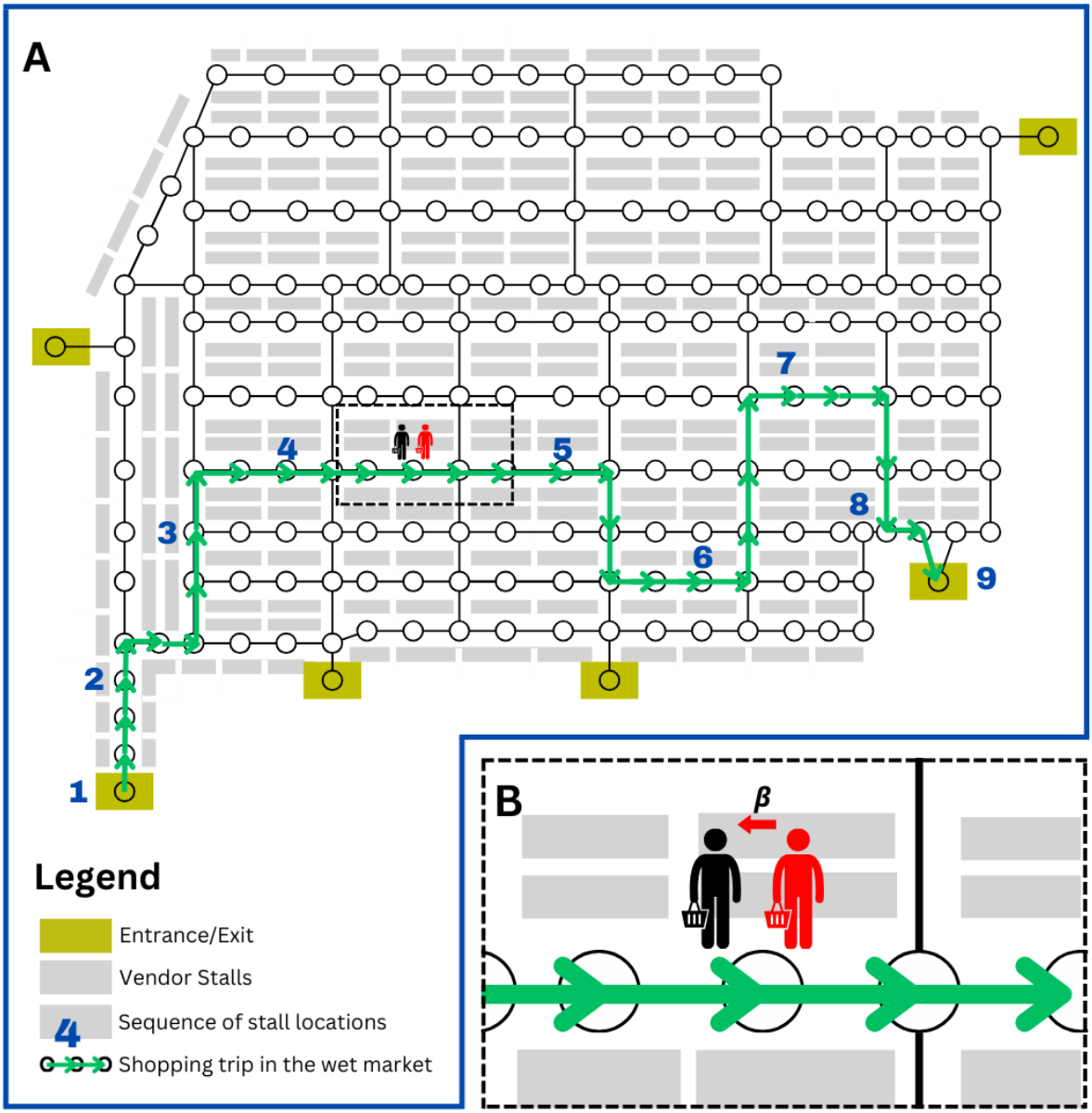


Figure 3.3: Overview of the agent-based model. (A) Example shopping trip. (B) Virus transmission with transmission rate β.

3.4 Flow

In this part, we describe the processes of the model along with the parameters that we used in the simulations.

Initially, the market is empty. Buyers start to arrive at the market during the opening times. We denote this time as H hours. During the COVID-19 pandemic, the public market of Baguio's operation hours is from 5:00 AM to 5:00 PM according to a report by [14]. Thus, we set $H = 12$. We also use the Poisson process to define a constant arrival rate λ of buyers in the simulation. This process ensures that at any each time step during the simulation, there will be λ buyers arriving at random. We choose $\lambda = 2.88$. During the COVID-19 pandemic, the Baguio City Government only allowed one person per household to visit the public market [14]. There are 100,220 households in Baguio by 2020 [27]. Moreover, in a study assessing food security in the Philippines during the pandemic, 29% out of 331 households were found to shop at wet markets [35]. We infer from this that an average of 29,063.8 households in Baguio shopped at the city market during the pandemic. Moreover, Filipinos shop for groceries at an average of two times per month during the pandemic [18]. From this we deduce that there are an average of $29,063.8/2 = 14,531.9$ buyers per week or 2,075.99 average buyers per day in the city market. Dividing this by the total opening time, that is 12 hours or 720 minutes, we have an average of 2.88 buyers arriving per minute.

Next, each random arriving buyer is assigned a random shopping trip and categorized as either susceptible or infectious. Susceptible means that these buyers are not infectious but they can potentially be infected when they come in contact with an infectious buyer. A buyer entering the store can be infectious or susceptible depending on the probability p_I or the proportion of the infected buyers in the store. Here, we choose $p_I = 12.8\%$. According to the report from the Department of Health (DOH), the total number of infected persons in Baguio as of January 8, 2024 is 46,906 [15]. In a census from the Philippine Statistics Authority on May 1, 2020, the population of the "highly urbanized" city is 366,358 [27]. Using these information we compute the proportion of infected persons p_I inside the market as $46{,}906/366{,}358 = 0.128$.

After being assigned a shopping trip and a disease category, a buyer enters the market. Each buyer starts at a random entrance node in the network based on its own shopping

trip. Then, before the buyer moves on to each node, the model checks for three things.

First, the model checks whether there are any other susceptible or infected buyers present in the current node. If the said buyer is categorized as a susceptible buyer, then the simulation checks whether an infected buyer is in the same node as them. If there are any, the model determines the total amount of time they are in the same node with each other, which is also known as the exposure time for that certain susceptible buyer. In the simulation, we define $E_S^{(i)}$ as each buyer's exposure time to infected buyers i, that is when both the susceptible and infectious buyer was in the same zone. The buyers' cumulative exposure time is then $E_S^{(i)} = \sum_i E_S^{(i)}$. A susceptible buyer is further classified as an exposed buyer when they have a nonnegative exposure time. All exposed buyer can become infected after their trip by the probability $min(\beta E_S, 1)$ for $\beta > 0$. In short, the probability of an exposed buyer being infected is a linear function of how long they spent in close proximity with infectious buyer/s. We illustrate how virus transmission works in our model in Figure 3.3B. Otherwise, if a buyer is classified as an infected buyer, then the model simply checks how many susceptible buyers are in the same node as them.

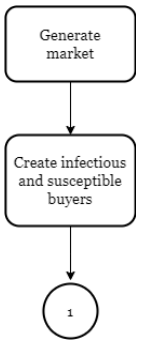
Second, the model computes some random time T for which the buyer stays at the current node to browse or shop. This time is distributed exponentially with mean τ. We choose an exponential distribution for τ to account for the behavior of shoppers where they tend to spend less time browsing, looking, or inspecting items towards the end of their shopping trip due to less energy or fatigue [1]. In the simulation, we set $\tau = 0.48$. We infer this value based on the open data provided by Google Maps [16] on Baguio Public Market, where buyers spend at least an average of 20 minutes in the area. We use this information along with the average length of the 1 million pre-generated shopping trips, that is 40.42 nodes. Thus, $20/40.42 = 0.49$ or 29.4 seconds per node on average. The buyer shops at a node whenever the next node in their shopping trip is the same as their current node. Otherwise, we assume that the buyer is simply browsing or moving along the nodes.

Third, the model checks for how much buyers are in the next node. Here, we assume that all nodes in the network can only hold a maximum capacity of four buyers at a time. Hence, before a buyer moves on to the next node in their shopping trip, the model checks if that node contains four buyers or more. If yes, then this buyer waits until the next node becomes less occupied. Then, the buyer moves on to that next node. If the next node in a buyer's shopping trip is not fully occupied, the buyer simply moves to that node. Furthermore, before entering the market, new arriving buyers also check the occupancy of the entrance node or the first node in their shopping trip before entering the market.

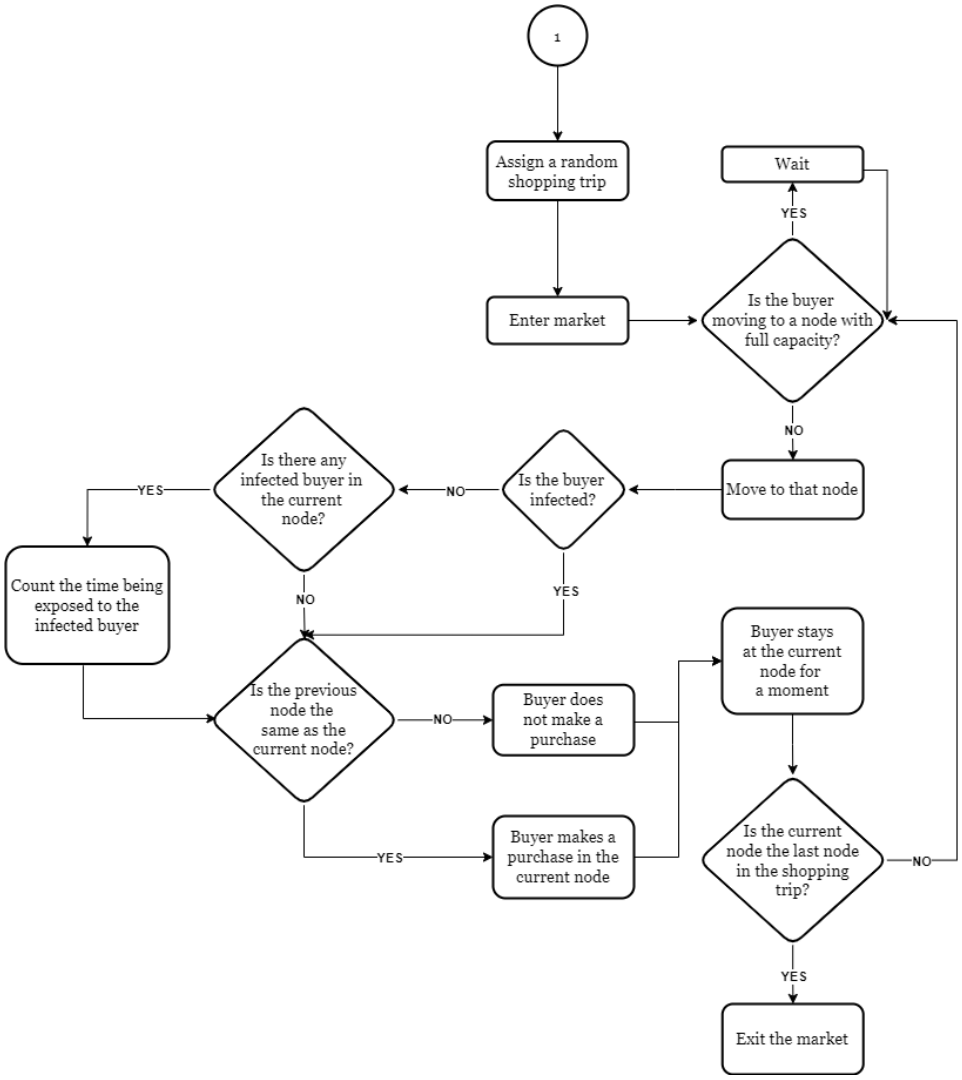
Finally, when the buyer arrives at the exit or the final node in their shopping trip, they wait for another time T before they are removed from the simulation. When the simulation reaches H hours or the total operating hours of the market, no buyer can longer enter the market. Then, the simulation stops.

In the model, we suppose susceptible buyers become infected in proportion to the time they are in close contact with infectious buyers, that is whenever they are both in the same node in the network. Transmission through respiratory droplets such as sneezing, coughing, or talking rather than airborne and fomite transmission is considered due to it being the primary mechanism of infection. The rate of transmission β where an infectious buyer infects another susceptible buyer whenever they are at the same node is 1.41×10^{-9} per minute. The mean transmission per contact from [34] is found to be 2.11×10^{-8}. We use this data to compute the average transmission rate of the disease assuming that the duration of contact per average from is 15 minutes, that is $\beta = 2.11 \times 10^{-8}/15 = 1.41 \times 10^{-9}$ per minute. Since this value and calculation is similar to [41], we assume a 95% confidence interval from the said study as well: $(1.15 \times 10^{-9}, 1.66 \times 10^{-9})$. We summarize the flow of the model in Figure 3.4.

The agent-based model results in two primary values; the average number of susceptible buyers that gets infected with the disease, and; the daily, average infection chance for each susceptible buyer inside the market.



(a) Network and buyer generation



(b) Buyer flow

Figure 3.4: Model Flow

3.5 Specifications

We summarize the parameters containing the modeled concepts and their corresponding references in Table 3.1:

Table 3.1: Model Specifications

Parameter	Default Value	Reference
Parameters (μ, σ) of log-normal distribution for number of items bought	(0.07, 0.76)	[32]
Length of opening hours H	12 hours	[14]
Arrival rate λ	2.88 buyers/min	[14][27][35][18]
Percentage of infectious buyers p_I	12.8%	[15][27]
Transmission rate β	1.41×10^{-9} per min	[34][41]
Average waiting time per node τ	0.49 min	[1][16]
Market layout	Fixed	Personal visits

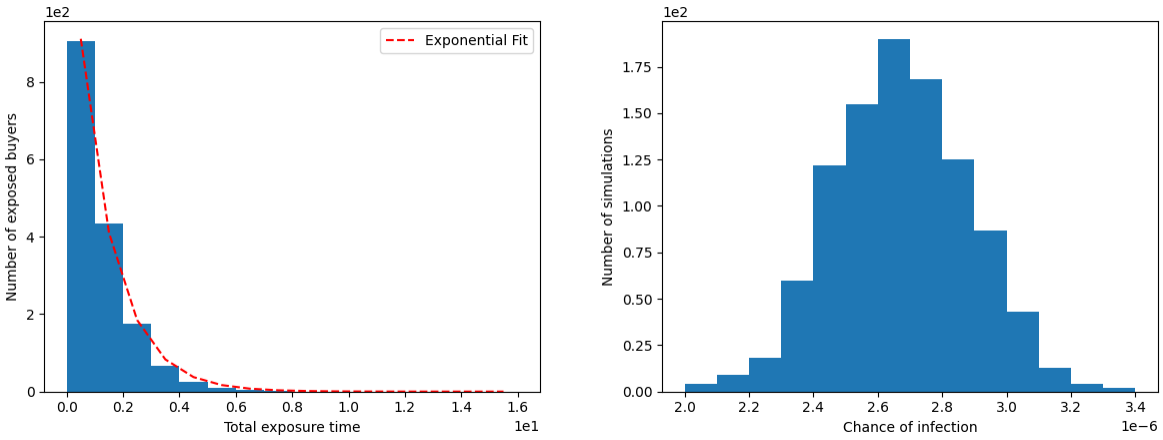
3.6 Execution

Finally, we execute the agent-based model using Python 3.12 in Visual Studio Code on a Lenovo V15 IAP personal computer running Windows 10 Pro with 16 GB RAM and 12th Gen Intel Core i5-1235U 1.30 GHz.

Chapter 4

Results and Discussion

We perform 1000 simulations using the parameters listed in Table 3.1. Each simulation run corresponds to a single day in the public market during its opening times. In this section, we list the recorded outputs throughout these simulations.



(a) Mean exposure times per exposed buyer (b) Mean infection chance per simulation

Figure 4.1: Average exposure times and infection chances across 1000 simulations.

Based on the simulations, there are about 52.8 buyers on average inside the public market at any given time, each shopping for an average of 19.8 minutes in total. Moreover, the cumulative exposure time of all susceptible buyers to infected buyers in the market is at 1,986.1 minutes per day on average. If the disease is transmitted at a rate of $\beta = 1.41 \times 10^{-9}$ infections per minute, then multiplying these values we find an average of 2.67×10^{-6} infections per day inside the public market.

Each susceptible buyer is also exposed for one minute on average to an infected buyer in the market. Among the susceptible buyers, the chance of being infected is 1.48×10^{-9}. In addition, 89.25% buyers or 1622.8 out of these susceptible buyers were

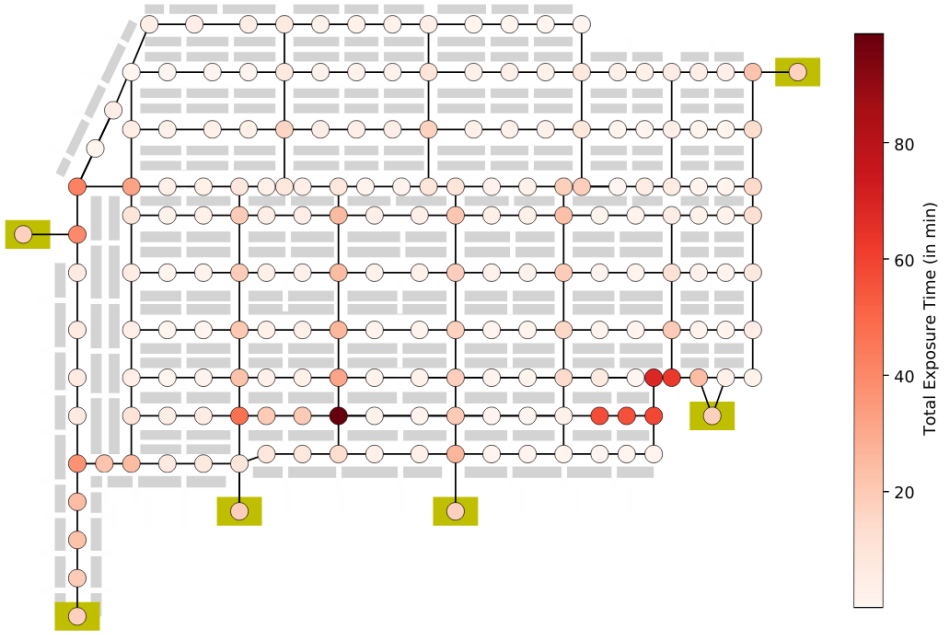
exposed to infected buyers. Among these exposed buyers, the average exposure time to infected buyers is 1.2 minutes. In Figure 4.1a, the exposure time for each exposed buyer appears to follow a negative exponential distribution that reaches up to 16.2 minutes in our simulations. According to many health agencies, a close contact is defined when there is a 15-minute cumulative exposure within 2 meters [8][7][17]. These people have the highest likelihood of getting infected from COVID-19 [6]. Still, during a 24-hour window an additional exposure could happen outside the market or in a different setting, like in another block in the market [8][29]. Therefore, it is still crucial to manage the exposure times of buyers inside the market. The infection chance per susceptible buyer is not largely deviated across 1000 simulations, as shown in Figure 4.1b. We list all output data from our simulations in Table 4.1. Furthermore, the model is able to log the total cumulative exposure time for each node in the market network as well as the average number of buyers visiting every node across 1000 simulation runs. In Figure 4.2, we see the entrances, exits, and along the vertical aisles have the highest exposure times and number of buyer traversals. This information reveals where inside the public market could be changed to reduce exposure times between susceptible and infected buyers.

