Behavior Model Recovery in Agent-Based Environments

A Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment
of the requirements for the degree

Doctor of Philosophy

by

Roy Lee Hayes

December 2018

APPROVAL SHEET

This Dissertation
is submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Author Signature: 

This Dissertation has been read and approved by the examining committee:

Advisor: Peter Beling

Committee Member: William Scherer

Committee Member: Stephen Adams

Committee Member: Gerard Learmonth

Committee Member: Worthy Martin

Committee Member: _____

Accepted for the School of Engineering and Applied Science:

Craig H. Benson, School of Engineering and Applied Science

December 2018

Abstract

Complexity science is a new field of study that seeks to understand complex systems. The term complex systems refers to a system that is made up of individual components, which have non-linear interactions with each other, resulting in unpredictable, emergent properties. To better understand these systems, researchers use agent-based models that use a bottom-up approach to simulate complex environments. Observed emergent properties are recreated in agent-based models by providing agents with a set of rules and allowing them to interact with other agents and their environment.

While agent-based models have found academic recognition, they have made a negligible impact in affecting policy and operational decisions in government and industry. The author contends that ineffective validation methods result in low acceptance of agent-based model findings. Approximately 90% of agent-based simulations are either not validated or only validated using non-statistical techniques. The simulations that are validated using statistical methods only confirm that the simulation outcomes match historical data, which just proves that the model is one of potentially many that matches historical data. Therefore, there is little basis for decision-makers to have faith that the results of the simulation will accurately reflect the real-world system in previously unseen states.

The research in this dissertation relies on a validation maturity model created by Harmon and Youngblood, which codifies procedures for a convincing validation. The author asserts that it is insufficient to validate that the outcome of the model matches historical outcomes. Instead, practitioners must also empirically derive both agent classes and agent behavior models to certify that the model accurately reflects the real-world system. This dissertation examines the challenges that arise when empirically deriving agent classes and agents' behavior models.

To demonstrate the broad applicability of agent-based models the author presents two models in different regulatory domains. The first model examines regulating High-Frequency Traders (HFT) in the

commodity markets. There is an ongoing debate about the benefits of HFT traders, whose dominant presence in the markets may improve market liquidity but may also increase market volatility. The author finds that HFT's can exasperate extreme volatility events, such as the Flash Crash of 2010. The simulation also suggests potential regulations, such as a minimum quote life, need to balance volatility mitigation and order trade time. The second model looks at the contentious topic of mass shootings and potential regulations. The simulation found that Dianne Feinstein's proposed ban on assault weapons would have had limited effectiveness because it would not have regulated the rate-of-fire of any firearm.

Similar to many other agent-based practitioners, the author's models have academic acceptance but have had little effect on decision-makers, which question the validity of the models. To improve agent-based model validation methods, the author explores the challenges in empirically deriving agent classes. Size-First Hierarchical clustering is introduced, which is a modification to traditional agglomerative hierarchical clustering. Experiments demonstrate that Size-First Hierarchical clustering outperforms several conventional unsupervised clustering methods on integer clusters with loosely defined nuclei.

There are times when it is insufficient to model individuals as a cluster of agents, and thus an explicit behavior model is required. The author presents his M.S. thesis work as a case study in behavior model recovery. The work is presented because it lays out the foundational issues in behavior model recovery. Using the University of Virginia McIntire Hedge Fund Tournament, the author shows that Classification and Regression Trees (CART) can recover trading strategies with a high degree of accuracy. An action feature representation is shown to increase the accuracy of classification trees. Additionally, stepwise linear regression was integrated with traditional regression tree models. However, it was found to be unstable outside of the training dataset.

To model realistic systems, behavior models will need to be recovered over a large state space, where there is only partial observation of real-world actors. Previous research in this area assumes either a large observation window or that the test and training state space are identical. The dissertation

compares Neural Networks, Classification and Regression Trees, and Inverse Reinforcement Learning in terms of their ability to recover behavior models in partially observed Markov environments. CART was found to outperform the other two models, which contradicts several publish findings.

The author's research demonstrates that agent-based modes are effective at assessing potential regulations. To improve the acceptance of agent-based models' findings, researchers should validate both their agent classes and agents' behavior model against their real-world counterparts. Recovering behavior models through observations of experts is a method to both build the agents' behavior model and validate them simultaneously. Additionally, the dissertation findings suggest CART is a fast, robust method for recovering behavior models from observations.

Acknowledgments

If I succeeded, it is because others have helped me along the way. When I was in 6th grade Mrs. Long, a guidance counselor, took an interest in me. Based on my standardized test scores and grades, she felt that I was qualified to be in the gifted in talented (GT) program and fought to get me in. It was the first time someone outside of my family validated that I had potential.

When I was in high school Mrs. Corbin, another counselor, took an interest in me and placed me in several college preparatory programs, which allowed me to take a free trip to the University of Virginia and tour the campus. Mrs. Corbin also found scholarships for me to attend college. Without Mrs. Corbin, I likely would not have applied or attended UVA.

Even before my first year in college, Ms. Vallas who ran the diversity office was already helping me adjust to college life. I attended her summer program for rising high school students, where I took three courses. This experience helped me understand what it took to succeed in college, but it also provided me a social network of support. I will attest that without Ms. Vallas my grades would not have been as good as they were, and I would have struggled in my adjustment from high school to college.

I never intended to go to graduate school until Professor Learmonth gave me the opportunity to perform research under him. From emulations of satellites to aloha protocol simulations, Professor Learmonth helped me learn how to conduct scientifically rigorous studies and present the findings in an understandable way. He also pushed me to apply to the Ph.D. program, which has culminated in this work.

Without the dedication and patience of Professor Beling and Scherer, I would have never finished the Ph.D. program. I will be the first to admit that I am stubborn. Both Professors allowed me to explore my ideas and passions while shepherding me through this process, which admittedly I did not make easy. They provided me with internships at the Commodity Future Trade Commission and work experience with the Consumer Financial Protection Agency. Throughout this experience, they pushed me to think critically about my findings and dive deeper into the research topics.

Then there are my parents who sacrificed more than I can repay to allow me to achieve my goals. My mom drove me to every Tae Kwon Dao practice, basketball game (even though I was terrible), job interview, and even my first date. She stayed on my school work and forced me to practice multiplication tables and handwriting at the kitchen table. Without my mom's dedication to me, I would not have dedicated myself to this work. My dad was just as committed. He is a great role model, even coaching my basketball team (did I mention I was terrible). I remember he would stay up late after getting home from work to help me learn algebra and calculus, to ensure my grades stayed high. Without my dad, I would not have gained the mathematic fundamentals to perform this research.

To all my family and friends all I can say is thank you for forcing me to stand when I wanted to sit. You pushed me when I didn't want to move and challenged me when I tried to take the easy paths. It is because of your help that I succeed.

Table of Contents

Chapter 1: Introduction - Agent-based policy models and their limitations

In 1995, 14 gray wolves were released into Yellowstone National Park [1]. These wolves fed on the elk, whose population size significantly rose without the presence of an apex predator. In the early 2000s, Ripple and Beschta discovered a linear correlation between valley depth and height of plants, such as cottonwood [2]. They hypothesized that the reintroduction of an apex predator deters elk away from areas where they are easily captured, such as a valley floor. With fewer elk in these areas, vegetation could grow back, providing additional food and habitat for other animals. Beschta and Ripple also assert a link between wolves, elk, and the channel dynamics of the Gallatin River Basin. Before the reintroduction of wolves, the elk population grew uncontrollably and consumed 95% of the willow cover, which led to more frequent river bank collapses [3].

Yellow Stone is an example of a complex system, which is characterized by having individual elements that interact with one another in a non-linear manner. Additionally, the aggregate outcome of these interactions is impossible to predict by studying the different components in isolation [4]. Complex systems arise in both the human-made and natural world. An essential characteristic of complex systems is that quantifying the impact of a single element on the rest of the system is difficult. While many scientists believe grey wolves are necessary to Yellowstone ecological balance, many also think their impact is overstated [5].

Emma Marris found that vegetation coverage, such as the willow, is still decreasing in Yellowstone. She asserts elk have not been scared away from their preferred feeding grounds [6]. Facing the certainty of starving or the possibility of being eaten, the elk venture out to feed. Additionally, the Yellowstone's beaver population has declined because elk out-competed them for the same food source, the willow plant [6]. The smaller beaver population resulted in fewer beaver damns. Without the damns

the water table of the national park fell, making a less ideal habitat for the willow plant and ultimately resulting in the continued decline of willow plants. Furthermore, beaver also eat cottonwood, and a decreased beaver population could account for the increased size of the cottonwood plants [7].

In an attempt to better understand complex systems, a new domain called complexity science has emerged, which studies systems with many interconnecting components that create emergent behaviors, which could not be predicted by examining the individual components separately [8], [9]. An example of an emergent behavior is the theory of bird flocking. Historically it was commonly believed that bird flew in the characteristic V-shape because a centralized leadership structure commanded them. However, in 1987 Craig Reynolds [10] was able to simulate realistic bird flocking behavior by having each bird follow a simple set of rules:

1. Separation – keep a minimum distance from your neighbor,
2. Alignment – match the average heading and speed of your neighbors,
3. Cohesion – Head for the center of mass of your neighbors.

By following the simple rules and without communicating, simulated birds can create the emergent behavior of flocking. This phenomenon is not unique to birds and arises in ant colonies, fish, and even human-made environments such as the stock market.

The type of simulation Craig Reynolds created is known as an agent-based model, and they are heavily utilized by complexity scientist. Agent-based models are computational models that allow the simulation practitioner to represent feedback loops. By forcing each agent to follow a set of rules and allowing them to interact with one another, complex behaviors can be created.

Agent-based models are comprised of autonomous entities, which make independent decisions based on private and public information [11]. These entities are known as agents and can take on various roles in a simulation. If we were to model Yellow Stone National Park as an agent-based model, there would be agents that represent wolves, elk, beavers, other animals, and people. Agents interact with their

environment and other agents by making independent decisions based on their own private and public information. The environment in such a model would comprise the terrain of the park (e.g., plants, rivers, valleys, etc.). Repeated interactions amongst agents and their environment cause emergent phenomena, such as the relationship between increased river bank collapsing and a decrease in the wolf population.

Heath et al. found that agent-based models are used in a variety of domains, such as business, economics, public policy, and social science [12]. As opposed to other simulation methods, such as differential equations, agent-based models allow for the modeling of individual agents that follow an explicit rule set [13], making agent-based models an ideal experimental platform for the social systems listed above.

The traditional method of isolating a control group is difficult, if not impossible, for many social systems. However, agent-based models allow for the repeated trials over a wide range of parameters. The findings of agent-based models could be used to inform decisions regarding social systems if their results were believed to be accurate. The problem is that many agent-based models lack sufficient validation, which limits confidence in their findings and has resulted in the slow adoption of agent-based models by decision-makers [14], [15].

Approximately 90% of agent-based simulations are either not validated or only validated using non-statistical techniques [12]. An improperly validated model can lead to confirmation bias, where the accepted model reproduces the researcher's expected outcomes [16]. Confirmation bias is especially dangerous to the new field of complexity science, as agent-based models can become a tool to reinforce a preconceived notion. The next section describes the difference between verification and validation, as well as the challenges of validating agent-based models

Verification and Validation of Agent-Based Models

Defining the difference between verification and validation is essential. Verification is defined as, "ensuring the computer programing of the computerized model and its implementation are corrected [17]." Validation differs from verification because validation focuses on proving both the internal and final outputs of a simulation that accurately reflect a domain. Schlesinger et al. [17] define validation as, "substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model."

To further highlight the difference between verification and validation the author uses a simple conceptualized model of the simulation process, which was presented by Robert Sargent [18]. A simulation practitioner first starts with the system they are trying to model and develops a conceptual model that represents the system. The conceptual model is encoded into a computerized model, which is then run over a set of parameters to infer information about the original system.

Verifying that the computerized model accurately implements the conceptualized model is a straightforward task. However, validating that an agent-based simulation accurately reflects its domain is a challenge because many traditional validation techniques focus solely on the output of the model [18]. For example, a simple validation test that is often used is Degenerate testing, which involves selecting the extreme values for parameters and ensuring the model returns the expected results, such as zero arrivals corresponds to a zero throughput [19].

While proving that a simulation accurately represents extreme circumstances increases confidence in the model, it does not ensure that the model accurately reflects its domain. Therefore, multiple validation techniques are traditionally used. Another common validation technique is historical validation [20]. The initial parameters of the model are set to historical values and the output of the model is compared to the historical results. This method is beneficial for systems that are well known, as it ensures the simulation is accurately representing the system.

Agent-based models are used to simulate highly dynamic systems that are difficult, if not impossible, to understand fully. Therefore, validating only the output of an agent-based model proves that the model is one of potentially many that accurately reflect the system over a set of historical parameters. However, agent-based models are not developed to examine the system under a historical parameter set. Instead, they are used to study how the system will react to changes in the environment or agent behavior. Therefore, to increase confidence in agent-based modeling, practitioners must develop methods to validate that the model will accurately represent the system in previously unseen environments.

Before proceeding further, it is vital to define accuracy, which the author defines as correctly representing the relationship between the parameters of interest and the output of the system. Heath et al. found that there are roughly three categories of simulations [12]:

1. Generators – are simulations that are used to generate a hypothesis about little known complex systems. The goal of these simulations is to produce a hypothesis that can be tested either by observing the real system or in a higher fidelity simulation.
2. Mediators – are simulations that represent an understood system but does not have the fidelity to predict an outcome accurately. These simulations are used to understand better the dynamics of the systems and how its components interact. Additionally, they are also used to create new hypotheses.
3. Predictors – are simulations of a fully understood system. An example is a queue system where people wait in line to be served. Predictor simulations use data collected from the real system to predict values of interest, such as wait time.

Heath et al. found that no agent-based model qualified as a predictor [12]. Therefore, the goal of an agent-based model is to represent how a change in the environment or agent behavior will affect the system

but not predict the exact value. An accurate agent-based model will show if an environmental change is more likely to harm one agent over another but will not quantify the precise effect of the change.

Previous researchers have examined how to validate models in a manner that improve confidence in their results. Naylor and Finger [21] proposed a multi-stage validation process, where the model is first based on a set of articulable assumptions and theories. Where possible, the proposed theories and assumptions are tested empirically using historical data. The final validation check is to ensure over a set of historical data the model accurately reflects the output of the real-world system. Under this framework, both the model's underlying theories and final outputs are validated, which increases the confidence in the model's results.

Youngblood and Harmon [22] expanded on Naylor and Finger's idea and created a simulation validation maturity model. They created five levels of validation and linked each level to how much confidence a decision-maker has in a model validated at that level. The table below describes each level.

Table 1 Validation Maturity Model found in [22]

Level	Model Theory and Validation Criteria Derived From	Judge of Validity	Validation Criteria	Confidence
0	None	None	None	No
1	Subject Matter Expert	Subject Matter Expert	Expert Knowledge of user needs	No
2	Subject Matter Expert	Subject Matter Expert	Explicit specification of needed entities, entity properties, and behaviors together with SME knowledge of users' needs for accuracy	No
3	Derived from a single source or dataset	Independent observer	Explicit specification of needed entities, entity properties, behaviors, minimum needed accuracies, and property ranges over which minimum accuracies can be guaranteed	No
4	Sampled from multiple independent sources and correlated statistically with estimates of uncertainties	Independent observer	Explicit specification of needed entities, entity properties, behaviors, minimum needed accuracies, property ranges over which minimum accuracy can be guaranteed, and desired confidences in validation evidence	Yes
5	Rigorously derived from multiple independent sources and correlated statistically with estimates of uncertainties	Formal Proof	Formal Specification of needed entities, entity properties, behaviors, minimum needed accuracies, property ranges over which minimum needed accuracies, property ranges over which minim accuracies can be guaranteed, and desired confidences in validation evidence	Yes

In addition to traditional historical value validation, a level 5 validation requires that an agent-based model's agents, agents' behavior model, and environmental factors be empirically derived from and validated against multiple independent data sources, which ensures that the model is accurately representing the complex system. However, many agent-based models are not validated pass level 2. For example, the previously described bird flocking model was created by a single expert, who did not derive the model's theory or validation criteria from a formal observation of bird flocking. Therefore, while the model reflects some aspects of bird flocking there is little confidence that birds only follow the previously defined three rules of separation, alignment, and cohesion.

Numerous agent-based models use only a single source of information when deriving and validating their model. Real-world actors can have non-stationary behavior models and using only a single source of data fails to account for this. Therefore, researchers cannot guarantee a simulation accurately reflects a system overtime without validating the model over multiple data sources, representing different time periods. Lastly, the author has not found any agent-based models that have proved their agents' behaviors accurately represent the real-world actors they simulate.

It is important to state that a level 5 validation does not guarantee that a model will accurately represent the system in a previously unseen state. However, it does increase confidence in the results because both the simulation outcomes and the mechanisms that created them have been validated against multiple independent data sources. Therefore, researchers can state that the model likely reflects the complex system.

The hypothesis statement of this dissertation is that to improve the acceptance and validity of agent-based modeling, models must be validated at a level 5 maturity. Specifically, researches must empirically derive the agents and validate that agents' behaviors match the observed real-world actors. The next section discusses each chapter of the dissertation and how it addresses the challenges of empirically deriving both agents and their behavior models.

Challenges Facing Empirical Derivation of Agents and Agent Behaviors

The second chapter of the dissertation focuses on illustrating how agent-based models can inform decision-makers about social systems and potential policies. The author first presents an agent-based model of the E-Mini S&P 500 Futures market, which is used to study how potential regulations will affect the volatility of the market. A second model explores how possible assault weapon regulations would affect an active shooter scenario. While both models were accepted by academics, achieving 62 citations over three papers, they made a negligible impact on real-world decision-makers. The author asserts that

the limited acceptance was because neither model was validated to a level 5 acceptance. Specifically, the author did not empirically derive or validate agent behavior models.

Most agent-based models are made of heterogeneous classes of agents, which are defined by expert knowledge. However, to both increase validity and confidence in agent-based models' results, the classes need to be empirically derived from data. The third chapter addresses the challenge of how to autonomously group agents based into classes based on their behavior models. The problem is that some real-world actors use behavior models that are rarely seen. These seldom seen behavior models can be considered outliers in the dataset. Clustering a dataset with outliers can result in otherwise different entities being grouped [23]. We present Size-First Hierarchical Clustering which is shown to be more resilient to outliers in loosely packed, isotropic datasets.

There are circumstances, such as oligopoly markets, where it insufficient to represent real-world actors as a set of different classes. Oligopolies are characterized by having a small number of participants. Therefore, a slight change in the environment that affects one of the participants can have a significant effect on the market. In these markets, it is essential to reflect the behavior model of the real-world actor accurately. Chapter 4 presents work done for the author's Master's degree that highlights the effectiveness of Classification and Regression Trees (CART) in recovering behavior models from historical data.

Chapter 5 compares Cart to a Neural Network (NN) model and an Inverse Reinforcement Learning (IRL) model in a more realistic scenario. In chapter 5 we examine which machine learning technique is better at recovering behavior models in low observation, large state space environments. By their very nature, complex systems have large state spaces and real-world data that highlight an individual actor's behavior model is likely to be sparse. Therefore, it is important to explore which machine learning technique most accurately recovers the behavior model of an actor under these circumstances. The last chapter summarizes the findings of the dissertation and presents possible future work.

Chapter 2: Agent-Based Policy Models and Their Limitations

Chapter2 highlights two agent-based models that the author developed, which demonstrate agent-based simulations broad applicability to policy making. The goal of this chapter is to demonstrate to the reader that agent-based models can identify relationships between agents and environmental factors. Thereby, allowing regulators to understand the cost-benefit of a proposed regulation and to tailor the regulation to maximize its effect on its intended parameter, while minimizing any adverse effects. The first agent-based model presented in this chapter is a zero-intelligence financial agent-based model. The agent-based model is used to examine potential regulation that limits market volatility and the findings are presented.

A second agent-based model is presented that examines the proposed Dianne Feinstein Assault Weapon Ban and its effect on the number of people killed or wounded in an active shooter scenario. Like the financial agent-based model, the goal of the simulation is to identify the relationship between parameters of interest (e.g., rate-of-fire, accuracy, range, magazine capacity) and outcomes of interest (e.g., the number of people killed or wounded). is tested using the agent-based model. The second agent-based model. The next section presents the zero-intelligence financial agent-based model.

Zero-intelligence Financial Agent-based Model

Financial markets are an ideal environment to apply agent-based models as mediators. Even though the efficient market theory has been increasingly disproved [24], over large time window significantly irrational agents will lose money and exit the market. In other words, if an agent is losing money, they will either change their strategy or go bankrupt. Additionally, algorithmic trading has taken over for human-based trading. Algorithmic trading is commonly defined as the use of computer algorithms to automatically make trading decisions, submit orders, and manage those orders after submission [25].

We developed a 1/32nd agent-based model of the E-Mini S&P 500 futures to examine the Flash Crash and proposed financial policies.

Background on Algorithmic Trading and the Flash Crash

Algorithmic trading is so prevalent that the speed of order submission has emerged as a principal characteristic for distinguishing trading agents. Market participants known as high-frequency traders (HFTs), can trade hundred times a second, using fast (often deterministic) algorithms, as well as, specialized network connections and operation rules with the trading exchanges [26]. HFTs are often orders of magnitude faster in order submission than other trading agents, and even other algorithms [26].

The rise of algorithmic trading has had broad and direct impacts on the financial markets. Concerns about the effects of high-frequency trading on the price discovery process and market price stability were first widely acknowledged after the events of May 6, 2010. During which the world markets were perturbed by one market participant's algorithm that caused a sharp price drop, known as a Flash Crash, in the E-Mini S&P futures market. The action in the E-Mini market spread to other futures and equities market. The overall market fell nearly 6% in minutes, after which it recovered nearly as quickly [27]. Kirilenko et al. showed that the critical events in the Flash Crash was caused by algorithmic trading.

An agent-based model was developed to test Kirilenko et al. hypothesis. Based on previous literature the market was divided into subcategories of traders [27], [28]. However, to avoid presuming a specific trading strategy for any of these agents a zero-intelligence model was used, with the agents' behavior limited by constraints such as positions limits.

Financial Agent-based models

Zero-intelligence agent-based models are simulations where agents' actions are drawn from a set of underlying random distributions. One of the first zero-intelligence financial agent-based models was developed by Maslov, who modeled a financial limit order book. Maslov's [29] model introduces a new trader at each time step that is either a buyer or a seller with equal probability. The trader places a limit

order with probability q and trades at the market price with probability (1 - q). The price of the limit order is uniformly randomly offset from the last traded price, as a first-order approximation of participant behavior. Orders are not canceled or modified and remain until executed. The model produced a price time-series exhibiting characteristics in loose agreement with empirical studies.

Challet and Stinchombe [30] examine Island ECN order book data from a physicist's point of view. The model they propose is a particle system model, which is analogous to a zero-intelligence ABM. The mass of the particle is the size of the order, and the price is the spatial position on a one-dimensional lattice. Placing an order is a deposition, an order cancelation is an evaporation, and a crossing limit order is an annihilation. At each time step, an order is placed with probability p, and a price drawn from a normal distribution around the best quote. Depending on the parameters the model can produce empirically observed characteristics of price returns and volatility clustering.

Although Challet and Stinchombe's model can characterize an order book, it does not characterize the participants in the model. Additionally, Maslov's model incorrectly assumes that traders place limit orders at a uniform distance away from the last trade price. According to the data provided by the Commodity and Futures Trade Commission, high-frequency traders, which make up 70% of the trades in the equity market [27] and 35% of trades in the E-mini S&P 500, place 60% of all their orders one tick or closer to the last trade price. Additionally, cancelation of orders was a significant contributor to the flash crash [27]. Thus, without this crucial functionality, these modes are unable to accurately replicate modern markets as well as simulate extreme liquidity withdrawal.

Model Design

To overcome this limitation we developed six categories of agents, the description of each category coming from [27] and [28]. The categories are described below:

1. Fundamental Buyers and Sellers – take long or short positions on the asset during the entire duration the markets exists and trade with a low frequency.

2. Market Makers – take the position of straddling both sides of the market by taking long and short positions on an asset. Intermediaries' trades are meant to give the market liquidity.

3. Opportunistic – take a long or short position on the asset during the market day like a fundamental trader. However, they implement trading strategies that make them resemble Intermediaries because they do not take a significant position. These traders are believed to be using statistical or news-based strategies since they all tend to take the same position (i.e., long or short) [27], [28]. This phenomenon is known as herding, and in the simulation, it is created by providing the opportunistic trading class with external price news using the equation $Herd_t = Herd_{t-1} + U(-N, N)$, where N is a percentage of the current price.

4. High-Frequency – take long or short positions on an asset for short periods and trading with high-frequency near the best-ask and best-bid sides of the book. HFTs in the simulation use a momentum strategy. As the bid/ask queues becomes imbalanced HFTs will tend to trade in the same direction as the imbalance. In other words, if there are more bids in the order book HFTs have a higher probability of placing a buy order. This strategy was found in the data provided by the CFTC. HFTs will allow themselves to take large positions for short periods of time but will try to be neutral by the end of the day.

5. Small Traders – take either a long or short positions on the asset during the entire duration of the markets exist and trade with a very low frequency.

From work done by Kirilenko et al. [27] the following set of trader characteristic data was taken and used to derive the following market, which was then used to construct the agents.

Table 2: Simulated E-Mini S&P 500 Market Participant Description

Trader Type	# of Traders	Trade Speed	Position Limits	Market Volume
Small	6880	2 hours	-30 – 30	1%
Fundamental Buyers	1268	1 minute	-∞ – ∞	9%
Fundamental Sellers	1276	1 minute	-∞ – ∞	9%
Market Makers	176	20 seconds	-120 – 120	10%
Opportunistic	5808	2 minutes	-120 – 120	33%
High Frequency	16	0.35 seconds	-3000 – 3000	38%

Actual market participants were classified by using the following two variables.

1. Trade Speed – Average amount of time taken between order placements or cancelations.
2. Position Limit – Number of contracts allowed to be held.

Using order book data from the E-Mini S&P 500 contract provided by the CFTC and the classification process, we described empirically the style in which the agents placed orders into the order book. Each class's order placement is described by its order size and order price distribution.

1. Order Size – Distribution of order quantity size.
2. Order Price Selection – Distribution describing the number of ticks an order's price was about the last trade price.

These four variables were calibrated to create the six classes of agents in the agent-based model. These characteristics keep the traders in the realm of zero intelligence because the agents do not interact with market data to determine their actions.

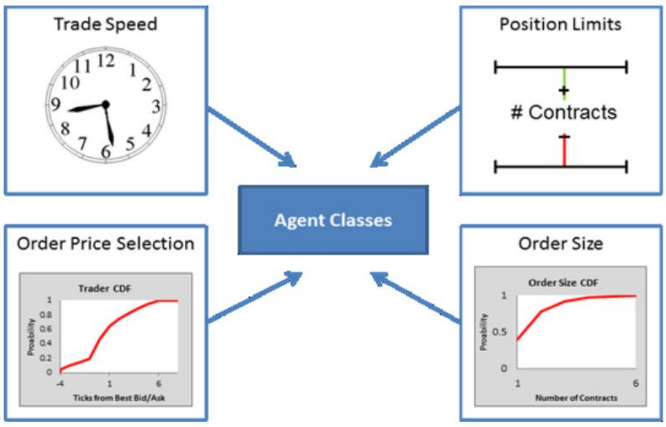


Figure 1 Components of an Agent Class

Agent-based Model Validation

In the creation of a properly functioning market simulation, it is a necessity to strike a balance between characterizing the market structure and participants accurately and making the market work in a manner that behaves similarly to the "stylized facts" [31]. "Stylized facts" are statistical, financial time series phenomena that are typically found in market data. We demonstrate the accomplishment of the task mentioned above by achieving the following set of criteria for validation:

- Trader Order and Execution Rates
- Order Book Construction Stylized Price Characteristics
- Distribution of Price Returns
- Volatility Clustering
- The absence of Autocorrelation of Returns
- Aggregation of Returns

Trader Order and Execution Rates

To demonstrate that the agent-based model has similar trade volume and cancelation rates as the real and simulated S&P 500 E-Mini markets is provided

Table 3 Order Comparisons

Trader Type	Simulated Volume	Actual Volume	Simulated Cancelation Rate	Actual Cancelation Rate
Small	1%	1%	40%	20 – 40%
Fundamental Buyers	10%	9%	44%	20 – 40%
Fundamental Sellers	10%	9%	44%	20 – 40%
Market Makers	10%	10%	35%	20 – 40%
Opportunistic	31%	33%	50%	40 – 60%
High Frequency	38%	38%	77%	70 – 80%

Order Book Constructions

The shape that the limit order book takes in an equilibrium state resembles 'V' formation when looking at the queue of orders away from the best bid /ask. This order book shape was first observed and described by looking at the Paris Bourse order book [32].

This structure has been an interest in econ-physicists who have tried to determine simple models that attempt to recreate this shape. One such model accomplished this by slowly modifying order prices closer to the current best bid/ask over a time given horizon [29]. More advanced models use an advanced calibration on the market data to construct an order placement schema and cancellation methods [32].

The arrival and cancellation process is another mechanism used to recreate the limit order book. Zovko and Farmer [33] looked at the creation of the limit order book from the perspective of the average size of orders placed at different price points. They suggested that this order process could be roughly estimated through a stochastic process of arrivals and cancelations using a Poisson distribution.

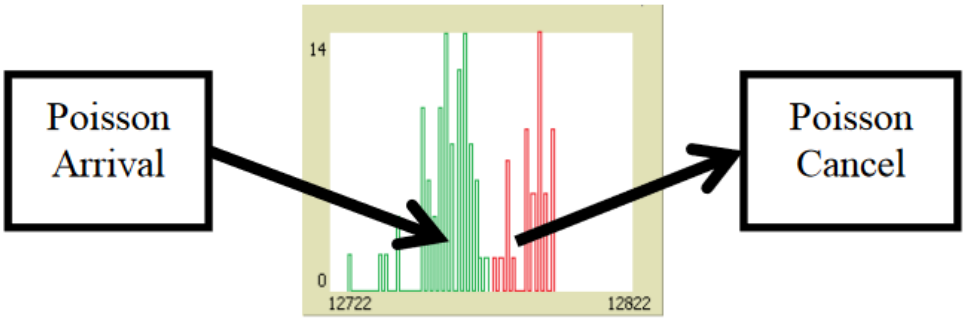


Figure 2 Simulated Order Book

The model presented in this paper uses multiple agents and order types that are drawn from their specific probability distribution to determine order prices selection. In combination with a Poisson arrival and cancelation process with a mean that is relative to the speed that a trader class is allowed to place and cancel orders. This process allows us to capture a 'V' structure without having to compromise our model by having a single agent class (see Figure 2).

Distribution of Price Returns

Empirical distributions of financial returns and log returns are fat-tailed. Mandelbrot [34] observed that the tails of the distribution of prices changes are extraordinarily long and the sample second moment typically varies in an erratic fashion. This observation has caused various suggestions Regarding the form of the distribution, ranging from the Student-t, hyperbolic, normal inverse Gaussian, and others, but no consensus exists for the form of the tails for all markets.

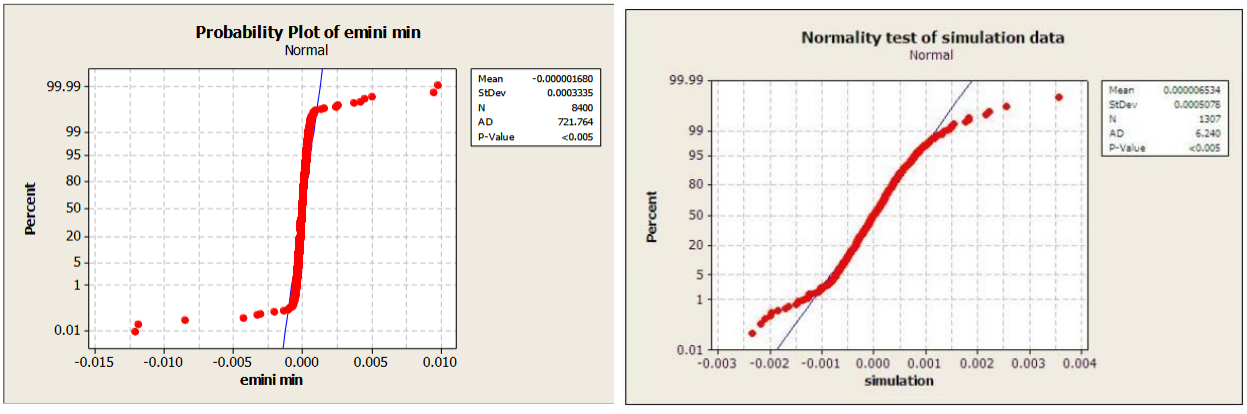


Figure 3 Price Normality Comparison

In the observed data from the S&P 500 E-mini and the simulation, we can observe this phenomenon by testing for the normality of the distribution of price returns. Figures 3 illustrate that both the real and simulated date diverges from normality at the tails.

Volatility Clustering

The characteristic of volatility clustering means that price changes tend to follow other price changes of the same size and is shown by the decay in the autocorrelation of price return variance. First discovered by Mandelbrot [34] and was finally translated into agent-based models by Kirman and Teyssiere [35] when they discovered a model would exhibit autocorrelation patterns in the absolute returns if a variable was 'herded' by the positive or negative opinion of an asset.

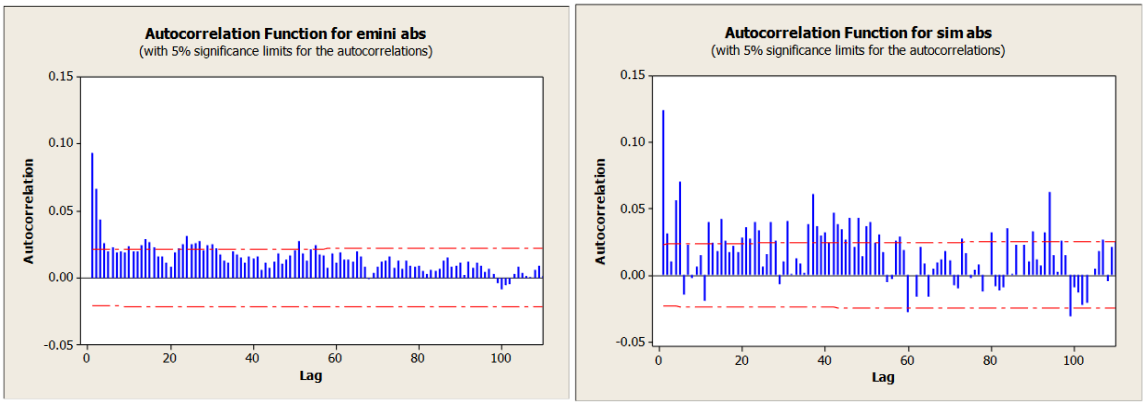


Figure 4 Volatility Clustering Comparison

The simulation implements the same herding variable, which influences the decision of opportunistic traders in the model (see Figures 4). The herding variable creates a similar autocorrelation pattern in the absolute returns that is seen in the S&P 500 E-Mini Futures Contract's minute price returns.

Absence of Autocorrelation

In proving that markets are efficient, it has been common practice to show that there is no predictability between price movements. In demonstrating this, we look at the autocorrelation or returns to show that there is no predictability of markets. Figures 5 illustrate that this property exists in both the real and simulated market data.

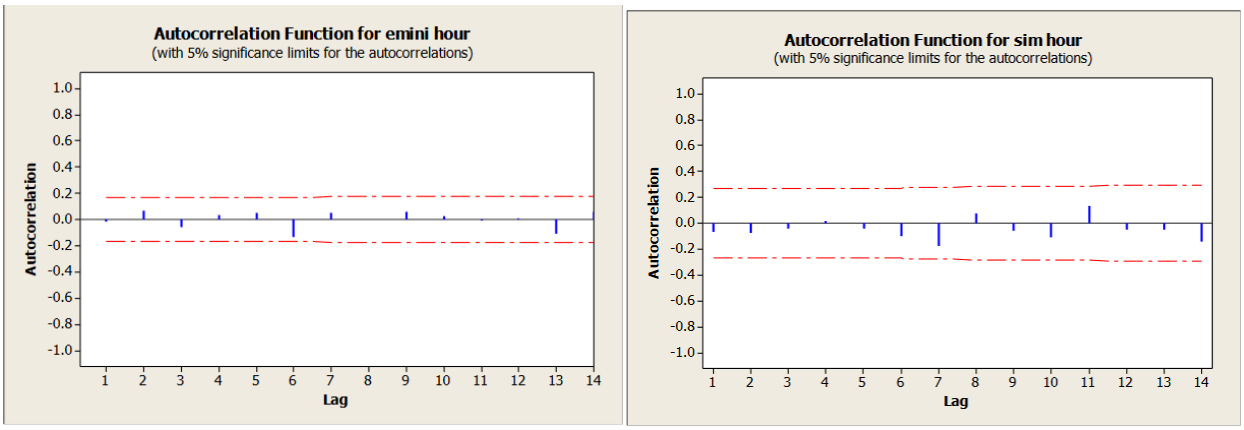


Figure 5 Autocorrelation of Price Comparison

Aggregation of Returns

The last stylized fact we examined was, as returns are calculated over an increased time scale the distribution approaches the Gaussian form. This cross-over phenomenon was noted by Kullmann et al. [36], where the evolution of the Pareto exponent of the distribution with the time scale is studied.

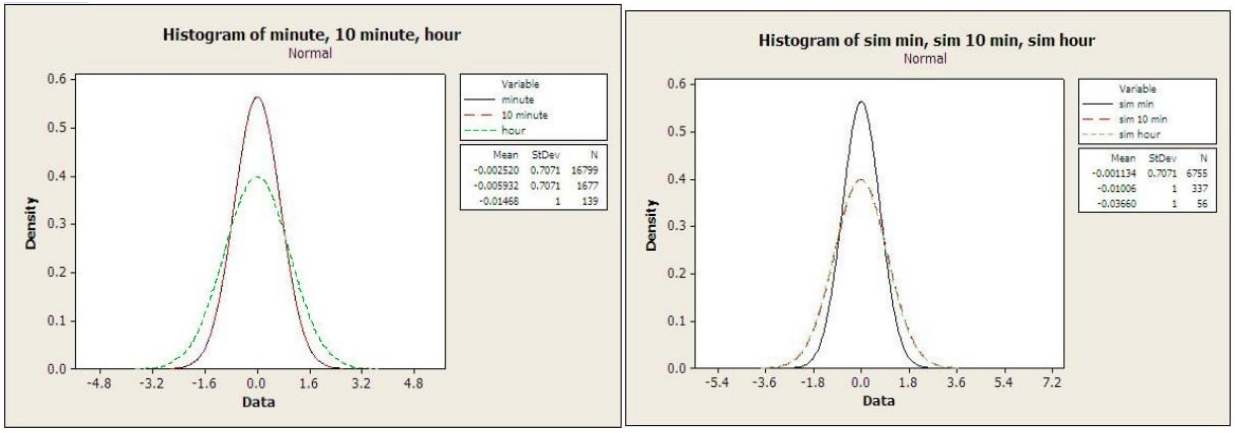


Figure 6 Aggregation of Returns Comparison

Figures 6 illustrate the standardized distributions of returns for S&P 500 index for June 2011 and the simulation. As the time scale increases, the more Gaussian both sets of distributions become. The next section illustrates how the agent-based model described in this paper can be used to examine the market phenomenon.

Replicating the Flash Crash

The agent-based model was used to examine the flash crash of May 6th, 2010. An agent that replicates the trading algorithm blamed for the Flash Crash was introduced to the model [27], [28]. The

agent tries to sell a large number of contracts. The agent examines the previous minute of trading and executes an aggressive sell order for 9% of the trading volume. Market makers and HFT are constrained by a rule, which forces them to lower their position level if they reach their position limit. Additionally, market makers were calibrated to withdraw from the market if the price fell 24 ticks below the moving average. Furthermore, Fundamental traders withdraw from the market, and stop-loss orders are triggered if the price drops 70 ticks below the starting price. Lastly, a market pause initiates when the price drops more than 1.3% in a second. The simulation was run with no other outside influences. The graph of the price and the moving average volume is displayed below. Additionally, the graph of the actual E-mini S&P 500 flash crash is displayed for comparison in Figure 7.

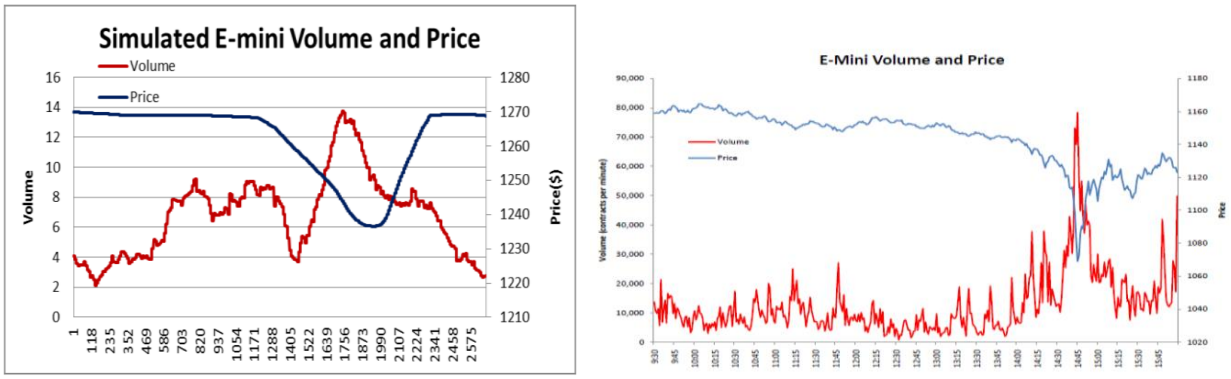


Figure 7 Simulated and Real-World E-Mini Volume and Price

Since the simulated market is 1/32nd of the actual of the E-mini market, some scaling issues arise. Most noticeably the prominent spike in volume is not as defined in the simulated market. This is because there is not as much liquidity in the market; thus it does not take as much trade volume to cause a flash crash. However, the simulation accurately represents a spike in the volume corresponding to the flash crash.

The first noticeable impact in the market was the Bid depth that began to decay at a fast pace, once the large agent began aggressively selling. After which fundamental traders begin pulling out of the market and the order book depth on both sides of the market crashes, which is illustrated in Figure 8.

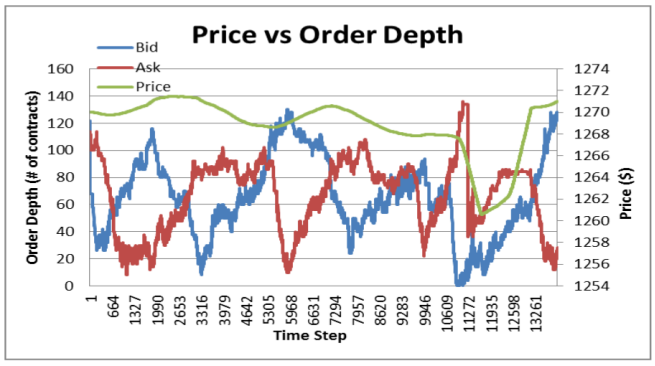


Figure 8 Simulated Price vs Order Depth

The crash in bid depth occurs at the same time the price begins an exponential decline (i.e., flash crash). This relationship between rapid liquidity loss and rapid price drops is not a novel finding. However, this leads to the question: "Is it possible to prevent the rapid loss of liquidity by changing parameters in the market?"

Discussion of High-Frequency Trader Impact on the Simulated Market

In recent years High-Frequency traders have been gaining an increasing amount of press. Their rapid speed allows them to trade on millisecond price fluctuations. One benefit High-Frequency traders' claim to provide is an increase in liquidity (i.e., the ability to rapidly buy or sell without moving price)[37]. However, questions have arisen about whether High-Frequency traders contributed to the flash crash. In the CFTC and SEC Report [28] it was asserted that High-Frequency traders consumed a significant amount of the initial large sell order. Once these algorithms reached their contract holding constraints, they began aggressively selling in front of a large trader, which helped to drive the priced down but was it the reason for the flash crash? An experiment was run to determine if High-Frequency trader enabled the flash crash. The size of the large sell orders was adjusted as well as the number of High-Frequency traders to see how these affected the minimum price of the simulation (i.e., a measure of how bad the flash crash is). A total number of 40 simulations were run for each variable pair, and the minimum price in each simulation was recorded. The median price for each variable pair is illustrated in the Figure 9.

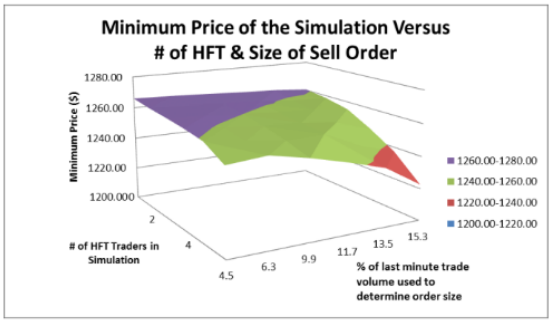


Figure 9 Number of HFT vs. Sell Order Size

As the number of High-Frequency traders was reduced to zero, the minimum price of the simulation increased [38], which illustrates that HFTs are necessary for the events on May 6th to occur. It has been hypothesized that HFT traded amongst themselves, as liquidity dried up. This rapid trading is known as "Hot Potato." The Staff of the CFTC & SEC [28] discovered that the rapid trading increased the trade volume, which further increased the large execution algorithm sell orders. This self-perpetuating cycle was concluded to be the cause of the flash crash.

If the execution algorithm is parameterized to execute smaller trade sizes, the severity of the price drop decreases. Although, the cycle mentioned above is occurring the market can absorb it because the aggressive sell orders increase at a slower rate, leading to the conclusion that the Flash crash would not have occurred without HFT and large sell orders from the execution algorithm [38].

Background on Financial Policy Making

The assertion that HFTs are harming the market leads to the question, "what can we do about it?" The Commodity and Future Trade Commission (CFTC) is mandated to ensure fair and orderly markets. Since the inception of CFTC in 1974, markets have evolved and so has the definition of fair and orderly. The Dodd-Frank Act passed July 2010, imposed new rules and regulations on the Financial Industry [39]. One of the goals of the act is to update regulations to reflect the new financial environment better. However, since the rulemaking process redefines 'fair' behavior in the market, a detailed analysis of the cost and benefits of each rule must be examined.

When Congress passes a law, it is a policy statue, which federal agencies are required to implement. Detailed regulations are developed through a process known as rulemaking. The rulemaking process for the CFTC is laid out in the Commodity Exchange Act (CEA) and the Federal Administrative Procedures Act (FAP) [40], [41].

