12-14-2017

Towards Data-driven Simulation Modeling for Mobile Agent-based Systems

Nicholas Aiden Keller

TOWARDS DATA-DRIVEN SIMULATION MODELING FOR MOBILE AGENT-BASED SYSTEMS

by

NICHOLAS KELLER

Under the Direction of Xiaolin Hu, PhD

ABSTRACT

Simulation modeling provides insight into how dynamic systems work. Current simulation modeling approaches are primarily knowledge-driven, which involves a process of converting expert knowledge into models and simulating them to understand more about the system. Knowledge-driven models are useful for exploring the dynamics of systems, but are handcrafted which means that they are expensive to develop and reflect the bias and limited knowledge of their creators. To address limitations of knowledge-driven simulation modeling, this dissertation develops a framework towards data-driven simulation modeling that discovers simulation models in an automated way based on data or behavior patterns extracted from systems under study. By using data, simulation models can be discovered automatically and with

less bias than through knowledge-driven methods. Additionally, multiple models can be discovered that replicate the desired behavior. Each of these models can be thought of as a hypothesis about how the real system generates the observed behavior. This framework was developed based on the application of mobile agent-based systems. The developed framework is composed of three components: 1) model space specification; 2) search method; and 3) framework measurement metrics. The model space specification provides a formal specification for the general model structure from which various models can be generated. The search method is used to efficiently search the model space for candidate models that exhibit desired behavior. The five framework measurement metrics: flexibility, comprehensibility, controllability, compossability, and robustness, are developed to evaluate the overall framework. Furthermore, to incorporate knowledge into the data-driven simulation modeling framework, a method was developed that uses System Entity Structures (SESs) to specify incomplete knowledge to be used by the model search process. This is significant because knowledge-driven modeling requires a complete understanding of a system before it can be modeled, whereas the framework can find a model with incomplete knowledge. The developed framework has been applied to mobile agent-based systems and the results demonstrate that it is possible to discover a variety of interesting models using the framework.

INDEX WORDS: Swarm, System Entity Structure (SES), Flock, Crowd, Methodology

TOWARDS DATA-DRIVEN SIMULATION MODELING FOR MOBILE AGENT-BASED SYSTEMS

by

NICHOLAS KELLER

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2017

TOWARDS DATA-DRIVEN SIMULATION MODELING FOR MOBILE AGENT-BASED SYSTEMS

by

NICHOLAS KELLER

Committee Chair: Xiaolin Hu

Committee: Robert Harrison

Xin Qi

Ying Zhu

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

December 2017

DEDICATION

To my Mother and Father.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Xiaolin Hu for teaching me how to conduct and document my research. I would also like to thank my Committee members Dr. Robert Harrison, Dr. Xin Qi, and Dr. Ying Zhu. Finally, I would like to thank the Computer Science Department at Georgia State University for giving me this opportunity.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Introduction to Modeling

People model things all the time whether they realize it or not. In conversation we use a model of the other person to leave out details that we think they are either uninterested in our already know. Sometimes our internal models are wrong, such as when a computer science PhD student teaches their first class and doesn't realize that the class hasn't fully learned Java syntax yet. People revise their models based on experience that is either direct; the look of confusion on students' faces, or vicariously such as through books.

A model is an abstract representation of some aspect of the real world. All models are wrong, because they aren't a perfect representation of the real world, but some are useful. Creating a useful model requires determining what to leave out of a model and what to include based on its purpose. When deciding whether to keep lifesavings as cash or in stocks it is useful to know that inflation will overtime eat away at the purchasing power of cash. When day trading stock at a hedge fund a much more detailed model of inflation and its relationship to interest rates is needed. The more detailed a model the more expensive it is to create and use. For the hedge fund this extra cost is worth it, but for the individual investor it is a waste. The goal of all models is to provide some knowledge about the real system by simplifying it.

1.2 Computer Modeling

Although people are natural modelers they frequently aren't particularly good modelers. Many systems are well understood by experts; however, the dynamics of the system are too complicated to understand without the help of computers. One such system is supply chain logistics, where delays and feedback loops can create behavior that people have a hard time

predicting without stock and flow computer models [22]. Computers make it possible to build, analyze, and use complex models.

Computers, which don't have the same processing limitations as people, can simulate the interaction between multiple components. For example, a model of a crowd can model the decision process of each person [21]. A computer model of child maltreatment can take into account the sharing of resources between families in a heterogeneous community [23]. A stock and flow model of the housing market can capture the complexity of the system in a way an un-aided human mind never could.

The data processing abilities of computers also makes it possible to analyze and use models in ways that aren't possible with peoples' internal models. For example, language translation models make language translation faster and cheaper than using human translators; although it is not as accurate yet. The model of the world that self-driving cars create and use has the potential to substantially reduce accidents, because computers can react faster than humans to possible collisions and can see more through better sensors. Even when a computer model is easy to understand and interpret, which is not necessarily the case in the language translation and self-driving car examples, computers lets us run thousands of experiments.

Computer modeling has two broad applications: 1) making predictions, and 2) testing hypotheses about how real systems work. Some models like those of weather and image recognition are focused on prediction, whereas others like the Boids model of flocking [3] are focused on hypothesis testing. The Boids model [3] hypothesizes that just three behaviors lead to bird flocking behavior; alignment, collision avoidance, and cohesion. This hypothesis is supported when the boids model generated flocking behavior similar to that observed in nature. Frequently, prediction and hypothesis testing are pursued in parallel when a model is developed

based on some theory and then, after it is validated by comparing its behavior to the real system, it is used to make some predictions about the system. Computer modeling can be approached from a data-driven or knowledge-driven perspective as will be discussed next.

1.3 Data-Driven Modeling

Data-driven modeling is used to refer modeling techniques that use data to build models. Data-driven modeling also goes by the names artificial intelligence or machine learning. Some examples of big data are: data collected from Tesla's cars while they are driven [31], labeled images taken from the internet [24], and United Nations (U.N.) meeting translations [25]. Data-driven modeling makes creating large models fast and relatively inexpensive. It is a lot easier to build a language translation program using data, then to hand code every quirk of a language. Furthermore, it may not even be possible to handcraft a language translation program, because the language may evolve faster than you can encode a model for it. The goal of data-driven modeling is primarily prediction.

The primary disadvantage of data-driven modeling is that it results in incomprehensible models, which makes them difficult to validate. Although on a technical level a model created using data can be read, on a conceptual level it is opaque; it's just a jumble of numbers that hides its structure. In an admission of this problem, DARPA recently asked for proposals on how to assure the correctness of data-driven models [26]. The only way to check the correctness of data-driven models is through tests with sample data. Typically data used to train a model is split into data used for training and data used for testing the trained model for accuracy. Testing a model on data used to train it can lead to a model looking better than it is because of overfitting the training data. Unlike code, data-driven models can't be examined to see if they are correct. This leaves open the possibility that a model will behave incorrectly in a scenario it wasn't tested

against. For example, a computer vision program used to recognize street signs was trained to ignore a stop sign with a sticky note on it [27]. It is impossible to test for all kinds of malicious information that might be surreptitiously embedded in a data-driven model. An alternative to data-driven modeling is knowledge-driven modeling that gets rid of the opaqueness problem of data-driven models, but sacrifices the ability to quickly build large complex models using data.

1.4 Knowledge-Driven Modeling

Knowledge-driven modeling is used to refer to modeling techniques that use knowledge from experts to build models. Unlike data-driven models, these models are handcrafted. A given model is made up of components taken from theory, which results in comprehensible models. For example, theories of crowd behavior that rely on the individual decisions of people in the crowd are best represented with agent-based models, where each person is modeled individually [21]. Other theories see people evacuating a building as flows of people in a macro level network model where nodes are rooms and edges represent the flow capacity of doors and hallways [28]. Although different, it is possible that these two models can both be correct at their own level of abstraction. Computer modeling can help formalize a theory, test it, and work out its implications. If a system is well understood (or we think it is) then knowledge-driven modeling is a good fit.

Unlike data-driven models knowledge-driven models can be simulated. For this reason knowledge-driven modeling is sometimes also called simulation modeling. A simulation is a story about how a system works and is similar, but distinct from prediction. Simulation is about the journey not just the destination; how a system reaches an end state is just as important as what that end state is. This contrasts to models which are built exclusively to make predictions,

such as stock market trading bots [29]. Simulation models are built out of components inspired by theories about how a system works.

At a mechanical level simulation models are rules about how a system state changes from one time to another. Let us say we want to model the trajectory of a soccer ball for 10 seconds just after its been kicked into the air. A discrete time model would split up the trajectory into equal time segments; say 1 second per segment. The modeling problem then becomes predicting what the location of the ball will be 1 second in the future based on the soccer ball's speed, the effect of gravity, wind resistance, and the possibility that it will hit people or the ground. Since the real world doesn't operate in 1 second increments the model will be inaccurate. A collision that might occur at 1.1 seconds will instead be represented as occurring at either 1 or 2 seconds. To improve accuracy the time segments can be made smaller, but this increases the computational cost of running the model.

Continuing the soccer ball example, we can improve model accuracy while potentially speeding up computation by using a continuous time model that calculates the next event. In this example, the soccer ball is flying through the air and the next possible events are that it hits the ground or hits another player (to improve efficiency the rare event of a goal is intentionally not modeled). If the next event is say 5 seconds in the future then only 1 event needs to be calculated instead of 5 separate discrete time states. The ball of course could have more than 10 events over 10 seconds (for example, the ball bouncing), but this model would still be better than the discrete time model, because it would simulate those events. The only down side is that calculating the next event can be very time consuming; each player has to be tested to see if and when it will hit the soccer ball and the earliest event is chosen. Sometimes the cost of calculating the next event can be high enough to make a discrete time model more efficient.

Discrete time models can be made as accurate as needed by decreasing the time increment and are easier to develop than continuous time models. It is for this reason that one modeling technique is to develop a discrete time model and once its value is proven it can be converted (with great effort) to a continuous time model to improve efficiency. With this in mind the model used in this dissertation is a discrete time model.

1.5 Current Limitations of Knowledge-Driven Modeling

A system can be represented by multiple simulation models that each approaches the system from a different level of abstraction; as in the crowd behavior example mentioned in the previous section. What distinguishes simulation modeling from other modeling techniques is that it uses abstractions that relate to some theory about a system. One issue in simulation modeling is that the process of converting theory into a simulation model is ad-hoc. However, the real elephant in the room of simulation modeling is that a simulation model is only as strong as the theory it is built upon; what if that theory is flawed or incomplete?

A simulation model that replicates the behavior of a real world system is not necessarily an accurate representation of it. If a model reproduces observed behavior, then this lends support to the theories behind the model, but doesn't prove those theories are correct. A model whose purpose is purely predictive, not a simulation model, is evaluated based on the accuracy of its predictions. A simulation model in addition to its predictive accuracy is also evaluated by how accurately it represents the real system. If the real system is opaque, and almost all real systems are at least partially unobservable, then the model's predictive accuracy is used to evaluate a simulation model's structure. However, this method cannot fully validate a model because a model can be predictive without being structured like the real system. It is not without merit either, because a model structured like the real system should behave like the system. It is for

this reason that a simulation model can be thought of as an embodied hypothesis about how a system works and not necessarily an actual abstraction of the real system.

If a simulation model perfectly replicates the behavior of the real system, than there is a high likelihood that its structure reflects the structure of the underlying system; however, there is usually ambiguity when comparing model to system behavior. When modeling a system the goal is to replicate the "behavior" of the system and not the exact time series states of the system. Even if a model perfectly represented a system small variations in initial conditions would lead it to vary from the real system slightly. Therefore, measurements of behavior are subjective even when quantitative functions are used. For example, the path of a hurricane can be measured quantitatively both in the real system and a model, but what level of difference is acceptable? This can partially be answered by simply preferring models with better predictions, but when is a model as good as it can be? In many domains it is even a challenge to quantitatively measure behavior. An early model of bird blocking, called Boids [3], presented pictures of simulated bird flocks as a form of evidence.

A simulation model that reproduces real system behavior provides some knowledge about that system, but should not be thought of as the final say in how a system works. However, the painstaking process by which simulation models are created by hand results in too few models of a system. Furthermore, since models are generated by hand they can only test theories and not generate new ones, because they reflect the preconceptions of their creators, which results in biased models. The solution is to generate human readable (comprehensible) models automatically. These models would be easy to interpret like knowledge driven models, but would not have the bias inherent in the creation of knowledge driven models. The framework presented in this dissertation takes the first step towards realizing this solution.

1.6 Agent-Based Modeling

The framework presented in this dissertation uses agent-based modeling, which is a type of knowledge-driven modeling. In an agent-based model each entity (agent) in a system is modeled separately. For example, in crowd model each person is modeled individually or in a model of traffic each car would be modeled separately. Each agent can only observe part of the environment and neighboring agents. For example, a person in a crowd can only observe their neighbors and nearby obstacles. Although each agent acts separately based on local knowledge a collective behavior emerges. Agents act based on what they see their neighbors doing, so information and behavior can "ripple" through an environment. For example, one car randomly slows down, which causes the car behind it to slow down and the car behind that to slow down as well and so on until a traffic jam results; this has been demonstrated with real cars on a circular closed course [30]. Agents don't have to take up physical space. For example, they could be people connected through a social network or companies in an economy connected through supply chains.

Mobile agent models represent a large subset of agent-based models and is the type of model studied using the presented framework. A mobile agent model is one were agents move in a space representative of a real world environment. The space can be 2D as is the case with people in a crowd. Or 3D as is the case with simulated flocks of birds and fish. Or even 1D in a model of a 1 lane road. In a mobile agent model, agents can view nearby agents and environmental features (walls in a room for example). A simple model of vision is that everything within some distance to the agent is visible. For mobile agent models of living things a vision model based on a field of view is more realistic. However, although people have a limited field of view if they are constantly looking around themselves, then it might be realistic

to model their field of view as being 360° or at least greater than their physiological field of view. The framework focuses on mobile agent modeling, although it can be generalized to other types of modeling, because it is a significant area of research.

1.7 Outline of Framework

This dissertation proposes a framework for using data to automatically generate simulation models, so that models can be created faster and with less bias than current handcrafting simulation modeling methodologies. A simulation model is a hypothesis about how a system works, so a framework for generating these hypotheses with less bias is a significant contribution. Existing data-driven modeling methods are focused on prediction and result in black box models that provide no insight into how systems produce behavior unlike the proposed method. This is the first framework for discovering simulation model structure automatically using data. To demonstrate the utility of the proposed framework it was applied to the mobile agent domain, although the framework is general enough to be applied to any domain. The major contributions of the dissertation are to the mobile agent domain and modeling methods more generally.

The approach is to define a very large space of possible models of real systems, then search through this space for a model which generates a real systems behavior. Since models can be discovered automatically, multiple models can be discovered, which opens up a whole new way of studying systems. Multiple models means multiple hypothesis about how a real system works, whereas existing modeling techniques have limited themselves to only one hand crafted model, hypothesis, at a time. Under this framework the work of a modeler shifts from creating a model from scratch to the task of defining the model space and creating behavior description functions. The model space allows for a range of possible models instead of prescribing a

specific model, which both reduces modeler bias and has the potential to speed up model development.

The three broad research challenges were model space specification, model search, and the analysis of discovered models. Unlike black-box modeling the model space must contain comprehensible models whose behavior is easily understood by an examination of the model. The challenge is creating a model space that contains an interesting variety of models, but remains comprehensible. The model space is searched using an application of the genetic algorithm, but as will be discussed later there are issues unique to the framework that must be addressed. The model space search requires a model evaluation algorithm (fitness function) which requires a precise definition of the behavior being looked for. The fitness of a model is measured as the difference between the behaviors a model generates and the idealized behavior. As discussed in *1.5 Current Limitations of Knowledge-Driven Modeling* defining behavior quantitatively is challenging. The model that is discovered by the search process must be cleaned up to remove non-essential model components and thoroughly analyzed to yield insights into the system. Cleaning up a discovered model is a completely new modeling step that isn't required for data-driven models that are purely predictive or knowledge driven models that are comprehensible by virtue of being handcrafted.