Table 4.1: Simulation results

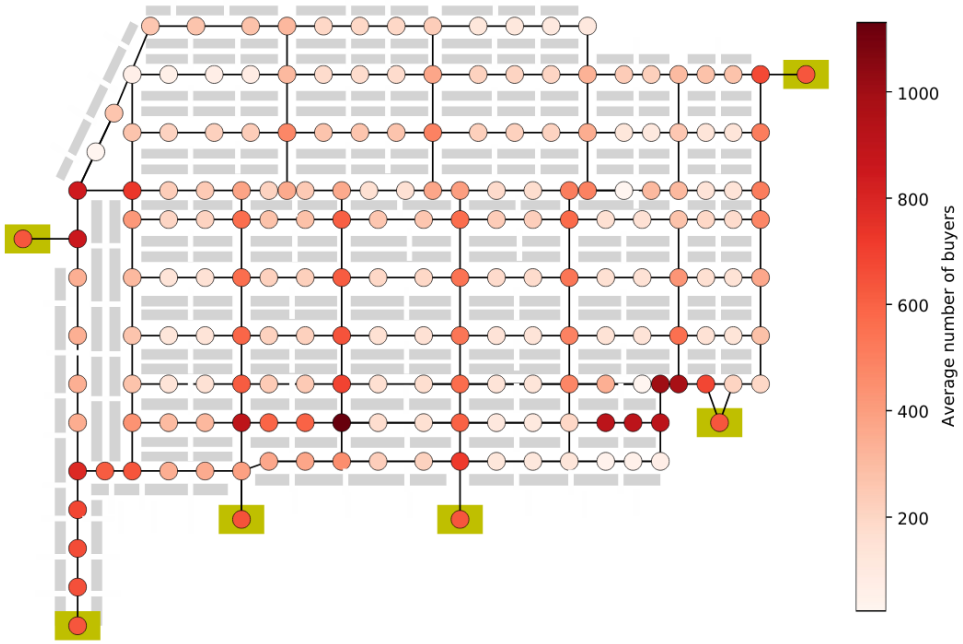
Metric	Mean	Std. Dev.	Min. i	Max. i
Number of buyers	2075.4	45.5	2051	2156
Number of susceptible buyers	1809.1	41.9	1835	1824
Number of infected buyers	266.3	16.1	216	332
Mean number of buyers in store	52.8	1.7	52.9	55.5
Mean buying time (in min)	19.8	0.2	19.6	20
Cumulative exposure time (in min)	1896.1	148.8	1491.5	2364.6
Exposure time per sus. buyer (in min)	1	3.5	0.8	1.3
Exposure time per exposed buyer (in min)	1.2	3.1	0.9	1.5
Percentage of sus. buyers with exposure	89.25%	2.63%	85.07%	93.80%
Number of infections*	2.67×10^{-6}	2.10×10^{-7}	2.10×10^{-6}	3.33×10^{-6}
Chance of infection per sus. buyer**	1.48×10^{-9}	1.16×10^{-10}	1.15×10^{-9}	1.83×10^{-9}

*Cumulative exposure time $\times \beta$

**Num. of infections / Num. of sus. buyers



(a) Average total exposure time per nodes in the network



(b) Average number of buyers per nodes in the network

Figure 4.2: Heatmaps of the market network

In Table 4.2, the exposure times for the mean 1,622.756 exposed buyers across 1000 simulation runs are listed. It appears that most exposed buyers are exposed only for a brief period of time, mainly less than a minute or two. However, there are still buyers who are exposed for more than 15 minutes, which increases their risk of developing infection.

Table 4.2: Exposure times for exposed buyers across 1000 simulations

Exposure time (in min)	Average number of exposed buyers	Exposure time (in min)	Average number of exposed buyers
0-1	904.047	8-9	0.551
1-2	434.181	9-10	0.223
2-3	175.51	10-11	0.098
3-4	67.554	11-12	0.033
4-5	25.532	12-13	0.023
5-6	9.801	13-14	0.009
6-7	3.758	14-15	0.003
7-8	1.43	>15	0.003
		Total	1622.756

In Table 4.3, the chance of infection across 1000 simulations is shown. It appears that most simulation averages 2.4×10^{-6} to 2.9×10^{-6} in terms of infection chance.

Table 4.3: Chance of infection across 1000 simulations

Chance of Infection	Number of simulations	Chance of Infection	Number of simulations
2.0×10^{-6} to 2.1×10^{-6}	4	2.7×10^{-6} to 2.8×10^{-6}	168
2.1×10^{-6} to 2.2×10^{-6}	9	2.8×10^{-6} to 2.9×10^{-6}	125
2.2×10^{-6} to 2.3×10^{-6}	18	2.9×10^{-6} to 3.0×10^{-6}	87
2.3×10^{-6} to 2.4×10^{-6}	60	3.0×10^{-6} to 3.1×10^{-6}	43
2.4×10^{-6} to 2.5×10^{-6}	122	3.1×10^{-6} to 3.2×10^{-6}	13
2.5×10^{-6} to 2.6×10^{-6}	155	3.2×10^{-6} to 3.3×10^{-6}	4
2.6×10^{-6} to 2.7×10^{-6}	190	3.3×10^{-6} to 3.4×10^{-6}	2
		Total	1000

4.1 Reducing market capacity and buyer arrival rate

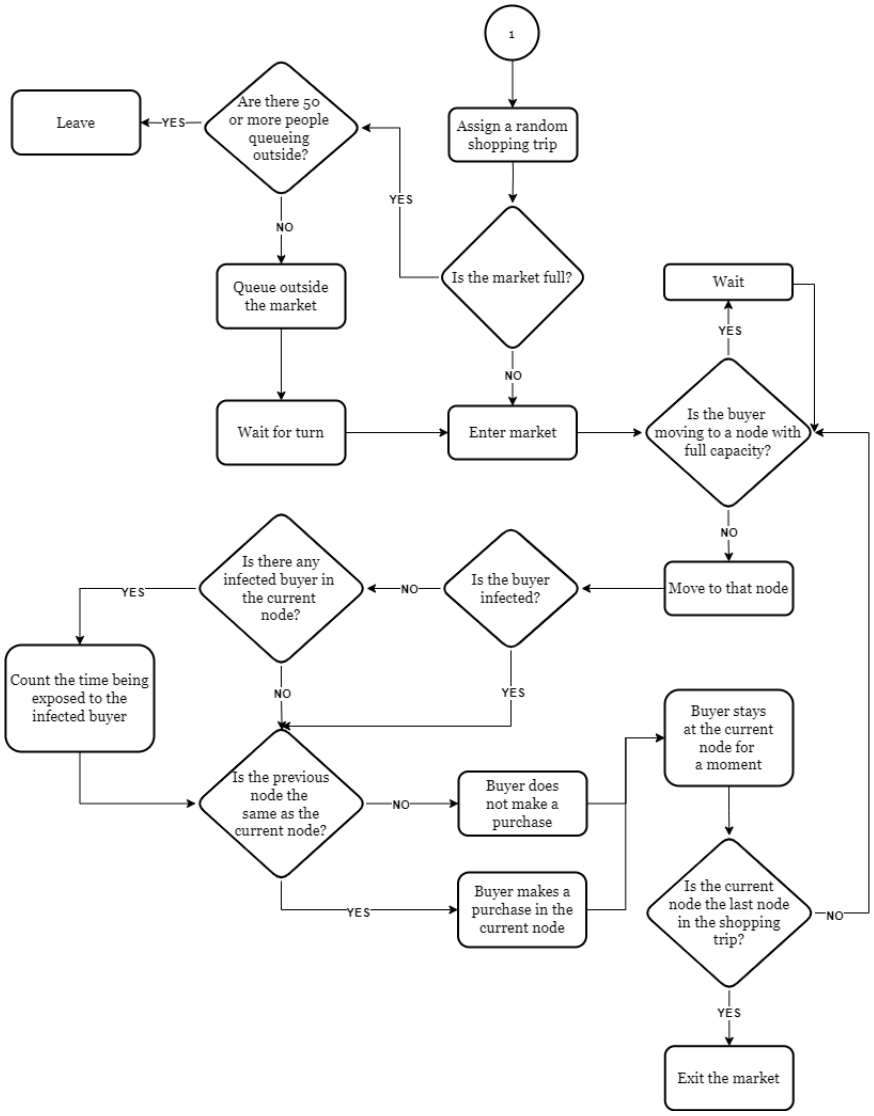
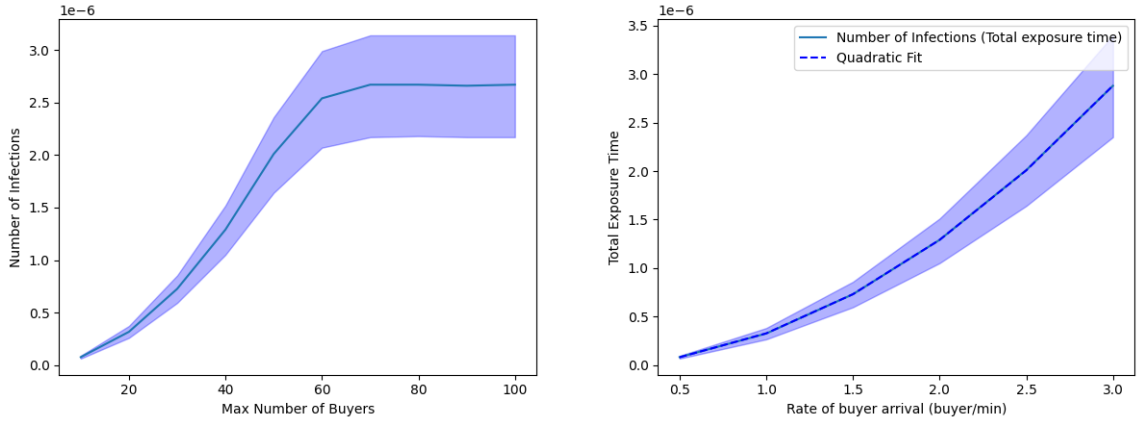


Figure 4.3: Model process with market capacity

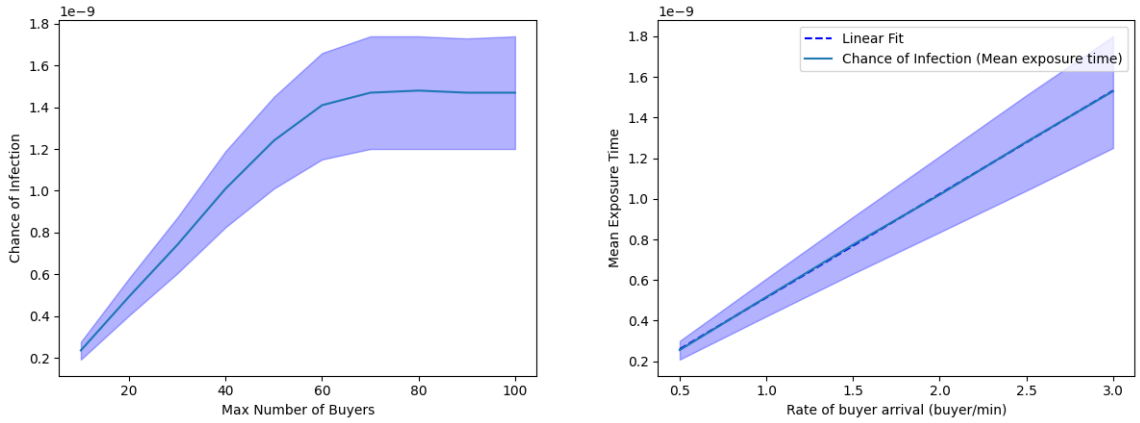
In addition to our model, we create a restriction factor by simulating a maximum capacity of buyers allowed inside the public market while others wait or queue outside. Buyers outside can only enter the market whenever buyers inside the market are below the maximum capacity allowed. We show this additional process in Figure 4.3. In the model, the estimated infection chance and amount of infections significantly decreases when the amount of buyers inside the market is restricted, as shown in Figures 4.4a and

4.4c. Also, the estimated number of infections relatively becomes constant as we set the maximum capacity to 53 coming from the fact that the number of buyers does not exceed this number across our simulations.

Furthermore, we can reduce the number of buyers in the market by limiting the arrival rate λ of entering buyers. Here, the infection chance increase linearly while the number of infections increase quadratically, as shown in Figures 4.4b and 4.4d.



(a) Num. of infections with market capacity (b) Num. of infections with arrival rate



(c) Chances of infection with market capacity (d) Chances of infection with arrival rate

Figure 4.4: Changes in the number and chances of infection with market capacity and buyer arrival rate

In Table 4.4, the number of infections per maximum number of buyers allowed in the market is listed. Likewise, the less buyers are able to be in the market at any given time,

the less number of infections are there. As we also increase the number of maximum capacity above 60, the cumulative exposure times levels to 1,890 minutes. This occurs due to the nature of the simulation as discussed above.

Table 4.4: Number of infections per maximum number of buyers allowed in the market.

Max. num. of buyers	Cumulative exp. time	Num. of infections	CI* (Lower)	CI* (Upper)
10	53.6	7.56×10^{-8}	6.17×10^{-8}	8.90×10^{-8}
20	224.9	3.17×10^{-7}	2.59×10^{-7}	3.73×10^{-7}
30	514.6	7.26×10^{-7}	5.92×10^{-7}	8.54×10^{-7}
40	916.3	1.29×10^{-6}	1.05×10^{-6}	1.52×10^{-6}
50	1422.3	2.01×10^{-6}	1.64×10^{-6}	2.36×10^{-6}
60	1802.4	2.54×10^{-6}	2.07×10^{-6}	2.99×10^{-6}
70	1891.1	2.67×10^{-6}	2.17×10^{-6}	3.14×10^{-6}
80	1892.3	2.67×10^{-6}	2.18×10^{-6}	3.14×10^{-6}
90	1889.9	2.66×10^{-6}	2.17×10^{-6}	3.14×10^{-6}
100	1891.1	2.67×10^{-6}	2.17×10^{-6}	3.14×10^{-6}

* CI = Confidence Interval

Similarly, in Table 4.5, we expect the chances of infection to rise as we increase the buyer capacity of the market.

Table 4.5: Chances of infection per maximum number of buyers allowed in the market.

Max num. of buyers	Num. of sus. buyers	Chance of infection	CI* (Lower)	CI* (Upper)
10	322.10	2.35×10^{-10}	1.91×10^{-10}	2.76×10^{-10}
20	642.64	4.93×10^{-10}	4.02×10^{-10}	5.81×10^{-10}
30	979.73	7.41×10^{-10}	6.04×10^{-10}	8.72×10^{-10}
40	1,277.62	1.01×10^{-9}	8.25×10^{-10}	1.19×10^{-9}
50	1,623.21	1.24×10^{-9}	1.01×10^{-9}	1.45×10^{-9}
60	1,802.38	1.41×10^{-9}	1.15×10^{-9}	1.66×10^{-9}
70	1,808.35	1.47×10^{-9}	1.20×10^{-9}	1.74×10^{-9}
80	1,807.78	1.48×10^{-9}	1.20×10^{-9}	1.74×10^{-9}
90	1,809.14	1.47×10^{-9}	1.20×10^{-9}	1.73×10^{-9}
100	1,808.16	1.47×10^{-9}	1.20×10^{-9}	1.74×10^{-9}

* CI = Confidence Interval

In terms of buyer arrival rates, the results show that the number of infections increase as more buyers enter the market, as shown in Table 4.6.

Table 4.6: Number of infections per rate of arrival of buyers entering the market.

Rate of arrival	Cumulative exp. time	Num. of infections	CI* (Lower)	CI* (Upper)
0.5	56.87	8.02×10^{-8}	6.54×10^{-8}	9.44×10^{-8}
1	229.53	3.24×10^{-7}	2.64×10^{-7}	3.81×10^{-7}
1.5	517.04	7.29×10^{-7}	5.95×10^{-7}	8.58×10^{-7}
2	912.04	1.29×10^{-6}	1.05×10^{-6}	1.51×10^{-6}
2.5	1425.87	2.01×10^{-6}	1.64×10^{-6}	2.37×10^{-6}
3	2045.81	2.88×10^{-6}	2.35×10^{-6}	3.40×10^{-6}

* CI = Confidence Interval

Similarly, the chances of infection increase as more buyers enter the market per minute, as shown in Table 4.7.

Table 4.7: Chances of infection per rate of arrival of buyers entering the market.

Rate of arrival	No. of sus. buyers	Chance of infection	CI* (Lower)	CI* (Upper)
0.5	314.53	2.55×10^{-10}	2.08×10^{-10}	3.00×10^{-10}
1	627.51	5.16×10^{-10}	4.21×10^{-10}	6.07×10^{-10}
1.5	942.43	7.74×10^{-10}	6.31×10^{-10}	9.11×10^{-10}
2	1256.32	1.02×10^{-9}	8.35×10^{-10}	1.21×10^{-9}
2.5	1569.18	1.28×10^{-9}	1.04×10^{-9}	1.51×10^{-9}
3	1883.21	1.53×10^{-9}	1.25×10^{-9}	1.80×10^{-9}

* CI = Confidence Interval

4.2 Enforcing a mandatory facemask policy

We model the enforcement of a facemask policy by reducing the transmission rate as used in [41]. In [25], the relative risk reduction value (RRR) is found to be 0.17. Using this value and multiplying to the transmission rate factor β, we effectively reduce the number of buyer infections and the infection chance per susceptible buyer both by 83%.

In particular, the number of buyer infections reduces from 2.67×10^{-6} to 4.54×10^{-7}. Meanwhile, the infection chance decreases from 1.48×10^{-9} to 2.51×10^{-10}.

4.3 Implementing a one-way aisle setup

In this part, we assess the effectiveness of one-way layout in the public market. Currently, the market network only consists of links that buyers can traverse in both directions. According to [41], one-way setups are believed to help with managing the flow and distribution of buyers. We test this assumption by implementing this to our model by changing the market network into a directed network, where only links near the entry and exit points as well as along the vertical aisles in the network are flowing in a two-way direction.

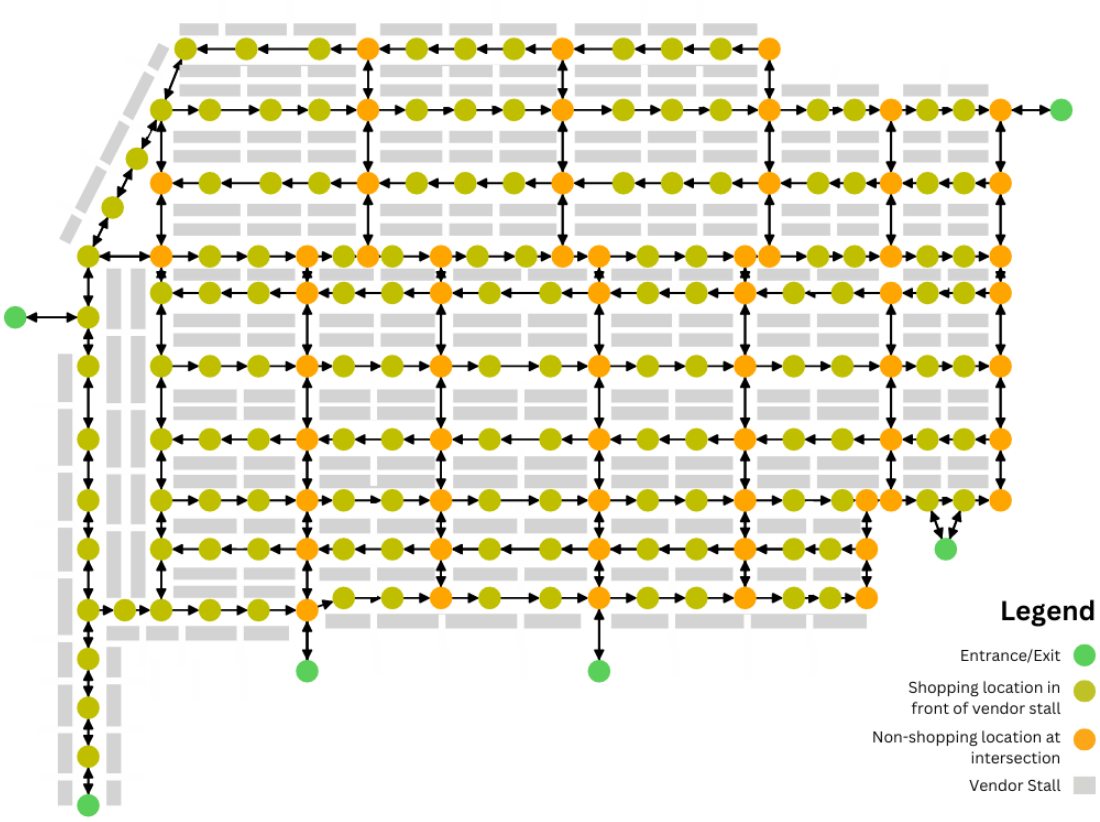
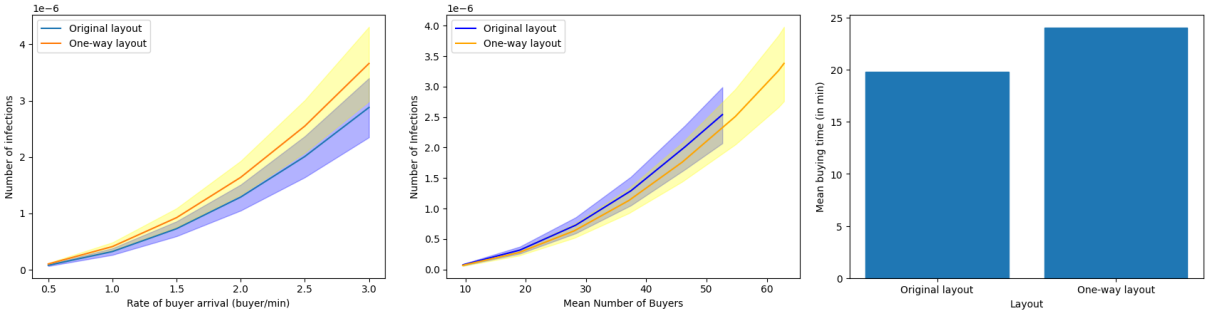


Figure 4.5: One-way aisle setup

In Figure 4.5, we show the one-way aisle layout. In the case of the shopping trips, we pregenerated a different set of 1,000,000 shopping trips to account for the uni-directionality in the one-way aisle layout network. Aside from that, no modifications were made on how buyers pick up their items or any other processes in generating the shopping trips. Instead, we simply change the routes buyers take during their shopping trips.



