Making agents in multi-agent systems adaptive with the inclusion of a dynamic epsilon-greedy value

Nishq Ravindranath

Supervisors: Dr. Dhirendra Singh, Dr. Vincent Lemiale

Minor Thesis

School of Computer Science and Information Technology

RMIT University

Melbourne, Australia

June, 2022.

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of this thesis is the result of work which has been carried out since the official commencement date of approved research program; and any editorial work, paid or unpaid, carried out by a third party is duly acknowledged.

Nishq Ravindranath
19th June, 2022

Acknowledgements

I would like to acknowledge and extend my gratitude to my supervisors Dhirendra Singh and Vincent Lemiale for their support, guidance and mentorship over the course of the last year, and for providing me the opportunity to conduct research and work with them. I would also like to thank the course supervisor, Hai Dong, for facilitating the course and providing guidance. Finally I would like to thank my friends and family, without whom I would never be able to make it as far as I have.

Summary

Since the COVID-19 pandemic, a lot of research work has been done to simulate the different policy interventions employed by countries to curb the spread of the virus, while balancing the impact on the economy. Many of these simulations do not account for people's ability to adapt and react to new situations and global events such as COVID-19, and presume people maintain their pre-pandemic behaviours. In this thesis we explore how to make agents in such simulations more adaptive so that they closely mimic their real-world counterparts. We also discuss the usage of this adaptive property in other applications that simulate multiple agents.

Abstract

This thesis explores the inclusion of adaptive agents in a multi-agent system that learn using a performance-based, dynamic epsilon-greedy value for exploration and exploitation. The calculation of this epsilon-greedy value is done based on a normalization function that scales each agent against an ideal that represents the goal of the system. The gap identified in current research is the lack of both implementing and subsequently understanding agent adaptiveness to policy intervention changes during a pandemic. For this research, we devised a simplified grocery store model wherein agents have to adapt in order to avoid crowds during a pandemic. Multiple normalization functions were experimented with, and from our experiments, it was found that a logarithm function with a multiplicative constant of 0.25 resulted in the least time taken for convergence. However, the choice of the normalization function depends on the application it is being used for and the scoring function that is crafted. This work can be used to help understand agent adaptiveness to policy changes during a pandemic, which can help with lessening the negative economic impact of them. Furthermore, the mechanism of a dynamic epsilon-greedy value for adaptiveness can also be applied in other multi-agent applications.

Contents

1 Introduction

The COVID-19 pandemic has brought about many new challenges to countries and their local governments in planning and implementing state or nationwide policies that can balance curbing the spread of the virus while managing to keep certain sectors open to the general public. The various policies adopted in different parts of the world have not only impacted the spread of the virus differently but have also had severe economic consequences [1]. Agent-based modelling has been used extensively in order to simulate the spread of the virus among populations [2]. This has aided with the ability to make predictions as well as assist with policy decisions such as lockdowns, mobility restrictions, mask mandates and social distancing protocols for governments across the world. However, many of these research works lack the property of having agents adapt to these policy changes and further understanding how that adaptiveness can impact the value of these policy changes, with respect to the economical consequences.

Negative economic consequences were suffered as a result of compulsory restrictions such as full lockdowns, store limits and curfews in many different parts of the world. Varying degrees of these compulsory restrictions and different hybrid policies [3] were employed by different countries. Agent-based models with non-adaptive agents tend to model these agents with static plans or schedules [4] [5], which assumes pre-established behaviours. This is why compulsory restrictions tend to result in the best solution in terms of curbing the spread of the virus. However, in the real world, people tend to be reactive and adaptive to new situations and policies. They may have an intrinsic need to avoid catching the virus themselves without being directed to follow compulsory restrictions, which can cause them to adapt to the situation by modifying their behaviours. This is an aspect that is often unaccounted for. Therefore, by modelling this in agent-based models, and then observing the rate of spread of the virus, we may be able to more accurately assess when and how certain policies and restrictions with respect to sectors important to the economy can be loosened or lifted. It can thus allow for a better estimate about the impact on the economy, as well as a better assessment of the quality of different policy interventions, in terms of curbing the spread of the virus.

This idea served as the foundation behind this research, which is to assign each agent in an agent-based modelling simulation adaptive properties that allow them to maximize a specific utility. At an application-agnostic level, this utility is determined by each agent's performance with respect to a certain goal. Agents attempt to maximize this utility by either exploring a set of actions at their disposal if they are under-performing, or sticking with and exploiting their current set of actions or strategies if they are performing well. In many reinforcement learning applications, this exploration and exploitation of the set of actions by agents is represented by an epsilon-greedy value in the range of [0,1] that acts as a probability [6]. This value is commonly set as a static parameter at the beginning of simulations, often multiplied by scalars that act as decay rates to slow down exploration and have more exploitation in the simulations over time [6].

In this research, we propose the use of this parameter in a different way - that of assigning each agent their own epsilon-greedy value that changes dynamically based on their performance. This epsilon-greedy value is computed for each agent by normalizing their performance relative to either the best performing agent, or a known ideal performance value, using a normalization function. Different normalization functions with unique curves and properties result in different epsilon-greedy values; thus, resulting in differences in overall exploration and exploitation by the agents. For the purpose of this research, we explored 3 classes of these functions with different constants for each, and analyzed how they perform relative to each other. We did this by applying this idea of a dynamic epsilon-greedy value to a grocery store application model we devised in NetLogo [7].

The grocery store application focuses on agents that visit a grocery store during a pandemic, with the goal of reducing crowdedness in the grocery stores. As smaller crowds result in fewer person-to-person contacts, this can help reduce the spread of the virus. An overcrowding threshold is set in the application, and is akin to person limits that were set in grocery stores in different parts of the world [8]. This however, is not a hard limit or

compulsory restriction on the number of agents allowed in the store, but a goal that represents an ideal or an optimal system. The agents adapt themselves in order to achieve this optimal system. This is done since we want to know how long it would take for agents to adapt this way without compulsory restrictions, and only by using their own intrinsic need to avoid crowds. The overcrowding threshold thus serves as an intuition the agent has about how they should avoid crowds. Agents in the model adapt towards the goal of distributing themselves into different time slots in such a way that they stay below this overcrowding threshold. They adapt using a dynamic epsilon-greedy value that determines their exploration and exploitation of the set of available time slots during which they can go grocery shopping. A complete description of the model can be found in Section 3.4.1. Additionally, a technical description of the model in the ODD format [9] can be found in Appendix A.

1.1 Importance

This research is relevant for both the ongoing and any possible future pandemics. The gap identified in current research is the lack of both implementing and subsequently understanding agent adaptiveness to policy changes, which this research aims to fulfill. Such an approach can be used to better assess real-world policy changes, with more precision and flexibility through monitoring this adaptiveness, as the models can more accurately represent human behaviour, with results that are closer to the real world. This can then help retail or grocery stores with planning and inventory management, by understanding what times of the day are most crowded. Furthermore, it can also help inform the opening and closing hours, which can then be used to plan staff rotations more efficiently.

The technique utilized here can be expanded upon and applied to different contexts such as schools and universities, hospitality venues (restaurants, bars, cafes, etc.), transportation services, businesses and corporations, etc. Policies that account for agent adaptiveness can then be devised to cater to the opening of individual sectors, while also being tailored to specific demographics through understanding which policy works best and for whom, instead of completely opening everything or shutting everything down. This is of particular interest to governments at the local and federal level all over the world, since tailored policies that account for adaptiveness as such can help to keep the economy afloat, which is a high priority for many countries. The success of this research will therefore help us to be better prepared for future pandemics.

Furthermore, the mechanism employed for adaptiveness using a dynamic epsilon-greedy value assigned to each agent can be extended to other applications as well. Multi-agent applications that involve any kind overarching goal of the system that the agents can maximize their utility towards can adopt this mechanism in a simple way to make their systems adaptive. This can include applications in planning and scheduling, traffic simulations, economic modelling and multi-agent games just to name a few.

1.2 Research Questions

- How can an epsilon-greedy value be applied dynamically to multi-agent simulations?
- Does the inclusion of adaptive properties in agents result in the emergence of an optimal system?
- What is the best epsilon normalization function for this purpose?

2 Literature Review

2.1 Adaptive Agents

Adaptive agents encompass a very broad field that includes many different applications. Some common applications include multi-agent games [10], predicting financial markets [11], predator-prey simulations [12], robotics and automation coordination techniques [13, 14, 15], autonomous driving [16] as well as traffic simulation tools like MATSIM [17]. There are also many different mechanisms for learning and adapting that are used in this field - Reinforcement Learning [18], Genetic Algorithms [19], Co-evolutionary Algorithms [20], and Machine Learning [21] just to name a few.

Since this research falls into the broader domain of multi-agent learning, it is important to consider where it fits. One way of organizing multi-agent learning is into the following categories: agents that learn communication, agents that learn coordination, agents that model other agents by reasoning, and emergent system behaviours that arise from multi-agent settings [22]. The first 3 are not relevant to the application in this research since we are assuming that in the real world, when individuals are modifying their schedules in order to minimize contacts with others in grocery stores, they do not communicate their schedules to strangers, or try to cooperate/coordinate their schedules with them either. They also do not think about which specific strangers will go grocery shopping at the same time as them, instead they make estimates about crowdedness based on their past experiences. This research falls under the category of analyzing emergent system behaviours. This is appropriate to the application considered in this research since we will be analyzing the emergent impact on the system from adaptive agents.

An important consideration when dealing with adaptive agents is the issue of dynamic, non-stationary environments [23]. Multi-agent settings bring about a new challenge for adaptive agents: the problem of agents having to model a non-stationary environment that becomes unpredictable due to other adaptive agents being present in and interacting with it. In a dynamic environment, the agents have to sense, update and model information about changes in the environment which can result in more overhead compared to stationary environments, where information from the environment is unchanged for the lifetime of the simulation [24]. Furthermore, it can also cause a high amount of variation in the learning process for each agent, since the environment keeps changing. In the grocery store application considered here, agents sense the environment as a function of the crowdedness of the store, and subsequently change and adapt their time slots. This in turn affects the crowdedness of the store in those time slots, changing the environment and causing the agents to adapt again. The environment in the application considered here is thus dynamic and non-stationary.

When designing a system with adaptive agents, it is imperative to know where the learning happens. The learning can happen either at a global level, with the system learning the strategies for all agents as a collective, or at the local agent level, where the "learning" effect on the system is an emergent property. Works that look at role-based cooperation have found that local agent learning (when there is no explicit communication or coordination protocols between agents) is most effective through stigmergy, i.e. the effects of each agent's actions on the environment [25]. This research hopes to emulate this stigmergy, by having the agents explore the environment and adapt their strategies accordingly.

Exploration of an environment by agents here entails exploring or trying out actions from a set of actions available to the agents, and the effect that carrying out those actions has on the environment and other agents. Exploiting is the notion of sticking with a set of actions that has already worked for an agent, thus exploiting what is best for them [6]. This trade-off between exploration and exploitation is commonly represented by an epsilon-greedy value that acts as s either a probability or a trade-off ratio with respect to how much exploration and exploitation is done from the set of actions, and is thus usually in the range of [0,1] [6]. Many popular reinforcement learning applications use a static epsilon-greedy value for all agents that has a set rate of decay [26, 27, 28].

It is important to consider the work that has been done previously with using an epsilon-greedy value in a more dynamic way. There has been research conducted previously that has explored varying the rate of decay based on the environment and the agent's reward [29]. There have also been works that evaluate the effects of increasing the exploration parameter in situations where the environment is not well known and the agent is uncertain [30]. Furthermore, there are different exploration schemes that are used in reinforcement learning such as Boltzmann exploration, which is a classic exponential weighting strategy based on the Boltzmann distribution [31, 32]. There are works [33, 34, 35] that have taken a more performance-based approach that we are proposing in this project, wherein the agents are encouraged to explore more when their weekly score is lower relative to other agents, and exploit more when their weekly score is higher relative to other agents.

2.2 COVID-19 Modelling

There has been an increasing amount of research in the agent-based modelling space since the start of the pandemic to simulate the spread of COVID-19. A lot of the work published has been around the simulating of different policy interventions and restrictions [36, 37, 38] to understand the spread of the virus, the impact on the economy [39], as well as forecasting predictions based on current scenarios in different countries [40]. However, many of these approaches consist of non-adaptive agents that follow static plans or schedules, and don't have any capacity for learning to maximize any kind of utility during the simulations, which is the key difference that we are proposing is useful as a part of this research. While there have been a few studies that have explored using reinforcement learning in an agent-based setting [41], they only simulate system-level learning, and not agent-level learning, or agent-level adaptations.

Since we are looking specifically at grocery stores for the application that is considered in this research, we looked at how they were modelled during the pandemic. In terms of modelling agents that go to grocery stores during the pandemic, just as with the research that has been done to simulate the spread of disease, the simulations conducted are in non-learning settings [42]. There was also a lot of work done to focus on in-store specific policies, such as mask-wearing, agent movement restrictions with one-way aisles, changing store layouts, capacity limits, and social distance markers to assess their importance in curbing the spread of the virus [43, 44]. Furthermore, there have been approaches that conducted research on which types of locations too close to avoid the spread of the virus, by simulating different combinations of location types and ranking each one's importance [45].

2.3 The El Farol Bar Problem

The grocery store application we considered in this research is loosely similar to the El Farol Bar problem [46], which is classed as a special case of market entry games [47] in which simulated agents choose whether or not to go to a bar on a Thursday night, based on how crowded they believe the bar will be. Agents have an adjustable "memory", or attendance history of the bar the last few times, as well as a number of "strategies", or static weights that are multiplied by the attendance history from the memory at their disposal. Under optimal settings with a reasonable crowdedness threshold, the system can converge to an optimum with each Thursday night being under the crowdedness threshold [48].

2.3.1 Agent Adaptiveness in the El Farol Bar Problem

Reinforcement learning has been added to the El Farol problem previously in unique ways [49, 50]. Franke [50] assigned probability values to the different actions available to the agents. Based on an agent's performance, positive or negative reinforcement was given to these actions that altered the probability values, resulting in certain actions being chosen more or less often. While this technique works well for the El Farol problem, it may not scale well to problem here, as the state space tends to grow exponentially large with reinforcement

learning problems when more than one agent is taken into consideration. The state space in this research extends beyond a simple "going" or "not going". As described in Section 3.4, agents are given a choice in the number of time slot configurations that they want to complete their shopping quota in (1 visit, 2 visits or 4 visits), and the choice to pick from time slots on any given day within the opening hours of the grocery stores. Due to the different permutations and combinations of those two (which differs for each agent based on their shopping quota), the state space can grow exponentially large.

Bell [51] took the approach of testing the impact on the equilibrium of the system from increasing the proportion of adaptive learning agents, noting that it leads to an endogenously evolving, non-stationary environment. Agents were awarded positively for going when the bar wasn't crowded, and not going when the bar was crowded, a policy that is not pertinent to our research since the agents don't know the crowdedness of the store unless they decide to go. It was observed that the adaptive system rapidly converged to a fixed average or aggregate behavior despite the slow convergence of individual learning rules. Furthermore, it was also observed that the behaviour of individual agents and the system as a whole was very sensitive to the initial conditions of the simulations, and could result in dramatically different results.

3 Methodology

3.1 Background

3.1.1 Agent-based Modelling

Agent-based models are computer simulations that are used to observe the interactions between decision-making entities called agents [7]. Agents are assigned certain attributes and properties, as is the environment in which they interact. These interactions can result in emergent phenomena that uniquely differ from the effects of the agents individually.