The FAP requires a regulatory body follow formal steps before implementing a new rule. The optional first step is to publish an advance notice of a proposed rule. The notice presents preliminary information on the subject area. Additionally, public insight is requested to help shape the rule. After this initial step, a proposed rule is published. The proposed rule is a draft that contains the analysis and the justification for the rule.

The proposed rule is open to the public for comments. Written issues and concerns about the rule can be sent to the agency during this period. The CFTC must address every comment. However, comments can be grouped if they contain similar content. Thereby, relieving CFTC from having to address each comment separately. After the public comment period is closed, the final rule is written and published. This rule lays out the new regulatory policy that will be implemented. Additionally, this rule will address all public comments, as well as, contain the final analysis and justification for this rule.

The CEA requires the CFTC to complete additional steps in the rulemaking process. A notable step is to "Consider the Costs and Benefits" of any proposed regulation. Specifically, the CFTC is required to complete the following task:

A) Consider how the regulation affects the protection of market participants and the public,

B) Consider how the regulation affects efficiency, competitiveness, and financial integrity of futures markets,

C) Consider how the regulation affects price discovery,

D) Consider how the regulation affects risk management practices,

E) Consider how the regulation affects other public interest.

It is important to note that the CFTC will consider the cost-benefit analysis but is not required to alter the rule based on the findings. Additionally, the cost-benefit analysis does not have to contain quantifiable costs and benefits.

Private industries have sued on the grounds of inadequate cost-benefit sections. December 2011 the Securities Industry and Financial Markets Associations along with the International Swaps and Derivatives Association sued the CFTC over the Position limits rule [42]. Specifically, one claim made is that the CFTC, "failed to give serious consideration to the significant costs that the position limits rule will impose on commodity markets and the broader economy [42]." This is not the first lawsuit filed claiming a regulatory body's cost-benefit analysis was insufficient. In July 2010, a Security and Exchange Rule was overturned citing inadequate cost-benefit analysis [43].

Agent-based modeling can help regulators examine the cost and benefits of proposed rules. By examining the affect a rule has on each agent class, part A and D in the list can be addressed. Additionally, by looking at market measures, part of B, C, and E in the list of above can also be studied.

Previous Financial Policy Agent-based models

While the previously defined agent-based model does not accurately predict expected values of statistics, such as volatility. It can define the shape of a tradeoff curve as parameters of a rule are varied. Agent-based models illustrate how the rule may affect different traders and aid policies maker in creating targeted regulations that have minimal side effects on the financial market. Additionally, it can help policymakers justify their regulations to the public.

To mitigate HFTs' potential negative consequences, such as increased volatility [44]. The SEC contemplated a new rule [45], which would set a minimum time before an order can be canceled or

modified. Using the previously defined agent-based model a cost-benefit analysis was run to determine the tradeoffs between volatility, spread, market depth, the time between trades and liquidity ratio.

Agent-based modeling of potential public policy is not a novel concept. In 2008, the European Union developed an agent-based model design to test macroeconomic policies. This model incorporated several smaller models including credit, financial, and labor markets [46].

The financial aspect of the model is comprised of several kinds of agents, including households, firms, banks, a government agent and a central bank. Firms sell stocks to finance development of products, while the government agent issues bonds to finance debt. Household agents invest in these assets to make a profit. Household agents and firms have learning capabilities [46]. Therefore, it is not possible to determine if the learning capability or the interaction of agents is the cause of the market outcome. Additionally, it is possible that an altered learning algorithm will create different results. For this reason, the agent-based model was kept as a zero-intelligence model.

Frank Westerhoff [47] studied the use of agent-based modeling in several policy decisions. In his models' agents can choose between a fundamental trading strategy and a technical trading strategy. Agents switch between strategies depending on which was more profitable, creating a herding effect that generates the volatility clustering seen in real markets. Several rules were examined, most notably was trading halts. Trading halts pause trading when a price move is larger than a determined amount. He found that trading halts limited spikes and crashes. Additionally, he found that relative profits of agents remained unchanged by trading halts. Westerhoff's model can characterize an order book it does not characterize the participants in the model [47].

Experimental Design

In this cost-benefit analysis, the rule is defined as 'traders are prevented from modifying or canceling an order until a minimum time has expired.' The rule was defined this way to stop traders from

effectively canceling an order by modifying it away from the best bid or ask until the minimum time expires.

The minimum quote life rule seeks to decrease market volatility and large price movements. However, High-Frequency traders claim that this rule will force them to reevaluate their risk compensation and place orders farther from the best bid and ask, which might cause spreads to rise, liquidity to decrease, and time to trade to increase. The goal of the experiment is to examine the impact of a minimum quote life rule on the E-Mini market.

The minimum time that an order must remain untouched was varied from 0-1.25 seconds. Over the study, exogenous price indications were consistent for each simulation run. Additionally, empirical distributions that agents used to trade remained the same throughout the experiment. The underlying assumption is that all traders will trade in the same manner before and after the rule is implemented. The study allows for the examination of the rule's impact if the market is unchanged. This assumption is likely false in the real world. However, since the strategies are unknown for each trader, it becomes impractical to estimate how order placement might change.

The cost-benefit analysis examined Market level variables. The spread, distance between the best bid and ask, is calculated and recorded for each minute. Additionally, we calculate liquidity ratio every minute. Liquidity ratio is a measure of how much a trade affects the current price. The formula is presented below [48].

$$Liquidity\ Ratio = \frac{\sum Volume * Price}{\sum |\Delta Price|}$$

The larger the liquidity ratio, the less effect a single trade has on price movement. Additionally, several other measures of liquidity, market depth, and time between trades were studied. Lastly, we calculate the absolute value of minute price returns. Since each simulation experiences the same price

herding indication, a larger price return is indicative of higher volatility and not a fundamental difference in price between simulation runs.

Effects of The Minimum Quote Life Rule on The Simulated Market

Since the minimum quote life rule is designed to dampen volatility [45], which HFTs have been theorized to increase [44], it is natural to first examine if volatility changes as the minimum quote life rule is made stricter. As illustrated in Figure 10, over most of the parameter space volatility is shown to have an inverse linear relationship with the minimum order lifetime. Additionally, the variance in the volatility decreases as the minimum lifetime is increased, indicating that HFTs do increase volatility in the market and implementing this rule will decrease it.

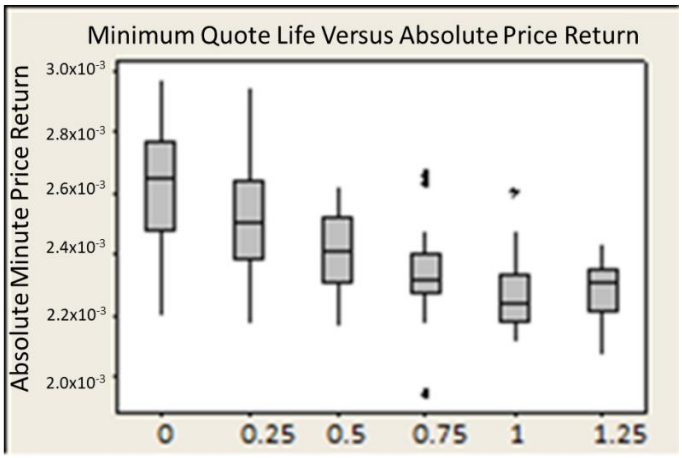


Figure 10 Determining Absolute Value of Minimum Price Returns

The HFT participants claim they improve many market parameters such as liquidity [49]. Liquidity can be measured in many ways. However, in general terms, it is the ability to buy or sell stocks and contracts quickly, without moving the price. HFT firms have stated an implementation of a minimum quote life rule would harm liquidity. We examine this claim by reviewing several measures of liquidity. Bid-ask spread, the amount of money that must be paid to execute a trade immediately is calculated every minute. Figure 11 illustrates the minimum quote rule lowers the bid-ask spread. However, the decrease is not statistically significant.

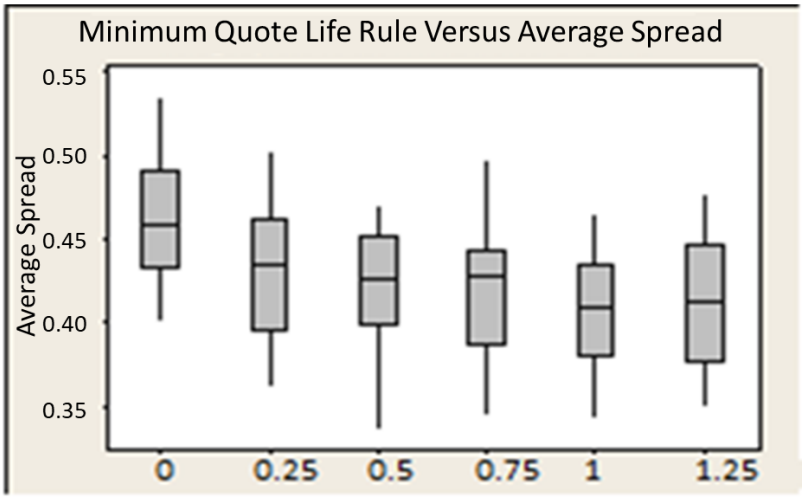


Figure 11 Average Daily Spread

The results show in Figure 11 will be discussed in greater detail later. It is important to note that the ticket size in the E-mini is $0.25. Therefore, in almost every simulation the average spread was always between 1 – 2 tickets. Another measure of liquidity is known as liquidity ratio. Liquidity Ratio indicates how many contracts or stocks can be traded without increasing the price. The higher the liquidity ratio, the more contracts or stocks can be traded without a significant price change. Figure 12 illustrates a non-statistically significant increase in market liquidity after the rule is implemented.

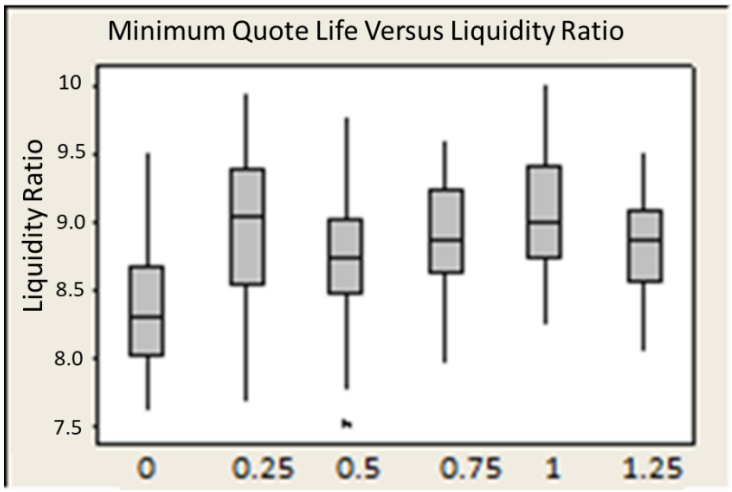


Figure 12 Study of Liquidity Ratio

An additional measure of liquidity is how long a resting order must wait before it is executed. HFTs cross the spread a significant amount, so it is believed that they lower this wait time. Figure 8 illustrates a

minimum quote life rule increases the time between trades. Furthermore, it has a linear relation over much of the parameter space, similar to the absolute value of returns.

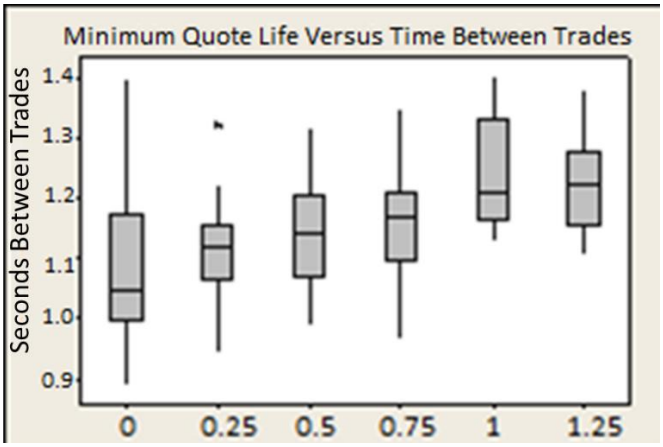


Figure 13 Time Between Trades

The last measure of liquidity examined is market depth. Figure 13 presents the average daily market depth of the order book. The larger the market depth, the more orders that can be executed without a substantial price change. However, the increase in market depth is not statistically significant over the tested parameter space.

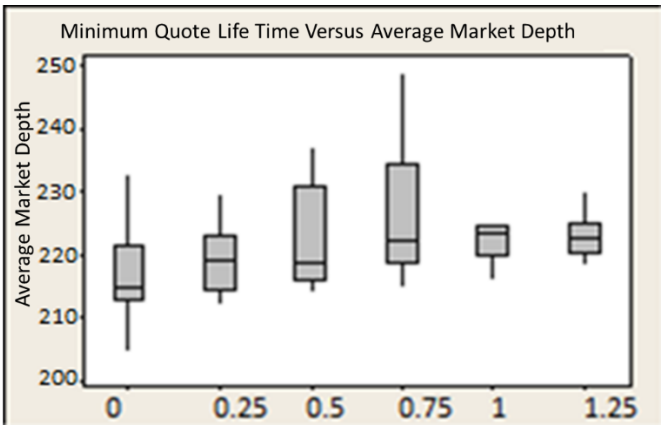


Figure 14 Studying Market Depth

The simulation of the minimum quote life rule generated novel findings. Firstly, the rule only affects HFT trading strategies. HFTs are the only class of traders that routinely cancel or modify a new order within 1.25 seconds of placing it. Since this rule focus on traders with small time horizons, HFTs would be forced to develop new strategies costing them time and money. Although, they represent a

small number of market participants they generate 35% of the trading volume in the E-Mini market and upwards of 70% in equity markets [27]. Therefore, any new strategy that is developed has the potential to cause even greater volatility then current strategies.

As the minimum quote lifetime is increased the absolute value of minute price returns decreased. We can attribute the decline in volatility to HFTs because they are the only class of trader directly affected by this rule change. HFTs aggressively cross the spread more frequently than other trading classes. In the model traders are only allowed one active order at a time, the minimum quote life rule acted as a throttle for HFTs.

This throttle prevented HFTs from crossing the spread as frequently, thus decreasing the overall number of trades and increasing the time between trades. Since HFTs make up a large percentage of trades, as their speed is decreased the time between trades increases. The relationship between HFT speed and trading rate, the aggressive HFT strategy implemented in the simulation, and the herding mechanism explains why HFTs increase volatility.

As positive news enters the market through the herding mechanism, opportunistic traders begin placing buy orders, which increase the probability that HFTs will place aggressive buy orders. The increase in aggressive trading adds pressure to prices and makes price tick up. As negative news enters the market the opposite occurs, and prices fall. These price shocks occur at constant times for all simulation runs. The minimum quote rule limits HFTs ability to increase the price when positive news enters the market and decrease the price when negative news enters the market, by decreasing the amount they can trade in a given period. This effectively dampens price movement, decreasing volatility, which Figure 15 illustrates.

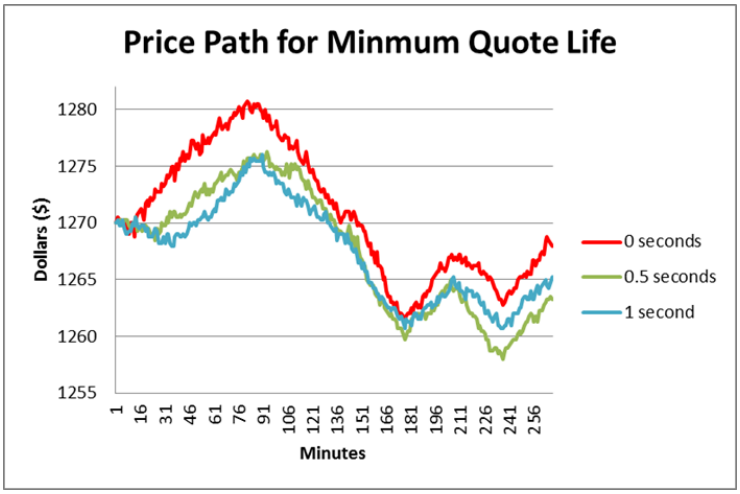


Figure 15 Measuring Price Paths

The graph was limited to three price paths for readability purposes. However, it illustrates that HFTs cause a considerable impact on price movement. As the minimum quote life rule limits their speed, their ability to affect price decreases and volatility consequently drops.

Market depth, liquidity ratio, and bid-ask spread all have no significant change in values between 0.25- and 1.25-seconds minimum lifetime implementation because HFTs are not the only trading class that crosses the spread. Both passive and aggressive orders arrive from other trading classes at a random frequency and cause trades to occur. These orders add liquidity to the market and limit the effects of throttling HFTs. In other words, the aggregate effect of other trading classes not hindered by this rule create bounds on the overall market parameters. Therefore, this rule may lower HFTs ability to increase liquidity but its effects on variables, such as market depth, will be limited by other trading classes.

Discussion of The Results

The analysis illustrates that the current debate over this rule has been framed in the wrong context. Currently, the debate focuses on bid-ask spread versus volatility. Bid-ask spread is bounded by the aggregate trades of other classes, which have time horizons longer than seconds. The debate should focus on the tradeoff between order trade time and volatility [39]. An example of a trading strategy that can be hurt by this rule is an ETF, which is tracking an index. If the ETF cannot acquire the necessary shares

in a given timeframe, then they must issue market orders and pay the bid-ask spread. As the minimum lifetime is increased the probability that an ETF will have to issue a market order increases.

This rule will cause traders with time constraints to issue market orders more frequently. Depending on where the parameters of the minimum quote life rule are set, it is possible to increase the cost to traders with fixed time limits. Detail studies across different securities and commodity should be done to determine what the minimum quote time should be. There is no guarantee that the optimal quote lifetime will be the same for each security or commodity.

Using the agent-based model regulators can derive a justification for why this rule should be implemented. Additionally, regulators can determine which traders will be directly affected by this rule. Lastly, the potential tradeoffs are illuminated through the simulation. Agent-based modeling will not replace current fundamental techniques of economics. However, it can facilitate decision-makers understanding of the complex nature of the financial market. The model framework presented in this paper can be calibrated to any financial market. Thereby, making it a ubiquitous regulation tool.

This model makes several assumptions that limit its accuracy. First, it assumes that traders only have one active order. Data supplied by the CFTC shows that HFTs generally have multiple orders on both sides of the book. The current model does not examine how this rule will affect a multiple live order strategy. Additionally, the model assumes zero-intelligence because a trading strategy is difficult to recreate from order book data. For this reason, it is impossible to use this model to accurately determine how this rule will affect different trading classes profit and losses. The next section presents an agent-based model that was developed to examine a potential gun control measure.

Zero-Intelligence Agent-based Model of Mass Shootings

Agent-based models provide a framework to discuss contentious policy decisions. Every mass shootings, such as Las Vegas and Parkland, reignite the debate on gun control. The tragedy at Sandy Hook

Elementary School in Newton, Connecticut, left 27 victims dead, including 20 elementary students, and led to the proposal of a bill to ban high capacity magazines and assault weapons.

Assault weapons and high-capacity magazines bans differ from gun registration and background checks because these bans would place new restrictions on American freedoms by explicitly barring Americans from manufacturing, purchasing, and possessing these items. Senator Dianne Feinstein (D-Calif.) put forth a bill to ban both high-capacity magazines and assault weapons. This bill was a source of heated debate, with gun advocates and gun control proponents disagreeing on its effectiveness.

To further the national debate on gun control, we use an agent-based model is used to examine the number of people shot under different small arms configurations (i.e., magazine capacity, gun's accuracy, gun's range, etc.). We have also made the model open source, allowing other researchers to leverage the model. The goal is to see how different parameters affect the number of people shot. With this information, we can answer the following two questions. First, how effective are the proposed assault weapons and high-capacity magazines ban? Second, what parameters are the most important in determining the number of people wounded or killed? Once these parameters are known, regulations can be designed to mitigate lives lost in mass shooting tragedies.

It should be noted that this model is limited to interpreting the efficacy of the assault weapons and high-capacity magazine ban. Results of this model should not be extended to other policy decisions, such as the deployment of security guards. The model was not designed to examine security guards, and the different level of training security guards have.

Background on Mass Shootings in the United States

There are many definitions of mass shootings. Depending on the criteria used to define mass shootings, there have been at least 62 mass shootings in the United States between 1982 - 2013 [50]. The criteria used to define mass shootings in this paper were the following:

1. the shooter took the lives of at least four people,

2. the shooting occurred in a public place,

3. the shooting was not tied to gang violence.

In these shootings, a total of 143 firearms were recovered. Semiautomatic handguns made up 47% of the recovered firearms, while rifles made up 18%. Figure 16 below illustrates the proportion of weapons recovered at mass shootings [50].

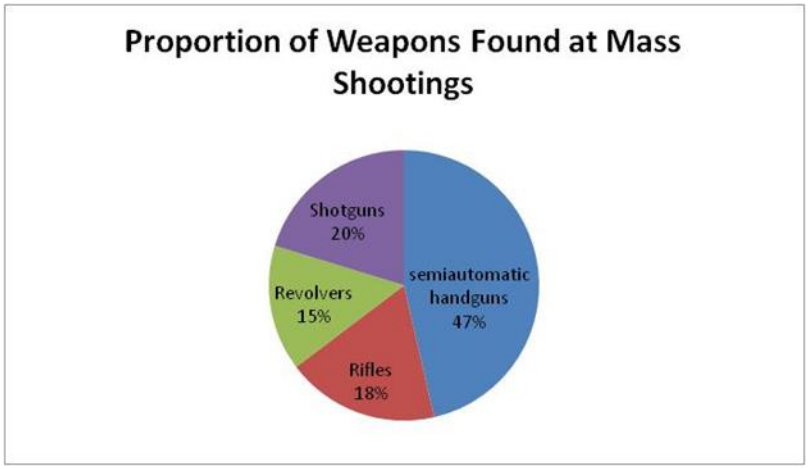


Figure 16 Proportion of Weapon types

It is interesting to note that 48 of the 143 weapons would be banned under Senator Feinstein's assault weapons ban. A total of 42 weapons contained high-capacity magazines, and 20 firearms were classified as assault weapons [50].

From 1999 to 2012, there were approximately 28 mass shootings in the United States, with ten occurring in 2012 alone. It should be noted that the Brady Campaign identified a larger number of mass shootings by using different criteria. However, the important thing to note is that mass shootings have become more frequent [51].

Public mass shootings are generally carried out by males acting alone [52]. Of the 28 mass shootings that took place between 1999 to 2012, 71% occurred indoors, which is an important statistic because being inside a facility limits the number of ways a person can escape. Additionally, approximately

10% of mass shootings resulted in a gunfight between authorities and the perpetrator. This means that in the majority of shootings the gunman either committed suicide, surrendered to an armed response, or escaped before police arrived [51], [52].

Explanation of the 2013 Assault Weapons Ban

Regardless of how mass shootings are classified, they are becoming deadlier, with the deadliest occurring in October 2017, which resulted in 57 killed and almost 500 people injured. The Assault Weapons Ban proposed by Senator Diane Feinstein's would make it illegal to purchase the assault weapons and high-capacity magazines identified in the ban [53]. An assault weapon is explicitly defined for rifles, shotguns, and pistols. For a rifle to be considered an assault weapon, it must be semiautomatic, meaning a repeating rifle that "utilizes a portion of the energy of a firing cartridge to extract the fired cartridge case and chamber the next round; and requires a separate pull of the trigger to fire each cartridge" [53]. Additionally, to classify a rifle with a detachable magazine as an assault weapon it must contain at least one of the following:

- A pistol grip
- A forward grip
- A folding, telescoping or detachable stock
- A grenade launcher or rocket launcher
- A barrel shroud

A rifle with a permanently attached magazine and a magazine capacity larger than ten rounds are considered an assault weapon. A notable exception is for magazines that are of a tubular design and only capable of firing .22 caliber rimfire ammunition [53]. For a pistol with a detachable magazine to be classified as an assault weapon, it must also be semiautomatic and have one of the following characteristics:

- a threaded barrel,
- a second pistol grip,
- a barrel shroud,
- the ability to accept a detachable magazine at some location outside of the pistol grip,
- be a semiautomatic version of an automatic design.

Pistols with permanently attached magazines that accept more than ten rounds are classified as assault weapons. An assault weapon is any shotgun with a revolving cylinder or a semiautomatic shotgun with at least one of the following characteristics:

- a folding, telescoping, or detachable stock,
- a pistol grip,
- a permanently attached magazine with the capacity to accept more than 5 rounds.

The bill goes on to specifically outlaw several types of guns, such as all AR and Kalashnikov (AK) models, as well as several Bushmaster, Barrett, and Thompson guns [53]. Additionally, the bill bans high-capacity magazines, defined as magazines with the capability of accepting more than ten cartridges. It is important to note that the ban does not limit the muzzle velocity, caliber, or rate of fire of any weapon that is not outlawed by this bill.

Background on Small-arms Agent-based Models

There is a precedent for using agent-based simulations to model combat scenarios. Erlenbruch [54] designed an agent-based simulation of German peacekeeping operations. The simulation assessed different strategies against an armed mob. German peacekeeping troops, consisting of a platoon equipped with rifles, personnel carriers, and infantry fighting vehicles, were assigned to protect a high-value target. The peacekeeping units had a choice between a defensive and aggressive strategy. The effectiveness of each strategy was determined by the number of peacekeepers and civilians wounded or

killed, as well as if the mob entered the high-value target. They found that an aggressive policy resulted in higher overall mission effectiveness.

The methodology mentioned above was an extension of a simulated peacekeeping unit defending a location from an unarmed mob [55]. A “tit-for-tat” strategy was compared to disproportionate response strategy, with the outcomes being measured by the number of peacekeepers and civilians injured or killed. The analysis found that responding aggressively against an unarmed mob resulted in fewer people wounded than a measured response. The model predicted people would be scared off faster when the peacekeepers were more aggressive, resulting in an overall decrease in people injured or killed.

It is important to note that lethality was measured deterministically, with agents given a certain amount of health points. Each weapon has a specific damage score. When a weapon hits an agent, the damage score is subtracted for that agent’s remaining health points. Agents’ with health points of zero or less are considered dead [55].

Agent-based modeling has also been used to simulate larger-scale conflicts. Choo et al., [56] simulate two forces engaging in armed combat. The agents are classified into a red team and blue team. The blue team’s objective is to capture a facility held by the red team. The red team has an evolutionary algorithm, which alters its strategy over different simulation runs. The goal is to test the robustness of the blue team’s strategy to changes in the behavior of the red team. They found that red team can improve their survivability by 27% by becoming more aggressive and relying on ambushes. The model highlights the pros and cons of the blue team’s strategy and thus provides the decision-maker with more information. These studies demonstrate the acceptance of agent-based modeling by military leaders.

Yuna Wong [57] describes in his dissertation the need to model non-combatants in military operations. He separates non-combatant behavior into simple and complex. Simple behavior encompasses running away and hiding behind objects. In some circumstances, it can be assumed that

civilians will run away or hide once combat is initiated. However, in scenarios such as Somalia complex behaviors consisting of civilians rioting and acting as human shields occurred. The accurate modeling of non-combatants allows for realistic civilian casualty estimates. The agent-based simulation in this paper builds off the rich literature of combat agent-based models and expands the use of combat ABM into public policy modeling.

Mass Shooting Model Design

The purpose of this simulation is to determine how different parameters affect the number of people shot in mass shooting events. This model does not examine the lethality of the parameters. Instead, it is assumed that by lowering the number of people shot in mass shootings, the number of fatalities in these shootings will also decrease [58], [59].

There are several reasons that this study opted not to examine lethality. Firstly, previous agent-based models have demonstrated that accurately modeling lethality is not necessary to generate relevant and credible findings [54], [55]. Furthermore, the authors do not have the necessary anatomy knowledge to accurately predict or measure the lethality of a bullets trajectory through a human body. Additionally, the bill proposed by Senator Feinstein does not limit the caliber of bullets, which correlates to both how fast a projectile is fired and how heavy it is, ultimately determining the likelihood of a projectile killing or passing through the human body. Since the proposed bill did not limit the caliber of a projectile the model did not consider bullet penetration of walls or people.

Two variations of the simulation were created; one is an outdoor scenario, and the other is an indoor scenario. Once the parameters that most affect the number of people wounded or killed are identified, the potential effectiveness of Senator Feinstein's bill can be assessed. Additionally, recommendations can be made to mitigate the negative consequences of mass shootings.

The purpose of experimenting with both an indoor and an outdoor scenario is to see if the importance of parameters changes between the scenarios. There are three types of agents in these

simulations: civilians, armed security guards, and the gunman. The rules that each agent class follows are described below. The gunman and security guards' rules are held constant across scenarios.

Civilian Outdoor:

1. Turn 180 degrees from gunman ± N(0,1)
2. Move forward at assigned *running speed*
    a. If someone in the way, move to within 0.3 feet of them
3. Considered safe if distance from *gunman* > *gun's range* + *escape distance*

Civilian Indoor:

1. Turn in direction of closest door ± N(0,1)
2. Move forward at assigned *running speed*
    a. If someone in the way, move to within 0.3 feet of them
    b. If wall/object in the way move to within 0.3 feet of wall/object
3. Considered safe if exit the room

Gunman Indoor/Outdoor:

1. Are there bullets in the magazine? If not, reload for this turn and skip all remaining rules (reload takes assigned *reload time*)
2. Aim at closest civilian ± N(0,σ)
3. Fire *X* rounds
    a. If less than *X* rounds left in the magazine, fire remaining rounds.
4. Move towards most recent targeted agent at (move at assigned *running speed)*

Security Guard Indoor/Outdoor:

1. Are there bullets in magazine? If not, reload for this turn and skip all remaining rules (reload takes assigned *reload time*)
2. If there is a civilian within 1.5 feet of a straight line between agent and the gunman, move to step 5 for this turn (won't shoot if a civilian is in the way)
3. Aim at closest gunman ± N(0,σ)
4. Fire *X* rounds at gunman
    a. If less than *X* rounds left in the magazine, fire the remaining rounds.
5. Move toward the gunman at *running speed*
    a. If gunman is within the distance, run this turn then tackle him

The simulation is considered complete if all civilians have escaped or been shot, or the gunman has been shot/tackled. The parameters (italicized above) that are used are defined below.

1. Running speed – feet/second the agent can run
2. X – the number of bullets fired a second
3. σ – the standard deviation a gunman or security guard aims off center
4. Capacity – number of rounds the magazine of the agent can hold
5. Gun's Range – the range at which the bullet fired horizontally from 5 feet 5 inches will hit the ground
6. Escape Distance – the distance past a gun's range that an agent needs to reach to be considered safe (assumed to be 200 feet in the simulation)

The simulation environment consists of a two-dimensional Cartesian world. When it is either the gunman's or a security guard's turn to act, they aim directly for the center of their target. The bullet they fire is perturbed from this intended trajectory. This error represents both the gun's and the shooter's accuracy. A smaller perturbation is indicative of a more accurate gun and shooter. The perturbation is a

normally distributed random variable. The baseline standard deviation (*σ)* of the random variable is calculated from the maximum effective distance of the gun. The maximum effective distance of a gun is considered the distance at which an average trained shooter can hit a human torso, assumed 1.4 feet in diameter, 50% of the time. Assuming a normal distribution, it becomes possible to determine the standard deviation or baseline measure of the accuracy of each weapon. Take for example the AR-15, which has an effective range of 1804.46 feet [60].

$$\theta = \tan^{-1} \frac{radius\ human\ torso}{effective\ range}$$

Using the equation above, the AR-15's $\theta = 0.022\ degrees$. In other words, 50% of the time the AR-15 will fire a bullet within 0.022 degrees of the intended aiming line. The mean of the random normal distribution is assumed to be 0. We use the equation σ=θ/0.67, to determine the standard deviation [61].

In the case of the AR-15, the standard deviation is 0.033. The shooter causes any additional deviation. Therefore, to represent a less accurate shooter, the standard deviation can be increased. We now have the information necessary to plot the trajectory of a bullet fired. The figure below illustrates the process. The shooter aims at the center of the target. For the gunman, this is the closest civilian, and for the security guards, this is the gunman. However, the bullet leaves at a trajectory that is altered by random normal distribution; for the AR-15 this is N(0,0.033 + human error). The bullet travels on this adjusted trajectory, and the first person that is less than 0.7 feet (assumed radius of the torso) off of the trajectory and within the range of the gun is considered shot. The bullet is assumed to travel instantaneously and does not penetrate through a person or wall.

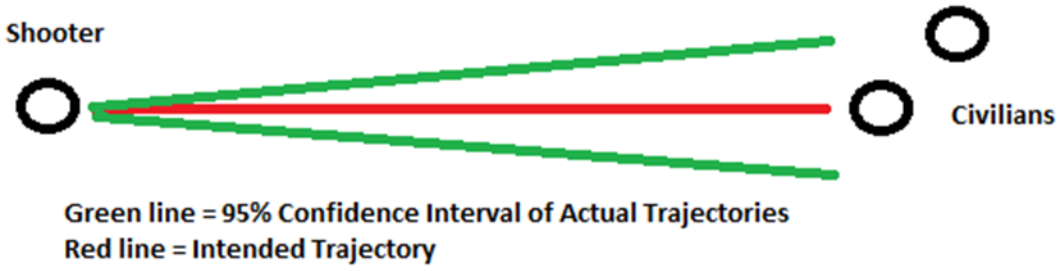


Figure 17 Bullet Trajectory

The range of the gun is calculated by assuming the gun is fired horizontally from a height of 5 feet 5 inches. Throughout the bullet's flight, it is assumed to travel at the muzzle velocity. For the AR-15, this is 3250 feet/second. Using the equation below, this gives the AR-15 a range of 1100 feet [60]. Where *X* represents distance, *v* represents velocity, *a* represents acceleration, and *t* represents time [62]. Therefore, in the outdoor simulation an agent must be a distance of 1300 feet from the gunman to be considered safe. The Beretta handgun, on the other hand, has a range of 400 feet, which means the escape distance from a gunman armed with a Beretta is 600 feet.

$$X = X_0 + v_0t + 1/2at^2$$

The last two variables used to define a weapon were magazine capacity and rate of fire. For small arms, there is a measure called the effective rate of fire, which is the rate of fire that maximizes both numbers of shots and accuracy. For the AR-15 this is 45 rounds/minute, and assuming a standard 30 round magazine, the AR-15 can be described in the graph below [60].

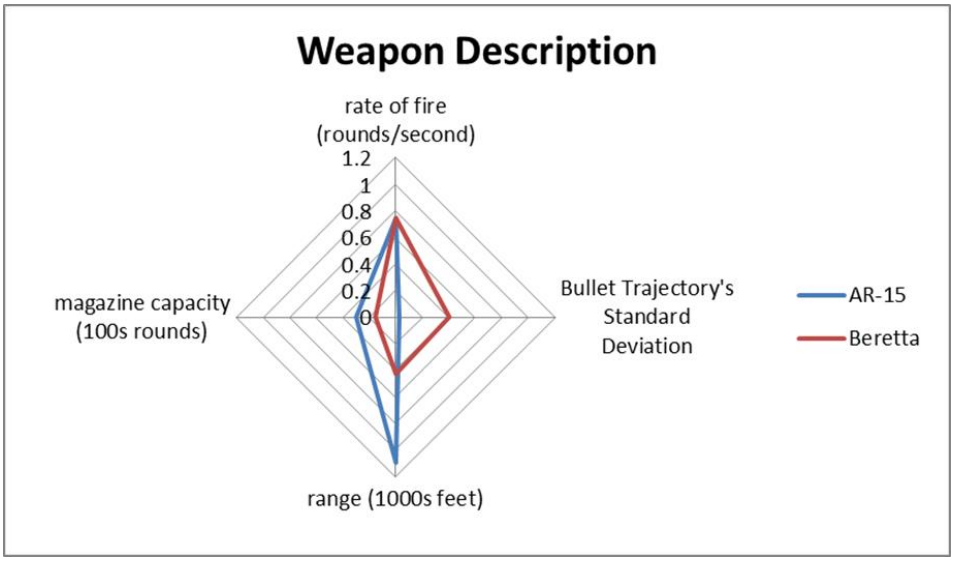


Figure 18 Graphical Description of Firearms

The procedure mentioned above was also used to describe the Beretta handgun. Figure 18 illustrates the difference between the Beretta and the AR-15, range is represented in 1000s of feet and magazine capacity is represents in 100s of rounds. The AR-15 has a more extended range, carries more rounds, and is more accurate than the Beretta. However, their rate of fire is the same. Using the four parameters different combinations of gun configuration and gunman's skill can be tested.

Model Validation

On July 20th 2012, James Holmes committed a mass shooting in a midnight showing of The Dark Knight Rises. Mr. Holmes was armed with a 12-gauge Remington 870 Express Tactical Shotgun, Smith & Wesson M&P 15 semi-automatic rifle, and a Glock 22 40-caliber handgun. It is important to note that all the aforementioned firearms violated Senator Dianne Feinstein assault weapon ban. After releasing tear gas into the theater, James Holmes fired approximately 110 rounds, striking 58 people [63]–[65]. The Aurora Colorado shooting was used as a test case to validate the previously mentioned simulation assumptions.

The movie theater layout was created using the minimum recommended dimensions for aisles and width of chairs [66], [67]. It is important to note that the movie theater is assumed to be flat. That is

to say rear, seats are not raised above seats in front of them. Civilians are assumed not to climb over chairs. However, approximately 10% of agents will hide in place. These civilians will not be harmed, but other people crossing over them can only travel at 90% of their normal speed. If a civilian is shot, they are assumed to fall to the ground and will slow other civilians crossing over them by 90%. The average civilian running speed is slowed to approximately 3 feet/second.

According to witness statements James Holmes remained close to the front theater exit. Therefore, the simulated gunman is assumed not to move. Additionally, the five shotgun rounds fired by Mr. Holmes were not simulated. Forty-five rounds were fired from the M&P 15, and sixty rounds were fired from the Glock 22. Assuming a standard fifteen round magazine the Glock was reloaded three times. In the simulation this reload was assumed to take two seconds. The M&P15 was assumed to have a standard deviation of 2 degrees off center, while a Glock was assumed to have 3 degrees. The firing rate for both firearms was assumed to be 1 round a second. The range of the firearms were not a factor, due to the size of the room. Figure 19 displays the simulated movie theater.

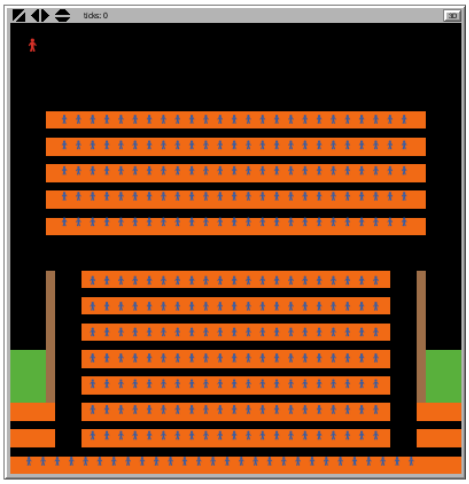


Figure 19 Simulated Movie Theater

The simulation was run 100 times, and the number of people shot was recorded. The model is assumed to be accurate if the actual casualty number is close to the range of simulated casualties. The range of simulated casualties was (44 – 58). The actual number of people shot was 58, which is the

maximum of the simulated range, which is to be expected because Mr. Holmes fired five additional shots from a shotgun, which was not simulated. Considering the additional weapon discharges the simulation is accurately predicting the number of people shot. Therefore, we assert that the previously mentioned assumptions do not significantly affect the outcome of the simulation. This simulation is stored at OpenABM.org [59]. To ensure generalizable findings, two experimental simulations were created to test the efficacy of Senator Dianne Feinstein's assault weapons and high-capacity magazines bill.

Experimental Procedure

The purpose of the experiment is to determine the critical factors that determine the number of people shot during a mass shooting. The order of events is as follows:

1. the gunman acts according to the previously defined rules,
2. every civilian is chosen at random and then acts according to the previously defined rules,
3. every security guard is chosen at random and then acts according to the previously defined rules,
4. repeat steps 1-3 until the stopping condition is reached.

Every complete sequence of steps 1-4 is assumed to take one second. The parameters that are examined include the parameters of the gunman's weapon (e.g., the rate of fire, accuracy, range, magazine capacity) and the number of armed security guards. Each parameter group is run thirty times to provide a large sample size. All security guards are initialized with the following set of values.

1. Magazine capacity = 15
2. Range = 500
3. Standard Deviation of Trajectory (Accuracy- the smaller the number the better the accuracy) = 1
4. Rate of Fire = 1 round/second

5. Running Speed = 21.67 feet/second (80th percentile in 17-year-old boy shuttle run)
6. Reload Time = 2 seconds (was held constant for both gunman and security guard)

The gunman is assumed to be in the 30th percentile in 17-year-old boy shuttle run, giving him a running speed of 12.24 feet/second. Civilians are assumed to have a truncated normally distributed running speed, with a mean at 12.77 feet/second, a standard deviation of 8.9 feet/second, and a minimum running speed of 1 foot/second.

Simulation Results

The outdoor scenario begins with a gunman starting in the center of a 200-person crowd, with the farthest person being no more than 420 feet away. As the scenario begins, the crowd disperses in all directions in an attempt to flee from the gunman. Each combination of the parameters depicted in Figure 20 is run 30 times and illustrates the average number of people shot over all simulations run, while that parameter equals the given value.

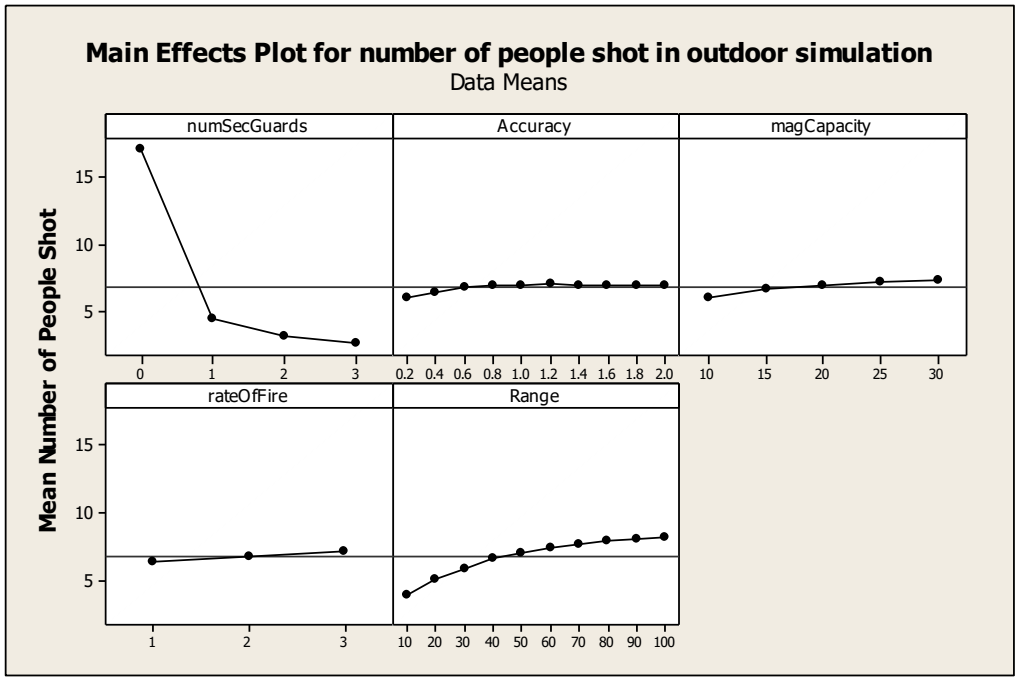


Figure 20 Main Effects Plot for Outdoor Simulation

The above figure illustrates that increasing the number of security guards decreases the number of people shot. There is a significant reduction in the number of people wounded or killed when the first security guard is added to the simulation, and smaller and smaller reductions as more are added, illustrating a diminishing return. Since the gunman acts first, he is likely to shoot somebody on his first turn. Even if a guard is close by, the gunman has the element of surprise and can shoot somebody before the guard can react.

The main effects plot also shows that the standard deviation of trajectory angles (accuracy) has little effect on the number of people shot. As accuracy is decreased by increasing the standard deviation, the number of people shot goes up slightly. There are many people in the crowd, and by shooting in a wider area, the gunman is likely to hit multiple people in the first couple of seconds, thereby slightly increasing the number of people shot.

To get a better understanding of the parameter values that most affect the number of people shot, an ordinary least squares model was fit. The equation below gives the best linear estimation of the number of people shot; the model gives a coefficient of determination of 53.34%. The equation below is used to demonstrate the relative importance of the different parameters.

*Number of people shot = -4.4 * Number security guards + 0.004 * range + 0.06 * magazine capacity + 0 .4 * rate of fire + .4* accuracy + 8.6*

The above equation provides a great deal of insight into what parameters affect the number of people wounded or killed during mass shootings. For everyone additional armed security guard, an estimated four people will be saved. The reason this number seems small compared to the drop seen in Figure 20 is because it is fitted over all tested values of security guards. As stated earlier, there is a diminishing return in adding security guards. Therefore, this value must consider this diminishing return and can be interpreted as the average number of lives saved for each additional security guard. The first security guard provides a significant reduction in the number of people shot and the subsequent security

guards provide smaller reductions thus diminishing the overall average reduction in the number of people wounded or killed for each additional security guard.

The rate of fire has the second largest effect on the number of people shot. For every additional bullet fired per second, an average of 0.4 additional people will be shot. Furthermore, for every 500 additional feet in range, an average of 2 additional people will be wounded or killed. The above equation provides better insight into which variables matter, but it does not tell the entire story.

There are interactions effects (non-linear relationships) that exist between a subset of the variables, which Figure 21 illustrates. For readability purposes, only the most significant interactions are illustrated in the graph below. If there are no armed security guards, the range has a much more significant effect on the number of people wounded than it does when there is a security guard present. The simulation illustrates that the shooter is capable of engaging targets farther away from him because armed resistance has not confronted him. However, once armed resistance is added into the simulation, the shooter does not have time to engage targets at a farther range. As the range of the firearm increases, the maximum capacity of the firearm becomes more important. The reload time is only two seconds, but it is evident that as the magazine size is increased, the number of people shot on average will also increase.