(a) With buyer arrival rate (b) With market capacity (c) Mean buying time

Figure 4.6: One-way and original layout results with market capacity, buyer arrival rate, and mean buying time.

Based on the model, setting up a one-way aisle seems to increase the number of infections, as shown in Figure 4.6. This is due to the average buying time between the two layouts as shown in Figure 4.6c. In the original layout, there are only an average of 40.42 nodes per buyer. Meanwhile, in the one-way aisle setup, the average buying trip per buyer is 49 nodes. Therefore, the mean buying time increases on average as there are more buyers inside the market at any given time, thereby increasing the chances and frequency of more infections happening. In Figure 4.6b, it appears that the one-way setup of the market slightly helps if there are less than 60 buyers on the store on average. However, with no modified parameters, the number of infections are still much higher in the one-way setup, as shown in Table 4.8. Moreover, the number of infections in the one-way layout rises dramatically when the market capacity rises to 70, as shown in Table 4.9, which effectively increases the overall number of infections.

Table 4.8: One-way simulation results

Metric	Mean	Std. Dev.	Min. i	Max. i
Number of buyers	2074.7	47.1	2106.0	2263.0
Number of susceptible buyers	1808.7	44.2	1890.0	1930.0
Number of infected buyers	266.0	16.3	216.0	333.0
Mean number of buyers in market	62.9	2.4	65.6	67.3
Mean buying time (in min)	24.0	0.3	24.0	24.6
Cumulative exp. time (in min)	2400.6	188.4	2053.7	3342.7
Exposure time per sus. buyer (in min)	1.3	4.3	1.1	1.7
Exposure time per exposed buyer (in min)	1.5	4.0	1.3	2.1
Percentage of sus. buyer with exposure	89.27%	2.63%	82.59%	88.65%
Number of infections	3.38×10^{-6}	2.66×10^{-7}	2.90×10^{-6}	4.71×10^{-6}
Chance of infection per sus. buyer	1.87×10^{-9}	1.47×10^{-10}	1.53×10^{-9}	2.44×10^{-9}

*Cumulative exposure time $\times \beta$
**Num. of infections / Num. of sus. buyers

In Table 4.9, the mean number of buyers in the one-way setup appears to rise and level to 62.9 as we restrict the maximum number of buyers in the market above 60.

Table 4.9: One-way: number of infections per mean number of buyers in the market.

Max. num. of buyers	Mean num. of buyers	Cumulative Exp. Time	Num. of Infections	CI* (Lower)	CI* (Upper)
10	9.5	47.2	6.66×10^{-8}	5.43×10^{-8}	7.84×10^{-8}
20	18.9	196.7	2.77×10^{-7}	2.26×10^{-7}	3.26×10^{-7}
30	28.1	449.1	6.33×10^{-7}	5.16×10^{-7}	7.45×10^{-7}
40	37.1	800.4	1.13×10^{-6}	9.21×10^{-7}	1.33×10^{-6}
50	46.0	1,250.0	1.76×10^{-6}	1.44×10^{-6}	2.08×10^{-6}
60	54.8	1,783.5	2.51×10^{-6}	2.05×10^{-6}	2.96×10^{-6}
70	61.9	2,310.6	3.26×10^{-6}	2.66×10^{-6}	3.84×10^{-6}
80	62.9	2,400.6	3.38×10^{-6}	2.76×10^{-6}	3.98×10^{-6}
90	62.9	2,395.4	3.38×10^{-6}	2.75×10^{-6}	3.98×10^{-6}
100	62.9	2,409.1	3.40×10^{-6}	2.77×10^{-6}	4.00×10^{-6}

* CI = Confidence Interval

In addition, the number of infections increase in the one-way aisle setup as more buyers enter the market, as shown in Table 4.10.

Table 4.10: Number of infections per cumulative exposure time and arrival rate of buyers in the market.

Rate of arrival	Cumulative Exp. Time	Number of Infections	CI* (Lower)	CI* (Upper)
0.5	73.08	1.03×10^{-7}	8.40×10^{-8}	1.21×10^{-7}
1	292.28	4.12×10^{-7}	3.36×10^{-7}	4.85×10^{-7}
1.5	656.05	9.25×10^{-7}	7.54×10^{-7}	1.09×10^{-6}
2	1160.44	1.64×10^{-6}	1.33×10^{-6}	1.93×10^{-6}
2.5	1811.09	2.55×10^{-6}	2.08×10^{-6}	3.01×10^{-6}
3	2598.51	3.66×10^{-6}	2.99×10^{-6}	4.31×10^{-6}

* CI = Confidence Interval

Likewise, the chance of infection per mean number of buyers increase as we increase the number of allowed buyers in the market, as shown in Table 4.11.

Table 4.11: One-way: chance of infection per mean number of buyers in the market.

Max number of buyers	Mean num. of buyers	No. of sus. buyers	Chance of infection	CI* (Lower)	CI* (Upper)
10	9.5	266.3	2.50×10^{-10}	2.04×10^{-10}	2.94×10^{-10}
20	18.9	531.9	5.21×10^{-10}	4.25×10^{-10}	6.14×10^{-10}
30	28.1	794.5	7.97×10^{-10}	6.50×10^{-10}	9.38×10^{-10}
40	37.1	1,058.0	1.07×10^{-9}	8.70×10^{-10}	1.26×10^{-9}
50	46.0	1,317.5	1.34×10^{-9}	1.09×10^{-9}	1.58×10^{-9}
60	54.8	1,574.8	1.60×10^{-9}	1.30×10^{-9}	1.88×10^{-9}
70	61.9	1,785.3	1.82×10^{-9}	1.49×10^{-9}	2.15×10^{-9}
80	62.9	1,809.9	1.87×10^{-9}	1.53×10^{-9}	2.20×10^{-9}
90	62.9	1,808.9	1.87×10^{-9}	1.52×10^{-9}	2.20×10^{-9}
100	62.9	1,809.6	1.88×10^{-9}	1.53×10^{-9}	2.21×10^{-9}

* CI = Confidence Interval

At the same time, the chance of infections increases as we raise the volume of buyers entering the market per minute. We show this in Table 4.12.

Table 4.12: Infection chance per rate of arrival and number of susceptible buyers.

Rate of Arrival	Num. of sus. buyers	Chance of Infection	CI* (Lower)	CI* (Upper)
0.5	314.4	3.28×10^{-10}	2.67×10^{-10}	3.86×10^{-10}
1	628.6	6.56×10^{-10}	5.35×10^{-10}	7.72×10^{-10}
1.5	943.5	9.80×10^{-10}	8.00×10^{-10}	1.15×10^{-9}
2	1254.6	1.30×10^{-9}	1.06×10^{-9}	1.54×10^{-9}
2.5	1571.2	1.63×10^{-9}	1.33×10^{-9}	1.91×10^{-9}
3	1883.6	1.95×10^{-9}	1.59×10^{-9}	2.29×10^{-9}

* CI = Confidence Interval

4.4 Combining Policies

We group the discussed policies we use in Table 4.13.

Table 4.13: Policies

Group 1	Restricting the capacity of buyers allowed inside the market
Group 2	Reducing rate of arrival buyers entering the market
Group 3	Enforcing a mandatory face mask policy
Group 4	Implementing a one-way aisle setup inside the market

Note that these policies are independent of each other, thereby combining them leads to increased effectiveness in managing the number of buyers getting infected and the likelihood or chances of infection in the market. We list a possible combination of policies in Table 4.14. From this, it appears that the one-way setup only increases both number and chances of infection by 27%. On the other hand, the most effective combination of policies appears to be observed when using a facemask policy along with reducing the rate of buyers entering the market. In particular, the number and chances of infections from this combination alone shows a 96% and 92% decrease in the two metrics, respectively.

Table 4.14: Combination of Policies

Policy	Number of infections	Chance of infections	Relative change in number of infections*	Relative change in chance of infections**
Restrict maximum capacity of buyers in the market to 75% of mean number of buyers	1.29×10^{-6}	1.01×10^{-9}	-52%	-31%
Decrease buyer arrival by 50%	6.62×10^{-7}	7.33×10^{-10}	-75%	-50%
Face mask policy	4.54×10^{-7}	2.51×10^{-10}	-83%	-83%
Face mask policy + lessen arrival rate by 50%	1.13×10^{-7}	1.25×10^{-12}	-96%	-92%
Face mask policy + Restrict maximum capacity of buyers in the market to 75% of mean number of buyers	2.20×10^{-7}	1.72×10^{-12}	-92%	-88%
One-way aisle layout	3.38×10^{-6}	1.87×10^{-9}	+27%	+27%

*From original value 2.67×10^{-6}

**From original value 1.48×10^{-9}

Chapter 5

Conclusion and Recommendation

In this study, we have shown an agent-based model drawn upon [41] for simulating virus transmission among buyers in a public market setting. In particular, we presented how COVID-19 is transmitted among buyers in Baguio City's public market as they browse, shop, and come in contact with infectious buyers. Consequently, we used the exposure time or the time spent in the same node by susceptible buyers to infectious buyers and calculated the risk of infection from the disease.

Furthermore, we presented the model by using the data unique to Baguio City's demographics as well as the disease COVID-19. We demonstrated the hotspot locations inside the market where most infections may occur. Moreover, we have implemented a number of policies and estimated how they help in decreasing susceptible buyers' exposure times; that is, the transmission of the disease such as restricting the capacity of buyers inside the market, reducing the rate of arrival of buyers, enforcing a face mask policy, and using a one-way setup. Out of all these policies, we found that using a one-way setup was ineffective in decreasing the amount of time that susceptible buyers are exposed to infected buyers, as it only increases the time buyers spend inside the market which significantly increases their chances to potentially meet infected buyers.

On the other hand, we find that the best policy comes from the combination of controlling the rate of arrival of buyers together with a mandatory face mask policy. This combination lead to dramatically reducing the amount of buyers getting infected as well as their likelihood of contracting the disease inside the market. Finally, we encourage policy makers to use this model as a means to inform them of bottlenecks within markets that lead to congestion of buyers and the best market policy to be applied. For future work, we encourage investigating how vendors also affect the transmission of the disease inside the market and how buyers change their paths according to the crowdedness of a node.

List of References

[1] R. F. Baumeister, *The Psychology of Irrationality: Why People Make Foolish, Self-Defeating Choices*, Oxford University Press eBooks, Feb. 2003. DOI:10.1093/oso/9780199251063.003.0001.

[2] B. Birnir and L. Angheluta, *The build-up of aerosols carrying the sars-cov-2 coronavirus, in poorly ventilated, confined spaces*, Research Square, (2020). doi:http://10.21203/rs.3.rs-110711/v1.

[3] K. Buchanan, *Regulation of wild animal wet markets*, Library of Congress, (2020). https://www.loc.gov/law/help/wet-markets/index.php.

[4] G. Buonanno, L. Stabile, and L. Morawska, *Estimation of airborne viral emission: Quanta emission rate of sars-cov-2 for infection risk assessment*, Environment International, 141 (2020), p. 105794. doi:http://10.1016/j.envint.2020.105794.

[5] V. Cabreza, *Baguio market isn't your usual 'palengke'*, INQUIRER.net, (2021). https://newsinfo.inquirer.net/1409385/baguio-market-isnt-your-usual-palengke.

[6] Centers for Disease Control and Prevention, *How to talk to your close contacts*. https://www.cdc.gov/coronavirus/2019-ncov/downloads/tell-your-contacts.pdf.

[7] Centers for Disease Control and Prevention, *Operational considerations for adapting a contact tracing program to respond to the covid-19 pandemic*. https://www.cdc.gov/coronavirus/2019-ncov/global-covid-19/operational-considerationscontact-tracing.html.

[8] Centers for Disease Control and Prevention, *Public health guidance for community-related exposure*. https://www.cdc.gov/coronavirus/2019-ncov/php/public-health-recommendations.html.

[9] Centers for Disease Control and Prevention, *Scientific brief: Sars-cov-2 and potential airborne transmission*, February 2021. https://www.cdc.gov/coronavirus/2019-ncov/more/scientific-brief-sars-cov-2.html.

[10] Centers for Disease Control and Prevention, *Covid-19 timeline*, Centers for Disease Control and Prevention, (2023). https://www.cdc.gov/museum/timeline/covid19.html.

[11] Columbia Public Health, *Agent-based modeling*. https://www.publichealth.columbia.edu/research/population-health-methods/agent-based-modeling.

[12] I. Cooper, A. Mondal, and C. G. Antonopoulos, *A sir model assumption for the spread of covid-19 in different communities*, Chaos, Solitons and Fractals, 139 (2020), p. 110057. doi:http://10.1016/j.chaos.2020.110057.

[13] H. Dai and B. Zhao, *Association of the infection probability of covid-19 with ventilation rates in confined spaces*, Building Simulation, 13 (2020), pp. 1321–1327. doi:http://10.1007/s12273-020-0703-5.

[14] D. Dennis Jr. and P. M. Geminiano, *Baguio limits crowds at public market*, Philippine News Agency, (2020). https://www.pna.gov.ph/articles/1097661.

[15] Department of Health Philippines, *Covid-19 case tracker*, Jan. 2024. https://doh.gov.ph/diseases/covid-19/covid-19-case-tracker/.

[16] Google, *Google maps*, 2024. https://maps.app.goo.gl/RYHenJSW9KynTqXRA.

[17] GOV.UK, *Guidance for contacts of people with confirmed coronavirus (covid-19) infection who do not live with the person*. https://www.gov.uk/government/publications/guidance-for-contacts-of-people-with-possible-or-confirmed-coronavirus-covid-19-infection-who-do-not-live-with-the-person.

[18] M. J. J. Gumasing, Y. T. Prasetyo, S. F. Persada, A. K. S. Ong, M. N. Young, R. Nadlifatin, and A. A. N. P. Redi, *Using online grocery applications during the covid-19 pandemic: Their relationship with open innovation*, Journal of Open Innovation: Technology, Market, and Complexity, 8 (2022), p. 93. doi:http://10.3390/joitmc8020093.

[19] R. A. Hammond, *Considerations and Best Practices in Agent-Based Modeling to Inform Policy*, National Academies Press (US), 2015. https://www.ncbi.nlm.nih.gov/books/NBK305917/.

[20] Y. Hao, Y. Wang, M. Wang, L. Zhou, J. Shi, J. Cao, and D. Wang, *The origins of covid-19 pandemic: A brief overview*, Transboundary and Emerging Diseases, 69 (2022). doi:http://10.1111/tbed.14732.

[21] H. Hu, *Fluid mechanics*, Elsevier, (2012). doi:http://10.1016/c2009-0-63410-3.

[22] D. Kaziyeva, P. Stutz, G. Wallentin, and M. Loidl, *Large-scale agent-based simulation model of pedestrian traffic flows*, Computers, Environment and Urban Systems, 105 (2023), p. 102021–102021. doi:http://10.1016/j.compenvurbsys.2023.102021.

[23] C. J. Lasco Gideon, *Food (in)security in times of pandemic*, INQUIRER.net, (2020). https://opinion.inquirer.net/128372/food-insecurity-in-times-of-pandemic.

[24] Z. Lau, K. Kaouri, and I. Griffiths, *Modelling airborne transmission of covid-19 in indoor spaces using an advection-diffusion-reaction equation*, arXiv preprint, (2020). arXiv:201212267.

[25] T. Li, Y. Liu, M. Li, X. Qian, and S. Dai, *Mask or no mask for covid-19: A public health and market study.*, PloS one, (2020). https://doi.org/10.1371/journal.pone.0237691.

[26] R. O. Macalinao, J. C. Malaguit, and D. S. Lutero, *Agent-based modeling of covid-19 transmission in philippine classrooms*, Frontiers in Applied Mathematics and Statistics, 8 (2022). doi:http://10.3389/fams.2022.886082.

[27] Philippine Statistics Authority, *Highlights on household population, number of household, and average household size of baguio city (2020 census of population and housing) — philippine statistics authority*, 2022. https://rssocar.psa.gov.ph/content/highlights-household-population-number-household-and-average-household-size-baguio-city.

[28] S. Plata, S. Sarma, M. Lancelot, K. Bagrova, and D. Romano-Critchley, *Simulating human interactions in supermarkets to measure the risk of covid-19 contagion at scale*, arXiv preprint, (2020). arXiv:200615213.

[29] J. C. Pringle, J. Leikauskas, S. Ransom-Kelley, B. Webster, S. Santos, H. Fox, S. Marcoux, P. Kelso, and N. Kwit, *Covid-19 in a correctional facility employee following multiple brief exposures to persons with covid-19—vermont, july–august 2020*, Morbidity and Mortality Weekly Report, 69 (2020), p. 1569. doi:http://10.15585/mmwr.mm6943e1.

[30] R. Shadi, A. Fakharian, and H. Khaloozadeh, *Mathematical modeling of the novel coronavirus pandemic in iran: A model with vaccination*, 2022 8th International Conference on Control, Instrumentation and Automation (ICCIA), (2022), pp. 1–6.

[31] S. Shao, D. Zhou, R. He, J. Li, S. Zou, K. Mallery, S. Kumar, S. Yang, and J. Hong, *Risk assessment of airborne transmission of covid-19 by asymptomatic individuals under different practical settings*, Journal of Aerosol Science, 151 (2021), p. 105661. doi:http://10.1016/j.jaerosci.2020.105661.

[32] H. Sorensen, S. Bogomolova, K. Anderson, G. Trinh, A. Sharp, and R. Kennedy, *Fundamental patterns of in-store shopper behavior*, Journal of Retailing and Consumer Services, 37 (2017), p. 182–194. doi:http://10.1016/j.jretconser.2017.02.003.

[33] C. Sun and Z. Zhai, *The efficacy of social distance and ventilation effectiveness in preventing covid-19 transmission*, Sustainable cities and society, 62 (2020), p. 102390. http://10.1016/j.scs.2020.102390.

[34] B. Tang, N. L. Bragazzi, Q. Li, S. Tang, Y. Xiao, and J. Wu, *An updated estimation of the risk of transmission of the novel coronavirus (2019-ncov)*, Infectious Disease Modelling, 5 (2020), pp. 248–255. DOI:http://10.1016/j.idm.2020.02.001.

[35] J. Villanueva, J. Austria, K. Faronilo, A. Sunga-Lim, E. Replan, J. Sevilla-Nastor, R. Abuyan, and N. Peyraube, *Effect of lockdown on food security during the covid-19 pandemic in the philippines: Two months after implementation*, Philippine Journal of Science, (2022). https://philjournalsci.dost.gov.ph/images/pdf/pjs_pdf/vol151no4/effect_of_lockdown_on_food_security_during_the_COVID-19_Pandemic_in_the_Phils_.pdf.

[36] V. Vuorinen, M. Aarnio, M. Alava, V. Alopaeus, N. Atanasova, M. Auvinen, N. Balasubramanian, H. Bordbar, P. Erästö, R. Grande, N. Hayward, A. Hellsten, S. Hostikka, J. Hokkanen, O. Kaario, A. Karvinen, I. Kivistö, M. Korhonen, R. Kosonen, J. Kuusela, S. Lestinen, E. Laurila, H. J. Nieminen, P. Peltonen, J. Pokki, A. Puisto, P. Råback, H. Salmenjoki, T. Sironen, and M. Österberg, *Modelling aerosol transport and virus exposure with numerical simulations in relation to sars-cov-2 transmission by inhalation indoors*, Safety Science, 130 (2020), p. 104866. doi:http://10.1016/j.ssci.2020.104866.