3.1.2 Adaptive Agents

Adaptive agents, in the context of this research, are those that have the capacity to learn or optimize their behavior, by choosing from a set of actions available to them. They do this based on their experience, either with the environment or with other agents, which is measured by their score (or utility), using a scoring function.

3.1.3 Scoring Functions

Scoring functions, also commonly known as reward functions, are functions that are crafted to assign scores to agents in a simulation based on their performance. The scores assigned using these functions are then used to have the agents improve by making them alter their actions, decisions or strategies from a set that is available to them at their disposal.

3.1.4 Exploration and Exploitation

Exploration and exploitation are common terminologies associated with how an agent chooses to navigate about a state space, given a set of actions available to them. In the context of this research, more exploration means trying out new actions that the agent hasn't tried before, whereas exploitation means sticking with a previous set of actions that was getting the agent a high score.

3.1.5 Grocery Store Application

As mentioned in the introduction, the application considered in this research is that of agents going to a grocery store for shopping during a pandemic. The simulation is broken up into hourly time slots that represent the opening hours of the store, during which the agents can visit it for carrying out their shopping. Each agent is given a shopping quota, which represents the number of time slots they need to go during the week in order to complete their weekly shopping. Agents are randomly assigned these time slots at the beginning of the simulation. At the end of every week, the agents are scored using a scoring function that is based on the relative crowdedness of all their time slots. This is covered in more depth in Section 3.2. The score is then used to compute the epsilon-greedy value of each agent using an epsilon normalization function (covered in Section 3.3), which acts as a probability value that dictates whether they will explore or exploit other time slots. The specific details of the model setup are discussed in Section 3.4.1. Additionally, a complete technical description of the model is available in the ODD format in Appendix A

3.2 Intuition behind Dynamic Epsilon-Greedy Value

As mentioned in the introduction, this research uses an epsilon-greedy value dynamically to have agents achieve a specific goal. This is done by having the epsilon-greedy value for each agent change based on their performance, which is determined by an application-specific scoring function. This is calculated for each agent at the end of an iteration, with higher scores awarded to agents who perform better with respect to the goals of the application. For example, in the grocery store application, each agent's score is computed at the end of the week, with higher scores given to those agents who avoid crowds and go to the grocery store during sparsely-crowded time slots, and lower scores given to those agents who go during dense and overcrowded time slots.

The scoring function chosen for the purpose of the application in this research is defined as follows:

$$score = (\frac{1}{current\_attendance}) * 100 \qquad \textbf{(Equation 3.1)}$$

This function simply takes the inverse of the current attendance at the store and multiplies it by a 100 (to ensure easier calculations and better readability of graphs). Scoring the agents in this way as a function of the current attendance results in fewer attendees in the store yielding higher scores and vice versa. The scores for each time slot are added up for each agent that cumulates in their total weekly score. For example, given a *current_attendance* of 16 people in a store, the subsequent score can be calculated as:

$$score = (\frac{1}{16}) * 100$$

$$score = 6.25$$

Whereas a lower *current_attendance* will yield a higher score, as such:

$$score = (\frac{1}{8}) * 100$$

$$score = 12.5$$

Based on this score that is assigned to each agent, the epsilon-greedy values for each agent are then calculated, using an epsilon normalization function. Different classes of functions can be used for this calculation, and were experimented with for the purpose of this research that we cover in Section 3.3. These epsilon normalization functions can either normalize the values based on the minimum and maximum scores of all the agents, or, as we did for the grocery store application, against an "ideal score". This ideal score is actually the score that each agent would receive on average if the system was at an optimal. For example, in the grocery store application, an optimal would equate to all time slots being under a specific overcrowding threshold that we set.

The intuition behind this is that we do not want the agents to greedily optimize towards the maximum possible achievable score in the simulation. This is because depending on the nature of the setup it could result in a zero-sum game where a higher score for one agent comes at the expense of other agents receiving lower scores. While the average score would still be the same, there would be a heavily skewed distribution that favors some and not others, and it would not address the purpose of achieving an optimal system, which in the application considered here would prevent overcrowdedness in all time slots. Therefore, an ideal score is preferable for applications that have zero-sum games in their setup. For the grocery store application, the ideal score is determined using the following function:

$$ideal\_score = (\frac{1}{overcrowding\_threshold}) * shopping\_quota * 100 \qquad \textbf{(Equation 3.2)}$$

The ideal score is computed in terms of the inverse of the overcrowding threshold for each time slot, multiplied by the shopping quota, which is each agent's weekly number of

time slots that they go shopping, and a 100. This is just like the way the score per time slot is computed in Equation 3.1, albeit with an overcrowding threshold incorporating the goal of the system, and accounting for the shopping quota because we are not just considering the score per time slot but the ideal score over all time slots. An example of how this is computed, with an *optimal_attendance* of 17 and a *shopping_quota* of 4, and can be seen as follows:

$$ideal\_score = (\frac{1}{17}) * 4 * 100$$

$$ideal\_score = 23.5$$

The epsilon-greedy value for each agent is then computed by normalizing each agent's weekly score against the ideal score (using an epsilon normalization function) in a range of 0 to 1. Agents that have a higher score have a higher epsilon-greedy value, since they need to exploit their strategy that is already yielding them a high score, whereas agents that have a lower weekly score have a lower epsilon greedy-value, since they need to explore the other strategies that may yield them a better score.

The intuition behind this parameter therefore is that it is merit-based, with agents that perform worse having to try other strategies from the actions available to them in order to get better scores. And since these values are normalized relative to an ideal or a maximum possible, it is the system as a collective that is trying to optimize for the best solution. This is visualized in Figure 1, where we can see how all agents start off with maximum exploration, and, over multiple iterations, they tend more towards exploitation, since they are adapting, performing better and getting scores closer to the ideal. The collective agent set as a whole is thus improving their performance. Once they all settle on the best strategy (or time slot configuration for the grocery store application), eventually all the agents tend more towards exploitation.

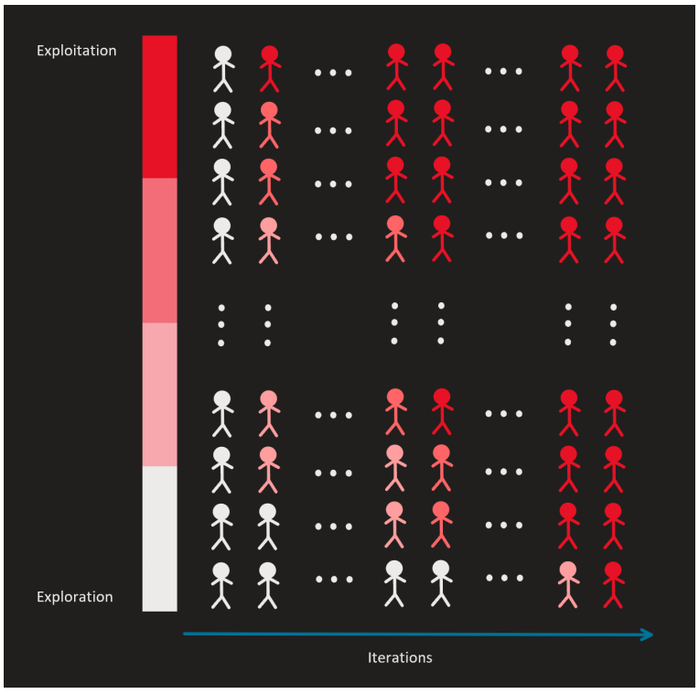

Figure 1: Epsilon-greedy values for agents visualized over iterations.

3.3 Epsilon Normalization Functions

In order to compute a dynamic epsilon-greedy value for agents based on their scores, different functions with different properties were considered. We started off by picking a base function - one that simply divides the weekly score by the ideal score to get an epsilon-greedy value that is proportional to the weekly score. However, it was realized that in order to change this relative proportion to skew more towards either exploration or exploitation, this base function would need to be raised to the power of different constant values, which would increase/decrease the overall exploration and exploitation of the agents.

Furthermore, entirely different classes of functions with with varying constant values can be experimented with for even more varied results. The logarithmic and sigmoid class of functions were considered as well for experimentation, since they have different properties due to the nature of their curves, and varying constants associated with them can result in unique differences for the overall exploration and exploitation of agents. This is examined in more detail in the sections below.

3.3.1 Base Functions

This class of functions is represented by $\frac{w_s^c}{i_s^c}$, where w_s is each agent's weekly score, i_s is the ideal score agents can achieve, and c is the power to which both are raised.

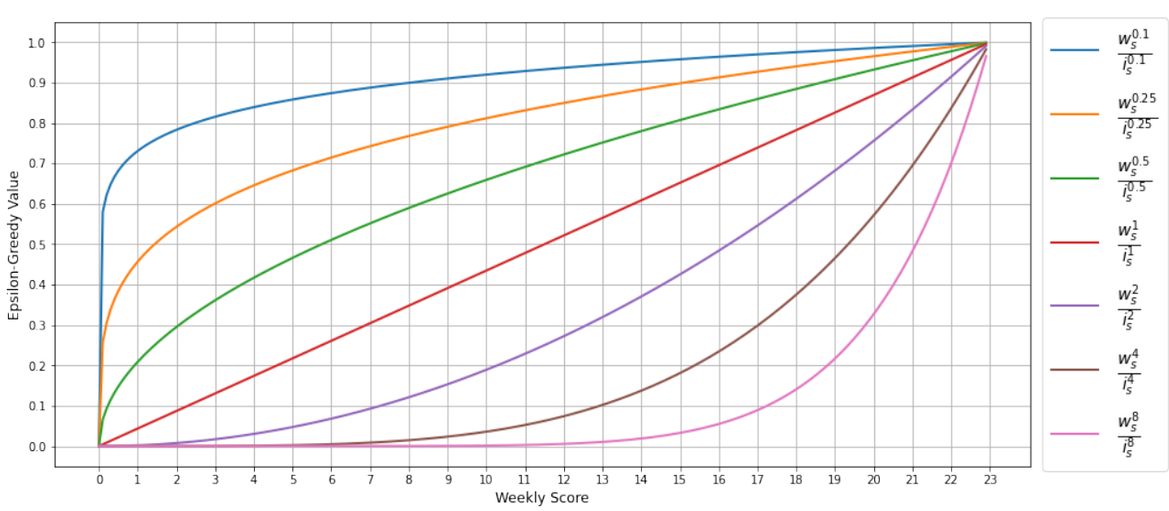

Figure 2: Base class of functions, raised to different powers of c.

This class of functions acts as a simple metric to get the proportion of one agent's performance over the ideal score. Figure 2 shows how changing the value of c impacts the epsilon-greedy value of the agent, with the x-axis representing the weekly score of the agent, and the y-axis representing the corresponding epsilon-greedy value. These findings can be summarized as follows:

- When $c = 1$, the function results in an epsilon-greedy value that is proportional to the agent's weekly score relative to the ideal score.
- As c becomes smaller than 1, the functions tend to become more concave in nature. The weekly-score required to cross the threshold of an epsilon-greedy value of 0.5 decreases, and as such a lower score is required to exploit. This results in higher epsilon-greedy values for all agents, leading to less exploration overall and more exploitation.
- As c becomes greater than 1, the functions tend to become more convex in nature. The weekly-score required to cross the threshold of an epsilon-greedy value of 0.5 increases, and as such a higher score is required to exploit. This results in lower epsilon-greedy values for all agents, leading to more exploration overall and less exploitation.

3.3.2 Logarithmic Functions

This class of functions is represented by $\frac{log(c * w_s)}{log(c * i_s)}$, where w_s is each agent's weekly score, i_s is the ideal score agents can achieve, and c is the constant that both are multiplied to.

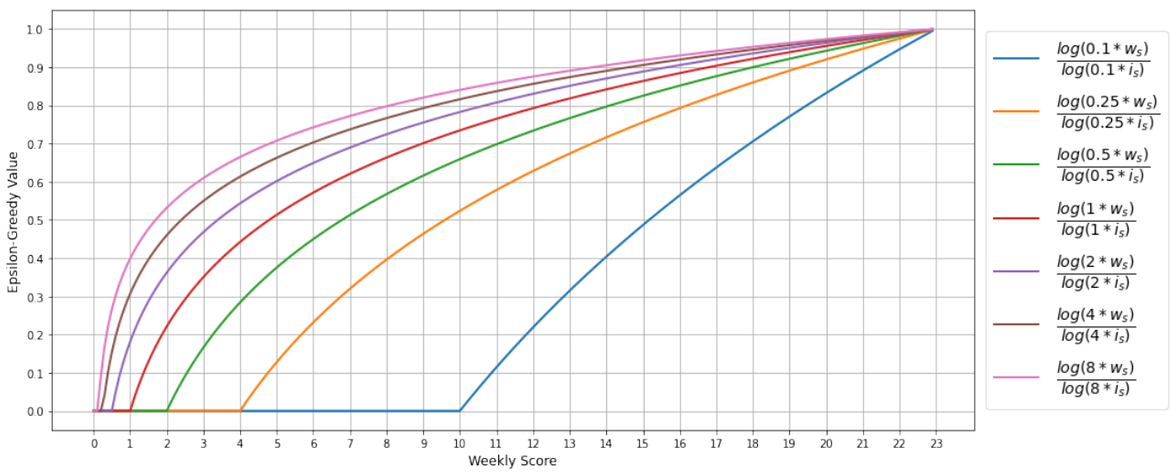

Figure 3: Logarithmic class of functions, multiplied by different values of c.

The logarithmic class of functions are concave in nature, and on average tend to lead to less exploration and more exploitation. Figure 3 shows how changing the value of c impacts the epsilon-greedy value of the agent, with the x-axis representing the score of the agent, and the y-axis representing the corresponding epsilon-greedy value. These findings can be summarized as follows:

- Increasing the value of c from 0.1 as depicted in Figure 3 tends to make the functions more concave in nature. The weekly-score required to cross the threshold of an epsilon-greedy value of 0.5 decreases, and as such a lower score is required to exploit. This results in higher epsilon-greedy values for all agents, leading to more exploitation overall and less exploration.
- The functions are thresholded, meaning a certain minimum score is required to have an epsilon-greedy value greater than 0. This threshold is greatest when c is equal to 0.1, with a threshold of 10, and decreases as the value of c increases.

3.3.3 Sigmoid Functions

This class of functions is represented by $\frac{1}{1 + e^{-c(w_s - \frac{i_s}{2})}}$, where w_s is each agent's weekly score, i_s is the ideal score agents can achieve, and c is the constant that determines the shape of the function.