The framework was applied to the mobile agent domain, but is not limited to this domain. The mobile agent modeling domain was chosen because of its broad applicability. Mobile agent modeling has been applied to building evacuations [3], understanding swarms of animals [1], and swarm robotics [4]. The framework uses agent-based modeling to study mobile agents interacting in a 2D environment with obstacles (such as walls).

To evaluate how well the framework was applied to the mobile agent domain we developed five measurement metrics: 1) flexibility, 2) comprehensibility, 3) controllability, 4) composability, and 5) robustness. If the framework were applied to another domain, than these measurement metrics could be used to evaluate this application. The framework is flexible if it can be used to discover a variety of non-trivial mobile agent models. Discovered models that are easily interpreted are comprehensible. A model is controllable if a modeler can comprehend it and then modify it to achieve predictable changes in behavior. Controllability is important because it demonstrates comprehensibility, but it is also of practical value in and of itself. For example, a model of pedestrians might be modified to change the personal space between people to reflect different culturally specific expectations of personal space. This would allow a model trained on one set of historical data to be used in another context. This sort of out of context model re-use is impossible with typical data-driven models. In the framework, models are composed of competing behaviors; cohesion and collision avoidance for example. Compossability means that discovered or hand crafted behaviors can be predictably combined to create new models. Compossability is important because it makes the framework more useful by allowing modelers to mix and match already discovered or known (hand crafted) behaviors. It makes the creation of a library of useful behaviors possible. Finally, models must be robust or they aren't useful. Robust models are one that aren't highly sensitive to specific conditions like the number of people, the size of the environment, or the number of obstacles in the environment. These metrics should be used to evaluate the framework's application to the mobile agent domain.

An outline of the framework is given in figure 1.1 below. The search step requires a model space to search through and historical data to compare model behavior to. After the search

step which can take many hours a candidate model is output. The candidate model is "fit," but is still considered a candidate because it is a hypothesis about how the system works. The suitability of a particular framework implementation is determined by how well it satisfies the five measurement metrics.

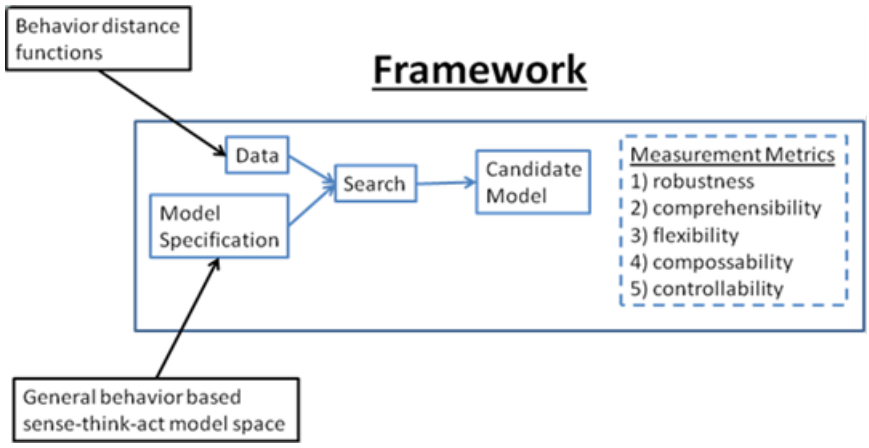

Figure 1-1 Framework

1.8 Incorporating Incomplete Knowledge Into the Framework Through System Entity Structures (SESs)

The framework uses System Entity Structures (SESs) to rigorously incorporate incomplete knowledge of mobile agent behavior. One of the earliest publications on SES is from 1990 [2]; however, its application in this framework is novel. Typically SES is used to represent different system structures, but in the framework it is used to represent incomplete knowledge of a system. Whereas knowledge driven modeling forces a modeler to completely understand a system to model it, the framework uses SES to allow the modeling of partially understood systems. Most systems are incompletely understood, so this technique can greatly expand the number of systems that can be represented by comprehensible models that provide insight into system behavior.

SESs are tree like structures that show all the different possible decompositions of a system model into its component entities. For example, a SES could be used to model a car. The car SES would show all the different possible cars that could be made from different components, such as different models of engine. An example of a car SES is given in figure 5-1 in chapter 5. A SES is pruned to create a Pruned Entity Structure (PES) which represents specific entities that are used to create a model. The last step is convert the PES into a model of the system that can be simulated.

In the case of the mobile agent framework, the entities are behavior restrictions. A behavior restriction defines a set of behaviors that satisfy those restrictions. The restrictions represent knowledge that the modeler wants to incorporate into the search process. When a behavior is unrestricted it means that the modeler has no knowledge about the purpose of a behavior (even whether it is needed in the first place). For example, let us say that a modeler is modeling a situation where child agents follow a teacher agent on a field trip. The modeler knows that at least one of the child agent's behaviors must be restricted to observe the teacher agent. Even though the modeler doesn't completely understand the system they can still provide what they do know. A PES in the framework represents a set of restricted and unrestricted behaviors. The PES is converted into a model by randomly selecting behaviors from the set of behaviors defined by the behavior restrictions.

1.9 Organization

The rest of this dissertation is organized as follows. The review of literature in chapter 2 places my research within the modeling and mobile agent domains. Chapter 3 presents the details of the framework including the model space specification, search method, and framework metrics. Chapter 4 is devoted to presenting and analyzing the models discovered using the

framework. Finally, chapter 5 discusses how the framework defined in chapter 3 is extended to incorporate incomplete knowledge through the use of System Entity Structures (SESs). Chapter 5 also includes an experimental results section where the SES extended framework is used to discover a lane formation model.

2 REVIEW OF LITERATURE

2.1 Introduction

This dissertation makes contributions to the mobile agent modeling domain. Mobile agent models fall within one of three major categories; knowledge-driven, data-driven, and hybrid. Knowledge-driven models are handcrafted by a modeler to reflect their understanding of the system. A knowledge-driven model is the formalization of theory into a program that allows for simulation, therefore these models are easy to understand. Data-driven models are derived from data in an automatic or semi-automatic manner. Data-driven models are focused on prediction and unlike knowledge-driven models, they do not provide knowledge about the underlying causes of system dynamics. Hybrid models combine an upper layer that determines agents' overall movement objectives (data-driven) with a microscopic lower level model which controls agent collision dynamics (knowledge-driven).

The proposed framework is unique in that it combines features from data-driven and knowledge-driven mobile agent modeling. Like data-driven modeling the framework automatically discovers models using data (or a quantitative description of the desired behavior). However, unlike data-driven modeling the proposed framework results in models that are comprehensible and provide insight into the underlying causes of mobile agent behavior. Like knowledge-driven modeling the framework produces readable models, but unlike knowledge-driven modeling the models are produced automatically (and with less bias). Thus the proposed

framework represents the first step in a new mobile agent modeling category that I propose should be called "data-driven simulation modeling."

2.2 Mobile Agent Modeling

A system can be simulated using a mobile agent-based model if it contains many similar agents, such as people, that move around in a shared environment, act autonomously, and only have local knowledge (and possibly global knowledge about the environment; like a familiar building's layout). Groups of animals whether they be flocks of birds, herds of cattle, or schools of fish can be studied using mobile agent modeling [3] [18]. Mobile agent-based modeling can also be used to study how groups of people move through a building [4], crowd [5], or in traffic [6]. Swarms of robots are an increasingly important area of research [7].

Studies of animal systems are mainly focused on theoretical research, whereas studies of human mobile-agent systems tend to have a more practical focus. Models of car traffic can be used to design road systems and to predict congestion on roads [19] [20]. Models of pedestrians can be used to design stadiums for large numbers of people under regular conditions and to safely evacuate people in emergencies [4]. Models can also be used to understand how people move through an existing space, so that this space can be modified to improve its efficiency [21].

There are three broad mobile agent modeling approaches; knowledge-driven, data-driven, and hybrid. Knowledge-driven models are handcrafted models that implement some theory about how a system works. Data-driven models use data to automatically generate a mobile agent model. Hybrid models have parts that are handcrafted and parts that are generated from data. Each of these mobile agent modeling methods has disadvantages relative to the proposed framework.

2.3 Knowledge-Driven Mobile Agent Modeling

Knowledge-driven mobile agent modeling results in very readable models. Experts can examine a model and decide whether they agree with how it abstracts the real system. Further validation comes from comparing its behavior to that of the real system. These types of models are best suited for modeling simple system behavior. As the behavior being modeled becomes more complex the knowledge needed to create the model becomes less available or reliable. Furthermore, the model becomes more complex which reduces its readability.

Knowledge-driven models can be used to help us understand how a system works, so we can intervene intelligently. For example, by understanding how traffic jams are created we can design more efficient traffic light signaling systems [9]. Another example is that by understanding how people can get crushed in emergency building evacuations [8], we can design better buildings and evacuation procedures.

Knowledge-driven models may use a fitness function to tune their parameters, but their structures are fixed by the modeler, which introduces bias. This contrasts to the proposed framework which explores the structure of models. The proposed framework reduces the bias inherent to knowledge-driven models while still discovering models that are as comprehensible as knowledge-driven models.

2.4 Data-Driven Mobile Agent Modeling

Data-driven mobile agent modeling results in unreadable, but predictive, models. Data can take the form of video of a crowd that is processed to calculate trajectories for individual people [11]. Another possible source of data is sensors in highway networks [12]. These models can only be evaluated based on the quality of their predictions, because the models are opaque.

Data-driven mobile agent modeling can be used to make predictions for systems that there exists data for. One application is predicting how a traffic jam will affect an entire road system [9].

The model created in [10] is typical of data-driven modeling. In this paper the authors pay a group of people to create crowd behaviors and then they use the video of the crowd to learn a model. The model they learn is a large set of state-action trajectories that represent what action an agent should take given its current state. To simulate a crowd using their model, agents are randomly positioned initially and then each agent decides what to do by finding the state-action pair with the most similar state, and then taking the associated action. They provide convincing visual evidence that it works. Their goal is prediction, so it is not an issue that the database of state-action pairs provides no insight into the underlying reasons behind peoples' movements.

Like the proposed framework, data-driven modeling is largely automatic and has less bias than knowledge driven modeling. However, data-driven models are incomprehensible and do not provide insight into how systems works. In contrast the proposed framework results in comprehensible models.

2.5 Hybrid Mobile Agent Modeling

Hybrid mobile agent models, also sometimes called multi-level models, combine data and knowledge-driven modeling techniques. A knowledge driven layer defines the microscopic interactions between agents; for example, how they avoid collisions with one another and obstacles. A data-driven layer defines the macroscopic behavior of agents such as a goal they are heading towards. Some representative hybrid mobile agent models are discussed next.

The model presented in [15] is typical of hybrid mobile agent models. In this model each agent has collision avoidance behaviors, which are handcrafted. The macroscopic layer handles

goal selection and the overall path of each agent. The macroscopic layer was learned using trajectories extracted from video. The microscopic collision avoidance behaviors they describe seem reasonable, but they are still a source of modeler bias.

Many hybrid mobile agent models use navigation fields. Navigation fields are a common trajectory control technique where the 2D environment is overlaid with a virtual grid where each grid position has an associated vector which influences agents there. A trajectory field controls the flow of agents at a macroscopic level. A trajectory field's influence is combined with handcrafted microscopic collision avoidance behaviors to determine agent behavior. In [16] and [17] the authors extract navigation fields from video of a crowd. The disadvantage of this technique is that since the underlying reasoning behind the trajectory fields is not learned a new trajectory field must be created for each environment. If the environment does not exist yet, like building plans, then a trajectory must be handcrafted and there is no way to validate a handcrafted trajectory field without real video from the location. Furthermore, like all hybrid models the knowledge-driven microscopic layer introduces modeler bias.

When hybrid models are presented the authors frequently emphasize the novelty of the macroscopic behavior and mention the microscopic behavior as an afterthought that is only needed because they are using an agent based model. However, the knowledge-driven microscopic behaviors deserve more scrutiny because they introduce modeler bias into an otherwise data-driven method. Furthermore, the data-driven layer doesn't provide any knowledge about the system. Hybrid models are not an alternative to the proposed framework, because the data-driven macroscopic layer isn't comprehensible and the knowledge driven layer is biased like all handcrafted knowledge driven models.

3 SIMULATION MODELING FRAMEWORK

3.1 From Knowledge-Driven Simulation Modeling to Data-Driven Simulation Modeling

To support the automated discovery of models a formal model structure is needed. This chapter provides a specification of the model space including agents and the world. This chapter also documents the model search and analysis process.

The existing knowledge-driven modeling method is shown in figure 3-1 below. In this modeling process system knowledge and the desired behavior are given to the modeler who then handcrafts a model. The handcrafted candidate model is then simulated and its behavior compared to the desired behavior. If the discrepancy is large then the model is refined through an additional modeling cycle. The system knowledge is often a qualitative description of the desired behavior subject to human bias. The ad hoc modeling process is a skill that can be learned (often in graduate school), but also incorporates modeler bias. All types of modeling, and in fact almost all computer science, require some element of human judgment, but it should be minimized as much as possible. Ultimately a human must decide what the meaning of computer output is; however, if the output is reduced to a few easily interpreted quantitative metrics this process is easier. The framework significantly improves upon this ad-hoc process by reducing the amount of human judgment (bias) required to generate models.

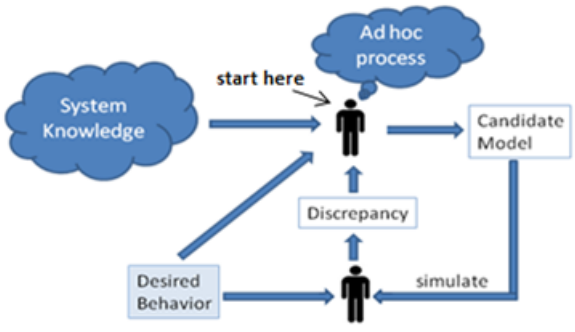

Figure 3-1 Knowledge-driven simulation modeling process

The framework uses the iterative modeling process, shown in figure 3-2 below. This process assumes that a model with the desired behavior exists in the model space; if this is not the case, then the definition of the model space itself becomes part of the iteration process. In creating the model space specification the modeler should ere on the side of caution and make it bigger than needed if in doubt. The first step is to decide what desired behavior is to be modeled (line formation for example). The desired behavior is formalized into one or multiple desired behavior functions. If multiple functions are used to define the desired behavior, than care must be taken when combining them into the fitness function; see *3.3.4 Composite Fitness Functions*. The "Search Algorithm" step takes the model space and the fitness as input and returns a model which is fit; this search can take several hours on a typical PC for a complex model. The resulting model often has "junk behaviors," like junk DNA, which don't have a significant impact on behavior, these are removed through simplification to create a candidate model. The framework is unique in its inclusion of a simplification step, because in typical data driven modeling no simplification is needed to make models easier to read (because they won't be examined).

The candidate model's behavior is qualitatively compared to the qualitative defined desired behavior and the modeler decides whether the discrepancy is big enough to justify another round of modeling. It's important to note that the qualitative analysis of behavior can never completely be avoided, because all quantitative model results must inevitably be qualitatively judged on whether they are good enough. A candidate model can be "bad" for three possible reasons: 1) desired behavior function doesn't capture the desired behavior, 2) the model space doesn't contain a good model, or 3) the search algorithm wasn't run long enough. Figure 3

assumes that enough time is given to search and the model space is well defined, so that leaves the case where the desired behavior function needs to be improved.