[37] World Health Organization, *Transmission of sars-cov-2: Implications for infection prevention precautions*, 2020. https://www.who.int/news-room/commentaries/detail/transmission-of-sars-cov-2-implications-for-infection-prevention-precautions.

[38] World Health Organization, *Coronavirus disease (covid-19): How is it transmitted?*, December 2021. https://www.who.int/news-room/q-a-detail/coronavirus-disease-covid-19-how-is-it-transmitted.

[39] World Health Organization, *Covid-19 deaths dashboard*, 2024. https://data.who.int/dashboards/covid19/deaths.

[40] M. Worobey, J. I. Levy, L. M. Serrano, A. Crits-Christoph, J. E. Pekar, S. A. Goldstein, A. L. Rasmussen, M. U. G. Kraemer, C. Newman, M. P. G. Koopmans, M. A. Suchard, J. O. Wertheim, P. Lemey, D. L. Robertson, R. F. Garry, E. C. Holmes, A. Rambaut, and K. G. Andersen, *The huanan seafood wholesale market in wuhan was the early epicenter of the covid-19 pandemic*, Science, 377 (2022), p. 951–959. doi:http://10.1126/science.abp8715.

[41] F. Ying and N. O'Clery, *Modelling covid-19 transmission in supermarkets using an agent-based model*, PLOS ONE, 16 (2021), p. e0249821. doi:http://10.1371/journal.pone.0249821.

Appendix A

Tables for Original Market Layout

A.1 With no modified parameters

Table A.1: With no modified parameters.

Metric	Mean	Std. dev.
num_cust	2075.356	45.5195
num_S	1809.092	41.9428
num_I	266.264	16.1478
total_exposure_time	1896.108	148.8122
num_cust_w_contact	1622.756	49.0043
mean_num_cust_in_store	52.8369	1.7191
max_num_cust_in_store	75.423	4.1021
num_contacts	7740.442	597.3480
mean_shopping_time	19.8165	0.2375
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	768.54	18.2538
total_time_crowded	143.2667	15.0879

A.2 With maximum capacity of buyers

Table A.2: With maximum capacity of 10 buyers

Metric	**Mean**	**Std. dev.**
num_cust	369.012	10.1056
num_S	322.101	10.8469
num_I	46.911	6.7323
total_exposure_time	53.6172	9.6833
num_cust_w_contact	113.217	14.1994
mean_num_cust_in_store	9.6065	0.1491
max_num_cust_in_store	10.0	0.0
num_contacts	218.471	36.3473
mean_shopping_time	19.8314	0.5466
num_waiting_people	358.996	10.1066
mean_waiting_time	92.9525	2.5182
store_open_length	748.51	14.5359
total_time_crowded	0.1226	0.1779

Table A.3: With maximum capacity of 20 buyers

Metric	**Mean**	**Std. dev.**
num_cust	737.085	14.2560
num_S	642.643	15.5655
num_I	94.442	9.6272
total_exposure_time	225.6326	25.9840
num_cust_w_contact	375.946	24.0397
mean_num_cust_in_store	19.0263	0.3619
max_num_cust_in_store	20.0	0.0
num_contacts	922.461	99.8947
mean_shopping_time	19.7975	0.3862

num_waiting_people	716.827	14.2068
mean_waiting_time	47.6487	0.9852
store_open_length	755.88	16.7901
total_time_crowded	2.4119	0.8777

Table A.4: With maximum capacity of 30 buyers

Metric	**Mean**	**Std. dev.**
num_cust	1102.106	17.3848
num_S	960.603	18.7350
num_I	141.503	11.2422
total_exposure_time	514.5982	44.4568
num_cust_w_contact	696.607	26.2770
mean_num_cust_in_store	28.2970	0.5818
max_num_cust_in_store	30.0	0.0
num_contacts	2102.269	174.0563
mean_shopping_time	19.8095	0.3125
num_waiting_people	1070.382	17.5913
mean_waiting_time	31.5650	0.5613
store_open_length	761.43	17.7361
total_time_crowded	12.4316	2.1118

Table A.5: With maximum capacity of 40 buyers

Metric	**Mean**	**Std. dev.**
num_cust	1465.331	20.9474
num_S	1277.62	22.3175
num_I	187.711	13.0046
total_exposure_time	916.3445	64.8644
num_cust_w_contact	1037.323	26.6545
mean_num_cust_in_store	37.4601	0.7872
max_num_cust_in_store	40.0	0.0
num_contacts	3742.037	258.9347
mean_shopping_time	19.8128	0.2813
num_waiting_people	1417.78	22.2799
mean_waiting_time	22.9533	0.5155
store_open_length	764.93	17.9532
total_time_crowded	36.9649	3.7637

Table A.6: With maximum capacity of 50 buyers

Metric	**Mean**	**Std. dev.**
num_cust	1822.266	22.8452
num_S	1589.663	24.1681
num_I	232.603	14.6238
total_exposure_time	1422.3492	89.9627
num_cust_w_contact	1377.844	26.9713
mean_num_cust_in_store	46.4208	1.0766
max_num_cust_in_store	50.0	0.0
num_contacts	5808.895	358.2819
mean_shopping_time	19.8278	0.2472
num_waiting_people	1733.816	45.0272
mean_waiting_time	16.3343	1.0644

store_open_length	768.51	19.3818
total_time_crowded	82.9970	6.1144

Table A.7: With maximum capacity of 60 buyers

Metric	**Mean**	**Std. dev.**
num_cust	2067.464	41.1044
num_S	1802.376	39.6767
num_I	265.088	15.5922
total_exposure_time	1860.5486	135.1992
num_cust_w_contact	1614.095	43.0409
mean_num_cust_in_store	52.6357	1.6822
max_num_cust_in_store	60.0	0.0
num_contacts	7591.929	541.8221
mean_shopping_time	19.8337	0.2374
num_waiting_people	1116.766	313.5146
mean_waiting_time	3.5730	1.7783
store_open_length	769.31	18.1536
total_time_crowded	134.6837	12.3159

Table A.8: With maximum capacity of 70 buyers

Metric	**Mean**	**Std. dev.**
num_cust	2074.169	45.7203
num_S	1808.351	43.4067
num_I	265.818	16.0792
total_exposure_time	1891.1243	146.0356
num_cust_w_contact	1622.105	47.4825
mean_num_cust_in_store	52.7735	1.8404
max_num_cust_in_store	69.9	0.4628

num_contacts	7717.515	582.4103
mean_shopping_time	19.8264	0.2441
num_waiting_people	122.905	86.5800
mean_waiting_time	0.9813	0.5123
store_open_length	769.7	20.5459
total_time_crowded	141.5389	14.9778

Table A.9: With maximum capacity of 80 buyers

Metric	**Mean**	**Std. dev.**
num_cust	2073.609	44.5083
num_S	1807.777	41.9555
num_I	265.832	17.0909
total_exposure_time	1892.2769	146.7194
num_cust_w_contact	1621.438	47.0379
mean_num_cust_in_store	52.7848	1.7940
max_num_cust_in_store	74.906	3.4913
num_contacts	7721.521	588.1228
mean_shopping_time	19.8228	0.2381
num_waiting_people	5.154	12.4376
mean_waiting_time	0.1613	0.2946
store_open_length	769.26	19.4636
total_time_crowded	142.0539	14.7320

Table A.10: With maximum capacity of 90 buyers

Metric	**Mean**	**Std. dev.**
num_cust	2074.849	45.3576
num_S	1809.142	42.5402
num_I	265.707	16.5998
total_exposure_time	1889.8880	145.3584
num_cust_w_contact	1621.318	47.2095
mean_num_cust_in_store	52.8403	1.7277
max_num_cust_in_store	75.286	4.0173
num_contacts	7713.346	582.2276
mean_shopping_time	19.8230	0.2380
num_waiting_people	0.062	0.9301
mean_waiting_time	0.0028	0.0340
store_open_length	768.72	17.9746
total_time_crowded	142.7427	14.7817

Table A.11: With maximum capacity of 100 buyers

Metric	**Mean**	**Std. dev.**
num_cust	2073.827	47.0630
num_S	1808.159	43.6784
num_I	265.668	16.1771
total_exposure_time	1891.0635	150.3996
num_cust_w_contact	1620.941	50.5017
mean_num_cust_in_store	52.7792	1.8405
max_num_cust_in_store	75.056	4.0089
num_contacts	7713.335	600.8004
mean_shopping_time	19.8189	0.2389
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0

store_open_length	769.23	19.8721
total_time_crowded	142.5085	15.3474

Table A.12: With maximum capacity of 75% of mean number of buyers

Metric	**Mean**	**Std. dev.**
num_cust	1464.860	20.079
num_S	1276.832	21.829
num_I	188.028	13.083
total_exposure_time	918.089	65.237
num_cust_w_contact	1036.424	26.545
mean_num_cust_in_store	37.414	0.840
max_num_cust_in_store	40.0	0.0
num_contacts	3748.345	259.878
mean_shopping_time	19.818	0.272
num_waiting_people	1417.217	21.166
mean_waiting_time	22.968	0.480
store_open_length	765.870	19.340
total_time_crowded	36.770	3.896

A.3 With modified rate of buyer arrival

Table A.13: With rate of 0.5 buyer per minute

Metric	**Mean**	**Std. dev.**
num_cust	360.613	18.4288
num_S	314.526	17.1095
num_I	46.087	6.6404
total_exposure_time	56.8693	11.8952
num_cust_w_contact	117.344	17.8011
mean_num_cust_in_store	9.4292	0.5697
max_num_cust_in_store	17.858	1.8856
num_contacts	232.115	46.0672
mean_shopping_time	19.8087	0.5789
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	747.92	15.3724
total_time_crowded	0.1973	0.2438

Table A.14: With rate of 1 buyer per minute

Metric	**Mean**	**Std. dev.**
num_cust	720.353	26.8383
num_S	627.506	25.5261
num_I	92.847	9.7514
total_exposure_time	229.5347	32.1735
num_cust_w_contact	373.145	31.6852
mean_num_cust_in_store	18.6309	0.8421
max_num_cust_in_store	30.774	2.4209
num_contacts	938.625	126.7210
mean_shopping_time	19.8106	0.3948

num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	756.39	16.7257
total_time_crowded	3.0514	1.1337

Table A.15: With rate of 1.5 buyers per minute

Metric	**Mean**	**Std. dev.**
num_cust	1081.193	32.6166
num_S	942.429	30.2516
num_I	138.764	11.4254
total_exposure_time	517.0389	56.0110
num_cust_w_contact	685.823	39.1410
mean_num_cust_in_store	27.7996	1.0911
max_num_cust_in_store	42.991	2.9664
num_contacts	2110.07	222.2506
mean_shopping_time	19.8274	0.3276
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	761.37	17.6787
total_time_crowded	13.8179	2.9734

Table A.16: With rate of 2 buyers per minute

Metric	**Mean**	**Std. dev.**
num_cust	1440.647	37.6318
num_S	1256.318	35.1401
num_I	184.329	13.2389
total_exposure_time	912.0352	85.5039
num_cust_w_contact	1019.127	41.9635
mean_num_cust_in_store	36.8878	1.3729
max_num_cust_in_store	54.943	3.5146
num_contacts	3722.979	334.3668
mean_shopping_time	19.8053	0.2913
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	764.15	18.6286
total_time_crowded	39.4474	6.0046

Table A.17: With rate of 2.5 buyers per minute

Metric	**Mean**	**Std. dev.**
num_cust	1799.863	43.5672
num_S	1569.183	40.4662
num_I	230.68	15.4294
total_exposure_time	1425.8746	121.8322
num_cust_w_contact	1360.275	48.1512
mean_num_cust_in_store	45.8870	1.6538
max_num_cust_in_store	66.533	3.8217
num_contacts	5817.196	492.1719
mean_shopping_time	19.8181	0.2621
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0

store_open_length	767.61	19.5748
total_time_crowded	87.2221	10.8864

Table A.18: With rate of 3 buyers per minute

Metric	**Mean**	**Std. dev.**
num_cust	2159.498	49.4000
num_S	1883.213	45.3820
num_I	276.285	16.4171
total_exposure_time	2045.8102	154.6243
num_cust_w_contact	1704.405	50.4552
mean_num_cust_in_store	54.9410	1.8203
max_num_cust_in_store	78.074	4.0972
num_contacts	8349.259	618.5260
mean_shopping_time	19.8154	0.2283
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	769.33	18.3954
total_time_crowded	163.5804	16.7373

Table A.19: Reducing 50% buyer arrival rate

Metric	**Mean**	**Std. dev.**
num_cust	1035.439	31.932
num_S	903.352	30.245
num_I	132.087	11.913
total_exposure_time	469.564	53.530
num_cust_w_contact	641.840	38.583
mean_num_cust_in_store	26.652	1.052
max_num_cust_in_store	41.372	2.878

num_contacts	1916.577	213.215
mean_shopping_time	19.799	0.336
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	759.550	17.222
total_time_crowded	11.838	2.654

Appendix B

Tables for One-way Market Layout

B.1 With no modified parameters

Table B.1: With no modified parameters

Metric	**Mean**	**Std. dev.**
num_cust	2074.661	47.1136
num_S	1808.655	44.2480
num_I	266.006	16.3454
total_exposure_time	2400.5756	188.3669
num_cust_w_contact	1635.933	49.4605
mean_num_cust_in_store	62.8633	2.3714
max_num_cust_in_store	89.072	4.5828
num_contacts	9801.920	757.9850
mean_shopping_time	24.0538	0.2920
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	784.52	23.1799
total_time_crowded	167.4112	17.2176

B.2 With maximum capacity of buyers

Table B.2: With maximum capacity of 10 buyers

Metric	**Mean**	**Std. dev.**
num_cust	305.458	9.2262
num_S	266.274	9.6761
num_I	39.184	6.0687
total_exposure_time	47.2159	9.8450
num_cust_w_contact	83.378	12.3505
mean_num_cust_in_store	9.5480	0.1844
max_num_cust_in_store	10.0	0.0
num_contacts	192.79	37.3437
mean_shopping_time	24.0433	0.7358
num_waiting_people	295.454	9.2275
mean_waiting_time	111.1757	3.2185
store_open_length	755.91	18.3223
total_time_crowded	0.0723	0.1431

Table B.3: With maximum capacity of 20 buyers

Metric	**Mean**	**Std. dev.**
num_cust	609.996	13.4876
num_S	531.932	14.4154
num_I	78.064	8.5026
total_exposure_time	196.6821	24.2079
num_cust_w_contact	283.515	20.5462
mean_num_cust_in_store	18.8711	0.4221
max_num_cust_in_store	20.0	0.0
num_contacts	803.443	94.4218
mean_shopping_time	24.0163	0.5329

num_waiting_people	589.862	13.4612
mean_waiting_time	57.6838	1.3210
store_open_length	765.25	20.0883
total_time_crowded	1.4018	0.6599

Table B.4: With maximum capacity of 30 buyers

Metric	**Mean**	**Std. dev.**
num_cust	911.319	16.3696
num_S	794.543	17.194
num_I	116.776	10.785
total_exposure_time	449.0794	43.6916
num_cust_w_contact	535.622	25.8082
mean_num_cust_in_store	28.0522	0.6700
max_num_cust_in_store	30.0	0.0
num_contacts	1834.916	174.5184
mean_shopping_time	24.0517	0.4376
num_waiting_people	880.58	16.3945
mean_waiting_time	38.6094	0.7626
store_open_length	771.26	21.0679
total_time_crowded	7.0664	1.5981

Table B.5: With maximum capacity of 40 buyers

Metric	**Mean**	**Std. dev.**
num_cust	1213.273	18.6464
num_S	1057.983	20.2945
num_I	155.29	11.5639
total_exposure_time	800.4401	63.0587
num_cust_w_contact	812.698	26.0161
mean_num_cust_in_store	37.1033	0.9363
max_num_cust_in_store	40.0	0.0
num_contacts	3264.975	251.4195
mean_shopping_time	24.0275	0.3731
num_waiting_people	1170.186	18.8459
mean_waiting_time	28.5590	0.5449
store_open_length	775.65	22.0196
total_time_crowded	21.3830	2.8266

Table B.6: With maximum capacity of 50 buyers

Metric	**Mean**	**Std. dev.**
num_cust	1511.472	21.208
num_S	1317.451	23.350
num_I	194.021	13.296
total_exposure_time	1250.026	88.217
num_cust_w_contact	1095.391	25.924
mean_num_cust_in_store	45.976	1.252
max_num_cust_in_store	50.0	0.0
num_contacts	5102.303	348.491
mean_shopping_time	24.032	0.337
num_waiting_people	1450.995	23.357
mean_waiting_time	22.141	0.541

store_open_length	780.33	23.680
total_time_crowded	49.317	4.497

Table B.7: With maximum capacity of 60 buyers

Metric	**Mean**	**Std. dev.**
num_cust	1805.171	24.035
num_S	1574.789	25.053
num_I	230.382	14.131
total_exposure_time	1783.519	110.759
num_cust_w_contact	1377.617	26.240
mean_num_cust_in_store	54.800	1.507
max_num_cust_in_store	60.0	0.0
num_contacts	7279.902	439.475
mean_shopping_time	24.055	0.319
num_waiting_people	1706.688	43.724
mean_waiting_time	16.646	0.944
store_open_length	782.77	23.732
total_time_crowded	95.392	6.439

Table B.8: With maximum capacity of 70 buyers

Metric	**Mean**	**Std. dev.**
num_cust	2047.21	33.575
num_S	1785.274	32.998
num_I	261.936	15.634
total_exposure_time	2310.563	153.298
num_cust_w_contact	1607.596	34.723
mean_num_cust_in_store	61.943	2.088
max_num_cust_in_store	70.0	0.0

num_contacts	9431.667	613.373
mean_shopping_time	24.060	0.296
num_waiting_people	1450.633	307.294
mean_waiting_time	6.017	2.732
store_open_length	785.84	26.412
total_time_crowded	152.198	11.337

Table B.9: With maximum capacity of 80 buyers

Metric	**Mean**	**Std. dev.**
num_cust	2074.907	44.863
num_S	1809.882	41.673
num_I	265.025	16.058
total_exposure_time	2400.592	184.915
num_cust_w_contact	1636.537	46.558
mean_num_cust_in_store	62.862	2.463
max_num_cust_in_store	79.997	0.055
num_contacts	9792.056	740.191
mean_shopping_time	24.060	0.287
num_waiting_people	267.592	147.748
mean_waiting_time	1.420	0.658
store_open_length	784.97	27.400
total_time_crowded	166.205	16.195

Table B.10: With maximum capacity of 90 buyers

Metric	**Mean**	**Std. dev.**
num_cust	2073.619	45.945
num_S	1808.885	42.851
num_I	264.734	16.422
total_exposure_time	2395.381	181.690
num_cust_w_contact	1634.959	47.148
mean_num_cust_in_store	62.822	2.349
max_num_cust_in_store	87.543	2.700
num_contacts	9777.151	731.806
mean_shopping_time	24.055	0.284
num_waiting_people	18.865	27.712
mean_waiting_time	0.421	0.450
store_open_length	784.72	24.110
total_time_crowded	167.021	16.945

Table B.11: With maximum capacity of 100 buyers

Metric	**Mean**	**Std. dev.**
num_cust	2075.321	46.483
num_S	1809.638	44.210
num_I	265.683	16.280
total_exposure_time	2409.073	182.609
num_cust_w_contact	1637.498	49.027
mean_num_cust_in_store	62.903	2.274
max_num_cust_in_store	88.875	4.344
num_contacts	9832.902	732.732
mean_shopping_time	24.054	0.280
num_waiting_people	0.563	3.413
mean_waiting_time	0.020	0.101

store_open_length	784.15	23.610
total_time_crowded	167.365	17.116

B.3 With modified rate of buyer arrival

Table B.12: With rate of 0.5 buyer per minute

Metric	**Mean**	**Std. dev.**
num_cust	360.413	18.690
num_S	314.372	17.392
num_I	46.041	6.827
total_exposure_time	73.075	15.589
num_cust_w_contact	120.44	18.337
mean_num_cust_in_store	11.270	0.684
max_num_cust_in_store	20.568	2.141
num_contacts	297.905	60.363
mean_shopping_time	24.038	0.693
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	759.27	19.045
total_time_crowded	0.244	0.281