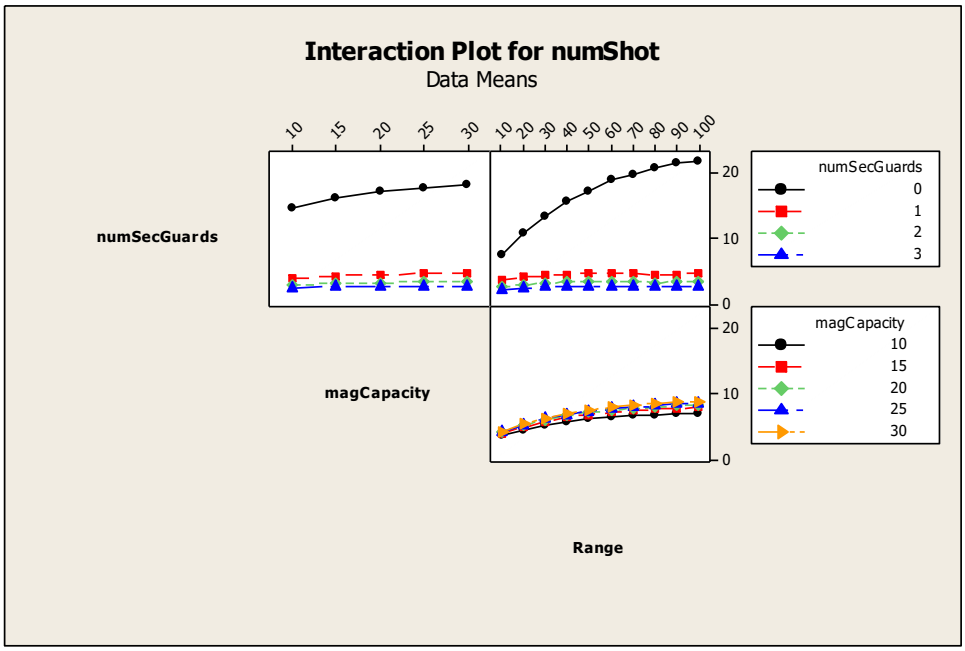


Figure 21 Outdoor Interaction plot

The same analysis was done for an indoor example. Two hundred people are in a 50 feet x 50 feet room. The exit door size is calculated using the 2000 NFPA Life Safety Code, which requires that a room with an occupancy of 200 people have a door with an exit width of 40 inches or more [68]. The simulation starts with the gunman (in red) in the southwest corner of the room and the exit door along the north wall, which Figure 22 shows. This simulation is stored at OpenABM.org [58].

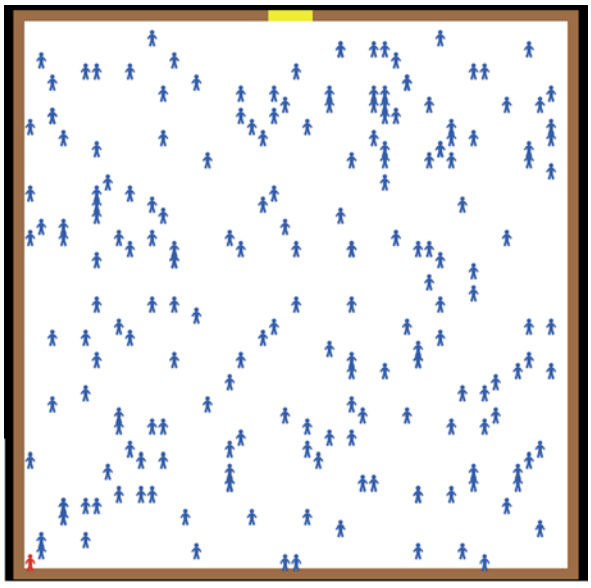


Figure 22 Initialization of Indoor Simulation

All the same parameters are tested, except for the range of the weapon because the minimum range tested exceeds the maximum length of the room. The main effects plot is presented in Figure 23, along with the Least Square linear regression model. The linear model has a coefficient of determination of 65%.

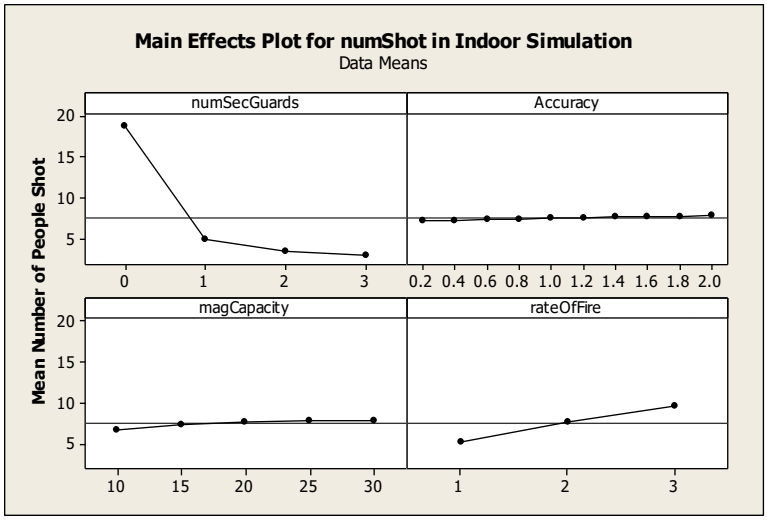


Figure 23 Main Effects Plot for Indoor Scenario

Figure 23 illustrates that magazine capacity and accuracy have minimal effect in this scenario. The magazine capacity has a small effect because the scenario is limited to one room. Thus, the gunman may only reload once or twice before the entire simulation is concluded. If the simulation is expanded to more massive complex, the magazine capacity will increase in importance. The main effects plot also illustrates that increasing the number of security guards again has the most significant effect on the limiting the number of people shot.

*Number of people shot = -4.9 * Number security guards + 0.06 * magazine capacity + 2.1 * rate of fire + .4* accuracy + 9*

The results of the indoor simulation demonstrate that the rate of fire is more critical in confined areas than in open areas. For every additional round of fire per second, on average of two more people are wounded or killed in the simulation. Doors and to lesser degree hallways, which act as funnels (see

Figure 24). As people approach a door, they are forced to assemble in a narrow space, which means the gunman is likely to hit somebody because the crowd of people becomes denser. The indoor simulation, even though it is limited to one room, has more people killed on average, which is exemplified by the intercept terms of the linear model. This finding is consistent with real-world events. By selecting a place with a limited number of possible egresses, a shooter can maximize the number of people shot.

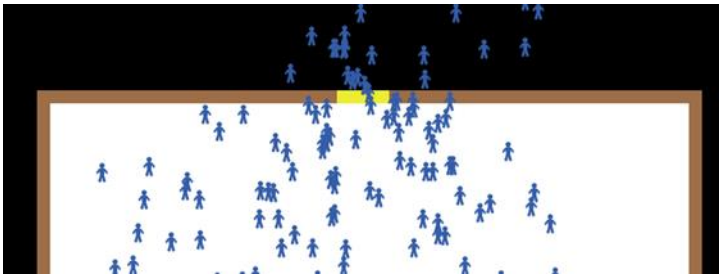


Figure 24 Example of Funneling

As before, the linear model does not tell the whole story; there are interactions effects between parameters. Figure 25 shows that the rate of fire is more important when there are no armed security guards present, which is intuitive. The gunman can sustain a more extended rate of fire, thereby increasing the number of people shot.

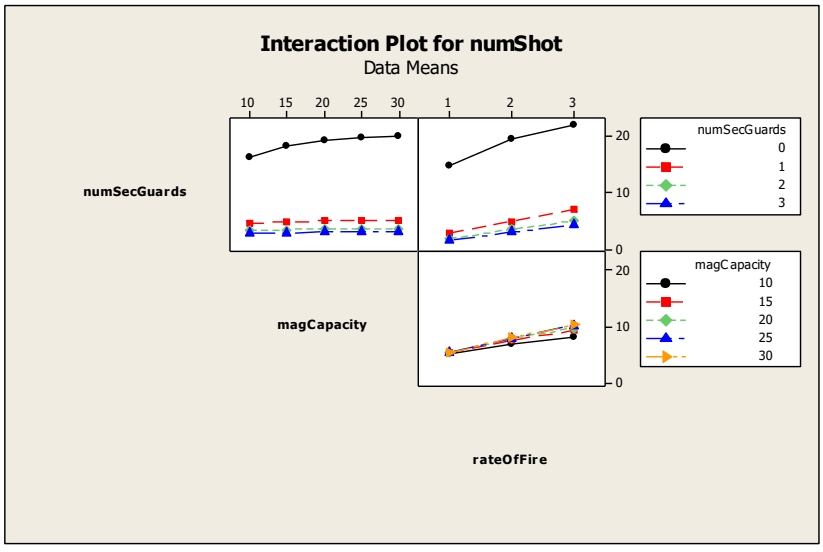


Figure 25 Interaction Effects of Indoor Simulation

Also, as the rate of fire increases so does the importance of the magazine size. The larger the magazine capacity, the more rounds a shooter can fire within a finite amount of time and the more people he can shoot.

Using the simulation, we examined the assault weapons ban proposed by Senator Feinstein. In its current configuration, the ban does not limit the rate of fire[53]. The ban would allow for detachable magazines as well as semiautomatic pistols, shotguns, and rifles. In the best possible case, the assault weapons ban may make firearms slightly less accurate by disallowing forward grips and barrel shrouds. Additionally, the simulation illustrates that in crowded mass shooting events, lowering a weapon's accuracy may increase the number of people shot.

As the crowd becomes denser, such as when people are going through a door, a shooter can widen his shooting area generating a deadlier impact. As the crowd assembles to get through a door, the crowd becomes significantly wider than it is deep. Thus, there are fewer people in front of a civilian then beside him, making the gunman more likely to hit somebody if he widens his target area. Therefore, we conclude that the assault weapons ban will not have limited effect on the number of people shot during mass shootings.

Discussion of Mass Shooting Simulation Results

Excluding adding armed guards, of the parameters tested limiting a weapon's rate of fire, making it impractical to fire a weapon more than a defined number of times in a finite period, is the best way to decrease the number of people shot [69]. The ban on high-capacity magazines seeks to accomplish this by forcing a shooter to reload more often. For example, if forcing a shooter to reload lowers the shooter's rate of fire by 0.2 bullets per second, this lowers the rate of fire by 12 bullets per minute. More extreme proposals could include banning semi-automatic weapons, which would lower the rate of fire because it would require the operator to perform an additional task when firing. Additionally, banning detachable

magazines and requiring that magazines only allow for bullets to be individually loaded (e.g., revolver design) would lower the rate of fire because it would take longer to reload the weapon.

According to the results of the simulation, an example of an effective regulation is Florida's bump-fire stock ban. Following the shooting in Parkland on February 14, 2018, Florida moved to ban bump-fire stocks. The law states, "a person may not import into this state or transfer, distribute, sell, keep for sale, offer for sale, possess, or give to another person a bump-fire stock [70] ." While a bump-fire stock was not used in the Parkland shooting, they were used in the Las Vegas, Nevada shooting, which is the deadliest mass shooting in U.S. History. The ban on "bump-fire stock" will prevent a mass shooter from increasing the rate of fire of the weapon, resulting in lives being saved.

The results of the model cannot be used to promote adding security guards. Numerous factors determine a security guard's effectiveness, from training to combat experience. Therefore, a prototypical "highly trained" guard was simulated to examine the nonlinear effects that arise as guards are added to the simulation. Although the model determines adding security guards is the number one way of decreasing casualties, it does not consider the different skill levels that security guards will have. Therefore, this finding cannot be used to promote adding additional security guards.

The simulation was designed with the purpose of gaining insight into the critical parameters of mass shootings. A higher fidelity model can be created to examine the studied parameters' effect on lethality, as opposed to the number of people shot. Additionally, allowing for bullets to pass through walls and people, as well as designing more realistic environments can also be used to increase the fidelity of the model. However, this model provides a framework for a more substantive debate on gun control by forcing regulators to ask the question, 'what is this regulation limiting and how would that affect a gunman's ability to perform a mass shooting?'

Agent-based Model's Limitation: The Need for Behavior model Recovery

The previous sections demonstrate how zero-intelligence agent-based models can be used to recreate emergent phenomena (e.g., the Flash Crash) and test policy decisions before implementation. As illustrated, agent-based models can be used in different domains to allow decision-makers to gain a better understanding of their complex environment. However, while the previous models received academic acceptance, achieving over 60 citations between the two modes, the models made negligible impacts on decision-makers or society. It is the authors contention that this is because the models lack sufficient validation that limits the applicability of their findings, resulting in the slow adoption by decision-makers [14], [15].

The validation issues experience by the majority of agent-based models have led to the slow adoption and application of agent-based models by real-world decision-makers [71]. To increase the validity and acceptance of agent-based models, researchers must be transparent and statistically validate the outcomes of their models. However, that alone is insufficient. Since agent-based modeling is a bottom-up framework, we assert that both the existence of agents and their behavior models must be empirically derived from observations of real-world actors.

The next chapter presents a Size-First Hierarchical Clustering, a new method for autonomously deriving classes of agents from observations of real-world actors. Autonomously clustering agents helps to validate the model and reduces research bias that can occur. However, real-world datasets contain outliers, which makes clustering challenging. The next chapter explores these issues and compares Size-First Hierarchical Clustering to other traditional clustering techniques.

Chapter 3: Comparison of Agglomerative Hierarchical Clustering of a Behavior Model Dataset with Outliers

Chapter three explores Size-First Hierarchical clustering, a new autonomous clustering technique, as a means of empirically deriving agents' classes. Thereby validating the existence of a class of agents and mitigating research bias that can occur when developing a simulation.

Many agent-based models are made up of agents that are minor variations of the same strategy [26], [62]. The behavior models of agents often require tuning parameters, so the model's outcome matches real-world observations [29], [57]. In the previously presented financial agent-based model we tuned the frequency, placement, and cancellation rate of orders for each class of agents, such that they matched observed rates [38]. However, in [38] as in other agent-based models, this requires the researchers to cluster agents into groups, which is generally done by hand or supervised clustering, leading agents to be clustered in predetermined strategy groups. This methodology makes several assumptions, which the researcher could not find discussed in the agent-based literature:

1. it assumes that the researcher has accurately described all "significant" classes of agents found in the real-world,
2. by extension it assumes that all "significant" real-world agents fit into the researcher's defined classes,
3. any real-world agent or class not modeled would not significantly change the emergent outcomes of the simulation.

This model framework is biased by a researcher's preconceived notion of the complex system and is paradoxical to objective science. Under this framework, agent-based models are made up of agents whose behavior model is perceived to be known or easily described and not necessarily reflective of real-

world agents. Lastly, since this framework does not recover behavior models from real-world actors, it can only assert that the real-world actors are using one of many potential behavior models that could result in the observed emergent property. Heath et al. [12] demonstrate this by using an animal reproduction model to produce the same emergent properties found in single server queuing system by mapping reproduction rates and lifespan to key queue parameters. While this example is extreme, it illustrates the point that many different behavior models can result in the same emergent properties.

Therefore, a more academically rigorous methodology is to recover strategies (i.e., agent classes) from observational datasets autonomously. This methodology mitigates the bias of the researcher. It also explicitly results in agents' whose state-action pairs are similar to observed real-world actors, improving the validity of the model because both the agents' behavior model and the emergent properties are shown to mimic reality.

Strategies can be represented as clusters of behavior models. A traditional method for representing a behavior model is a Markov process. In a Markov process the current environmental state is represented by a set of features S. At each time step, an agent selects an action A from a set of possible actions. An agent is assumed to be maximizing the long-term value of its reward function R. The state that the agent transitions to, after selecting an action in the current state, is governed by a transition probability P. The Markov process allows an agent's behavior model to be expressed as the policy which optimizes the tuple $M = (S, A, P, \gamma, R)$, where γ is a reward discount factor.

Some real-world actors will apply strategies that are rarely seen. These rarely seen strategies can be considered outliers in the dataset. Clustering a dataset with outliers is a challenge, as the addition of an outlier can result in otherwise different objects being clustered together [23]. One option is to identify and remove the outlier from the clustering process and classify the outliers at the end of the clustering process. However, this can be computationally expensive for large dataset. Instead, we examined

different agglomerative hierarchical clustering methods, to assess how each one performed on a dataset containing outlying behavior models. The next section provides an overview of clustering, which is followed by a discussion of the experimental dataset. We use a simple 2-dimensional dataset, as well as a more complex dataset made from game replays of StarCraft: Broodwar, a real-time strategy game. We then layout the experiment and the results. The final section of the chapter is a discussion of the results.

Background on Clustering Methodologies

By leveraging the mature field of autonomous clustering, agent-based models can mitigate researcher bias in the model development process. Autonomous clustering attempts to define the underlying structure of a dataset with minimal human intervention [74]. It accomplishes this by locating data points within a dataset that have a natural homogeneity. The collection of similar data points is known as a cluster [74]. Intuitively, data points within a cluster are more similar to each other than they are to data points in a different cluster [75].

Agglomerative Hierarchical Clustering

Autonomous clustering can be segmented into two variations agglomerative hierarchical clustering and iterative partitional algorithms [74], [75]. Iterative partitional clustering separates the dataset by optimizing a criterion function that is defined locally or globally. Partitional clustering is affected by outliers, as many algorithms use the squared error criterion. In other words, iterative partitional clustering algorithms have high plasticity because the clusters produce by the algorithm can be altered by the addition of a few outliers. Agglomerative hierarchical clustering produces more stable clusters, which is why we focus on different versions of agglomerative hierarchical clustering for this study [75].

Agglomerative hierarchical clustering takes a set of objects and clusters them into mutually exclusive groups, each having members that are as much alike as possible. This clustering is accomplished

by reducing the number of groups from *n* to *n-1*, in a manner that minimizes the loss of information from the group. The process is repeated until the number of clusters is reduced from *n* to *1* [76].

Hierarchical clustering iteratively groups together a set of objects $O = \{o_1, o_2 \ldots, o_N\}$, until one cluster remains containing all objects in the set. Each object is made up of features $o_1 = \{f_1, f_2, \ldots, f_x\}$. The distance between each pair of objects is calculated using the object's features and stored in an NxN matrix, known as a similarity matrix. The method for calculating similarity depends on the domain and the type of data the features are made up of (e.g., Numerical, categorical, ordinal). However, the Euclidean distance is most often used and is the technique used in this dissertation. The pseudocode for hierarchical clustering is presented below.

Algorithm: Pseudocode for Agglomerative Hierarchical Clustering

1) $c \in C$, where C is the set
2) of all clusters
3) $O \in c$, where O is the set of all objects in cluster c
4) S is equal to the number of $o \in o$, where o is an object in O
5) N is the number of objects being clustered
6) At $T = 0$, the number of $c \in C$ is equal to N, i.e., initially, every object is placed in its own cluster
7) c_i is combined with c_j, where c_j has $\wedge D_{ij}\ \forall\ c \in C,\ i \neq j$ (ties are broken randomly), combine c_i with the closest cluster c_j
8) Repeat step 6 until all clusters are combined into a single cluster

The term D_{ij} in the pseudocode above represents the intracluster distance between cluster *i* and *j*. The method for defining intracluster distance is left up to the researcher and is strongly domain dependent [77]. Different intercluster distance measures will result in different clusters [75]. A class of

widely used intercluster distance measures is known as generalized mean operator, and the equation is shown below.

$$G_\alpha(d_1, d_2, \dots, d_N) = \left(\frac{1}{N}\sum_{k=1}^{N} d_k^\alpha\right)^{1/\alpha}$$

In the equation above the set of $\{d_1, d_2, \dots, d_N\}$ is the collection of all the pairwise distances between objects in cluster *i* and objects in cluster *j*. The α is selected by the researcher and determines the mean operator version that is used in the classification. Commonly selected α values are listed below:

$$G_\alpha(d_1, d_2, \dots, d_N) = MIN[d_k] \quad \alpha \to -\infty \quad Min\ Operator\ (Sinlge\ Link)$$

$$G_\alpha(d_1, d_2, \dots, d_N) = \frac{N}{\frac{1}{d_1}+\frac{1}{d_2}+\cdots+\frac{1}{d_N}} \quad \alpha \to -1 \quad Harmonic\ Mean$$

$$G_\alpha(d_1, d_2, \dots, d_N) = \frac{d_1 + d_2 + \cdots + d_N}{N} \quad \alpha \to 1 \quad Arithmetic\ Mean$$

$$G_\alpha(d_1, d_2, \dots, d_N) = \frac{(d_1^2 + d_2^2 + \cdots + d_N^2)^{1/2}}{N} \quad \alpha \to 2 \quad Euclidian\ Mean$$

$$G_\alpha(d_1, d_2, \dots, d_N) = Max[d_k] \quad \alpha \to \infty \quad Max\ Operator\ (Complete\ Link)$$

To explore how changing α affects the cluster results let us recall $d_{i,j}$ as the intracluster distance between cluster *i* & *j*. It is important to note that cluster *i* & *j* are disjointed subsets, $i \cap j$ [78]. Now let's assume that *i* and *i'* are two clusters such that $i \subset i'$, then for any *j*

$$Min(i, j) \geq Min(i', j) \ \ and\ conversly\ Max(i, j) \leq Max(i', j).$$

Therefore, if α is set to $-\infty$ the larger cluster of *i'* will be selected for merging with cluster *j*. Setting the $\alpha = -\infty$ is known as Single Link clustering and tends to favor clustering large groups together. Single Link clustering will usually result in several large clusters and a few small clusters. Single Link clustering has an inherent chaining effect where it produces elongated clusters that stretch through the feature space [79].

By contrast, complete link clustering occurs when $\alpha = \infty$. Under the complete link framework, the smaller cluster of *i* will be selected for merging with cluster *j*, illustrating that smaller clusters are more likely to be considered the closest cluster to its neighbor. Complete link clustering generally results in uniform size clusters that are isotropic in shape [80].

By varying α a researcher can indirectly control the size and shape of the resulting clusters. In practice, single and complete link are the most often used α values, with each being the optimal selection depending on the application [81].

While most hierarchical clustering the utilizes generalized mean operator method, it is not the only intercluster distance measure. A popular distance measure utilizes fuzzy logic, which dynamically changes the intercluster distance measure. Let *V* be the cardinality of cluster *i + j*. Using cardinality, we can make clusters that meet 'fuzzy' constraints, such as wanting many clusters or at least X clusters. An example of a fuzzy distance measure is below [78]:

$$if\ V\ is\ A\ then\ d_{ij} = Min[d_k],$$

$$if\ V\ is\ B\ then\ d_{ij} = Max[d_k].$$

The above distance metric states that if the union of cluster *i & j* will result in a cluster with *A* elements then the intercluster distance is the minimum distance between an object in cluster *i* and an object in cluster *j*. If the union will result in a cluster with *B* elements, then intercluster distance is the maximum distances between objects in cluster *i & j*.

To perform the case where we want X clusters of about equal size, we first use the transform function U = V/N that turns V into a proportion of the total number of objects N. We also define a fuzzy operator below:

$$E(r) = 1 \quad for\ r \leq \frac{1}{X}$$

$$E(r) = \frac{p - r}{p - \frac{1}{X}} \quad for \ \frac{1}{X} \leq r \leq p$$

$$E(r) = 0 \quad for \ r \geq p$$

Using the fuzzy subset, we can create a fuzzy intercluster distance method

$$d_{ij} = E\left(U_{ij}\right) Min[d_k] + \left(1 - E\left(U_{ij}\right)\right) Max[d_k].$$

If the union the union of two clusters will result in a single cluster having a proportional size greater than *p* then the Max intercluster distance measure will be used. As discussed earlier, when using the Maximum method smaller clusters are more likely to have a closer distance to their neighbor. Therefore, this acts as a regulator, preventing clusters from growing too large. The value *p-1/x* can be thought of as the tolerance in cluster size, such that all clusters are "about" the same size.

The previous intercluster distance measures use local information to determine the distance between each cluster and its neighbor. Context clustering takes into account not only the distance between two clusters but also the effects of surrounding neighbors [82].

Look at figure 26, in both scenarios B is closer to A. However, in scenario 2 there are three other clusters that are closer to A than B. When taking this into account it may be more appropriate to first cluster B with C, instead of clustering B with A. To achieve this outcome, we can use mutual neighbor distance [83].

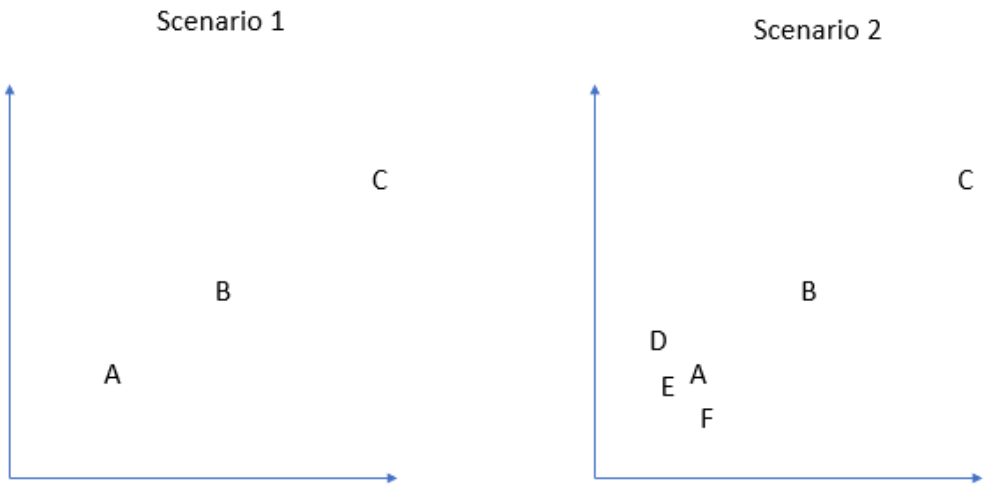


Figure 26 Mutual Neighbor Distance (Figure originally found in [75]*)*

Mutual neighbor distance is a method for assessing how close two clusters are in relation to all other neighboring clusters. The formula for mutual neighbor distance is [75]

$$MND(X_iX_j) = NN(X_iX_j) + NN(X_jX_i).$$

The function NN is called the neighbor number, and $NN(X_iX_j)$ is the rank X_j receives when comparing how close all clusters are to X_i. The NN(A,B) in scenario one is 1, but in scenario two the value is 4. Therefore, in scenario 2 clusters *B & C* (MND=3) is closer than clusters *A & B* (MND =5). As illustrated, agglomerative hierarchical clustering is flexible and can be adapted to many different dataset structures.

Clustering with outliers – a review

There is no single, agreed-upon definition of an outlier. An outlier is researcher, domain, and dataset-specific. For one dimensional data, an outlier is often considered to be any data point falling above or below 1.5 times the interquartile range. As the dimensionality of the dataset increases so does the difficulty of defining and efficiently detecting outliers. Knox and Ng defined an outlier as a data point that has less than X neighbors within a Y distance [84].

Knox and Ng propose a uniformly global measure of an outlier. However, Figure 27 illustrates a circumstance where a uniform global measure would either label a significant amount of the datapoints as outliers or fail to identify the red data point as significantly different than its closely compact neighbors.

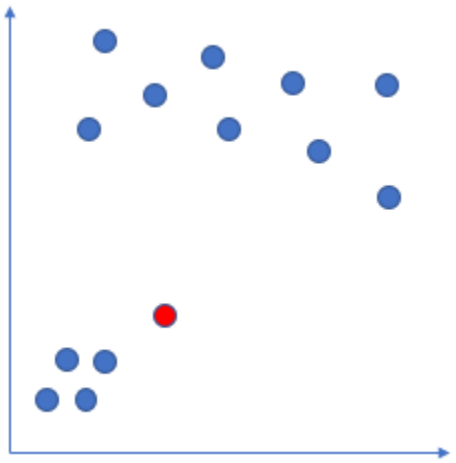


Figure 27 Local Outlier

Breunig et al. assert that Figure 27 represents a local outlier and that these outliers exist in many real-world datasets [85]. A local outlier is an object that is isolated from its surrounding neighbors but may still be within the overall dataset. Breunig proposes a method which compares the density of a datapoints neighbor to the distance between the data point and its neighbors [85].

The research into outlier identification is extensive and will not be covered in this review. Instead, we will focus on methods for clustering given the dataset contains outlier. Ester et al. [86] proposed using a density-based algorithm, known as density based scanning (DB-SCAN). Intuitively a cluster in the dataset occurs when a subset of data points is more densely located than the surrounding data points. Taking advantage of this Ester et al. present a notion of density, which is used to locate natural clusters in the dataset.

They first define the notion of an Eps-Neighborhood (NEps) for a point *p*, which is all the points *q* whose distance from *p* is less than or equal to Eps. One approach would be to state that a data point is

part of a dense cluster subset if it has a minimum number of points (MinPts) within its Eps-Neighborhood. However, this criterion fails because there are points which are on the edge of the cluster that would not meet this definition.

To address this issue Ester et al. define the notion of directly density-reachable, density-reachable, and density-connected. A point (*q*) is directly density-reachable from *p* if it meets the following to conditions:

1. $q \in N_{Eps}(p)$, q is a point in p Eps-Neighborhood and
2. $|N_{Eps}(q)| \geq$ MinPts, q contains at least the minimum number of points within its Eps-Neighborhood.

A point (p_1) is considered density-reachable to (p_2) if there is a chain of points that are directly density-reachable from p_1 to p_2. Lastly, the point p_1 is density-connected to p_2 if there is a third point p_3, which both p_1 and p_2 are density-reachable from p_3. The figure below illustrates these concepts.

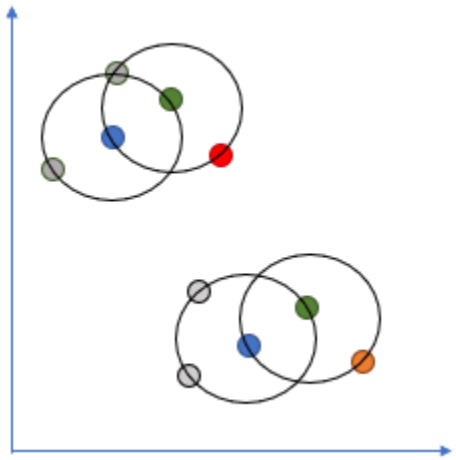


Figure 28 Density Example

For the sake of the example let's state the black circles represent the Eps distance and MinPts = 3. In that case, the blue and red points in the upper left-hand corner of Figure 28 meet the definition of density-connected because the green point is directly density-reachable by both red and blue. However,

the orange and blue points in the lower right-hand corner are not density-connected because the green point only has 2 points within its Eps-Neighborhood, which means it does not qualify as a direct density-reachable point.

Using this formulation Ester et al. [86] defines a cluster as all points in the dataset that are density-connected. To avoid having to iterate over outlier clusters Ester et al. recommend preprocessing the data to detect outliers. They propose using K-distance, which is the distance between a point and its K^{th} farthest neighbor. All points in the dataset are then sorted in the descending order of their k-distance value. The first inflection point seen when graphing the sorted order is considered the threshold, see figure 29.

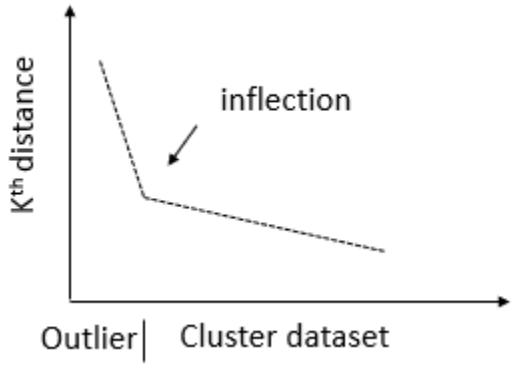


Figure 29 K^{th} Distance Ranking

All points below the inflection point are outliers and removed from the dataset before clustering. At the end of the clustering process, all outliers and points not clustered can be added to clusters using a classification method, such as k-nearest neighbor. This methodology allows the clustering of non-isotropic shapes of data points, such as a ring.

Karypis et al. proposed another method known as Chameleon [87], which can cluster a dataset containing outliers, without explicitly treating outliers differently. Uniquely, the Chameleon algorithm uses both iterative portioning and agglomerative hierarchical clustering. The initial starting point is a complete nearest neighbor graph of the entire dataset.

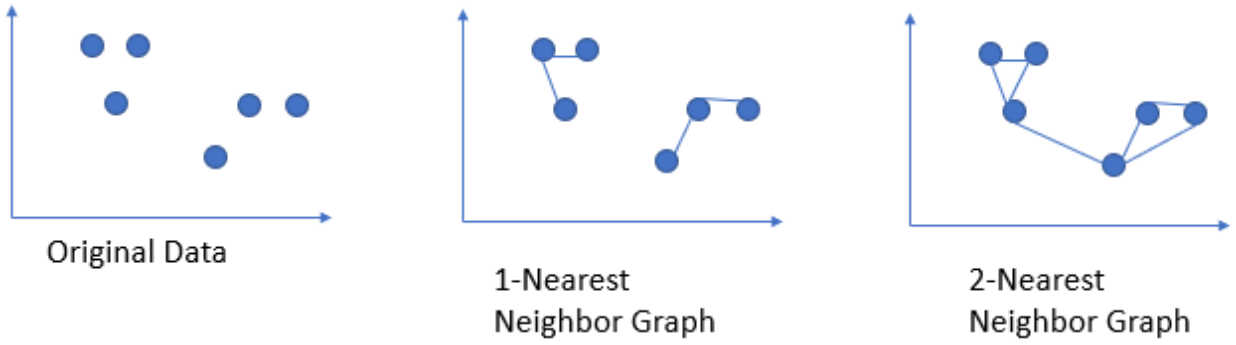


Figure 30 Nearest Neighbor Graph

Figure 30 illustrates the idea of the nearest neighbor graph, in which each data point is connected to its k-nearest neighbors. A connected graph occurs when there exists a path between each data point all other data points in the dataset. Starting from a connected graph, the hMETIS algorithm is used to iteratively split the graph into smaller and smaller clusters [88]. The largest cluster of all subcluster is selected, and hMETIS finds the minimum amount of edge cuts required to sperate the cluster into two separate clusters, each containing at least 25% of the data points in the original cluster. The process continues until the largest cluster is less than or equal to a user-defined minimum size, Karypis et al. recommended 1 – 5% of the overall data points.

Once the hMETIS algorithm has completed segmenting the dataset, Chameleon uses an agglomerative hierarchical clustering algorithm to combine the sub clusters. Only The clusters that have inter-connectivity greater than a user defined metric (T_{RI}) are eligible to be clustered. Inter-connectivity is defined as the sum of edge cuts required to split the cluster into approximately two equal parts. Therefore, the edge cuts required to split the cluster the more interconnected the cluster is.

Of the clusters that have an inter-connectivity > TRI the two clusters that have a relative closeness (RC) higher than a user-defined parameter (T_{rc}) are clustered together. If multiple combinations of clusters meet both criteria than the two clusters with the highest relative closeness score are clustered together. The equation below defines relative closeness:

$$RC(C_i, C_j) = \frac{\bar{S}EC_{(C_i,C_j)}}{\frac{|C_i|}{|C_i + C_j|}\bar{S}EC_{(C_i,)} + \frac{|C_j|}{|C_i + C_j|}\bar{S}EC_{(C_j)}}.$$

The value $\bar{S}EC_{(C_i, C_j)}$ is the average distance between a point in cluster C_i and one in C_j. This is normalized using the measure $\bar{S}EC_{(C_i,)}$ & $\bar{S}EC_{(C_j)}$ which is the average edge distance that is cut when bisecting the cluster C_i & C_j respectively into two equal size sub-clusters. Chameleon is halted when no combination of clusters has an inter-connectivity > T_{RI} and a relative closeness score > T_{rc}.

Guha et al. developed yet another method known as CURE [89], which randomly samples a subset of the data points and clusters that subset, allowing CURE to run faster on large datasets. CURE starts with first picking a random sample of size S from the data set. All data points in size S start out in their own cluster. The two closest clusters are merged together using the simple link distance measure. A representative set of 'well scattered' data points are kept for the newly merged clusters; the rest of the data points are removed from the dataset. To calculate the new representative data points, the flowing pseudocode is used.

1. Select farthest point P_1 from the clusters (C_i) mean
2. While REPsize < θ
    a. Select P_i to be farthest point from P_{i-1} left in C_i

The points selected become the representative points for cluster C_i all other points not selected are removed from the dataset. The points making up the representative cluster are shifted toward the mean calculated in step 1 by α percent. This shrinks the effects of outliers, as they will be moved more than points closer to the cluster's mean. This process continues until K clusters remain in the dataset. Then all points previously removed from the dataset can be classified using the K-nearest neighbor algorithm.

CURE handles outliers in multiple ways. Firstly, there is some chance outliers are removed from the dataset during the random sampling phase. Secondly, the effects of outliers are diminished because all points are moved towards the clusters mean by α percent. Outliers will move further than a point closer to the mean. Additionally, CURE implements an outlier removal process. Once the current number of clusters drops below 1/3rd of the original number of clusters all clusters made up of 1 or 2 data points are eliminated from the dataset. This outlier removal is again performed just before reaching K clusters.

The last algorithm presented in this review was created by Almeida et al. and provides an automatic method for detecting and removing outliers in a dataset [90]. Almeida et al. leverage ideas proposed in the DBSCAN algorithm [86] to develop the automatic outlier detection and elimination seen below.

1. Set iteration counter j = 1
2. Repeat until number of discarded objects = 0
    a. Calculate $\bar{d}_j$
    b. Set R = 4($\bar{d}_j$)
    c. Calculate $\bar{c}_j$(R)
    d. Discard objects i if c_i<1/3($\bar{c}_j$)
    e. Increase j
3. Repeat process with R = 2($\bar{d}_j$)

In the algorithm above $\bar{d}_j$ is the average nearest neighbor distance and $\bar{c}_j$(R) is the average number of neighbors found within R distance from a data point. By removing data points that have fewer than 1/3rd of the average neighbors within R distance we are removing points that are not close to a considerable number of objects.

Simple link agglomerative hierarchical clustering is then used to cluster all data points into a single cluster. The 1st and 3rd quartile is calculated for the squared length of the links found in the complete graph. Any link whose square length is found to be larger the six times the interquartile range is cut, segmenting the dataset into natural clusters. At this point, all removed outliers can be classified using K-nearest neighbor. Like DBSCAN, CHAMELEON, and CURE this algorithm was shown to be able to classify non-isotropic cluster, as seen in the Figure below.

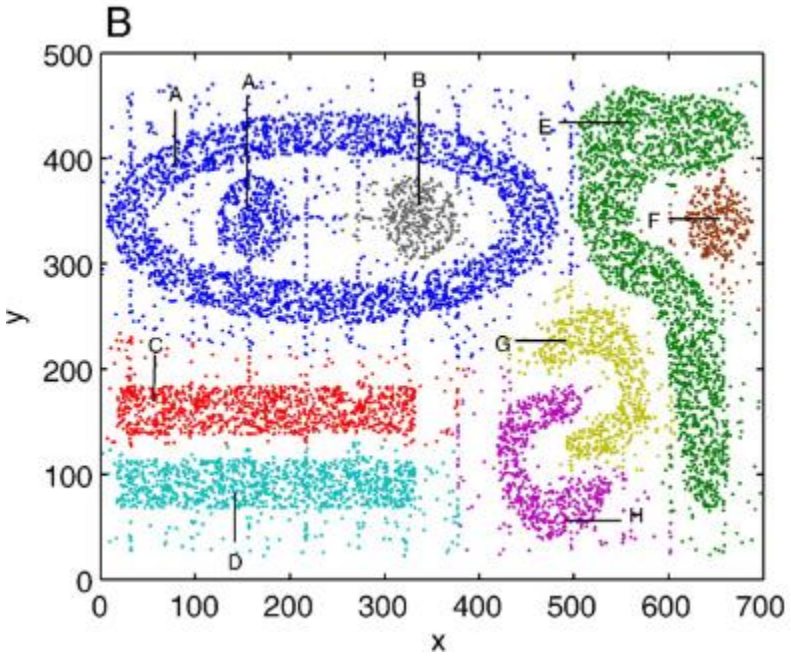


Figure 31 Clustering Result presented in [90]

The benefit of Almeida et al. algorithm is that it does not require input from a practitioner. Given a sufficiently high dimensional state space, it can be challenging to select the correct parameters or know how many natural clusters exist, leading researchers to repeatedly run the clustering algorithm with new parameters until the outcome matches their intuition. This is in part because there is no universal definition of a "good" cluster because it is application specific.

We compare DB-Scan and Almeida et al. algorithm to an algorithm we developed known as Size-First hierarchical clustering. We hypothesize that while these algorithms can find non-isotropic clusters a

more straightforward algorithm can be used to identify clusters of behavior models. While we expect to see outliers, we believe that in at least some behavior model data sets the natural clusters will be isotropic. Taking advantage of this structure a more straightforward algorithm is presented.

Size-First Hierarchical Clustering

The previous section discussed different agglomerative hierarchical clustering methods, which were created to cluster datasets with outliers. However, CURE requires the practitioner specifying a specific number of clusters. When clustering behavior models, this leads to bias in the strategy discovery process. Domain experts may have a preconceived notion of how many strategies exist in a dataset and therefore could extract that exact number of clusters.

DBSCAN and CHAMELEON do not suffer from the same bias as CURE but still require user-specified parameters. Depending on the parameters both DBSCAN and CHAMELEON can return different cluster size. Figure 33 below illustrates DBSCAN returning two different cluster results based on user-selected parameters. The figure below is initially from [87] and illustrates how DBSCAN is sensitive to parameter selection. In a sufficiently high dimensional space a priori it is difficult to select good parameters.

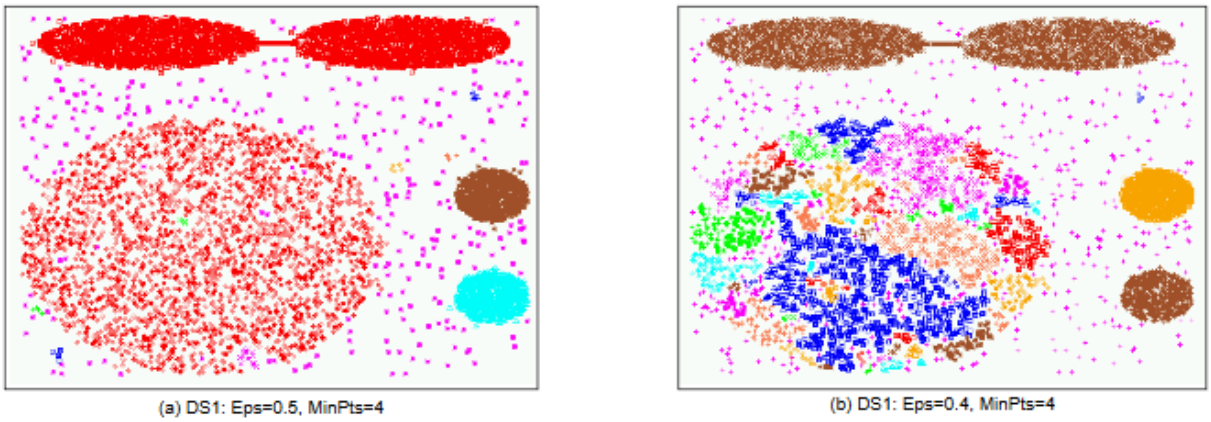


Figure 32 DB Scan Returning Different Results from [87]

Therefore, we have modified the traditional hierarchical clustering algorithm to consider cluster size explicitly, which has the effect of clustering outliers with their nearest clusters and de-conflicting distinct strategies that would have been clustered together because of the existence of an outlier.

Almeida et al. address both these issues by developing an algorithm that requires no user input. While their algorithm is capable of clustering non-isotropic clusters, it requires several extra steps. Firstly, outliers are required to be removed through an iterative process of calculating the average nearest neighbor distance and removing data points that have a small set of neighbors within that distance. After which a connected graph is constructed using the hierarchical single link method. The square of each link must then be calculated, and the interquartile range determined, which is used to separate the complete graph into clusters. Lastly, the outliers are classified using k-nearest neighbors.

While this method is robust, for large datasets the added steps could require significantly more calculations. We hypothesize that some behavior model datasets contain isotropic clusters. For these applications, a more straightforward algorithm we developed called Size-First Hierarchical clustering can be used without significant loss in cluster quality.

Traditional agglomerative hierarchical clustering groups objects successively based on their dissimilarities. Similar objects are clustered early in the process, and dissimilar clusters are grouped later in the process. The dendrogram of traditional hierarchical clustering in Figure 2 illustrates this process. Let's assume that objects in Figure 2 are clustered based on their distance measure to one another. Using the traditional method object B & C and D & E would be clustered together. For the sake of demonstrating let's now assume both the minimum and maximum distance between a combined cluster of B & C is closer to D & E than object A. Under the traditional method, B & C would be clustered with D & E, leaving object A without a cluster until the final level.

Traditional hierarchical simple clustering can lead to clusters that contain only a small number of objects. In the previous example, object *A* is not clustered until the final level, which indicates that it is an outlier. Traditional hierarchical clustering of build orders creates a few large clusters and many small clusters, indicating that most people follow similar strategies.

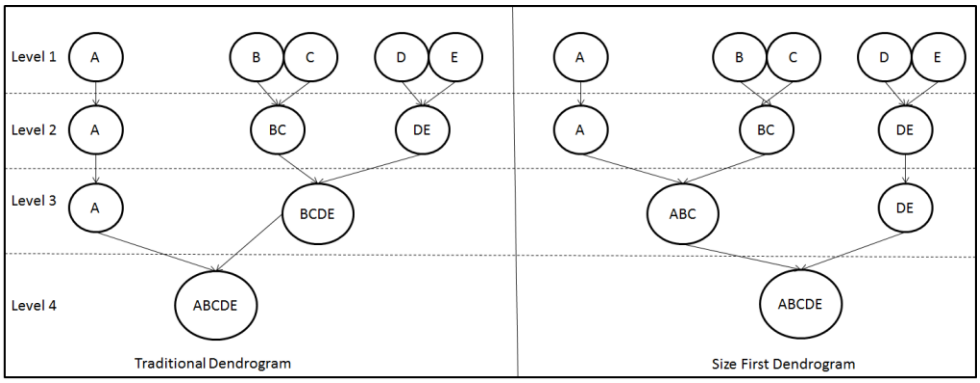


Figure 33 Traditional vs. Size-first Hierarchical Clustering

In the size-first clustering method, the smallest size cluster is grouped with its nearest neighbor. Figure 33 object A is clustered with objects B and C in level 3. This methodology eliminates small outlying clusters while maintaining differentiation between otherwise distinct clusters. To the best of our knowledge, this is the first use of size-first hierarchical clustering in an academic paper. The pseudocode is presented below.