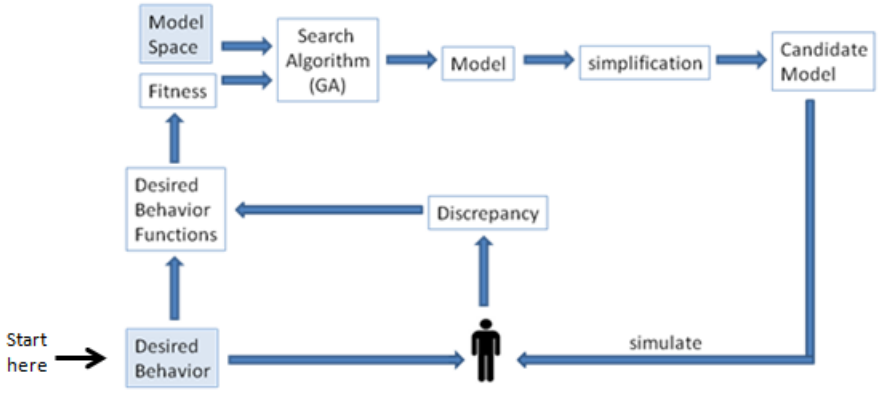

Figure 3-2 The framework modeling process

The framework currently uses time-series data from simulations to evaluate models; however, the framework could be extended to include observations from a real system directly as shown in figure 3-3 below. Notice how the extension is able to re-use the search algorithm, model space, and simplification method. The system to model is the real system; a crowd of people for example. The historical data is observations of the real system. For example, the historical data of a crowd would be the trajectories of people in that crowd that are extracted from video of the crowd. Lots of progress has been made towards extracting pedestrian trajectories from video [11]. The behavior feature functions are functions that are very similar in purpose and form to the desired behavior functions. A behavior feature function measures some aspect of a systems behavior. For example, it might measure the density of a crowd. The output of the behavior feature function is a feature; the specific density of a crowd for example. Features are then fed into the rest of the framework. When the search algorithm wants to evaluate the fitness of a model it applies the behavior feature functions to that model and then compares the resulting values to the feature values of the real system. For example, does a model

have a density similar to the real system. Like the existing modeling process the extended modeling process has a feedback step where the system to model is compared to the candidate model. If there is a discrepancy, then behavior feature functions can be added or modified. For example, the exact mathematical way that density is measured can be tweaked. A modeler might have to decide between measuring average density, minimum density, maximum density, or some other variation. The specifics of a behavior feature functions or desired behavior functions are significant.

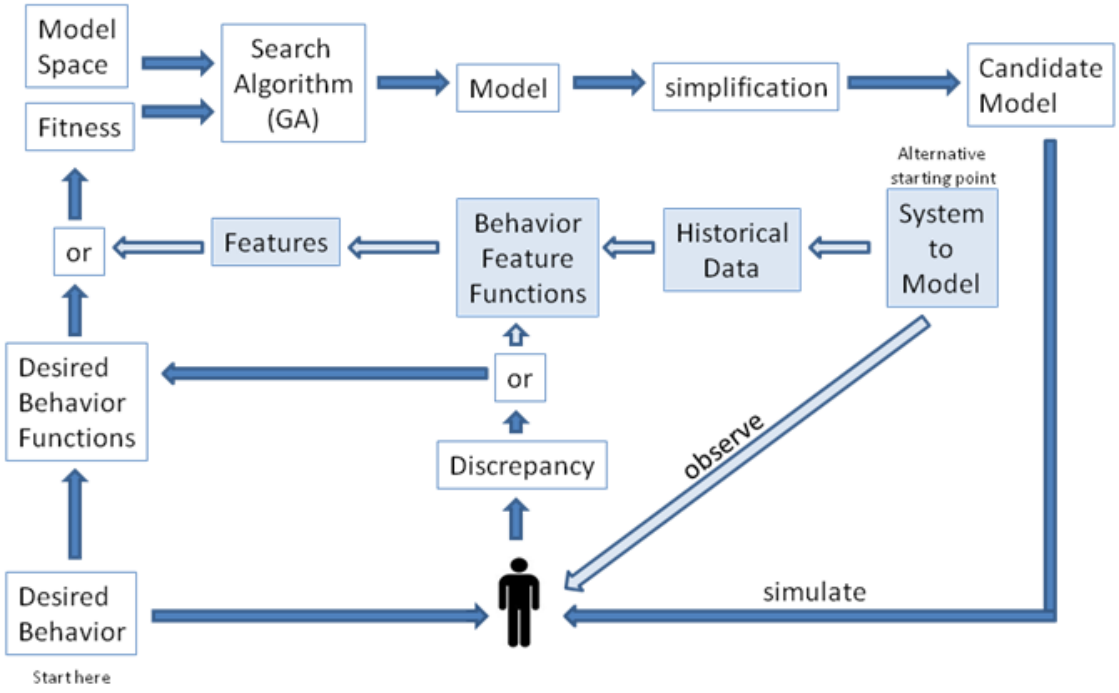

Figure 3-3 Extending the framework modeling process

3.2 Framework Overview

The simulation modeling framework has three major components: a model space specification, a search algorithm, and a set of measurement metrics. The model space specification provides a formal specification for the general model structure from which various models can be generated. It must be flexible enough to represent a wide variety of possible agent

behaviors. Furthermore, it must capture agent behavior in a way that is comprehensible and human readable. This latter requirement is important because in order for the model space to be controllable it must be possible for a person to modify a model by hand and know the impact this will have on behavior. The search algorithm is used to efficiently search the model space for candidate models that exhibit desired behavior. It must be able to accommodate a variety of objective functions specifying the search criteria and to discover robust models according to the criteria. Finally, the framework evaluation metrics are used to evaluate the degree to which the framework is flexible, robust, comprehensible, controllable, and composable. These measurement metrics can be used to evaluate an application of the framework to a different domain or a future extension to the model space specification.

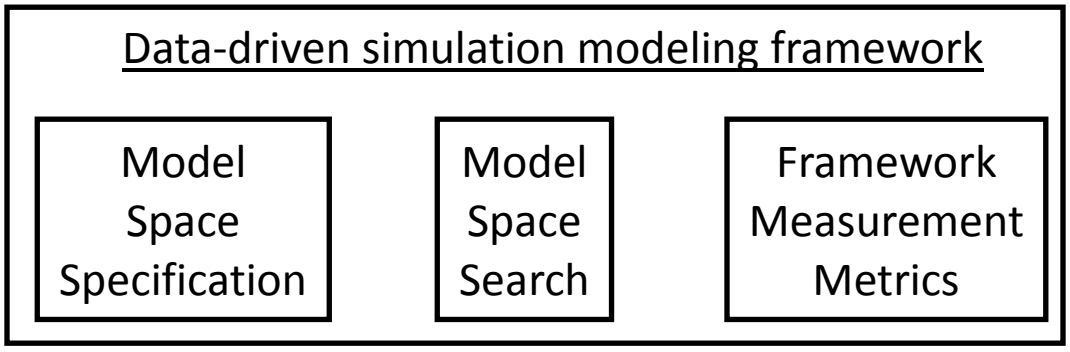

Figure 3-4 Data-driven simulation modeling framework components

This chapter is organized as follows. First the model space specification is described. Second the model space search method is described. The model space search method section describes the search process and defines the specific fitness functions used in this chapter and the next chapter. After that the framework evaluation metrics (flexibility, robustness, comprehensibility, controllability, and composability) are described in detail. The final section presents experimental results.

3.3 Model Space Specification

To support the automated discovery of models a formal model structure is needed. This section provides a specification of the model space including agents and the world. Each agent corresponds to a real world mobile agent such as a person, car, or animal. The framework is designed to represent mobile agents in the abstract, but it can be extended through arbitrary properties, discussed in detail later, to represent specific kinds of mobile agents such as people. Each agent takes up a 2D circular space and the world wraps vertically and horizontally. It is also possible for the environment to bounded by walls and in these scenarios world wrapping is turned off. Deciding whether a model should wrap or not is consequential because agents' interactions with walls require learning additional behaviors. Circular impermeable obstacles can be added to the environment to test the ability of the framework to learn obstacle avoidance behaviors. In all experiments in this chapter the positions and headings of agents are randomly set at the start of each simulation and their behavioral rules cause them to organize into collective patterns. In the lane formation experiments in the next chapter, the agents' positions are randomized but not the heading.

Formally**,** a model, **m**, is composed of a set of entities, **E**, and a description of the world, **w**. In general, each entity has a set of properties, whose values can either be fixed or change dynamically. The entities are: agents, **A**, obstacles, **O**, and zones, **Z**. Agent entities are able to dynamically update their angle (also called "heading"), speed, and position properties over time, i.e., they can move in 2D space. Each agent has a set of behaviors that update the angle and speed properties. The position property is updated each timestep using the previous position, heading, and speed as input. Obstacle entities are static entities which act as obstacles in the space. Zones are Rectangular areas that can either be permeable or not. Permeable zones

represent areas that agents can inhabit. Impermeable zones represent barriers such as walls. Agent entities can be extended to represent any entities that exhibit certain dynamic behavior or whose properties can be dynamically updated. For example, it may model an energy source where the amount of energy dynamically changes over time. Nevertheless, the focus of this dissertation is on mobile agent movement patterns and thus only the position, angle, and speed properties are used. The formal model space definition is given below.

m* = <*E*, *w*>;** *Model* ***m *is composed of a set of entities,* ***E****, and the world,* ***w****.*
w* = <*width*, *height*>, where the world is a 2-D space that wraps vertically and horizontally*
E* = {*A*, *O*, *Z*}, where:*
A *is the set of agents*
O *is the set of obstacles*
Z *is the set of zones*
o* ϵ *O*, where:*
o* = <*x*, *y*, *r*>, where:*
x *and* ***y*** *are the coordinates of the obstacle's center*
r *is the radius of the obstacle*
z* ϵ *Z*, where:*
z* = <*x*, *y*, *w*, *h*>, where:*
x *and* ***y*** *are the coordinates of the zone's lower left corner*
w *and* ***h*** *are the width and height of the zone, which is rectangular*

3.3.1 Agents

Each agent has their own set of properties and adopts a behavior-based structure to control how those properties can be changed based on their current value and observations of the environment and other agents' properties. Although a modeler can design agents to have the same set of behaviors and properties, as in the models in the results section of this chapter, this is not forced by the framework. The framework is flexible enough to handle whatever agent properties the modeler decides to include.

Each agent, **a**, has its own location, $\mathbf{l_a}$ (also referred to as "position"), heading, $\mathbf{h_a}$ (also referred to as "angle"), and speed, $\mathbf{s_a}$. The heading is the direction an agent is facing and moving. As an agent moves during a simulation its location, heading, and speed change. The location,

heading, and speed of each agent are randomly initialized at the start of a simulation. The movement vector is intentionally separated into heading and speed to allow each one to be controlled by behavioral rules separately. Each agent has a field of view, $\mathbf{fov}_{\mathbf{a}}$, defined by an angle, $\mathbf{angle}_{\mathbf{a}}$, and a view distance, $\mathbf{viewDistance}_{\mathbf{a}}$, which determines what it sees (figure 3-5 below). Entities can optionally be set as globally visible, which means that agents can see them regardless of distance. The goal zones in the lane formation experiments in chapter 5 are globally visible. For modeling flexibility each agent can have its own fov, but in the experiments each agent has the same **fov**. This flexibility is important for modeling groups of heterogeneous agents; for example, a taller person can see further in a crowd than a shorter person. An agent can perceive all other entities (agents, obstacles, or zones), i.e., know their property values, within its field of view. Each agent has a set of arbitrary modeler defined properties, $\mathbf{P}_{\mathbf{a}}$. For example, a model simulating how children follow a teacher on a field trip could have a "type" property to define agents as either being a student or the teacher. Agents have a set of behavior groups, **BG**. Each behavior group, **bg**, contains one or more behaviors that modify one agent property. No two behavior groups can modify the same property. All behaviors modifying the same property belong to the same behavior group. Different agents can have different behavior groups due to the fact that their properties are different. In the current implementation of the framework there is a behavior group for the angle and speed properties. The next sub-section explains the behavior groups in more detail. During a simulation, each agent uses its own observations and state at timestep t to determine through behavioral rules what its new speed and heading (and/or any other properties) will be at time t + 1. The position at time t + 1 is projected from the position at time t using the heading and speed at time t + 1. The formal definition of agents is given below.

$a \in A$
$a = \langle h_a, s_a, l_a, fov_a, P_a, BG_a \rangle$, *where:*
h_a = *heading of agent*
s_a = *speed of agent*
$l_a = \langle x_a, y_a \rangle$; *is the location of the agent*
$fov_a = \langle angle_a, viewDistance_a \rangle$; *is the field of view of the agent*
P_a = *set of arbitrary modeler defined properties*
BG_a = *set of behavior groups, where* $bg \in BG_a$

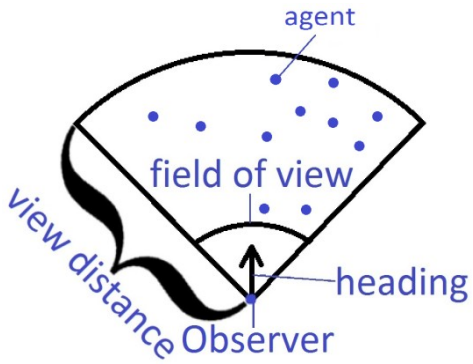

Figure 3-5 An agent's field of view determines what entities (agents, obstacles, and zones) it can see

3.3.2 Behavior Groups

An agent can have multiple behavior groups where each behavior group, **bg**, contains behaviors that modify one agent property. Behavior groups give the model space specification a general way of modifying agent properties. No two behavior groups can modify the same agent property, and all behaviors modifying the same property belong to the same behavior group. Behavior groups can modify an agent's heading, speed, and arbitrary modeler defined properties. A behavior group is composed of the name of the property a behavior group modifies, $\mathbf{p}_{\mathbf{bg}}$, a set of behaviors, $\mathbf{B}_{\mathbf{bg}}$, and a set of weights, $\mathbf{W}_{\mathbf{bg}}$, for those behaviors. For the experiments in chapter 4, one behavior group is used for angle and another for speed. If other agent properties were modified, then each of these would also belong to an associated behavior group. If an agent's behavior group is empty for a particular property than the default property value is used. In the current framework, the default heading is the agent's current heading. The default speed is the

max speed, which was chosen instead of the current speed in order to bias agents towards movement. The default speed behavior will become significant in the results section when it plays a critical role in establishing personal space between agents.

$bg = \langle p_{bg}, B_{bg}, W_{bg} \rangle$; *is a specific behavior group, where:*
p_{bg} *is the property the behavior group modifies, where:*
no two behavior groups can modify the same property
p_{bg} *refers to* $\epsilon \{h_a, s_a\} \cup P_a$
B_{bg} *is the set of behaviors, where* $b \epsilon B_{bg}$
W_{bg} *is the set of weights, where:*
$w \epsilon W_{bg}$
$w \geq 0$
$|W_{bg}| = |B_{bg}|$
$\sum(W_{bg}) = 1$

A behavior, **b**, in a behavior group, **bg**, corresponds to one influence on one property of an agent. In the real world it might correspond to a behavior like; avoid hitting another pedestrian. The multiple behaviors belonging to the same behavior group will have a compound influence on the property the behaviors influence. In the current framework, this compound influence is based on a weighted average calculation. As described above, each behavior has an associated weight. A behavior group calculates the weighted average of these competing influences from all behaviors belonging to the behavior group to decide the compound influence. A behavior can be activated or not (more on this later), if no behaviors in a behavior group are activated, then the corresponding default behavior is chosen.