Table B.13: With rate of 1 buyer per minute

Metric	**Mean**	**Std. dev.**
num_cust	720.999	26.863
num_S	628.603	25.307
num_I	92.396	9.276
total_exposure_time	292.277	39.788
num_cust_w_contact	380.97	31.109
mean_num_cust_in_store	22.284	1.054
max_num_cust_in_store	35.928	2.680
num_contacts	1194.232	157.736
mean_shopping_time	24.050	0.483
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	768.47	19.762
total_time_crowded	3.526	1.199

Table B.14: With rate of 1.5 buyers per minute

Metric	**Mean**	**Std. dev.**
num_cust	1082.03	33.016
num_S	943.46	29.971
num_I	138.57	11.721
total_exposure_time	656.051	71.885
num_cust_w_contact	696.561	39.542
mean_num_cust_in_store	33.199	1.389
max_num_cust_in_store	50.569	3.425
num_contacts	2679.319	287.451
mean_shopping_time	24.011	0.385
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0

store_open_length	773.03	20.947
total_time_crowded	16.060	3.256

Table B.15: With rate of 2 buyers per minute

Metric	**Mean**	**Std. dev.**
num_cust	1439.069	37.912
num_S	1254.585	36.038
num_I	184.484	13.600
total_exposure_time	1160.443	105.144
num_cust_w_contact	1030.577	42.971
mean_num_cust_in_store	43.897	1.751
max_num_cust_in_store	64.365	3.833
num_contacts	4736.385	423.238
mean_shopping_time	24.046	0.349
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	778.81	24.708
total_time_crowded	46.044	6.532

Table B.16: With rate of 2.5 buyers per minute

Metric	**Mean**	**Std. dev.**
num_cust	1801.696	44.518
num_S	1571.196	41.695
num_I	230.5	15.495
total_exposure_time	1811.090	152.385
num_cust_w_contact	1375.533	47.256
mean_num_cust_in_store	54.752	2.052
max_num_cust_in_store	78.705	4.296

num_contacts	7391.380	619.547
mean_shopping_time	24.051	0.316
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	781.880	22.040
total_time_crowded	102.339	11.874

Table B.17: With rate of 3 buyers per minute

Metric	**Mean**	**Std. dev.**
num_cust	2159.057	46.448
num_S	1883.619	43.865
num_I	275.438	16.820
total_exposure_time	2598.506	199.821
num_cust_w_contact	1716.492	47.857
mean_num_cust_in_store	65.468	2.352
max_num_cust_in_store	92.287	4.730
num_contacts	10605.748	804.067
mean_shopping_time	24.081	0.285
num_waiting_people	0.0	0.0
mean_waiting_time	0.0	0.0
store_open_length	784.820	22.642
total_time_crowded	192.566	18.829

Appendix C

Network Map with Node Numbers

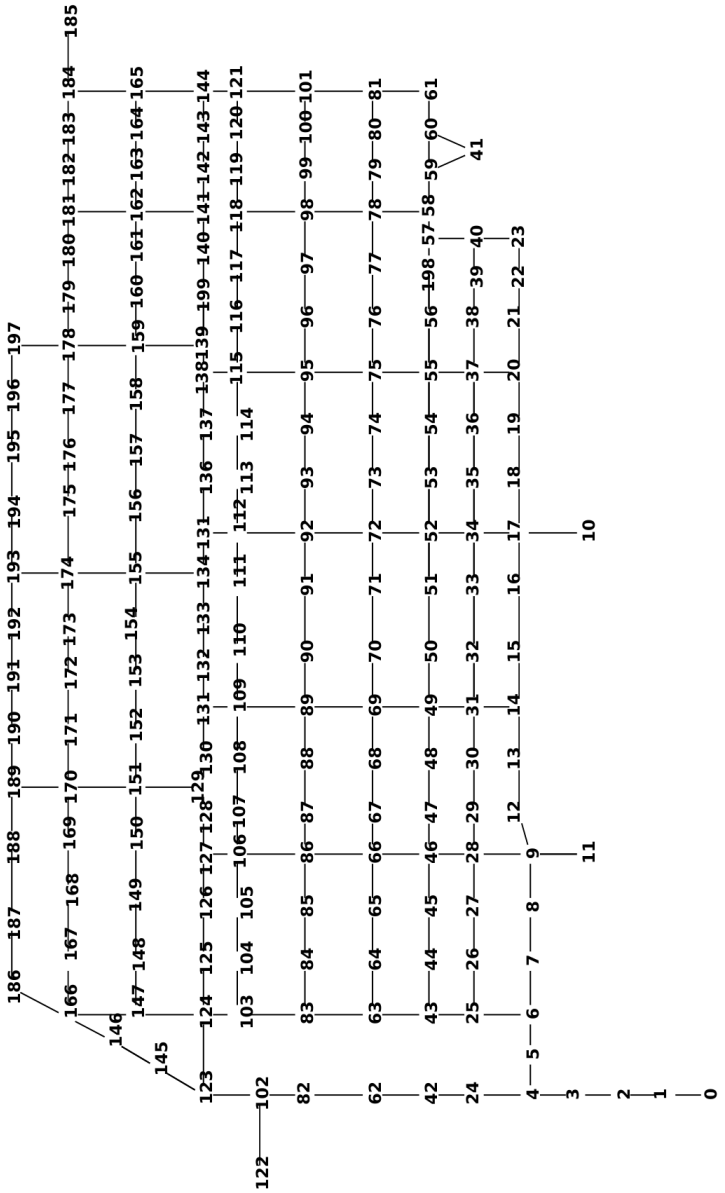


Figure C.1: Map with node numbers.

Appendix D

Tables for heatmaps in Figure 4.2

Table D.1: Average exposure times per node across 1000 simulations

Node	Average total or cumulative exposure times
31	98.710 563 07
57	68.898 977 32
58	62.870 355 15
40	58.307 288 75
38	57.216 801 79
39	56.189 077 95
28	46.923 802 27
123	41.916 333 78
102	39.582 540 55
4	36.467 919 75
124	32.189 111 43
49	31.662 508 19
17	26.359 573 23
69	26.272 225 45
109	24.803 948 72
89	24.738 022 28
6	24.425 074 03
3	24.287 680 89
59	23.969 923 95
115	23.265 531 25
2	22.184 696 52

46	22.161 748 51
5	21.980 843 6
184	21.719 253 54
112	21.079 635
66	20.352 337 88
30	20.083 002 91
1	19.721 180 22
78	19.606 740 3
29	19.457 881 01
34	19.384 926 63
106	18.992 456 1
95	18.922 636 58
86	18.854 802 89
52	18.512 706 14
11	18.420 330 65
92	18.409 547 99
139	18.331 633 97
138	17.966 108 98
10	17.551 906 36
122	17.536 712 53
0	17.514 683 29
41	17.455 288 74
185	17.416 291 43
72	17.385 250 83
155	17.180 138 79
151	16.127 834 46
75	15.362 090 85
144	14.441 541 18
55	13.750 371 45

14	13.295 824 49
165	13.230 941 78
121	12.749 165 76
98	11.321 703 3
25	10.222 838 05
135	9.970 355 583
174	9.780 078 755
103	9.676 720 817
134	9.196 293 813
9	9.127 932 781
159	8.659 556 451
129	8.575 106 478
131	8.461 159 399
127	8.282 436 769
13	8.252 025 773
12	7.707 489 936
101	7.307 446 974
178	7.138 609 323
170	7.124 518 76
8	7.096 423 889
189	6.685 167 397
24	6.563 012 715
7	6.508 754 465
56	6.342 209 751
82	6.222 358 533
42	6.150 901 684
62	6.119 682 026
140	5.859 566 669
141	5.604 223 909

27	5.333 110 002
83	5.092 987 417
181	4.980 809 458
118	4.706 053 135
26	4.648 512 475
188	4.601 083 203
147	4.392 200 143
108	4.316 237 406
154	4.195 741 99
107	4.194 353 56
81	4.166 972 082
187	4.090 976 754
153	4.081 384 981
152	4.054 148 848
146	4.024 027 595
162	3.929 224 986
43	3.896 264 089
63	3.896 029 124
182	3.877 368 949
186	3.812 214 807
130	3.756 420 098
110	3.749 400 599
111	3.748 057 228
183	3.745 996 873
193	3.694 779 346
150	3.365 008 715
113	3.337 019 877
48	3.313 796 164
126	3.264 511 826

179	3.218 921 024
156	3.140 324 691
47	3.121 067 612
15	2.942 788 591
114	2.910 194 371
125	2.863 625 831
149	2.856 129 367
157	2.844 407 133
175	2.814 982 651
88	2.758 543 364
68	2.648 073 989
180	2.635 483 954
87	2.585 833 622
158	2.560 731 346
16	2.557 327 927
148	2.555 504 527
176	2.547 492 761
105	2.513 021 104
60	2.500 526 602
67	2.423 107 942
128	2.418 236 462
192	2.315 936 735
190	2.237 596 832
177	2.227 946 622
119	2.200 089 496
191	2.141 786 433
104	2.140 079 72
37	2.129 900 353
61	2.115 501 431

90	2.079 534 499
91	2.060 208 957
136	1.968 694 796
173	1.906 866 405
51	1.888 569 553
93	1.876 300 982
171	1.841 544 486
172	1.830 818 462
32	1.772 859 141
120	1.722 542 521
99	1.694 721 67
137	1.683 393 816
94	1.665 105 937
50	1.604 668 173
116	1.588 738 563
132	1.553 358 369
33	1.453 065 182
71	1.450 332 113
133	1.439 733 959
117	1.429 705 115
96	1.428 887 837
70	1.425 725 875
79	1.402 042 056
85	1.390 980 792
45	1.368 380 465
97	1.347 071 025
76	1.288 715 493
100	1.247 915 86
73	1.229 151 945

65	1.222 585 646
142	1.206 997 884
53	1.191 419 855
143	1.135 170 454
74	1.133 905 33
77	1.116 357 498
84	1.074 585 607
44	1.072 477 96
80	1.057 066 612
194	1.034 646 278
163	1.030 294 439
160	1.019 546 763
54	1.007 827 838
18	0.986 414 291
164	0.950 572 927
20	0.946 811 642
64	0.893 092 069
195	0.875 320 353
19	0.827 062 453
35	0.790 543 896
197	0.736 458 358
161	0.704 454 596
196	0.676 226 414
36	0.637 435 157
169	0.594 475 422
23	0.534 452 66
168	0.405 964 42
166	0.337 646 559
167	0.280 820 114

22	0.246 638 022
21	0.144 423 866
198	0.087 078 484
145	0.082 853 18
199	0.082 455 721

Table D.2: Average number of unique buyer traversals per node across 1000 simulations

Node	**Average number of unique buyers**
31	1130.289
57	1001.52
58	982.166
38	913.073
40	912.974
39	907.285
28	903.105
102	856.899
123	844.513
4	787.723
124	730.332
17	715.602
49	693.446
59	684.735
3	683.02
184	673.107
2	667.081
1	650.596
11	646.592
69	641.898
6	639.067

122	635.987
10	634.477
0	634.391
185	633.569
41	633.061
89	618.861
46	618.611
5	616.363
109	608.083
34	600.35
30	597.345
29	593.733
66	586.871
115	575.884
52	565.572
112	564.995
106	563.216
86	559.718
78	559.107
92	543.677
95	537.335
72	534.992
144	510.39
138	509.893
165	509.1
75	494.496
139	494.173
155	491.186
121	479.622

55	478.518
151	473.201
14	454.91
25	437.056
98	426.034
103	419.868
135	395.289
9	391.391
127	379.16
174	373.036
13	370.458
134	369.625
101	369.151
12	365.263
129	360.01
131	359.294
8	353.403
159	350.812
24	347.71
7	343.529
56	341.96
42	341.777
82	341.012
62	340.093
178	332.563
189	312.293
27	311.284
140	307.901
83	307.232

170	306.432
141	304.96
181	302.258
26	300.801
108	281.935
81	279.198
182	277.982
107	277.332
188	275.495
183	273.199
118	273.161
43	269.98
147	269.554
63	267.424
154	267.42
152	266.618
187	265.856
146	265.279
153	264.86
186	260.467
110	259.966
111	258.175
162	255.217
48	247.925
126	244.389
130	244.211
47	243.546
179	237.197
113	237.108

125	233.616
150	232.258
193	231.289
114	230.391
15	226.837
88	225.482
156	225.269
180	223.262
149	221.473
87	220.725
68	219.284
157	217.951
16	217.475
175	216.393
67	214.43
148	213.235
105	212.799
158	211.673
128	210.746
176	208.859
60	207.252
177	202.353
104	202.278
192	194.954
119	194.078
90	193.876
190	193.722
37	193.631
61	192.97

91	192.132
191	191.969
93	181.712
136	179.925
120	178.378
173	177.817
171	176.303
94	174.974
172	174.848
137	173.156
32	170.587
51	166.711
50	165.825
99	163.09
116	160.868
33	160.795
70	158.533
132	158.168
96	157.415
71	156.445
117	156.032
133	156.009
97	152.643
85	150.42
79	149.554
45	149.019
100	147.579
76	145.704
73	143.389

77	141.151
142	139.506
84	138.825
44	138.147
65	138.033
53	137.519
74	136.243
143	136.187
80	133.636
54	130.591
163	130.114
164	127.334
64	126.791
20	124.584
18	121.195
160	119.774
194	119.195
19	113.328
195	111.387
35	108.515
197	107.55
196	104.857
161	104.76
36	101.603
169	81.067
168	69.374
23	68.116
166	64.461
167	60.315