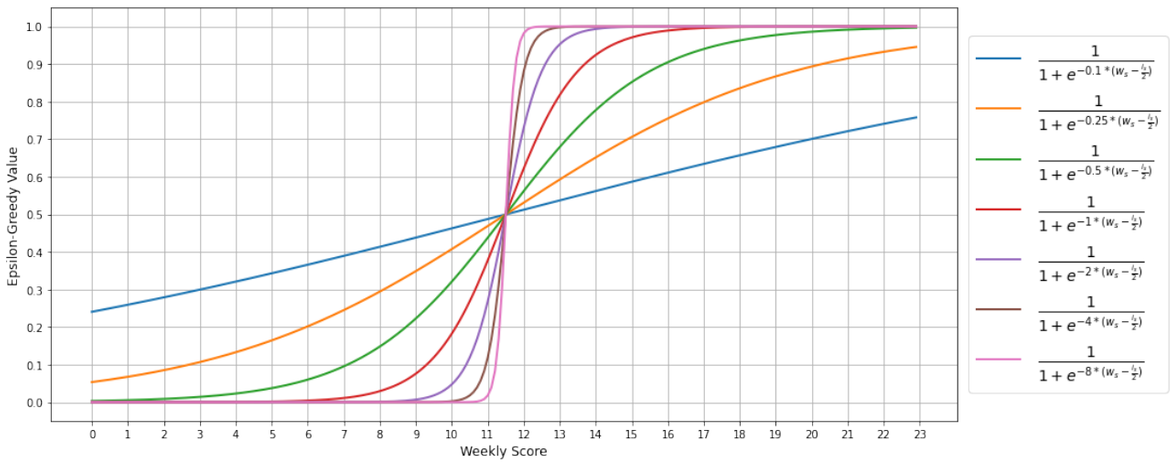

Figure 4: Sigmoid class of functions, with different values for constant c.

The sigmoid class of functions are convex for the lower half, and concave for the upper half of the ideal score. Figure 4 shows how changing changing the value of c impacts the epsilon-greedy value of the agent, with the x-axis representing the score of the agent, and the y-axis representing the corresponding epsilon-greedy value. These findings can be summarized as follows:

- Increasing the value of c from 0.1 results in the functioning becoming less linear in nature, with more agents with poor weekly scores getting 0 for their epsilon-greedy values, and more agents with high weekly scores getting 1 for their epsilon-greedy values. This is due to the nature 'S' shape of the sigmoid curve becoming steeper, and results in more homogenous epsilon-greedy values overall as the value of c is increased, to the point where the lower half of agents ranked by score just explore, and the upper half just exploit.

3.3.4 Average Epsilon-Greedy Values From Each Class of Functions

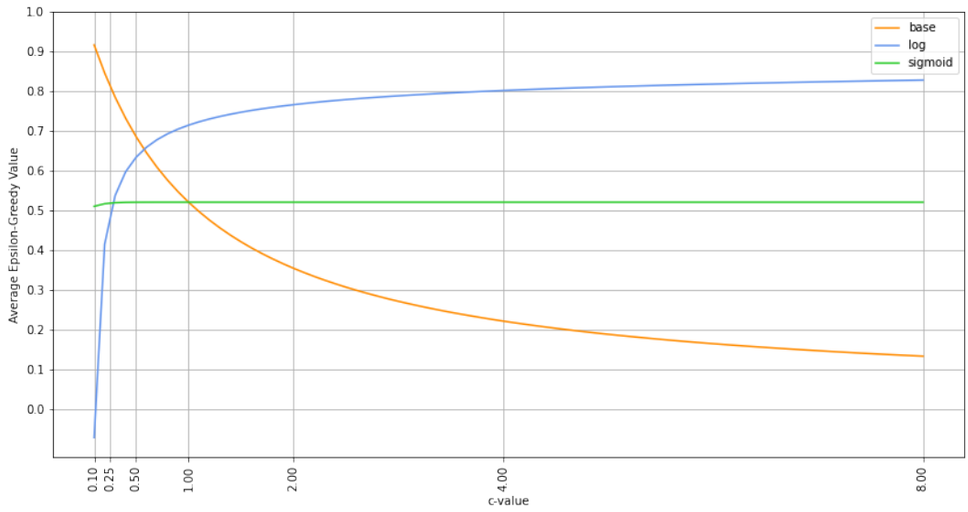

Figure 5: Average epsilon-greedy values for different epsilon normalization functions based on c.

Figure 5 depicts the average epsilon-greedy values for each epsilon normalization function for each value of c. It is worth noting that this may not be representative of the actual average epsilon-greedy value in a model with agents and their different scores, since this average assumes an equal distribution of each score up to the ideal score, and the actual score

distribution of the agents may vary depending on other factors. For example, in the grocery store application, these factors include the number of time slots, the overcrowding threshold, the weekly shopping quota and so on. However, this figure does provide an intuition about how much exploration/exploitation each value of c can result in relative to each other value of c, for each normalization function.

For the base class of functions, it is clear from From Figure 5 that increasing the value of c results in less exploitation and more exploration overall. For the logarithmic class of functions, it shows that increasing the value of c results in more exploitation and less exploration, having the opposite effect to the base class of functions. Due to the nature of the logarithmic curve, this opposite effect is not proportional, that is, the functions aren't inversely proportional to each other for different values of c, however it still paints an interesting picture of the average epsilon-greedy value, and it is worth exploring what value of c is the best performing for this class of functions, relative to the others.

Given the monotonic nature of the sigmoid class of functions, and their property of being convex for values less than a particular point, and concave for values greater than that point, all the sigmoid functions in Figure 4 that have a c value greater than 0.5 have the same average epsilon-greedy value. This is not the case with c values less than 0.5 as those functions never result in an epsilon-greedy value of either 0 or 1. Therefore since their range is smaller they do not have the same average epsilon-greedy value as the other c values.

3.4 Grocery Store Application

3.4.1 Model Setup

The model used for the purposes of testing this research was written in NetLogo [52], which is a popular programming language and environment used by students and researchers for the purposes of modelling multi-agent settings and observing the results . The NetLogo model that was created for the purpose of this research can be found here.

The model acts as a simplification of agents going to a grocery store during a pandemic. Each 'tick' in the NetLogo model corresponds to a time slot. There are 112 total time slots for each week in the model, which are split up into 16 time slots per day, for each hour from 8am to 12am for each day of the week from Monday to Sunday. Every agent is given a shopping quota of 4, meaning they are given 4 slots from all the available 112 slots, and during those slots in the simulation they go to the grocery store.

There are time slot configurations associated with these 4 time slots for each agent, that dictate the number of visits each agent makes to the grocery store. These are as follows:

- 1 visit: All time slots are scheduled consecutively, back-to-back, on a single day.
- 2 visits: Shopping quota split into two halves, with each half scheduled on a separate day, but consecutively on their own day.
- 4 visits: Shopping quota split into four quarters, with each quarter scheduled on a separate day, but consecutively on their own day.

This is further illustrated in Figure 6, where we can see an example of what the 3 different time slot configurations can look like for 3 different agents. It should be noted though that these can, and do overlap in the actual model for different agents. They have just been shown separately here in this figure for the purpose of illustrating the idea.

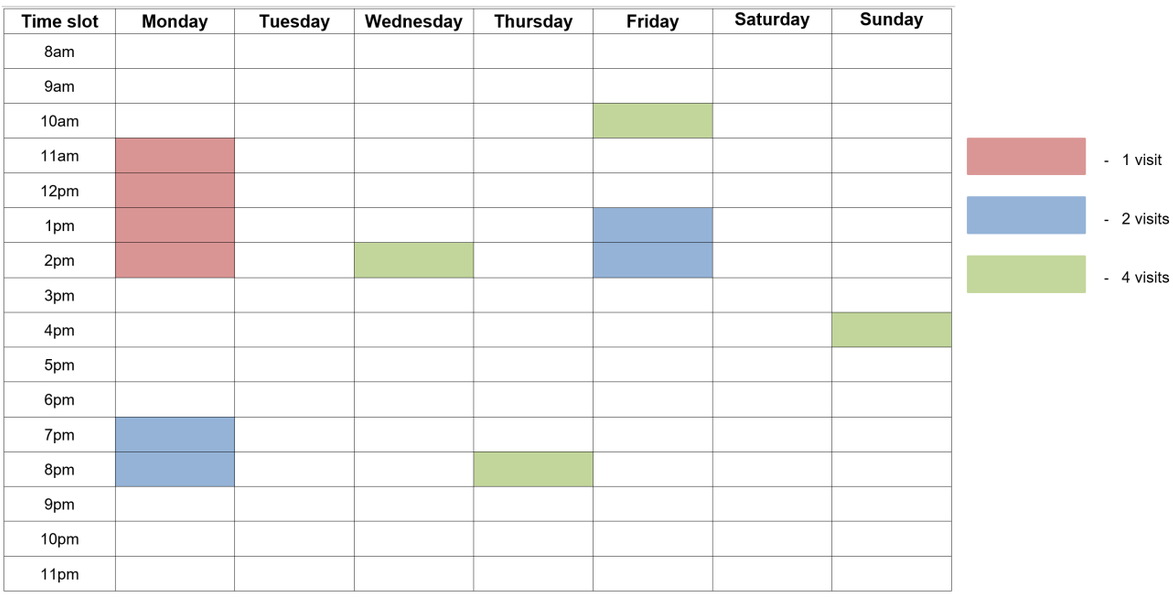

Figure 6: Weekly schedule with different time slot configurations.

Before the first week, the agents are randomly assigned time slots. During each time slot, the attendance is calculated and stored, and at the end of week, this attendance is used to calculate the weekly score for each agent using Equation 3.1. Based on their score, and the ideal score, epsilon-greedy value for each agent are then calculated in the range [0,1] using the different epsilon normalization functions as described in Section 3.3. Agents that perform better in terms of their score, get a higher epsilon-greedy value meaning they are more likely to exploit their current set of chosen time slots, whereas agents that perform worse get a lower epsilon-greedy value, meaning they are more likely to explore other alternative time slots. Functionally, this is implemented as follows: agents pick 2 random numbers, both in the range of [0,1]. Both numbers are then compared against their epsilon-greedy value, and if both are greater than the epsilon-greedy value, then the agents pick a new time slot configuration (or new number of visits) and new time slots at random. If only the first random number is greater than the epsilon-greedy value, then the agents keep the same time slot configuration, but randomly pick new time slots within this configuration, for any day of the week. If none of the random numbers are greater than the epsilon-greedy value, then the agents stick with their current time slots.

The intuition here is that agents with very low epsilon-greedy values are probabilistically encouraged to change not just their time slots, but also their time slot configurations, meaning they explore more from the space of all time slots, whereas the ones with slightly low epsilon-greedy values change only their time slots within their current time slot configuration, and the ones with high epsilon-greedy values change nothing at all, since they scored well and should exploit the low attendance counts in their existing time slots. This process is repeated for each week until all the time slots are under the overcrowding threshold, and the system converges to an optimum, where all the agents are exploiting. The complete process flow of the simulation is described in the ODD in Appendix A.

3.4.2 Model Evaluation

The model developed is evaluated in terms of its success with respect to the research questions. There are many different pillars of evaluation used for this research, with some being inspired by existing literature with similar problems.

Maes [53] conducted general-purpose research that outlines an evaluation strategy for adaptive agents, some of which apply to our problem here. Most relevant is the emphasis on the action selection of the agents, and whether the agents are favoring actions that contribute positively towards achieving their goals. This can be assessed in our research by counting the number of agents that are switching their time slots (and time slot configurations), and checking whether this subsequently results in a reduction in the grocery store attendance,

and eventually being below the overcrowding threshold every week. Another general-purpose evaluation is whether agents improve on the basis of past experience. This can be assessed in terms of the average epsilon-greedy value of all agents. If it is closer to 1, then most of the agents have improved over time and are exploiting their best set of time slots. There is not, however, any mechanism to prevent an agent from not using a previously chosen set of time slots that did not work for them. Giving the agents a memory should, in theory, improve their learning mechanism and result in faster convergence, however that is beyond the scope of this application.

Bell [51] evaluated reinforcement learning agents in an El Farol problem by progressively increasing the number of adaptive agents across various simulations, and noting the effect on the overall system and overcrowding threshold. This, however, is not pertinent to our research since we are assuming all agents are adaptive, and are not concerned with a hybrid setup. Franke [50] also evaluated reinforcement learning agents in an El Farol model, however they assessed the success of the adaptive agents by comparing the actual attendance to the overcrowding threshold over iterations. For our research, this is a strong indicator of agent learning and adaptiveness, and is a metric for evaluation. Chen and Gostoli [54] evaluated their research by counting the number of agents that adopted the optimal strategy. In our system, agents that adopt an optimal strategy are those that have their epsilon-greedy value equal to 1 (for exploitation). Therefore we can count the number of agents with their epsilon-greedy as 1, and divide it by the number of total agents to understand how close the system is to optimum.

Based on previous research as well as research questions outlined for this thesis, the model was evaluated on the following criteria:

- Does the system converge to an optimum under the overcrowding threshold?
  - Is the grocery store attendance below the overcrowding threshold for an entire week?
  - Does each agent's epsilon greedy parameter converge to 1?
- How many weeks does it take for the system to converge to optimum?
- How many agents switch their time slots and time slot configurations over time?
- What class of functions, and c-value is ideal for normalizing the scores of agents in this application?

3.4.3 Data Collection and Analysis

The data for evaluation was collected using the BehaviourSpace tool that comes with NetLogo. The tool allows for running multiple simulations and the subsequent monitoring and collection of certain variables at each tick (which in our case is each time slot). The model was setup with 400 adaptive agents, and one grocery store. Given the 112 different possible slots, and the shopping quota of 4 for every agent, if we were to average out the agents across each time slot, we would have approximately 14.3 agents in each time slot. We set the overcrowding threshold to 17, slightly above this averaged-out attendance value of 14.3, since setting it to 14.3 would result in lower scores overall, and fewer agents achieving the optimal, therefore making the learning slower. Moreover, since we don't give the agents any kind of heuristic for picking time slots, setting the overcrowding threshold that low can result in instances of too many random swaps to other time slots without reaching to any stable solution, with any epsilon normalization function. Given an overcrowding threshold of 17 and the shopping quota of 4, we can use Equation 3.2 to calculate the ideal score, which comes out to be 23.5.

For each of the different epsilon normalization functions, different values of c, in the range of [0.1, 8], (as displayed in Figures 2, 3 and 4) were experimented with. For each of these c values, the model was run for 1000 iterations with a time limit of 4000 ticks. This number of iterations ensured that the the initial distribution for Week 1 averages out to be

the same across all values of c, and that subsequent results are sufficiently averaged, and one-time outliers that might skew comparisons are eliminated. The time limit of 4000 ticks was set based on experimental observations with regards to how many ticks most simulations tended to take to converge. It is also computationally intensive and time consuming to make the simulation time too long, and this resulted in a balanced trade-off that allowed for the simulations to be run and analyzed in a decent time-frame. Furthermore, this time limit is only a maximum cap, the simulations were compared against each other in terms of their own time taken for convergence.

The results were outputted to .csv files, which contain the variables that needed to be tracked for analysis. BehaviourSpace allows the tracking of variables at each time step, which in our case corresponds to a time slot. The following variables were tracked in the results:

Table 1: Model variable names and their meanings

Variable Name	**Variable Meaning**
attendance	The attendance at the grocery store at any given time slot
mean-score	Mean score across all agents
min-all-scores	Minimum score across all agents
max-all-scores	Maximum score across all agents
unique-scores	Number of unique scores, used to estimate gradation of scores
mean-epsilon	Mean epsilon across all agents
min-epsilon	Minimum epsilon across all agents
max-epsilon	Maximum epsilon across all agents
unique-epsilons	Number of unique epsilons, used to estimate gradation of epsilons from normalization function(s)
num-agents-one-roll	Number of agents that switched their time slots this week
num-agents-two-roll	Number of agents that switched their time slots and time slot configurations this week (subset of num-agents-one-roll)
overcrowded-agent	Number of agents that are considered overcrowded, defined here as an agents that has at least one overcrowded time slot
num-overcrowded-timeslots	Number of time slots that are overcrowded
count-swapped-one-roll	Total number of agents that switched their time slots
count-swapped-two-roll	Total number of agents that switched their time slots and time slot configurations

The output data was then processed using python scripts in order to derive more metrics and generate figures that help visualize and interpret the results, as can be seen in the results section. From experimentation, the best value of c for the best epsilon normalization function was chosen for analysis. Since the simulations were run for a 1000 iterations for each c value, all the metrics were averaged over each time-slot to produce an 'average' run. The calculation of these metrics begin from the end of week 1, since the first week is randomly initialized. The number of weeks taken to converge for each c value was calculated by looping over each iteration's attendance, then taking the index of the week after which the attendance in the shop consistently remains below the overcrowded threshold of 17. After this is calculated for each iteration, it is averaged to get the mean time taken for convergence as well as the standard deviation. All the code that was written for the analysis as well as the corresponding graphs and figures can be viewed in the jupyter notebooks here.