Algorithm: Pseudocode for Size-first Hierarchical Clustering

1) $c \in C$, where C is the set of all clusters
2) $B \in c$, where B is the set of all build order in cluster c
3) S is equal to the number of $b \in B$, where b is a build order in B
4) N is the number of build orders being clustered
5) At $T = 0$, the number of $c \in C$ is equal to N, i.e., initially, every build order is placed in its own cluster

6) c_i is the cluster with *min(S) ∀ c ϵ C, i.e.,* c_i is the cluster with the smallest *S*

7) c_i is combined with c_j, where c_j has $\wedge D_{ij}$ $\forall c \epsilon C, i \neq j$ (ties are broken randomly), combine c_i with the closest cluster c_j

8) If multiple clusters tie for the smallest *S*, then, among those, the clusters with $\wedge D_{ij}$ (closest neighboring cluster) is clustered first (if ties remain, they are broken randomly)

9) Repeat step 6-8 until all clusters are combined into a single cluster

Our size-first hierarchical clustering will terminate once it reaches a single cluster, like all other hierarchical clustering algorithms. To limit the expert knowledge required we present a criterion used to determine the significant branching (clustering) level.

Selection Criteria for the Appropriate Number of Clusters

Most clustering techniques require human intervention to determine the appropriate number of clusters, typically represented by *K*. As stated previously, this can introduce human bias. Furthermore, many clustering techniques do not discern whether the clusters reflect a structure that is present in the datasets [91]. Such clustering methods will always define *K* clusters regardless of the natural structure of the dataset.

Using a methodology similar to silhouettes [91], we apply the average inter-cluster distance divided by the average intra-clusters distance in this study. This measure compares how tight clusters are with how close they are to one another. The cluster number that minimizes the modified silhouettes value is assumed to be the natural cluster set. The pseudocode to calculate this measure is as follows:

Algorithm: Average Inter-cluster Distance Divided by Average Intra-cluster Distance

1) $c \epsilon C$, where *C* is the set of all clusters and c is a specific cluster

2) **Calculate Inter-cluster Distance:**

3) $\forall\ c \in C$ calculate D_c, where D_c is the average distance between build orders within cluster c

4) Calculate D_{inter}, i.e., the average of all D_c

5) **Calculate Intra-cluster Distance:**

6) Calculate $D_{ci,cj}$, where $D_{ci,cj}$ is the average distance between a build order in c_i and one in $c_j, i \neq j$

7) Calculate D_{intra}, i.e., the average of all $D_{ci,cj}$

8) Calculate D_{inter}/D_{intra}

The number of clusters that minimizes the value of D_{inter}/D_{intra} is assumed to be the natural number of clusters in the dataset. However, using this methodology on size-first hierarchical clusters results in many small clusters because size first clustering forces outliers to be grouped into clusters, and clustering outliers increase the value of D_c for a cluster.

The goal of size-first clustering is to eliminate small outlying clusters, which allows for better analysis of behavior model groups. Therefore, we impose a secondary size restriction. For *K* clusters to be an acceptable solution, the size of the smallest cluster must be greater than *X*, where *X* is a size value determined by an expert. This is similar, to an expert picking the maximum number of possible clusters. Note that the selected clustering solution will be that with the smallest D_{inter}/Di_{ntra} value that also has no clusters smaller than *X*. Meaning size-first hierarchical clustering will return between 2 – 10 for an X = 10% of the original dataset. Size-first clustering limits the expert bias as the algorithm will seek to find the most 'natural cluster' representation within the expert's constraint and may return fewer clusters than the maximum allotted.

Size-First Hierarchical clustering does not require ancillary steps to identify and remove clusters, allowing it to run faster than Almeida et al. and DB-SCAN algorithm. Additionally, it limits expert bias by having the expert define the minimum size of the clusters, instead of the exact number of clusters to be

returned. The algorithm then finds the most natural cluster representation within the expert's constraint. We compare Size-First Hierarchical Clustering to Almeida et al. and DB-Scan algorithm within a simple clustering dataset and StarCraft Brood War, a real-time strategy game. The next section describes a simple experiment comparing size-first hierarchical clustering to both DB-Scan and Almeida et al.'s algorithm.

Simple 2-Dimensional Clustering Comparison

The previously discussed outlier clustering techniques were tested on continuous space environment, that had densely packed cluster nuclei, as seen in figure 34.

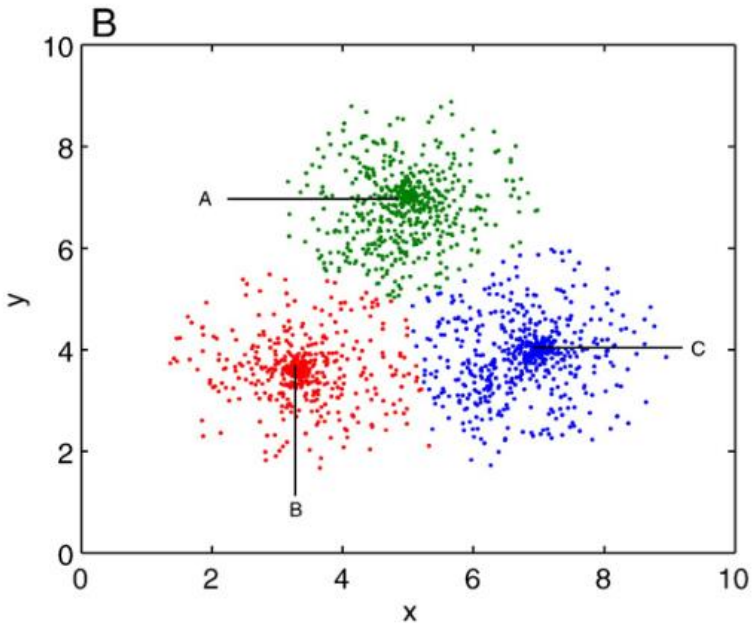


Figure 34 Almeida et al. Clustering presented in [90]

We hypothesized Almeida's algorithm would struggle to accurately cluster datasets that contain overlapping points, which is more likely to occur in integer datasets. Therefore, we created the data set seen in figure 35 below.

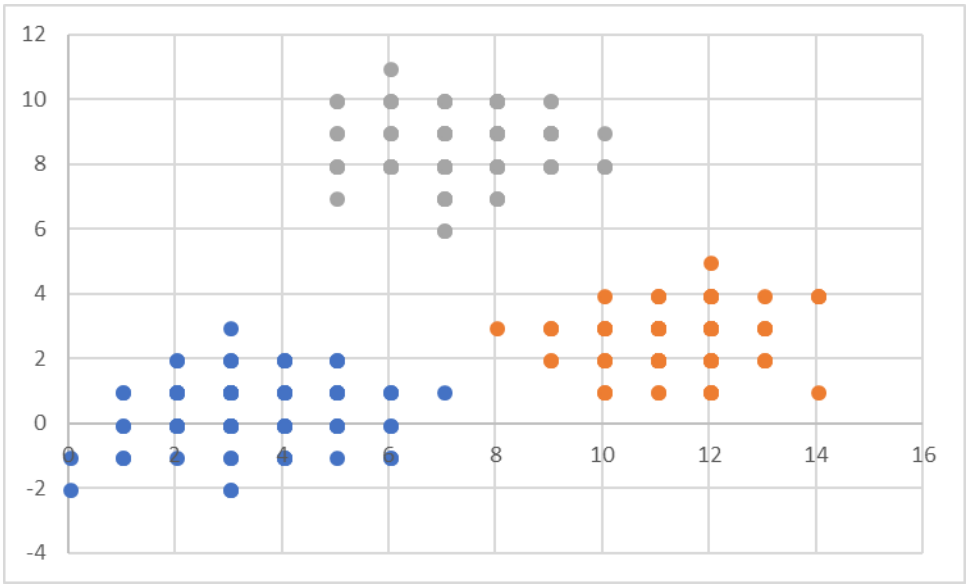


Figure 35 Integer Dataset with Overlapping Points

In the dataset, each cluster is made up of 100 points but only about 22 unique values. Visually it is clear to see that there are three clusters in the dataset. However, when Almeida et al.'s algorithm is applied, it returns 42 separate clusters, as seen in figure 36. Almeida's algorithm cuts links between any objects that are greater than 6×(IQR). However, because there are so many data points that are overlapping the IQR equals 0. Therefore, any points that are not overlapping are considered separate clusters. Outliers that were removed in the earlier process are clustered using K-nearest neighbor classification, where K = 3. The result is the cacophony of clusters seen in figure 36.

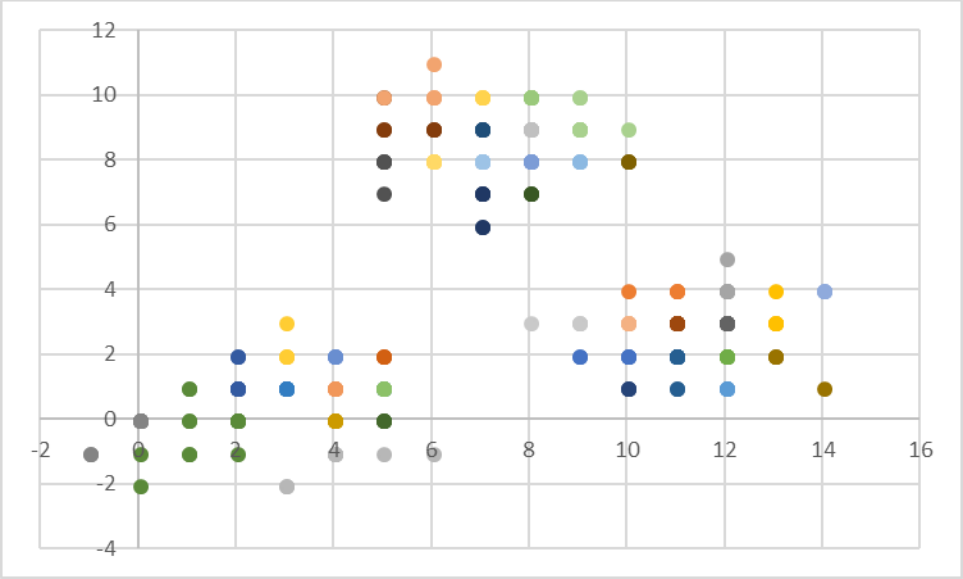


Figure 36 Alemida et al.'s clustering

Depending on the parameters selected both DB-SCAN and Size-First can return the correct number of clusters. Both Algorithm results are susceptible to parameter selection. For example, Size-First will return 7 clusters if the minimum size is set to 10%, 4 clusters if the size is set at 15%, and 3 if the size is set to 20%.

DB-SCAN, on the other hand, is significantly more sensitive to Eps-distance. Figure 37 illustrates the DB-SCAN k-distance for K = 4. The first inflection point occurs at 1.415, but some people may determine that 1 is a better inflection point. Remember any object with a K-Distance greater than the cutoff is removed as an outlier and then classified later. Table 3 shows the number of clusters is affected by these parameter selections.

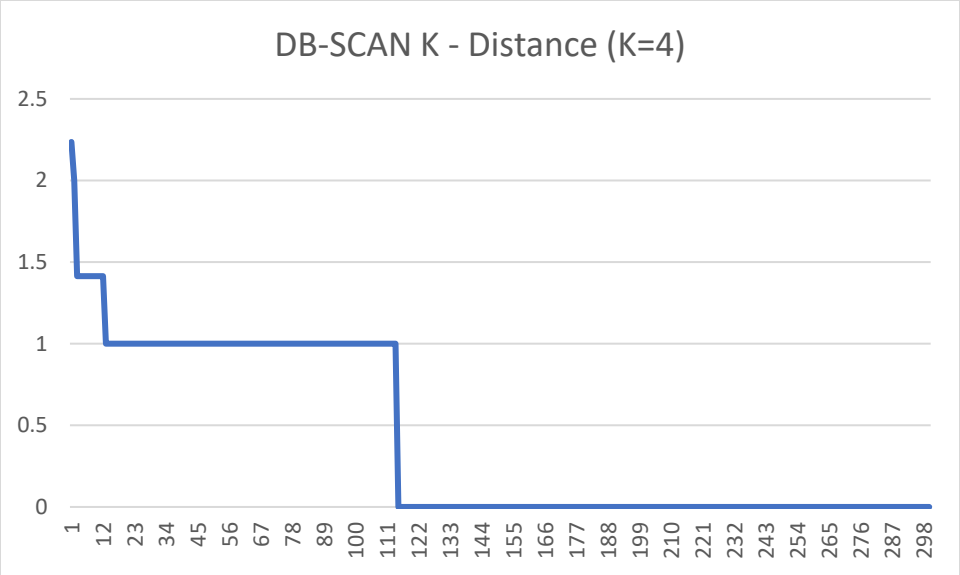


Figure 37 DB Scan K - Distance

Table 4 DB-SCAN Number of Clusters

Outlier Distance	Eps-Distance	Min Points	Number Clusters
1.415	1.5	4	3
1	1.5	4	3
1.415	1	4	38

This simple experiment shows that Almeida's algorithm can struggle with integer base datasets. Additionally, it illustrates DB-SCAN is affected by the Eps-Distance. Size-First Hierarchical clustering is also affected by its single parameter selection, but it is not as sensitive. With promising findings, we move on to a more complex dataset utilizing StarCraft Brood War. The next section will give an overview of StarCraft Brood War, as well as behavior model recovery research that has been performed in StarCraft.

Background StarCraft Brood War

We tested game replays from StarCraft: Brood War, which is a real-time strategy (RTS) game based on the popular StarCraft game released in 1998. A game replay captures all actions taken by players;

thus, the entire game can be replayed and analyzed. In StarCraft, players develop and support an army with the goal of defeating their opponent's army. Players perform actions in real time, as opposed to turn-based decision-making in classic games such as Chess or Go.

Partially due to its complexity, StarCraft has become a popular test-bed for AI research [92]. Players collect a finite amount of two in-game resources, minerals and gas, which are used to pay for the construction of units and buildings. At the beginning of the game, players can only create a small subset of all the possible units and buildings. Additional unit and building types are unlocked as a player constructs prerequisite buildings and researches technology. Units unlocked later in the game are stronger than units that are available at the beginning of the game. However, late-game units are significantly more expensive.

In this setup, players face a resource-constrained optimization problem. Spending resources on a building or unit early in the game can prevent a player from constructing one later. Each player's solution to this problem is known as the build order (i.e., the order in which a player construct units and buildings). For this research, we consider the build order as a representation of a player's strategy (i.e., behavior model) because it determines the composition of the player's army at given time. For example, a player may choose to build a large number of weak/cheap, early game units or wait to build a smaller number of powerful/expensive, late game units.

The composition of the army gives clues about the player's intention. An army composed of early game units will likely attack its opponent often, preventing them from gathering the necessary resources required to build late game units. To make the game more interesting, StarCraft is made up of three playable factions: Terran, Zerg, and Protoss. Each faction has specialized units, which results in different army composition (strategies) being used depending on the match-up (e.g., ZvP or TvZ).

The primary motivation of this project is to test our hypothesis that for some behavior model datasets Size-First Hierarchical clustering can outperform DB-SCAN and Almeida et al. without requiring the outlier mitigation steps. However, an additional motivation for this work is that to date, no artificial intelligence (AI) system for real-time strategy (RTS) games has defeated any professional players [93] because RTS AI systems struggle to adapt to an opponent's strategy and tactics. In this paper, we define a strategy as the composition of a player's army. Currently, advanced RTS AI systems classify opponents into strategies defined by an "expert". However, experts can be biased towards well-known and easily described strategies that are not necessarily strategically valuable [94]–[96].

For this study, we utilized the dataset constructed by Synnaeve and Bessière [97]. They examined StarCraft: Brood War game replays and split each replay into three files that are easier to analyze. This paper uses only one of those files, which specifies the time when buildings/units are created and destroyed. Using Synnaeve and Bessière's [82] dataset, we examined both when and the order in which a player constructed units and buildings. Size-First Hierarchical clustering algorithm is compared to Almeida et al., with both algorithms used to group players with similar build orders into clusters. A cluster is a group of players that trained units and constructed buildings in a similar order (e.g., training marines before constructing a tank) and at similar instances in the game. The next section goes into a review of previous research performed on real-time strategy games.

Artificial Intelligence Research in Real-Time Strategy Games

In 2003, Michael Buro [98] asserted the need for AI research in the area or Real-Time Strategy games. He identified that Real-Time Strategy games contain several AI problems. Listed below are the types of AI problems Buro identified.

1. **Resource Management**. As stated previously players face a resource constrained optimization problem. Training units and constructing buildings deplete limited resources. The AI must also identify the resources needed and gather them.

2. **Decision Making Under Uncertainty**. To increase the difficulty of these games, the players' knowledge of the playing map is restricted to a small area around their own military assets. Therefore, players must make decisions without full information about their opponent.
3. **Spatial and Temporal Reasoning**. Using natural features such as choke points to your advantage and waiting to launch attacks at the optimal time.
4. **Collaboration**. Coordinating with other AI bots and human players to aid in the overall objective of defeating an opponent.
5. **Adversarial Real-Time Planning**. Determining and taking the necessary actions to defeat an adversary.
6. **Opponent Modeling**. Determining an opponent's behavior model (e.g. strategy).

To help the reader place this work in context, we present notable works in the area of opponent modeling. Accurate opponent modeling provides a player with greater ability to achieve a reward that is greater than the theoretical optimal. For example, in rock-paper-scissors, the optimal solution is a random policy evenly distributed over rock, paper, and scissors. However, if the opponent only selects rock, then the optimal solution is to select paper [99]. This simple example illustrates the need for opponent modeling.

Bakkes et al. [100] developed an opponent-modeling framework with features that broadly represent players as defined by an expert. In it, opponents are clustered into groups based on expert-defined features. Bakkes et al. compared the win percentages of their RTS AI bot with and without opponent-modeling augmentation and found that opponent modeling increased the win percentage by 10%.

To incorporate uncertainty into opponent modeling, Synnaeve and Bessière [101] used a Bayesian framework for opponent modeling. Experts were used to classify historical games into strategies. Like the

model developed by Bakkes et al., expert features represent players' build order. The Bayesian framework compared the current opponent's construction schedule to that of a test set. As the opponent performs more actions, the system updates its prediction of the opponent's strategy. Synnaeve and Bessière found that they could match the expert's classification 60-70% of the time, depending on which faction was being classified.

Hsieh and Sun [102] represented strategies as trajectories through a lattice, whereby the nodes represented different units, buildings, and technologies. This methodology deviates from the approaches mentioned above because researchers need not select their classification features. This method discovers popular trajectories, but similar strategies are not clustered. Potentially, hundreds of minor variations of each strategy could be identified.

Leece and Jhala [103] recently explored the use of Markov random fields (MRF) to predict the current composition of an opponent's army. The difference between this approach and that developed by Hsieh and Sun is that MRFs are undirected graphical nodes; consequently, the game state could return to a previously visited node. Leece and Jhala's methodology was found to improve previously published results when their predictions were compared to the baseline prediction of the average unit count across the training set, which is a standard set by Hostetler et al. [104]. However, it did not resolve the issue of minor variations of each strategy being classified as their own strategy.

Dereszynski et al. [94] improved on the work of Hsieh and Sun [102] by developing a hidden Markov model that allowed for the prediction of future player actions. Here, players' strategies are modeled as trajectories through a Markov state space, and each state in the hidden Markov model corresponds to the probability of constructing a given building. Dereszynski et al. were able to predict which units an opponent will train using the Hidden Markov mode transition matrix.

Stanescu and Čertickỳ [105] proposed using answer set programming to predict unit production. This methodology differs from those in the aforementioned studies because it expressly uses game mechanics, such as unit cost. Utilizing game replays, Stanescu and Čertickỳ then selected the most probable unit combination for a given game phase from the remaining possible combinations. They found that their predictions were reliable up to three minutes into the future.

Two notable studies using hierarchical clustering to classify an opponent's strategy. Drachen et al. [106] used hierarchical clustering to cluster Tomb Raider: Underworld players into four high-level player types. These types roughly explained player behavior throughout the game. Yasui et al. [107] used hierarchical clustering to identify strategies in RoboCup soccer games. In their study, a strategy was defined as the team's deployment pattern and the initial trajectory of the ball. The next section explains the experimental methodology.

Experimental Methodology

Hierarchical clustering requires a measure of similarity between the clustered objects. Here, we employ military composition (i.e., how many of each unit and buildings a player possesses), which is a natural way to describe a player's army and limits the use of expert-crafted features.

The military composition is a time-dependent measure. Players can construct buildings and train units in different orders but still create the same military composition. For example, in Figure 1 player one trains two soldiers in time period 1 (T_1) and constructs a vehicle in the following time period (i.e., T_2). The other player trains and constructs the same units but in reverse order. At the end of T_2, both players have the same army composition built in different orders. This simple example demonstrates how military composition can vary with time.

To account for army composition over time, we divide the game into 30-second intervals. This methodology has been used by Derezynski et al. [94] and Synnaeve and Bessière [101]. This approach

allows us to examine both the time and order in which units are trained and buildings are constructed. We summarize each interval *t* using a cumulative observation vector, O_t= (M_1,…, M_U), where *U* is the total number of distinct military units and buildings and M_u is the current number of military unit or building of type *u* that exist at the end of time interval *t*. Units trained and buildings constructed during time *t* are added to the cumulative sum. Buildings and units destroyed during time *t* are subtracted from the sum.

The 30 second time interval will affect the number of clusters that are created. The order of construction within each time interval is obscured. Therefore, if the time interval is increased the clustering algorithm will lose information and making it harder to discern between similar but distinct strategies. However, decreasing the time interval leads the algorithm to classify insignificant variations in a strategy as distinct and increases the processing time. Derezynski et al. [94] identified 30 seconds as an optimal time, which the authors adopted for consistency amongst research methods.

As shown in Figure 38, player 1 has two marines at the end of *t = 1,* and the cumulative observation vector is O_1 = (2_{Marine},…,$0_{hellion}$). At the end of T_2, player one has constructed a hellion, making the cumulative observation vector O_2 = (2_{Marine},…,$1_{hellion}$). This methodology provides a snapshot of a player's military composition every 30 seconds.

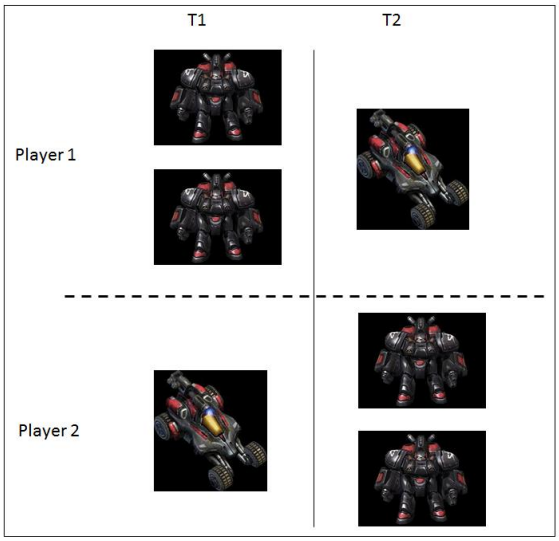


Figure 38 Example Build Order.

Hierarchical clustering uses a single value that measures the dissimilarity between two objects being clustered. However, we measure military composition every 30 seconds; therefore, we calculate the difference between two players every 30 seconds and sum the total difference for all *t* in time period *T*, where *T* is the maximum number of examined time periods. The following equation expresses the similarity measure used in this study:

$$D_{i,j} = \sum_{t=1}^{t=T} \left| O_{i,t} - O_{j,t} \right|, i \neq j.$$

The equation above measures the dissimilarity between two distinct players' (i & j) build orders. The absolute value of the difference between each unit and building is summed for time period t, which is then summed over the total number of time periods T.

Calculating similarity using this process allows deviations in build orders to be considered. A dissimilarity of 0 means that both players had the same military composition at the end of each 30-second time interval. We assume unit training and building construction is indicative of the player's strategy. Therefore, it is important to calculate both build order similarity over time, as well as the difference in final army composition.

Similarity is calculated between all players in the same faction (i.e., Terran, Zerg, and Protoss), thereby creating a symmetric distance matrix. Note that we opted to use absolute values and not the squared difference, which was done deliberately because we did not want to increase the dissimilarity between two players artificially. For example, squaring the difference in Figure 1 would give $D_{t=1} = (2 - 0)^2 + (0 - 1)^2$, thereby resulting in an overall difference of $D_{1,2} = 5$.

Results and Discussion

We have chosen the dataset constructed by Synnaeve and Bessière [97], which has 7,647 games. The dataset provides a file for each game that specifies when a building/unit is created or destroyed. This work leverages those files. Additionally, the authors selected this data set because most research has been historically performed on StarCraft: Brood War, allowing this paper to add to a rich literature set.

In this case, matches with more than two players were eliminated from the dataset because of potential player cooperation, which could significantly change their behaviors. (In future work, we might examine how cooperation changes strategies.) In this study, another stipulation is that the games lasted for at least five minutes. We also assumed that each player in a match has independent build order, meaning the build order of their opponent does not affect their own.

We expect that as each game progresses players' behaviors to diverge into different strategies. Therefore, the number of strategies (i.e., clusters) increase as time progresses. To examine how clusters, evolve over the course of a game, we re-cluster replays every minute, starting at 5 minutes and continuing as far as 11 minutes (note that not all games last for 11 minutes). The number of build orders over time is shown in Table 1. The average length of a game in the dataset is 16 minutes meaning most games last longer than 11 minutes. The maximum number of build orders occur at the 5-minute mark due to games ending before the next re-clustering period. As shown in the table, the number of build orders investigated at 5 minutes is 14,785, while the number investigated at 11 minutes is 13,869.

Table 5 Number of Build Orders Over Time

Race	5 min	6 min	7 min	8 min	9 min	10 min	11 min
Terran	5,060	5,043	5,007	4,971	4,922	4,869	4,809
Zerg	4,466	4,436	4,385	4,326	4,258	4,187	4,109
Protoss	5,259	5,248	5,197	5,148	5,076	5,013	4,951

The upper time limit was selected to be 11 min because nearly 100% of games feature combat by this time (i.e., the destruction of a unit or building by an opponent), which is the same methodology Dereszynski et al. [94] used. They justified this decision by stating, "… in the early game, players execute their strategies in relative isolation, whereas later in the game, actions are increasingly dictated by tactical considerations such as the composition of the opponent's army or the outcomes of key battles." Once a player experiences combat, they may drastically change their strategy (build order), thereby confounding the classification algorithm. Limiting the data to 11 minutes allows us to study player strategies before largescale battles.

Figure 39 is a dendrogram of the Terran build orders used in this study. The red boxes highlight outlying clusters. Note how small these clusters are compared with the other branches. Also, like in the previous fictitious example, one build order is not clustered until the last connection, which motivates this study.

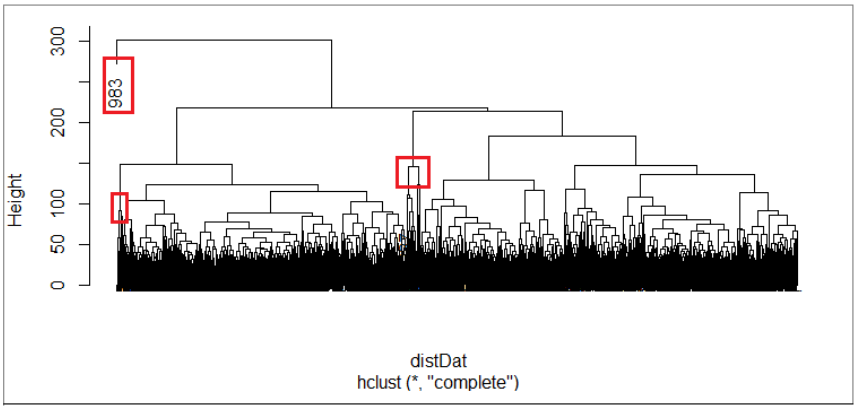


Figure 39 Dendrogram of Terran Build Order

We first examine Almeida et al.'s algorithm, and the number of strategy clusters per each faction can be seen below. Using the threshold $6 \times (IQR)$, resulted in the algorithm in many cases returning a single cluster, meaning the algorithm detected a single strategy for Protoss players 11 minutes into the game. By examining previous research and interviews with professional StarCraft players, we know that there is more than a single strategy. For example, [108] used an expert to defined six strategies for each faction and group the replays into these six strategies. Almeida et al.'s algorithm recovering one strategy means the dataset is made up of loosely packed clusters that are relatively close to each other, which Almeida's algorithm is failing to detect.

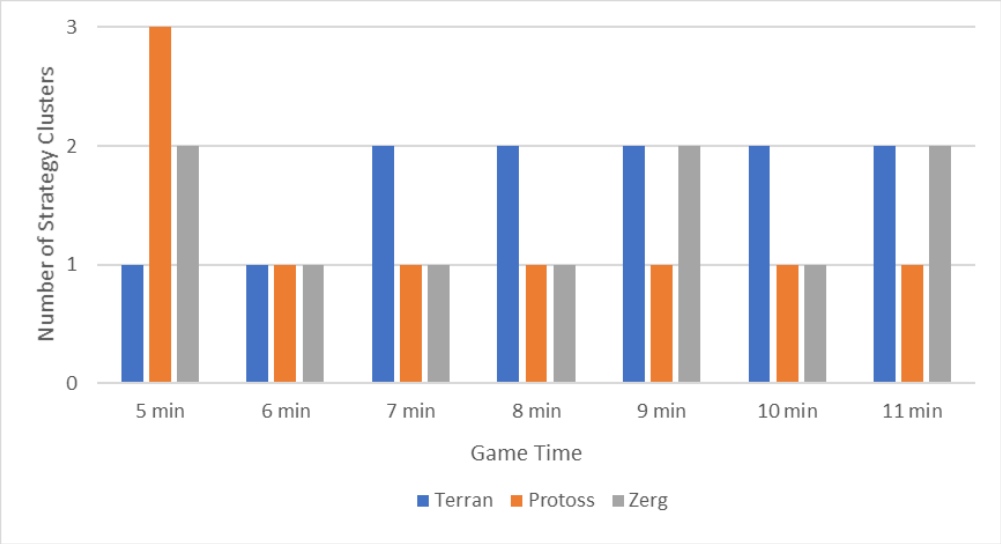


Figure 40 Almeida's Clustering Results

Lowering the threshold resulted in more clusters but the cluster results still contradict previous research and professional players' statements. For example, lower the threshold to $4.5 \times (IQR)$ resulted in 7 strategies being identified for Terran players at 11 minutes. However, two out of the seven clusters comprised 99% of the observations. The algorithm determined that a player can have a mechanized based army (e.g., planes and tanks) or an infantry-based army (e.g., marines). While this is true it misses the nuances of army composition, which makes up individual strategies. Almeida's algorithm is unable to identify clusters that are relatively close to each other.

DB-SCAN also has issues identifying multiple strategies in the dataset. Figure 40 shows the sorted K-Distance for Terran players at 5 minutes. For this study K was set to 4, meaning figure 40 shows the sorted distances between each player and their 4th closest neighbor. The red circle is where the cutoff was selected. Any points above the circle were considered outliers and removed from the dataset. The outliers were later classified using K-nearest neighbor, where we used the 3 closest neighbors.

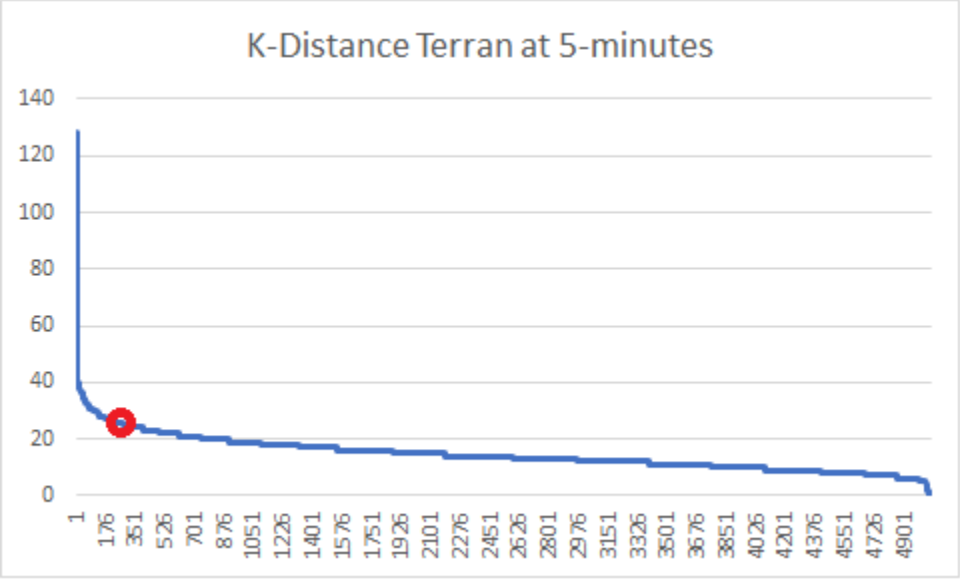


Figure 41 K-Distance Terran's at 5 minutes

The Eps distance was set to the outlier cutoff distance and the MinPts was set to 4. DB-SCAN gives no method for identifying the proper Eps distance or MinPts. Therefore, we used the outlier cutoff which was shown in the previous experiment to be sufficient. DB-SCAN identifies significantly more clusters than Almeida earlier in the game. However, at 5, 6, & 7 minutes there is one cluster that made up of at least 90% of the observations. Essentially, DB-SCAN identifies a single strategy and a small number of outlying strategies, which is inconsistent with previous research.

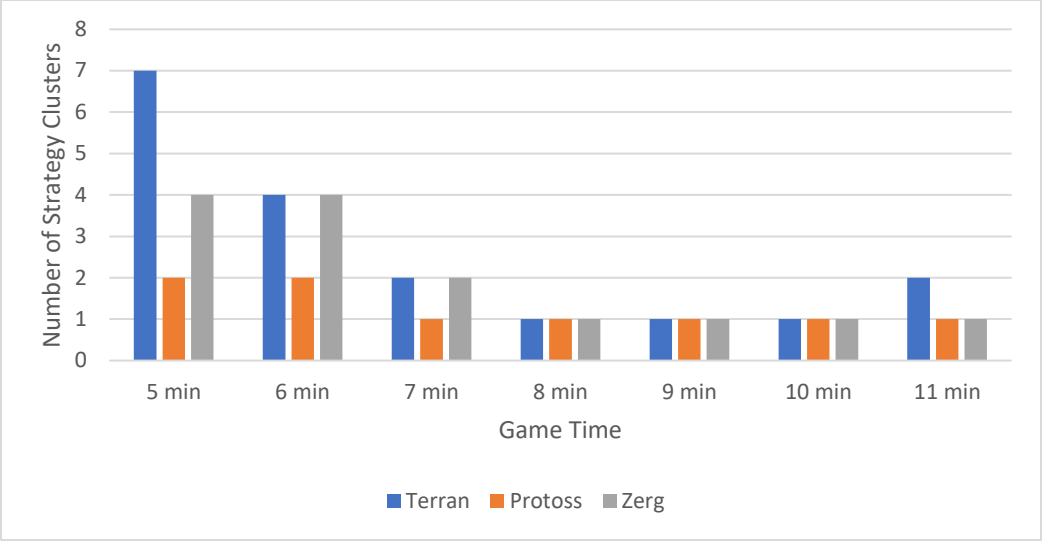


Figure 42 DB-SCAN Clustering Result

Seeing disappointing results from Almeida and DB-SCAN, we applied size-first hierarchical clustering. In Figure 43, the minimum cluster size is defined as 10% of the total number of build orders. It is shown in this figure that the number of clusters (i.e., strategies) increases as game time increases and that the number of clusters is time-dependent. Players take divergent paths only to converge and perform similar actions during the remaining examined period.

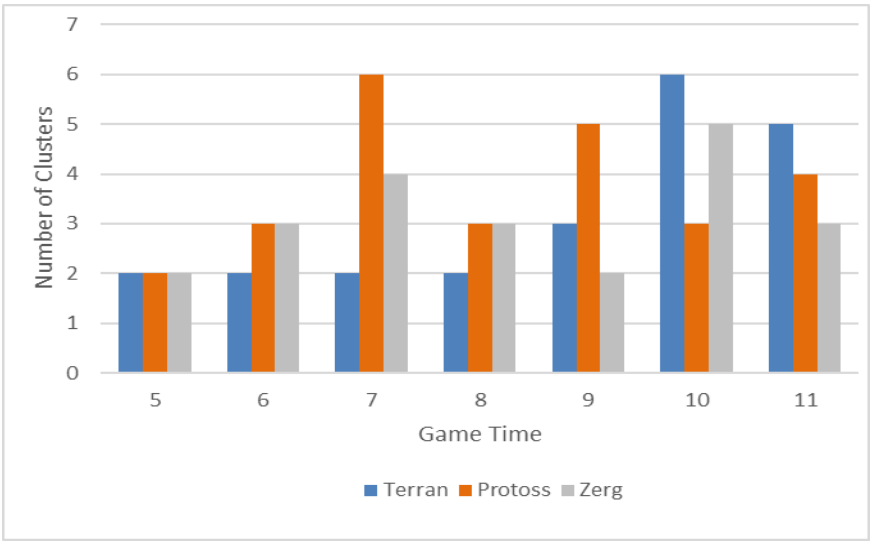


Figure 43 Cluster Count with a 10% Size Requirement

It is shown in Figure 44 how the number of clusters varies relative to the minimum cluster size. There is a significant change as the minimum size is raised from 5% to 10% of the total number of build orders. However, there is a much smaller change from 10% to 15%. A less restrictive size requirement will identify clusters that are smaller and more tightly compact. However, this may make it more challenging to discern strategic differences between clusters because several clusters are likely to be minor variations of the same strategy. Unless otherwise stated, the remaining experiments use a 10% size restriction, which provides cluster counts that are comparable to those in previously published papers.

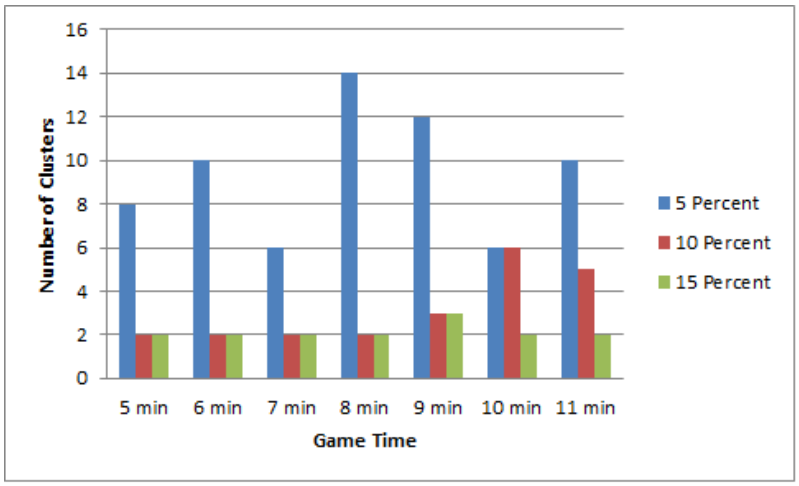


Figure 44 Terran Cluster Count by Minimum-size Requirement

While the clustering results are more in line with previous research, we have not proved our clusters are 'good.' We found no agreed-upon measure to judge the strategic value of clusters. We did not want to use experts to judge our clusters because that could lead to bias owing to expert knowledge and opinion, which could be flawed. The goal of this analysis is to produce unsupervised clusters that represent distinct strategies. Thus, we define success as creating clusters that identify strategies that meet the following criteria.

1) **Clusters are different.** The composition of each cluster's army should differ, or the build order should be distinctly different.

2) **Strategies can be used to identify the opponent's faction.** Knowing a player's strategy should allow us to identify their opponent's faction.

3) **Strategies have different likelihoods of winning.** The likelihood of winning changes based on the clusters of the two players.

Qualitative Differences in Strategies

An effective clustering algorithm should identify strategies that utilize different build orders or result in distinct military compositions. Using principal component analysis (PCA), we can visualize the differences among clusters. PCA reduces high-dimensional data to a smaller number of variables by taking correlated variables and transforming them into non-correlated linear variables. In this study, PCA was employed independently for each faction. The entire observation vector O for all build orders of a given faction was input to the PCA algorithm, resulting in more than 700 variables for each faction (i.e., Terran, Zerg, and Protoss). However, PCA allows the entire observation vector O of each player to be represented by three variables.

Figure 6 contains plots of all the build orders for each faction and games lasting at least 11 minutes against their first three PCA values identified by the R statistical software. Note that strategies found via size-first hierarchical clustering are given unique colors. The graphs in Figure 6 shows that the clusters have little overlap, indicating that the strategies found have different build orders.

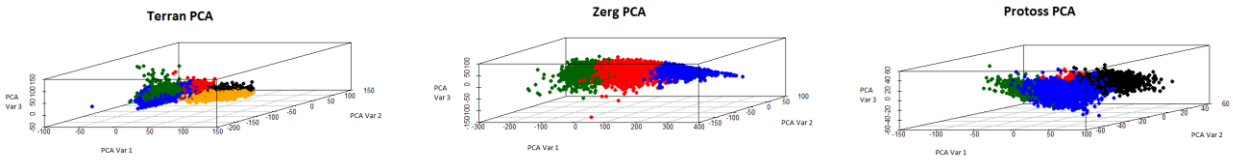


Figure 45 PCA Scatter Plot by Faction

The variables with the highest eigenvalue occur 7 minutes or later into the game and tend to be variables focused on the existence of late-game technology/units. For example, Terran's variable with the highest eigenvalue is the number of battlecruisers that exist between 10 minutes and 10 minutes 30

seconds. Terran's that field a battlecruiser this early have foregone early game defenses and rushed to create this late-game technology. Successfully fielding a battlecruiser this early in the game will generally lead to a Terran victory. For Zerg, the variable with the highest eigenvalue is the number of Ultralisk Caverns between 7 minutes 30 seconds and 8 minutes. An Ultralisk Cavern allows for the creation of the late game Ultralisk unit and is equivalent to the Terran investing in battlecruiser technology. Lastly, Protoss's variable with the highest eigenvalue is the number of gateways existing between 10 minutes 30 seconds and 11 minutes, which determine how many units Protoss can create. The number of gateways is an indication of how much Protoss is investing in mid and early game units versus late game technology. As seen in the following sections a discerning difference between strategies is the investment a player makes in late game technology.

Terran strategies were clustered into five groups, as shown in Table 5. Two of these strategies focus on training marines, while the other three concentrate on creating mechanized units. Strategies 1, 2, and 3 build tanks and focus less on infantry units. These strategies differ in the order that buildings are constructed. Strategy 4 has fewer units than strategy 5, which indicates that these players have experienced early game combat.

Table 6 Average Terran Military Composition At 11 Min

	# of Marines	**# of Medics**	**# of Tanks**	**# of Wraith**	**# of Barracks**	**# of Factories**	**# of Academy**	**# of Machine Shops**	**# of Starports**
Strategy 1	3.1	0.12	0.99	0.12	1.13	1.29	0.18	0.87	0.26
Strategy 2	2.97	0.07	1.2	0.07	1.06	1.33	0.12	0.95	0.21
Strategy 3	2.86	0.03	1.19	0.05	1.07	1.33	0.09	0.99	0.20
Strategy 4	6.18	0.58	0.34	0.07	1.68	0.6	0.59	0.35	0.18
Strategy 5	6.89	0.72	0.14	0.08	1.84	0.42	0.71	0.19	0.16

The information shown in Table 5 provides further insight into the intent of the strategies. Strategy 5 has a more substantial investment in barracks, with most Strategy 5 players constructing at least two barracks. Additionally, Strategy 5 players construct the most academies, which allow for upgrades to infantry units, indicating that Strategy 5 players will seek to attack with an infantry-based

military. Most Strategy 5 players did not construct a factory by 11 minutes. Factories enable access to late game technology; the lack of their construction indicates that Strategy 5 players are 'rushing' to build an army and engage in early game aggression. The goal among Strategy 5 players is to attack an opponent before they have martialed a defense. Strategy 4 is also an infantry-based military, but players are diversifying their ploys by adding more tanks. Strategy 4 players will attack slightly later than Strategy 5 players, but with a more diverse and powerful military.

The first three strategies focus on a mechanized military, with players constructing factories. That is an indication that these players will be seeking to invest in late game technology. Additionally, most players in these strategies construct machine shops, which enable them to upgrade their siege tanks. However, strategy 1 players are similar to Strategy 4 players, in that they are constructing a balance military composition. Additionally, Strategy 1 players will likely attack earlier than Strategy 2 and Strategy 3 players, but later than Strategy 4 players. Strategies 2 and 3 are the most similar, but they differ regarding the order by which players construct military assets.

Qualitatively, the PCA graphs indicate that each strategy is distinct owing to the limited cluster overlap. It is demonstrated in Table 5 that the final army composition of the clusters is different, which indicates players are implementing different strategies.

The goal of this sub-section is to demonstrate that the proposed size-first hierarchical cluster method provides clusters with little overlap and distinct military compositions. In the next subsection, we highlight the strategic value of the strategies by using them to predict an opponent's faction.

Predicting Opponent Faction

Each faction in StarCraft has specialized characteristics, which leads players to select strategies based on their opponent's faction [109]. Therefore, we chose to examine player strategy over the course

of a game by observing how a player transitions between the clusters represented in Figure 4. We can use this information to determine the opponent's faction.

To test this hypothesis, we used a Bayesian inference model for predicting the opponent's faction. Equation 2 is Bayes' theorem. Here, *P* (F) is the a priori probability that the opponent is of a given faction (i.e., Terran, Zerg, or Protoss). *P* ($X \rightarrow Y$) is the probability of the player transitioning from cluster X at t_1 to cluster Y at t_2, where t_2 is one minute later than t_1. *P* ($X \rightarrow Y \mid F$) is the probability that, given your opponent is of faction F, you will choose to transition from cluster X to Y over the next time period. This equation results in the probability that an opponent is of faction F given the transition from cluster X to Y.

$$P(F|X \rightarrow Y) = \frac{P(X \rightarrow Y|F)\ P(F)}{P(X \rightarrow Y)}$$

The Bayesian inference model is trained for each faction separately and is used to predict the opponent's faction in each match. The model is trained using 90% of the matches (which are selected randomly from all the data), and it predicts the opponent's faction in the remaining matches. Initially, we assume that the opponent is equally likely to belong to all three factions. After every minute, we update the probability of the opponent being a given faction based on the player's transition between strategy clusters. After the final transition at 11 min, the probabilities and the actual opponent's faction are stored. Note that this process is repeated 1,000 times to ensure robust results.

The data from the 1,000 trials have been used to create the receiver operating characteristics (ROC) curves shown in Figure 7. ROC curves compare the true and false positive rates of a prediction. If the area under the curves (AUC) is greater than 0.5 (marked by the solid black line), the prediction is better than random chance.