22	51.215
21	41.966
145	23.569
199	23.51
198	23.327

Appendix E

Source Code

```
pos = {
    0: (12, -4), 1: (12, 0), 2: (12, 4), 3: (12, 8), 4: (12, 12), 5: (15, 12), 6: (18, 12), 7: (22, 12),
    8: (26, 12), 9: (30, 12), 10: (54, 7), 11: (30, 7), 12: (33, 13), 13: (37, 13), 14: (41, 13),
    15: (45, 13), 16: (50, 13), 17: (54, 13), 18: (58, 13), 19: (62, 13), 20: (66, 13), 21: (70, 13),
    22: (73, 13), 23: (76, 13), 24: (12, 17), 25: (18, 17), 26: (22, 17), 27: (26, 17), 28: (30, 17),
    29: (33, 17), 30: (37, 17), 31: (41, 17), 32: (45, 17), 33: (50, 17), 34: (54, 17), 35: (58, 17),
    36: (62, 17), 37: (66, 17), 38: (70, 17), 39: (73, 17), 40: (76, 17), 41: (82.5, 17), 42: (12, 21),
    43: (18, 21), 44: (22, 21), 45: (26, 21), 46: (30, 21), 47: (33, 21), 48: (37, 21), 49: (41, 21),
    50: (45, 21), 51: (50, 21), 52: (54, 21), 53: (58, 21), 54: (62, 21), 55: (66, 21), 56: (70, 21),
    57: (76, 21), 58: (78, 21), 59: (81, 21), 60: (84, 21), 61: (87, 21), 62: (12, 26), 63: (18, 26),
    64: (22, 26), 65: (26, 26), 66: (30, 26), 67: (33, 26), 68: (37, 26), 69: (41, 26), 70: (45, 26),
    71: (50, 26), 72: (54, 26), 73: (58, 26), 74: (62, 26), 75: (66, 26), 76: (70, 26), 77: (74, 26),
    78: (78, 26), 79: (81, 26), 80: (84, 26), 81: (87, 26), 82: (12, 32), 83: (18, 32), 84: (22, 32),
    85: (26, 32), 86: (30, 32), 87: (33, 32), 88: (37, 32), 89: (41, 32), 90: (45, 32), 91: (50, 32),
    92: (54, 32), 93: (58, 32), 94: (62, 32), 95: (66, 32), 96: (70, 32), 97: (74, 32), 98: (78, 32),
    99: (81, 32), 100: (84, 32), 101: (87, 32), 102: (12, 36), 103: (18, 38), 104: (22, 38),
    105: (26, 38), 106: (30, 38), 107: (33, 38), 108: (37, 38), 109: (41, 38), 110: (45, 38),
    111: (50, 38), 112: (54, 38), 113: (58, 38), 114: (62, 38), 115: (66, 38), 116: (70, 38),
    117: (74, 38), 118: (78, 38), 119: (81, 38), 120: (84, 38), 121: (87, 38), 122: (6, 36),
    123: (12, 41), 124: (18, 41), 125: (22, 41), 126: (26, 41),
    127: (30, 41), 128: (33, 41), 129: (35, 41), 130: (37, 41), 131: (41, 41), 132: (44, 41),
    133: (48, 41), 134: (51, 41), 135: (54, 41), 136: (58, 41), 137: (62, 41), 138: (66, 41),
    139: (68, 41), 140: (75, 41), 141: (78, 41), 142: (81, 41), 143: (84, 41), 144: (87, 41),
    145: (14, 45), 146: (16, 49), 147: (18, 47), 148: (22, 47), 149: (27, 47), 150: (31, 47),
    151: (35, 47), 152: (39, 47), 153: (43, 47), 154: (47, 47), 155: (51, 47), 156: (56, 47),
    157: (60, 47), 158: (64, 47), 159: (68, 47), 160: (72, 47), 161: (75, 47), 162: (78, 47),
    163: (81, 47), 164: (84, 47), 165: (87, 47), 166: (18, 53), 167: (22, 53), 168: (27, 53),
    169: (31, 53), 170: (35, 53), 171: (39, 53), 172: (43, 53), 173: (47, 53), 174: (51, 53),
    175: (56, 53), 176: (60, 53), 177: (64, 53), 178: (68, 53), 179: (72, 53), 180: (75, 53),
    181: (78, 53), 182: (81, 53), 183: (84, 53), 184: (87, 53), 185: (92, 53), 186: (20, 58),
    187: (25, 58), 188: (31, 58), 189: (35, 58), 190: (39, 58), 191: (43, 58), 192: (47, 58),
    193: (51, 58), 194: (56, 58), 195: (60, 58), 196: (64, 58),
    197: (68, 58), 198: (74, 21), 199: (72, 41)
}



# FOR ORIGINAL LAYOUT
```

```
edges = [
    (0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 12), (12, 13),
    (13, 14), (14, 15), (15, 16), (16, 17), (17, 18), (18, 19), (19, 20),
    (20, 21), (21, 22), (22, 23), (25, 26), (26, 27), (27, 28), (28, 29), (29, 30), (30, 31),
    (31, 32), (32, 33), (33, 34), (34, 35), (35, 36), (36, 37), (31, 38), (38, 39), (39, 40),
    (43, 44), (44, 45), (45, 46), (46, 47), (47, 48), (48, 49), (49, 50), (50, 51),
    (51, 52), (52, 53), (53, 54), (54, 55), (55, 56), (56, 57), (57, 58), (58, 59), (59, 60), (60, 61),
    (63, 64), (64, 65), (65, 66), (66, 67), (67, 68), (68, 69), (69, 70), (70, 71),
    (71, 72), (72, 73), (73, 74), (74, 75), (75, 76), (76, 77), (77, 78), (78, 79), (79, 80), (80, 81),
    (83, 84), (85, 86), (86, 87), (87, 88), (88, 89), (89, 90), (90, 91), (91, 92),
    (92, 93), (93, 94), (94, 95), (95, 96), (96, 97), (97, 98), (98, 99), (99, 100), (100, 101),
    (103, 104), (104, 105), (105, 106), (106, 107), (107, 108), (108, 109), (109, 110), (110, 111),
    (111, 112), (112, 113), (113, 114), (114, 115), (115, 116), (116, 117), (117, 118), (118, 119),
    (119, 120), (120, 121), (123, 124), (124, 125), (125, 126), (126, 127), (127, 128), (128, 129),
    (129, 130), (130, 131), (131, 132), (132, 133), (133, 134), (134, 135), (135, 136), (136, 137),
    (137, 138), (138, 139), (139, 140), (140, 141), (141, 142), (142, 143), (143, 144), (166, 147),
    (148, 149), (149, 150), (150, 151), (151, 152), (152, 153), (153, 154), (154, 155), (155, 156),
    (156, 157), (157, 158), (158, 159), (159, 160), (160, 161), (161, 162), (162, 163), (163, 164),
    (164, 165), (166, 167), (167, 168), (168, 169), (169, 170), (170, 171), (171, 172), (172, 173),
    (173, 174), (174, 175), (175, 176), (176, 177), (177, 178), (178, 179), (179, 180), (180, 181),
    (181, 182), (182, 183), (183, 184), (184, 185), (186, 187), (187, 188), (188, 189), (189, 190),
    (190, 191), (191, 192), (192, 193), (193, 194), (194, 195), (195, 196), (196, 197), (4, 24),
    (24, 42), (42, 62), (62, 82), (82, 102), (102, 123), (123, 146), (146, 186), (6, 25), (25, 43),
    (43, 63), (63, 83), (83, 103), (103, 124), (124, 147),
    (9, 11), (11, 28), (28, 46), (46, 66), (66, 86), (86, 106), (106, 127),
    (14, 31), (31, 49), (49, 69), (69, 89), (89, 109), (109, 131),
    (10, 17), (17, 34), (34, 52),
    (52, 72), (72, 92), (92, 112), (112, 135),
    (20, 37), (37, 55), (55, 75), (75, 95), (95, 115), (115, 138),
    (58, 78), (78, 98), (98, 118), (141, 162), (162, 181),
    (61, 81), (81, 101), (101, 121), (121, 144), (144, 165), (165, 184),
    (139, 159), (159, 178), (178, 197),
    (134, 155), (155, 174), (174, 193),
    (129, 151), (151, 170), (170, 189),
    (41, 59), (41, 60), (102, 122),
    (23, 40), (40, 57),
    (123, 145), (145, 146), (84, 85), (148, 147), (9, 11), (139, 199), (51, 198)
]

# FOR ONE-WAY SETUP
# edges = [
#      (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 12), (12, 13), (13, 14), (14, 15), (15, 16),
(16, 17), (17, 18), (18, 19), (19, 20),
#     (20, 21), (21, 22), (22, 23), (26, 25), (27, 26), (28, 27), (29, 28), (30, 29), (31, 30),
# (32, 31), (33, 32), (34, 33), (35, 34), (36, 35), (37, 36), (38, 37),
# (38, 31), (39, 38), (40, 39), (148, 147), (56, 198), (139, 199), (122, 102),  (124, 123), (199, 140),
(185, 184),
```

```
#     (43, 44), (44, 45), (45, 46), (46, 47), (47, 48), (48, 49), (49, 50), (50, 51),
#     (51, 52), (52, 53), (53, 54), (54, 55), (55, 56), (57, 58), (58, 59), (59, 60), (60, 61),
#     (64, 63), (65, 64), (66, 65), (67, 66), (68, 67), (69, 68), (70, 69), (71, 70),


# (72, 71), (73, 72), (74, 73), (75, 74), (76, 75), (77, 76), (78, 77), (79, 78), (80, 79), (81, 80),
#     (83, 84), (85, 86), (86, 87), (87, 88), (88, 89), (89, 90), (90, 91), (91, 92),
#     (92, 93), (93, 94), (94, 95), (95, 96), (96, 97), (97, 98), (98, 99), (99, 100), (100, 101),
#     (104, 103), (105, 104), (106, 105), (107, 106), (108, 107), (109, 108), (110, 109), (111, 110),
(84, 85),  (102, 122),


# (112, 111), (113, 112), (114, 113), (115, 114), (116, 115), (117, 116), (118, 117), (119, 118),
# (120, 119), (121, 120), (123, 124), (124, 125), (125, 126), (126, 127), (127, 128), (128, 129),
#     (129, 130), (130, 131), (131, 132), (132, 133), (133, 134), (134, 135), (135, 136),
# (136, 137), (137, 138), (138, 139), (140, 141), (141, 142), (142, 143), (143, 144),


#     (166, 167), (167, 168), (168, 169), (169, 170), (170, 171), (171, 172), (172, 173), (173, 174),
# (174, 175), (175, 176), (176, 177), (177, 178), (178, 179), (179, 180), (180, 181), (181, 182),
# (182, 183), (183, 184), (184, 185), (187, 186), (188, 187), (189, 188), (190, 189), (191, 190),
# (192, 191), (193, 192), (194, 193), (195, 194), (196, 195), (197, 196), (198, 57),

# (149, 148), (150, 149), (151, 150), (152, 151), (153, 152), (154, 153), (155, 154),
# (156, 155), (157, 156), (158, 157), (159, 158), (160, 159), (161, 160), (162, 161), (163, 162),
# (164, 163), (165, 164),


#     # verticals
#     (4, 24), (24, 42), (42, 62), (62, 82), (82, 102), (102, 123), (146, 166), (166, 186),
#     (147, 166),  (6, 25), (25, 43), (43, 63), (63, 83), (83, 103), (103, 124), (124, 147),
#     (11, 9), (11, 28), (28, 46), (46, 66), (66, 86), (86, 106), (106, 127),
#     (14, 31), (31, 49), (49, 69), (69, 89), (89, 109), (109, 131),
#     (10, 17), (17, 34), (34, 52), (52, 72), (72, 92), (92, 112), (112, 135),

#     (20, 37), (37, 55), (55, 75), (75, 95), (95, 115), (115, 138),
#     (58, 78), (78, 98), (98, 118), (141, 162), (162, 181),
#     (61, 81), (81, 101), (101, 121), (121, 144), (144, 165), (165, 184),
#     (139, 159), (159, 178), (178, 197), (0, 1), (1, 2), (2, 3), (3, 4),


#     (134, 155), (155, 174), (174, 193),
#     (129, 151), (151, 170), (170, 189),
#     (41, 59), (41, 60),
#     (23, 40), (40, 57),
#     (123, 145), (145, 146),   (9, 11),
```

```
#     (24, 4), (42, 24), (62, 42), (82, 62), (102, 82), (123, 102),
#     (4, 3), (3, 2), (2, 1), (1, 0),
#     (166, 147), (147, 124), (124, 103), (103, 83), (83, 63), (63, 43), (43, 25), (25, 6),
#     (127, 106), (106, 86), (86, 66), (66, 46), (46, 28), (28, 9),


#     (189, 170), (170, 151), (151, 129),
#     (193, 174), (174, 155), (155, 134),
#     (131, 109), (109, 89), (89, 69), (69, 49), (49, 31), (31, 14), (11, 9),
#     (135, 112), (112, 92), (92, 72), (72, 52), (52, 34), (34, 17), (17, 10), (197, 178),
#(178, 159), (159, 139), (138, 115), (115, 95), (95, 75), (75, 55), (55, 37), (37, 20),

#     (181, 162), (162, 141), (118, 98), (98, 78), (78, 58), (57, 40), (40, 23), (59, 41), (60, 41),
# (184, 165), (165, 144), (144, 121), (121, 101), (101, 81), (81, 61), (9, 11),
#     (186, 166), (166, 146), (146, 145), (145, 123), (147, 166)

# ]


from covid19_supermarket_abm.utils.create_store_network import create_store_network
G = create_store_network(pos, edges)
# G = create_store_network(pos, edges, directed=True) # For directed network

import networkx as nx
import matplotlib.pyplot as plt

# pos = nx.get_node_attributes(G, 'pos')
node_color_map = {}
stalls = 0
for i in range(200):
    if i in [9, 57, 189, 170, 151, 129, 106, 86, 66, 46, 28, 127, 193, 174, 155, 134, 112, 92, 72, 52,
    34, 17, 197, 178, 159, 139, 115, 95, 75, 55, 37, 23, 49, 58, 78, 98, 118, 121, 101, 81, 61, 144,
    165, 184, 181, 162, 141, 131, 109, 89, 69, 31, 14, 20, 40, 135, 138, 124, 147]:
        node_color_map[i] = 'orange'
    elif i in [0, 122, 185, 41, 10, 11]:
        node_color_map[i] = '#5ACF59'
    else:
        node_color_map[i] = 'y'
        stalls += 1
nx.set_node_attributes(G, node_color_map, 'color')
node_colors = [node_color_map[node] for node in G.nodes()]
f,ax = plt.subplots(1,1, figsize=(11,8))
nx.draw_networkx(G, pos=pos, node_color=node_colors, node_size=170, font_size=6)

import matplotlib.pyplot as plt
```

```
import networkx as nx
from matplotlib.patches import Rectangle

f,ax = plt.subplots(1,1, figsize=(11,8))

nx.draw_networkx(G, pos=pos, node_color=node_colors,node_size=100, font_size=6, with_labels=False)
ax.add_patch(Rectangle((9.5,-4),1.2,37,linewidth=1,facecolor='lightgrey', zorder=-1))
ax.add_patch(Rectangle((13.5,-4),1.2,13,linewidth=1,facecolor='lightgrey', zorder=-1))
ax.add_patch(Rectangle((13.5,13),1.2,27,linewidth=1,facecolor='lightgrey', zorder=-1))
ax.add_patch(Rectangle((15.5,13),1.2,27,linewidth=1,facecolor='lightgrey', zorder=-1))
ax.add_patch(Rectangle((13.5,9.5),15,1.2,linewidth=1,facecolor='lightgrey', zorder=-1))
ax.add_patch(Rectangle((31.5,10.5),21,1.2,linewidth=1,facecolor='lightgrey', zorder=-1))
ax.add_patch(Rectangle((55,10.5),21,1.2,linewidth=1,facecolor='lightgrey', zorder=-1))
########################
ax.add_patch(Rectangle((19,13),10,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,14.5),10,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,18.3),10,1.2,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,22),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,23.5),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,27.6),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,29),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,33.6),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,35.2),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,39),10,1,linewidth=1,facecolor='lightgrey'))
########################
ax.add_patch(Rectangle((31,14.4),9.5,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((31,18.3),9.5,1.2,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((31,22),9.5,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((31,23.5),9.5,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((31,27.6),9.5,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((31,29),9.5,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((31,33.6),9.5,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((31,35.2),9.5,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((31,39),9.5,1,linewidth=1,facecolor='lightgrey'))
########################
ax.add_patch(Rectangle((42,14.4),11,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((42,18.3),11,1.2,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((42,22),11,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((42,23.5),11,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((42,27.6),11,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((42,29),11,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((42,33.6),11,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((42,35.2),11,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((42,39),11,1,linewidth=1,facecolor='lightgrey'))
########################
ax.add_patch(Rectangle((55,14.4),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((55,18.3),10,1.2,linewidth=1,facecolor='lightgrey'))
```

```
ax.add_patch(Rectangle((55,22),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((55,23.5),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((55,27.6),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((55,29),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((55,33.6),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((55,35.2),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((55,39),10,1,linewidth=1,facecolor='lightgrey'))
########################
ax.add_patch(Rectangle((67,14.4),8.5,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((67,18.3),8.5,1.2,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((67,22),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((67,23.5),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((67,27.6),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((67,29),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((67,33.6),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((67,35.2),10,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((67,39),10,1,linewidth=1,facecolor='lightgrey'))
########################
ax.add_patch(Rectangle((79,22),7,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((79,23.5),7,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((79,27.6),7,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((79,29),7,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((79,33.6),7,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((79,35.2),7,1.1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((79,39),7,1,linewidth=1,facecolor='lightgrey'))
########################
ax.add_patch(Rectangle((19,42.7),15,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,44.3),15,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,48.7),15,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19,50.3),15,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19.5,54.1),14.5,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19.5,55.7),14.5,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((19.5,59.1),14.5,1,linewidth=1,facecolor='lightgrey'))
########################
ax.add_patch(Rectangle((36,42.7),14.5,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((36,44.3),14.5,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((36,48.7),14.5,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((36,50.3),14.5,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((36,54.1),14.5,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((36,55.7),14.5,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((36,59.1),14.5,1,linewidth=1,facecolor='lightgrey'))
########################
ax.add_patch(Rectangle((52, 42.7),15,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((52,44.3),15,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((52,48.7),15,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((52,50.3),15,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((52,54.1),15,1,linewidth=1,facecolor='lightgrey'))
```

```
ax.add_patch(Rectangle((52,55.7),15,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((52,59.1),15,1,linewidth=1,facecolor='lightgrey'))
########################
ax.add_patch(Rectangle((69, 42.7),8,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((69,44.3),8,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((69,48.7),8,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((69,50.3),8,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((69,54.1),8,1,linewidth=1,facecolor='lightgrey'))
########################
ax.add_patch(Rectangle((79, 42.7),7,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((79,44.3),7,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((79,48.7),7,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((79,50.3),7,1,linewidth=1,facecolor='lightgrey'))
ax.add_patch(Rectangle((79,54.1),7,1,linewidth=1,facecolor='lightgrey'))
########################
ax.add_patch(Rectangle((10.5,42),18,1,angle=63,linewidth=1,facecolor='lightgrey'))
########################
# ax.add_patch(Rectangle((9.5,-5.5),5,3,linewidth=1,facecolor='y', zorder=-1))
# ax.add_patch(Rectangle((4,34.5),5,3,linewidth=1,facecolor='y', zorder=-1))
# ax.add_patch(Rectangle((27.5,5.5),5,3,linewidth=1,facecolor='y', zorder=-1))
# ax.add_patch(Rectangle((51.5,5.5),5,3,linewidth=1,facecolor='y', zorder=-1))
# ax.add_patch(Rectangle((80,15.5),5,3,linewidth=1,facecolor='y', zorder=-1))
# ax.add_patch(Rectangle((89.5,51.5),5,3,linewidth=1,facecolor='y', zorder=-1))

plt.box(False)
plt.savefig('fig3.pdf')
plt.show()

from covid19_supermarket_abm.path_generators import get_path_generator
import json

with open('10^6.json', 'r') as file:
    full_paths = json.load(file)

print(len(full_paths))
path_generator_function, path_generator_args = get_path_generator(path_generation='empirical',
full_paths=full_paths)

import json
import os
import pickle
import logging

import networkx as nx
import pandas as pd

from simulator import simulate_several_days
```

```
"""
Functions to run simulations and save results
"""

def run_one_simulation_and_record_stats(config_name, num_iterations, config_dir='.',
data_dir='.', results_dir='.'):
    """Make one simulation with no multipliers."""
    print('Running simulation with no modified parameters')
    config_filename = os.path.join(config_dir, f"{config_name}.json")
    config_original = {'arrival_rate': 2.88,  # 2.88 Poisson rate at which customers arrive
           'traversal_time': 0.49,  # mean wait time per node
           'num_hours_open': 12,  # store opening hours
           'infection_proportion': 0.128,  # proportion of customers that are infectious
          #  'max_customers_in_store': 40,
            'with_node_capacity': True,
            'node_capacity': 4,
         }
    config = config_original
    logging.info(f'Loaded config file: {config_filename}')

    # Do simulations
    df_cust, df_num_encounter_per_node_stats, df_exposure_time_per_node_stats = simulate_several_days
    (config,G,
    #   extra_outputs,
    path_generator_function,path_generator_args,num_iterations)
    this_results = df_cust, df_num_encounter_per_node_stats, df_exposure_time_per_node_stats

    results_folder = os.path.join(results_dir, 'results')
    if not os.path.isdir(results_folder):
        print(f'Created {results_folder}')
        os.mkdir(results_folder)

    # Save results
    filename1 = os.path.join(results_folder, f'{config_name}_{num_iterations}_1.parquet')
    df_cust.to_parquet(filename1)
    df_num_encounter_per_node_stats.columns = df_num_encounter_per_node_stats.columns.astype(str)

    filename2 = os.path.join(results_folder, f'{config_name}_{num_iterations}_2.parquet')
    df_num_encounter_per_node_stats.to_parquet(filename2)
    df_exposure_time_per_node_stats.columns = df_exposure_time_per_node_stats.columns.astype(str)

    filename3 = os.path.join(results_folder, f'{config_name}_{num_iterations}_3.parquet')
    df_exposure_time_per_node_stats.to_parquet(filename3)
    print(f'Results saved in {filename1}, {filename2}, {filename3}.')
    return this_results
```

```
results = run_one_simulation_and_record_stats('config', 1000) #1000 simulations
```

```
% SIMULATOR.PY
import multiprocessing
from itertools import repeat

import networkx as nx
import numpy as np
import pandas as pd
import simpy
import logging
from tqdm import tqdm

import core as core
from covid19_supermarket_abm.utils import istarmap  # enable progress bar with multiprocessing


def simulate_one_day(config: dict, G: nx.Graph, path_generator_function, path_generator_args: list):
    # Get parameters
    num_hours_open = config['num_hours_open']
    logging_enabled = True
    raise_test_error = config.get('raise_test_error', False)  # for debugging purposes
    with_node_capacity = config.get('with_node_capacity', False)
    max_customers_in_store_per_sqm = config.get('max_customers_in_store_per_sqm', None)
    floorarea = config.get('floorarea', None)
    if max_customers_in_store_per_sqm is None:
        max_customers_in_store = config.get('max_customers_in_store', None)
    else:
        if floorarea is not None:
            max_customers_in_store = int(max_customers_in_store_per_sqm * floorarea)
        else:
            raise ValueError('If you set the parameter "max_customers_in_store_per_sqm", '
                             'you need to specify the floor area via the "floorarea" parameter