The following libraries were used for the data analysis:

- pandas - Pandas dataframes were used to load the data and do data processing.
- seaborn - For constructing plots relevant to the distribution of agent scores and epsilons over the weeks.
- matplotlib - For constructing all plots made using matplotlib in jupyter notebooks.

- statistics - The statistics library was used for calculating the mean, standard deviation, median and mode from different lists.
- scipy - Used to calculate area under the curve for different normalization functions.

4 Results and Discussion

4.1 Summary of Results

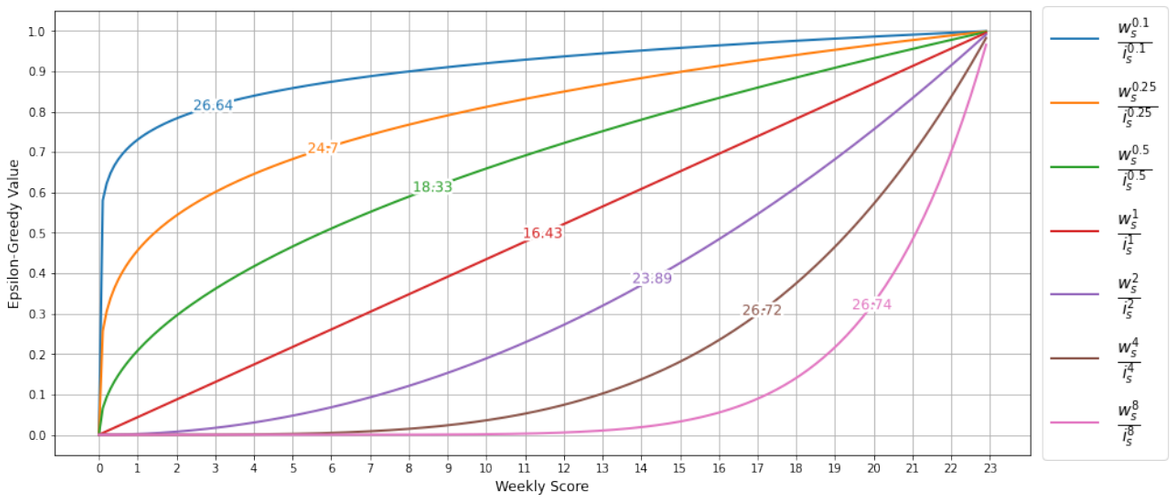

Figure 7: Time taken for convergence in weeks for the base class of functions with varied values of c.

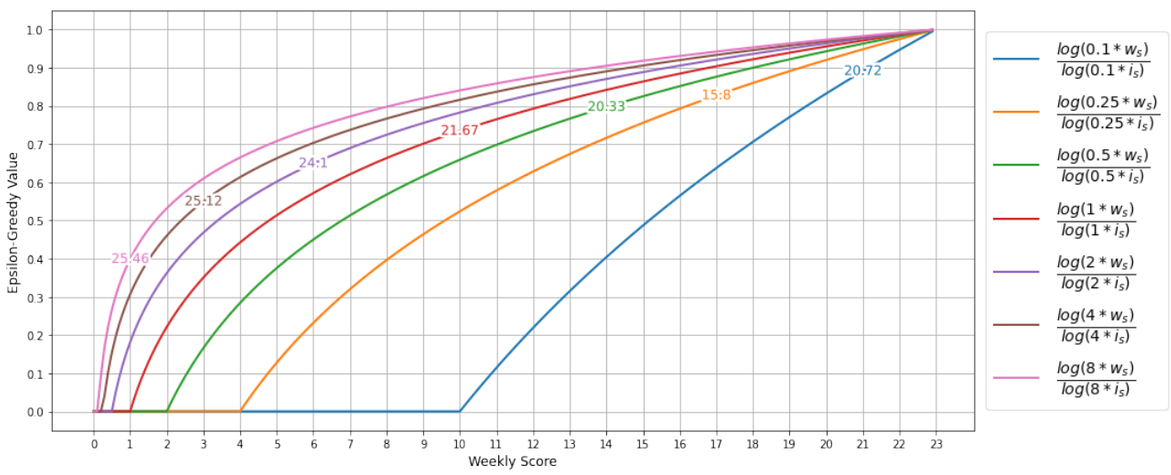

Figure 8: Time taken for convergence in weeks for the logarithmic class of functions with varied values of c.

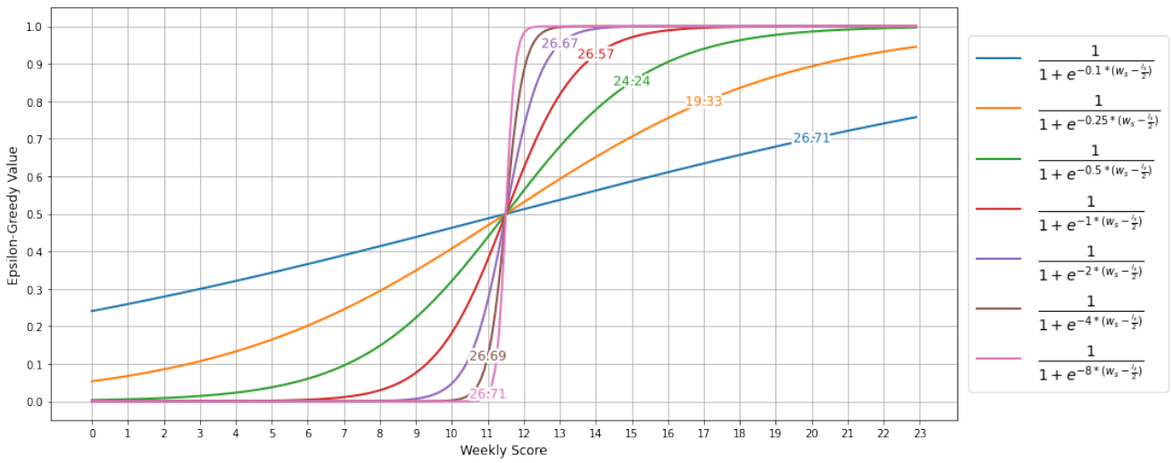

Figure 9: Time taken for convergence in weeks for the sigmoid class of functions with varied values of c.

Figures 7, 8, and 9 show the results for each of the different values of c for all 3 epsilon normalization functions, with the numbers along the lines representing the time taken (in weeks) for each of the curves to converge to an optimum. The best performing for each of these epsilon normalization functions, in terms of the time taken for convergence is summarized in Table 2. Here we can see that all of them have a 100% convergence rate, meaning that all 1000 iterations were able to converge to an optimum in the set time limit of 4000 time slots in the model. We examine the results from the best performing epsilon normalization function, the logarithm function, with it's best performing value of c, 0.25, in the section below.

Table 2: Best Performing values of c for each epsilon normalization function

Function Name	**Convergence Rate**	**Mean number of weeks for convergence**	**Best value of c**
$\frac{w_s^c}{m_s^c}$	100%	Mean = 16.43 Std. deviation = 5.6	1
$\frac{log(c * w_s)}{log(c * m_s)}$	100%	Mean = 15.8 Std. deviation = 5.7	0.25
$\frac{1}{1 + e^{-c(w_s - \frac{m_s}{2})}}$	100%	Mean = 19.33 Std. deviation = 7	0.25

4.1.1 Best Performing Function

In terms of the shortest time taken for convergence, the logarithm function with c value of 0.25 results in the best balance of exploration and exploitation relative to the agents' scores. Figure 5 showed that the average epsilon-greedy value for this function is 0.48. This signifies that given an even and equal distribution of scores in the range of [0, ideal-score], there is likely to be almost the same amount of exploration and exploitation among the entire set of agents. However, this may differ in practice, since there is not necessarily an even and equal distribution of scores each week. Thus, in order to understand how this fares in practice, and how much exploration and exploitation is actually carried out, it is important to review the actual distribution of the weekly scores from the experiments.

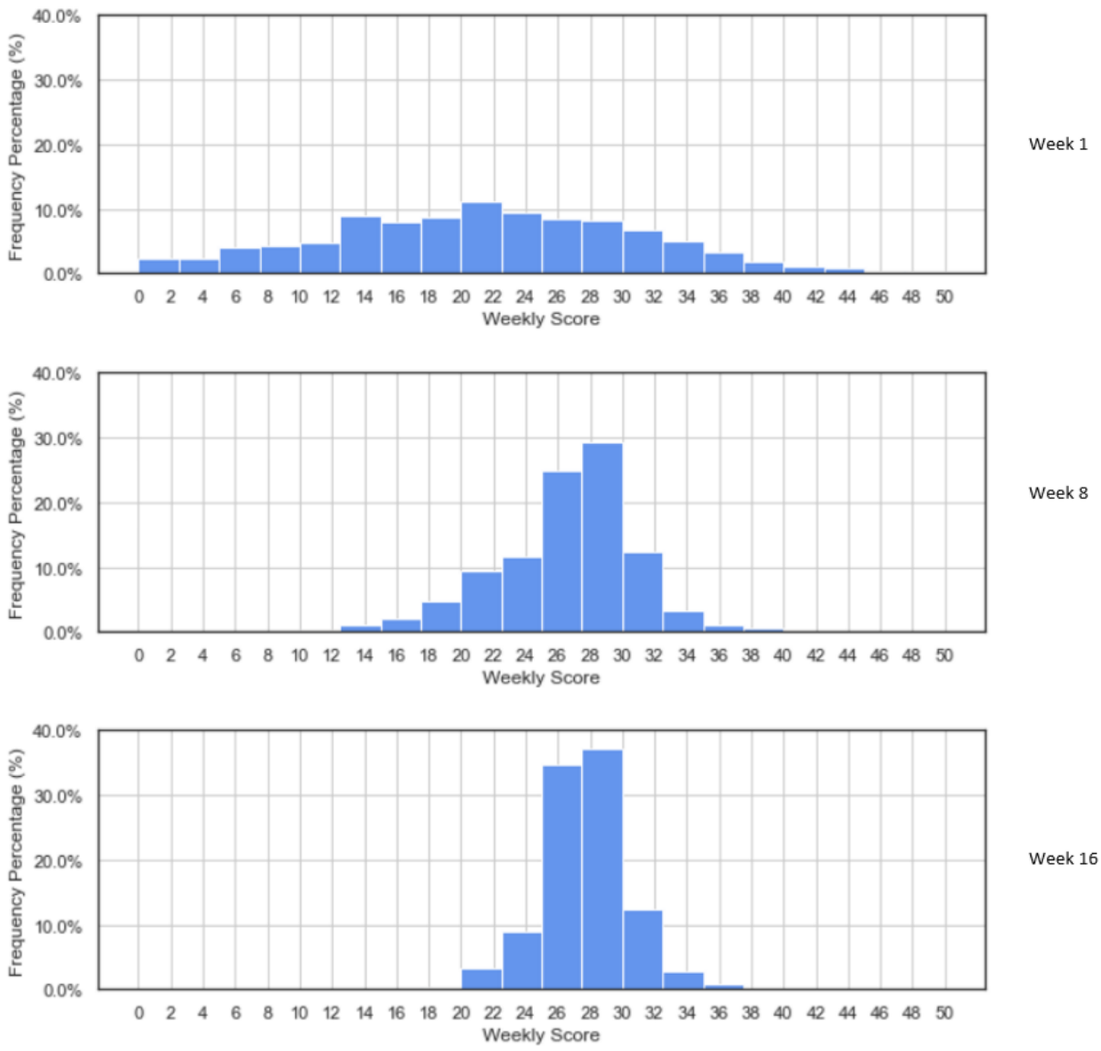

Figure 10: Distribution of weekly scores over time.

As can be seen from Figure 10 and Table 6 (see Appendix B), the scores in week 1 are not evenly distributed in the range of [0, ideal-score], and a lot of agents have scores that exceed the ideal-score of 23.5, due to the uneven distribution of agents in time slots. This distribution of scores in week 1 has a very flat distribution, with a mean of 21.51 and a high standard deviation of 9.74. As the weeks progress and the agents explore and change their time slots, the agents start to get distributed into time slots more evenly and as a result get higher scores relative to the ideal score of 23.5, as can be seen from the week 8 figure in Figure 10. At this point the mean score has already crossed the ideal score of 23.5, however the system has not converged yet. By Week 16, it can be seen that due to the time slots being distributed more evenly, the distribution of scores converges to a mean of 27.67 with a standard deviation of 2.58, resulting in a symmetrical distribution.

In order to fully understand the impact of these changes in the distribution of scores each week, it is important to look at the subsequent impact it has on the distribution of the epsilon-greedy values each week, since they both influence each other's changes in distributions.