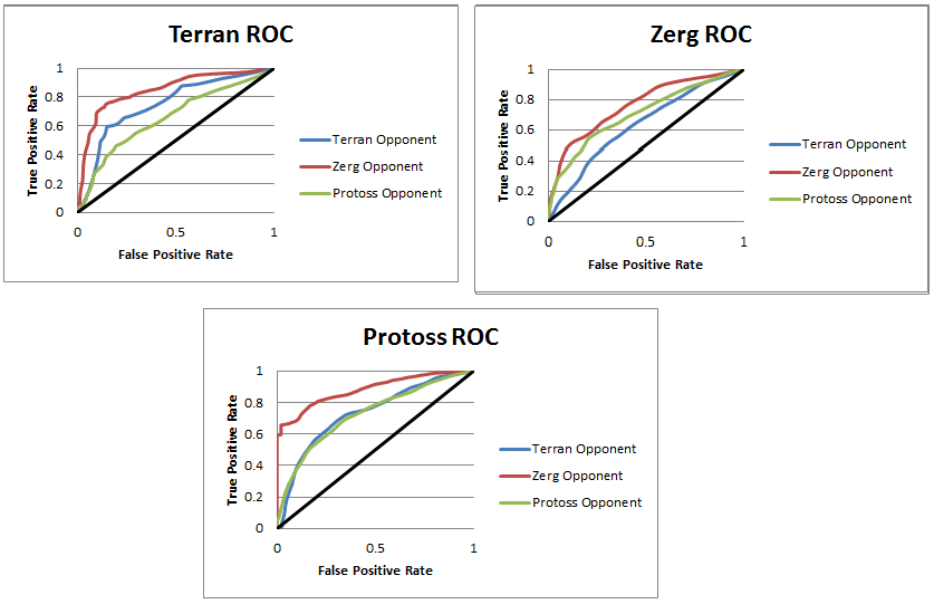


Figure 46 ROC Curve

The closer the AUC is to 1, the more likely the model will correctly predict the opponent's faction. Notice that Zerg opponents consistently have the highest AUC for each player type, meaning that the Zerg faction is the easiest to identify. Earlier aggression is required to defeat the Zerg faction, and therefore their opponent must always prepare for early combat.

Table 7 Area Under ROC Curve

	Terran Player	**Zerg Player**	**Protoss Player**
Terran Opponent	0.749	0.632	0.730
Zerg Opponent	0.844	0.765	0.882
Protoss Opponent	0.657	0.706	0.725

This methodology can be used to identify if strategic decisions are well balanced for different classes of characters. StarCraft uses a multi-faction methodology, with each class having different units, powers, and strategies. Therefore, it should be expected that players of one faction will use a specific strategy against another. However, as shown in Table 6, it is consistently easier to identify when a player is playing against a Zerg opponent, regardless of that player's faction, meaning that players playing against Zerg use a unique strategy, which they do not use on any other faction. This might limit the enjoyment of the game as it will become monotonous if all players believe there is only one way to beat an opponent.

This methodology can be used to quickly determine if players are implementing a myriad of strategies against each other or relying on a single basic strategy. Game developers can then update the game to ensure that the game is not repetitive. This methodology can be directly applied to any game that uses a build order structure, such as StarCraft 2, DOTA, or League of Legends.

The goal of this sub-section is to illustrate the strategic value of the clusters. As hypothesized, the strategy that players select is an indication of the opponent's faction. It is shown in Figure 7 and Table 3 that, regardless of the player's faction, their cluster trajectory can be used to estimate the opponent's faction, and so the clusters found, therefore, have a certain degree of strategic value.

Game Outcomes Are Affected by Strategy Selection

In our final experiment, we examined the strategic value of the clusters. As mentioned above, all players in a cluster are assumed to follow a similar strategy, with the exception to this assumption occurring where the clusters overlap. As demonstrated in the previous sub-section, strategy selection is based in part on the opponent's faction. We hypothesize that the likelihood of winning is affected by the strategy of both players.

The numbers of each type of match-up in the dataset are listed in Table 7; for this, we examined only matches that lasted for at least 11 minutes.

Table 8 Number of Match-ups

TvT	431
TvZ	1571
TvP	1891
ZvZ	171
ZvP	1908
PvP	406

In this experiment, we explored how strategy selection affects a player's likelihood of winning. The first player to leave a game is assumed to forfeit and therefore accept defeat. If no player leaves the

match, then the player with the highest number of combat units at the end of the game is declared the winner.

The frequency at which a Terran strategy is used against specific factions is detailed in Table 8. Since all players are professional, we can assume that the employed strategy has been optimized against their opponent's faction. For example, when a Terran player selects Strategy 1 52% of the time, their opponent belongs to the Zerg faction. As stated in the previous subsection, Strategy 1 focuses on late-game mechanized units, such as siege tanks. Since most players use Strategy 1 against Zerg opponents, we infer that mechanized units are more effective against the Zerg faction than against the Protoss faction. Strategy 5 focuses on producing early-game marines. 91% of the opponents against whom Strategy 5 is used belong to the Protoss faction, which indicates that Protoss players are vulnerable to early game harassment by marines.

Table 9 Terran Strategy Selection by Opponent Faction

Strategy	Terran	Zerg	Protoss
Strategy 1	34.4%	52.2%	13.5%
Strategy 2	27.7%	66.8%	5.5%
Strategy 3	20.7%	78.9%	0.4%
Strategy 4	3.6%	23.4%	73.0%
Strategy 5	0.1%	8.5%	91.4%

For this arrangement, we divide player-strategy selection into two categories: conventional, and unconventional. Conventional strategy selection is the selection of a strategy most often used against the opponent being faced. Therefore, Strategy 2 in Table 8 would be a conventional selection if the opponent belonged to the Zerg faction, and unconventional for any other opponent. We disregard same-faction match-ups (i.e., Terran vs. Terran, Zerg vs. Zerg, and Protoss vs. Protoss – in Table 6) because none of the factions has a strategy that has been used most often against an opponent of the same faction. The win percentages based on match-up and strategy selection are shown in Table 6. In 580 games, where both Terran (T) and Zerg (Z) players chose conventional strategies, the Terran faction won 53% of the games.

However, if the Terran faction selects an unconventional strategy (i.e., strategies 4 or 5) against a conventional Zerg opponent, then the Terran player increases their likelihood of winning to 60%.

Table 10 Win Percentage by Match-Up and Strategy Selection

Match-Up	Only Conventional	Only Player 1 Conventional	Only Player 2 Conventional	Only Non-Conventional
TvZ	53% (580)	52% (811)	60% (78)	57% (102)
TvP	45% (1,749)	42% (52)	51% (86)	- (4)
ZvP	51% (1,271)	37% (130)	53% (438)	39% (69)

Table 9 is slightly misleading because each strategy selection has a different number of occurrences. Most players use conventional strategies, resulting in only a few matches with one or both players implementing a non-conventional strategy. Therefore, any comparison must consider the differences in the number of matches.

We can use the Marascuilo Procedure to compare the win percentage of each strategy selection by match-up. Using a multiple pairwise T-Test would cause alpha inflation [110]. However, the Marascuilo Procedure enables a simultaneous test of the differences between all pairs of proportions when there are several populations under investigation [111]. The test value for the Marascuilo Procedure is calculated using Equation 3, in which the critical value is $\chi^2_{1-\alpha,k-1}$:

$$r_{ij} = \sqrt{\chi^2_{1-\alpha,k-1}}\sqrt{\frac{p_i(1-p_i)}{n_i} + \frac{p_j(1-p_j)}{n_j}} \qquad (3)$$

Table 7 shows the test value of Marascuilo Procedure by match-up and strategy selection. All critical values in table 7 are based on α = 0.05. The results show that player strategy selection does not statistically affect the win percentage. However, the strategy selection in Zerg vs. Protoss (Z vs. P) match-ups show marginal significance, shown by the italicized values and yellow cells in table 10, in terms of affecting the win percentage and pass the Maracuilo test when α = 0.10. The test is limited by the size of the dataset, as most players implement a conventional strategy.

Protoss is slightly more likely to win when they select a non-conventional strategy. By examining the military composition of the conventional and non-conventional strategies, it is possible to interpret the results. Conventional Zerg players quickly expand their base, taking multiple resource-rich areas. In response to this, conventional Protoss players invest in late game technology and expand to a second base. However, this plays to Zerg's strength, providing time to build a large army and attack before Protoss is ready. The non-conventional strategy for Protoss forgoes building the second nexus (base) and involves investing in 4 'Dragoon' units, which are early game ground units. This allows the Protoss player to harass the Zerg player, preventing the latter from growing their base and ultimately slightly increasing the Protoss player's chances of winning.

Table 11 Marascuilo Procedure test value by Match-Up and Strategy Selection

	Only Conventional vs. Only Player 1 Conventional		**Only Conventional vs. Only Player 2 Conventional**		**Only Conventional vs. Only Non-Conventional**		**Only Player 1 Conventional vs. Only Player 2 Conventional**		**Only Player 1 Conventional vs. Only Non-Conventional**		**Only Player 2 Conventional vs. Only Non-Conventional**	
Match-up	**Test Value**	**Critical Value**	**Test Value**	**Critical Value**	**Test Value**	**Critical Value**	**Test Value**	**Critical Value**	**Test Value**	**Critical Value**	**Test Value**	**Critical Value**
TvZ	1.00%	9.03%	7.00%	19.7%	4.00%	17.7 %	8.00%	19.4%	5.00%	14.%	3.00%	24.6%
TvP	3.00%	17.0%	6.00%	13.51%	-	-	9.00%	21.3%	-	-	-	-
ZvP	14.0%	14.8%	2.00%	9.20%	12.0%	20.09%	16.0%	16.2%	2.00%	24.1 %	14.0%	22.1%

Statistical significance was limited by the relatively small number of matches, with players executing a non-conventional strategy. In future work, we aim to utilize a more extensive database of build orders from StarCraft 2 and League of Legends. However, size-first hierarchical clustering can be used to identify a potential strategy for domination. Protoss players that invest in early 'Dragoon' units and harass the Zerg opponent may be more likely to win the game. This methodology can be used within games with build order mechanics to autonomously discover emerging strategies and examine the balance of a game.

Professional players invest time in scouting an opponent base. Their objective is to develop an understanding of their opponents opening strategy, with the goal of countering the discovered strategy. The limited statistical significance found in table 10 indicates that professional players either quickly adapt to an opponent's strategy or mid and end-game late game strategies have a more significant effect on the game's outcome. Little academic research has been done on how strategies evolve over the length of a game and how this evolution affects the game's outcome. Therefore, this remains an open question.

Autonomous Clustering with Outliers – Conclusion

To mitigate research bias, we proposed to cluster behavior models autonomously. However, as we illustrated, real-world datasets of behavior models contain outliers, which can cause traditional clustering methods to group unique strategy clusters.

Current agglomerative hierarchical clustering methods were developed to identify clusters with densely packed nuclei. Additionally, these methods were not demonstrated on integer-based data sets. Therefore, we created a simple 2-dimensional integer dataset and tested two popular clustering algorithms. We found that Almeida's algorithm was unable to correctly cluster the dataset because the overlapping data points resulted in the interquartile range being 0, which meant that all unique data points were classified as their own cluster.

We found that DB-Scan is sensitive to its Eps distance. A change in Eps distance can result in a significant change in the number of clusters reported. Additionally, Ester et al. [86] did not give a method for calculating the correct Eps distance, which could lead a researcher to cluster the dataset multiple times until it returns a set of clusters that match their preconceived notions. While there is no formal definition of alpha inflation (i.e., running more statistical test increases the likelihood of a type 2 error), multiple clustering attempts adds bias to the findings.

We presented Size-First Hierarchical clustering as a way of addressing these limitations. Size-first clustering requires a single input from the researcher, which is the required minimum size of any cluster. We modified the traditional clustering method to group the smallest cluster with its nearest neighbor. Using the Silhouettes criterion, size-first clustering identifies the most natural clustering group that meets the expert defined minimum size. When we compared size-first to the aforementioned algorithms on the 2-dimensional data set, we found that it significantly outperformed Almeida's and was more stable than DB-SCAN.

Using StarCraft as an abstraction of a resource constrained decision problem, we compared algorithms on a real-world behavior model data set. Both DB-SCAN and Almeida struggled to identify clusters because they had loosely packed nuclei and relatively close to one another. The principal component analysis illustrates how close the clusters are to one another. Size-first clustering was able to recover clusters that were more in line with previous research and statements made by professional players.

Since there are no agreed-upon methods for judging the quality of a clustering result, we proposed a paradigm that states clusters must unique (i.e., little overlap) and strategically valuable. We demonstrate that size-first clustering identified unique strategies, which allowed us to predict the faction of the opponent. However, we were unable to show that the strategies we identified changed the outcome of the game. The lack of significance may have come from the limited sample size, which could be fixed by using StarCraft 2 and its substantially more extensive database of replays. Size-first clustering was shown to outperforms two popular clustering algorithms on a behavior model dataset, while requiring significantly less expert input. Agent-based model practitioners can leverage size-first clustering to identify strategy clusters in a less bias manner, resulting in a more objective and better-validated model.

Chapter 4 presents a case-study performed for the Author's Master's Thesis. It is presented to lay the foundation of behavior model recovery, which is built upon in chapter 5. There are times when it is insufficient to model agents as classes, such as an oligopoly. Additionally, industry is resistant to agent-based models even though they reproduce the emergent properties because available models have not sought to explain individual behaviors, such as the strategies employed by technical traders [29], [38], [112]–[114]. The next chapter provieds a case study in recovering strategies in a fiancial market, with the goal of improving the validity of financial agent-based models.

Chapter 4: Case Study in Apprenticeship Learning using the McIntire Hedge Fund Tournament

Financial markets are complex environments comprised of many interrelated factors, including, perhaps most importantly, many different traders. Although these traders may share the goal of increasing capital, they often have—at least in the short term—partially independent objectives. Furthermore, even when traders seek to achieve the same objectives, they often accomplish them in drastically different ways. Nonetheless, these traders are interracting with one another, since the the timing, volume, and price of each individual trade ultimately affect the broader market environment. These interactions of independent traders create the emergent properties known as stylized facts [115]. There are three such emergent properties: (1) no auto correlation of price returns, (2) auto correlation of absolute price change, and (3) aggregate normality of price returns over long time frames [33], [38], [112].

The need is for a viable, comprehensive model that will take into account the interactions of traders by reproducing these stylized facts. Such a model will help academic theorists, industry professionals, and regulators better understand financial markets. A comprehensive model will allow for identification of systemic risk, while also providing a test bed for regulatory reforms [112], [116]. There have been, in fact, many different models that have reproduced these stylized facts; however, these models have been confined to academic applications because industry and government agencies have not found them reliable. This resistance has largely been due to the fact that available models have not sought to explain individual behaviors, such as the strategies employed by technical traders [29], [38], [112]–[114]. Instead, finance models have focused on mimicking actions by fitting a probability distribution. Trading strategies are not captured through this method—which means, for example, that regulators cannot determine how a new regulation will affect an individual trader.

There are several reasons that financial models have been designed to follow probability distributions. First, in the past academics did not have access to traders' identities; all orders and trades were anonymous. This policy was in place as a means of safeguarding privacy; it also prevented other traders (as well as academic theorists) from reverse engineering any individual strategies. Moreover, up until recently orders have traditionally been placed by people. People are not perfectly rational [114], [117], [118]. Therefore, it is hard to recover an individual trader's strategy with any degree of accuracy, because any such strategy will necessarily reflect human idiosyncracy and, in some instances, error. Additionally, there is no guarantee that a person will use the same strategy in all market conditions [119].

This chapter examines the question of whether it is possible to determine a decision space relating to an individual trading strategy, such that there is a one-to-one relationship between a state and an action. Two key new factors have made determining a decision space plausible. First, the Commodity and Futures Trading Commission has changed its regulations so that identifying information regarding orders and trades is now accessible to academic researchers. Additionally, algorithmic traders now account for 70% of the trades in equity markets and 35% in commodity markets. Algorithmic traders are computers that trade autonomously [27], [28]. For the purposes of this study, it is assumed that autonomous algorithms follow deterministic decision rules. In other words, the algorithm will perform the same action given that the algorithm finds itself in the same state. These two new factors increase the possibility of recovering strategies used by a majority of market participants. To date, however, very little research has been done in this area: this study aims to address that need by exploring methods for reverse engineering trading strategies.

As a case study, this chapter examines algorithmic trading data from the McIntire School of Business Hedge Fund Tournament. This chaoter will provide a proof of concept by determining the decision space of the winning team and reproducing it in an out-of-sample test. The McIntire Hedge Fund Tournament is designed to teach students about electronic trading platforms and portfolio management.

Each team is given a basket of stocks and options, which must be held for the full duration of the tournament. Teams are also provided cash, which they use to hedge their positions. The team that most closely follows a 1% annualized return over the entire tournament wins.

The McIntire School trading tournament is an ideal test case because all algorithms entered are deterministic in nature. Additionally, teams are limited to trading a relatively low number of securities, the objectives of the teams are clear, and all trades are captured with team identifiers. The challenge, however, is that teams may use a variety of approaches to reach their objectives, complicating the task of recovering individual team strategies. This chapter, then, is a feasibility study on whether it is possible to identify a single trading strategy in an environment where there are a plethora of ways to reach the same objective.

The following two sections will provide a literature review of static and dynamic trading and return replication techniques, respectively. The paper will then review the machine-learning technique recursive partitioning, also known as classifiication and regression tree. An empirical test and results section follows. In the results section, both the testing procedure and results are presented. A more detailed explanation of the hedge tournament can also be found in this section. The paper then concludes with final thoughts.

Trading and Return Replication: Static Method

Hedge fund cloning is a research topic that focuses on mimicking returns seen by hedge funds. The seminal paper on this subject was written by William Sharpe in 1992. Sharpe examined open-end mutual funds offered by Vanguard between 1985 and 1989. An open-end mutual fund is comprised of securities purchased with money contributed by many investors. Mutual funds have a manager or managers who buy and sell securities for the entire fund. These managers are mandated to meet or exceed returns of asset classes [120]. An asset class is a grouping of the same type of securities, such as corporate bonds. Below are the 12 asset classes Sharpe used in his study:

1. Treasury bills—Salomon Brothers' 90-day Treasury Bill Index

2. Intermediate-term government bonds—Lehman Brothers' Intermediate-Term Government Bond Index

3. Long-term government bonds—Lehman Brothers' Long-Term Government Bond Index

4. Corporate bonds—Lehman Brothers' Corporate Bond Index

5. Mortgage-related securities—Lehman Brothers' Mortgage-Backed Securities Index

6. Large-capitalization value stocks—Sharpe/BARRA Value Stock Index

7. Large-capitalization growth stocks—Sharpe/BARRA Growth Stock Index

8. Medium-capitalization stocks—Sharpe/BARRA Medium Capitalization Stock Index

9. Small-capitalization stocks—Sharpe/BARRA Small Capitalization Stock Index

10. Non-U.S. bonds—Salomon Brothers' Non-U.S. Government Bond Index

11. European stocks—FTA Euro-Pacific Ex Japan Index

12. Japanese stocks—DTA Japan Index

Sharpe's hypothesis was that a mutual fund's return can largely be explained by the average return of the asset classes to which the fund is exposed, and that therefore the returns of mutual funds should depend more on their relative exposure to a particular asset class and less on the specific securities they hold. To test this hypothesis, Sharpe examined monthly returns of mutual funds and attempted to find linear combinations of asset classes that best explained these returns. Below is the linear regression equation Sharpe used:

$$R_i = [b_1 F_{i1} + b_3 F_{i3} + \cdots + b_n F_{in}] + \epsilon_i$$

where R_i represents the return estimated by the model for a specified month, F represents the return provided by each asset class during specified month, and b represents the relative weight placed on each asset class. In other words, $F_{1,1}$ represents the return on investment seen by treasury bills in the first month of the data set, and b_1 is the relative exposure a particular mutual fund has to treasury bills. Using the aforementioned equation, we can generate an estimate of the mutual fund's return for the first month in the data set (R_1). Note that $\sum_{j=1}^{n} b_{ij} = 1$, for all i [120]. The term ϵ_i represents the amount of return that is unexplained by this model. To determine how effective the model is at explaining the behavior of a particular mutual fund over several months, a coefficient of determination is calculated as

$$R^2 = 1 - \frac{Var(\epsilon_i)}{Var(R_i)}.$$

The closer to 1 the R^2 is, the better the model represents the data set. Sharpe's article shows results ranging from 92% to 97%. In other words, Sharpe's model was able to explain the returns of individual mutual funds by determining the relative weight of each asset class within each fund—that is, how much capital the fund invested in each asset class. The mutual funds Sharpe examined implemented a buy-and-hold strategy and were limited to little or no leverage. Given these parameters, Sharpe's study clearly illustrates that the differences in the returns of mutual funds can be explained by the different weights each fund places on individual asset classes. This is easily demonstrated in Sharpe's regression analysis [120]. However, other investment vehicles such as hedge funds are not bound by these restrictions [121]: hedge funds are able to carry leverage and can use a more dynamic trading strategy. As a result, Sharpe's regression analysis does not work for these institutions [120].

In 1997, William Fung and David Hsieh revised Sharpe's regression technique so that it was better suited for more dynamic trading strategies [121]. Fung and Hsieh took into account three critical additional factors: asset class (F), trading strategy, and leverage (w). Asset class, as in Sharpe's paper, was defined as a grouping of the same type of securities. Trading strategy factored in whether the manager

was long or short on the asset. Leverage refered to the quantity of the asset held. Leverage can be thought of as the weight placed on an asset, as in Sharpe's paper. The one major difference, however, is that in Fung and Hsieh's model leverage was not constrained to sum to one but could range from − infinity to + infinity. Fung and Hsieh's model is as follows [121]:

$$R_t = \sum_{k=1}^{N} W_{kt} F_{kt} + \epsilon_t$$

The advantage of this model is that it is more sensitive to trading techniques. It allows for $w = -1$, in the event a hedge fund is shorting one contract, or $w = 2$, in the event that the hedge fund is long on two contracts. As in Sharpe's study, this model was run over several months and a regression analysis was applied, resulting in a median coefficient of a determinant of 25%. The results are significantly lower than in Sharpe's mutual fund analysis. Fung and Hsieh concluded that the dynamic strategies employed by hedge funds lower the ability of linear models to explain monthly returns [121]. Hedge funds change their portfolio allocation several times in a month, which is obscured in monthly returns.

Through the use of principal components, Fung and Hsieh discovered five general classes of hedge fund trading styles:

1. Systems/Opportunistic—technically driven traders, who also make bets on market events

2. Global/Macro—traders who only participate in the most liquid markets in the world (i.e., currencies and government bonds)

3. Value—traders who buy securities that are perceived to be undervalued

4. Systems/Trend Following—technically driven traders, who tend to follow the overall trend of the market

5. Distressed—traders who invest in companies near or recently emerging from bankruptcy

Fung and Hsieh attributed only 43% of the monthly return variances to these five trading styles [121]. However, even though these trading styles explain only a segment of the monthly return variances, they do suggest several key conclusions. For example, hedge funds that specialize in buying securities of companies that are undervalued are very sensitive to changes in the U.S. equity market.

In 1997, Brown and Goetzmann [122] classified Morningstar mutual funds into trading styles based on monthly returns. Unlike Fung and Hsieh's classifications, however, Brown and Goetzmann's study did not assume a linear model. As Fung and Hsieh illustrated, a linear model's predictive power degrades if the mutual fund has a dynamic strategy. To better model dynamic strategies, Brown and Goetzmann allowed the weights on asset classes to vary through time. Their model is illustrated below:

$$R_i = [b_{t1}F_{t1} + b_{t1}F_{t1} + \cdots + b_{Tn}F_{in}] + b_0 + \epsilon_i$$

Brown and Goetzmann used a regression model identical to Sharpe's model except b_{i1} is allowed to change through time. If we assume the asset classes are the same as the Sharpe model, b_{i1} represents the fund's exposure to treasury bills during the first month in the data set. Additionally, b_0 is a constant that is specific to each strategy. To estimate b_{in} and to classify the mutual funds into groups, the following procedure was run:

1. Specify the number of trading style classes (K)
2. Specify the time period of classification (T)
3. Specify the number of trading funds (N)
4. For all combinations of N funds divided into K classes, complete the followng calculation:

$$\bar{R}_t' = \sum_{i=1}^{N} \frac{R_{it}}{var(R_{it} - \bar{R}_t)}$$

5. Select the combination that provides:

$$min \sum_{j=1}^{K} \sum_{t=1}^{T} \sum_{i=1}^{N} (R_{ti} - \bar{R}'_{tj})^2$$

for all funds (i) in class (j)

6. Run a regression on each class, treating individual months as independent data sets, and estimate b_{Tn} for each

Brown and Goetzmann were able to derive trading styles and state meaningful interpretations of the results. For example, a trading classed named "Global Timing" dynamically increases and decreases its exposure to the U.S. market [122]. However, Brown and Goetzmann's method did not improve upon Fung and Hsieh's principal component method, explaining only 37% of the monthly return variation.

Using different hedge fund data, Brian Liang performed the same regression model that Fung and Hsieh developed. Lang's findings were consistent with Fung and Hsieh's results. However, he interpreted B_0, also known as the intercept term, as the manager's contribution to the returns [123]. Liang found that the intercept term contributes between −5% and 1% for each asset group. In other words, hedge fund managers are likely to negatively affect returns.

In 2007, Hasanhodzic and Lo developed a regression model similar to Fung and Hsieh's model. The only significant difference is that Hasanhodzic and Lo limited their regression model to five asset classes, each of which had exchange traded fund (ETF) counterparts. The goal of this analysis was to provide linear combinations of ETFs that the average investor could use to gain returns similar to those of hedge funds. Additionally, Hasanhodzic and Lo claimed this approach would prevent the overfitting seen in Fung and Hsieh's principal component/regression approach. Overfitting occurs when a model excels in a training phase but performs poorly in out of sample test. The five ETF classes used in this regression model are as follows [124]:

1. US Dollar Index

2. Bond: Lehman Corporate AA Intermediate Bond Index

3. Credit: Lehman BAA Corporate Bond Index

4. S&P 500 Index

5. Goldman Sachs Commodity Index

Using a 24-month rolling window to avoid a look-ahead bias, Hasanhodzic and Lo illustrated that they were able to develop a linear combination ETF strategy that had higher returns than the market. However, the linear clones did not perform as well as the hedge fund counterparts [124]. This conclusion again reinforces Fung and Hsieh's finding that the dynamics of hedge fund strategies are obscured by the aggregation of data into monthly returns.

Merrill Lynch (Merrill Lynch Factor Index) and Goldman Sachs (Absolute Return Tracker Index) have both created indices that are supposed to mimic hedge fund returns. In 2010, Meyfredi et al., spurred on by these hedge fund replication indices, applied Kalman filtering to their own hedge fund replication. Kalman filtering uses a regression equation similar to Brown and Goetzmann's. The significant difference is that Kalman filtering requires at each time step, B_t (weight placed on the asset class) is estimated from B_{t-1} using a transition matrix. Therefore, not only does B have to be estimated, but the transition matrix A also has to be estimated. In the equation developed by Meyfredi et al., η and ϵ are normally distributed error terms with a mean equal to 0 [125]:

$$B_t = AB_{t-1} + n_t,$$
$$R_t = B_t'F + \epsilon_t.$$

The Kalman filter model gives the best R^2 value, ranging from 62% to 80%, when the nonlinear models are applied to the same 24-month rolling window employed by Hasanhodzic and Lo. This indicates that the Kalman filter is capable of recovering a larger amount of the dynamic hedge fund trading strategy.

However, when applied to out-of-sample data, the Kalman filter provided an average annualized excess return of −0.04, which is significantly lower than the returns of the original hedge funds [125]. Excess returns are returns that exceed a market benchmark, such as the S&P 500. This indicates either that the Kalman filter overfit the data or that hedge funds do not carry out the same strategy from month to month.

As the preceding review suggests, for the past two decades, academics, individuals, and institutions have tried to reverse engineer successful strategies. By mimicking these strategies, investors can achieve similar returns without incurring the upfront cost of research. Recently, subscription services have begun to offer investors the opportunity to pay for the privilege of automatically having their accounts mimic successful traders. This has allowed subscribers to see all of the trades associated with a profitable trading strategy. The proposed methodology in this chapter can be applied to the subscription service data with the hopes of better understanding real-world trading strategies and applying these trading strategies to finacial agent-based models. The next section will discuss some of these techniques in greater detail.

Trading and Return Replication: Dynamic Methods

Long-term securities, such as stocks, can be used to replicate complex instruments. The original example is when the purchase of a stock through riskless borrowing is used to replicate the payoff curve of an option on said stock. If volatility is known, the Black-Scholes formula can be used to exactly replicate an option.

An option gives the holder the right but not the obligation to buy (call option) or sell (put option) a stock at a specified price (K). The duplication of an option price is a simple form of trading strategy replication. To replicate the strategy, the Black-Scholes formula is used to calculate Delta (Δ). Delta measures the option price change when the underlying stock increases by one dollar. The formula for delta is given below:

$$d_1 = \frac{ln(\frac{S}{K}) + (r - \delta + .5\sigma^2)t}{\sigma\sqrt{t}}$$

$$\Delta_{Call} = e^{-\delta T} N(d_1)$$

$$\Delta_{Put} = e^{-\delta T} N(d_1) - 1$$

Using the current price of the stock (S), volatility of the stock (σ), is the risk free intrest rate (r), the dividends the stock pays (δ), the strike price of the option (K), and the time until the option expires (t), a relationship between the option price movement and the stock price movement can be established. The function N(x) is the cumulative standard normal comulative distribution function; it gives the probability that a number randomly drawn from a standard normal distribution falls below x [126]. If Δ and σ are assumed to be constant, it becomes trivial to calculate the payoff curve of an option:

Buy Δ amount of stocks

Borrow $Ke^{-rt} N(d_1 - \sigma\sqrt{t})$ amount of cash

The problem with the Black-Scholes approach is that Δ changes as the stock price changes. Therefore, delta must be recalculated at every time step, and a dynamic strategy of buying and selling the stock must be used to replicate the option. Dynamic replication of options is a branch of research that seeks to determine the optimal way of replicating an option payoff curve. When simplifying assumptions—such as σ is constant and the market is always open—are removed, the problem becomes nontrivial. In fact, perfect replications are impossible in real-world markets [127].

Assume you are given an option with a payoff curve of F(P_T , Z_T), where P_T is the price of the underlying asset at time T and Z_T is a vector of a state variable that dictates the value of the option at

time T. The value of the option does not correlate perfectly to P_T. For specific examples where the aforementioned correlation is not exact, see Bertsimas and Kogan [127]. The objective of dynamic replication is to find a combination of stock shares and borrowed cash to $minE[V_T - F(P_T, Z_T)]^2$. In other words, the objective is to minimize the square difference between the portfolio value (V_T) and the option value at expiration. Therefore, a measure of how close the option's payoff curve is replicated is the squared error, as shown in the following equation:

$$\epsilon = \sqrt{Min_{\theta_T} E[V_T - F(P_T, Z_T)]^2}$$

Bertsimas and Kogan. impose the constraint that the portfolio be self-financed. Aside from the initial capital necessary to establish the first position, all changes in cash (B) being borrowed and stocks (θ) being held must be financed by the buying and selling of stocks. The equation below illustrates the constraint:

$$P_{t_{i+1}}(\theta_{t_{i_1}} - \theta_{t_i}) + B_{t_{i+1}} - B_{t_i} = 0\,,$$

which implies:

$$V_{t_{i+1}} - V_{t_i} = \theta_{t_{i_1}}(P_{t_{i+1}} - P_{t_i})$$

Assuming that the expected return divided by variance for any period is bounded, there exists an optimal replication strategy that gives the lowest possible $\mathcal{E}$ [127]. Dynamic programing is used to work backward from the terminal payoff states, thereby identifying the optimal strategy. Kat and Palaro extend this process to replicating a fund of funds [128]. A fund of funds is a hedge fund that is made up of smaller hedge funds. Kat and Palaro attempt to recover the risk profile of observed hedge funds (i.e., the Sharpe ratio profile). Kat and Palaro use a single asset that can be traded to replicate hedge fund returns. Using their replication strategy, Kat and Palaro were able to get within 10% of the observed hedge fund returns [128].

The problem with Kat and Palaro's methodology is that it assumes that a single asset can be used to replicate a hedge fund's risk profile. This single asset, however, may not be identifiable; even if it is, it may not be adequate to mimic the hedge fund as a whole. In many cases, individuals and organizations employing Kat and Palaro's methodology end up basing their results on a series of actions, such as multiple trades, that taken together represent a strategy—rather than on the movement of a single asset.

Expanding on dynamic programming a company named Adaptrade Software claims to able to replicate strategies by using past trades. This software examines past trades and their outcomes, such as profits/losses occurring from a trade. Using a genetic algorithm, the software combines technical trading rules (e.g., moving average, relative strength index, etc.) to generate a strategy that mimics the observed trades and outcomes. In its brochure the company claims that its clone strategy generated profits that were 3% lower than the observed strategy. It is important to note that Adaptrade does not claim the observed strategy follows the same rules as the clone strategy. In other words, Adaptrade has not claimed to have found a trading strategy decision space. Additionally, it has not published any out-of-sample testing. However, the fact that a commercial company has created reverse-engineered trading software suggests that there is a desire for a methodology to reverse engineer trading decisions. The following section will examine apprenticeship learning problem applied to trading strategy recovery.

Apprenticeship Learning

Recovering a trading strategy has many elements of a classical apprenticeship learning problem. A trading strategy is a set of decision rules that lead to the selection of a security, as well as, a decision to buy or sell the security and the amount of shares that will be traded. The apprentice is given a set of discrete time events to learn from. All security prices and market history are available to the observer. Additionally, the financial position of the trader is known (i.e., inventory, cash, profit, losses, etc.). The apprenticeship problem is to predict trader actions in future time periods.

To examine this problem, I used data from the McIntire Hedge Fund Tournament. The goal of the tournament is for particpants to hedge a portfolio of illiquid assets by buying an offsetting combination of stocks and options. The winner of the tournament is the team that comes the closest to making a 1% annualized return with their combined illiquid and liquid portfolio. In essence, the objective of the tournament is for teams to limit the variance of their portfolios. I chose to recover the winning team's trading strategy because they employed an algorithm that did not make any erroneous trades, such as selling a stock when none existed in inventory.

The winning team used a delta hedging strategy. This trading strategy involved making two decisions when hedging the portfolio. The algorithm first needed to choose which security to trade, and then had to determine the desired end-of-day delta exposure. The desired delta exposure determines the direction (i.e., taking a long versus a short position) as well as the volume (i.e., how many shares) are traded. Each decision is treated as an independent problem. The algorithm decisions are treated as two independent problems because this approach lowers the complexity of recovering the decision space. Instead of simultaneously solving for two independent variables this methodology allows for solving for one variable at a time. The following section will explain the methods used to determine which security the algorithm would choose to trade.

Determining Which Security to Trade

The trading algorithm has a set of security options that are available to trade. The securty options have different expiration prices and therefore different monetary valuations. For this section we make the simplifying assumption that at the end of day the portfolio has a delta exposure of 0. In other words, every trade that is taken leads to a delta exposure of 0. We hypothesize that the observed algorithm is attempting to complete this task. Based on this hypothesis, we assume that the recovered algorithm will accomplish the goal of maintaining a delta neutral portfolio.

The Need for Action Features

The trading algorithm selects an option to trade, but all actions (trades), under the aforementioned assumptions, lead to a delta exposure of 0. It is our hypothesis that the trading algorithm has larger-level objectives that it is trying to meet, such as maximizing delta change per dollar. Therefore, to accurately reverse engineer trading strategies it becomes necessary to discover action features that facilitate these higher-level objectives. For example, option A may be worth $1.00, while option B may be worth $0.50. Although both options can be used to delta hedge a portfolio, an algorithm that attempts to maximize delta change per dollar may select option B over option A because option B has a higher delta-to-dollar ratio. This difference could lead the algorithm to use option B when hedging the portfolio.

Supervised learning traditionally assumes that actions are chosen based on their probability of transitioning to a desired state. However, the relatively small number of observations in the training set (approximately 45 trading days) led us to use a state space representation employing action features. Therefore, we assumed that the actions were chosen based on a comparison to other actions. Features are given to each potential trade. Action features are indicator variables (1, 0) that describe potential actions. If a subset of indicator variables are found to be always on or off in actions that are chosen, and the reverse is the case for actions that are not chosen, then we conclude that these features are used to select which security to trade. The following features were applied to the tournament data:

1. Security is an option – action will trade an option.
2. Security expires in May – action will trade an option that expires in May.
3. Security is already owned – action will trade a security that has an open position.
4. Lowest price option – action will trade the lowest price (cheapest) option.
5. Lowest absolute value trade – action will result in the lowest absolute value for a trade.

Assuming that all trades reduce delta exposure to zero and transaction cost is taken into account

6. Largest delta/price ratio – action will trade option with the highest delta/price ratio.

7. Largest short – generates most cash from shorting.

8. Sign switch – action results in a switch from a long to short position or vice versa.

The results of this method will be presented and explained in the empirical results section. However, I will note that certain features, such as (1) “Security is an option" and (2) “Security expires in May," were found in all selected actions. In other words, the team’s strategy was to hedge using only options that expired in May.

Supervised Learning Techniques: Recursive Partitioning

An open-source version of classification and regression trees, known as recursive partitioning, was used to determine the trading algorithm [129]. A superset of all possible actions (trades) A was made for each trading day. A trade was marked positive if the algorithm implemented the trade on that day; otherwise it was marked as negative. Additionally, each action had a set of features (F_{ai}) that described the action, and each trading day had a set of features (F_{sj}) that described the current market conditions. Our objective in employing recursive partitioning was to use the set of features to predict which trades would be marked as positive and which would be marked as negative.

Recursive partitioning results in a binary tree. A binary tree consists of one root (origin) node that splits the data into two groups based on a specified condition. The method for building binary trees begins with selecting a single feature that best splits the data into two groups. (In this context, the term *best* refers to the split that maximizes the separation between classes.) The subgroups are then split into smaller groups, and they in turn are again split into yet smaller groups, in a recursive process. This continues until each class is completely classified or until a minimum observation threshold is reached in each node [26]. Below is the equation used to determine which feature to use as the partition rule:

$$\max[P(L)P(C_1) + P(R)P(C_2)].$$

The proportion of observations that is classified to the left branch of the tree is represented by $P(L)$ and the proportion to the right is represented by $P(R)$. The proportion of observations on the left branch classified as category 1 is denoted as $P(C_1)$, while the proportion of observations on the right branch classified as category 2 [129] is denoted as $P(C_2)$. The results section will illustrate that it is necessary for actions to contain features.

Determining End-of-Day Delta Exposure

When recovering a trading strategy, there are a few basic functions that must be determined. First, it is essential to identify what will trigger the strategy—that is, what enables the strategy to become active; then, once the strategy is active, it is necessary to ascertain which security will be traded. (The preceding sections described techniques for accomplishing this task.)

The final part of the strategy involves determining whether to buy or sell shares and how many shares to trade. In this section, the previous assumption that all trades lead to a delta exposure of 0 is dropped. Instead, the machine-learning technique known as regression trees is used to predict the delta exposure at the end of the day. Using the Black-Scholes formula, the desired end-of-day delta exposure, and the asset being traded, it becomes trivial to determine not only the trade direction (i.e., buy or sell) but also the desired quantity to be traded. Therefore, in this section we will predict desired end-of-day delta exposure as a means of determining both the direction of a trade and the quantity being traded.

The apprenticeship learning algorithm is presented as a set of state features (F_{sj}). For the purposes of this study, we assume that the option being traded is known, having been arrived at using the methodology discussed in the previous section. Therefore, intrinsic action values such as price of a trade are treated as state features in the method we examine here. These features represent the current state at the beginning of the day. Additionally, the learning algorithm is given the dependent variable delta

exposure (D_j) at the end of the day. Using the state features listed below, the learning algorithm is trained to predict the end-of-day delta exposure.

1. Security Family Delta Exposure: How much the value of securities currently held in inventory changes if the underlying stock value increases by $1.
2. Security Family Gamma Exposure: How much the delta of securities currently held in inventory changes if the underlying stock value increases by $1.
3. Delta of the Option being Traded.
4. Gamma of the Option being Traded.
5. Price of the Option being Traded.
6. Current Cash Account: the current cash in the account.
7. Current Margin Value: the amount of Margin open.
8. Portfolio Value - Target Value : difference between the current portfolio value and the target value.

The mean value of the dependent variables (delta exposure of trades) that are partitioned into the same class is used as the prediction for that class. The splitting criteria seek to maximize the between-groups sum of squares [129], as follows:

$$\max SST - (SSL + SSR).$$

where

$$SST = \sum (y_i - \bar{y})$$

In the above equation, SSL is the sum of squares for the left node and SSR is the sum of squares for the right node. The algorithm terminates if it cannot generate a minimum distance between the sum-of-square error of the children versus the parent node.

Using the mean value of the dependent variable gives one possible prediction for a potentially large number of observations. Although this makes the regression trees tractable, it limits their predictive power. Therefore, in this chapter the method of regression trees was augmented to use stepwise linear regression instead of the mean value as the final node's prediction. Stepwise linear regression adds and removes variables in a standard linear regression model. The goal is to provide a linear model that has the highest adjusted R^2 value. For all potential variables a simple linear regression model is fitted. The t* statistic is taken for each model to test whether or not the slope is 0. Below is the equation for the t* statistic:

$$t^* = \frac{b_1}{MSE/\sum(X_i - \overline{X})^2}.$$

The X variable with the largest t* value that also has a p-value less than 0.15 is added into the model [31]. Assuming that the "delta of the option" had the largest t* value and had a p-value less than 0.15, then it would be added into the model. In the next iteration, all models will contain "delta of the option," and all other possible variables will be added to the model one at a time. A t* statistic is taken of each one, and the process repeats. If there are two or more variables in the model, the procedure changes: all variables in the model have their t* statistic taken and p-values calculated. If a variable has a p-value greater than 0.15, then it is removed from the model, after which the aforementioned process of adding variables is applied again [130]. This process continues until no more variables can enter or leave the model. Stepwise regression allows for a closer prediction of the response variables. However, it also creates the possibility of overfitting to the data. Overfitting can lead to erratic behavior of the model during out-of-sample testing. In the next section, both versions of regression trees will be used and their performance compared to one another.

Empirical Tests and Results

McIntire Hedge Fund Tournament

The McIntire Hedge Fund Tournament is held semiannually at the University of Virginia. The goal of the hedge fund tournament is to hedge a portfolio of illiquid assets using stocks and options, in such a manner that the portfolio provides a 1% annualized return throughout the tournament. All trading must be done electronically, which allows for the capture of every trade in the tournament. It is important to note that teams are assumed to be too small to change the price of the asset they are trading.

The tournament is held over approximately three hours continuous time period. Each minute is equivalent to a simulated day, with a bid and ask price being presented for each asset. Teams possess a basket of stocks and options that they are not allowed to trade during the tournament. With $16 million in cash, teams must take offsetting positions to limit the volatility of their overall portfolio. Additionally, teams must try to consistently deliver a 1% annualized return. A measurement known as the tracking error (TE) is taken every Sunday. The team with the lowest cumulative tracking error is crowned the winner. Below is the equation for the tracking error:

If Portfolio Value $>$ = Target:

$$TE = (PV - T)/2.$$

Otherwise,

$$TE = (T - PV).$$

The target (T) in the equation above is the value of the inital portfolio plus a 1% annualized return to date. To minimize tracking errors, teams enter long or short positions in stocks or options to offset the illiquid position. Five rules limit the ability of teams to enter positions:

1. Margin account value must not exceed $22 million
2. Teams must have at least 30% of the margin account value in cash at all times
3. Teams cannot trade a specific asset more than once a day

4. For a trade to be initiated, the required amount of cash must be present in the portfolio
5. A team cannot trade any asset that has a value of 0 dollars

Assets are broken up into families. Twelve asset families make up the tradeable world for the tournament. These asset families are derived from the following twelve stocks:

1. Apple
2. AIG
3. Citigroup
4. Dell
5. Walt Disney
6. General Electric
7. Gold
8. Google
9. Coca-Cola
10. Microsoft
11. Transocean
12. UBS

A stock's value throughout the tournament is tuned to the historical volatility and trend of the stock from the previous six months. Through the use of Brownian motion, a price path is produced that is similar to historical data but that does not exactly replicate it. Using the Black-Scholes equation, the price for each option is calculated. There are two sets of options: the first expires in March, and the second expires in May. Additionally, there are five puts and calls for each security at each expiration month; the options have varying strike prices. This results in 252 securities that can be traded during the tournament. However, some securities are placed into the illiquid portfolio, lowering the number of tradeable securities to 221.

Twelve teams competed in the tournament. However, only three teams were eligible to have their strategies recovered. The majority of the teams made a large number of errors when trading, resulting in

large tracking errors, after which these teams stopped trading. As a result, there was not enough information to recover the trading strategies usedy by these teams. Additionally, some teams traded manually. Manual trading adds another layer of uncertainty. The assumption in recovering algorithmic trading strategies is that algorithms presented with the same state information will make the same decision every time. Since this study involved a proof of concept, one team in particular, the winning team, was used to illustrate the results.

Results Determining Which Stock or Option Will Be Traded

The majority of the teams used delta hedging to accomplish the goal of variance reduction. This can be seen by comparing the delta exposure of each team's strategy to that of a non-hedged portfolio. To determine which asset would be chosen to trade, we assumed, for simplicity, that all trades resulted in a delta exposure of 0.

A variation of recursive partitioning is used on all subsequent results. The major contribution of this research is that it demonstrates that action-based features improves trading strategy recovery. To obtain the following results. Recursive partitioning was run on the first two months of trading data. Using the resulting partitions, the following two months of trading data were used as an out-of-sample test. The accuracy measures below indicate the percentage of possible actions correctly labeled in out-of-sample testing.

Table 12 Security Selection Accuracy

Asset Symbol	State Features	Action & State Features
AAPL	100%	99%
AIG	100%	100%
C	100%	84%
DELL	98%	99.5%
DIS	57%	91%
GE	96.9%	96.9%
GOLD	58%	100%
GOOG	57%	100%
KO	100%	100%
MSFT	100%	82%
RIG	94%	81%
UBS	57%	89%
Average	84.8%	93.5%

As can be seen in the above table, using only state space features results in lower accuracy rate. Although the state space plays a role in the algorithm's decision making, it is impossible to generate an accurate replica using the state space alone. This suggests that the algorithm is comparing possible actions. In other words, two actions lead to similar states, so the algorithm chooses the cheaper of the two options. Intuitively, however, we might expect the algorithm to make such a choice, since machine learning does take into account an action feature space.