                             in the config.')

    # Set up environment and run
    env = simpy.Environment()
    store = core.Store(env, G, max_customers_in_store=max_customers_in_store,
    logging_enabled=logging_enabled)
    if with_node_capacity:
        node_capacity = config.get('node_capacity', 2)
        store.enable_node_capacity(node_capacity)
    path_generator = path_generator_function(*path_generator_args)
```

```
# env.process(_)
env.process(core._customer_arrivals(env, store, path_generator, config))
env.process(core._stats_recorder(store))
env.run(until=num_hours_open * 60 * 10)

# Record stats
core._sanity_checks(store, raise_test_error=raise_test_error)
num_cust = len(store.customers)
num_S = len(store.number_encounters_with_infected)
shopping_times = list(store.shopping_times.values())
waiting_times = np.array(list(store.waiting_times.values()))
b = np.array(waiting_times) > 0
num_waiting_people = sum(b)
if num_waiting_people > 0:
    mean_waiting_time = np.mean(waiting_times[b])
else:
    mean_waiting_time = 0

num_contacts_per_cust = [contacts for contacts in store.number_encounters_with_infected.values()
if contacts != 0]
df_num_encounters_per_node = pd.DataFrame(store.number_encounters_per_node, index=[0])
df_num_encounters_per_node = df_num_encounters_per_node[range(len(G))]
df_exposure_time_per_node = pd.DataFrame(store.time_with_infected_per_node, index=[0])
df_exposure_time_per_node = df_exposure_time_per_node[range(len(G))]
exposure_times = [val for val in list(store.time_with_infected_per_customer.values()) if val > 0]
results = {'num_cust': num_cust,
           'num_S': num_S,
           'num_I': num_cust - num_S,
           'total_exposure_time': sum(store.time_with_infected_per_customer.values()),
           'num_contacts_per_cust': num_contacts_per_cust,
           'num_cust_w_contact': len(num_contacts_per_cust),
           'mean_num_cust_in_store': np.mean(list(store.stats['num_customers_in_store'].values())),
           'max_num_cust_in_store': max(list(store.stats['num_customers_in_store'].values())),
           'num_contacts': sum(num_contacts_per_cust),
           'shopping_times': shopping_times,
           'mean_shopping_time': np.mean(shopping_times),
           'num_waiting_people': num_waiting_people,
           'mean_waiting_time': mean_waiting_time,
           'store_open_length': max(list(store.stats['num_customers_in_store'].keys())),
           'df_num_encounters_per_node': df_num_encounters_per_node,
           'df_exposure_time_per_node': df_exposure_time_per_node,
           'total_time_crowded': store.total_time_crowded,
           'exposure_times': exposure_times,
           }

# logs_data = {'log': store.logs}
# df = pd.DataFrame(logs_data)
```

```
    # df.to_parquet("logs_config_new_1000.txt", index=True)

    with open('logs_final_paths.txt', 'a') as f:
        for log in store.logs:
            f.write(log + '\n')
    f.close()

    if floorarea is not None:
        results['mean_num_cust_in_store_per_sqm'] = results['mean_num_cust_in_store'] / floorarea
        results['max_num_cust_in_store_per_sqm'] = results['max_num_cust_in_store'] / floorarea
    # results['logs'] = store.logs
    return results


def simulate_several_days(config: dict,
                          G: nx.Graph,
                        #   extra_outputs,
                          path_generator_function,
                          path_generator_args: list,
                          num_iterations: int = 1000,
                          use_parallel: bool = False):
    """Run several simulations and return selected number of stats from these simulations"""

    # Run simulations
    if use_parallel:
        args = [config, G, path_generator_function, path_generator_args]
        repeated_args = zip(*[repeat(item, num_iterations) for item in args])
        num_cores = multiprocessing.cpu_count()
        with multiprocessing.Pool(num_cores) as p:
            with tqdm(total=num_iterations) as pbar:
                results = []
                for i, results_dict in enumerate(p.istarmap(simulate_one_day, repeated_args)):
                    results.append(results_dict)
                    pbar.update()
    else:
        results = []
        for _ in tqdm(range(num_iterations)):
            results_dict = simulate_one_day(config, G, path_generator_function, path_generator_args)
            results.append(results_dict)

    # Initialize containers to save any scalar statistics
    df_num_encounters_per_node_list = []
    df_exposure_time_per_node_list = []
    stats_dict = {}
    cols_to_record = [key for key, val in results_dict.items()
                      if isinstance(val, (int, np.integer)) or isinstance(val, (float, float))]
    cols_to_record += ['exposure_times']
```

```
    cols_not_recording = [key for key in results_dict.keys() if key not in cols_to_record]
    logging.info(f'Recording the scalar stats for {cols_to_record}.')
    logging.info(f'We are not recording {cols_not_recording}.')
    for stat in cols_to_record:
        stats_dict[stat] = []

    # Record encounter stats as well
    logging.info('Recording stats for df_num_encounters_per_node, df_exposure_time_per_node')
    for i in range(num_iterations):
        results_dict = results[i]
        df_num_encounters_per_node_list.append(results_dict['df_num_encounters_per_node'])
        df_exposure_time_per_node_list.append(results_dict['df_exposure_time_per_node'])
        for col in cols_to_record:
            stats_dict[col].append(results_dict[col])
    df_stats = pd.DataFrame(stats_dict)
    df_num_encounter_per_node_stats = pd.concat(df_num_encounters_per_node_list).reset_index(drop=True)
    df_encounter_time_per_node_stats = pd.concat(df_exposure_time_per_node_list).reset_index(drop=True)
    return df_stats, df_num_encounter_per_node_stats, df_encounter_time_per_node_stats


% CORE.PY
import datetime
import logging
import random
import uuid
from typing import List, Optional

import networkx as nx
import numpy as np
import simpy


class Store(object):
    """Store object that captures the state of the store"""

    def __init__(self, env: simpy.Environment, G: nx.Graph, max_customers_in_store: Optional[int] = None,
                 logging_enabled: bool = True,
                 logger: Optional[logging._loggerClass] = None):
        """

        :param env: Simpy environment on which the simulation runs
        :param G: Store graph
        :param logging_enabled: Toggle to True to log all simulation outputs
        :param max_customers_in_store: Maximum number of customers in the store
        """
        self.G = G.copy()
        self.customers_at_nodes = {node: [] for node in self.G}
```

```
        self.infected_customers_at_nodes = {node: [] for node in self.G}
        self.customers = []
        self.infected_customers = []
        self.env = env
        self.number_encounters_with_infected = {}
        self.number_encounters_per_node = {node: 0 for node in self.G}
        self.arrival_times = {}
        self.exit_times = {}
        self.shopping_times = {}
        self.waiting_times = {}
        self.customers_next_zone = {}  # maps customer to the next zone that it wants to go
        self.is_open = True
        self.is_closed_event = self.env.event()
        self.time_with_infected_per_customer = {}
        self.time_with_infected_per_node = {node: 0 for node in self.G}
        self.node_arrival_time_stamp = {}
        self.num_customers_waiting_outside = 0
        self.total_time_crowded = 0
        self.crowded_thres = 4
        self.node_is_crowded_since = {node: None for node in self.G}  # is None if not crowded,
        else it's the start time
        self.logs = []
        self.logging_enabled = True
        self.logger = logger

        # Parameters
        self.node_capacity = np.inf
        self.with_node_capacity = False
        if max_customers_in_store is None:
            self.max_customers_in_store = np.inf
        else:
            self.max_customers_in_store = int(max_customers_in_store)
        # if logger is None:
        #     self.logging_enabled = False
        # else:
        #     self.logging_enabled = True
        self.counter = simpy.Resource(self.env, capacity=self.max_customers_in_store)

        # Stats recording
        self.stats = {}

    def open_store(self):
        assert len(self.customers) == 0, "Customers are already in the store before the store is open"
        self.is_open = True

    def close_store(self):
        # self.log(f'Store is closing. There are {self.number_customers_in_store()} left in the store. ' +
```

```
        #          f'({self.num_customers_waiting_outside} are waiting outside')
        self.log(f'Market closed.')
        self.is_open = False
        self.is_closed_event.succeed()

    def enable_node_capacity(self, node_capacity: int = 2):
        self.with_node_capacity = True
        self.node_capacity = node_capacity

    def number_customers_in_store(self):
        return sum([len(cus) for cus in list(self.customers_at_nodes.values())])

    def move_customer(self, customer_id: int, infected: bool, start: int, end: int) -> bool:
        if self.check_valid_move(start, end):
            if start == end:  # start == end
                self._customer_wait(customer_id, start, infected)
                # self.log(f'Customer {customer_id} stays at present location to buy something.')
                has_moved = True
            elif self.with_node_capacity and len(self.customers_at_nodes[end]) >= self.node_capacity \
                    and start not in [self.customers_next_zone[cust] for cust in
                    self.customers_at_nodes[end]]:
                # Wait if next node is occupied and doesn't work.
                # self.log(f'Customer {customer_id} is waiting at {start}, ' +
                #          f'since the next node {end} is full. [{self.customers_at_nodes[end]}]')
                self._customer_wait(customer_id, start, infected)
                has_moved = False
            else:
                # self.log(f'Customer {customer_id} is moving from {start} to {end}.')
                self._customer_departure(customer_id, start, infected)
                self._customer_arrival(customer_id, end, infected)
                has_moved = True
        else:
            raise ValueError(f'{start} -> {end} is not a valid transition in the graph!')
        return has_moved

    def check_valid_move(self, start: int, end: int):
        return self.G.has_edge(start, end) or start == end

    def add_customer(self, customer_id: int, start_node: int, infected: bool, wait: float):
        # self.log(f'New customer {customer_id} arrives at the store. ' +
        #          f'({infected * "infected"}{(not infected) * "susceptible"})')
        self.arrival_times[customer_id] = self.env.now
        self.waiting_times[customer_id] = wait
        self.customers.append(customer_id)
        if not infected:
            # Increase counter
            self.number_encounters_with_infected[customer_id] = 0
```

```
            self.time_with_infected_per_customer[customer_id] = 0
        else:
            self.infected_customers.append(customer_id)
        self._customer_arrival(customer_id, start_node, infected)

    def infect_other_customers_at_node(self, customer_id: int, node: int):
        other_suspectible_customers = [other_customer for other_customer in
        self.customers_at_nodes[node] if
                                       other_customer not in self.infected_customers_at_nodes[node]]
        # if len(other_suspectible_customers) > 0:
            # self.log(
            #     f'Infected customer {customer_id} arrived in {node} and' +
            #     f' met {len(other_suspectible_customers)} customers')
        for other_customer in other_suspectible_customers:
            self.number_encounters_with_infected[other_customer] += 1
            self.number_encounters_per_node[node] += 1

    def get_infected_by_other_customers_at_node(self, customer_id: int, node: int):
        num_infected_here = len(self.infected_customers_at_nodes[node])

        # Track number of infected customers
        if num_infected_here > 0:
            # self.log(
            #     f'Customer {customer_id} is in at zone {node} with {num_infected_here}
            infected people.' +
            #     f' ({self.infected_customers_at_nodes[node]})')
            self.number_encounters_with_infected[customer_id] += num_infected_here
            self.number_encounters_per_node[node] += num_infected_here

    def _customer_arrival(self, customer_id: int, node: int, infected: bool):
        """Process a customer arriving at a node."""
        self.customers_at_nodes[node].append(customer_id)
        self.node_arrival_time_stamp[customer_id] = self.env.now
        if infected:
            self.infected_customers_at_nodes[node].append(customer_id)
            self.infect_other_customers_at_node(customer_id, node)
        else:
            self.get_infected_by_other_customers_at_node(customer_id, node)
        num_cust_at_node = len(self.customers_at_nodes[node])
        if num_cust_at_node >= self.crowded_thres and self.node_is_crowded_since[node] is None:
            # self.log(f'Node {node} has become crowded with {num_cust_at_node} customers here.')
            self.node_is_crowded_since[node] = self.env.now

    def _customer_wait(self, customer_id: int, node: int, infected: bool):
        if infected:
            self.infect_other_customers_at_node(customer_id, node)
        else:
```

```
            self.get_infected_by_other_customers_at_node(customer_id, node)

    def _customer_departure(self, customer_id: int, node: int, infected: bool):
        """Process a customer departing from a node."""
        self.customers_at_nodes[node].remove(customer_id)
        if infected:
            self.infected_customers_at_nodes[node].remove(customer_id)
            s_customers = self.get_susceptible_customers_at_node(node)
            for s_cust in s_customers:
                dt_with_infected = self.env.now - max(self.node_arrival_time_stamp[s_cust],
                                                      self.node_arrival_time_stamp[customer_id])
                self.time_with_infected_per_customer[s_cust] += dt_with_infected
                self.time_with_infected_per_node[node] += dt_with_infected
        else:
            i_customers = self.infected_customers_at_nodes[node]
            for i_cust in i_customers:
                dt_with_infected = self.env.now - max(self.node_arrival_time_stamp[i_cust],
                                                      self.node_arrival_time_stamp[customer_id])
                self.time_with_infected_per_customer[customer_id] += dt_with_infected
                self.time_with_infected_per_node[node] += dt_with_infected

        num_cust_at_node = len(self.customers_at_nodes[node])
        if self.node_is_crowded_since[node] is not None and num_cust_at_node < self.crowded_thres:
            # Node is no longer crowded
            total_time_crowded_at_node = self.env.now - self.node_is_crowded_since[node]
            self.total_time_crowded += total_time_crowded_at_node
            # self.log(
            #     f'Node {node} is no longer crowded ({num_cust_at_node} customers here. ' +
            #     f'Total time crowded: {total_time_crowded_at_node:.2f}')
            self.node_is_crowded_since[node] = None

    def get_susceptible_customers_at_node(self, node):
        return [c for c in self.customers_at_nodes[node] if c not
        in self.infected_customers_at_nodes[node]]

    def remove_customer(self, customer_id: int, last_position: int, infected: bool):
        """Remove customer at exit."""
        self._customer_departure(customer_id, last_position, infected)
        self.exit_times[customer_id] = self.env.now
        self.node_arrival_time_stamp[customer_id] = self.env.now
        self.shopping_times[customer_id] = self.exit_times[customer_id] -
        self.arrival_times[customer_id]
        # self.log(f'Customer {customer_id} left the store.')

    def now(self):
        return f'{self.env.now:.4f}'
```

```
    def log(self, string: str):
        # if self.logging_enabled:
        #     self.logs.append(f'[Time: {self.now()}] ' + string)
        # if self.logger is not None:
        #     self.logger.debug(f'[Time: {self.now()}] ' + string)
        if self.logging_enabled:
            self.logs.append(string)
        if self.logger is not None:
            self.logger.debug(string)



def customer(env: simpy.Environment, customer_id: int, infected: bool, store: Store, path: List[int],
             traversal_time: float, thres: int = 50):
    """
    Simpy process simulating a single customer

    :param env: Simpy environment on which the simulation runs
    :param customer_id: ID of customer
    :param infected: True if infected
    :param store: Store object
    :param path: Assigned customer shopping path
    :param traversal_time: Mean time before moving to the next node in path (also called waiting time)
    :param thres: Threshold length of queue outside. If queue exceeds threshold, customer does not enter
    the queue and leaves.
    """

    arrive = env.now

    if store.num_customers_waiting_outside > thres:
        #store.log(f'Customer {customer_id} does not queue up, since
        we have over {thres} customers waiting outside ' +
                #   f'({store.num_customers_waiting_outside})')
        return
    else:
        store.num_customers_waiting_outside += 1

    with store.counter.request() as my_turn_to_enter:
        result = yield my_turn_to_enter | store.is_closed_event
        store.num_customers_waiting_outside -= 1
        wait = env.now - arrive

        if my_turn_to_enter not in result:
            #store.log(f'Customer {customer_id} leaves the queue after waiting {wait:.2f} min,
            as shop is closed')
            return
```

```
        if my_turn_to_enter in result:
            #store.log(f'Customer {customer_id} enters the shop after waiting {wait :.2f} min
            with shopping path {path}.')
            store.log(f'{path}')
            start_node = path[0]
            store.add_customer(customer_id, start_node, infected, wait)
            for start, end in zip(path[:-1], path[1:]):
                store.customers_next_zone[customer_id] = end
                has_moved = False
                while not has_moved:  # If it hasn't moved, wait a bit
                    yield env.timeout(random.expovariate(1 / traversal_time))
                    has_moved = store.move_customer(customer_id, infected, start, end)
            yield env.timeout(random.expovariate(1 / traversal_time))  # wait before leaving the store
            store.remove_customer(customer_id, path[-1], infected)


def _stats_recorder(store: Store):
    store.stats['num_customers_in_store'] = {}
    env = store.env
    while store.is_open or store.number_customers_in_store() > 0:
        store.stats['num_customers_in_store'][env.now] = store.number_customers_in_store()
        yield env.timeout(10)

def _customer_arrivals(env: simpy.Environment, store: Store, path_generator, config: dict):
    """Process that creates all customers."""
    arrival_rate = config['arrival_rate']
    num_hours_open = config['num_hours_open']
    infection_proportion = config['infection_proportion']
    traversal_time = config['traversal_time']
    customer_id = 0
    store.open_store()
    yield env.timeout(random.expovariate(arrival_rate))
    while env.now < num_hours_open * 60:
        infected = np.random.rand() < infection_proportion
        path = path_generator.__next__()
        env.process(customer(env, customer_id, infected, store, path, traversal_time))
        customer_id += 1
        yield env.timeout(random.expovariate(arrival_rate))
    store.close_store()


def _sanity_checks(store: Store,
                   # logger: Optional[logging._loggerClass] = None, log_capture_string=None,
                   raise_test_error=False):
    infectious_contacts_list = [i for i in store.number_encounters_with_infected.values() if i != 0]
    num_susceptible = len(store.number_encounters_with_infected)
    num_infected = len(store.infected_customers)
```

```
    num_cust = len(store.customers)

    try:
        assert sum(infectious_contacts_list) == sum(store.number_encounters_per_node.values()), \
            "Number of infectious contacts doesn't add up"
        assert num_infected + num_susceptible == num_cust, \
            "Number of infected and susceptible customers doesn't add up to total number of customers"

        customers_at_nodes = [len(val) for val in store.infected_customers_at_nodes.values()]
        assert max(customers_at_nodes) == 0, \
            f"{sum(customers_at_nodes)} customers have not left the store.
            {store.infected_customers_at_nodes}"
        assert max([len(val) for val in store.customers_at_nodes.values()]) == 0, \
        f"{sum(customers_at_nodes)} customers have not left the store. {store.customers_at_nodes}"
        assert set(store.waiting_times.keys()) == set(store.customers), \
            'Some customers are not recorded in waiting times (or vice versa)'
        assert all([val >= 0 for val in store.waiting_times.values()]), \
            'Some waiting times are negative!'
        actual_max_customer_in_store = max(store.stats['num_customers_in_store'].values())
        assert actual_max_customer_in_store <= store.max_customers_in_store, \
            f'Somehow more people were in the store than allowed ' + \
            f'(Allowed: {store.max_customers_in_store} | Actual: {actual_max_customer_in_store})'

        assert store.num_customers_waiting_outside == 0, \
            f"Somehow, there are still {store.num_customers_waiting_outside} people waiting outside"
        if raise_test_error:
            raise RuntimeError("Test error")
    except Exception as e:
        time_string = datetime.datetime.now().strftime('%Y%m%d_%H%M%S')
        log_name = f'log_{time_string}_{uuid.uuid4().hex}.log'
        print(f'Sanity checks NOT passed. Something went wrong. Saving logs in {log_name}.')