3.3.3 Behaviors

A behavior is a general way of describing what entities in the environment are used to make a decision (filtering), what property of the relevant entities will be used (extract), and how to act given the property values of the relevant entities (combine and offset). A behavior contains: the category of entities it considers, $\mathbf{C_b}$, the set of types within that category it considers, $\mathbf{T_b}$, a set of flexible filters, $\mathbf{F_b}$, a property extraction function, $\mathbf{Extract_b}$, a property

combination function, $\mathbf{Combine_b}$, and an offset to add to the result of the combination, $\mathbf{Offset_b}$. An example of applying a behavior is illustrated in figure 4 below. After observed entities have been filtered by category and type the general filters, $\mathbf{F_b}$, decide what entities a behavior will use. Notice in the equations below that filters can be "chained." For example, select all agents within a distance of 10 with a speed slower than 2. Chained filters are flexible while still remaining comprehensible. The property extraction step decides what property of the used entities will be used to make a decision; any property can be used. The combination step combines the extracted properties, so that one decision can be made from multiple observations. The combination methods are shown in table 3-1; note that currently angle, speed, and position are the only extractable properties currently used; however, the framework could be extended to include arbitrary properties. The offset step offsets the combined property value to allow for more nuanced behaviors. Offsets can be negative or positive. Positive and negative angle offsets refer to different turning directions. Frequently the sign of the angle offset is not significant because the modeler does not care whether agents prefer turning left or right. The sign of the speed offset is always significant because it either slows down or speeds up an agent. The formal definition of a behavior is given below:

$\boldsymbol{b = <C_b, T_b, F_b, Extract_b, Combine_b, Offset_b>}$*, is a specific behavior where:*

$\boldsymbol{C_b}$ *is the category of entity the behavior accepts; it is either: agents, obstacles, or zones*

$\boldsymbol{T_b}$ *is a set of types of entity, within a category*

$\boldsymbol{F_b}$ *is the composition of filter functions, where:*

$\boldsymbol{F_b(E_{sense}) = (f_{p,r,1} \circ f_{p,r,2} \circ \ldots \circ f_{p,r,n})(E_{sense})}$*, where:*

$\boldsymbol{E_{sense} \subseteq E}$*, where* $\boldsymbol{E_{sense}}$ *is the set of entities that are sensed by an agent*

$\boldsymbol{f_{p,r,i}}$ *is the i'th filter, where:*

$\boldsymbol{p}$ *is the value of the filter property being used for filtering*

$\boldsymbol{r}$ *is the range that* $\boldsymbol{p}$ *must be within for the entity to pass the filter*

$\boldsymbol{Extract_b}$ *is the property extraction function*

$\boldsymbol{Combine_b}$ *is the property combination function*

offset$_b$ *is the offset; it can be positive or negative*

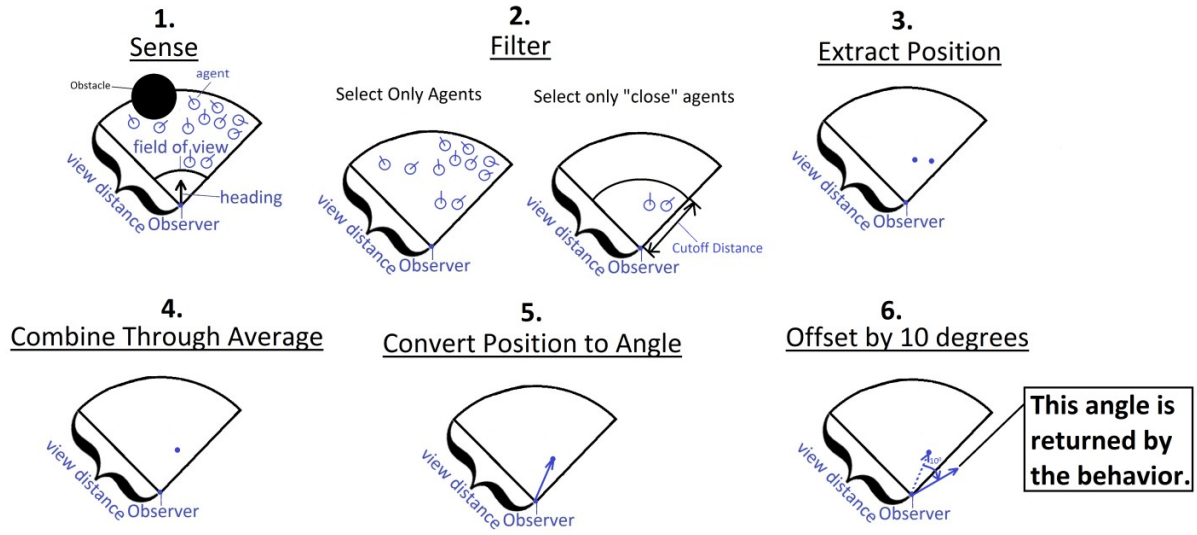

Figure 3-6 Example of behavior that turns the observing agent 10 degrees away from nearby agents

Table 3.1 Property combination functions for: position, angle, and speed

	Absolute combination	**Relative (to observer) combination**
angle	average, random	Closest (most similar to observer's heading), farthest
speed	average, random, fastest, slowest,	closest (most similar to observer's speed), farthest (most dissimilar speed)
position	average, random	farthest, closest

In table 3-2, below, for each entity category the possible types, extractable properties, and filter properties are listed. The table lists the values used in the experiments, but it should stressed that this is a general framework, so it is easy to add additional types, arbitrary properties, and additional filters if desired. For example, the model space could be extended to include types for "youngChild" and "parent" to build models where a young child is following their parent around. In a behavior, entities can be filtered to be of only one type or a behavior could allow

any type from a specific category. In all of the experiments, except the lane formation (in a hallway) experiments in chapter 5, the agents are all of one type. In the lane formation experiments the agents are either the left group or right group type depending on which side of the hallway they start on. Obstacles only have 1 type. Zone types (only used in the hallway lane formation experiments in chapter 5), can refer to the left start zone, right start zone, wall, or hallway. The extractable properties are properties that an observing agent can see in an observed entity. Note that the collision point extractable property of the agent and zone entity categories might not exist (be null) in which case the entity is filtered out. The filters measure some relationship between the observing agent (observer) and the observed entity (called "agent" when the entity category is agent). The observed entity is the entity that is trying to pass the filter.

Most of the filters are self-explanatory, but a few require some explanation. Filter property (2), for the agent entity category, refers to the degree to which an agent is heading towards the observer; it returns 0, when the agent is heading directly towards the observer. Filter property (3), for the agent entity category, refers to the degree to which the observer is heading towards the agent. Filter property (4), for the agent entity category, refers to the observer's heading line, which is a line parallel to the observer's heading and intersects with the observers position. This filter property measures how close an agent is to this line. Imagine that two agents are heading towards one another (same headings but rotated 180°), but slightly off center; this filter property measures how close they will be when they pass one another. It's a measure of how close agents are to colliding and is used in agent avoidance behaviors.

Table 3.2 Key framework variables: entity categories/types, extractable properties, and filter properties

Entity Category	Types within category	Extractable property(s)	Filter property
agent	regular, left group, right group	position, heading, speed, angle from observer to this agent, collision point with observing agent (can be null)	(0) distance to observer, (1) difference between agent and observer's heading,(2) difference between agent's angle to observer and agent's heading, (3) difference between observer's angle to agent and observer's heading, (4) agent's perpendicular distance from observer's heading line
obstacle	regular	position	(0) distance to obstacle edge from observer's center, (1) difference between observer's heading and angle from observer to obstacle's center, (2) perpendicular distance from observer's heading line
zone	leftStartGoal, hallway, rightStartGoal, wall	Collision point (can be null), closest point	(0) distance to closest point, (1) difference between observer's heading and angle to closest point, (2) distance to collision point

3.3.4 Simulation of a Model

In each iteration of the discrete time simulation, each agent goes through a sense-think-act cycle for each of its behavior groups. Discrete time modeling and the sense-think-act cycle are widely accepted modeling techniques [13]. Figure 3-7 illustrates how the sense-think-act cycle works for the behavior group that modifies an agent's angle. In the sense step an agent observes all entities in its field of view and view distance (see figure 3-6 above). In the think step an agent applies behavioral rules using its own state and the state of the entities it observes as input. The output of the think step is the new desired value for an agent's heading and speed. In the act step agents try to achieve the desired heading and speed subject to turning speed and acceleration limitations. This can be used to represent human, animal, or vehicle movement limitations. The formal simulation procedure is presented after figure 3-7.

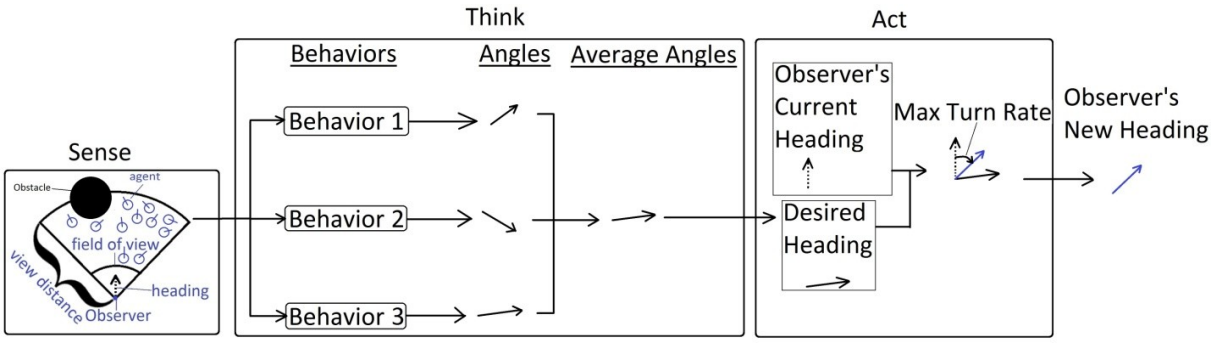

Figure 3-7 Sense-think-act cycle controlling an agent's heading

Simulation of one discrete timestep in a model with one behavior group:

Sense $_{l,fov,h}$ ***: E → E*** $_{sense}$, *where every element of* ***E*** $_{sense}$ *is within the field of view,* ***fov****, of the agent located at* ***l*** *with a heading* ***h***

F $_{b}$***: E*** $_{sense}$ ***→ E*** $_{filtered}$, *where* ***E*** $_{filtered}$ *is the set of entities that pass all filters*

Extract $_{b}$***: E*** $_{filtered}$ ***→ Properties****, where* ***Properties*** *is the set of property values extracted from the entities in* ***E*** $_{filtered}$

Combine $_{b}$***: Properties → property****, where* ***property*** *is the result of combining the set of property values,* ***Properties****, into one value*

b $_{desired\_state}$ ***= property + offset*** $_{b}$

bg $_{desired\_state}$ ***= ApplyBehaviorGroups: a*** $_{t}$ ***→ a*** $_{t,stateUpdated}$*; for each behavior group's agent property get as close to the desired state as possible subject to physical limitations (like max turning rate).*

Project: a $_{t,stateUpdated}$ ***→ a*** $_{t+1}$*; project an agent's new position using its previous position and its current heading and speed*

3.4 Model Space Search

The model specification space defined previously is only useful if it can be effectively searched through for models that exhibit desired behaviors. In the subsections below the search method and fitness functions used are described. The method whereby discovered models are simplified to remove extraneous model components is also described.

3.4.1 Genetic Algorithm

A Genetic Algorithm (GA) is used to search through the previously described model specification space. GAs are a popular search method inspired by biological evolution [14]. First a random set of models is created. Each model in this *population* of models is called a *solution* regardless of the behavior it creates when simulated. In the implementation, each solution is represented by an instance of a Java class which represents the general form of a model.

During each GA step the population evolves and becomes more fit. First all models in the population are simulated with random initial conditions. Then each model's behavior is evaluated with a *fitness function* to determine how close it is to the desired behavior. A fitness function is a formal quantitative definition of the behavior being sought. Translating a qualitative description of the desired behavior into a specific fitness function is frequently non-trivial. The next set of steps will create the new population in the following way: 1/4 of the new population comes from mutating the best 1/4 from the old population, which allows a good solution to become even better through mutation; 1/4 of the new population comes from keeping the best 1/4 of the old population unchanged, which ensures a good model isn't lost to a bad mutation; 1/4 of the new population comes from randomly exchanging behaviors between models, *crossover*. A model might have good behaviors mixed in with bad behaviors and the goal of crossover is to create new models out of those good behaviors. The final 1/4 of the population is a set of completely randomly generated models whose purpose is to inject new behaviors into the population. This process continues until the best model in the population passes a fitness threshold, or the computational budget is spent, at which point the GA stops and records the best model as the final solution.

A slight variation on the previously described GA is used in the search process. The framework adds the concept of a GA "cycle." Each cycle has a number of generations (75 in this dissertation's results). Each cycle does a GA search for model of a certain size. The first cycle has models with 1 behavior, the second cycle has models with 2 behaviors and so on until the max number of behaviors have been searched. The random starting models for the population of the next cycle is based on the previous cycles population, but with 1 extra behavior added. The idea is that the most important behaviors will be learned first and then additional less important (but still desired) behaviors can be learned later. As an example, in a model with a maximum of 3 angle behaviors there would be 3 cycles, where the first cycle has 1 behavior per model and the last cycle has 3 behaviors per model. The best model from this process is the model with the highest fitness drawn from the population at the end of each cycle. It is possible that the best model comes from a cycle that isn't the last. For example, if there are 3 cycles, then the best model might come from the 2nd cycle and only have 2 behaviors.

Once a model is discovered it is polished before moving onto the simplification step. The GA discovers a good model and the polishing step tries to improve on this model by trying slight variants. Polishing is a local search process whereby the fitness of random mutations of the discovered model are tested and the fittest mutated model is chosen. The models in the result section were polished by testing 1,000 random mutations of the discovered model. Furthermore, the fitness was averaged over 10 simulations, which is more than the 3 used in the GA search, because model polishing needs to be able to distinguish slight differences in model fitness.

3.4.2 Simplification

The models that are returned from the genetic algorithm often have lots of "junk behaviors" (like junk DNA); behaviors that don't have a measureable impact on the desired

model behavior. One goal of the framework is to discover comprehensible models, so junk behaviors must be removed through a process called "simplification". The goal of simplification is to find the least complex submodel that exhibits the same emergent behavior (being measured) as the un-simplified model. A submodel has a subset of the un-simplified model's behaviors and filters. A model's complexity is equal to the sum of the total number of behaviors and filters it has. It is possible, but in practice unlikely, that simplifying a model results in multiple distinct submodels of equal complexity. To improve efficiency, the simplest submodels are checked first and once a good submodel is found more complex submodels are ignored.

A submodel's behavior is compared to the un-simplified model's behavior using a fitness function. All fitness metrics measure the distance from some ideal behavior; hence lower fitness function values are fitter. A submodel is considered fit if its deviance from the ideal behavior is less than or equal to the deviance of the un-simplified model. There can be significant variation (caused by different random initializations) in the measured fitness of a model across simulations, even if the model behavior is good in all the simulations. To overcome this randomness the fitness threshold is determined by averaging the unsimplified model over 10 simulations and then doing that 10 times and taking the largest average as the threshold (the worst average fitness). Then a submodel's fitness is averaged over 10 simulations and is considered fit if its average is under the fitness threshold.