Results Determining Delta Exposure

It is important to recall that by determining the desired end-of-day delta exposure both the direction of a trade and the number of shares traded can be calculated. The same methodology reviewed in the previous section was used to examine the ability of regression trees to predict delta exposure. Each security family was treated as its own independent machine-learning problem. Regression trees were trained on the first two months of trade data for each security family, and the following two months were used as out-of-sample validation results. Both the mean value prediction and the stepwise linear regression prediction were calculated for each day. The graph below shows that using stepwise linear

regression can provide a better fit to the training data. Both variations of the regression tree are presented with the entire two months of observed data, which are used to create the classification partitions and generate predictions.

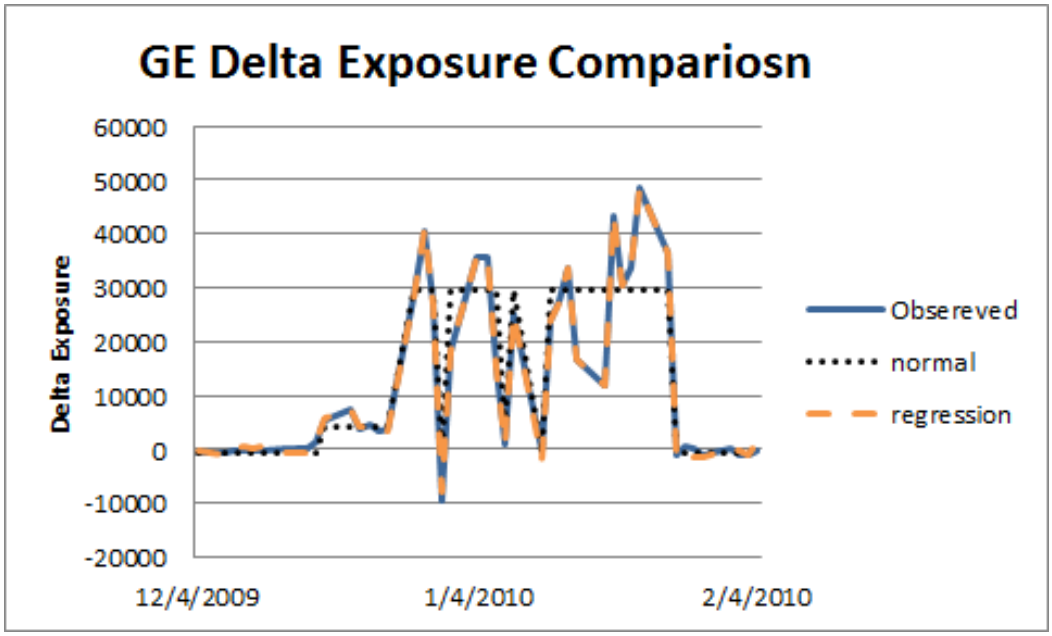


Figure 47 GE Delta Exposure

The regression tree for GE is illustrated below. The first delta exposures given in the final nodes are the predictions using the mean of the dependent variables. The second delta exposures given are the predictions using stepwise linear regression.

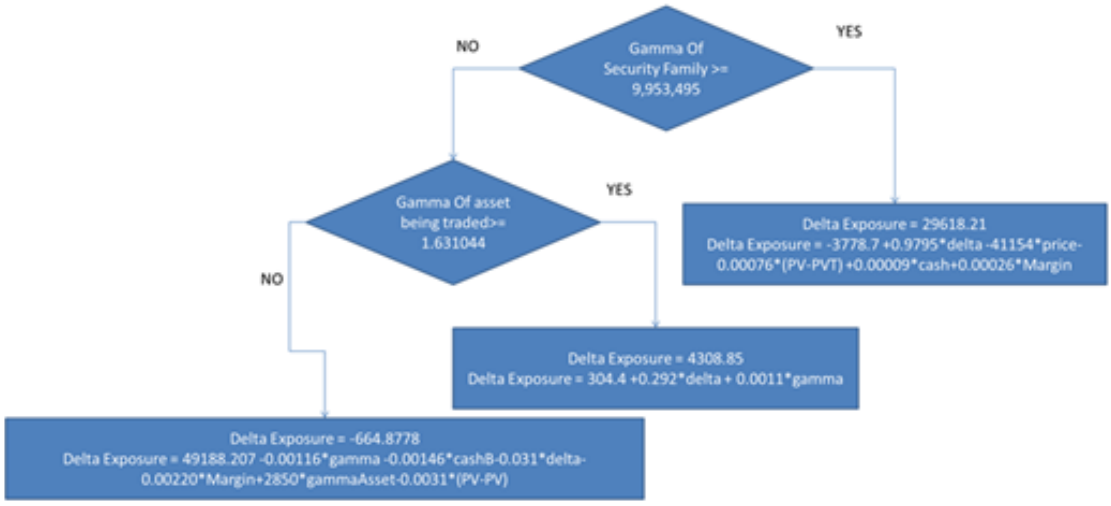


Figure 48 Step-wise Linear Regression Tree

Regression trees do not examine how an error in a prediction will affect subsequent decisions. Therefore, we implemented both regression tree outcomes and ran them over the entire four months. The results from the security family DELL best illustrate the overall finding.

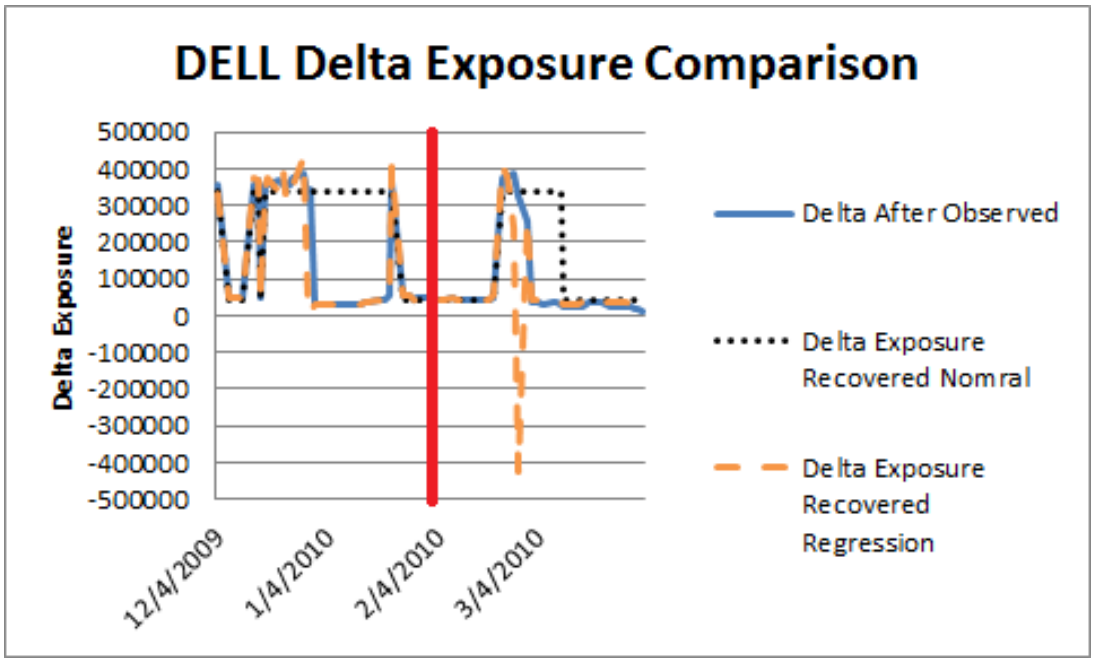


Figure 49 Dell Delta Exposure

The above graph illustrates the observed delta exposure, the delta exposure recovered using the normal regression tree method, and the delta exposure recovered using the regression tree method augmented with stepwise linear regression. The red line indicates the transition from the training set to the validation set. The stepwise technique does a better job in matching the observed data in the training set. This trend continues into the validation set, until the regression technique makes a rapid and sudden divergence from the observed delta exposure. This occurs because the values presented to the linear equation are outside the training set values. Thus, the regression model generates an unexpected result. This is quickly corrected, but it illustrates an inherent instability in this type of methodology.

The regression tree method that uses the mean of the dependent variables as its prediction and cannot detec the nuisances in delta exposure. The regression tree aglorithim will classifiy states with different values for their factors, such as the amount of avialable cash, into the same terminal node. Therefore, it takes longer for the mean regression tree to recognize the change in delta exposure. However, it does not have the sudden and rapid departure from the observed delta exposure that was seen in the regression model. This means the model is more stable than the stepwise regression model but not as responsive.

As stated above, the desired delta exposure is estimated so the direction and quantity of the trade can be predicted. Having this information allows us to examine how closely we match the observed algorithm. A cumulative difference between the observed algorithm tracking error and both variations of the regression trees' tracking error is calculated and presented in the graph below. The ideal recovered algorithm would have a horizontal line at 0, meaning it exactly matched the tracking error of the observed algorithm. Figure 50 illustrates that the mean prediction regression tree performs better mimics the stepwise variation. This is because the mean prediction is stable and does not suffer as badly from values in data that have not been seen before.

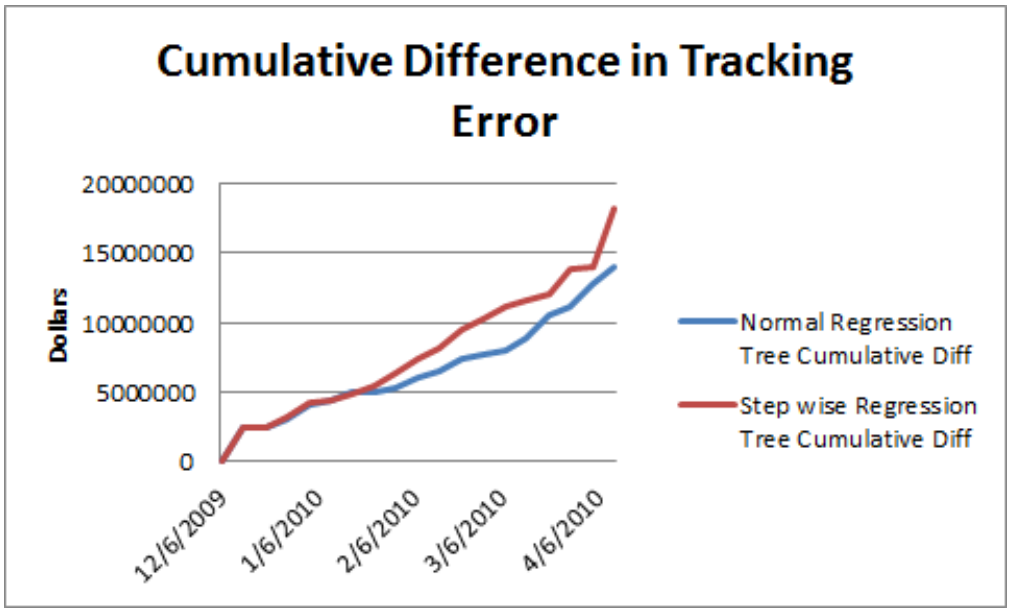


Figure 50 Difference in Tracking Error

The final section will offer a discussion of future work and lessons learned from this research.

Conclusions

The primary goal of this chapter was to present a case-study on recovering a trading strategy from trade-level data. To accomplish this goal, several unorthdox techniques had to be used. It was necessary to creating an action feature space to improve the accuracy of the recovered decision space. Treating actions as if they have properties made intuitive sense, although this has rarely been done in practice. And as this study demonstrated, generating an action feature space made it possible to use recursive partitioning directly in the recovery process.

Two variations of regression trees were used to determine the desired delta exposure. It was determined that although stepwise regression can be used to augment a regression tree's prediction, it has the tendency to overfit. Specifically, if out-of-sample data is provided to the recovered algorithm there is a potential for unpredictable behavior. Therefore, researchers should further explore a hybrid model, where the step-wise regression tree is used on within-sample data and the standard regression tree model

is used on out-of-sample data. The standard regression tree model uses the mean of dependent variables that are categorized in the same node as the prediction for that class. This leads to a lag in recognizing state transitions. However, it prevents the aforementioned unexpected behaviors.

The results also illustrate that recursive partitioning and regression trees were unable to perfectly determine the delta exposure. This could mean three things: First, it could indicate that the trading algorithm uses information outside the state space that was defined. Second, the algorithm could have partitions that are nonlinear and thus would not be found in the linear recursive partition and regression trees used in this paper. Third, since the delta of any particular option is based on an estimate of the volatility of the underlying stock, the volatility estimate used in this study could have differed from that of the trading competition participants.

This type of technique can help regulators better understand modern-day trading algorithms. Recovering trading algorithms will allow regulatory agencies to enforce existing policies; in particular, it will enable them to identify cases where traders have manipulated the market through misuse of algorithms. Furthermore, it will allow regulators to create artifical environments in which to test regulatory polcies. These environments can contain real-world trading algorithms, which will permit these agencies to conduct in-depth cost-benefit analyses prior to implementing new regulations. Such an artifical environment would also allow regulators to indentify scenarios in which the market might move in an unexpected way.

Regulators are not the only party that might benefit from this type of techique. Exchanges could create the same type of artificial environment as described above. Within this enviorment, pricing insentives could be explored with the objective of making the exchange more profitable. Furthurmore, the exchange could potentially sell artificial scenarios to private investors interested in training algorithms, an arrangement that would be mutually beneficial both to the exchange and to private

investors. The more testing a private investor can do prior to launching an algorithm, the less likely the algorithm is to malfuction and cause a larger problem. Furthermore, by testing the algorithm over a wider range of scenarios, an investor can more effectively determine and optimize its profitability.

This chapter has illustrated some of the challenges associated with recovering strategies. In the the hedge fund tournament, it was possible to determine which asset would be traded. However, the authors benefited from obsererving the full behavior model of the alogirthim. In real-world enviornemnts the actor's behavior model will be repersented as trajectory through a large state-space. There has been limited research into recovering behavior models in partial observation environements. The chapter 5 examines the more realistic case of recovering behavior modes in partially observed environments and compares different machine learning algorithims in partial observation enviornments.

Chapter 5: Apprenticeship Learning in Partial Observation Markov Environments: A Comparison of Different Machine Learning Techniques

Agent-based models' validity is increased by empirically deriving behavior models from repeated observations of real-world actors. We define a behavior model as a model that assigns an action for every possible state. The researchers acknowledge that a real-world actor may have a behavior model that is stochastic, having a probability of selecting a given in action in each state [131]. However, the researchers focus on deterministic behavior models, where the actor will always perform the same action in a specific state space, such as a High-Frequency trading algorithm.

Recovering a deterministic behavior model is of interest to the United States Military, who recently created a Small Business Innovative Research Project on the topic. The goal of the project was to, "develop warfighting decision support tools and reverse-engineering processes by allowing AI technology to observe automated smart red forces compete with smart blue forces within a simulation [132]." Over the past decade the Marine Corps and other the U.S. Military branches have procured several individual simulator systems that are either proprietary in nature or have dissimilar behavioral and entity models (e.g. JSAF, ModSAF, OneSAF, DISAF, JCATS) [133] , making it difficult to ensure that all platforms simulate opponents with similar behaviors and abilities. As computer and simulation technology matures, the U.S. Marine Corps is seeking to move to a standard architecture, which will allow Marine Corps trainees to face a consistent adversary across many different scenarios.

Utilizing behavior model recovery methodologies, such as deep learning, the U.S. Marine Corps is seeking recover strategies (behavior models) from previously procured simulations, which will allow them to implement these models into their common architecture, preventing the need to redevelop an existing behavior model for the common simulation architecture, thus recouping some of the Marine Corps initial

investment. The methodology will also allow the Marine Corps to capture the behavior model of a trainee allowing for more precise training.

Agent-based model practitioners can also leverage these machine learning algorithms to recover an agent's behavior model from observations of real-world actors. There are circumstances where it is insufficient to develop generalized agents to represent a class of actors, as discussed in chapter 3. For example, when developing new regulations for oligopoly markets, like the electricity market [134], such a model cannot accurately show how each firm will bear the cost of the regulation. Depending on a firm's specific strategy they could be more affected than a competitor. Since regulatory shocks to markets with a small number of participants can have unpredictable consequences [134], it is essential to have a higher fidelity model when studying regulations of oligopoly.

Previous research in behavior model recovery works under two assumptions. Firstly, they assume the training and testing dataset contains the same state space. Secondly, the studies assume they have enough observations to examine the state space sufficiently. However, in real-world environments, these assumptions are violated.

Take the U.S military example mentioned above. The behavior models of previously developed agents will be captured and placed in a unified simulation. However, this newly unified simulation will likely place the recovered behavior models in situations that were not seen in the previously developed simulation. Thus, the recovered behavior model is selecting actions for states that it was not trained on. Additionally, the machine learning techniques cannot rely on thousands of replays to recover the behavior model of a soldier using the training simulator. While machines can quickly play the same scenario many times, we would expect humans to play the same scenario for a small number of times. Therefore, it is unlikely that we will be able to see the trainee make a decision in all possible states.

In this chapter, we compare Inverse Reinforcement Learning, Classification Trees, and Neural Networks to assess how well each algorithm recovers an 'expert' agent's deterministic behavior model in a partial observation Markov environment. We then compare how well each algorithm predicts the expert agent's actions in a previously unseen environment. For this study, we again use StarCraft: Brood War as a testbed. The next section presents previous works in the area of behavior model recovery.

Literature Review: Behavior Model Recovery

Learning a behavior model through observation of an expert is a form of apprenticeship or imitation learning and falls under the more generalized methodology of supervised learning, a domain expert corrects an agent's action. The field of apprenticeship learning developed in the 1980s to address the 'knowledge-acquisition bottleneck,' which describes the time-consuming process of completely defining a domain for an artificially intelligent system [135], [136]. In large domains, it is impossible to program a response to all possible states. Therefore, apprenticeship learning attempts to learn generalized rules, through observations of an expert, that will allow an AI system to respond like an expert in an unseen state.

Applying apprenticeship learning to mimicking human behavior is not a novel concept. In 1991, Brian Arthur [137] proposed calibrating artificial agents to behave like human subjects. His goal was to develop an agent that showed the same limited rationality as human subjects when making economic decisions. Brian Arthur tested calibrating an agent to resemble humans in the multi-arm bandit problem, where a decision-maker has N possible choices, and each choice generates a different payoff based on an unknown static distribution. Through repeated trials, a decision-maker attempts to maximize their reward by selecting the choice with the highest payoff. However, the decision-maker must balance exploiting a source that returns a substantial reward and exploring other sources that may return a higher reward.

Brian Arthur found that his artificial agent could resemble human behavior if a learning decay were set to zero, which means the agent would value information discovered in the 100th trial as much as

it valued information discovered in the 1st trial. The finding indicates that human subjects' value new information as much as historical information, which could lead them to not converged to the optimal solution. Author [137] used his calibrated agent to predict human subjects' ability to converge to the optimal solution, finding that if the optimal solution is 15 percent better than the next best choice the artificial agent converged to the correct solution.

Roth and Erev [138] used a simple imitation algorithm to mimic human behavior in two simplified market games and a bargaining game. The algorithm increased the probability of performing an action if a positive return was received, which is analogous to choosing to repeat a successful action. In the first market game, there are ten buyers and one seller. All buyers place a single bid to the seller, which is unknown the other buyers. The seller than chooses to accept the highest bid or reject all bids. If a seller accepts the highest bid that buyer receives a payoff equal to the value of the object minus what the buyer paid for it and all other buyers receive a payoff of zero. The seller receives a payoff equal to what the buyer paid for the object. In the event, the seller rejects all bids all participants receive a payoff value of zero.

All participants repeat this game ten times, presumably learning with each trial. The human participants quickly converge to the Nash-Equilibrium, which is the game-theoretic best outcome if all participants behave rationally [139]. By tuning the algorithm to perform the same few actions that human participants did the algorithm converged to the Nash-Equilibrium at approximately the same rate as human participants [138]. The same finding was found in the second market game, which was a cooperative bidding game.

In the bargaining game, the initial bidder proposed how to split a pot of money and the second participant chose whether to take or reject the bid. If the second participant accepted the bid, the payoff was given based on the proposed split and if the participant rejected the bid than everyone received zero.

Both Humans and the calibrated imitation algorithm were found not to reach a Nash-Equilibrium, potentially indicating that humans follow a 'repeat successful action strategy' when approaching repeated trial games [138].

The previous strategies illustrate that an algorithm, which increases the probability of selecting an action that was successful, is a reasonable approximation of basic human strategies. However, this mimicking behavior does not provide guidance in a more complex environment. For example, the simple game of chess has many different states, and it is not immediately apparent if a given action returns a positive reward. Additionally, unlike the previously described games, the actions in chess are dependent on the current state space and not merely historical outcomes of past selections. Therefore, as the domain increases in complexity, a new framework is needed to imitate human behavior accurately.

Reinforcement learning is built on the Markov Decision Process (MDP) framework described in chapter two [140]. As stated previously MDPs are defined by a set of possible states *S*, actions *A*, and a transition matrix *P* that determines the probability of moving to state s' after taking action a in state s. Agents following an MDP are said to be maximizing a reward function R while incorporating a discount function γ [141]. The reinforcement learning framework allows for a natural representation of state-dependent actions. Additionally, by calibrating the reward function, artificial agents can be made to mimic expert behaviors.

In the reinforcement learning paradigm, agents learn a policy (π), which specifies an action in each state that leads to the highest possible reward. For this research, we will assume a deterministic policy. Researchers commonly use the method called value iteration to recover the optimal policy. A state's value *V(s)*, is the total expected discounted reward attained by following the optimal policy starting from s. Using the recursive relationships below we can recover the optimal policy [140]:

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s,a,s')V(s'),$$

$$V(s) = max_{a' \in A} Q(s, a').$$

The *quality* of action a in state s, defined as *Q(s, a)*, is the expected reward received for taking action a in state s plus the discounted reward of all following states weighted by their likelihood. The value of each state is based on the highest quality action.

In the reinforcement learning paradigm, an agent explores possible actions in each state, updating its Q-value as it receives rewards. The researchers Watkins and Dayan demonstrated that a reinforcement learning agent converges to the correct Q-values assuming all possible state-action pairs are tested infinitely often, and new rewards are combined with historical values using a slow exponentially weighted average [142]. Once the Q-values have converged the optimal policy is to select the action with the highest *quality*.

Reinforcement learning has been applied to a wide variety of task from simple games such as backgammon to more complicated tasks such as robotic control [143]–[145]. Through repeated trials, the agent uses the expert defined reward function to discover a policy that achieves a performance level comparable to an expert.

One challenge not addressed by reinforcement learning is that real-world actors may have different, unknown reward functions [146]. For example, one firm may have a higher risk aversion than another firm, which will lead to different behaviors. The reinforcement learning framework does not provide a mechanism for discovering an actor's reward function from repeated observations, making it difficult to ensure a reinforcement learning agent faithfully replicate a real-world actor in a simulated environment [147].

Inverse Reinforcement Learning

Inverse reinforcement learning attempts to recover an actor's reward function, which they are assumed to be maximizing, by observing the actions the actor takes in multiple states. Value iteration,

described previously, allows for the discovery of a policy that closely matches the actor's observed policy. The recovered policy can be implemented in an agent-based model allowing for a higher fidelity simulation [147].

Like reinforcement learning, the inverse reinforcement learning framework is built on Markov Decision Process and Q-Learning. As mentioned previously inverse reinforcement learning assumes that an actor is following an optimal policy for its reward function. Therefore, for the following must be true [148]:

$$a_1 \equiv \pi(s) \epsilon \; argmax_{a \in A} \sum_{s\prime} P_{sa}(s') V^{\pi}(s') \;\; \forall \, s \in S$$

$$\Leftrightarrow \boldsymbol{P}_{a_1} \boldsymbol{V}^{\pi} \;\succcurlyeq \boldsymbol{P}_{a} \boldsymbol{V}^{\pi} \;\; \forall \, a \in A \backslash a_1$$

$$\Leftrightarrow (\boldsymbol{P}_{a_1} - \boldsymbol{P}_{a})(\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1})^{-\mathbf{1}} \boldsymbol{R} \;\succcurlyeq 0 \;\; \forall \, a \in A \backslash a_1$$

The above equations state that if an actor is observed selecting an action at a given state, then that action must return an expected discounted reward equal to or greater than selecting any other action in that same state. Inverse reinforcement learning seeks to find a reward function that makes the above statement true.

Over a finite state space, the above formulation results in a set of reward functions that meet the criterion. However, a reward function of zero in each state space meets the criterion mentioned above. Therefore, a secondary constraint is added to select a unique reward function. Ng and Russel [148] propose to select the reward function that $Max \; \sum_{i=1}^{N} Min_{a \in \{a2, \ldots, ak\}} \{ (\boldsymbol{P}_{a_1} - \boldsymbol{P}_{a}) (\boldsymbol{I} - \gamma \boldsymbol{P}_{a_1})^{-\mathbf{1}} \boldsymbol{R} \} - \boldsymbol{\lambda} \|\boldsymbol{R}\|$. The selected reward function maximizes the expected reward between the actor's observed policy and that of the next closest policy while minimizing the complexity of the reward function.

While the above formulation is easily solvable for an environment and policy that is entirely known, in most real-world applications a policy will be given by a set of trajectories through the state space. Ng and Russel [148] proposed using iterative simulations to solve the inverse reinforcement learning problem where the policy is only known through sample trajectories.

They assume the goal is to define the reward function that maximizes $E[\boldsymbol{V}^{\pi^*}(S_o)]$, which is the expected value of following the sample trajectory policy starting at a single, fixed state. The pseudo code for the Ng and Russel's algorithm is presented below [148].

Algorithm: Pseudocode Inverse Reinforcement Learning from Sample Trajectories

1) Perform a trajectory sample following π^1 from a base case policy that can be random for simplicity
2) Calculate **R** that $Max\ \sum_{i=1}^{K} P(\boldsymbol{V}^{\pi^*}(S_o) - \boldsymbol{V}^{\boldsymbol{\pi_i}}(S_o))$ $\begin{cases} P(x) = x,\ x \geq 0 \\ P(x) = 2x, otherwise \end{cases}$
3) Perform a new trajectory simulation following π_{i+1}, which is the optimal policy for the reward function calculated in step 2.
4) Repeat step 2 and 3 until a user-defined criterion is met.

The above algorithm attempts to find a reward function that maximizes the total value between the expert trajectory and all other trajectories. If at any time a simulated trajectory receives a higher reward than the expert trajectory, a penalty is given that is twice the difference. Ng and Russel used the above methodology to recover the reward function in the common grid-world and mountain-car problems [148].

Abbeel and Ng modified the previous algorithm to apply it to apprenticeship learning further. Instead of finding the **R** that $Max\ \sum_{i=1}^{K} P(\boldsymbol{V}^{\pi^*}(S_o) - \boldsymbol{V}^{\boldsymbol{\pi_i}}(S_o))$, they find the reward function that maximizes the difference between the expert trajectory and that of the closest trajectory, then use reinforcement learning to generate an optimal policy for the recovered reward function. The policy is

added to the set of recovered policies and the process is repeated until a discovered policy differs from the expert by less than ε [147]. The pseudo code can be seen below.

Algorithm: Pseudocode Apprenticeship Learning using Inverse Reinforcement Learning

1) Perform a trajectory sample following π^1 from a base case policy that can be random for simplicity
2) Calculate $\boldsymbol{R}_i$ such that $t = Max\left[Min_{i\epsilon(0\ldots N)}\left[\boldsymbol{V}^{\pi^*}(S_o) - \boldsymbol{V}^{\boldsymbol{\pi_i}}(S_o)\right]\right]$
3) Stop if t < ε
4) Use reinforcement learning to finding the optimal $\boldsymbol{\pi_i}$ for $\boldsymbol{R}_i$
5) Compute estimate for $\boldsymbol{V}^{\boldsymbol{\pi_i}}(S_o)$
6) Set i = i +1 and go back to step two.

The algorithm will terminate once a reward function and associated policy is discovered, which generates an estimated reward that is sufficiently close to the observed expert trajectory. Using this algorithm Abbeel and Ng demonstrated the ability to replicate driving behaviors in a driving simulation.

Inverse reinforcement learning has been applied to a wide variety of apprenticeship learning problems. Abbeel et al. developed an autonomous RC helicopter that learned to perform acrobatic skills from observing an expert [149]. Using inverse reinforcement apprenticeship learning algorithm, Abbeel et al. iteratively discovered reward functions whose optimal policies brought them closer to the expert policy. They had to use an expert to help select reward functions because many of the proposed policies where not safe to fly.

Vasquez et al. applied inverse reinforcement apprenticeship learning to help a robot learn navigational rules in crowded human environments [150]. They assert that people navigate using unofficial social rules (e.g., move to the right when walking). Using a simulation of pedestrian traffic, they

trained a simulated robot to follow these unofficial rules by learning from example trajectories. The robot's reward function traded off relative position, velocity, crowd density, and social repulsion when creating its reward function. Vasquez et al. [150] report the reward function learned through inverse reinforcement learning outperforms manually calibrated reward functions. However, a human control robot consistently provided a smoother route, indicating that humans have additional knowledge not captured in the reward function.

Lee and Popović extended the previously mentioned inverse reinforcement apprenticeship learning method to behavior styles of animated character control [151]. Behavior styles are a sequence of motions which could be as simple as walking from one location to another. However, behavior styles can also contain nuisances, such as walking cheerfully or stealthily. The use of reinforcement learning based character controls has increased, but their sensitivity to small changes in the reward function makes them difficult to use [151]. A minor change in the reward function can lead to drastically different results. Lee and Popovic' proposed to use inverse reinforcement apprenticeship learning to achieve the desired behaviors from training on a small set of trajectories.

Lee and Popovic´ [151] modified Abbeel and Ng's algorithm to ensure that a single optimal policy was discovered. Lee and Popović's called their algorithm convex inverse reinforcement learning algorithm. They first use the standard inverse reinforcement apprenticeship learning algorithm to build a convex hull around the reward space. They then proposed a new reward function that is normal to the currently proposed reward function and it its closes neighbor. The process repeats until the reward function is less than ϵ. Lee and Popović proved their algorithm was guaranteed to converge to an optimal solution for their application. Additionally, they demonstrated that inverse reinforcement apprenticeship learning algorithm could extract a reward function from sample trajectories that allowed the replication of complex behaviors like running or diving away from an explosion.

Other researchers have expanded the basic inverse reinforcement learning algorithm, making it more robust. Ramachandran and Amir [146] demonstrate how to combine prior knowledge and example trajectories to select a reward function, from a set of known reward functions, that best explains the observed behavior. They assume the expert is selecting actions that maximize its Q-value as defined earlier in this paper. Therefore, the reward function that maximizes the Q-value is the most likely reward function the expert is following. Using an exponential distribution of likelihood, they model the probability an expert is following a given reward function as $P\big((s_i, a_i)\big|\boldsymbol{R}\big) = \frac{1}{Z_i} e^{\alpha Q(s_i, a_i, \boldsymbol{R})}$, where α is the confidence interval that the expert will select the action with the highest Q value and Z is a constant normalization value [146]. They found that the Bayesian inverse reinforcement learning can outperform the standard inverse reinforcement algorithm, assuming a set of possible candidate reward functions is known.

Ziebart et al. [152] further expand on Ramachandran and Amir's work by applying the exponential distribution of likelihood to all possible reward functions. They seek to maximize $\sum logP(\zeta|R, T)$, where ζ is the example trajectory, and T is the transition matrix. Ziebart et al. technique is known as maximum entropy inverse reinforcement learning. They recursively compute the probability of the trajectory given the MDPs transition matrix and proposed reward function. Starting from the end state and working backward they calculate $\sum P(s_k \big| s_i, a_{i,j}\big) e^{\boldsymbol{R_i}}$, which is the probability mass associated with each branch step along the path. Assuming a stationary state space, maximizing this probability for the observed trajectories guarantees a policy that will achieve the same reward. Their findings illustrate that maximum entropy inverse reinforcement learning can be applied to driving route recommendations. It outperformed both Bayesian inverse reinforcement learning and maximum margin planning.

Up to this point, all the inverse reinforcement learning algorithms assume a linear reward function. Levin and Popović [153] expanded the field by developing Gaussian inverse reinforcement learning, allowing for nonlinear reward functions. The Gaussian Process Inverse Reinforcement Learning

(GPIRL) maximizes $logP\left(\zeta\middle|\boldsymbol{K}_{\boldsymbol{R},u}^{T}\boldsymbol{K}_{\boldsymbol{u},\boldsymbol{u}}^{-1}\boldsymbol{u}\right)+logP(\boldsymbol{u},\boldsymbol{\theta}|\boldsymbol{X}_{\boldsymbol{u}})$, which is the log likelihood of the trajectory plus the log likelihood of the Gaussian parameters at state $\mathbf{X_u}$. **K** is the covariance matrix, **R** is the non-linear reward function, **u** represents the rewards associated with the feature coordinates $\mathbf{X_u}$, and $\boldsymbol{\theta}$ is the kernel parameter for the Gaussian distribution [153]. GPIRL was able to recover a policy that mimicked a human driver in a driving simulation and outperformed maximum entropy inverse reinforcement learning. The agent was able to learn by observing a human subject not to speed if near a police vehicle.

Choi and Kim [154] extended inverse reinforcement learning into the field of partially observed Markov Decision Process. In a realistic environment, it is unlikely for an MDP transition probability to be fully known. Instead, the agent will learn the transition probability through observations and can infer the likelihood of transitioning to s' based on a belief function, which is described below.

$$b_z^a(s') = P(s'|b,a,z) = \frac{O(s',a,z)\sum_s T(s,a,s')b(s)}{P(z|b,a)}$$

In the equation above, z is the set of observed trajectories and $O(s',a,z) = P(z|s',a)$, which is the probability of observing trajectory z given the agent performs the action a and arrives in state s'. The value of $b(s)$ is the probability that the state is s at the current timestep. Under this formulation the value function is calculated as $V^*(b) = Max_a\left[\sum_s b(s)R(s,a) + \gamma\sum_{s',Z} T(s,a,s')O(s',a,z)V^*(b_z^a)\right]$, which is the maximum expected discounted reward starting in state s. The expert is believed to be following the policy that $Max\ \sum_{i=1}^{K} P(\boldsymbol{V}^{\pi^*}(S_o) - \boldsymbol{V}^{\boldsymbol{\pi}_i}(S_o))\begin{cases} P(x) = x,\ x \geq 0 \\ P(x) = 2x, otherwise \end{cases}$.

Chloe and Kim examined the performance of partial observation inverse reinforcement learning on benchmark problems like grid world and 1-dimensional maze. Learning from an expert's trajectory, partial observation inverse reinforcement learning converged to the expert policy within 300 iterations. However, the agent was given an additional 2000 non-expert trajectories to help build its belief function.

Inverse reinforcement learning provides a versatile method for recovering a behavior model that mimics a real-world actor. However, previous research focuses on comparing the recovered behavior model to the expert, without changing state parameters. The stationary state assumption is not valid in agent-based simulations because the goal of the model is to test the agent's behavior and its effects on the environment and other agents in unseen states. This research extends the field by applying inverse reinforcement learning to partially observed Markov environments. To further complicate the task, unlike Choi and Kim we assume our agent is only able to observe a small number of trajectories before starting its value iteration. We then compare the agent's behavior to that of the expert in a previously unseen environment. We use backward propagation Neural Network and CART as benchmark algorithms. The goal is to see which algorithm recovers a policy that best mimics the expert policy in both a seen and an unseen environment. The next section presents the experimental design.

Experimental Design

Like in chapter 2, we again use StarCraft: Brood War as the experimental environment. Motivated by the previously mentioned Small Business Innovative Research project, we focus on learning a behavior model that mimics an expert's control of a small number of units in combat. The task of controlling units is known as micromanagement. While this is less researched than higher level strategy recovery, such as build order, there has been some work in the area. The next subsection presents a quick literature review of apprenticeship learning focused on micromanagement in video games.

Literature Review of Apprenticeship Learning for Video Games

Academic researchers have primarily used two types of video games, first-person shooters and real-time strategy games. First-person shooters are high fidelity games where a player explores a three-dimensional environment and engages in armed combat against a human or computer agent. Real-time strategy games, such as StarCraft, require a player to develop a military by gathering in-game resources,

researching technology, and training/constructing military units. Like first-person shooters, the goal of real-time strategy games is to eliminate the opponent.

Researchers have used these high-fidelity games to explore challenges in behavior model recovery, in military-like simulations. To manage the complexity most researchers, decompose the game into different levels of abstraction and recover the player's strategy for each layer. Thurau et al. [155] defined these three levels as strategic, tactical, and reactive. Reactive behaviors are low-level behaviors such as how far a player leads a target when aiming a weapon. Recovering reactive behavior models is outside the scope of this project because they have no strategic or tactical importance and are more indicative of physical motor skill.

The strategic layer defines the player's high-level action and is the focus of this paper. Examples of strategic objectives include attacking an opponent, retreating from an opponent, and identifying and moving to a more tactically advantageous position. A player's strategic behavior model determines under what conditions a player performs each high-level action. Thurau et al. [156] used a neural gas and potential field algorithms to determine the route a human took when moving from strategic objective to strategic objective, resulting in a set of waypoints that an autonomous agent could follow when moving towards a strategic objective, regardless of the agent's initial starting position.

Figure 48 illustrates Thurau et al.'s concept. By observing a human player, the ideal path to an objective can be identified. In the case below, players do not take the shortest path; instead, they take the longer but more concealed route. The blue arrow in the left box is the autonomous agent's starting point. Notice the shortest path (red line) is straight but human players were observed taking the longer path (green line). The result of this observation is the right box, which illustrates potential fields, the green arrows that act like currents in the ocean. Give a random starting point the agent will follow the local

current. Potential fields allow agents to move in a human-like manner regardless of their initial starting point.

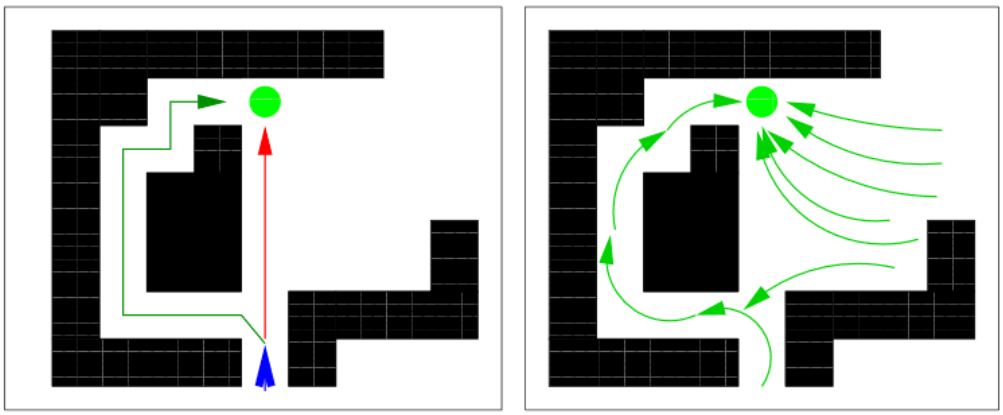


Figure 51: Potential field illustration Originally Presented in [150]

Gorman and Humphrys [157] used Neural Networks to identify when players in a real-time strategy game decided to attack their opponent. Their NN model considered the weapon the player had equipped, the distance of the enemy, and the relative velocity (both relative speed and direction) of the enemy when deciding whether to engage the opponent. The model was trained by watching two human players compete in a first-person shooter game. The model showed a high degree of accuracy in determining when each player would engage their opponent.

Benjamin Geisler [158] compared artificial Neural Networks, classification trees, and Bayesian belief networks ability to recover behavior model of an expert and predict the expert's action in a first-person shooter game. An expert played capture the flag in the game *soldiers of fortune 2*, while state-action pairs were collected every 100 milliseconds. In capture the flag, players attempt to collect the other team's flag and return it to their base. The states were defined by a set of features that described health and location of both the opponents and the player, as well as distance to a goal location. Geisler focused on predicting four actions:

1. when the player will accelerate,

2. what direction the player will move,
3. what direction the player will face,
4. when will the player jump?

The dataset was divided into a training set (83%) and a testing set (17%). Each algorithm was trained on the same training set data and predicted the actions in the unseen testing set. The accuracy rate of each algorithm is compared with each other and a baseline, which is the error rate achieved if the majority category was always selected [158]. For example, the player does not jump most of the time, which makes not jumping the baseline action and has an error rate of 3%. The artificial Neural Network algorithm was the best in every category and classification trees was second best. Excluding the case of determining when the player will jump, all three machine learning algorithms outperformed the baseline. In the jumping case, all the algorithms over-predicted the prevalence of jumping.

To improve the accuracy of the machine learning algorithms, Geisler implemented an ensemble algorithm framework, which uses multiple trained algorithms to achieve better prediction accuracy. The first method examined was bagging, which is where an algorithm is trained on a random subset of the training set [158]. The training is performed multiple times, creating a collection of models. The testing set is classified based on the collection's most commonly predicted outcome. Bagging improved the accuracy of the Neural Network and classification tree algorithm by 3% but did not affect the Bayesian belief algorithm. Geisler also examined boosting, which similar to bagging. However, boosting increases the likelihood of misclassified instances being selected again for the next iteration [159]. Using boosting ensemble Geisler improved the artificial Neural Network's performance by 10% over the bagging ensemble.

Researchers have also applied inverse reinforcement learning to behavior model recovery in video games. Tastan and Sukthankar [160] applied Ng and Russel's algorithm to policy recovery of two task. The

first was route planning, where they collected trajectories of an expert moving around a map collecting weapons and medicine. They then recovered a policy which closely mimicked the expert's behavior and found that the policy outperformed the built-in AI system when performing the same task on the same level. One crucial assumption they made was to assume the reward for an unobserved state is zero, which helped the agent to mimic the expert more closely.

Tastan and Sukthankar [160] also applied Ng and Russel's algorithm to recovering actions in combat. After observing an excerpt player's behavior, they segmented the game space into 12 states combing the agent's health and distance to the enemy. The agent could choose one of four actions, standing still, moving, shooting, or moving and shooting simultaneously. Their agent was judged to be more human-like than the built-in AI system.

Our research builds upon the previous research by first not assuming unseen states have a reward of zero. For a small number of finite-size trajectories over a state space, it is likely that an expert will not traverse every state, which does not mean the unseen states have zero value. Secondly, we directly compare Neural Networks, classification trees, and inverse reinforcement learning, which have shown promising results in apprenticeship learning tasks. To the author's knowledge, this is the first comparison of these algorithms in a partial observation Markov state space. Lastly, we compare how the recovered models predict the actions of the expert in an unseen environmental state, which is vital in an agent-based simulation. The next section describes the experimental procedure.

Experimental Procedure

Leveraging Wender and Watson paper, we developed a Q-learning reinforcement agent that learned to control a single unit and defeat a larger force [161]. The machine learning agent was used to generate the 'expert' trajectories used by the apprenticeship learning algorithms. The authors intentionally chose to use a reinforcement learning agent, instead of human subjects, to develop the trajectories. By using the reinforcement learning agent, the authors know the reward function that the

agent is attempting to optimize. Human subjects have unknown reward functions, making it difficult to assess how well inverse reinforcement learning recovered reward function of the expert. As previously mentioned, inverse reinforcement learning recovers reward function that makes all other trajectories less valuable than the expert's trajectory. However, it cannot be definitively stated that the expert is following that recovered reward function.

An additional benefit of using a reinforcement learning agent to generate the trajectory is that the authors can assess the accuracy of each apprenticeship learning algorithm. Accuracy is defined as the percentage of time the apprenticeship learning algorithm performs the action that the reinforcement agent would have performed in the same state. Previous research in apprenticeship learning rarely assesses the accuracy of their agent (i.e., how likely is it to predict the expert's action in a given state). For most researchers the goal is not to exactly mimic the expert's behavior, rather it is to achieve the same outcomes as the expert (e.g., perform acrobatics or drive a car safely). However, since the authors' goal is to use the behavior model within simulations to examine how the real-world actors might change their behavior in reaction to an environmental change, it is imperative that the recovered behavior models accurately reflect the expert's behavior model.

The Q-learning update policy that the reinforcement learning agent followed was $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a_t) - Q(s_t, a_t)]$. The value α is known as the learning rate and was set to 0.05. The discount value γ was set to 0.9. Both values were taken from Wender and Watson's paper [161]. Like in Wender and Watson's paper our reinforcement learning agent controls a StarCraft unit known as a Vulture, which is faster and stronger than a single Marine. Through repeated trials, the agent learns a policy that allows it to beat six Marines. Figure 53 shows the two opposing forces.

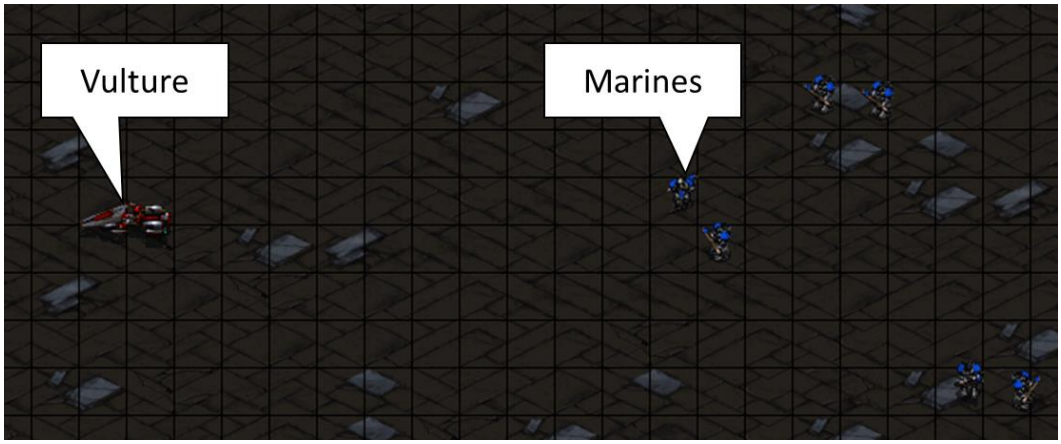


Figure 52 StarCraft Units

The reinforcement learning agent uses the states initially presented in Wender and Watson's paper. However, due to ambiguity in Wender and Watson's state definition, the authors had to make some assumptions [161]. A state is defined at a specific time and is made up of the values of the four features listed below.

- Is the weapon being reloaded – like in real-world environments, units in StarCraft need to reload their weapon after firing. Each unit has a different reload interval
- How many enemy units are in range – gives the number of enemy units that are currently within range of our agent's weapon
- What is the distance to the closest enemy unit – the distance is grouped into four ranges with 110, 140, and 170 pixels being the demarcation points.
- What is the health of the unit – the health is group into four categories of 25% (i.e., 0 -25%)

The reinforcement learning agent had two actions, attack or retreat. It is important to note that an action is only performed when there is at least one enemy unit within range of the agent. If the agent performs the retreat action, the following pseudocode describes how the agent retreats.