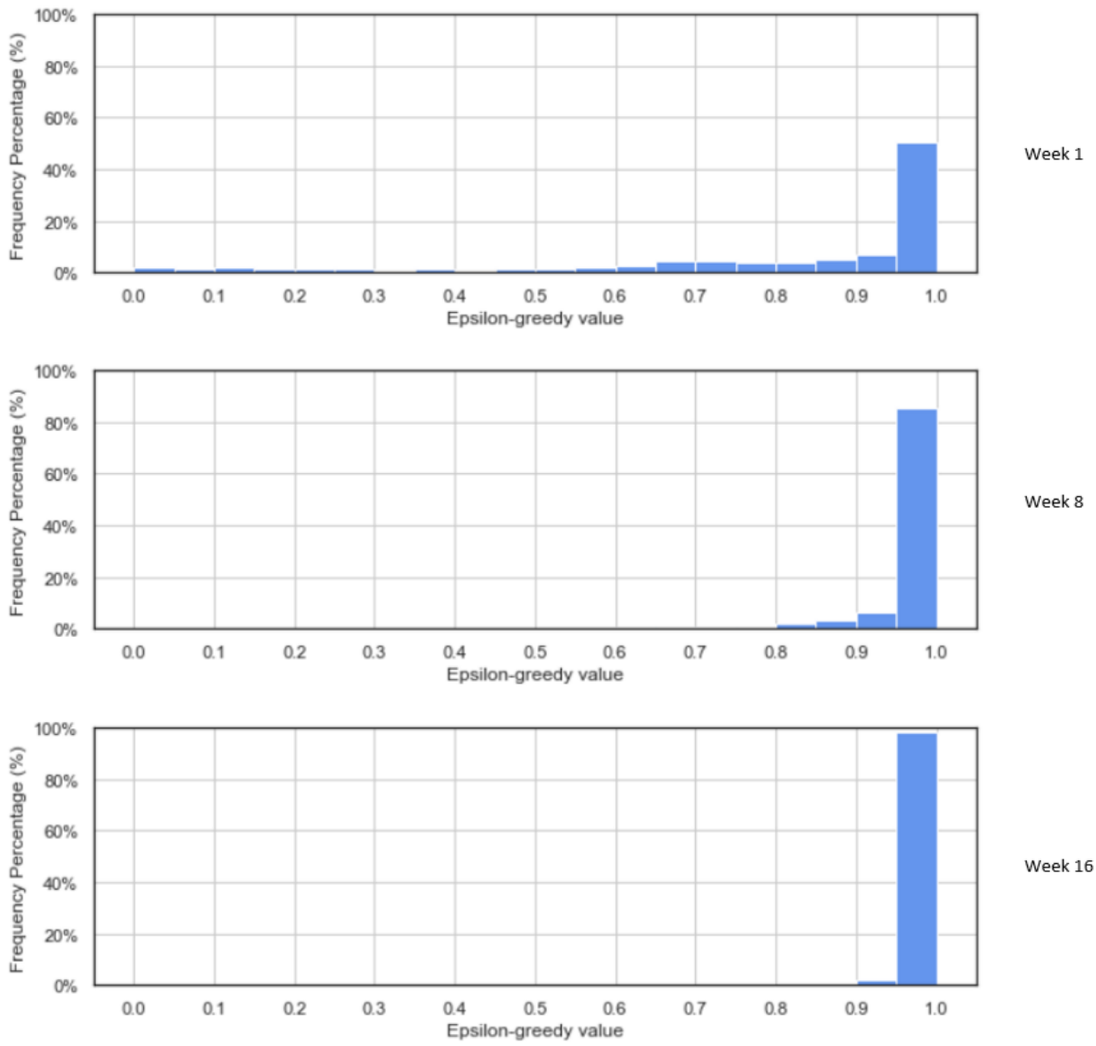

Figure 11: Distribution of weekly epsilon-greedy values over time.

Figure 11 shows how the distribution of epsilon-greedy values change over time, with the key metrics for each week in Table 7 (see Appendix B). The logarithm class of functions with a c value of 0.25 results in a very high mean epsilon-greedy value of 0.81 in week 1, which would result in more exploitation overall. Towards weeks 8 and 16 it can be seen that more agents get assigned an epsilon-greedy value of 1, owing to more agents achieving scores closer to the ideal-score, and the system converging to an optimum.

It is worth noting that since some agents get more than the ideal score, which, as discussed in the methodology section, is the score on average that an agent would have in an optimal system, the distribution of epsilon-greedy values skews towards having a mean closer to 1. This is due to an epsilon-greedy value equal to 1 being assigned to agents having a weekly score higher than the ideal score.

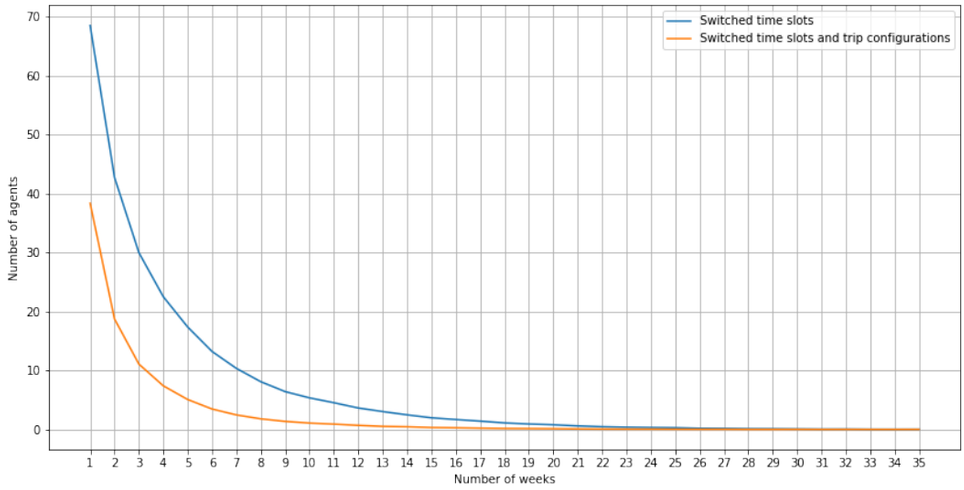

Figure 12: Average number of agents switching time slots over each week.

The distributions of scores and epsilon-greedy values do not tell us how many agents actually explore and exploit in practice in the simulations. Figure 12 shows how many agents change their time slots, as well as time slots and time slot configurations of each week of the simulation. Here it can be seen that 68 agents switch their time slots within their time slot configuration by the end of the first week, which is 17% of the population of agents, while 38 of those same agents swapped both their time slots and time slot configurations, which makes up 9.5% of the total population. Week 1 is also when we see the most number of agents switching time slots, with both values decreasing in the subsequent weeks. This is due to the agents' adaptiveness, and shows how the system as a whole is improving over time because more agents are getting higher scores (as can be seen from Figure 10) and thus exploiting more, needing to swap their time slots less.

It is also important to look at how much exploration and exploitation is carried out in total over all the weeks on average:

- Average count of agents that swapped time slots at the end of the simulations: 221.6
- Average count of agents that swapped time slots and time slot configurations at the end of the simulations: 114.6

From these values, we know that, on average, 55.4% swapped at least their time slots over the entire run of the simulations, meaning little over half do some kind of exploration, and 28.7% switch both the time slots and time slot configurations.

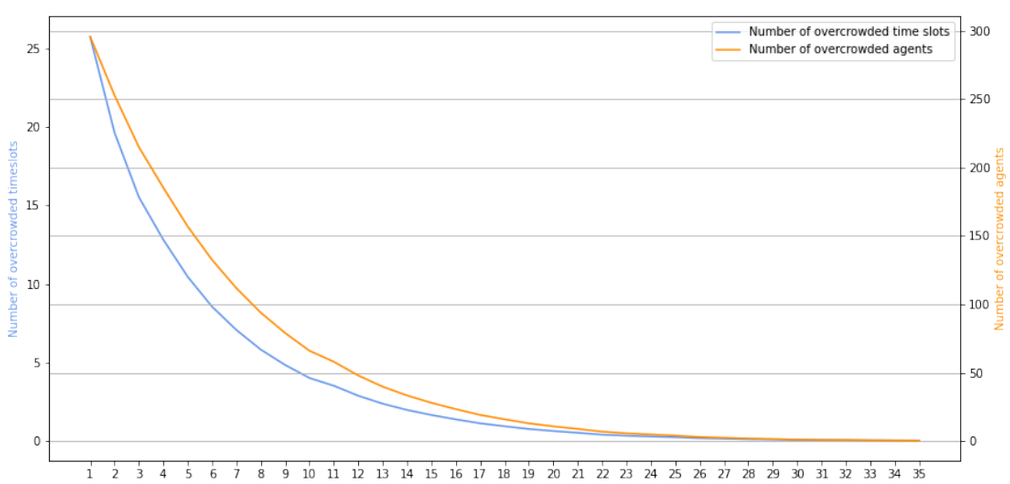

Figure 13: Average number of overcrowded agents and time slots over each week.

While it is useful to understand how much exploration and exploitation is happening, it is also important to understand just how much overcrowdedness is present in the initialization itself. Figure 13 tells us how many agents and time slots are overcrowded. An overcrowded agent here is defined as an agent that has at least one overcrowded time slot. There are initially 295 overcrowded agents, out of which 68 agents, or 23.05%, change time slots with this particular epsilon normalization function. This means that only 23.05% of agents that are classified as overcrowded actually explore instead of exploit. This disparity might be due to how we define overcrowded agents here, as well as being potentially due to many agents having only 1 out of their shopping quota of 4 time slots overcrowded that still score high because their non-overcrowded time slots make up for the 1 that is overcrowded in terms of their weekly score. The number of overcrowded agents decreases sharply in the subsequent weeks, as the system converges to optimum.

The average number of overcrowded timeslots mirrors the pattern of the average number of overcrowded agents, with the first week being the most overcrowded week, with approximately 25 time slots out of 112 (22.3%) being overcrowded. While this seems like a small number of time slots to be overcrowded in the initialization, during a pandemic, 22.3% of timeslots being overcrowded can be potentially very dangerous. It can result in a lot of infectious cases within a week, which given an R0 value greater than 1, will only result in the continual spread of the virus if the situation is not contained.

4.2 Analysis and Discussion

Each of the 3 best performing epsilon normalization functions has a 100% convergence rate, meaning that the agents were able to adapt the policy of the overcrowdedness threshold, and result in an optimal system. Whether the quickest time taken for this to happen is acceptable in the real world, is dependent on the goals with respect to the spread of the virus and balancing the economic consequences. For example, 15.8 weeks without hard limits on grocery store attendance, with the expectation that people distribute themselves evenly into the time slots within an overcrowding threshold based on their understanding of social distancing might be too long - and can result in a rapid increase in the spread of the virus. This, however, boils down to the R0 value of the virus. A low R0 value might make it acceptable, if the spread can be contained within a certain threshold. It might also be necessary to actually wait for people to adapt than put in hard restrictions in order to limit the negative impact on the economy, therefore these decisions are contingent on the priority of the policy makers.

4.2.1 Differences in Function Properties

When comparing the shape of each of the 3 epsilon normalization functions, it appears that they are very close to each other in terms of their shape and the corresponding epsilon-greedy values they output for each weekly score, as can be seen from Figure 14. In order to statistically compute their similarity, we computed the area under the curve for each of these functions, the results of which are summarized in Table 3.

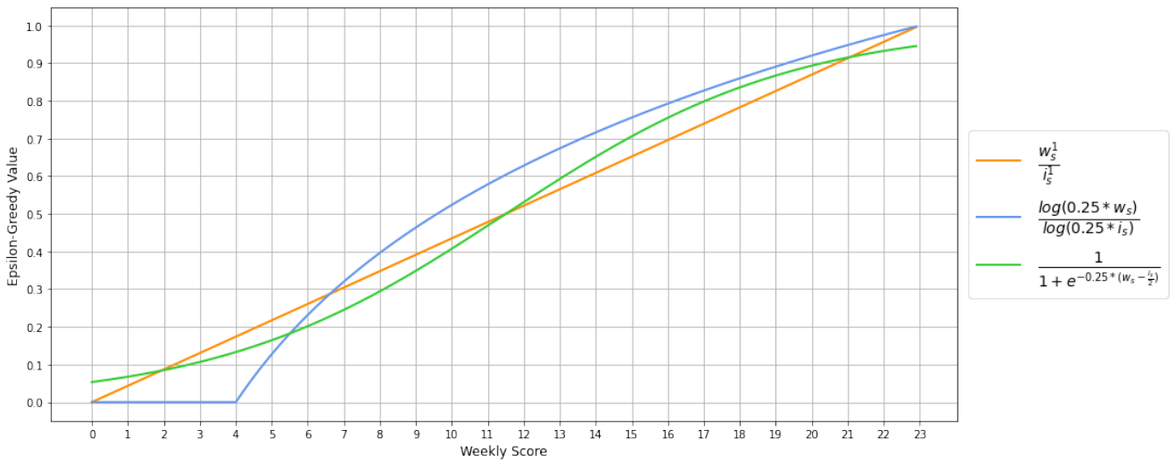

Figure 14: Different classes of epsilon normalization functions with their best performing values of c.

The area under the curve for the sigmoid and base class of functions with their best performing values of c are identical, whereas there is a difference of an area of 0.57, or 4.83% with the logarithm function. From Table 2, we can see that difference in time taken for convergence between the logarithm function and the base function on average is 3.94%, whereas it is 20.1% for the logarithm and sigmoid functions, which is considerably higher. The overall area under the curve, therefore, may not necessarily be indicative of performance. Even though the overall area may be similar, there are key differences in these function graphs at different intervals of weekly scores (the x-axis) that can result in considerable differences in the overall performance.

Table 3: Area under the curve for each of the best performing epsilon normalization functions.

Epsilon Normalization Function	***c***	**Area Under Curve**
Base	1	11.5
Logarithm	0.25	12.07
Sigmoid	0.25	11.5

The difference between the area under the curve between the logarithm function, and the base and sigmoid functions with their best performing values of c, is visualized in Figures 15 and 16. The red area indicates the weekly score range for which the logarithm function does more exploration than the other two, and the green area indicates the weekly score range for which the logarithm function does less exploration and more exploitation than the other two. In general, for lower scores, the logarithm function does more exploration, and for higher scores it does more exploitation, which is preferable for the purpose of this application. The larger disparity in time taken for convergence for the sigmoid function could potentially be due to the fact that its epsilon-greedy output is in the range of [0.5, 0.95].

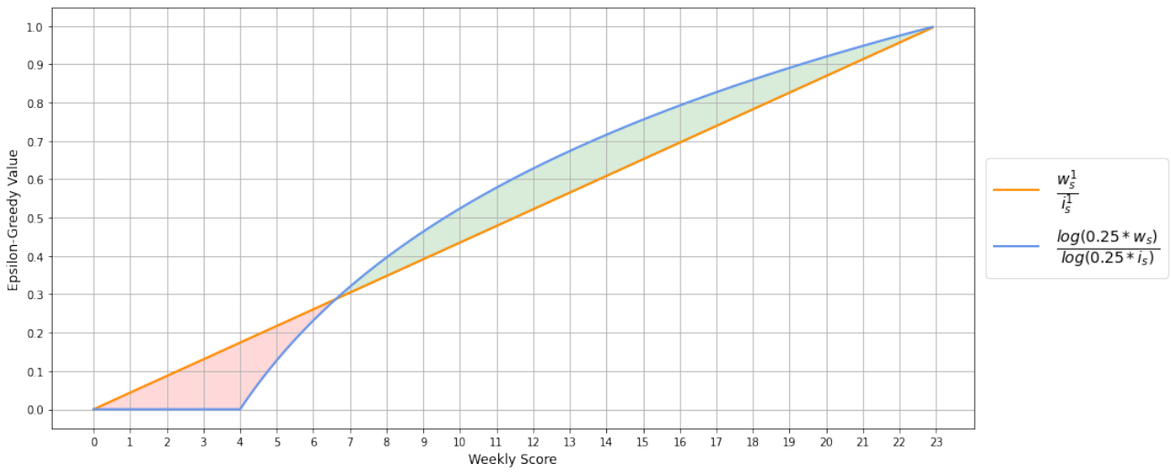

Figure 15: Difference in area under the curve for the best performing logarithm and base functions.