3.4.3 Fitness functions

Each fitness function (also called: "fitness metric", or simply "metric") measures how close the emergent behavior of agents implementing some behavioral model is to an ideal behavior. Fitness functions are used by the genetic algorithm to evaluate the fitness of models. The framework uses the convention for the value returned by fitness functions that lower values

are considered fitter because they represent less deviation from the ideal behavior. A fitness function could theoretically return 0, which would indicate idealized behavior; however, this is extremely unlikely. A model is considered "fit" if the value returned by its fitness functions is below a fitness threshold set by the modeler. A fit model may not exhibit the desired behavior if the fitness thresholds are too high, or as is often the case, the fitness function does not accurately quantitatively measure the qualitative behavior being looked for.

A fitness function is designed to measure only one aspect of behavior; however, frequently the behavior being searched for is a composition of multiple behaviors. For example, the snake behavior with obstacle avoidance (see figure 4-7 in chapter 4) can be represented by the composition of the following fitness metrics: alignment, snake shape, group, negative hollow circle, and touches obstacle. Frequently a model is discovered which was fit, but which did not exhibit the desired behavior; in these cases some behavior was not being represented by the existing fitness metrics, so an additional metric wad added. Combining fitness metrics into one fitness value is complicated by the fact that each fitness metric is in different units. This issue is addressed in the next section.

3.4.4 Composite Fitness Functions

A composite fitness function combines the results of two or more fitness functions. Each fitness function is designed to measure a very narrow behavior, so to discover models which exhibit complex behavior multiple fitness functions must be combined into a composite fitness function. Each fitness function is scaled until they are in the same "units" and then the scaled fitness values are multiplied and returned as the fitness of the model being evaluated. Multiplying scaled fitness values is better than adding them because it preserves the significance

of small numbers, which is important because it ensures that the GA always takes all fitness functions (that form the composite fitness function) into account.

The units of the values returned by different fitness functions will be different because they measure deviations from different ideal behaviors; however, they can be scaled so that similar values represent similar degrees of deviation from the ideal behaviors. To scale a raw fitness function value it is divided by a "reasonable" value for that fitness function, **scale**. Another way of interpreting this scaling factor is that it represents the point at which you will consider a model "fit". The resulting **scaledFitness** is less than or equal to 1 if it is fit. If a scaled metric is a little over 1, then it may also be fit; manual verification and judgment are needed.

A **ceiling** on **scaledFitness** is needed because not all fitness metrics have the same maximum **scaledFitness**. If not done, then one fitness metric could dominate the composite fitness score. Therefore all **scaledFitness** values are capped (the **ceiling** used is 3, which was determined to work experimentally). The ceiling is also called **smallestMaxFitness**. The scaled fitness value is given a floor of 1. If this is not done, then a model can appear fit when it has a very small value for one metric, but is bad in all other metrics. Although search constrains the minimum and maximum values of the scaled fitness this is unnecessary when a discovered (and fit) model is studied through sensitivity and robustness analysis. Carefully combining fitness metrics is needed to avoid scenarios where a model is discovered that is fit for one metric, but not the others. The equations for calculating the scaled fitness of a metric and combining the metrics into a composite fitness value are given below.

$$scaledFitness = min\,(max\,(raw\ fitness\ function\ value/scale,\ 1),\ ceiling)$$

$$compositeFitness = \prod_{i}^{metrics} scaledFitness_i$$

3.5 Framework Evaluation Metrics

An important contribution of this work is the five measurement metrics that were developed to evaluate the application of the framework to the mobile agent domain. If the framework were applied to another simulation modeling domain, then these metrics would be essential to evaluating whether it was applied correctly. The five simulation modeling goals are: flexibility, comprehensibility, controllability, compossability, and robustness. The degree to which the model space, also called "search space," can be re-used for multiple modeling problems; the number of distinct and useful models it contains, reflects its flexibility. The easier it is to understand discovered models in the model space the more comprehensible it is. A model space that is comprehensible and flexible will make modifying a discovered model to get predictable and significant changes in behavior easier; the ease of these modifications represents the degree of controllability. The easier it is to re-use discovered models or parts of models in new models, the more valuable and composable it is. Finally, discovered models must consistently (robustly) create the desired behavior to be useful. In the results section these five measurement metrics are used to evaluate the application of the framework to the mobile agent domain. These measurement metrics represent the five key ways in which the framework provides value to a simulation modeler.

3.5.1 Flexibility

A flexible framework is one that can discover many useful and interesting models. To achieve flexibility the model space specification must be capable of representing a broad variety of models. A flexible model space specification requires an effective search algorithm to take advantage of it. In our work, the flexibility of the framework is demonstrated by finding models for different behavior patterns (e.g., snaking behavior, circular behavior, lane formation, etc.).

3.5.2 Comprehensibility and Controllability

A key objective of the framework is that it results in controllable and comprehensible models. A model is comprehensible if a modeler can understand the structure and behavior of the model. In our work, this is supported because the model space specification is based on human-readable descriptions of how agents sense, think, and act in a dynamic environment. As a result, each model (including all its behaviors) discovered from the GA search method can be read and interpreted by human modelers. For example, a behavior that senses the closest obstacles and moves 180 degrees away from them can be interpreted as an avoidance behavior. A model is controllable if a modeler can determine how to modify an existing model into a related model that generates desired behavior that is similar but not identical to the unmodified model. Controllability is directly derived from comprehensibility because the latter provides a necessary condition for the former. Comprehensible models can provide valuable insights into systems, which is why modelers spend time handcrafting knowledge-driven models. Controlling comprehensible models enhances the value of simulation models by allowing them to be predictably modified. For example, a pedestrian model trained on data in one country could be adapted to another country with different personal space social rules by modifying the personal space behavior. In this work, the controllability of the models is demonstrated by modifying some of the behaviors and showing how the modified models generate predictable behavioral changes in the simulations.

3.5.3 Composability

Behaviors are composable if they can be combined to generate models with predictable behavior. Behaviors used for composition can be handcrafted or discovered previously. Composability enhances the value of the framework by encouraging the re-use of previously

discovered behaviors, which speeds up the modeling process and allows accumulated knowledge to be used. Composability means that well understood behaviors can be handcrafted quickly and combined with learned behaviors about which there is less prior knowledge. Composability is supported by comprehensibility, because to compose behaviors they need to be understood first. Furthermore, composability benefits from the modular design of behaviors that enables them to be combined and work together in well-defined manner.

3.5.4 Robustness and Sensitivity Analysis

Robustness is a measure of how well a model works in situations other than the one in which it was discovered. A robust model can be used in a variety of situations and can be effectively composed with other models. In the context of mobile agents, robustness is studied with respect to different numbers of agents, world sizes, and numbers of obstacles (when applicable). Robustness can be thought of as a measure of how applicable a model is to different scenarios. Robustness is a measure of when a model "breaks".

Sensitivity analysis is performed on discovered models to see how important the specific offset, filter ranges, and combination methods are for model behaviors. Conducting sensitivity analysis is a standard modeling procedure. Through this analysis one model can be expanded into many related models which exhibit the same behavior. Sensitivity analysis is done by modifying one component of a model and evaluating its fitness to find fit models related to the unmodified model.

In sensitivity analysis, to make the interpretation of the fitness of related models easier, the fitness of evaluated models is scaled by dividing the fitness metric value of the model being evaluated by the fitness metric value of the unmodified (and fit) model. This means that if the scaled fitness value of a modified model is around 1, than the modified model is as fit as the

original model. A scaled fitness value substantially above 1 is not fit. Furthermore, some models are evaluated with multiple fitness metrics in which case each fitness metric is scaled by the corresponding fitness metric value of the original unmodified model. Modifying a model is likely to have a varying effect on the fitness metrics used to evaluate it, which gives insight into how the model works.

3.5.4.1 Offset Sensitivity Analysis

Recall that each behavior has an offset that is applied after the combination step. Offset sensitivity analysis is performed by trying all (discretized) possible offset values for each behavior individually. Only one behavior's offset value is changed while everything else in the model is left unchanged. The results of this analysis are displayed in graphs where the x-axis refers to the offset value and the y-axis refers to the scaled fitness metric. Furthermore, the value of each individual fitness metric and the composite fitness is plotted to aid analysis.

3.5.4.2 Filter Range Sensitivity Analysis

Behaviors can, but are not required to, have filters. Filter range sensitivity analysis involves trying all (discretized) possible filter ranges. The results are displayed in a table where each element in a row has the same upper bound and each element in a column has the same lower bound. Elements in the table are scaled fitness values. In the results section, the original filter range is highlighted in yellow.

3.5.4.3 Combination Method Sensitivity Analysis

All behaviors must have a combination method. Combination method analysis entails changing the combination methods of each behavior one at a time. In many experiments the combination method doesn't have an impact on behavior (fitness), but for some it does. Some combination method must be chosen, so even if it turns out to not matter it's important to not

hardcode in one method, such as average, because that would be making the type of assumption that the framework seeks to avoid.

4 EXPERIMENTAL RESULTS AND ANALYSIS FOR DATA DRIVEN SIMULATION MODELING

In this chapter the framework will be used to discover, analyze, compose, and control a variety of models. This chapter will define the fitness functions that are used. After a model is discovered it must be polished and then simplified as described in the previous chapter. The simplified models are quite easy to interpret through manual examination, but sensitivity and robustness analysis is used to enhance our understanding of them. Once the models are well understood additional behaviors will be composed with them and their parameters will be tweaked to get variations of behavior (controllability). In the next chapter the framework will be extended and the complex lane formation model will be discovered.

4.1 Experimental Setup

The framework was tested by applying it to the mobile agent domain to discover a variety of models (flexibility) and demonstrating that the models are comprehensible, controllable, composable, and robust. All experiments have an identical setup except for their fitness function and other occasional deviations that will be mentioned in context. The default setup is described here. The model search space has a maximum of 4 angle and 1 speed behaviors. The search space must be large enough to contain a good model, but not so large that it is difficult to search through. Speed behaviors are less important and one is included to allow for agents to slow down to avoid collisions The default speed behavior is to go as fast as possible, so agents will speed up after a slow down behavior is no longer activated. It is significant from a methodology perspective, that all experiments use the same search space even when it is suspected that the fit

model has fewer than 5 behaviors, because it demonstrates the robustness of the search method. There are 25 agents, which through experience appears to be the minimum number of agents necessary to get consistent emergent behavior. More agents can be simulated; I've run the framework code with as many as 1,000, but dramatically increases the search time without an increase in quality of the models discovered. The world size is 500x500. Each GA cycle is given 75 generations and the population of models is 40. The fitness of each model is averaged over three separate simulations (different random initial conditions) to account for randomness. This means that over the course of a total GA search there are (5 cycles) * (75 generations/cycle) * (40 models tested/generation) * (3 simulations/model tested) = 45,000 model simulations. The runtime varies by machine and experiment, but is approximately 48 hours, so running and evaluating the fitness of 1 simulation takes about 4 seconds. Different experiments will take different amounts of wall clock time, but they all have the same computational budget as measured by number of simulations. Each simulation is run for 500 iterations and model burnin time is 400. The fitness metric resolution is 50, which means that each experiments' fitness is measured twice after burnin. More fitness measurements could be made, but the extra computational cost (and wall clock time) was not justified by the discovery of fitter models.

Unlike existing data-driven mobile agent modeling frameworks, model discovery is just the start of an extensive analysis, because understanding a model is just as important as whether it generates the desired behavior. The discovered model is polished, then the polished model is simplified. Unless stated otherwise all results are simplified models. It doesn't make sense to try and analyze newly discovered or polished models, because they frequently have junk behaviors that are irrelevant to the desired behavior.

An overview of the experimental design and analysis is given in table 4-1 below. The experiments can be split into two categories: 1) model discovery and 2) controllability & compossability. In the discovery experiments 3 models are discovered: snake shape, snake shape with obstacle avoidance, and hollow circle. The snake shape with obstacle avoidance uses the snake shape model discovered (ID = 1) as a starting point, so it just has to learn the obstacle avoidance behavior. After each model is discovered it is polished, then simplified. The simplified model is then analyzed using sensitivity and robustness analysis. In summary, each model discovery experiment has possibly as many as 9 steps: 1) discover model using GA, 2) polish model, 3) simplify model, 4) offset sensitivity analysis, 5) filter range sensitivity analysis (if the model has filters), 6) combination method analysis, 7) world size robustness, 8) number of agents robustness, and 9) number of obstacles robustness (if there are obstacles).

There are three controllability and compossability experiments. In the first experiment (ID = 4) the previously discovered snake shape model is modified (1 handcrafted behavior added) to preserve personal space. In the next experiment (ID = 5) the amount of personal space is controlled in the snake shape model. Finally, in the last experiment the Boids model of flocking [3] is handcrafted and this handcrafted model has the discovered obstacle avoidance behavior (from snake shape with obstacle avoidance experiment, ID = 2) added to it.

Table 4.1 Experimental design and analysis overview

Model discovery experiments				
ID	**model**	**Fitness metrics used**	**Obstacles**	**Analysis**
1	Snake shape	alignment, snake shape, group, negative hollow circle	0	Sensitivity analysis: offset, filter range, combination method Robustness: world size, number of agents
2	Snake shape with obstacle avoidance	alignment, snake shape, group, negative hollow circle, touches obstacle. (Note: The snake shape model (ID=1) was used as a starting point)	5	
3	Hollow circle	hollow circle	0	

Controllability & compossability experiments			
ID	**Input Model(s)**	**Experiment**	**Obstacles**
4	Previously discovered snake shape model (ID=1)	Add handcrafted personal space behavior	0
5	Previously created snake shape with personal space model (ID=4)	Control amount of personal space	0
6	Handcrafted Boids model & previously discovered snake shape with obstacle avoidance model (ID=2)	Add obstacle avoidance behavior, extracted from previously discovered snake shape with obstacle avoidance model (ID=2), to handcrafted Boids model	5

4.1.1 Fitness Metrics Used

The following subsections detail the fitness metrics used in the results section of this chapter and in the lane formation experiments in chapter 4. These metrics are combined through a composite fitness metric as described above, although they could be used separately if desired. A good metric measures one aspect of behavior and only that aspect. Metrics follow the convention that the lower value returned the more fit a model is (for that metric) because the less deviation from the ideal behavior there is. Another convention is that each metric described measures the fitness at one simulation timestep with the understanding that multiple timesteps after model burnin are averaged to determine fitness. The total number of timesteps averaged is: floor(((total simulation length) - (model burnin time))/resolution). For all experiments in this

chapter the resolution is 50. The lane formation experiments done in chapter 4 use a resolution of 1.

Sometimes metrics reflect behavior that we do not want to find. For example, in experiments looking for agents that follow one another in a line (snake shape, figure 4-3) it was found that sometimes models are discovered that have the agents following one another in a hollow circle pattern (figure 4-11). After deciding that this behavior was a bug and not a feature, a metric was added that would indicate that a model had formed the hollow circle pattern and that this was not desired by returning a large positive value. These are called "negative" metrics because they return high values when a bad behavior is found. Negative metrics use the fitness returned by their associated metric as follows:

$$negative\ metric = max(threshold - metric, 0)$$

In the negative hollow circle example, the metric value is the value returned by the hollow circle metric. The threshold represents the point at which a model is exhibiting behavior that we wish to avoid.