Algorithm: Pseudocode Retreat from Enemy Unit

1. If the agent is close to the corner of the map move to step 2. Close is defined as being located within the 5th or 95th percentile height and width pixel for the map
    a. Calculate a line between the agent and all other enemy units
    b. Find the value on each line that is 200 pixels further from the enemy unit
    c. Average these points together and move to the point. (If point out of range for map move to the maximum or minimum value possible)
    d. End action
2. Retreat but move away from the corner
    a. If the agent is traveling clockwise move 200 pixels towards the corner that is on the agent's right side, otherwise move 200 pixels towards the corner that is on the agent's left side.
    b. Repeat step two when the agent performs future retreating actions until the agent is between the 40th – 60th pixel distance in map height or width. (Note: the agent can still choose to attack)
    c. End action

The agent follows a weighted vector to retreat from the enemy units, which illustrated in Figure 3. If the agent's final location places it in a corner defined as being in either the 5th or 95th percentile in height and width distance, then the agent will move towards the corner to the right if the agent is traveling clockwise or to the left if the agent is traveling counter clockwise.

In figure 54, if the agent continued along its current path, it would find itself in the lower left-hand corner. Since the agent approached the corner from a clockwise trajectory, future retreat actions will move the agent 200 pixels towards the upper left-hand corner. Once the agent reaches a location between the 40th – 60th distance percentile in height or width, then the agent will return to using the weighted

vector method described in step one. It is important to note that at any time the agent can choose to attack the enemy.

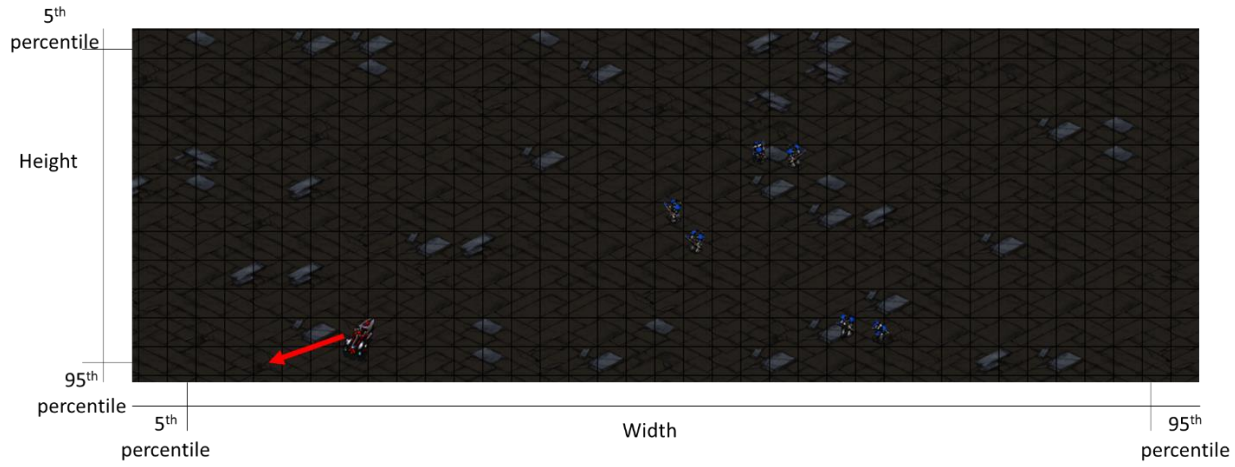


Figure 53 Agent's Retreat Action

When the agent performs the attack action, it identifies the enemy unit within its range that has the lowest health. It then targets and fires on that agent. The goal is to limit the amount of damage taken by the enemy by targeting and killing the weakest enemy first. The reward function the agent follows is given by the equation below.

$$Reward_{t+1} = \sum_{i} \left(EnemyUnitHealth_{i_t} - EnemyUnitHealth_{i_{t+1}} \right)$$

$$- \left(AgentHealth_t - AgentHealth_{t+1} \right)$$

Like in Wender and Watson's paper, the agent was trained over 1000 trials and follows a greedy policy. The agent learns a policy that allows it to defeat the marines approximately 90% of the time, which was less than reported by Wender and Watson's findings [161]. We treat the trained reinforcement agent as the expert in this study, and its trajectories are passed to the models described in the next section. We have released both the source code and the StarCraft map used to train the reinforcement learning algorithm.

Model

This research uses the java version of the StarCraft: Brood War Application Programming Interface (BWAPI). Therefore, all the source code is written in Java and leverages Java libraries. The source code for the classification tree and inverse reinforcement learning models are made available for download as open source code. However, the Neural Network model is based off software written by ZaneAcademy and is available for purchase [162]. Therefore, we have opted not to provide the source code for the Neural Network model. The next section describes the apprenticeship models used in this paper.

Classification Tree

Classification and Regression Trees (CART) is a term credited to Leo Breiman [163] and describes a decision tree algorithm that can be used for either classification or regression. Decision trees have been used in many different rule induction applications. Giordana et al. [164] found that decision trees were able to develop diagnostic rules for pump maintenance. They collected 209 examples of an expert maintenance manager deciding on whether to stop a machine for maintenance. The rules generated by the decision tree was found to outperform hand-crafted rules. Michie et al. [165] used a decision tree algorithm to improve the decision process of American Express applicants. They were able to achieve 70% accuracy of borderline applicants. Fayyad et al. [166] used decision trees to classify celestial objects. After learning on a training set, they were able to classify objects 94% of the time correctly.

In this chapter, we focus on decisions trees used for binary classification (i.e., when is the unit attacking or retreating). The authors use WEKA a java library [167] that implements the C 4.5 tree classifier, originally presented by Quinlan [168].

The classifier algorithm will iteratively segment the tree until each leaf node contains only one class. In our experiment, there are only two classes, attack or retreat. The classifier segments the dataset based on maximizing the 'Gain Ratio' splitting criterion given below:

$$Info(D) = -\sum_{j=1}^{c} p(D,j) * log_2(p(D,j))$$

$$Gain\ (D,T) = Info(D) - \sum_{i}^{k} \frac{|D_i|}{|D|} * Info(D_i)$$

$$Split(D,T) = -\sum_{i}^{k} \frac{|D_i|}{|D|} * log_2\left(\frac{|D_i|}{|D|}\right)$$

$$\text{Gain Ratio } = Gain\ (D,T)/Split(D,T)$$

In the above set of equations D represents the data set and $p(D,j)$ represents the proportion of D that is in the j[th] class. The value of $Info(D)$ is the initial uncertainty about the class that a case belongs to. $Gain\ (D,T)$ is the cumulative decrease in uncertainty based on each segmentation made by the algorithm. The variable *T* is specifying the tree being tested. $Split(D,\ T)$ measures what is known as the split information, which is how much value was gain by splitting the dataset. The more the classes in a dataset are segmented by a split, the better the split information. Gain Ratio is maximized by decreasing uncertainty using splits that cause the largest segmentation in classes [168].

The tree returned by the C 4.5 tree classifier algorithm tends to over fits datasets. Therefore, a pruning technique is often used to generalize the results and improve the accuracy of the classification. Pruning starts at the leaf nodes and iteratively removing them until the complexity criterion is minimized. The equation below gives the complexity criterion used by the C 4.5 tree classifier algorithm:

$$C_\alpha(T) = L(T) + \alpha|T|,$$

Where $L(T)$ is the loss associated with tree T and $|T|$ is the number of leaf nodes. The variable α is known as the tuning parameter and was set to 0.25, which is WEKA's default. The authors test both a pruned and unpruned tree model.

Backpropagation Artificial Neural Network

The first artificial Neural Network was developed by Warren McCulloch and Walter Pits [169]. While there are several different variations of Neural Network models, this work focuses on using feedforward Neural Networks and implements a basic backpropagation algorithm.

The Neural Network seen below uses the feature space, previously described, as inputs to the first neuron layer. There is a single hidden layer made up of three neurons and one output neuron that specifies attack or retreat. The network is completely connected, but for readability we only show the links for the 1st neuron in layer one.

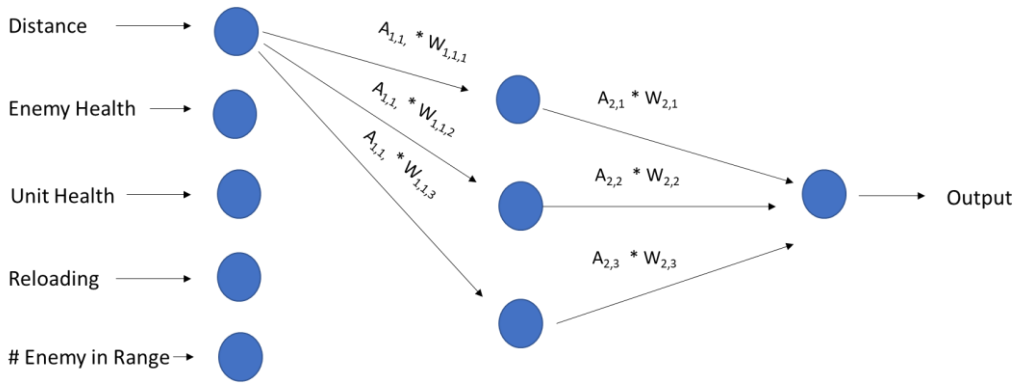


Figure 54 Neural Network Model

The activation function and the weighted sum equations are given below. For the first layer, there is no activation function, and the input values are passed as activation outputs without any modification.

$$Weighted\ Sum = \sum A_i \times W_i$$

$$Activation\ Output = \frac{1}{(1 - e^{-weighted\ Sum})}$$

The weighted sum is the sum of the activation output of each node in the previous layer multiplied by its associated weight. A sigmoid function is used to ensure that the activation output is bounded between (-1, 1) and since we bound the weights between [0, 1], the activation output bound in our model is [0, 1).

The authors used ZaneAcademy's Neural Network software, which implements a simple backward propagation algorithm to adjust the weights in the Neural Network. To help other researchers better understand the Neural Network and because we did not release the software source code, the backward propagation algorithm is given below.

Algorithm: Backward Propagation

1. Calculate derivative for Output Node:

$$derivative_{Output\ Node} = Activation_{output\ Node} * (1 - Activation_{output\ Node})$$

2. Calculate error of output node:

$$Error_{Ouput\ Node} = (Target\ Result - Activation_{output\ Node}) * derivative_{\ Output\ Node}$$

3. For all weights connected to output node adjust weights:

$$Weight = Weight + Learning\ Rate * Error_{Ouput\ Node} * Activation_{output\ Node}$$

4. Perform propagation for all hidden nodes
    a. Calculate derivative for hidden Node:

$$derivative_{Hidden\ Node} = Activation_{Hidden\ Node} * (1 - Activation_{Hidden\ Node})$$

    b. Calculate error for hidden Node [the weight value below is the weight between the hidden node (layer 2) and the output node (layer3)]:

$$Error_{hidden\ Node} = (Weight_{hidden,ouput} - Activation_{output\ Node}) * derivative_{\ Output\ Node}$$

    c. For all weights connected to hidden node adjust weights:

$$Weight_i = Weight_i + \ Learning\ Rate * Error_{hidden\ Node} * Input_i$$

Training samples were presented to the Neural Network one at a time, and backward propagation was performed after each sample. The learning rate was set to 0.8, which was the software's default value. The Neural Network was iterated over the training dataset 2000 times.

Case-Based Reasoning Inverse Reinforcement Learning

The author's implement a Case-Based Reasoning Inverse Reinforcement Learning (CBIRL) algorithm. To the authors' knowledge, this is the first use of this algorithm. As mentioned earlier traditional inverse reinforcement learning assumes that the transition matrix is known. While Choi and Kim [154] extended inverse reinforcement learning to partially observed Markov environments, they still assumed 2000 trajectories for environments with no more than 129 states.

The size of the study's state space is 34.9 million states, which makes it impossible to create an accurate transition matrix a small number of observed trajectories. A large number of simulations can be ran to approximate the transition matrix but this increases the computational complexity of the inverse reinforcement learning solution. Therefore, we apply case-based reasoning to approximate the next state. Cadena and Garrido used case-based reasoning and fuzzy logic to build a StarCraft Agent that was able to defeat the built-in AI system [170]. Wender and Watson [171] integrated case-based reasoning and reinforcement learning to learn a policy that allowed the algorithm to control a small number of units in StarCraft combat. Their findings suggest that case-based reinforcement learning converges faster than the standard reinforcement learning algorithm presented above. Building off Wender and Watson's research, we apply case-based reasoning to the inverse reinforcement learning problem.

Broadly speaking case-based reasoning is the process of using an experience to predict or select the next state or action [172]. There are four steps to case-based reasoning.

1. Retrieve – The agent selects the closest state where it has seen an attack performed and it selects the closest state where it has observed a retreat action. These observations can come from trajectories provided to the agent or actions the agent has performed in previous iterations.
2. Reuse – The agent uses the s' transition to by each of the retrieved states as the estimate of its next state if it performed the associated action.
3. Revise – Once an action is selected and the agent transition to s' it updates P(s, a)
4. Retain – The agent stores the updated P(s, a)

Case-based reasoning allows inverse reinforcement learning to simultaneously build the transition matrix P and solve for the reward function.

The CBIRL algorithm presented in this paper leverage Ng and Russel's algorithm [18] for recovering a reward function from expert trajectories. The reader will recall the Ng and Russel's assume that they can simulate policies through the state space. We use case-based reasoning to estimate the transition matrix as the algorithm simulates policies through the state space. We iteratively solve for the reward function that maximizes the linear optimization problem seen below:

$$Max \sum_{i=1}^{K} \boldsymbol{V}^{\zeta^*}(S_o) - \boldsymbol{V}^{\zeta_i}(S_o) \;\; s.t.$$

$$\sum_{i} \boldsymbol{R_i} = 1$$

$$\boldsymbol{V}^{\zeta^*}(S_o) \geq \boldsymbol{V}^{\zeta_i}(S_o) \; \forall \; i$$

The above linear optimization finds the reward function that maximizes the total reward difference between the expert's trajectory (ζ^*) and all other observed trajectories. The reward function must make the value of the expert trajectory greater than or equal to every other observed trajectory. Additionally,

the sum of the reward function feature weights must equal to 1, which assures that the trivial solution of $\boldsymbol{R} = 0$ is never returned.

As mentioned early, a state is made up of a set of features. Additionally, we assume that the reward function **R** is a set of linearly combined weights that are associated with each state feature {$W_{distance}$, $W_{unitHealth}$, $W_{enemyHealth}$, $W_{\#Enemies}$, $W_{reloading}$}. The agent is assumed to receive the reward upon entering the state, such that $R(s) = \sum_i S_i \times W_i$. The authors use a static scaling function to adjust for the difference in range between the features. For example, the distance can range from [0, 120] but reloading is a binary value.

The reader will recall that Ng and Russel's algorithm requires both an expert and base trajectory. The base trajectory is generated by modifying the trained reinforcement learning agent to select the suboptimal action approximately 35% of the time. The modified agent provides a trajectory that is similar to the expert's trajectory and is analogous to using a less skilled player to create the base trajectory.

As in Ng and Russel's algorithm, we simulate a new trajectory which attempts to maximize the reward function calculated in the current iteration. Abbeel and Ng [147] proposed using a simplified abstraction of the problem to calculate the optimal policy. In this work, the authors use case-based reasoning to provide an estimation of the next step depending on the action selected by the agent. Using multilevel sorting the closest state is the one that:

- Level 1 - s^{CB} matches the current reload state of the agent
- Level 2 - minimizes $|s_{\#Enemies} - s^{CB}_{\#Enemies}|$, has the minimum difference in the number of enemies in range,
- Level 3 – minimizes $|s_{unitHealth} - s^{CB}_{unitHealth}|$, has the minimum difference in the unit health,
- Level 4 - minimizes $|s_{enemyHealth} - s^{CB}_{enemyHealth}|$, has the minimum in between the enemy health,

- Level 5 - minimizes $|s_{Distance} - s^{CB}_{Distance}|$, has the minimum difference in the distance to the closest unit.

The order of the sort was selected using expert knowledge. When the agent calculates the expected reward for attacking it maps the current state (s) to the closest case-based reasoning state (s^{CB}), where an attack action was observed. The agent uses the observe P(s^{CB}, a) as a prediction for s' and calculates the estimated reward for taking action *a* in state *s* as $R(s,a) = \boldsymbol{s'} \cdot \boldsymbol{R}$. The agent does the same for the retreat action and selects the action with the highest estimated reward. The agent repeats this process until the game ends by either the agent dying or eliminating all enemy units.

The authors found that the reward function converged after ten iterations. The reward function whose trajectory killed the greatest number of enemies, while maintaining the highest health is selected as the reward function the expert is using. This selection process is analogues to Abbeel and Ng's using an expert to select reward functions for helicopter flight [149]. The next section describes experiments and presents the results.

Experiment

Recovering Behavior Models in Partially Observed Markov state space

For this experiment, the expert trajectories were generated from the previously described reinforcement learning agent. The RL agent uses a state space size of 224. Given the small size of the state space, the agent is guaranteed to traverse most of the states after several trajectories. Therefore, we augment the features described above and do not bucket distance or health in the trajectories provided to the apprenticeship learning models and also add an additional state space feature representing total enemy health, which increases the state space size to 34.9 million states. Adding the additional state feature simulates an imperfect representation of the expert's behavior space, which is likely to occur in real-world scenarios. However, the author concedes that many of the states are impossible to enter if both forces are attempting to win and that many of the states are almost identical. However, the challenge

for the apprenticeship learning models will be determining in what states the reinforcement agent will attack, given that the trajectories only traverse a small fraction of the states.

Before testing the machine learning algorithms, the authors established a baseline using naïve policies. The strategy of 'always retreat,' 'always attack,' and 'randomly select attack or retreat with equal probability' were used to control the Vulture unit in the StarCraft test scenario discussed above. Additionally, the reinforcement agent was passively run alongside each strategy and recorded when it agreed and disagreed with the strategy's action, which captures how often the naïve strategies performed an action that matches the action the reinforcement agent would have taken given the current state. The policy of 'always retreat' matched the reinforcement agent's action approximately 15% of the time, and random selection matched approximately 50%.

The 'always attack' strategy, and the reinforcement agent's policy agreed approximately 75% of the time. The 'always attack' strategy never wins the game (i.e., never kills all six marines) but the high agreement level illustrates that the reinforcement agent has a strong preference to attack. It also demonstrates that similar behavior models can have different emergent outcomes. The 25% of the time the reinforcement agent would have retreated, prevents it from taking damage while it reloads and allows the agent to increase its lethality from three marines to six marines. Therefore, prediction accuracy alone is an insufficient measure to judge the quality of the recovered behavior models because there is no guarantee that a highly accurate behavior model will achieve the same outcomes as an expert. A quality behavior model must have high prediction accuracy and achieve the same outcome as the expert. The authors examine the health of the unit and the number of enemies eliminated by the end of the match, as additional measures of behavior model quality.

The authors examine the relationship between the number of expert trajectories observed and the quality of the behavior models recovered by each of the apprenticeship learning algorithms. Each

algorithm recovers behavior models using 2, 4, 6, 8, and 10 expert trajectory observations. Sixty experiments are run at each level of expert observation, with new trajectories being generated by the reinforcement agent for each experiment. The authors did not make changes to parameters between experiments, but due to some randomness in the time it takes the agent to act, trajectories are slightly different each time.

The previously described artificial Neural Network, classification tree, and the case-based Inverse Reinforcement Learning algorithms are compared at each trajectory level. Like the naïve algorithms above the recovered behavior models are used to control the Vulture unit in the previously defined StarCraft scenario. The reinforcement agent passively observes the recovered behavior models for one game and records how often the recovered behavior model aligned with the reinforcement agent's action (i.e., selected the action the RL agent would take in the current state), for this study we will call this measure prediction accuracy. It is important to note that the reinforcement agent uses the smaller state space, while the apprenticeship algorithms both control the Vulture unit and learn from trajectories expressed in the much larger state space defined above. The process of trajectory generation, behavior model recovery, and prediction accuracy assessment is performed 60 times for each apprenticeship learning algorithm, at each trajectory level, totaling 300 trials for each algorithm. The average prediction accuracy for each of the apprenticeship learning algorithm is listed in table 12, with 100% accuracy meaning the recovered behavior model accurately mimicked the reinforcement agent in each observed state during the match.

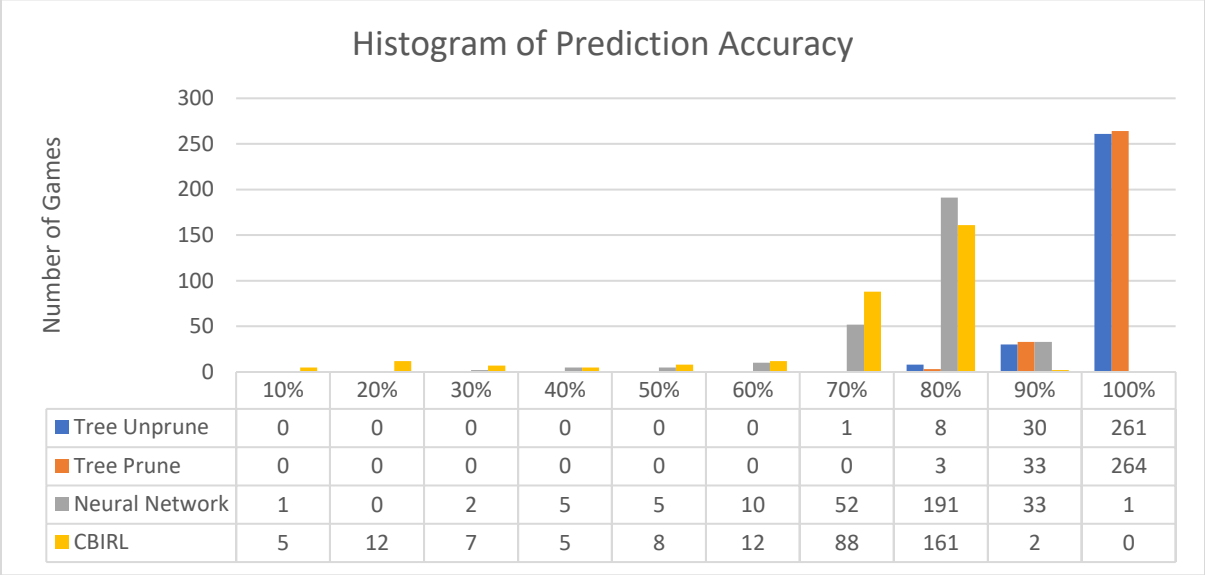
Table 13 Behavior Model Prediction Accuracy

NUMBER OF TRAJECTORIES	CLASSIFICATION TREE – UNPRUNED	CLASSIFICATION TREE – PRUNED	ARTIFICIAL NEURAL NETWORK	CASE-BASED INVERSE REINFORCEMENT LEARNING
2 EXPERT TRAJECTORIES	89.4%	91.0%	70.0%	67.9%
4 EXPERT TRAJECTORIES	96.0%	95.0%	73.1%	64.3%
6 EXPERT TRAJECTORIES	97.1%	96.4%	71.8%	63.4%
8 EXPERT TRAJECTORIES	98.0%	97.6%	72.8%	65.1%
10 EXPERT TRAJECTORIES	97.7%	98.5%	72.0%	65.1%
AVERAGE ACCURACY	**95.6%**	**95.7%**	**72.0%**	**65.2%**

Using single factor ANOVA, the authors find that increasing the number of trajectories only affected the tree classification algorithms' performance. The Neural Network model and the CBIRL did not change the prediction accuracy of any algorithm. The average game last approximately one minute and the behavior models select an action 12 times a second. Therefore, the maximum possible number of states seen after 2 trajectories is 1,440, and for 10 trajectories it is 14,400. The ANOVA findings indicate that there was no significant information gain by Neural Networks or CBIRL from seeing an additional 12,960 state-action pairs (note some of these state-action pairs are duplicates). The Single Factor ANOVA information for all algorithms can be seen in Appendix A.

A one tail Z-test finds that all the machine learning algorithms outperformed the naïve random strategy and received a P-Value of 0 from the one tail Z-test. The test used $H_o = 0.5$ and $H_a < 0.5$.

The authors ran a t-Test and found that the artificial Neural Network has a higher average prediction accuracy than CBIRL. The results of the test can be seen in Appendix B. However, as previously discussed, average prediction accuracy insufficiently measures the quality of the recovered behavior model or algorithm. Figure 56 shows a histogram of prediction accuracy. The x-axis value is the max value of a bin, so 10% includes [0%, 10%] and 100% encompasses (90%, 100%].



	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Tree Unprune	0	0	0	0	0	0	1	8	30	261
Tree Prune	0	0	0	0	0	0	0	3	33	264
Neural Network	1	0	2	5	5	10	52	191	33	1
CBIRL	5	12	7	5	8	12	88	161	2	0

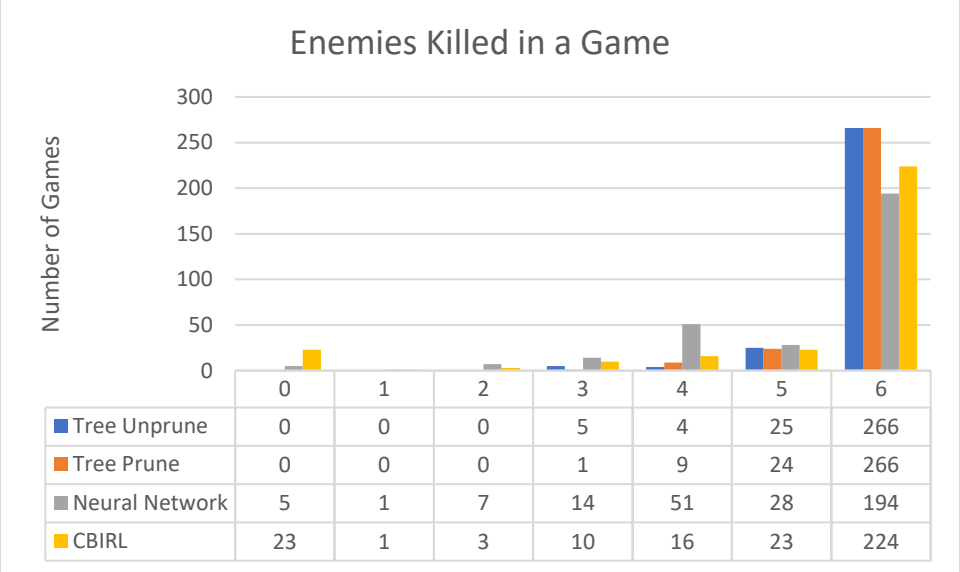
Figure 55 Histogram of Prediction Accuracy

Figure 56 highlights the differences in algorithm convergence. The classification tree algorithms outperformed the Neural Network model consistently converging to a strategy which more closely mimics the reinforcement agent and contradicts Benjamin Geisler [158] findings. Both versions of the classification tree algorithms recovered a behavior model that agreed with the reinforcement agent 100% of the time. Neither the CBIRL or artificial Neural Network algorithm recovered a behavior model that had a perfect agreement with the reinforcement agent.

Case-based Inverse Reinforcement Learning converged to a strategy that over retreats about 12%, significantly hurting the overall prediction accuracy of CBIRL. Further analysis of the data revealed that miss convergence occurs when the recovered reward function heavily values the Vulture unit's health,

leading to the optimal policy of retreating. Unit health was only positively reward feature in 70% of the observed miss convergences and the most heavily weighted in 94% of instances. By retreating frequently, the CBIRL behavior model simultaneously limits the damage taken and extends the game, maximizing the reward. The likelihood of converging to a strategy that retreats to often increases as the quality between the best expert trajectory and average of all expert trajectories approaches zero (i.e., the maximum health remaining health of any expert trajectory equals the average of all expert trajectories).

Although the artificial Neural Network has a higher agreement rate than CBIRL, the CBIRL algorithm is more likely to recover a behavior model that wins the game. Using the Z-test for two proportions, the authors find that the artificial Neural Network is statically less likely to win the game compared to CBIRL (P-Value = 0.00391). The authors defined winning the game as killing all six enemy units and all other case were considered a loss. Figure 53 illustrates the distribution of enemies killed by each algorithm.



	0	1	2	3	4	5	6
Tree Unprune	0	0	0	5	4	25	266
Tree Prune	0	0	0	1	9	24	266
Neural Network	5	1	7	14	51	28	194
CBIRL	23	1	3	10	16	23	224

Figure 56 Number of Enemies Killed in a Match

The artificial Neural Network algorithm is more likely than CBIRL to only kill three or four enemies, meaning that artificial Neural Network is more likely to converge to a strategy that is overly aggressive.

Strategies that kill only three or four enemy units do not retreat frequently enough and take unnecessary damage while reloading. This finding further underscores that average prediction accuracy is insufficient in judging the quality of a recovered behavior model or an algorithm. While the artificial Neural Network algorithm recovers behavior models that have a higher agreement with the reinforcement agent, CBIRL's recovered behavior models are more likely to achieve the emergent outcome of the reinforcement agent (i.e., eliminate all six enemy units). Both tree classification algorithms outperformed Neural Networks and CBIRL in prediction accuracy and achieving the most games won.

Applying Recovered Behavior Models to Unseen Environments

The goal of this research is to recovered behavior models from an expert's trajectory through a state space and used these behavior models to predict the expert's action in unseen environments. Benjamin Geisler [158] examined how recovered behavior models predicted an expert's action in unseen states. However, this research differs because the author changes environmental parameters, which change the transition matrix. The goal is to see if the recovered behavior models are resilient enough to be used in an agent-based model, where environmental factors will be altered to examine how the agent reacts and effects other agents and the environment.

The authors created a second scenario where the agent controlling the Vulture unit fights nine units, known as Zerglings. Figure 54 shows the scenario. The Zergling unit is weaker than the Marine unit and is a melee unit, meaning that it must be next to the enemy unit to deal damage. Altering the enemy unit changes the transition matrix because it increases the total number of enemy units, requires those units to be closer to the Vulture, alters the rate damage dealt to the Vulture, and changes the health distribution of the enemy units. Except for changing the enemy unit type, no other changes were made to the scenario.

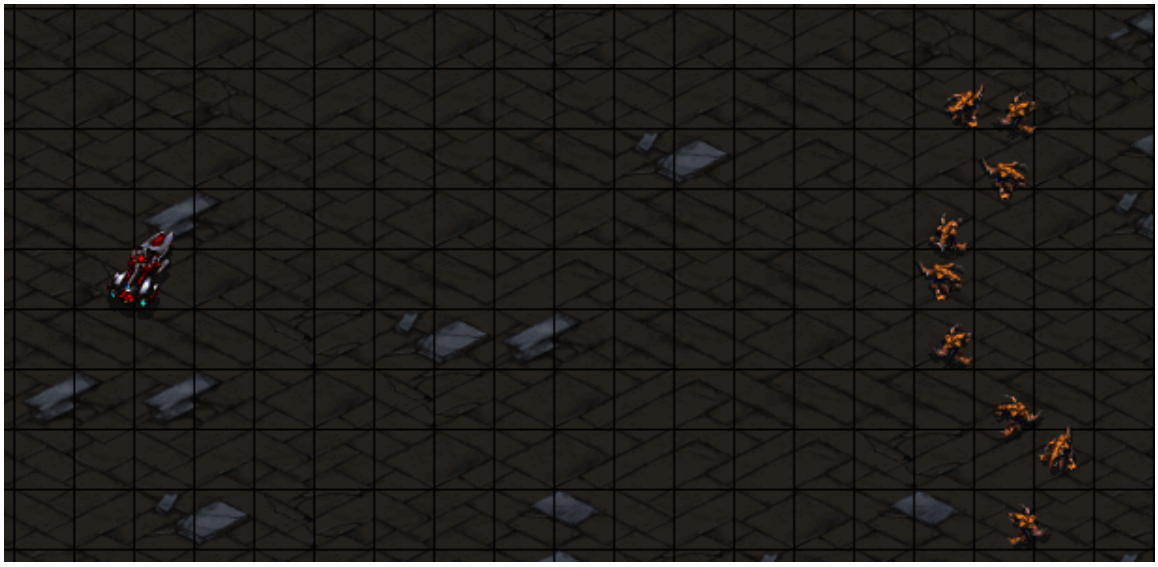


Figure 57: Vulture vs. Zerglings

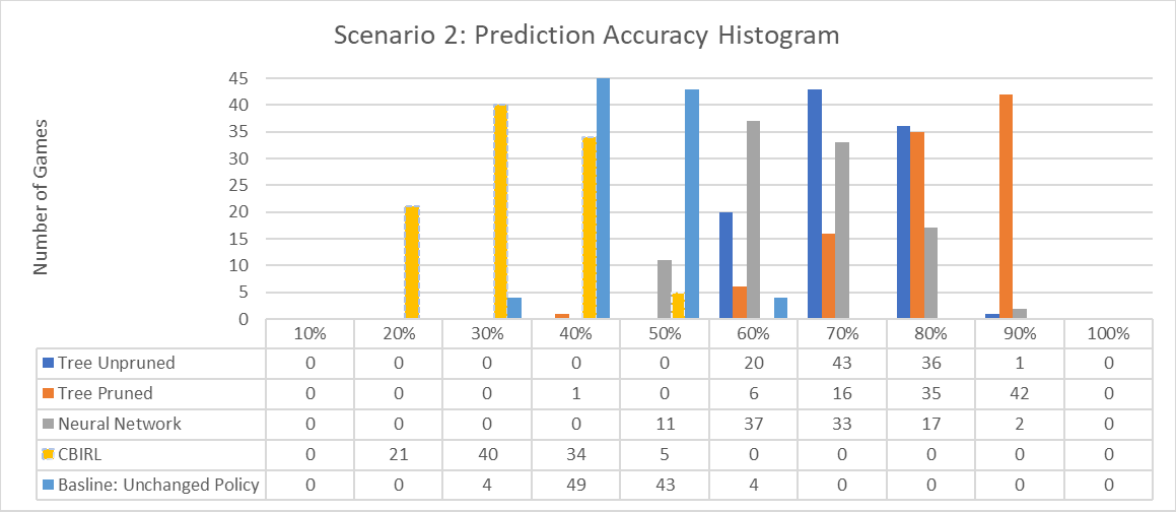
The reinforcement agent was retrained on the new scenario, using the reward function and methodology described in the previous section. The new expert policy never won a match and had an average kill rate of 5.6 units per game. This finding indicates that expert state space is not robust enough to allow the reinforcement agent to develop a winning strategy. This is likely caused by the state space feature representing distance, which is bucketed into four categories. Since the Zergling unit must be next to the Vulture to deal damage, all enemy units regularly enter the closest distance category. A more granular state space is needed to develop winning strategy.

The recovered behavior model, for each machine learning algorithm, that achieved the most kills while taking the least amount of damaged in the previous experiment were compared using the new scenario. These behavior models were chosen because they best achieved the expert's objective. All behavior models used the previously described state space.

While the Neural Network and classification tree behavior models can be directly applied to the new scenario the Case-Based Inverse Reinforcement Learning behavior model has to be tuned. Unlike the other machine learning algorithms CBIRL does not return a model capable of making decisions in a given state space, instead it returns a reward function which the expert is assumed to be optimizing. The authors

used 100 random trajectories through the state space to build CBIRL case-based library of transition matrix, which was then used by CBIRL to estimate the expected reward of an action in a given state. CBIRL implemented the greedy policy described in the previous experiment.

The unchanged expert policy learned in the previous experiment is applied to the new scenario and is used as a baseline, which is analogous to an expert's strategy being unaffected by the environmental change. As in the previous scenario we look at both the prediction accuracy (i.e., predicting the trained reinforcement agent's action in a given state) and the lethality of the behavior models. Figure 59 shows the behavior models' distribution of accuracy over 100 trials.



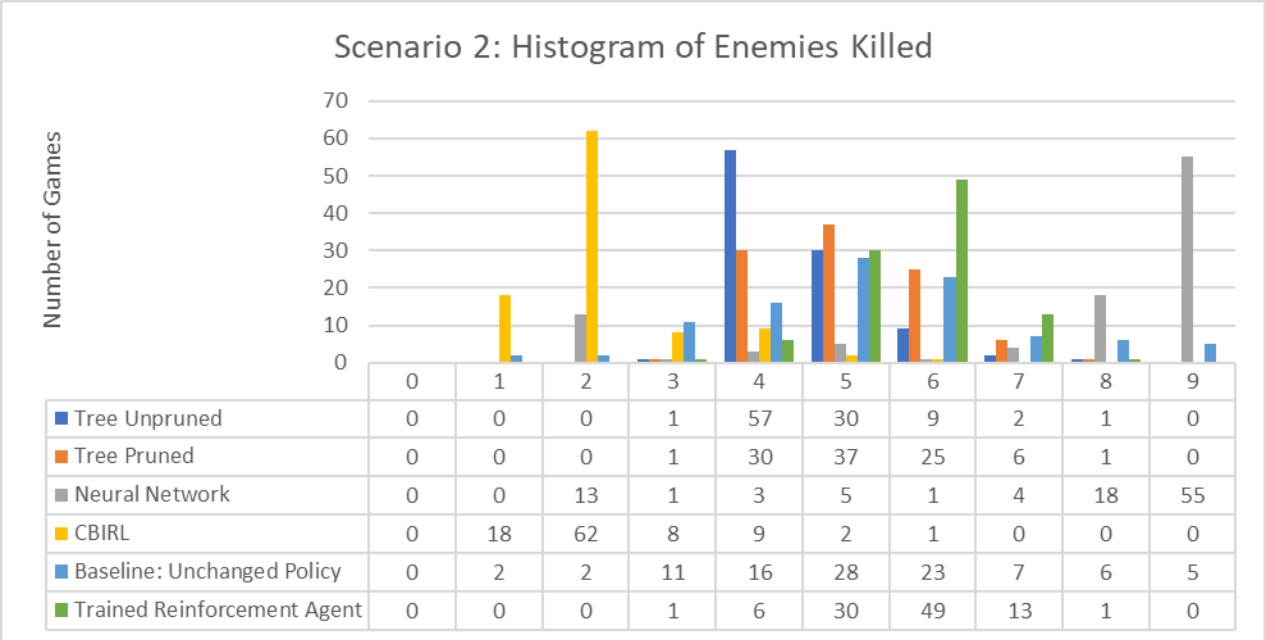
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Tree Unpruned	0	0	0	0	0	20	43	36	1	0
Tree Pruned	0	0	0	1	0	6	16	35	42	0
Neural Network	0	0	0	0	11	37	33	17	2	0
CBIRL	0	21	40	34	5	0	0	0	0	0
Basline: Unchanged Policy	0	0	4	49	43	4	0	0	0	0

Figure 58: Scenario 2: Prediction Accuracy Histogram

Using a T-test and assuming unequal variance the authors find that the Pruned Tree behavior model has the highest average accuracy (see appendix B). Figure 8 illustrates the previously mentioned finding and shows that both classification tree behavior models outperformed the other models. CBIRL and the unchanged expert policy from the previous experiment both fail to predict the reinforcement agent's behavior more than 50% of the time, showing the reinforcement agent changed its behavior to overcome the new enemy. It also demonstrated that the policy learned by CBIRL is less robust to

environmental changes than both Neural Networks and classification trees, which will be discussed further in the conclusion section.

As demonstrated by previous findings, prediction accuracy is not sufficient to judge the quality of a recovered behavior model. Figure 60 shows the histogram of units killed during a game. The Neural Network behavior model out performs all other models by a significant margin. While the Neural Network model was less accurate than the classification tree algorithms in predicting how the reinforcement agent would evolve in the new environment, it is more robust than other models, allowing it to win 55% of the time in a previously unseen environment. The Neural Network behavior model eliminated more enemies than the trained reinforcement agent, further highlighting the inadequacy of the reinforcement agent's state-space (See Appendix C).



	0	1	2	3	4	5	6	7	8	9
Tree Unpruned	0	0	0	1	57	30	9	2	1	0
Tree Pruned	0	0	0	1	30	37	25	6	1	0
Neural Network	0	0	13	1	3	5	1	4	18	55
CBIRL	0	18	62	8	9	2	1	0	0	0
Baseline: Unchanged Policy	0	2	2	11	16	28	23	7	6	5
Trained Reinforcement Agent	0	0	0	1	6	30	49	13	1	0

Figure 59: Scenario 2 Histogram of Enemies Killed

Using the Mood's Median test, the authors find that there is no statistical difference in the lethality of the pruned tree behavior model and that of the baseline. Additionally, the authors find that the unpruned tree behavior model kills less enemies than that of the pruned tree model (see Appendix

C). Pruned decision trees are designed to be more general, allowing their performance to be more resilient to environmental changes, which is illustrated by the authors findings. The CBIRL behavior model consistently killed the least number of enemy units. The next section will discuss the findings of the two experiments.

Conclusion

The authors examined how well classification trees, Neural Network, and Case-Based Inverse Reinforcement Learning recovered an expert's policy in a partially observed Markov environment. Using reinforcement learning an automated agent was trained on a combat scenario and used to generate expert trajectories. The previously mentioned machine learning algorithms recovered behavior models from a set of 2, 4, 6, and 8 expert trajectories and were compared on how well their actions matched the expert and how many enemy units they killed.

The reinforcement agent was then trained on a new combat scenario, where it encountered a previously unseen unit type. Although the reinforcement agent improved through training it was never able to win the second combat scenario, which indicates the state space did not provide enough information to the agent. For each machine learning algorithm, the recovered behavior model that killed the most enemy units, while taking the least damage was applied to the new scenario.

The authors find the research objective should dictate the machine learning strategy selected. The classification tree behavior out performed both CIBIR and the Neural Network algorithm when applied to the environment it was trained in, contradicting previously published research. However, when examining unseen environments, previous agent-based model research use machine learning methods to adapt agents. The authors' findings demonstrate that an expert may not converge to the optimal policy, as seen by the reinforcements agent failure to win the second scenario. In the second experiment the Neural Network behavior model outperformed the trained reinforcement agent (i.e., killed more enemy units) but the classification tree behavior models more accurately reflected the trained

reinforcement agent. In this case the classification tree behavior model should be used in the agent-based model because it better reflects the real-world agent, allowing the agent-based simulation to more accurately predict how an environmental change will affect the real-world actor.

Case-Based Inverse Reinforcement Learning behavior models were more lethal than Neural Network when applied to the environment they were trained in. However, CBIRL rarely recovers the expert's actual reward function, instead it finds a reward function that maximizes the total difference between the expert's policy and that of the observed trajectory. The recovered reward function that generated the most kills, while taking the least damage in experiment one is R = {$W_{Distance}$ = 0, $W_{\#EnemiesInRange}$ = 0.91, $W_{UnitHealth}$ = 0, $W_{EnemyHealth}$ = -0.45, $W_{CanFire}$ = 0.53}. The recovered reward function does not match the reinforcement agent's goal of maximize the difference between its health and the enemy units. However, while attempting to achieve its goal the reinforcement agent learns to use a kiting or 'hit and run' strategy.

In a kiting strategy the agent attacks a unit and then retreats to safety while it reloads. The reward function above causes this same behavior. The agent is rewarded for having more enemies within its weapons range and for decreasing the enemy units' health (i.e., attacking). The agent is also rewarded for having a loaded weapon. Since the agent cannot hurt the enemy units with an unloaded weapon and the retreat action more often proceeds a reload than an attack action in the expert's trajectory, the CBIRL agent retreats until it can reload.

The recovered reward function is highly tuned to its training environment, which allows it to better match the expert's emergent outcome (i.e., kill the six enemy units). However, when the linear reward function is applied to a feature space with bounds outside its training set, the model decreases in prediction accuracy. This is analogous to the step-wise linear regression model in the previous chapter, which failed to predict outcomes outside its training set bounds.

Future work will examine non-linear Inverse Reinforcement algorithms, which may be more resilient to feature spaces outside the bounds of the training set. Additionally, the authors will examine how ensemble model, combing multiple machine learning algorithms, performs on partially observed Markov state spaces. Lastly, the authors will increase both the state-space and the possible actions by increasing the number of allied units, recovering target priority, and recovering unit positioning. Target priority determines the order in which enemy units are attacked. Unit positioning determines the spacing of allied units during combat and retreating. Similar to real-world combat there are times that allied units should be close together (e.g., providing antiair cover) and there are other times when allied units should be spread out (e.g., under attack by mortars). The following chapter is the discussion section of the dissertation.

Chapter 6: Discussion

Agent-based policy models and their limitations

An agent-based simulation is a powerful tool for complexity scientist seeking to understand complex systems better [8], [9]. Many humanmade systems meet the definition of complex systems, which is a system that has individual elements that interact with one another in a non-linear manner. Developing policies for complex systems is challenging because it is impossible to predict how any change to the system will affect individual agents and the system as a whole.

We demonstrated that agent-based models are an effective tool for examining how potential policies will affect a complex system. Agent-based models can explore how potential regulations would affect emergent properties, allowing regulators to designed more targeted regulations. Chapter two illustrates agent-based modeling applicability by examining regulations in two separate domains. However, decision-makers find the validity of agent-based models questionable [14], [15]. Most agent-based models are not quantifiably validated [12]. Furthermore, models that are validated only examine the aggregate emergent outcomes, like the financial and active shooter models presented in chapter two. Examining emergent outcomes without empirically deriving agent classes or validating the behavior model of the agents makes any findings disputable [22]. Models that do not validate the behavior model of the agent or empirically derive agent classes are limited to stating that the model is one of potentially many other that lead to observed emergent outcomes.

A further limitation of not validating the behavior model is that an agent-based model is limited in its ability to predict how a change in an agent's behavior or system variable will affect other agents and the aggregate emergent properties. Agent-based practitioners use machine learning algorithms to mitigate this limitation [173], [174]. However, the objective function the machine learning algorithms are attempting to optimize is not proven to be the objectives of the agent's real-world counterparts. An

incorrect objective function can lead to behavior models that do not represent the real-world actor and inaccurate simulation results. The author asserts that the behavior models of agents must be validated against their real-world counterparts. Recovering the behavior model of a real-world actor through apprenticeship learning both develops an agent's behavior model and validates it against its real-world counterpart.

Challenges exist when recovering a behavior model. The simplest method is to group similar agents and define a strategy that broadly represents that group, which was the technique used by the author in the financial agent-based model presented previously. However, similar to the author work, the majority of agent-based models that group agents into classes use a supervised method, where the researcher defines the number of classes. Bias can be introduced into the study because the research has preconceived notions about the number of classes and the behavior model of each class. The supervised framework leads to agent-based models that are made up of agents whose behavior model is perceived to be known or easily described and not necessarily reflective of real-world actors.