        with open(log_name, 'w') as f:
            f.write('\n'.join(store.logs))
        if not raise_test_error:
            raise e
    if store.logger is not None:
        store.logger.info('Sanity checks passed!')
```

```
% PATH GENERATOR
import random
from typing import List, Dict, Optional
from covid19_supermarket_abm.utils.create_store_network import create_store_network
import networkx as nx
import numpy as np
from covid19_supermarket_abm.utils.create_synthetic_baskets import get_all_shortest_path_dicts
from covid19_supermarket_abm.utils.load_example_data import load_example_store_graph, load_example_paths
import json
```

```
"""The synthetic path generator generates a random customer path as follows:
    First, it samples the size K of the shopping basket using a log-
    normal random variable with parameter mu and sigma.
    Second, it chooses a random entrance node as the first node v_1 in the path.
    Third, it samples K random item nodes, chosen uniformly at random with replacement from item_nodes,
    which we denote by v_2, ... v_K+1.
    Fourth, it samples a random till node and exit node,
    which we denote by v_K+2 and v_K+3.
    The sequence v_1, ..., v_K+3 is a node sequence where the customer bought items, along the the
    entrance,
    till and exit
    nodes that they visited.
    Finally, we convert this sequence to a full path on the network using the shortest paths between
    consecutive nodes
    in the sequence.
    We use shortest_path_dict for this.
    For more information, see the Data section in https://arxiv.org/pdf/2010.07868.pdf

    The batch_size specifies how many paths we generate in each batch (for efficiency reasons).
    """

pos = {
    0: (12, -4), 1: (12, 0), 2: (12, 4), 3: (12, 8), 4: (12, 12), 5: (15, 12), 6: (18, 12), 7: (22, 12),
    8: (26, 12), 9: (30, 12), 10: (54, 7), 11: (30, 7), 12: (33, 13), 13: (37, 13), 14: (41, 13),
    15: (45, 13), 16: (50, 13), 17: (54, 13), 18: (58, 13), 19: (62, 13), 20: (66, 13), 21: (70, 13),
    22: (73, 13), 23: (76, 13), 24: (12, 17), 25: (18, 17), 26: (22, 17), 27: (26, 17), 28: (30, 17),
    29: (33, 17), 30: (37, 17), 31: (41, 17), 32: (45, 17), 33: (50, 17), 34: (54, 17), 35: (58, 17),
    36: (62, 17), 37: (66, 17), 38: (70, 17), 39: (73, 17), 40: (76, 17), 41: (82.5, 17), 42: (12, 21),
    43: (18, 21), 44: (22, 21), 45: (26, 21), 46: (30, 21), 47: (33, 21), 48: (37, 21), 49: (41, 21),
    50: (45, 21), 51: (50, 21), 52: (54, 21), 53: (58, 21), 54: (62, 21), 55: (66, 21), 56: (70, 21),
    57: (76, 21), 58: (78, 21), 59: (81, 21), 60: (84, 21), 61: (87, 21), 62: (12, 26), 63: (18, 26),
    64: (22, 26), 65: (26, 26), 66: (30, 26), 67: (33, 26), 68: (37, 26), 69: (41, 26), 70: (45, 26),
    71: (50, 26), 72: (54, 26), 73: (58, 26), 74: (62, 26), 75: (66, 26), 76: (70, 26), 77: (74, 26),
    78: (78, 26), 79: (81, 26), 80: (84, 26), 81: (87, 26), 82: (12, 32), 83: (18, 32), 84: (22, 32),
    85: (26, 32), 86: (30, 32), 87: (33, 32), 88: (37, 32), 89: (41, 32), 90: (45, 32), 91: (50, 32),
    92: (54, 32), 93: (58, 32), 94: (62, 32), 95: (66, 32), 96: (70, 32), 97: (74, 32), 98: (78, 32),
    99: (81, 32), 100: (84, 32), 101: (87, 32), 102: (12, 36), 103: (18, 38), 104: (22, 38),
    105: (26, 38), 106: (30, 38), 107: (33, 38), 108: (37, 38), 109: (41, 38), 110: (45, 38),
    111: (50, 38), 112: (54, 38), 113: (58, 38), 114: (62, 38), 115: (66, 38), 116: (70, 38),
    117: (74, 38), 118: (78, 38), 119: (81, 38), 120: (84, 38), 121: (87, 38), 122: (6, 36),
    123: (12, 41), 124: (18, 41), 125: (22, 41), 126: (26, 41),
    127: (30, 41), 128: (33, 41), 129: (35, 41), 130: (37, 41), 131: (41, 41), 132: (44, 41),
    133: (48, 41), 134: (51, 41), 135: (54, 41), 136: (58, 41), 137: (62, 41), 138: (66, 41),
    139: (68, 41), 140: (75, 41), 141: (78, 41), 142: (81, 41), 143: (84, 41), 144: (87, 41),
    145: (14, 45), 146: (16, 49), 147: (18, 47), 148: (22, 47), 149: (27, 47), 150: (31, 47),
    151: (35, 47), 152: (39, 47), 153: (43, 47), 154: (47, 47), 155: (51, 47), 156: (56, 47),
```

```
    157: (60, 47), 158: (64, 47), 159: (68, 47), 160: (72, 47), 161: (75, 47), 162: (78, 47),
    163: (81, 47), 164: (84, 47), 165: (87, 47), 166: (18, 53), 167: (22, 53), 168: (27, 53),
    169: (31, 53), 170: (35, 53), 171: (39, 53), 172: (43, 53), 173: (47, 53), 174: (51, 53),
    175: (56, 53), 176: (60, 53), 177: (64, 53), 178: (68, 53), 179: (72, 53), 180: (75, 53),
    181: (78, 53), 182: (81, 53), 183: (84, 53), 184: (87, 53), 185: (92, 53), 186: (20, 58),
    187: (25, 58), 188: (31, 58), 189: (35, 58), 190: (39, 58), 191: (43, 58), 192: (47, 58),
    193: (51, 58), 194: (56, 58), 195: (60, 58), 196: (64, 58),
    197: (68, 58), 198: (74, 21), 199: (72, 41)
}



# FOR ORIGINAL LAYOUT
edges = [
    (0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 12), (12, 13),
    (13, 14), (14, 15), (15, 16), (16, 17), (17, 18), (18, 19), (19, 20),
    (20, 21), (21, 22), (22, 23), (25, 26), (26, 27), (27, 28), (28, 29), (29, 30), (30, 31),
    (31, 32), (32, 33), (33, 34), (34, 35), (35, 36), (36, 37), (31, 38), (38, 39), (39, 40),
    (43, 44), (44, 45), (45, 46), (46, 47), (47, 48), (48, 49), (49, 50), (50, 51),
    (51, 52), (52, 53), (53, 54), (54, 55), (55, 56), (56, 57), (57, 58), (58, 59), (59, 60), (60, 61),
    (63, 64), (64, 65), (65, 66), (66, 67), (67, 68), (68, 69), (69, 70), (70, 71),
    (71, 72), (72, 73), (73, 74), (74, 75), (75, 76), (76, 77), (77, 78), (78, 79), (79, 80), (80, 81),
    (83, 84), (85, 86), (86, 87), (87, 88), (88, 89), (89, 90), (90, 91), (91, 92),
    (92, 93), (93, 94), (94, 95), (95, 96), (96, 97), (97, 98), (98, 99), (99, 100), (100, 101),
    (103, 104), (104, 105), (105, 106), (106, 107), (107, 108), (108, 109), (109, 110), (110, 111),
    (111, 112), (112, 113), (113, 114), (114, 115), (115, 116), (116, 117), (117, 118), (118, 119),
    (119, 120), (120, 121), (123, 124), (124, 125), (125, 126), (126, 127), (127, 128), (128, 129),
    (129, 130), (130, 131), (131, 132), (132, 133), (133, 134), (134, 135), (135, 136), (136, 137),
    (137, 138), (138, 139), (139, 140), (140, 141), (141, 142), (142, 143), (143, 144), (166, 147),
    (148, 149), (149, 150), (150, 151), (151, 152), (152, 153), (153, 154), (154, 155), (155, 156),
    (156, 157), (157, 158), (158, 159), (159, 160), (160, 161), (161, 162), (162, 163), (163, 164),
    (164, 165), (166, 167), (167, 168), (168, 169), (169, 170), (170, 171), (171, 172), (172, 173),
    (173, 174), (174, 175), (175, 176), (176, 177), (177, 178), (178, 179), (179, 180), (180, 181),
    (181, 182), (182, 183), (183, 184), (184, 185), (186, 187), (187, 188), (188, 189), (189, 190),
    (190, 191), (191, 192), (192, 193), (193, 194), (194, 195), (195, 196), (196, 197), (4, 24),
    (24, 42), (42, 62), (62, 82), (82, 102), (102, 123), (123, 146), (146, 186), (6, 25), (25, 43),
    (43, 63), (63, 83), (83, 103), (103, 124), (124, 147),
    (9, 11), (11, 28), (28, 46), (46, 66), (66, 86), (86, 106), (106, 127),
    (14, 31), (31, 49), (49, 69), (69, 89), (89, 109), (109, 131),
    (10, 17), (17, 34), (34, 52),
    (52, 72), (72, 92), (92, 112), (112, 135),
    (20, 37), (37, 55), (55, 75), (75, 95), (95, 115), (115, 138),
    (58, 78), (78, 98), (98, 118), (141, 162), (162, 181),
    (61, 81), (81, 101), (101, 121), (121, 144), (144, 165), (165, 184),
    (139, 159), (159, 178), (178, 197),
    (134, 155), (155, 174), (174, 193),
    (129, 151), (151, 170), (170, 189),
```

```
    (41, 59), (41, 60), (102, 122),
    (23, 40), (40, 57),
    (123, 145), (145, 146), (84, 85), (148, 147), (9, 11), (139, 199), (51, 198)
]

# FOR ONE-WAY SETUP
# edges = [
#      (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 12), (12, 13), (13, 14), (14, 15), (15, 16),
(16, 17), (17, 18), (18, 19), (19, 20),
#     (20, 21), (21, 22), (22, 23), (26, 25), (27, 26), (28, 27), (29, 28), (30, 29), (31, 30),
# (32, 31), (33, 32), (34, 33), (35, 34), (36, 35), (37, 36), (38, 37),
# (38, 31), (39, 38), (40, 39), (148, 147), (56, 198), (139, 199), (122, 102),  (124, 123), (199, 140),
(185, 184),
#     (43, 44), (44, 45), (45, 46), (46, 47), (47, 48), (48, 49), (49, 50), (50, 51),
#     (51, 52), (52, 53), (53, 54), (54, 55), (55, 56), (57, 58), (58, 59), (59, 60), (60, 61),
#     (64, 63), (65, 64), (66, 65), (67, 66), (68, 67), (69, 68), (70, 69), (71, 70),


# (72, 71), (73, 72), (74, 73), (75, 74), (76, 75), (77, 76), (78, 77), (79, 78), (80, 79), (81, 80),
#     (83, 84), (85, 86), (86, 87), (87, 88), (88, 89), (89, 90), (90, 91), (91, 92),
#     (92, 93), (93, 94), (94, 95), (95, 96), (96, 97), (97, 98), (98, 99), (99, 100), (100, 101),
#     (104, 103), (105, 104), (106, 105), (107, 106), (108, 107), (109, 108), (110, 109), (111, 110),
(84, 85),  (102, 122),


# (112, 111), (113, 112), (114, 113), (115, 114), (116, 115), (117, 116), (118, 117), (119, 118),
# (120, 119), (121, 120), (123, 124), (124, 125), (125, 126), (126, 127), (127, 128), (128, 129),
#     (129, 130), (130, 131), (131, 132), (132, 133), (133, 134), (134, 135), (135, 136),
# (136, 137), (137, 138), (138, 139), (140, 141), (141, 142), (142, 143), (143, 144),


#     (166, 167), (167, 168), (168, 169), (169, 170), (170, 171), (171, 172), (172, 173), (173, 174),
# (174, 175), (175, 176), (176, 177), (177, 178), (178, 179), (179, 180), (180, 181), (181, 182),
# (182, 183), (183, 184), (184, 185), (187, 186), (188, 187), (189, 188), (190, 189), (191, 190),
# (192, 191), (193, 192), (194, 193), (195, 194), (196, 195), (197, 196), (198, 57),

# (149, 148), (150, 149), (151, 150), (152, 151), (153, 152), (154, 153), (155, 154),
# (156, 155), (157, 156), (158, 157), (159, 158), (160, 159), (161, 160), (162, 161), (163, 162),
# (164, 163), (165, 164),


#     # verticals
#     (4, 24), (24, 42), (42, 62), (62, 82), (82, 102), (102, 123), (146, 166), (166, 186),
#     (147, 166),  (6, 25), (25, 43), (43, 63), (63, 83), (83, 103), (103, 124), (124, 147),
#     (11, 9), (11, 28), (28, 46), (46, 66), (66, 86), (86, 106), (106, 127),
#     (14, 31), (31, 49), (49, 69), (69, 89), (89, 109), (109, 131),
#     (10, 17), (17, 34), (34, 52), (52, 72), (72, 92), (92, 112), (112, 135),
```

```
#     (20, 37), (37, 55), (55, 75), (75, 95), (95, 115), (115, 138),
#     (58, 78), (78, 98), (98, 118), (141, 162), (162, 181),
#     (61, 81), (81, 101), (101, 121), (121, 144), (144, 165), (165, 184),
#     (139, 159), (159, 178), (178, 197), (0, 1), (1, 2), (2, 3), (3, 4),


#     (134, 155), (155, 174), (174, 193),
#     (129, 151), (151, 170), (170, 189),
#     (41, 59), (41, 60),
#     (23, 40), (40, 57),
#     (123, 145), (145, 146),   (9, 11),



#     (24, 4), (42, 24), (62, 42), (82, 62), (102, 82), (123, 102),
#     (4, 3), (3, 2), (2, 1), (1, 0),
#     (166, 147), (147, 124), (124, 103), (103, 83), (83, 63), (63, 43), (43, 25), (25, 6),
#     (127, 106), (106, 86), (86, 66), (66, 46), (46, 28), (28, 9),


#     (189, 170), (170, 151), (151, 129),
#     (193, 174), (174, 155), (155, 134),
#     (131, 109), (109, 89), (89, 69), (69, 49), (49, 31), (31, 14), (11, 9),
#     (135, 112), (112, 92), (92, 72), (72, 52), (52, 34), (34, 17), (17, 10), (197, 178),
#(178, 159), (159, 139), (138, 115), (115, 95), (95, 75), (75, 55), (55, 37), (37, 20),

#     (181, 162), (162, 141), (118, 98), (98, 78), (78, 58), (57, 40), (40, 23), (59, 41), (60, 41),
# (184, 165), (165, 144), (144, 121), (121, 101), (101, 81), (81, 61), (9, 11),
#     (186, 166), (166, 146), (146, 145), (145, 123), (147, 166)

# ]



# edges = [(1,0), (0, 1), (0, 2), (2, 3), (3, 2), (3, 1), (1, 3)]
# pos = {0: (0,0), 1: (0,1), 2: (1,0), 3: (1,1)}
G = create_store_network(pos, edges)
# print(G)
shortest_path_dict = get_all_shortest_path_dicts(G)
mu = 0.07
sigma = 0.76
entrance_nodes = [0, 11, 122, 10, 41, 185]
exit_nodes = [0, 11, 122, 10, 41, 185]
intersections = [9, 57, 189, 170, 151, 129, 106, 86, 66, 46, 28, 127, 193, 174, 155, 134, 112, 92, 72, 52,
34, 17, 197, 178, 159, 139, 115, 95, 75, 55, 37, 23, 49, 58, 78, 98, 118, 121, 101, 81, 61, 144, 165, 184,
181, 162, 141, 131, 109, 89, 69, 31, 14, 20, 40, 135, 138, 124, 147]
```

```
# item_nodes = [node for node in range(199) if node not in entrance_nodes and node not in exit_nodes]
item_nodes = [node for node in range(200) if node not in entrance_nodes and intersections]



def paths_generator_from_actual_paths(all_paths):
    num_paths = len(all_paths)
    while True:
        i = np.random.randint(0, num_paths)
        yield all_paths[i]


def path_generator_from_transition_matrix(tmatrix: List[List[int]], shortest_path_dict: Dict):
    while True:
        yield zone_path_to_full_path(create_one_path(tmatrix), shortest_path_dict)


def get_transition_matrix(all_paths, num_states):
    n = num_states + 1  # number of states

    transition_matrix = [[0] * n for _ in range(n)]
    for path in all_paths:
        for (i, j) in zip(path, path[1:]):
            transition_matrix[i][j] += 1
        transition_matrix[path[-1]][n - 1] += 1  # ending

    # now convert to probabilities:
    for row in transition_matrix:
        s = sum(row)
        if s > 0:
            row[:] = [f / s for f in row]
    return transition_matrix


def zone_path_to_full_path(zone_path, shortest_path_dict):
    full_path_dict = []
    for start, end in zip(zone_path[:-1], zone_path[1:]):
        full_path_dict += shortest_path_dict[start][end]
    return full_path_dict


def zone_path_to_full_path_multiple_paths(zone_path, shortest_path_dict):
    """Use this if there are shortest_path_dict gives
    multiple shortest_paths for each source and target"""
    L = []
    for start, end in zip(zone_path[:-1], zone_path[1:]):
        shortest_paths = shortest_path_dict[start][end]
```

```
        num_shortest_paths = len(shortest_paths)
        if num_shortest_paths > 0:
            i = np.random.randint(num_shortest_paths)
        else:
            i = 0
        L += shortest_paths[i]
    return L

def sample_num_products_in_basket_batch(mu, sigma, num_baskets):
    """
    Sample the number of items in basket using Mixed Poisson Log-Normal distribution.
    Reference: Sorensen H, Bogomolova S, Anderson K, Trinh G, Sharp A, Kennedy R, et al.
    Fundamental patterns of in-store shopper behavior.
    Journal of Retailing and Consumer Services. 2017;37:182{194.

    :param mu: Mean value of the underlying normal distribution
    :param sigma: Standard deviation of the underlying normal distribution
    :param num_baskets: number of baskets to sample
    :return: List of length num_baskets with the number of items in each basket
    """

    norm = np.random.lognormal(mean=mu, sigma=sigma, size=3 * num_baskets)
    num_items = np.random.poisson(norm)
    num_items = num_items[num_items > 0]
    assert len(num_items) >= num_baskets, \
        f"Somehow didn't get the enough non-zero baskets ({num_items} <= {num_baskets} (size))"
    return num_items[:num_baskets]


def create_random_item_paths(num_items, entrance_nodes, exit_nodes, item_nodes):
    """
    Create random item path based on the number of items in each
    basket and the shelves were items are located.
    We choose items uniformly at random from all item_nodes.
    We also choose a random entrance node, till node, and exit node
    (sampled uniformly at random from the
    corresponding nodes).
    """
    num_baskets = len(num_items)
    random_entrance_nodes = np.random.choice(entrance_nodes, size=num_baskets)
    # random_till_nodes = np.random.choice(till_nodes, size=num_baskets)
    random_exit_nodes = np.random.choice(exit_nodes, size=num_baskets)
    concatenated_baskets = np.random.choice(item_nodes, size=np.sum(num_items))
    break_points = np.cumsum(num_items)
    item_paths = []
    start = 0
    i = 0
```

```
    for end in break_points:
        entrance = random_entrance_nodes[i]
        # till = random_till_nodes[i]
        exit = random_exit_nodes[i]
        basket = [entrance] + list(concatenated_baskets[start:end]) + [exit]
        item_paths.append(basket)
        start = end
        i += 1
    return item_paths


def sythetic_paths_generator(mu, sigma, entrance_nodes, exit_nodes, item_nodes,
                                            shortest_path_dict, batch_size=1000000):
    """The synthetic path generator generates a random customer path as follows:
    First, it samples the size K of the shopping basket using a log-
    normal random variable with parameter mu and sigma.
    Second, it chooses a random entrance node as the first node v_1 in the path.
    Third, it samples K random item nodes, chosen uniformly at random
    with replacement from item_nodes,
    which we denote by v_2, ... v_K+1.
    Fourth, it samples a random till node and exit node, which we denote by v_K+2 and v_K+3.
    The sequence v_1, ..., v_K+3 is a node sequence where the customer
    bought items,
    along the the entrance, till and exit
    nodes that they visited.
    Finally, we convert this sequence to a full path on the network
    using the shortest paths between consecutive nodes
    in the sequence.
    We use shortest_path_dict for this.
    For more information, see the Data section in https://arxiv.org/pdf/2010.07868.pdf

    The batch_size specifies how many paths we generate in each batch (for efficiency reasons).
    """
    mylist = []
    while True:
        num_items = sample_num_products_in_basket_batch(mu, sigma, batch_size)
        item_paths = create_random_item_paths(num_items, entrance_nodes, exit_nodes, item_nodes)
        for item_path in item_paths:
            full_path = zone_path_to_full_path_multiple_paths(item_path, shortest_path_dict)
            mylist.append(full_path)
        break
    return mylist


def get_next_term(num_states, trow):
    return random.choices(range(num_states), trow)[0]
```

```
def create_one_path(tmatrix: List[List[int]]):
    num_states = len(tmatrix)
    start_term = 0
    end = num_states - 1
    chain = [start_term]
    length = 1
    while True:
        current_position = get_next_term(num_states, tmatrix[chain[-1]])
        if current_position == end:
            break
        elif length > 100000:
            print('Generated is over 100000 stops long. Something must have gone wrong!')
            break
        chain.append(current_position)
        length += 1
    return chain


def replace_till_zone(path, till_zone, all_till_zones):
    assert path[-1] == till_zone, f'Final zone is not {till_zone}, but {path[-1]}'
    path[-1] = np.random.choice(all_till_zones)
    return path


def get_path_generator(path_generation: str = 'empirical', G: Optional[nx.Graph]=None,
                       full_paths: Optional[List[List[int]]]=None,
                       zone_paths: Optional[List[List[int]]]=None,
                       synthetic_path_generator_args: Optional[list] = None):
    """Create path generator functions.
    Note that a zone path is a sequence of zones that a customer purchased items from, so
    consecutive zones in the sequence
    may not be adjacent in the store graph. We map the zone path to the full shopping path by
    assuming that customers walk shortest paths between purchases."""

    # Decide how paths are generated
    if path_generation == 'empirical':
        path_generator_function = paths_generator_from_actual_paths
        if full_paths is not None:
            path_generator_args = [full_paths]
        else:
            assert zone_paths is not None, "If you use path_generation='empirical', you need to
            specify either zone_paths or full_paths"
            assert G is not None, "If you use path_generation='empirical' with zone_paths,
            you need to input the store network G"
            shortest_path_dict = dict(nx.all_pairs_dijkstra_path(G))
            shopping_paths = [zone_path_to_full_path(path, shortest_path_dict) for path in zone_paths]
```

```
            full_paths = [zone_path_to_full_path(path, shortest_path_dict) for path in shopping_paths]
            path_generator_args = [full_paths]
    elif path_generation == 'synthetic':
        assert synthetic_path_generator_args is not None, \
            "If you use path_generation='synthetic', " \
            "you need to input synthetic_path_generator_args=" \
            "[mu, sigma, entrance_nodes, till_nodes, exit_nodes, item_nodes, shortest_path_dict]"
        assert type(synthetic_path_generator_args) is list, \
            "If you use path_generation='synthetic', " \
            "you need to input synthetic_path_generator_args=" \
            "[mu, sigma, entrance_nodes, till_nodes, exit_nodes, item_nodes, shortest_path_dict]"
        assert len(synthetic_path_generator_args) == 7, \
            "If you use path_generation='synthetic', " \
            "you need to input synthetic_path_generator_args=" \
            "[mu, sigma, entrance_nodes, till_nodes, exit_nodes, item_nodes, shortest_path_dict]"
        path_generator_function = sythetic_paths_generator
        path_generator_args = synthetic_path_generator_args  # [mu, sigma, entrance_nodes,
        # till_nodes, exit_nodes, item_nodes, shortest_path_dict]
    elif path_generation == 'tmatrix':
        assert zone_paths is not None, "If you use path_generation='tmatrix',
        you need to input zone_paths"
        assert G is not None, "If you use path_generation='tmatrix',
        you need to input the store network G"
        shortest_path_dict = dict(nx.all_pairs_dijkstra_path(G))
        shopping_paths = [zone_path_to_full_path(path, shortest_path_dict) for path in zone_paths]
        tmatrix = get_transition_matrix(shopping_paths, len(G))
        path_generator_function = path_generator_from_transition_matrix
        path_generator_args = [tmatrix, shortest_path_dict]
    else:
        raise ValueError(f'Unknown path_generation scheme == {path_generation}')
    return path_generator_function, path_generator_args


zone_paths = sythetic_paths_generator(mu, sigma, entrance_nodes, exit_nodes, item_nodes,
shortest_path_dict)

# file_path = '10^6_DIRECTIONAL.json' # for directed
file_path = '10_6.json'

with open(file_path, 'w') as json_file:
    json.dump(zone_paths, json_file)

print(f"The list has been written to {file_path}")
```