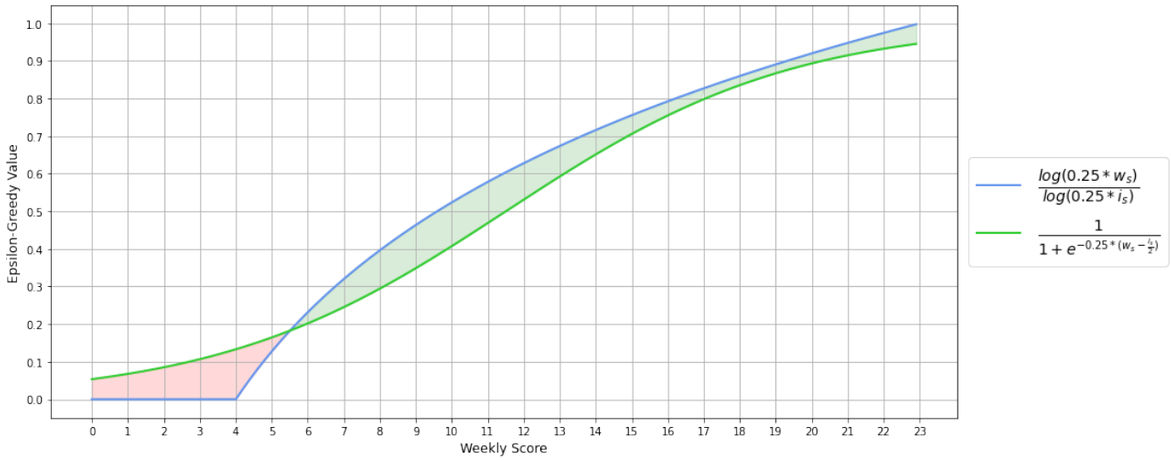

Figure 16: Difference in area under the curve for the best performing logarithm and sigmoid functions.

4.2.2 Results Comparison

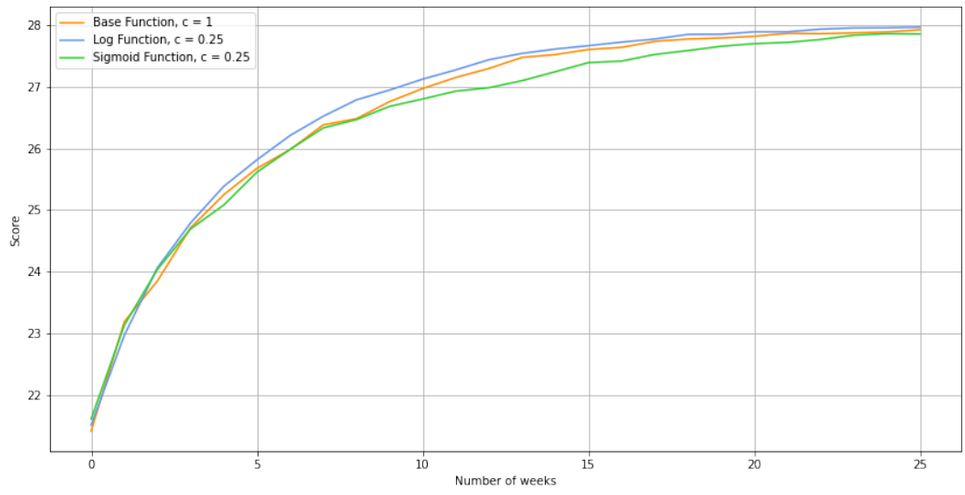

Figure 17: Mean scores averaged over all iterations for all epsilon normalization functions with their best performing values of c.

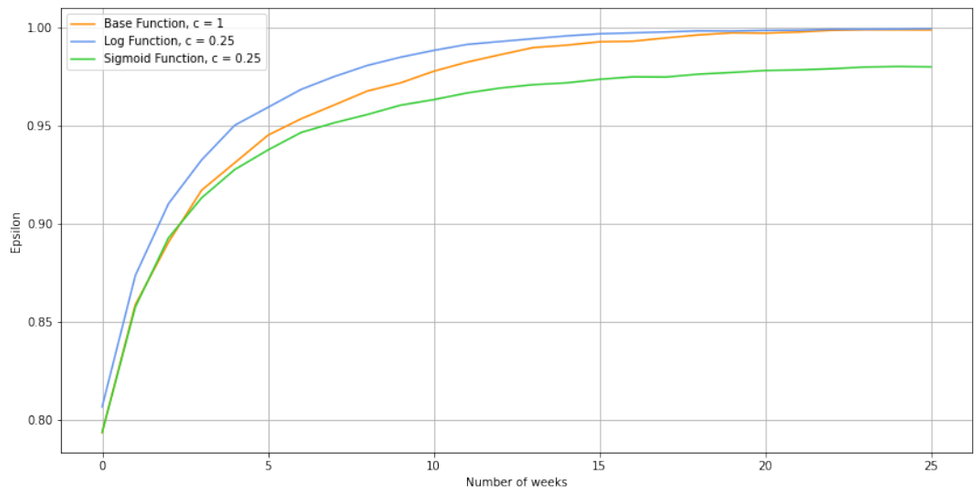

Figure 18: Mean epsilon-greedy values averaged over all iterations for all epsilon normalization functions with their best performing values of c.

The differences between the 3 best performing epsilon normalization function can also be evaluated in terms of the mean scores and mean epsilon-greedy values each week. This is depicted in Figures 17 and 18. From Figure 17 we can see that logarithm function, on average results in agents having a higher mean score than the other 2 functions consistently after the first 2 weeks, however, the differences between all 3 are very little in terms of scores. If we examine Figure 18, the logarithm function, potentially owing to the higher mean scores, as well as the overall higher epsilon-greedy values for agents that perform better than the base and sigmoid function as seen in Figures 15 and 16, results in a higher mean epsilon-greedy value for agents on average consistently for all weeks. Figure 18 also tells us that the sigmoid function does more exploration overall than the other two functions in practice. We can see a comparison of how much exploration was done on average by all 3 epsilon normalization functions with their best performing values of c in Figure 19.

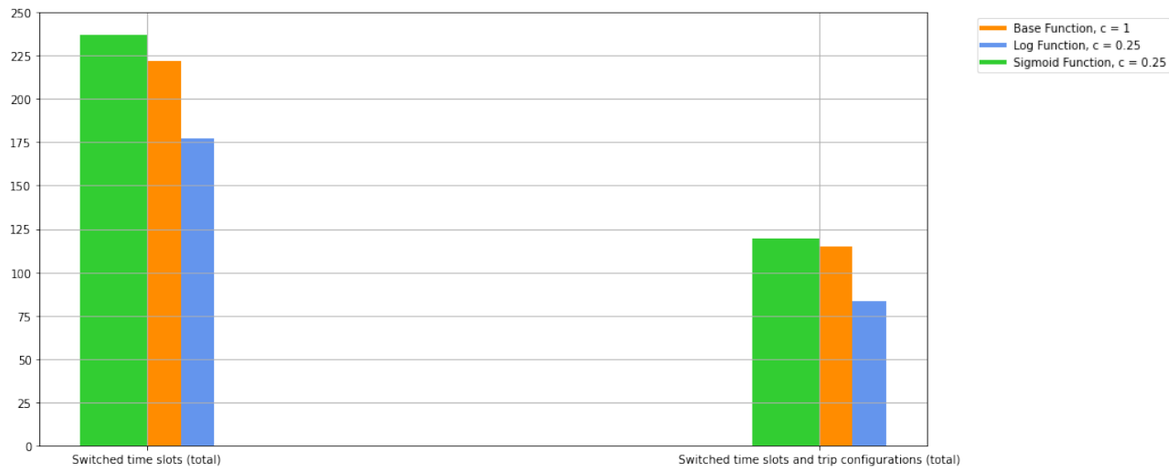

Figure 19: Comparison of total number of agents switching time slots at the end of the simulation for each class of epsilon normalization functions with their best performing values of c.

As expected, Figure 19 shows that the sigmoid functions does indeed result in the most amount of exploration, with the most number of agents switching their time slots and time slot configurations. We can also conclude that less exploration is better in this case, as the logarithm function with c of 0.25 has the least amount of switched time slots and time slot configurations, and performs the best across the three epsilon normalization functions. This could possible mean that that if there is too much exploration and agents switching slots, there is too much entropy or change in the system and it takes longer for convergence, whereas if there is too little exploration, nothing will change in the system and there will

always be overcrowdedness. The logarithm function here thus finds a perfect balance of the two that results in convergence in the least amount of time taken in weeks.

As discussed previously, the mean epsilon-greedy values in Figure 5 do not match up with the actual mean epsilon-greedy values in practice, due to the difference in weekly score distributions. As can be seen from Figure 18, the mean epsilon for all 3 epsilon normalization functions is much higher from the first week itself, and only continues to increase thereon. This is due to the way the scoring function is crafted for this particular application. Therefore, when applying adaptiveness to agents in this manner to other applications, it should be noted that the actual mean epsilon-greedy value from a particular epsilon normalization function, and the subsequent exploration and exploitation that it results in might differ based on the way the scoring function is crafted. The mean epsilon-greedy values in Figure 5 thus only give a relative intuition of what to expect with respect to each other function, but in practice these values can differ.

The scoring functions of other applications might also result in different initialization conditions that are closer or farther away from their optimum than the grocery store application was from its optimum. Therefore, depending on the initialization condition, the amount of exploration and exploitation may vary. For example, if in an application the initialized system is far away from the optimum, more exploration may be required by the agents in the system to converge to an optimum. Thus the right balance needs to be found through experimentation with different scoring functions or epsilon normalization functions.

5 Conclusion

In this research, we were able to demonstrate the ability of adaptive agents to converge to an optimal system using a dynamic epsilon-greedy value. This experiment can thus help us study how to better implement policy changes around people's adaptiveness, in order to achieve a specific goal such as keeping people under crowdedness thresholds to stop the spread of the virus, without hard restrictions or limits, in order to reduce the negative impact on the economy. With the addition of adaptiveness as a property to agents, and finding the subsequent time for the system to converge, we can understand how long it would take for people to adapt to policy changes and therefore make better numerical estimations about the spread of the virus. With respect to the grocery store context, these policy changes can include the opening and closing hours, times of the day during which items should be restocked, overall inventory management, as well as staff and employee management.

The dynamic epsilon-greedy value mechanism used in this paper is agnostic to the kind of model setup that is used in applications. It can be employed in other multi-agent learning applications without the need for any kind of sophisticated learning. This, however, is dependent on a few factors. For any model that is a zero-sum game, it is important to know beforehand what the ideal system looks like, and what the goal of the application is. This guides the crafting of the scoring or utility function, and also provides an ideal relative to which the epsilon normalization is done. Therefore, it is not applicable to models where the classes of problems require finding what an ideal or optimal system is. If the model is not a zero-sum game, the epsilon normalization can be done potentially relative to the highest score achieved by an agent. We did not, however, test this for the grocery store application.

In terms of the least amount of time taken for convergence for the grocery store application, the logarithm function with a c value of 0.25 was the best epsilon normalization function. Different applications may exhibit different results, and this is dependent on the distribution of the weekly score, which depends on how the scoring function is crafted. Since for our use case in the grocery store application the scoring function resulted in an uneven distribution of weekly scores, it is not possible to estimate the best normalization function and best value of c for other applications with different scoring functions. Therefore it has to be applied per application and experimented with, since there is no universal optimal balance for exploration and exploitation.

Furthermore, the rate of convergence achieved by a particular epsilon normalization function for an application may not mirror the same rate of convergence to an optimal by people in the real world. In fact, it is possible that people may never achieve an optimum at all due to various social factors that we cover in the section below. Therefore, in order to use the results from this study in the real world, the policy changes in the real world must be implemented while simultaneously studying the rate of convergence in adaptive agent models to make an estimation about people's adaptiveness, and thus finding out the appropriate epsilon normalization function for the model using real-world data.

5.1 Limitations and Future Improvements

One aspect that the El Farol model includes for its agents that we did not consider for the grocery store application was that of giving agents a memory of the previous time slots they have had, and also how crowded it was during those time slots. Adding a memory for each agent, with a configurable sliding window for the number of weeks to have this memory for is something that could potentially reduce the time taken for convergence. Furthermore, instead of randomly switching all their time slots if agents end up having to explore, we could have them only change the time slots that are overcrowded.

There are also aspects that would make the agents more closely resemble the behaviors of real people. On the whole, the grocery store application considered in this research serves as a simplification of the real world. This is further exacerbated by the fact that the agent population in this application is homogenous, with each agent having the same weekly shopping quota, same normalization function used for determining their epsilon-greedy value, and an open schedule during which they can switch to any time slot if they are to explore.

In the future, adding agent heterogeneity with varied shopping quotas, a hybrid of different epsilon normalization functions for each agent (with some not being adaptive at all), and different availabilities in their schedules for grocery shopping can make the application more accurately depict a real-world scenario.

The grocery store application model was also small in scale, with only 400 agents and 1 grocery store. In reality, multiple heterogenous agents with multiple grocery stores may exist within the same block. Furthermore, a shopping quota of 4 time slots per week for each agent might be unrealistic, however, in the future it might be more realistic to make the time slots themselves shorter to 15 minutes, with variable shopping quotas.

Dynamic and non-stationary environments are hard to model, and when supplemented with completely random, non-heuristic based action choosing, can result in the agents in the model constantly switching time slots without end since it is difficult to find a point to converge on. Therefore, for other applications it might be worth encoding the agents with a heuristic or intuition about what actions to potentially choose if they are to explore.

6 Appendix A

6.1 ODD

6.1.1 Purpose

The purpose of this model is to display how a simple adaptiveness property in agents in a simulation can result in the emergence of an optimized system – without the need for explicit communication and/or coordination with other agents. The case study chosen here is of agents that go to a grocery store during a pandemic and try to avoid crowds by changing their time slots at random, based on intuition about how they are performing relative to the ideal.

6.1.2 Entities, State Variables and Scales

Observer State Variables

Table 4: Observer State Variables

Variable Name	Variable Meaning	Variable Type	Variable Range	Default Value
attendance	The current attendance at the shop	Number (count)		
all-scores	List of all weekly scores of all agents	List		
num-trips	List of different number of trips agents can complete their quota in	List		[1 2 4]
days-of-the-week	List of days of the week	List		[“Monday” “Tuesday” “Wednesday” “Thursday” “Friday” “Saturday” “Sunday]
time-periods	List of time periods of the day represented by hours by default during which the shops are open	List		[8 . . . 23]
periods-of-the-day	Combination of each of the days-of-the-week with the time-periods	List (derived)		[“Monday 8” “Monday 9” . . . “Sunday 22” “Sunday 23”]
current-time-period	The current time period in the simulation	String		
current-time-period-index	Index value of the current time period in the list periods-of-the-day	Number		
home-patches	Agentset of green patches representing the residential area	Agentset		

Continued on next page

Table 4 – continued from previous page

Variable Name	Variable Meaning	Variable Type	Variable Range	Default Value
shop-patches	Agentset of blue patches representing the shop area	Agentset		
crowded-patch	Patch where we show the "CROWDED" label	Patch		
crowded?	Boolean value indicating whether or not the bar is crowded	Boolean	(true, false)	
all-scores	List of all the weekly scores of each agent	List		
min-all-scores	Minimum of all agent scores, resets every week	Number		
max-all-scores	Maximum of all agent scores, resets every week	Number		
mean-score	Average score of all agents, resets every week	Number		
unique-scores	Count of the number of unique scores	Number (count)		
all-epsilons	List of all the weekly epsilon-greedy values of each agent	List		
min-epsilon	Minimum of all agents' epsilon values, resets every week	Number		
max-epsilon	Maximum of all agents' epsilon values, resets every week	Number		
mean-epsilon	Avearge of all agents' epsilon values, resets every week	Number		
unique-epsilons	Count of the number of unique epsilons	Number (count)		
sum-epsilon	Sum of all agents' epsilon values	Number	(0 to n), n = number of agents	
exploitation-count	Count of all agents that are exploiting	Number (count)		

Continued on next page

Table 4 – continued from previous page

Variable Name	Variable Meaning	Variable Type	Variable Range	Default Value
num-agents-one-roll	Number of agents that roll the dice and have a roll greater than their own epsilon-greedy value, resets every week	Number (count)	(0 to n), n = number of agents	
num-agents-two-roll	Number of agents that roll the dice twice and have rolls greater than their own epsilon-greedy value both times, resets every week	Number (count)	(0 to n), n = number of agents	
count-swapped-one-roll	Total number of agents that roll the dice and have a roll greater than their own epsilon-greedy value	Number (count)	(0 to n), n = number of agents	
count-swapped-two-roll	Total number of agents that roll the dice twice and have rolls greater than their own epsilon-greedy value both times	Number (count)	(0 to n), n = number of agents	
setup-flag	Boolean variable used to check if the initial setup is complete	Boolean	(true, false)	
overcrowded-agent	Number of agents with at least one overcrowded timeslot	Number (count)	(0 to n), n = number of agents	
factor	'c' value that is used for the epsilon normalization functions	Number		1
min-score	Minimum possible score agents can achieve	Number		0
ideal-score	Ideal score	Number		23.5
adaptive?	Switch that controls turning the adaptive property of agents on/off	Switch	(On, Off)	On

Continued on next page

Table 4 – continued from previous page

Variable Name	Variable Meaning	Variable Type	Variable Range	Default Value
overcrowding-threshold	The overcrowding threshold value over which a shop is considered overcrowded. An optimal system would have this value as (number of agents * shopping quota) / number of timeslots.	Number		17
num-agents	Number of agents in the simulation	Number (count)		400

Agent State Variables

Table 5: Agent State Variables.