4.1.1.1 Snake Shape Metric

This metrics captures the extent to which agents are following one another in a line. For each agent, a, calculate how closely it is following its neighbor; call this metric $follow_i$, where a is the i'th agent:

1) find the closest neighboring agent, a_n
2) calculate the angle, Θ, from a to a_n
3) $follow_i$ = min(diff(Θ, h_a), diff(Θ, h_a + 180°)), where h_a is the heading of agent a

The first and second steps find the angle between a and its closest neighbor. The third step finds the difference between ***a***'s heading and the angle to its nearest neighbor. The "+ 180°"

is needed because an agent could be following its neighbor or vice versa. In the worst case $follow_i$ equals 90°. This metric returns:

$$metric = \left(\sum_{i}^{all\ agents} follow_i \right) / numberOfAgents$$

Additionally, to handle outlier situations, $follow_i$ is calculated for each agent using the closest and the second closest agent and the best value (lowest) is chosen as the value of $follow_i$. The snake shape scale is 10°.

4.1.1.2 Hollow Circle Metric

This metric is used to detect whether agents have formed a hollow circle shape. This metric doesn't precisely control the radius of the circle; it just measures whether one forms at all. If agents form a hollow circle than they should all be the same distance from center of the circle, where the center is the average of the agents' positions. The first step is to calculate the average position of all the agents, p_{avg}. Then the average distance from each agent to p_{avg} is calculated as follows: $d_{avg} = \left(\sum_i^{agents} distance(agent_i, p_{avg})\right) / numberOfAgents$. If the agents formed a perfect hollow circle, then the distance from each agent to p_{avg} would equal d_{avg}. The metric equals the extent to which agents deviate from perfection, which is calculated as follows:

$$metric = \frac{\sum_i^{agents} \left| distance(agent_i, p_{avg}) - d_{avg} \right| / numberOfAgents}{d_{avg}}$$

In the previous equation the numerator represents the average difference between d_{avg} and each agents distance to p_{avg}. The denominator scales the average by the radius of the circle; this ensures that the returned value is insensitive to the circle radius (so long as it is valid). The final point is that the radius of the circle must be within the interval [minRadius, maxRadius]. For this dissertation the interval was set such that the minimum radius implies a circle with a

circumference where all agents are just barely touching and the maximum radius implies a circle where all agents are separated by 2 other agents on average. The hollow circle scale was 0.1 (10% deviation from d_{avg}).

4.1.1.3 Negative Hollow Circle Metric

This negative metric measures the extent to which the agents *do not* form the hollow circle shape. This metric uses the value returned by the hollow circle metric described in the previous section. The negative metric threshold is 0.25, which means that as long as the hollow circle metric returns a fitness greater than this the model being tested is considered fit.

4.1.1.4 Group metric

This metrics measures the number of agents not in a group; the goal is to minimize the number of "straggler" agents. Imagine that each agent in a group is a node in a graph and each agent has an edge to every other group member, where the edge weight is equal to the distance between the two nodes it connects. A group is valid if there is a path from every agent to every other agent such that no edge has a distance greater than some threshold, maxNeighborDistance. Furthermore, groups must be of a minimum size, groupSizeThreshold. Finally, all groups must be of the maximum possible size while still remaining valid. The groups were created using a recursive function. In the experiments maxNeighborDistance equals 4 agent diameters (measured between the centers of agents; at 1 agent diameter two neighboring agents would be touching), and the groupSizeThreshold is 5 agents. This metric returns the number of agents not in a valid group at a given timestep. The group metric scale is 3, which means that if 3 or fewer agents are not in a group, then the model is still considered fit.

4.1.1.5 Alignment Metric

The alignment metric measures how closely aligned each agent is with its neighbors. Two agents are neighbors if they are within 12 agent radius of one another; this distance is fairly large, so that each agent will likely have a few neighbors to compare its heading to. Two agents are perfectly aligned if their headings are equal. For the i'th agent calculate the average difference, $diff_i$, between its heading and its neighbors' headings. This metric returns:

$$metric = \left(\sum_{i}^{all\ agents} diff_i \right) / numberOfAgents$$

It is possible that an agent has no neighbors. In this case $diff_i$ equals the alignment scale multiplied by the smallestMaxFitness. Therefore if all agents have no neighbors, then the scaled fitness will equal smallestMaxFitness; however, if only a few agents have no neighbors then the fitness will be less than smallestMaxFitness. If a worse value for $diff_i$ (larger than alignment scale multiplied by smallestMaxFitness) was used for agents with no neighbors then it would lead to a poor fitness score even if only one agent has no neighbors. The alignment metric scale is 20°, which means that two agents can have a heading different by 20° or less and be considered aligned. This value isn't 0° because agents need to adjust their heading to avoid obstacles and other agents.

4.1.1.6 Touches Obstacle Metric

This metric measures the number of agents touching an obstacle at a given timestep. An agent is considered to be touching an obstacle if: distance (agent.center, obstacle.center) <= 1.1 * (obstacle.radius + agent.radius). 1.1 is used instead of 1.0 to avoid rounding errors and because when an agent gets very close to an obstacle it is considered touching. The touches obstacle scale

is 1, which means that if 0 or 1 agents are touching an obstacle at a given timestep the model is fit for this metric (at that timestep).

4.2 Interpreting Discovered Models: Comprehensibility, Sensitivity Analysis, and Robustness

After a model is polished and simplified it is ready for analysis. The goal of analysis is to understand how a model works, so that it can be controlled through intelligent modification and composed with other models. The first step in analysis is to look at the model itself. All behaviors observe a category of entities (agents, obstacles, or zones); however, this column is omitted from these tables when all behaviors observe all other agents (within the field of view). Additionally, in these tables the combination method is omitted, because frequently the exact combination method does not impact fitness. In the combination method sensitivity analysis results the combination method used in the model originally is highlighted in yellow.

Beyond manual examination of models, sensitivity and robustness analysis both provide insights into models. Sensitivity analysis is conducted on the filter ranges (if applicable), combination method, and offset amount of behaviors. Often manual examination of a model is enough to understand how it works and the sensitivity analysis is used for confirmation. Other times sensitivity analysis can lead to new insights. The robustness of models is examined by varying the number of agents and world size. Although useful models are robust, and this is a framework goal, all models eventually fail if the number of agents or world size is too extreme (in either direction). When sensitivity or robustness results are analyzed the value of each fitness metric is included in addition to the total fitness. This makes it easy to see the impact that changing a parameter has on different fitness metrics; frequently a change will have a disproportionate impact on a subset of the fitness metrics. In the following sections the models

for snake shape, snake shape with obstacle avoidance, and hollow circle will be discovered and analyzed.

4.2.1 Snake Shape Model

The "snake shape" behavior is one where agents are closely following one another in a line as seen in figure 4-1. The snake shape behavior was learned using a composite fitness function that used the following metrics: alignment, snake shape, group, and negative hollow circle. Each of the four metrics has a role to play. The alignment metric finds models where agents are heading in the same direction; however, without the snake shape metric the agents might form a line perpendicular to the direction of movement instead of parallel to it. Without the group metric the agents might form small groups of agents. The negative hollow circle metric is needed because agents that form a big hollow circle are fit in the alignment, snake shape, and group metrics, but do not exhibit the desired behavior. Recall that each metric is allowed to deviate from 0 somewhat and still be considered fit (it depends upon the metric scale). Therefore, although a hollow circle of agents is not straight it is straight enough to be considered fit without the negative hollow circle metric.

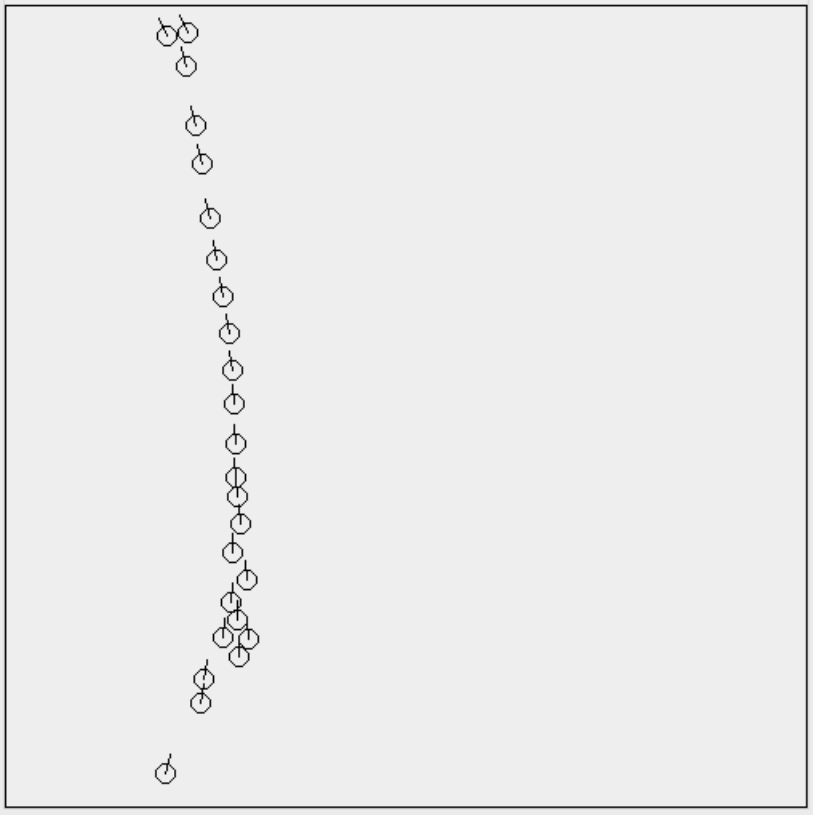

Figure 4-1 Snake shape model

This experiment demonstrates that automatically searching for a fit model forces the modeler to precisely define the behavior they are looking. If the snake model was handcrafted, then the modeling process would be to first create a model that looked qualitatively good, and then generate a fitness function after the fact to justify that it is good. For example, the snake shape metric at first glance seems like it might be sufficient to measure whether a snake shape is formed. This is flawed because there may be multiple models that are also fit according to that fitness function, but do not exhibit the desired behavior. When using the framework, if a modeler does not accurately define the behavior to be discovered, then they will discover models which are fit according to the fitness function, but look wrong. During my research this happened multiple times.

To analyze the discovered snake model given in figure 4-1 above it is first manually inspected. The offset of the first behavior is close to 0 and the property to extract is the angle to whatever agents get through filter, which suggest that this is a behavior that causes agents to be attracted to one another; cohesion. The filter causes the agents to only be attracted to agents that are in front of it, which makes sense because if agents were attracted to all agents then the result would be little balls of agents (see figure 4-2, where the model displayed has had the filter removed). The second behavior causes the heading of agents to align, because the offset is close to 0. The speed behavior seems unnecessary, but not detrimental to fitness. To test this the speed behavior was removed and the resulting model is fit as figure 4-3 below the table shows. The speed behavior was not removal during the simplification process due to the element of randomness when measuring fitness.

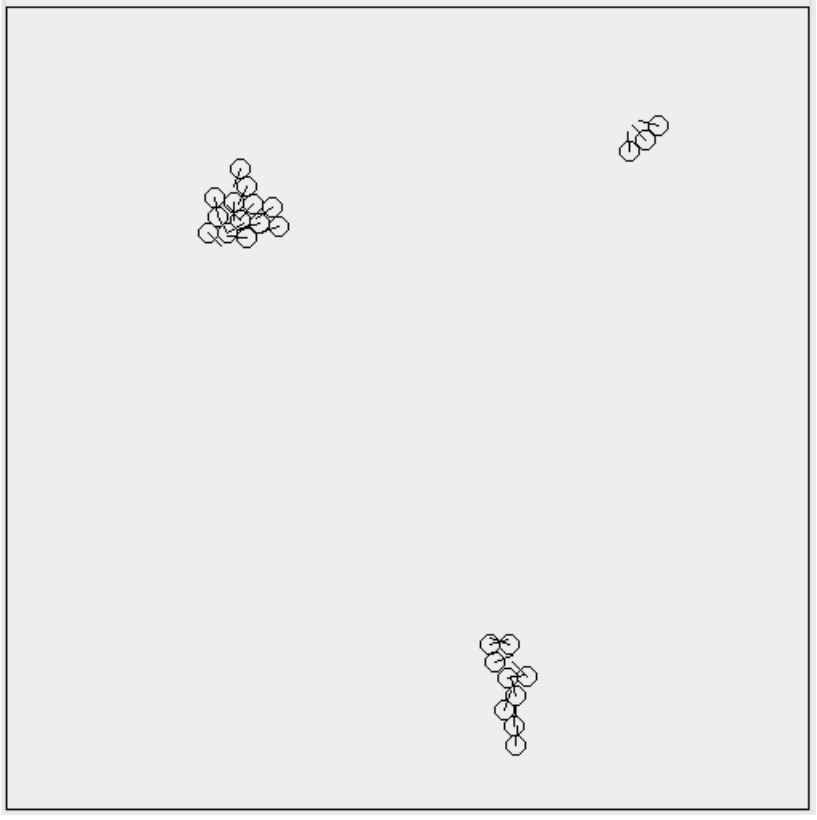

Figure 4-2 Snake shape model with behavior filter removed resulting in bad behavior

Table 4.2 Simplified snake model

Angle Behaviors				
ID	**1st Filter**	**Property to Extract**	**Offset**	**Interpreted Purpose**
1	difference between observer's angle to agent and observer's heading in [0.0, 54.0]	angle between observer and observed boid	-10.0	cohesion
2		heading	10.0	alignment
Speed Behaviors				
ID	**1st Filter**	**Property to Extract**	**Offset**	**Interpreted Purpose**
3		speed	1.5	unnecessary

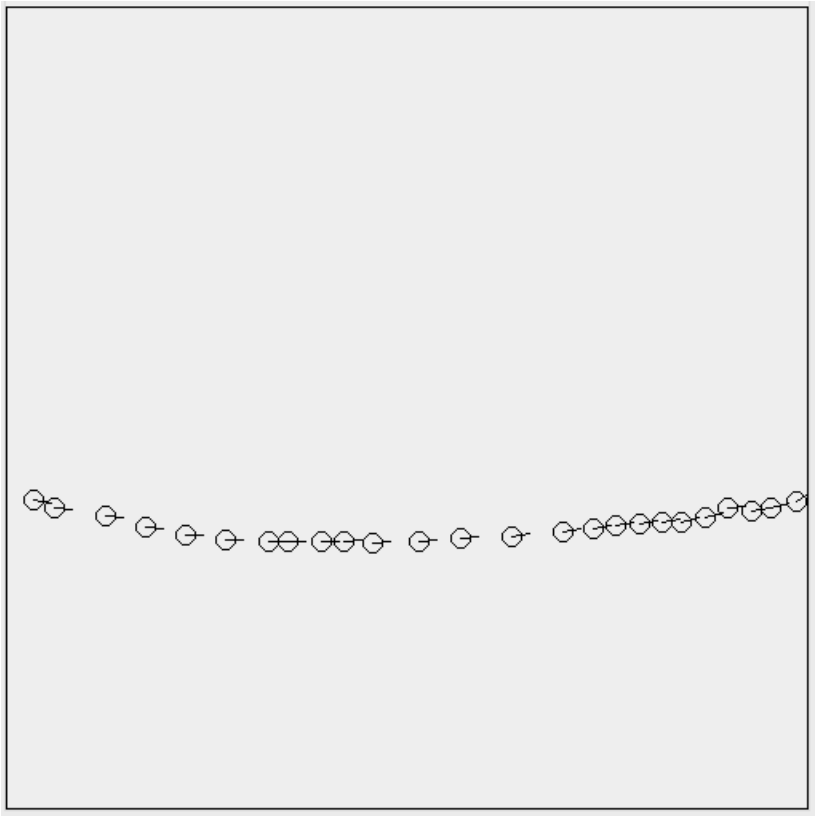

Figure 4-3 Snake model with unnecessary speed behavior removed

Now that we have theories about how the snake model works, let us test those theories by doing some sensitivity and robustness analysis. The first theory is that the offsets for the two angle behaviors can be thought of as 0 instead of their specific -/+ 10°. Figures 4-4 and 4-5 below show how changes in the value of the offset affect fitness. The first thing to note is that in both figures the negative hollow circle fitness is not effected, which means that changing the offset doesn't cause the agents to form a hollow circle. In both graphs the alignment metric is fit for a large range of offsets centered around 0°. The snake shape and group metric, however, are narrowly centered around 0°, which supports the theory that both offsets can be thought of as 0°. Although both behaviors are fit for an offset of 0°, there is a slight bias of the graph to the left for behavior 1 and to the right for behavior 2 corresponding to -10° and 10° respectively. This occurs because -10° and 10° cause the agent to turn in different directions; however, when both behaviors act at the same time they counteract one another.