Unsupervised Clustering of a Dataset with Outliers

The author proposes that unsupervised clustering can be used to recover cluster of behavior models, mitigating the previously mentioned bias. However, due to human irrationality outliers will inevitably exist in a dataset made up of behavior model actions. Outliers can cause otherwise distinct behavior models to be grouped. Traditional techniques to perform unsupervised clustering with outliers present, rely on the expert to specify parameters, which the clustering algorithm is extremely sensitive to, or rely on preprocessing the dataset, which can become computationally expensive as the dataset grows.

In this dissertation, we introduce Size-First Hierarchical clustering, which modifies the traditional hierarchical clustering to require that the smallest cluster is grouped with its closest neighbor. The author hypothesized that Size-First Hierarchical clustering would outperform existing clustering techniques for

datasets that are isotropic. Additionally, Size-First Hierarchical clustering does not require any preprocessing and limits expert input to defining the minimum cluster size. By applying the silhouette stopping condition, Size-First Hierarchical clustering locates natural clusters in the dataset that are larger than the minimum cluster size.

The results presented in this dissertation, using a simple 2-dimensional dataset, illustrate that Almeida's method is unable to cluster non-compact nuclei, integer clusters correctly. The author also finds that DB-SCAN is highly sensitive to the expert defined parameters. Size-First Hierarchical clustering was shown to be more stable than DB-SCAN and capable of clustering non-compact integer clusters.

Using StarCraft: Brood War, a popular real-time strategy game, we compared the previously mentioned clustering techniques in a more complex environment, with over 300 dimensions, making it impossible for a person to cluster by hand. Based on interviews with professional players and previously published research, the author expected approximately six strategy clusters to be recovered by the 11^{th} minute. Both Almeida's method and DB-SCAN return one or two clusters at the 11^{th} minute, indicating that they were unable to discern the difference between clusters.

Like the 2-dimensional dataset, the StarCraft clusters have loosely packed nuclei and are close together, which is illustrated in the principal component analysis. The findings suggest that both popular clustering techniques struggle to discern natural clusters in integer datasets, where the clusters do not have a tightly compact nucleus and close to each other.

Size-First Hierarchical clustering was found to outperform both methods and discover clusters that were strategically valuable. For integer datasets that are isotropic, Size-First Hierarchical clustering is a viable method that outperforms existing outlier clustering techniques and returns a natural cluster set with limited expert input, mitigating the potential expert bias.

The isotropic limitation of Size-First Hierarchical clustering comes from its arithmetic mean distance measure and silhouette stopping condition, which compares intracluster distance to intercluster distance. The author selected these methods because of a hypothesis that behavior model datasets are often isotropic. The arithmetic mean distance measure results in tighter clusters than the single link distance measure. The donut shape cluster with a second spherical cluster inside, shown in Figure 31, cannot be recovered using the arithmetic mean distance but it can be recovered using the single link distance measure. Additional research into a suitable stopping condition is required to expand Size-First Hierarchical clustering to non- isotropic datasets.

Using Decision Trees to Recover Behavior Models

Reducing the potential for bias by implementing unsupervised clustering improves both the validity and acceptability of agent-based models. However, alone this is insufficient to prompt decision-makers to accept agent-based models as a policy tool. Agent-based models must ensure that the artificial agents are behaving in a manner that is comparable to their real-world counterparts.

In the financial agent-based model that the author presents in Chapter 2, the agents were compared to their real-world counterparts using aggregate measures, such as their end of day positions. We did not validate that the agents' trade decisions mimicked real-world actors. In fact, since the model was zero-intelligence, we know that the agents did not behave in a manner that real-world actors would behave in. Therefore, it is difficult to assert that our findings are reliable.

Firstly, we may miss characterize how a policy change will affect a real-world actor implementing a strategy not accurately reflected by our model. Additionally, as agents change their strategy in response to policy changes our model is incapable of predicting how these strategies will affect the market. To mitigate this limitation many researchers, implement machine learning. However, as the researcher selects the objective function, which is rarely validated, this introduces bias into the model.

For example, many researchers assume the agents are attempting to maximize profit. While some real-world actors are attempting to maximize profit, many real-world traders have a different objective, such as hedging. Traders that implement a hedging strategy are limiting their losses. Recovering hedging behavior models is an ideal testbed because while the objective if known there are a plethora of strategies used to achieve it.

We implemented Classification and Regression Trees, also known as recursive partitioning, on teams in the University of Virginia McIntire Hedge Fund tournament. The objective was to recover the winning team's strategy. The author found that providing the classification tree algorithm with only a description of the state-space was insufficient for recovering the strategy. Instead, the author needed to provide action-features, as well as state-space features to recover an appropriate strategy. Action-features, such as the option's expiration date, allowed the classification tree algorithm to discern not only when a security will be traded but a preference between securities.

Illustrating that action-features improve the accuracy of the recovered behavior model is a significant finding. In complex behaviors such as hedging, real-world actors consider not just the current state of the environment but also the differences in potential action effects. For example, an option that expires in May can be more expensive but provides a more stable hedge against price volatility than options that expire earlier. Trading off cost versus hedging stability is inherent in a hedging strategy and not intuitively captured in a state-space definition. The author asserts that for strategies that operate in an environment with more than a binary choice and with actions that have inherent characteristics, it becomes necessary to provide both a state-space and action-features to a classification tree model.

To determine the quantity of a security to purchase or sell we applied regression trees. During the study, the authors integrated step-wise linear regression with regression trees. Instead of the terminal node being the average quantity, step-wise linear regression was run on each terminal node's dataset.

The authors found that step-wise linear regression has the potential to outperform traditional regression trees. However, during the testing data set the step-wise linear regression tree was shown to be volatile, rapidly diverging from the observed strategy and resulting in worse performance than traditional regression trees. This divergence was caused by an input value that was outside the training set's bounds.

Future research will look into a hybrid model where step-wise linear regression tree models are applied to environments that stay within the maximum and minimum bounds of the training dataset. Any input data falling outside these bounds should rely on the standard regression tree method to ensure stability.

Comparing Machine Learning Methods in a Partially Observed Markov Environments

While the previous findings demonstrate that classification trees are capable of recovering behavior models, real-world scenarios that are modeled using agent-based models have significantly larger states-spaces. Additionally, the real-world actors' behavior models are expressed by a small number of trajectories through the state-space. Combined these two features of real-world problems could potentially lead to classification trees to overfit the training set.

Methods for recovering behavior models in the previously defined environment are of interest to the U.S. Military, which would like to recover behavior models of trainees in simulated environments. The goal is to develop realistic enemies by replicating observed behavior. Additionally, an automated opponent can be trained against the recovered trainee's behavior model, allowing the training simulation to adapt to the trainee and thus making a more engaging training exercise.

The authors compared Classification Trees to Backpropagation Neural Networks and Case-Based Inverse Reinforcement Learning (CBIRL). Once again using StarCraft: Brood War, the author, used reinforcement learning to develop a behavior model that allowed a single unit to overcome a larger force.

The trained reinforcement agent acted as an expert, and the three machine learning algorithms were used to recover reinforcement agent's behavior model.

The reinforcement agent's state space was significantly larger than McIntire Hedge Fund Tournament, but the action space was limited to two actions, attack or retreat. The author's finding suggests that prediction accuracy alone is insufficient to judge how we a machine learning algorithm recovered a behavior model. Both the Neural Network model and Case-Based Inverse Reinforcement Learning failed to predict the expert's action more often than the naïve strategy of 'always attack'. However, both machine learning models recovered strategies that won the game, which the naïve strategy of 'always attack' fails to do.

The sequential nature of the game leads to the 'always attack' strategy having a high prediction accuracy because the trained reinforcement agent will always attack if an enemy is in range and its weapon is loaded. Additionally, the reinforcement agent will implement an 'always attack' strategy once its health reaches a critical level, analogous to a last stand. In StarCraft reloading is relatively fast and the agent quickly loses life when it doesn't retreat, thus an always attack strategy will often coincide with the trained agent's decision.

The trained reinforcement agent learns a kiting strategy, over repeated trials that allow it to win 90% of its games. In a kiting strategy, the agent retreats until it can finish reloading, allowing it to minimize the damage it takes. All three machine learning techniques identify the kiting strategy. While Neural Networks outperform CBIRL in prediction accuracy, CBIRL recovers behavior models that better implement the trained agent's kiting strategy. In other words, Neural Networks better replicate the agent's behavior, while CBIRL recovers a reward function that better expresses the goal the agent is attempting to accomplish (e.g., kill all six enemy units using the kiting strategy).

Classification trees were found to outperform both CBIRL and Neural Networks, which contradicts the findings reported by Benjamin Geisler. Pruned classification trees were able to achieve 90% prediction accuracy after observing the reinforcement agent twice, which indicates that classification trees need significantly fewer observations than CBIRL and Neural Network algorithms to converge to a recovered behavior model with high prediction accuracy. Additionally, the output of classification trees is a human-interpretable decision tree, allowing researchers to better understand how the recovered behavior model determines its decisions, improving researcher's ability to justify their agent's behaviors.

We selected the recovered behavior model of each machine learning method, which killed the most unit while taking the least damage and applied them to a new environment, where the enemy is an unseen unit type. The goal is to determine which recovered behavior model best represented how the agent would evolve to overcome its opponent. To have a baseline we kept the Q-matrix constant for the trained reinforcement agent, which is analogous to saying the agent does not change its behavior when combating a new enemy. A second reinforcement agent, using the same state-space as the first, is trained to combat the new unit type. We then compared how accurately recovered behavior models and the baseline agent predict the newly trained agent's behavior.

The newly trained agent was unable to defeat the new enemy unit type, indicating that the state-space was not sufficiently detailed enough. The original enemy unit (Marines) are ranged units, which stopped approaching at a specific distance. The new enemy unit (Zergling) are melee units, which means they must be next to the opponent to attack. The difference in combat style is not sufficiently captured in the distance state-space feature because the distance feature is bucketed into groups of four. Furthermore, there were more enemy units than in the original scenario. The author believed the baseline agent thought it was surrounded because a large number of enemy units were close to it, which lead the baseline agent to stand and fight rather than retreat.

It is important to highlight the reinforcement agent converged to a suboptimal strategy in the second experiment (i.e., did not win). Many agent-based practitioners implement learning mechanism without validating the objective function or the state-space the real-world actor is using. If such a framework was applied in this example, it would result in an agent using an optimal strategy but not representing the observed expert. Therefore, it is essential to validate the agent's behavior model is accurately representing its real-world counterpart.

CBIRL performed worse than the baseline in both prediction accuracy and the number of enemies killed. Like in chapter 4, CBIRL uses a linear function that is tuned to the observed state space. As previously mentioned there were more enemy units than in the training environment, and the enemy units were closer than the training environments. Since new input data was outside the bounds of the training set, CBIRL acted irradicably and was unable to predict the newly trained agent's actions. In future work, the author will apply a non-linear reward function in an attempt to increase the stability.

The classification tree behavior models were found to predict the newly trained agent the best and were significantly better than the baseline agent. The classification tree models were able to identify the distance cutoff in the first scenario where they would retreat. They applied this cutoff when facing the new enemy and closely mimicked the newly trained reinforcement agent.

In the second experiment the Neural Network recovered behavior model was found to consistently kill more enemy units than any other behavior model, including the trained reinforcement agent. Like CBIRL in the first scenario, the Neural Network recovered behavior model is more robust in its ability to achieve the expert's goal (i.e., eliminating all enemy units) but does not reflect how the expert will behave in new unseen scenarios. Future research can explore if non-linear inverse reinforcement learning has better performance.

Often agent-based practitioners apply learning mechanisms to predict how agents will evolve in unseen environments. However, the findings illustrate that experts can converge to suboptimal solutions, which will not be represented in agent-based simulations where all agents are applying optimal policies. Our research finds that classification trees are an effective method for recovering behavior models from a small number of observed trajectories. Trained classification tree models were found to accurately predict the behavior of an agent in an unseen environment. Agent-based researchers should apply classification trees to recover the behavior model of real-world actors they intend to simulate, which will increase the validity and acceptability of their model.

Bibliography

[1] S. H. Fritts *et al.*, "Planning and implementing a reintroduction of wolves to Yellowstone National Park and central Idaho," *Restor. Ecol.*, vol. 5, no. 1, pp. 7–27, 1997.

[2] W. J. Ripple and R. L. Beschta, "Wolf reintroduction, predation risk, and cottonwood recovery in Yellowstone National Park," *For. Ecol. Manage.*, vol. 184, no. 1–3, pp. 299–313, 2003.

[3] R. L. Beschta and W. J. Ripple, "River channel dynamics following extirpation of wolves in northwestern Yellowstone National Park, USA," *Earth Surf. Process. Landforms*, vol. 31, no. 12, pp. 1525–1539, 2006.

[4] U. Wilensky and W. Rand, *An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo*. MIT Press, 2015.

[5] E. Gertz, "Has The Reintroduction Of Wolves Really Saved Yellowstone?" Mar-2014.

[6] E. Marris, "Legend of the wolf," *Nature*, vol. 507, no. 7491, p. 158, 2014.

[7] M. M. Pollock, G. Lewallen, K. Woodruff, C. E. Jordan, and J. M. Castro, "The beaver restoration guidebook: Working with beaver to restore streams, wetlands, and floodplains," *United States Fish Wildl. Serv.*, 2015.

[8] S. Wolfram, *A new kind of science*, vol. 5. Wolfram media Champaign, 2002.

[9] L. Henrickson and B. McKelvey, "Foundations of 'new' social science: Institutional legitimacy from philosophy, complexity science, postmodernism, and agent-based modeling," *Proc. Natl. Acad. Sci.*, vol. 99, no. suppl 3, pp. 7288–7295, 2002.

[10] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM SIGGRAPH computer graphics*, 1987, vol. 21, no. 4, pp. 25–34.

[11] E. Bonabeau, “Agent-based modeling: Methods and techniques for simulating human systems,” *Proc. Natl. Acad. Sci.*, vol. 99, no. suppl 3, pp. 7280–7287, 2002.

[12] B. Heath, R. Hill, and F. Ciarallo, “A survey of agent-based modeling practices (January 1998 to July 2008),” *J. Artif. Soc. Soc. Simul.*, vol. 12, no. 4, p. 9, 2009.

[13] N. Gilbert, “AGENT-BASED MODELS,” *Sage*, vol. 153, 2008.

[14] M. Richiardi, R. Leombruni, N. Saam, and M. Sonnessa, “A common protocol for agent-based social simulation,” *J. Artif. Soc. Soc. Simul.*, vol. 9, no. 1, 2006.

[15] W. Rand and R. T. Rust, “Agent-based modeling in marketing: Guidelines for rigor,” *Int. J. Res. Mark.*, vol. 28, no. 3, pp. 181–193, 2011.

[16] P. K. Davis and D. Blumenthal, “The base of sand problem: A white paper on the state of military combat modeling,” 1991.

[17] S. Schlesinger, “Terminology for model credibility,” *Simulation*, vol. 32, no. 3, pp. 103–104, 1979.

[18] R. G. Sargent, “Validation and verification of simulation models,” in *Proceedings of the 36th conference on Winter simulation*, 2004, pp. 17–28.

[19] O. Balci, “How to assess the acceptability and credibility of simulation results,” in *Proceedings of the 21st conference on Winter simulation*, 1989, pp. 62–71.

[20] O. Balci and R. G. Sargent, “Some examples of simulation model validation using hypothesis testing,” in *Proceedings of the 14th conference on Winter Simulation-Volume 2*, 1982, pp. 621–629.

[21] T. H. Naylor and J. M. Finger, “Verification of computer simulation models,” *Manage. Sci.*, vol. 14, no. 2, p. B--92, 1967.

[22] S. Y. Harmon and S. M. Youngblood, “A proposed model for simulation validation process maturity,” *J. Def. Model. Simul.*, vol. 2, no. 4, pp. 179–190, 2005.

[23] M. Steinbach, L. Ertöz, and V. Kumar, “The challenges of clustering high dimensional data,” in *New directions in statistical physics*, Springer, 2004, pp. 273–309.

[24] J. Fox and A. Sklar, *The myth of the rational market: A history of risk, reward, and delusion on Wall Street*. Harper Business New York, 2009.

[25] A. P. Chaboud, B. Chiquoine, E. Hjalmarsson, and C. Vega, “Rise of the machines: Algorithmic trading in the foreign exchange market,” *J. Finance*, vol. 69, no. 5, pp. 2045–2084, 2014.

[26] M. Kearns, A. Kulesza, and Y. Nevmyvaka, “Empirical limitations on high frequency trading profitability,” *arXiv Prepr. arXiv1007.2593*, 2010.

[27] A. Kirilenko, A. S. Kyle, M. Samadi, and T. Tuzun, “The flash crash: The impact of high frequency trading on an electronic market,” *Available SSRN*, vol. 1686004, 2011.

[28] U. S. Securities, E. Commission, C. F. T. Commission, and others, “Findings regarding the market events of May 6, 2010,” *Washingt. DC*, 2010.

[29] S. Maslov, “Simple model of a limit order-driven market,” *Phys. A Stat. Mech. its Appl.*, vol. 278, no. 3–4, pp. 571–578, 2000.

[30] D. Challet and R. Stinchcombe, “Analyzing and modeling 1+ 1d markets,” *Phys. A Stat. Mech. its Appl.*, vol. 300, no. 1–2, pp. 285–299, 2001.

[31] N. Kaldor, “Capital accumulation and economic growth,” in *The theory of capital*, Springer, 1961, pp. 177–222.

[32] J. P. Bouchaud, M. Mezard, and M. Potters, “Statistical properties of the stock order books:

empirical results and models, Mathematics ArXiv." 2002.

[33] S. Mike and J. D. Farmer, "An empirical behavioral model of liquidity and volatility," *J. Econ. Dyn. Control*, vol. 32, no. 1, pp. 200–234, 2008.

[34] B. B. Mandelbrot, "The variation of certain speculative prices," in *Fractals and scaling in finance*, Springer, 1997, pp. 371–418.

[35] A. Kirman and G. Teyssiere, "Microeconomic models for long memory in the volatility of financial time series," *Stud. Nonlinear Dyn. Econom.*, vol. 5, no. 4, 2002.

[36] L. Kullmann, J. Töyli, J. Kertesz, A. Kanto, and K. Kaski, "Characteristic times in stock market indices," *Phys. A Stat. Mech. its Appl.*, vol. 269, no. 1, pp. 98–110, 1999.

[37] B. Biais, P. Woolley, and others, "High frequency trading," *Manuscript, Toulouse Univ. IDEI*, 2011.

[38] M. Paddrik, R. Hayes, A. Todd, S. Yang, P. Beling, and W. Scherer, "An agent based model of the E-Mini S&P 500 applied to Flash Crash analysis," in *Computational Intelligence for Financial Engineering & Economics (CIFEr), 2012 IEEE Conference on*, 2012, pp. 1–8.

[39] D.-F. W. S. Reform and C. P. Act, "Public Law 111-203," *US Statut. Large*, vol. 124, p. 1376, 2010.

[40] "Federal Administrative Procedure Act: United States Code Title 5 Chapter 2." .

[41] "Commodity Exchange Act: United States Code Title 7 Chapter 1." .

[42] I. S. and D. Association, "Case 1:11-cv-02146-RLW." The United States District Court for the District of Columbia, 2011.

[43] J. Holzer, "Court Deals Blow to SEC, Activists," *Wall Street Journal*, 2011.

[44] F. Zhang, "High-frequency trading, stock volatility, and price discovery," 2010.

[45] Securities, E. Commission, and others, “Concept release on equity market structure,” *Fed. Regist.*, vol. 75, no. 13, pp. 3594–3614, 2010.

[46] C. Deissenberg, S. Van Der Hoog, and H. Dawid, “EURACE: A massively parallel agent-based model of the European economy,” *Appl. Math. Comput.*, vol. 204, no. 2, pp. 541–552, 2008.

[47] F. H. Westerhoff, “The use of agent-based financial market models to test the effectiveness of regulatory policies,” *Jahrb. Natl. Okon. Stat.*, vol. 228, no. 2–3, pp. 195–227, 2008.

[48] D. A. Dubofsky and J. C. Groth, “Exchange listing and stock liquidity,” *J. Financ. Res.*, vol. 7, no. 4, pp. 291–302, 1984.

[49] J. Brogaard and others, “High frequency trading and its impact on market quality,” *Northwest. Univ. Kellogg Sch. Manag. Work. Pap.*, vol. 66, 2010.

[50] M. Follman, G. Aronsen, and D. Pan, “Mass Shooting Map,” *Mother Jones*, 2013.

[51] T. Telegraph, “History of mass shootings in the US since Columbine,” *The Telegraph*, 2012.

[52] E. Bagalman, S. W. Caldwell, K. M. Finklea, and G. McCallion, “Public mass shootings in the United States: Selected implications for federal public health and safety policy,” *Congr. Res. Serv.*, 2013.

[53] D. Feinstein, “S.150 - Assault Weapons Ban of 2013.” United States Senate, 2013.

[54] T. Erlenbruch, “Agent-based simulation of German peacekeeping operations for units up to platoon level,” 2002.

[55] R. F. A. Woodaman, “Agent-based simulation of military operations other than war small unit combat,” Monterey, California. Naval Postgraduate School, 2000.

[56] C. S. Choo, C. L. Chua, and S.-H. V. Tay, “Automated red teaming: a proposed framework for military application,” in *Proceedings of the 9th annual conference on Genetic and evolutionary*

computation, 2007, pp. 1936–1942.

[57] Y. H. Wong, “Ignoring the Innocent: Non-combatants in Urban Operations and in Military Models and Simulations,” 2006.

[58] R. Hayes and R. Hayes, “Mass Shooting Simulation.” CoMSES Computational Model Library, 2013.

[59] R. Hayes and R. Hayes, “Auroa Shooting Model.” CoMSES Computational Model Library, 2013.

[60] K. Dockery, *Future weapons*. Penguin, 2007.

[61] D. Bertsekas and J. Tsisikil, *Introduction To Probability*, 1st ed. Massachusetts: Atlanta Scientific, 2002.

[62] D. Giancoli, *Physics for Scientists & Engineers*. New Jersey: Prentice Hall, 2000.

[63] CBC News, “Batman premiere gunman looked like ‘assassin ready for war,’” *CBC News*, 20-Jul-2012.

[64] D. Fahrenthold, T. Heath, and J. Achenbach, “Aurora, Colo., shooting spree: A day of tears for victims and of twists in case,” *Washington Post*, 22-Jul-2012.

[65] J. Steffen, K. Lee, R. Parker, J. Bunch, J. Brown, and J. Ingold, “Family identifies 27-year-old victim of Aurora theater shooting,” *Denver Post*, Denver, 20-Jul-2012.

[66] “Aurora Theater Reopening to the Public for First Time Since Shooting.,” ABC 7 Denver, 2012.

[67] City Stars Cinnema, “Analysis of Existing Building Types,” 2008.

[68] Illinois OSFM Division of Technical Services, “CALCULATING OCCUPANT LOAD FOR AN ASSEMBLY OCCUPANCY USING THE 2000 NFPA LIFE SAFETY CODE,” 2010.

[69] R. Hayes and R. Hayes, “Agent-based simulation of mass shootings: Determining how to limit the

scale of a tragedy," *J. Artif. Soc. Soc. Simul.*, vol. 17, no. 2, p. 5, 2014.

[70] *Bump-Fire Stock*. Florida, 2018.

[71] P. Windrum, G. Fagiolo, and A. Moneta, "Empirical validation of agent-based models: Alternatives and prospects," *J. Artif. Soc. Soc. Simul.*, vol. 10, no. 2, p. 8, 2007.

[72] J. Geanakoplos *et al.*, "Getting at systemic risk via an agent-based model of the housing market," *Am. Econ. Rev.*, vol. 102, no. 3, pp. 53–58, 2012.

[73] L. Tang, J. Wu, L. Yu, and Q. Bao, "Carbon emissions trading scheme exploration in China: A multi-agent-based model," *Energy Policy*, vol. 81, pp. 152–169, 2015.

[74] C. Bean and C. Kambhampati, "Autonomous clustering using rough set theory," *Int. J. Autom. Comput.*, vol. 5, no. 1, pp. 90–102, 2008.

[75] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999.

[76] J. H. Ward Jr, "Hierarchical grouping to optimize an objective function," *J. Am. Stat. Assoc.*, vol. 58, no. 301, pp. 236–244, 1963.

[77] J. C. Bezdek, J. Keller, R. Krisnapuram, and N. Pal, *Fuzzy models and algorithms for pattern recognition and image processing*, vol. 4. Springer Science & Business Media, 2006.

[78] R. R. Yager, "Intelligent control of the hierarchical agglomerative clustering process," *IEEE Trans. Syst. Man Cybern. Part B*, vol. 30, no. 6, pp. 835–845, 2000.

[79] G. Nagy, "State of the art in pattern recognition," *Proc. IEEE*, vol. 56, no. 5, pp. 836–863, 1968.

[80] R. A. Baeza-Yates, "Introduction to Data Structures and Algorithms Related to Information Retrieval," in *Information Retrieval: Data Structures and Algorithms*, W. B. Frakes and R. A.

Baeza-Yates, Eds. Upper Saddle River, NJ: Prentice Hall, 1992, pp. 13–27.

[81] A. K. Jain and R. C. Dubes, “Algorithms for clustering data,” 1988.

[82] R. S. Michalski and R. E. Stepp, “Automated construction of classifications: Conceptual clustering versus numerical taxonomy,” *IEEE Trans. Pattern Anal. Mach. Intell.*, no. 4, pp. 396–410, 1983.

[83] K. C. Gowda and G. Krishna, “Agglomerative clustering using the concept of mutual nearest neighbourhood,” *Pattern Recognit.*, vol. 10, no. 2, pp. 105–112, 1978.

[84] E. M. Knox and R. T. Ng, “Algorithms for mining distancebased outliers in large datasets,” in *Proceedings of the International Conference on Very Large Data Bases*, 1998, pp. 392–403.

[85] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “LOF: identifying density-based local outliers,” in *ACM sigmod record*, 2000, vol. 29, no. 2, pp. 93–104.

[86] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” *Kdd*, vol. 96, no. 34, pp. 226–231, 1996.

[87] G. Karypis Eui-Hong Han Vipin Kumar, “CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling,” *Computer (Long. Beach. Calif).*, vol. 32, no. 8, pp. 68–75, 1999.

[88] G. Karypis and V. Kumar, “A hypergraph partitioning package.” Citeseer, 1998.

[89] S. Guha, R. Rastogi, and K. Shim, “Cure: an efficient clustering algorithm for large databases,” *Inf. Syst.*, vol. 26, no. 1, pp. 35–58, 2001.

[90] J. A. S. Almeida, L. M. S. Barbosa, A. A. C. C. Pais, and S. J. Formosinho, “Improving hierarchical cluster analysis: A new method with outlier detection and automatic clustering,” *Chemom. Intell. Lab. Syst.*, vol. 87, no. 2, pp. 208–217, 2007.

[91] P. J. Rousseeuw, “Silhouettes: a graphical aid to the interpretation and validation of cluster

analysis," *J. Comput. Appl. Math.*, vol. 20, pp. 53–65, 1987.

[92] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *Comput. Intell. AI Games, IEEE Trans.*, vol. 5, no. 4, pp. 293–311, 2013.

[93] G. Iuhasz, V. I. Munteanu, and V. Negru, "A Survey of Adaptive Game AI: Considerations for Cloud Deployment," in *Intelligent Distributed Computing VII*, Springer, 2014, pp. 309–315.

[94] E. W. Dereszynski, J. Hostetler, A. Fern, T. G. Dietterich, T.-T. Hoang, and M. Udarbe, "Learning Probabilistic Behavior Models in Real-Time Strategy Games.," in *AIIDE*, 2011.

[95] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, 2009, pp. 140–147.

[96] A. E. Gutierrez-Rodr\'\iguez, J. F. Mart\'\inez-Trinidad, M. Garc\'\ia-Borroto, and J. A. Carrasco-Ochoa, "Mining patterns for clustering on numerical datasets using unsupervised decision trees," *Knowledge-Based Syst.*, vol. 82, pp. 70–79, 2015.

[97] G. Synnaeve and P. Bessière, "A Dataset for StarCraft AI and an Example of Armies Clustering," in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[98] M. Buro, "Real-time strategy games: A new AI research challenge," in *IJCAI*, 2003, pp. 1534–1535.

[99] V. F. Farias, C. C. Moallemi, B. Van Roy, and T. Weissman, "Universal reinforcement learning," *Inf. Theory, IEEE Trans.*, vol. 56, no. 5, pp. 2441–2454, 2010.

[100] S. C. J. Bakkes, P. H. M. Spronck, and H. J. Van Den Herik, "Opponent modelling for case-based adaptive game AI," *Entertain. Comput.*, vol. 1, no. 1, pp. 27–37, 2009.

[101] G. Synnaeve and P. Bessiere, "A bayesian model for opening prediction in rts games with

application to starcraft," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, 2011, pp. 281–288.

[102] J.-L. Hsieh and C.-T. Sun, "Building a player strategy model by analyzing replays of real-time strategy games," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2008, pp. 3106–3111.

[103] M. Leece and A. Jhala, "Opponent state modeling in RTS games with limited information using Markov random fields," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, 2014, pp. 1–7.

[104] J. Hostetler, E. W. Dereszynski, T. G. Dietterich, and A. Fern, "Inferring strategies from limited reconnaissance in real-time strategy games," *arXiv Prepr. arXiv1210.4880*, 2012.

[105] M. Stanescu and M. Čertick\`y, "Predicting Opponent's Production in Real-Time Strategy Games With Answer Set Programming," *IEEE Trans. Comput. Intell. AI Games*, vol. 8, no. 1, pp. 89–94, 2016.

[106] A. Drachen, A. Canossa, and G. N. Yannakakis, "Player modeling using self-organization in Tomb Raider: Underworld," in *2009 IEEE symposium on computational intelligence and games*, 2009, pp. 1–8.

[107] K. Yasui, K. Kobayashi, K. Murakami, and T. Naruse, "Analyzing and learning an opponent's strategies in the RoboCup small size league," in *Robot Soccer World Cup*, 2013, pp. 159–170.

[108] G. Synnaeve and P. Bessiere, "A Bayesian model for RTS units control applied to StarCraft," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, 2011, pp. 190–196.

[109] A. Tavares, H. Azpúrua, A. Santos, and L. Chaimowicz, "Rock, Paper, StarCraft: Strategy Selection in Real-Time Strategy Games," in *Twelfth Artificial Intelligence and Interactive Digital*

Entertainment Conference, 2016.

[110] K. J. Levy, “Large-sample many-one comparisons involving correlations, proportions, or variances.,” *Psychol. Bull.*, vol. 82, no. 2, p. 177, 1975.

[111] S. T. Wagh and N. A. Razvi, “Marascuilo method of multiple comparisons (an analytical study of caesarean section delivery),” *Int J Contemp Med Res*, vol. 3, no. 4, pp. 1137–1140, 2016.

[112] R. Hayes, M. Paddrik, A. Todd, S. Yang, P. Beling, and W. Scherer, “Agent based model of the e-mini future: application for policy making,” in *Simulation Conference (WSC), Proceedings of the 2012 Winter*, 2012, pp. 1–12.

[113] J. H. Holland and J. H. Miller, “Artificial adaptive agents in economic theory,” *Am. Econ. Rev.*, vol. 81, no. 2, pp. 365–370, 1991.

[114] J. Conlisk, “Why bounded rationality?,” *J. Econ. Lit.*, vol. 34, no. 2, pp. 669–700, 1996.

[115] S.-H. Chen and C.-H. Yeh, “On the emergent properties of artificial stock markets: the efficient market hypothesis and the rational expectations hypothesis,” *J. Econ. Behav. Organ.*, vol. 49, no. 2, pp. 217–239, 2002.

[116] S. Morris and H. S. Shin, “Financial regulation in a system context,” *Brookings Pap. Econ. Act.*, vol. 2008, no. 2, pp. 229–274, 2008.

[117] R. J. Shiller, “Human behavior and the efficiency of the financial system,” *Handb. Macroecon.*, vol. 1, pp. 1305–1340, 1999.

[118] H. A. Simon, “Theories of decision-making in economics and behavioral science,” *Am. Econ. Rev.*, vol. 49, no. 3, pp. 253–283, 1959.

[119] S. Yang *et al.*, “Behavior based learning in identifying high frequency trading strategies,” in

Computational Intelligence for Financial Engineering & Economics (CIFEr), 2012 IEEE Conference on, 2012, pp. 1–8.

[120] W. F. Sharpe, “Asset allocation: Management style and performance measurement,” *J. Portf. Manag.*, vol. 18, no. 2, pp. 7–19, 1992.

[121] W. Fung and D. A. Hsieh, “Empirical characteristics of dynamic trading strategies: The case of hedge funds,” *Rev. Financ. Stud.*, vol. 10, no. 2, pp. 275–302, 1997.

[122] S. J. Brown and W. N. Goetzmann, “Mutual fund styles,” *J. financ. econ.*, vol. 43, no. 3, pp. 373–399, 1997.

[123] B. Liang, “On the performance of hedge funds,” *Financ. Anal. J.*, vol. 55, no. 4, pp. 72–85, 1999.

[124] J. Hasanhodzica and A. W. Lob, “CAN HEDGE-FUND RETURNS BE REPLICATED?: THE LINEAR CASE⋆,” *J. Invest. Manag.*, vol. 5, no. 2, pp. 5–45, 2007.

[125] N. Amenc, L. Martellini, J.-C. Meyfredi, and V. Ziemann, “Passive hedge fund replication--Beyond the linear case,” *Eur. Financ. Manag.*, vol. 16, no. 2, pp. 191–210, 2010.

[126] R. McDonald, *Derivative Markets*, 2nd ed. Boston, Massachusetts: Pearson Education, 2006.

[127] D. Bertsimas, L. Kogan, and A. W. Lo, “Hedging Derivative Securities and Incomplete Markets: An ε-Arbitrage Approach,” *Oper. Res.*, pp. 372–397, 2001.

[128] H. M. Kat and H. P. Palaro, “Who needs hedge funds? A copula-based approach to hedge fund return replication,” 2005.

[129] T. M. Therneau, E. J. Atkinson, and others, “An introduction to recursive partitioning using the RPART routines.” Technical Report 61. URL http://www. mayo. edu/hsr/techrpt/61. pdf, 1997.

[130] J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman, *Applied linear statistical models*, vol.

4. Irwin Chicago, 1996.

[131] D. Billings, D. Papp, J. Schaeffer, and D. Szafron, “Opponent Modeling in Poker.”

[132] U.S. Navy, “Warfighting Chess Games and Pieces.” 2018.

[133] B. G. Silverman, G. Bharathy, K. O’Brien, and J. Cornwell, “Human behavior models for agents in simulators and games: part II: gamebot engineering with PMFserv,” *Presence Teleoperators Virtual Environ.*, vol. 15, no. 2, pp. 163–185, 2006.

[134] D. W. Bunn and F. S. Oliveira, “Agent-based simulation-an application to the new electricity trading arrangements of England and Wales,” *IEEE Trans. Evol. Comput.*, vol. 5, no. 5, pp. 493–503, 2001.

[135] G. Tecuci and Y. Kodratoff, “APPRENTICESHIP LEARNING IN IMPERFECT DOMAIN THEORIES.”

[136] D. C. Wilkins, “Apprenticeship Learning Techniques for Knowledge Based Systems.” Defense Technical Information Center, 1988.

[137] W. B. Arthur, “Designing Economic Agents that Act like Human Agents: A Behavioral Approach to Bounded Rationality,” *The American Economic Review*, vol. 81. American Economic Association, pp. 353–359.

[138] A. E. Roth and I. Erev, “Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term,” *Games Econ. Behav.*, vol. 8, no. 1, pp. 164–212, Jan. 1995.

[139] J. F. Nash and others, “Equilibrium points in n-person games,” *Proc. Natl. Acad. Sci.*, vol. 36, no. 1, pp. 48–49, 1950.

[140] M. L. Littman, “Markov games as a framework for multi-agent reinforcement learning,” in *Machine Learning Proceedings 1994*, Elsevier, 1994, pp. 157–163.

[141] R. A. Howard, *Dynamic programming and Markov processes*. Wiley for The Massachusetts Institute of Technology, 1964.

[142] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, 1992.

[143] G. Tesauro, "Temporal difference learning and TD-Gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[144] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artif. Intell.*, vol. 55, no. 2–3, pp. 311–365, 1992.

[145] M. J. Mataric, "Reward functions for accelerated learning," in *Machine Learning Proceedings 1994*, Elsevier, 1994, pp. 181–189.

[146] D. Ramachandran and E. Amir, "Bayesian Inverse Reinforcement Learning."

[147] P. Abbeel and A. Y. Ng, "Apprenticeship Learning via Inverse Reinforcement Learning."

[148] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning.," in *Icml*, 2000, pp. 663–670.

[149] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An Application of Reinforcement Learning to Aerobatic Helicopter Flight."

[150] D. Vasquez, B. Okal, and K. O. Arras, "Inverse reinforcement learning algorithms and features for robot navigation in crowds: an experimental comparison," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 2014, pp. 1341–1346.

[151] S. J. Lee and Z. Popović, "Learning behavior styles with inverse reinforcement learning," in *ACM Transactions on Graphics (TOG)*, 2010, vol. 29, no. 4, p. 122.

[152] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum Entropy Inverse Reinforcement

Learning.," in *AAAI*, 2008, vol. 8, pp. 1433–1438.

[153] S. Levine, Z. Popovic, and V. Koltun, "Nonlinear inverse reinforcement learning with gaussian processes," in *Advances in Neural Information Processing Systems*, 2011, pp. 19–27.

[154] J. Choi and K.-E. Kim, "Inverse reinforcement learning in partially observable environments," *J. Mach. Learn. Res.*, vol. 12, no. Mar, pp. 691–730, 2011.

[155] C. Thurau, C. Bauckhage, and G. Sagerer, "Imitation learning at all levels of game-AI," in *Proceedings of the international conference on computer games, artificial intelligence, design and education*, 2004, vol. 5.

[156] C. Thurau, C. Bauckhage, and G. Sagerer, "Learning human-like movement behavior for computer games," in *Proc. Int. Conf. on the Simulation of Adaptive Behavior*, 2004, pp. 315–323.

[157] B. Gorman and M. Humphrys, "Imitative learning of combat behaviours in first-person computer games," *Proc. CGAMES*, 2007.

[158] B. Geisler, "An Empirical Study of Machine Learning Algorithms Applied to Modeling Player Behavior in a" First Person Shooter" Video Game," Citeseer, 2002.

[159] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.

[160] B. Tastan and G. R. Sukthankar, "Learning Policies for First Person Shooter Games Using Inverse Reinforcement Learning.," in *AIIDE*, 2011.

[161] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Broodwar," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, 2012, pp. 402–408.

[162] Z. Academy, “Neural Network w/ JAVA (Backpropagation 02) - Tutorial 10,” *Zane Acdemy*, 2017. [Online]. Available: http://www.zaneacademy.com/.

[163] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[164] A. Giordana, F. Neri, and L. Saitta, “Automated learning for industrial diagnosis,” *Fielded Appl. Mach. Learn. Morgan Kaufmann, San Fr. to be Publ.*, 1996.

[165] D. Michie, “Current developments in expert systems,” in *Proceedings of the Second Australian Conference on Applications of expert systems*, 1987, pp. 137–156.

[166] U. M. Fayyad, P. Smyth, N. Weir, and S. Djorgovski, “Automated analysis and exploration of image databases: Results, progress, and challenges,” *J. Intell. Inf. Syst.*, vol. 4, no. 1, pp. 7–25, 1995.

[167] G. Holmes, A. Donkin, and I. H. Witten, “Weka: A machine learning workbench,” in *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, 1994, pp. 357–361.

[168] J. R. Quinlan, “Improved use of continuous attributes in C4. 5,” *J. Artif. Intell. Res.*, vol. 4, pp. 77–90, 1996.

[169] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, 1943.

[170] P. Cadena and L. Garrido, “Fuzzy case-based reasoning for managing strategic and tactical reasoning in starcraft,” in *Mexican International Conference on Artificial Intelligence*, 2011, pp. 113–124.

[171] S. Wender and I. Watson, “Integrating case-based reasoning with reinforcement learning for real-time strategy game micromanagement,” in *Pacific Rim International Conference on Artificial*

Intelligence, 2014, pp. 64–76.

[172] A. Aamodt and E. Plaza, “Case-based reasoning: Foundational issues, methodological variations, and system approaches,” *AI Commun.*, vol. 7, no. 1, pp. 39–59, 1994.

[173] C. M. Macal and M. J. North, “Tutorial on agent-based modelling and simulation,” *J. Simul.*, vol. 4, no. 3, pp. 151–162, 2010.

[174] L. Tesfatsion, “Agent-based computational economics: Growing economies from the bottom up,” *Artif. Life*, vol. 8, no. 1, pp. 55–82, 2002.

Appendix A: Single Factor ANOVA Results

Single Factor ANOVA – Scenario 1: Pruned Tree

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
2 - Trajectory	60	54.41824	0.906971	0.003378
4 - Trajectory	60	56.97535	0.949589	0.001504
6 - Trajectory	60	57.85263	0.964211	0.001302
8 - Trajectory	60	58.5801	0.976335	0.000814
10 - Trajectory	60	59.12638	0.98544	0.000403

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.227477	4	0.056869	38.41603	7.12E-26	2.402248
Within Groups	0.436704	295	0.00148			
Total	0.664181	299				

Single Factor ANOVA – Scenario 1: Unpruned Tree

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
2 - Trajectory	60	53.65027	0.894171	0.004744
4 - Trajectory	60	57.57162	0.959527	0.001146
6 - Trajectory	60	58.27376	0.971229	0.00134
8 - Trajectory	60	58.73488	0.978915	0.00201
10 - Trajectory	60	58.60351	0.976725	0.002187

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.301304	4	0.075326	32.95908	1.01E-22	2.402248
Within Groups	0.674204	295	0.002285			
Total	0.975507	299				

Single Factor ANOVA – Scenario 1: Neural Network

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
2 – Trajectories	60	42.01735	0.700289	0.01549
4 – Trajectories	60	43.88827	0.731471	0.004473
6 – Trajectories	60	43.09786	0.718298	0.016741
8 – Trajectories	60	43.65091	0.727515	0.00516
10 – Trajectories	60	43.18504	0.719751	0.006288

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.034687	4	0.008672	0.900445	0.464017	2.402248
Within Groups	2.840975	295	0.00963			
Total	2.875662	299				

Single Factor ANOVA – Scenario 1: Case-Based Inverse Reinforcement Learning

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
2 – Trajectories	60	40.73944	0.678991	0.016708
4 – Trajectories	60	38.57988	0.642998	0.025044
6 – Trajectories	60	38.05185	0.634198	0.036961
8 – Trajectories	60	39.06667	0.651111	0.035098
10 – Trajectories	60	39.04951	0.650825	0.029004

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.067676	4	0.016919	0.592336	0.668443	2.402248
Within Groups	8.426099	295	0.028563			

Appendix B: T-Test Neural Network and CBIRL

t-Test Unequal Variance Scenario 1 Accuracy Neural Network Vs. CBIRL

	Neural Network	*CBIRL*
Mean	0.719464762	0.651624548
Variance	0.009617598	0.028407272
Observations	300	300
Hypothesized Mean Difference	0	
Df	481	
t Stat	6.025789637	
P(T<=t) one-tail	1.67518E-09	
t Critical one-tail	1.648027693	
P(T<=t) two-tail	3.35035E-09	
t Critical two-tail	1.964908164	

T-Test Unequal Variance Scenario 2 Accuracy Tree Algorithm Comparison

	Tree Unpruned	Tree Pruned
Mean	0.670036	0.754848
Variance	0.004486	0.00831
Observations	100	100
Hypothesized Mean Difference	0	
Df	182	
t Stat	-7.49733	
P(T<=t) one-tail	1.38E-12	
t Critical one-tail	1.653269	
P(T<=t) two-tail	2.76E-12	
t Critical two-tail	1.973084	

T-Test Unequal Variance Secnario 2 Unpruned Tree Vs. Neural Network

	Tree Unpruned	Neural Network
Mean	0.670036	0.610538
Variance	0.004486	0.008337
Observations	100	100
Hypothesized Mean Difference	0	
Df	182	
t Stat	5.254258	
P(T<=t) one-tail	2.06E-07	
t Critical one-tail	1.653269	
P(T<=t) two-tail	4.13E-07	

t Critical two-tail	1.973084

Appendix C: Mood's Median Test

Table 14: Mood's Median Test Comparing Neural Network to Trained Reinforcement Agent

C11	Median	N <= Overall Median	N > Overall Median	Q3 – Q1	95% Median CI
Expert Policy	6	86	14	1	(6, 6)
Neural Network	9	23	77	2	(8, 9)
Overall	6				

95.0% CI for median(Expert Policy) - median(Neural Network): (-3,-2)

Test

Null hypothesis	H_0: The population medians are all equal
Alternative hypothesis	H_1: The population medians are not all equal

DF	Chi-Square	P-Value
1	80.03	0.000

Table 15 Moods Median Test Comparing Classification Trees to Baseline

C2	Median	N < Overall Median	N >= Overall Median	Q3 – Q1	95% Median CI
Baseline	5	31	69	2	(5, 6)
Tree Pruned	5	31	69	2	(5, 5)
Tree Unprune	4	58	42	1	(4, 5)
Overall	5				

Test

Null hypothesis	H_0: The population medians are all equal
Alternative hypothesis	H_1: The population medians are not all equal

DF	Chi-Square	P-Value
2	20.25	0.000

Table 16: Mood's Median Test Comparing Pruned Tree to Baseline

C5	Median	N <= Overall Median	N > Overall Median	Q3 – Q1	95% Median CI
Baseline	5	59	41	2	(5, 6)
Tree Pruned	5	68	32	2	(5, 5)
Overall	5				

95.0% CI for median(Baseline) - median(Tree Pruned): (0,0)

Test

Null hypothesis	H_0: The population medians are all equal
Alternative hypothesis	H_1: The population medians are not all equal

DF	Chi-Square	P-Value
1	1.75	0.186

Table 17: Mood's Median Test Comparing Pruned and Unpruned Tree

C8	Median	N < Overall Median	N >= Overall Median	Q3 – Q1	95% Median CI
Tree Pruned	5	31	69	2	(5, 5)
Tree Unprune	4	58	42	1	(4, 5)
Overall	5				

95.0% CI for median(Tree Pruned) - median(Tree Unprune): (1,1)

Test

Null hypothesis	H_0: The population medians are all equal
Alternative hypothesis	H_1: The population medians are not all equal

DF	Chi-Square	P-Value
1	14.76	0.000