Variable Name	Variable Meaning	Variable Type	Variable Range	Default Value
chosen-num-trips	Chosen number of trips for the week	Number	(1, 2 or 4)	
chosen-timeslots	Chosen timeslots for the week, of length shopping-quota	List		
shopping-quota	Shopping quota for the week, in number of slots	Number		4
epsilon-greedy	Epsilon-greedy parameter that determines each agent's exploration/exploitation	Floating-point number	(0 to 1)	0
weekly-score	Each agent's score for the week	Number	(0 to max-score)	
at-shop?	Boolean indicating whether or not the agent is at the shop	Boolean	(true, false)	
overcrowded-timeslots	List of timeslots during which the store was overcrowded	List		
home-patch	Designated home patch	Patch		
changed-timeslots-one-roll	Boolean that is set to true if an agent has ever switched timeslots	Boolean	(true, false)	false

Continued on next page

Table 5 – continued from previous page

Variable Name	Variable Meaning	Variable Type	Variable Range	Default Value
changed-timeslots-two-roll	Boolean that is set to true if an agent has even swtiched both timeslots and total trips	Boolean	(true, false)	false

Scales Each tick in the model represents a time slot of the week (with a total of 112), during which the agents go to the shop. At the end of each week, or 112 time slots, the same process repeats. The simulation itself shows an environment with all the agents, and an area that represents the shop. When an agent has the current tick or time slot in their own list of *chosen-timeslots*, they move over to the shop area, otherwise they meander around the outside area.

6.1.3 Process Overview and Scheduling

- The model checks if the *current-time-period-index* is greater than 111, which basically checks if the previous week (112 timeslots) has elapsed. If it has, the model does the following:
  - Resets *current-time-period-index back* to 0.
  - Sets *num-overcrowded-timeslots* to the value in *num-overcrowded-timeslots-temp*.
  - Resets *num-overcrowded-timeslots-temp* back to 0.
  - Resets *overcrowded-agent* back to 0.
  - Asks all agents if they have an overcrowded timeslot in their timeslots. If so, the model increments *overcrowded-agent* accordingly.
  - Re-initializes list of *all-scores*.
  - Re-initializes list of *all-epsilons*.
  - Asks all agents for their weekly scores and adds them to the *all-scores* list.
  - Calculates the *mean-score*.
  - Calculates the *min-all-scores*.
  - Calculates the *max-all-scores*.
  - Calculates the number of *unique-scores*.
  - Asks all agents (if the *adaptive?* Is 'On') to set their *epsilon-greedy* based on the selected *epsilon-function* and *factor*, and then adds this value to *all-epsilons*. Resets the agents' *weekly-score* back to 0.
  - Calculates the *mean-epsilon*.
  - Calculates the *min-epsilon*.
  - Calculates the *max-epsilon*.
  - Calculates the number of *unique-epsilon*.
  - Asks all agents to scale their color based on their *epsilon-greedy* value
  - Resets the *exploitation-count* back to 0.
  - Counts the number of agents that have an *epsilon-greedy* greater than or equal to 1 in *exploitation-count*.
  - Calculates *sum-epsilon*.
  - Executes the "update-strategies-112slots" submodel.
  - Calculates *count-swapped-one-roll*.
  - Calculates *count-swapped-two-roll*.

- The model checks if the *current-time-period-index* is equal to 0, which means that either the simulation is running for the first time, or it is the beginning of a new week. The model then does the following:
  - Creates a temporary list *attendance-counts* of length 112 (the number of timeslots), and sets all the values to 0.
  - Loops over all the *periods-of-the-day* and asks all agents if they have each of those *periods-of-the-day* in their timeslots, if so it accordingly increments the *attendance-counts* list.
  - Resets all agents' *overcrowded-timeslots* back to 0.
  - Calculates the *mean-attendance*.
- The model checks if the *current-time-period-index* is less than or equal to 111, which basically means that the current week is still ongoing. The model then does the following:
  - Resets the *attendance* back to 0.
  - Sets the *current-time-period*.
  - Sets the label patch to an empty string.
  - Asks all agents to set their *at-shop?* to false.
  - Asks all agents if they have the *current-time-period* in one of their *chosen-timeslots*. If so, each agent that does is moved to a random empty patch in the shop, the *attendance* of the shop is incremented by 1, and the *at-shop?* of this agent is set to true. If not, each agent stays at home.
  - Sets *crowded?* to false.
  - Sets the *attendance* to the number of agents currently on crowded patches.
  - If *attendance* is greater than the *overcrowding-threshold*, the model show the text "CROWDED" and sets *crowded?* to true.
  - Asks all agents if they're at the shop. If they are and *crowded?* is true, *current-time-period* is added to the agents' *overcrowded-timeslots*. Additionally adds their reward to their total *weekly-score* using the function reward = ((1/*attendance*) * 100).
  - Increments the *time-period-index* by 1.

6.1.4 Initialization

- *setup-flag* is set to true.
- The following global variables are initialized to 0:
  - *mean-score*
  - *min-all-scores*
  - *max-all-scores*
  - *mean-attendance*
  - *overcrowded-agent*
  - *current-time-period-index*
  - *num-agents-one-roll*
  - *num-agents-one-roll*
  - *count-swapped-one-roll*
  - *count-swapped-two-roll*
- *num-trips* is initialized with its default value.
- *days-of-the-week* is initialized with its default value.
- *time-periods* is initialized with its default value.

- *periods-of-the-day* is initialized as a combination of *days-of-the-week* and *time-periods*.
- Default agent shape is set to "person".
- *all-scores* is initialized as an empty list.
- *all-epsilons* is initialized as an empty list.
- *home-patches* and *bar-patches* are set with their respective colors.
- Agents are created of count *num-agents*. They are initialized with the following variables:
  - *at-shop?* set to false.
  - *weekly-score* set to 0.
  - *shopping-quota* is set.
  - *epsilon-greedy* set to 0.
  - *chosen-num-trips* is randomly initialized to one of *num-trips*.
  - *overcrowded-timeslots* is initialized as an empty list.
  - *changed-timeslots-one-roll* is set to false.
  - *changed-timeslots-two-roll* is set to false.
- The "update-strategies-112slots" submodel is executed.
- *setup-flag* is set to false.
- Ticks are reset.

6.1.5 Submodels

There is one submodel that runs as a part of the main model, and that is the "update-strategies-112slots" function. It runs as follows:

- Resets *num-agents-one-roll* and *num-agents-two-roll* back to 0
- Asks all agents if their *overcrowded-timeslots* are non-empty or if the global variable *setup-flag* is true. If true, for each agent:
  - Temporary variable *days-roll* is set to a random float value from 0 to 1
  - Temporary variable *timeslots-roll* is set to a random float value from 0 to 1
  - If *timeslots-roll* is greater than or equal to the agent's *epsilon-greedy* value, the following happens:
    - *num-agents-one-roll* is incremented by one.
    - If *days-roll* is also greater than or equal to the agent's *epsilon-greedy*, *num-agents-two-roll* is also incremented by one. The agent's *chosen-num-trips* is also randomly selected again.
    - If *chosen-num-trips* is 1, the model chooses a day and randomly assigns the agent 4 consecutive slots on a random day
    - If *chosen-num-trips* is 2, the model chooses 2 separate random days, and assigns the agent 2 consecutive slots on each of those days
    - *chosen-num-trips* is 4, the model chooses 4 separate random days, and assigns the agent 4 random slots on each of those days

6.1.6 Design Concepts

Basic Principles The basic principle of this model is to demonstrate the real-world adaptiveness property of people who may change their behavior to avoid crowds. It is loosely based on the El Farol Bar problem and adapted here for the grocery store context during a pandemic. There is agent-level learning that happens with the epsilon-greedy valye that is unique to each agent, which is determined by the relative weekly-score to the ideal-score, therefore having an aspect of system level learning as well.

Emergence The behavior that emerges from the model is that all time slots have a shop attendance below the overcrowding threshold, and all agents learn the best time slot for them without going at a crowded time slot, (with the average epsilon-greedy value converging to 1 for exploitation).

The mechanism that supports this behavior is the random switching of time slots by the agents based on their epsilon-greedy value, with the number of agents switching slots also converges to 0 once the system finds the best configuration.

Adaptation In terms of adaptation, the major decisions that the agents make are whether or not they should change their time slots, and/or change their time slot configurations based on their relative weekly-score to other agents. Based on their epsilon-greedy value, the agents have the ability to choose among alternative time slots that are not the ones that they have already chosen that week, as well as completely change up the number visits they take from either 1, 2 or 4.

Interaction There is no communication between the agents of any kind in this model, however there is mediated interaction in the model, owing to the impact that each agent has on each other by being present or absent from the grocery store, which affects the *weekly-score* of each agent. The reason for this is simply owing to the real-world scenario that this model draws inspiration from – in the real world, people don't communicate their plans to strangers when shopping, and people can tell when a shop is crowded or empty (with certain amounts of variation for each person not explicitly modelled here), especially when there are social distancing protocols in place.

Stochasticity There are many elements of stochasticity in the model. Firstly, the initial time slot configurations and subsequent time slots for each agent are randomly assigned. Whenever an agent changes their time slot configurations and/or time slots after each week (based on their *epsilon-greedy* value), they are randomly assigned new ones. This randomness is included in order to show how even simple agents without a memory can lead to a system that converges to optimum.

Collectives There is an emergent collective of agents here that learn the optimal configuration for going to the grocery store. This collective emerges entirely from agent behaviours, and the relative impact of each agent's score on each other agent.

Observation The information from these simulations is collected using NetLogo's BehaviourSpace tool. The following variables are kept track of:

- *attendance* – to track the attendance over time.
- *mean-score* - to track how the mean score changes over time.
- *min-all-scores* - to track how the smallest score changes over time.
- *max-all-scores* - to track how the largest score changes over time.
- *unique-scores* - to track the number of unique scores over time.
- *mean-epsilon* - to track how the mean epsilon changes over time.
- *min-epsilon* - to track how the smallest epsilon changes over time.
- *max-epsilon* - to track how the largest epsilon changes over time.
- *unique-epsilons* - to track the number of unique epsilons over time.
- *num-agents-one-roll* – to track the weekly count of the number of agents that change their time slots.
- *num-agents-two-roll* – to track the weekly count of the number of agents that change their time slots and their total trips.

- *overcrowded-agent* – to track the number of agents with at least one overcrowded time slot, which is a weekly count of every agent with an *overcrowded-timeslots* list of length greater than 0.
- *num-overcrowded-timeslots* - to track the number of time slots that are overcrowded each week.
- *count-swapped-one-roll* - to track the total count of the number of agents that change their time slots.
- *count-swapped-two-roll* - to track the total count of the number of agents that change their time slots and their total trips.

These variables are then averaged over the number of simulations that are run to produce understand how the model is performing on average.

7 Appendix B

Table 6: Distribution of weekly scores for logarithm function with c value of 0.25

Week number	**Mean and standard deviation**	**Median**	**Mode**
1	• Mean = 21.51 • Std. deviation = 9.74	21.51	20
2	• Mean = 22.97 • Std. deviation = 7.94	23.09	21
3	• Mean = 24.06 • Std. deviation = 6.81	24.67	27
4	• Mean = 24.79 • Std. deviation = 6.13	25.77	27
5	• Mean = 25.38 • Std. deviation = 5.5	26.44	28
6	• Mean = 25.81 • Std. deviation = 5.04	26.76	28
7	• Mean = 26.21 • Std. deviation = 4.62	27.03	28
8	• Mean = 26.52 • Std. deviation = 4.22	27.22	28
9	• Mean = 26.79 • Std. deviation = 3.9	27.35	28

Continued on next page

Table 6 – continued from previous page

Week number	Mean and standard deviation	Median	Mode
10	• Mean = 26.95 • Std. deviation = 3.68	27.43	28
11	• Mean = 27.12 • Std. deviation = 3.42	27.48	28
12	• Mean = 27.27 • Std. deviation = 3.18	27.53	28
13	• Mean = 27.44 • Std. deviation = 2.97	27.58	27
14	• Mean = 27.54 • Std. deviation = 2.81	27.62	27
15	• Mean = 27.61 • Std. deviation = 2.7	27.65	27
16	• Mean = 27.67 • Std. deviation = 2.58	27.65	28

Table 7: Distribution of weekly epsilon-greedy values for logarithm function with c value of 0.25

Week number	Mean and standard deviation	Median	Mode
1	• Mean = 0.81 • Std. deviation = 0.27	0.95	1.0
2	• Mean = 0.87 • Std. deviation = 0.21	0.99	1.0

Continued on next page

Table 7 – continued from previous page

Week number	Mean and standard deviation	Median	Mode
3	• Mean = 0.91 • Std. deviation = 0.17	1.0	1.0
4	• Mean = 0.93 • Std. deviation = 0.14	1.0	1.0
5	• Mean = 0.95 • Std. deviation = 0.11	1.0	1.0
6	• Mean = 0.96 • Std. deviation = 0.1	1.0	1.0
7	• Mean = 0.97 • Std. deviation = 0.08	1.0	1.0
8	• Mean = 0.97 • Std. deviation = 0.07	1.0	1.0
9	• Mean = 0.98 • Std. deviation = 0.05	1.0	1.0
10	• Mean = 0.98 • Std. deviation = 0.05	1.0	1.0
11	• Mean = 0.99 • Std. deviation = 0.04	1.0	1.0
12	• Mean = 0.99 • Std. deviation = 0.03	1.0	1.0

Continued on next page

Table 7 – continued from previous page

Week number	Mean and standard deviation	Median	Mode
13	• Mean = 0.99 • Std. deviation = 0.03	1.0	1.0
14	• Mean = 0.99 • Std. deviation = 0.02	1.0	1.0
15	• Mean = 1.0 • Std. deviation = 0.02	1.0	1.0
16	• Mean = 1.0 • Std. deviation = 0.01	1.0	1.0

8 References

[1] Jasper Verschuur, Elco E. Koks, and Jim W. Hall. "Global economic impacts of COVID-19 lockdown measures stand out in high-frequency shipping data". en. In: *PLOS ONE* 16.4 (Apr. 2021). Ed. by Bing Xue, e0248818. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0248818. URL: https://dx.plos.org/10.1371/journal.pone.0248818 (visited on 06/19/2022).