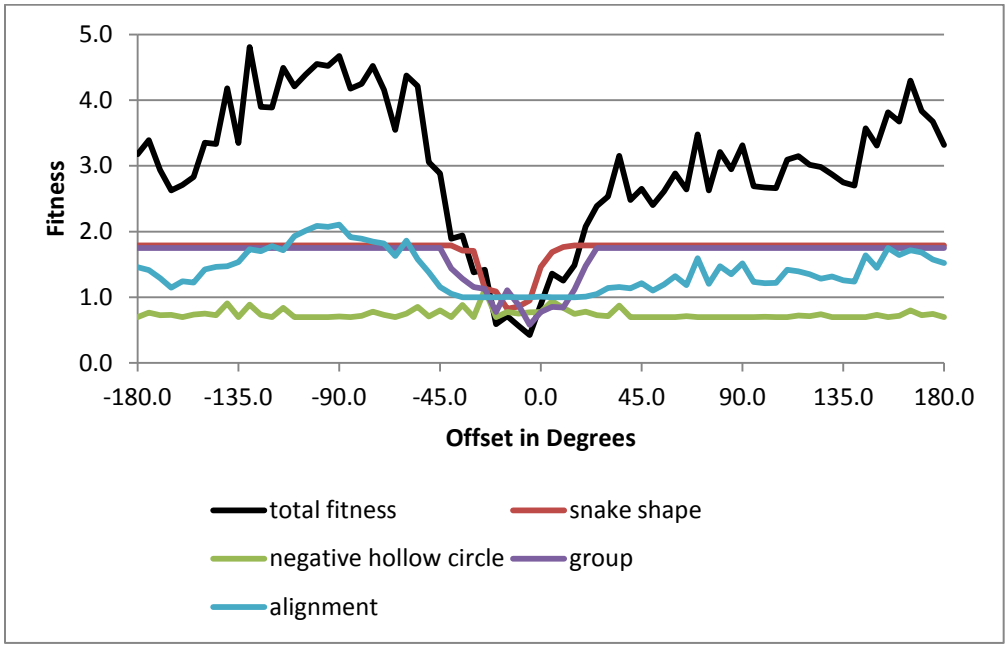

Figure 4-4 Snake shape model's 1st behavior's offset sensitivity analysis

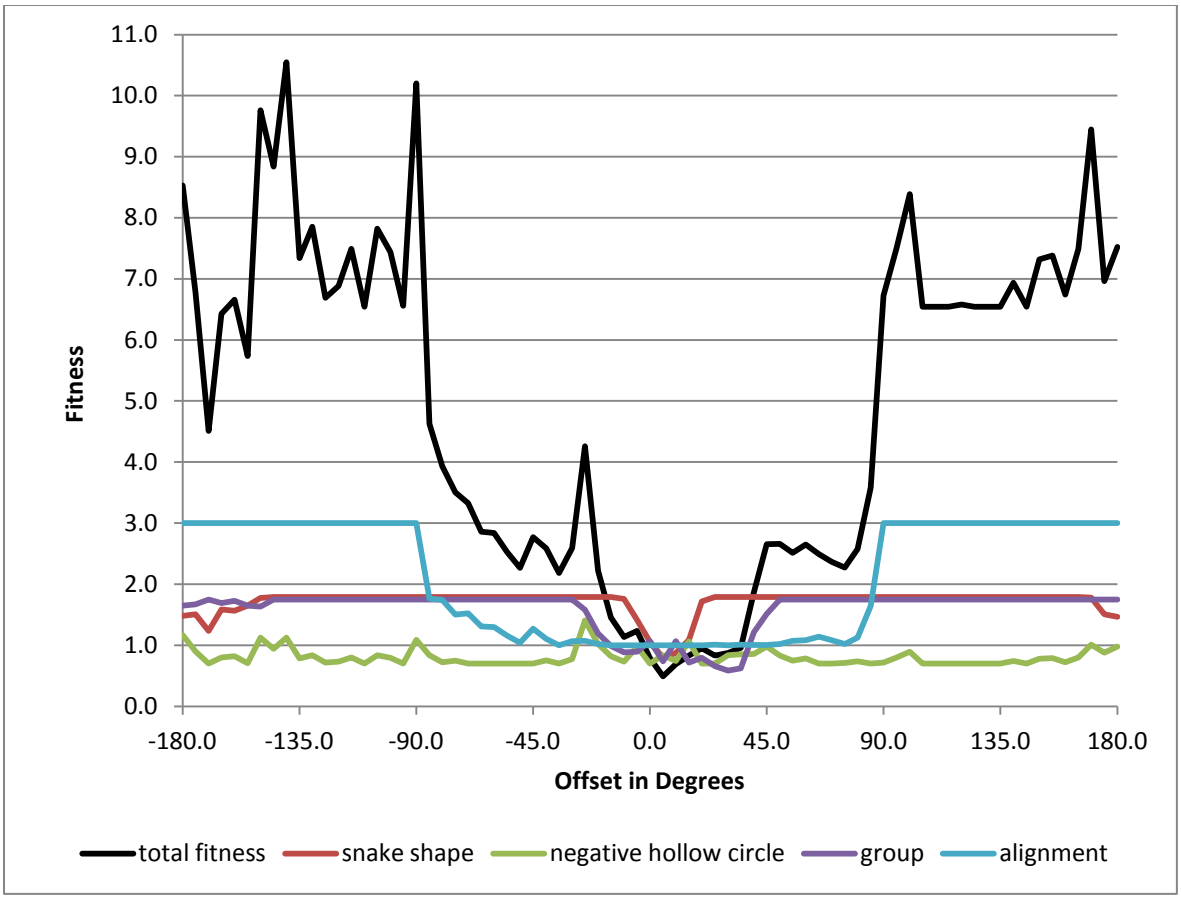

Figure 4-5 Snake shape model's 2nd behavior's offset sensitivity analysis

The next question is what significance each behavior's combination method has. The highlighted rows in table 4-3 correspond to the combination method used in the model. The speed behavior is absent from this table, because as discussed previously it was discovered not to effect behavior. The model works for most combination methods, but breaks down when the closest combination method is used for the second speed behavior. When the closest combination method is used for the second behavior each agent tries to match the heading of the agent whose headings is most similar. This means that if there is a group of agents they won't all end up going in the same direction. This causes the alignment metric to be very bad. When agents fail to align, they don't form the snake shape very well, so this causes a secondary effect on this metric. The

behavior that results when the second angle behavior's combination method is set to closest is shown in figure 4-6 below.

Table 4.3 Snake shape model combination method sensitivity analysis

1st angle behavior					
combination method	total fitness	snake shape	negative hollow circle	group	alignment
average	0.5	0.9	0.8	0.8	1.0
random	0.6	0.9	0.7	0.9	1.0
closest	1.2	1.4	0.8	1.0	1.0
furthest	0.4	0.7	0.8	0.7	1.1
2nd angle behavior					
combination method	total fitness	snake shape	negative hollow circle	group	alignment
average	0.6	0.9	0.7	0.9	1.0
random	0.3	0.7	0.7	0.6	1.0
closest	4.2	1.8	0.9	0.8	3.0
furthest	1.3	1.3	0.7	1.3	1.1

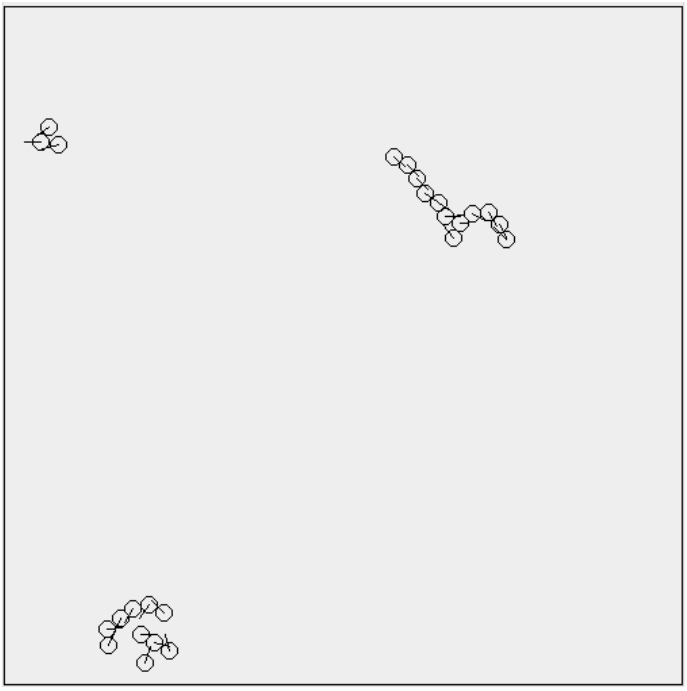

Figure 4-6 Snake shape model's 2nd angle behavior's combination method is set to "closest" resulting in bad behavior

Determining the significance of the filter of the first angle behavior is the next task. To do this the filter range is varied as shown in table 4-4 for the snake shape metric. Varying the filter range has a dramatic effect on the value of the snake shape metric as table 4-4 below shows. In table 4-4 the original filter range is highlighted. As the upper bound on the filter range increases fitness drops as agents expand their focus to include agents on the sides of them. As the lower bound increases agents stop heading towards agents in front of them. There is limited flexibility in what the filter range can be. For example, the ranges [0°, 72°] and [0°, 36°] both have a fitness of 1.0.

Table 4.4 Snake shape model's first behavior's filter sensitivity analysis with respect to snake shape fitness metric

		Lower bound of filter range									
		0	18	36	54	72	90	108	126	144	162
Upper bound of filter range	180	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9
	162	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9	
	144	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9		
	126	1.9	1.9	1.9	1.9	1.9	1.9	1.9			
	108	1.9	1.9	1.9	1.9	1.9	1.9				
	90	1.8	1.7	1.9	1.9	1.9					
	72	1.0	1.4	1.9	1.9						
	54	0.9	1.2	1.9							
	36	1.0	1.6								
	18	1.5									

The final step in the analysis of a model is to determine how robust the model is to variations in the number of agents and the world size. The model is robust to the number of agents. Although the model is robust for a variety of world sizes it has a reduced fitness for very small and very large worlds as shown in figure 4-7 below. Due to crowding the agents have a hard time spreading out to form a snake shape in very small worlds. In very large worlds the group metric worsens indicating that agents are having a hard time finding one another in the large world within the model burnin time.

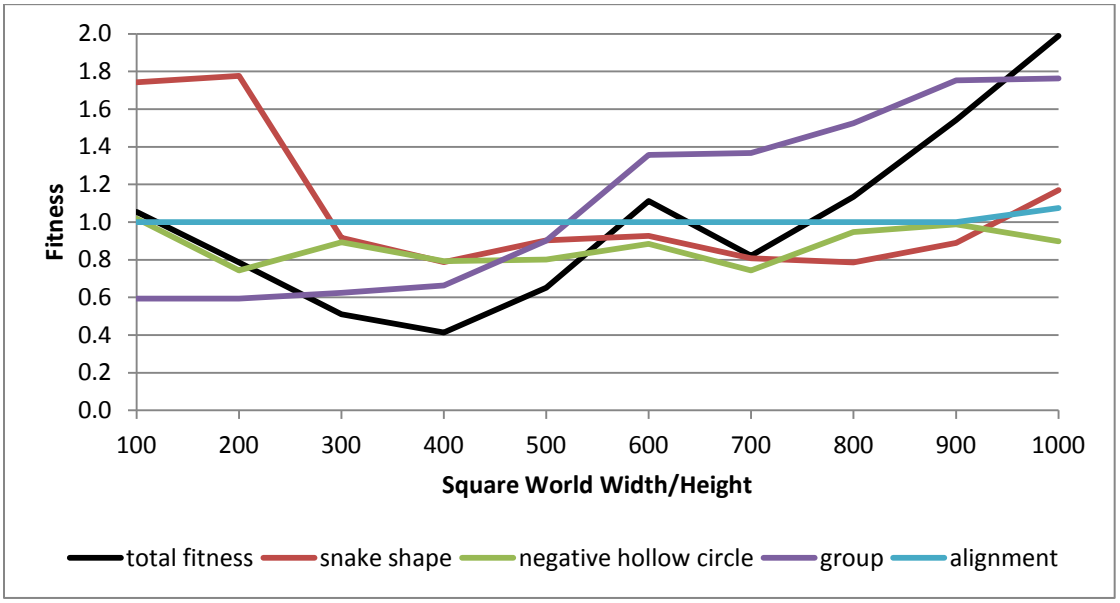

Figure 4-7 Snake shape model's world size robustness analysis

4.2.2 Snake Shape with Obstacle Avoidance

In this experiment the previously discovered snake shape model is used as a starting point to discover a model with obstacle avoidance in presence of 5 obstacles. This experiment demonstrates how existing models (whether discovered or handcrafted) can be used in the search process. This experiment uses the four metrics (alignment, snake shape, group, and negative hollow circle) used in the snake shape without obstacles experiment in addition to the touches obstacle metric. A simulation of the discovered model is given in figure 4-8. In the previous section on the simplified model was given; however, in this section to illustrate the polishing an simplification steps the discovered and polished models are also included in tables 4-5 and 4-6. Notice that the polished model is similar to the discovered model, but there are some differences in the filters. For example, the speed behavior's filter is removed and the filter range of the first

behavior is changed from [0°, 90°] to [0°, 54°]. No attempt should be made to interpret the raw discovered model or the polished model, because they both contain junk behaviors that are irrelevant to the observed behavior. Only the simplified model should be interpreted. When the polished model is simplified the complexity drops from 11 to 7, indicating the presence of a significant number of junks behavioral elements.