[2] Abdulla M. Alsharhan. "Survey of Agent-Based Simulations for Modelling COVID-19 Pandemic". en. In: *Advances in Science, Technology and Engineering Systems Journal* 6.2 (Mar. 2021), pp. 439–447. ISSN: 24156698, 24156698. DOI: 10.25046/aj060250. URL: https://astesj.com/v06/i02/p50/ (visited on 06/19/2022).

[3] Shahrizan Jamaludin et al. "COVID-19 exit strategy: Transitioning towards a new normal". en. In: *Annals of Medicine and Surgery* 59 (Nov. 2020), pp. 165–170. ISSN: 20490801. DOI: 10.1016/j.amsu.2020.09.046. URL: https://linkinghub.elsevier.com/retrieve/pii/S2049080120303514 (visited on 06/19/2022).

[4] Jonatan Gomez et al. "INFEKTA—An agent-based model for transmission of infectious diseases: The COVID-19 case in Bogotá, Colombia". en. In: *PLOS ONE* 16.2 (Feb. 2021). Ed. by Rebecca Lee Smith, e0245787. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0245787. URL: https://dx.plos.org/10.1371/journal.pone.0245787 (visited on 06/19/2022).

[5] David Alvarez Castro and Alistair Ford. "3D Agent-Based Model of Pedestrian Movements for Simulating COVID-19 Transmission in University Students". en. In: *ISPRS International Journal of Geo-Information* 10.8 (July 2021), p. 509. ISSN: 2220-9964. DOI: 10.3390/ijgi10080509. URL: https://www.mdpi.com/2220-9964/10/8/509 (visited on 06/19/2022).

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[7] Uri Wilensky and William Rand. *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo*. MIT Press, 2015. ISBN: 978-0-262-32812-8. URL: https://ieeexplore-ieee-org.ezproxy.lib.rmit.edu.au/book/7109293.

[8] Mohammad Delasay, Aditya Jain, and Subodha Kumar. "Impacts of the COVID-19 pandemic on grocery retail operations: An analytical model". In: *Production and Operations Management* (2021). Publisher: Wiley Online Library.

[9] Volker Grimm et al. "A standard protocol for describing individual-based and agent-based models". en. In: *Ecological Modelling* 198.1-2 (Sept. 2006), pp. 115–126. ISSN: 03043800. DOI: 10.1016/j.ecolmodel.2006.04.023. URL: https://linkinghub.elsevier.com/retrieve/pii/S0304380006002043 (visited on 06/19/2022).

[10] Liviu Panait and Sean Luke. "Cooperative Multi-Agent Learning: The State of the Art". en. In: *Autonomous Agents and Multi-Agent Systems* 11.3 (Nov. 2005), pp. 387–434. ISSN: 1387-2532, 1573-7454. DOI: 10.1007/s10458-005-2631-2. URL: http://link.springer.com/10.1007/s10458-005-2631-2 (visited on 05/02/2021).

[11] Neil F Johnson et al. "Application of multi-agent games to the prediction of financial time series". en. In: *Physica A* (2001), p. 6.

[12] Kam-Chuen Jim and C. Lee Giles. "Talking Helps: Evolving Communicating Agents for the Predator-Prey Pursuit Problem". en. In: *Artificial Life* 6.3 (July 2000), pp. 237–254. ISSN: 1064-5462, 1530-9185. DOI: 10.1162/106454600568861. URL: https://direct.mit.edu/artl/article/6/3/237-254/2350 (visited on 02/26/2022).

[13] Yongcan Cao et al. "An Overview of Recent Progress in the Study of Distributed Multi-Agent Coordination". en. In: *IEEE Transactions on Industrial Informatics* 9.1 (Feb. 2013), pp. 427–438. ISSN: 1551-3203, 1941-0050. DOI: 10.1109/TII.2012.2219061. URL: http://ieeexplore.ieee.org/document/6303906/ (visited on 02/26/2022).

[14] Veysel Gazi and Barış Fidan. "Coordination and Control of Multi-agent Dynamic Systems: Models and Approaches". In: *Swarm Robotics*. Ed. by Erol Şahin, William M. Spears, and Alan F. T. Winfield. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 71–102. ISBN: 978-3-540-71541-2.

[15] Cecilia Garcia Cena et al. “A cooperative multi-agent robotics system: Design and modelling”. en. In: *Expert Systems with Applications* 40.12 (Sept. 2013), pp. 4737–4748. ISSN: 09574174. DOI: 10.1016/j.eswa.2013.01.048. URL: https://linkinghub.elsevier.com/retrieve/pii/S0957417413000791 (visited on 02/26/2022).

[16] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. “Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving”. en. In: *arXiv:1610.03295 [cs, stat]* (Oct. 2016). arXiv: 1610.03295. URL: http://arxiv.org/abs/1610.03295 (visited on 02/26/2022).

[17] Kay W. Axhausen and ETH Zürich. *The Multi-Agent Transport Simulation MATSim*. en. Ed. by ETH Zürich et al. Ubiquity Press, Aug. 2016. ISBN: 978-1-909188-75-4. DOI: 10.5334/baw. URL: http://www.ubiquitypress.com/site/books/10.5334/baw/ (visited on 02/26/2022).

[18] Lucian Busoniu, Robert Babuska, and Bart De Schutter. “A Comprehensive Survey of Multiagent Reinforcement Learning”. en. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (Mar. 2008), pp. 156–172. ISSN: 1094-6977, 1558-2442. DOI: 10.1109/TSMCC.2007.913919. URL: https://ieeexplore.ieee.org/document/4445757/ (visited on 02/26/2022).

[19] Alain Cardon, Thierry Galinho, and Jean-Philippe Vacher. “Genetic algorithms using multi-objectives in a multi-agent system”. en. In: *Robotics and Autonomous Systems* (2000), p. 12.

[20] Larry Bull. “Coevolutionary Computation: An Introduction”. en. In: (), p. 16.

[21] “Genetic Algorithms and Machine Learning”. en. In: (), p. 5.

[22] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E. Taylor. “A survey and critique of multiagent deep reinforcement learning”. en. In: *Autonomous Agents and Multi-Agent Systems* 33.6 (Nov. 2019), pp. 750–797. ISSN: 1387-2532, 1573-7454. DOI: 10.1007/s10458-019-09421-1. URL: http://link.springer.com/10.1007/s10458-019-09421-1 (visited on 05/02/2021).

[23] Pablo Hernandez-Leal et al. “A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity”. en. In: *arXiv:1707.09183 [cs]* (Mar. 2019). arXiv: 1707.09183. URL: http://arxiv.org/abs/1707.09183 (visited on 05/02/2021).

[24] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. “Multi-Agent Systems: A Survey”. en. In: *IEEE Access* 6 (2018), pp. 28573–28593. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2831228. URL: https://ieeexplore.ieee.org/document/8352646/ (visited on 02/26/2022).

[25] C. H. Yong and R. Miikkulainen. “Coevolution of Role-Based Cooperation in Multiagent Systems”. In: *IEEE Transactions on Autonomous Mental Development* 1.3 (Oct. 2009), pp. 170–186. ISSN: 1943-0612. DOI: 10.1109/TAMD.2009.2037732.

[26] Iker Zamora et al. “Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo”. en. In: *arXiv:1608.05742 [cs]* (Feb. 2017). arXiv: 1608.05742. URL: http://arxiv.org/abs/1608.05742 (visited on 02/26/2022).

[27] Zhen Ni et al. “A reinforcement learning approach for sequential decision-making process of attacks in smart grid”. en. In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. Honolulu, HI: IEEE, Nov. 2017, pp. 1–8. ISBN: 978-1-5386-2726-6. DOI: 10.1109/SSCI.2017.8285291. URL: http://ieeexplore.ieee.org/document/8285291/ (visited on 02/26/2022).

[28] Kleanthis Malialis, Sam Devlin, and Daniel Kudenko. “Resource Abstraction for Reinforcement Learning in Multiagent Congestion Problems”. en. In: *arXiv:1903.05431 [cs]* (Mar. 2019). arXiv: 1903.05431. URL: http://arxiv.org/abs/1903.05431 (visited on 02/26/2022).

[29] Aakash Maroti. “RBED: Reward Based Epsilon Decay”. en. In: *arXiv:1910.13701 [cs]* (Oct. 2019). arXiv: 1910.13701. URL: http://arxiv.org/abs/1910.13701 (visited on 02/26/2022).

[30] Michel Tokic and Günther Palm. "Value-Difference Based Exploration: Adaptive Control between Epsilon-Greedy and Softmax". In: *KI 2011: Advances in Artificial Intelligence*. Ed. by Joscha Bach and Stefan Edelkamp. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 335–346. ISBN: 978-3-642-24455-1.

[31] Nicolò Cesa-Bianchi et al. "Boltzmann Exploration Done Right". en. In: *arXiv:1705.10257 [cs, stat]* (Nov. 2017). arXiv: 1705.10257. URL: http://arxiv.org/abs/1705.10257 (visited on 02/26/2022).

[32] Caroline Claus and Craig Boutilier. "The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems". en. In: (), p. 7.

[33] Huasen Wu, Xueying Guo, and Xin Liu. "Adaptive Exploration-Exploitation Tradeoff for Opportunistic Bandits". en. In: (), p. 9.

[34] Lilia Rejeb, Zahia Guessoum, and Rym M'Hallah. "The exploration-exploitation dilemma for adaptive agents". en. In: (), p. 12.

[35] Dhirendra Singh, Sebastian Sardina, and Lin Padgham. "Integrating Learning into a BDI Agent for Environments with Changing Dynamics". en. In: (), p. 6.

[36] Md. Salman Shamil et al. "An Agent-Based Modeling of COVID-19: Validation, Analysis, and Recommendations". en. In: *Cognitive Computation* (Feb. 2021). ISSN: 1866-9956, 1866-9964. DOI: 10.1007/s12559-020-09801-w. URL: http://link.springer.com/10.1007/s12559-020-09801-w (visited on 05/02/2021).

[37] Cliff C. Kerr et al. *Covasim: an agent-based model of COVID-19 dynamics and interventions*. en. preprint. Epidemiology, May 2020. DOI: 10.1101/2020.05.10.20097469. URL: http://medrxiv.org/lookup/doi/10.1101/2020.05.10.20097469 (visited on 05/02/2021).

[38] Nicolas Hoertel et al. *Facing the COVID-19 epidemic in NYC: a stochastic agent-based model of various intervention strategies*. en. preprint. Public and Global Health, Apr. 2020. DOI: 10.1101/2020.04.23.20076885. URL: http://medrxiv.org/lookup/doi/10.1101/2020.04.23.20076885 (visited on 05/02/2021).

[39] Petrônio C.L. Silva et al. "COVID-ABS: An agent-based model of COVID-19 epidemic to simulate health and economic effects of social distancing interventions". en. In: *Chaos, Solitons & Fractals* 139 (Oct. 2020), p. 110088. ISSN: 09600779. DOI: 10.1016/j.chaos.2020.110088. URL: https://linkinghub.elsevier.com/retrieve/pii/S0960077920304859 (visited on 03/21/2021).

[40] Chiara Reno et al. "Forecasting COVID-19-Associated Hospitalizations under Different Levels of Social Distancing in Lombardy and Emilia-Romagna, Northern Italy: Results from an Extended SEIR Compartmental Model". en. In: *Journal of Clinical Medicine* 9.5 (May 2020), p. 1492. ISSN: 2077-0383. DOI: 10.3390/jcm9051492. URL: https://www.mdpi.com/2077-0383/9/5/1492 (visited on 05/02/2021).

[41] Luis Miralles-Pechuán et al. "A Deep Q-learning/genetic Algorithms Based Novel Methodology For Optimizing Covid-19 Pandemic Government Actions". en. In: *arXiv:2005.07656 [physics, stat]* (May 2020). arXiv: 2005.07656. URL: http://arxiv.org/abs/2005.07656 (visited on 05/02/2021).

[42] Akihiro Nishi et al. "Network interventions for managing the COVID-19 pandemic and sustaining economy". en. In: *Proceedings of the National Academy of Sciences* 117.48 (Dec. 2020), pp. 30285–30294. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.2014297117. URL: http://www.pnas.org/lookup/doi/10.1073/pnas.2014297117 (visited on 05/02/2021).

[43] Robert A. Shumsky et al. "Retail store customer flow and COVID-19 transmission". en. In: *Proceedings of the National Academy of Sciences* 118.11 (Mar. 2021), e2019225118. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.2019225118. URL: http://www.pnas.org/lookup/doi/10.1073/pnas.2019225118 (visited on 05/02/2021).

[44] Fabian Ying and Neave O'Clery. "Modelling COVID-19 transmission in supermarkets using an agent-based model". en. In: *PLOS ONE* 16.4 (Apr. 2021). Ed. by Martin Chtolongo Simuunza, e0249821. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0249821. URL: https://dx.plos.org/10.1371/journal.pone.0249821 (visited on 05/02/2021).

[45] Seth G. Benzell, Avinash Collis, and Christos Nicolaides. “Rationing social contact during the COVID-19 pandemic: Transmission risk and social benefits of US locations”. en. In: *Proceedings of the National Academy of Sciences* 117.26 (June 2020), pp. 14642–14644. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.2008025117. URL: http://www.pnas.org/lookup/doi/10.1073/pnas.2008025117 (visited on 05/02/2021).

[46] William Brian Arthur. *The El Farol Bar Problem*. URL: http://tuvalu.santafe.edu/~wbarthur/elfarol.htm.

[47] R. Selten and W. Güth. “Equilibrium Point Selection in a Class of Market Entry Games”. In: *Games, Economic Dynamics, and Time Series Analysis*. Ed. by M. Deistler, E. Fürst, and G. Schwödiauer. Heidelberg: Physica-Verlag HD, 1982, pp. 101–116. ISBN: 978-3-662-41533-7.

[48] Mark Garofalo. “Modeling the ‘El Farol Bar Problem’ in NetLogo”. en. In: (), p. 6.

[49] Duncan Whitehead. “The El Farol Bar Problem Revisited: Reinforcement Learning in a Potential Game”. en. In: (), p. 32.

[50] Reiner Franke. “Reinforcement learning in the El Farol model”. en. In: (2003), p. 22.

[51] Ann Maria Bell. “Reinforcement Learning in a Nonstationary Environment: The El Farol Problem”. In: (1999).

[52] Seth Tisue and Uri Wilensky. “Netlogo: A simple environment for modeling complexity”. In: *International conference on complex systems*. Vol. 21. Boston, MA, 2004, pp. 16–21.

[53] Pattie Maes. “Modeling Adaptive Autonomous Agents”. In: *Artificial Life* 1.1_2 (Oct. 1993), pp. 135–162. ISSN: 1064-5462. DOI: 10.1162/artl.1993.1.1_2.135. URL: https://doi.org/10.1162/artl.1993.1.1_2.135 (visited on 06/19/2022).

[54] Shu-Heng Chen and Umberto Gostoli. “Agent-based modeling of the el farol bar problem”. In: *Simulation in Computational Finance and Economics: Tools and Emerging Applications*. IGI Global, 2013, pp. 359–377.