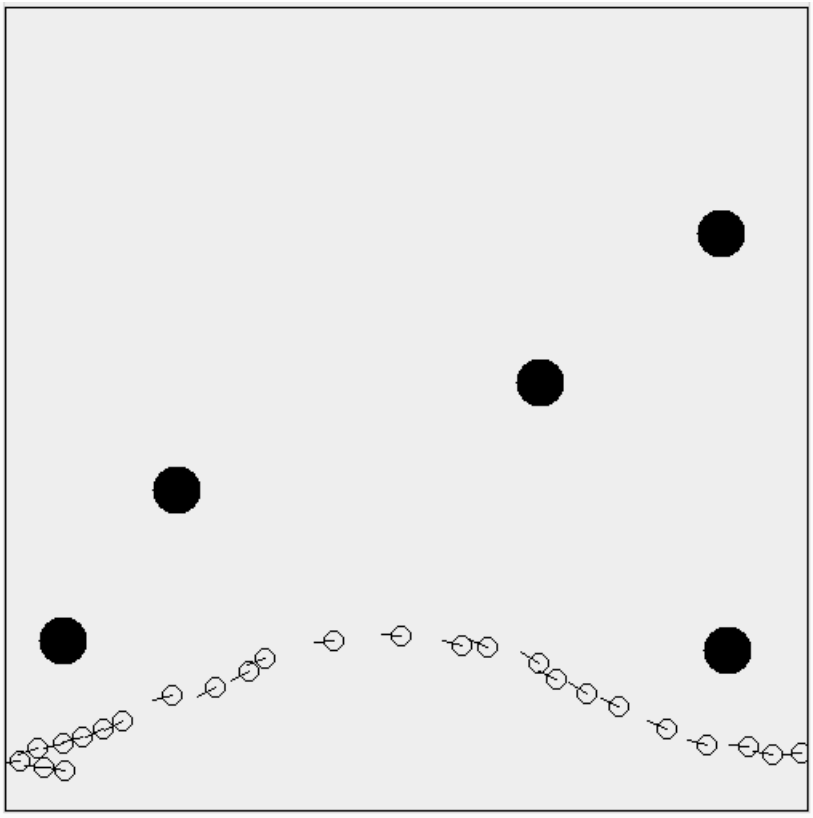

Figure 4-8 Snake shape with obstacle avoidance model

Table 4.5 Snake shape with obstacle avoidance model before polishing and analysis

Angle Behaviors					
ID	**observe**	**1st Filter**	**2nd Filter**	**Property to Extract**	**Offset**
1	agents	difference between observer's angle to agent and observer's heading in [0.0, 90.0]	none	position	0
2	obstacles	perpendicular distance from observer's heading line in [0.0, 67.5]	none	position	120.0
3	agents	none	none	heading	-10.0
4	obstacles	perpendicular distance from observer's heading line in [0.0, 60.75]	none	position	120.0
Speed Behaviors					
ID	**observe**	**1st Filter**	**2nd Filter**	**Property to Extract**	**Offset**
5	agents	difference between observer's angle to agent and observer's heading in [0.0, 90.0]	distance to observer in [0.0, 67.5]	speed	2.0

Table 4.6 Snake shape with obstacle avoidance model after polishing (but before simplification)

Angle Behaviors					
ID	**observe**	**1st Filter**	**2nd Filter**	**Property to Extract**	**Offset**
1	agents	difference between observer's angle to agent and observer's heading in [0.0, 54.0]	none	position	0°
2	obstacles	perpendicular distance from observer's heading line in [0.0, 67.5]	none	position	120.0°
3	agents	none	none	heading	-10.0°
4	obstacles	perpendicular distance from observer's heading line in [0.0, 60.75]	none	position	120.0°
Speed Behaviors					
ID	**observe**	**1st Filter**	**2nd Filter**	**Property to Extract**	**Offset**
5	agents	none	none	speed	1.8

The simplified model given in table 4-7 below is ready for analysis. This model's behavior is similar to the snake shape model give in the previous section, so the resulting models should be similar. The first angle behavior has an identical filter range and the offset is 0°

whereas in the previous section's model the offset was -10°. However, offset analysis on the previous section's models indicates that this difference isn't significant. Since both models' behaviors are essentially identical we can conclude that the purpose of this model's first behavior is also cohesion.

The second behavior is almost identical to the second behavior in the previous section's model. The only difference is that the offset is negative instead of positive; however, this isn't a significant difference because the sign only indicates the turning direction. Therefore we can conclude that like the previous section's model's second behavior this behavior also serves the purpose of alignment.

The only truly new behavior is the third behavior, which serves the purpose of obstacle avoidance. This is the only behavior to observe obstacles instead of agents. The offset is high, 120°, so that the agent turns away from obstacles. The filter causes the agent to prioritize avoiding nearby obstacles.

Table 4.7 Snake shape with obstacle avoidance model after polishing and simplification

Angle Behaviors					
ID	**observe**	**1st Filter**	**Property to Extract**	**Offset**	**Interpreted Purpose**
1	agents	difference between observer's angle to agent and observer's heading in [0.0, 54.0]	position	0°	cohesion
2	agents	none	heading	-10.0°	alignment
3	obstacles	perpendicular distance from observer's heading line in [0.0, 60.75]	position	120.0°	Obstacle avoidance

More can be learned about the third angle behavior by doing offset sensitivity analysis. The offset analysis for the third behavior is given in figure 4-9 below. Observe how the touches obstacle metric is good for any angle offset with an absolute value greater than 90°. It is

interesting how the exact value of the offset does not matter much once it passes an absolute value of 90°.

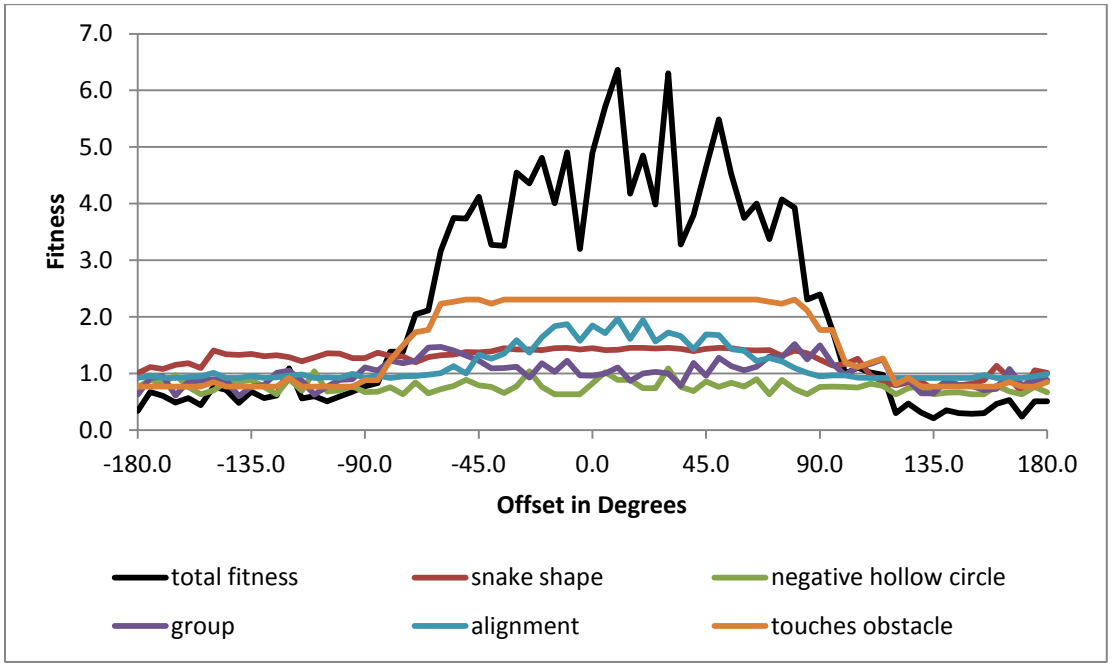

Figure 4-9 Snake shape with obstacle avoidance model's 3rd behavior's offset analysis

The next step is determining the significance of the filter. For the snake shape, negative hollow circle, group, and alignment metrics all fitness values for all filter ranges are fit and are between 0.7 and 1.3, so we can conclude that the filter range doesn't impact these metrics. The touches obstacle fitness varies between 0.8 (fit) and 2.0 (not fit), so it is of interest. The filter range sensitivity analysis for the third behavior and the touches obstacle metric is given in table 4-8 below. One observation is that when the filter range is very narrow; [54,60.8] and [0, 6.8] for example the fitness is bad. This occurs because obstacles are only avoided if they fit within this narrow range, which occurs rarely. Another observation is that the fitness is good when the filter range is at its maximum, [0, 67.5]. This is a hint that the filter might not be necessary. And in fact when the filter is completely removed the model remains fit.

Table 4.8 Snake shape with obstacle avoidance model's 3rd behavior's sensitivity analysis with respect to touches obstacle fitness metric

		Lower bound of filter range									
		0	6.8	13.5	20.3	27	33.8	40.5	47.3	54	60.8
Upper bound of filter range	67.5	0.9	0.9	1.4	1.0	1.4	1.0	1.5	1.3	1.7	1.5
	60.8	1.2	1.0	1.1	1.1	1.2	1.1	1.3	1.7	1.8	
	54	0.8	0.8	1.3	0.9	1.2	1.6	1.5	1.7		
	47.3	0.9	1.1	1.1	1.4	1.3	1.5	1.5			
	40.5	0.8	0.8	1.4	1.1	1.4	1.6				
	33.8	0.9	1.0	1.4	1.5	1.8					
	27	1.2	1.1	1.1	1.8						
	20.3	1.9	2.0	1.8							
	13.5	2.0	2.0								
	6.8	1.9									

Additional insight into the third behavior can potentially be gleaned through combination method sensitivity analysis. The original combination method is random, which is highlighted in table 4-9 below. For the most part the model is insensitive to the combination method, but when it is furthest it hurts the touches obstacle metric noticeably. The combination method furthest causes agents to avoid the furthest obstacle that they avoid. As expected this hurts fitness because it causes agents to avoid far obstacles at the expense of avoiding nearby obstacles. However, it is interesting that the touches obstacle metric is not worse than 1.4. The result shows that in practice the scenario described previously does not happen often enough to cause a catastrophic drop in fitness.

Table 4.9 Snake shape with obstacle avoidance model's 3rd behavior's combination method analysis

3rd angle behavior						
combination method	total	snake shape	negative hollow circle	group	alignment	touches obstacle
average	0.6	0.8	0.7	0.9	0.9	1.1
random	0.5	0.8	0.9	0.8	0.9	0.9
closest	0.3	0.8	0.6	0.8	1.0	0.8
furthest	0.8	0.9	0.6	1.0	0.9	1.4

To determine the utility of the model its robustness to variations in the world size, number of agents, and number of obstacles must be determined. As figure 4-10 below shows the model is robust to different world size except for very small and very large worlds. This model is more sensitive to small world sizes than the previous sections model, because the addition of obstacles increases crowding. Like the previous sections model, this model also has trouble grouping at very large world sizes that make it hard for agents to find one another within the burnin time. Figure 4-11 confirms the expectation that as the number of obstacles increase the touches obstacle metric gets worse. The model is robust with respect to the number of agents; like the previous section's model.

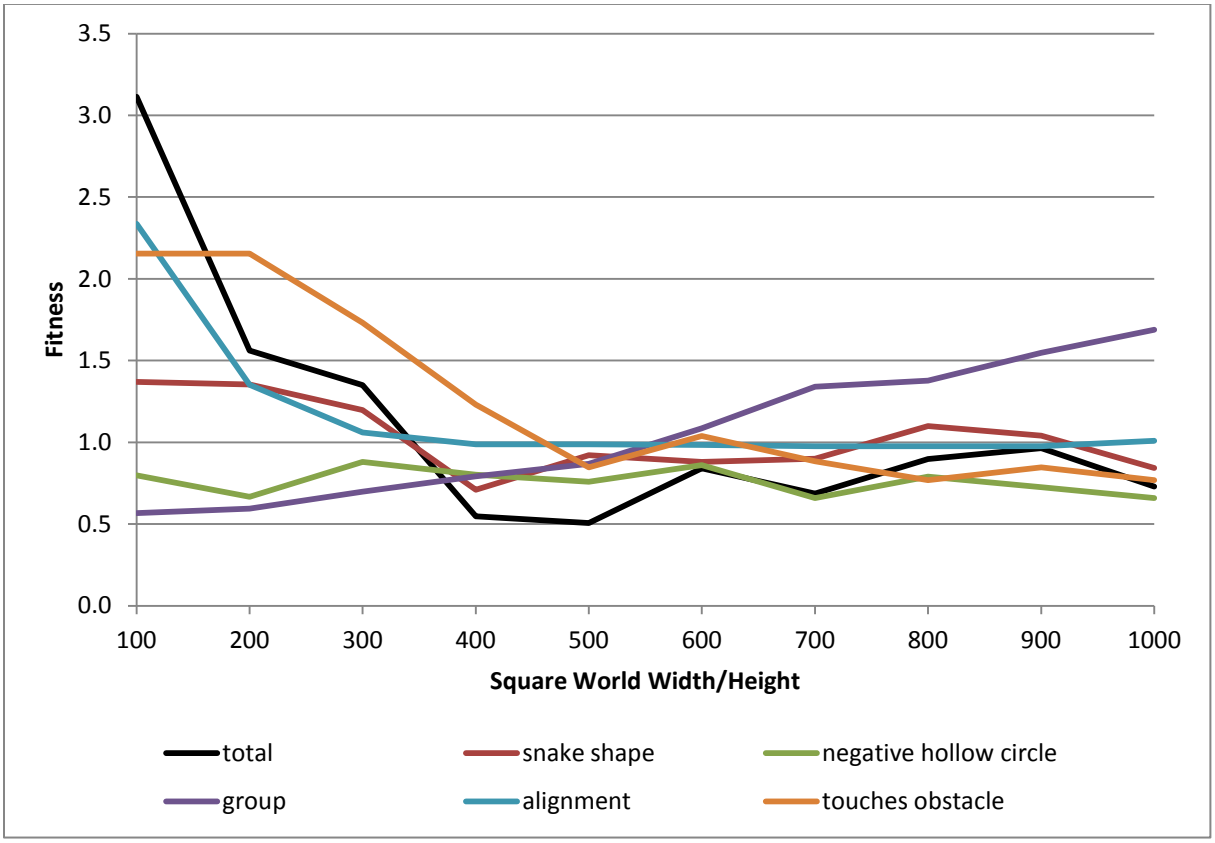

Figure 4-10 Snake shape with obstacle avoidance model's world size robustness analysis

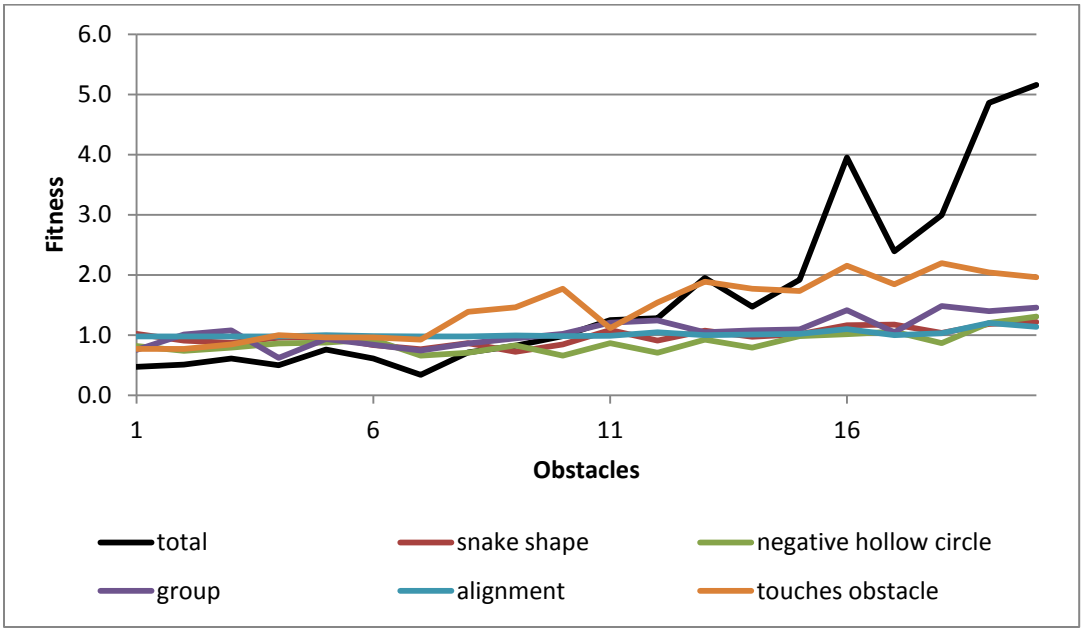

Figure 4-11 Snake shape with obstacle avoidance model's number of obstacles robustness analysis

4.2.3 Hollow Circle Shape Model

The hollow circle behavior is one where agents form a circle pattern as seen in figure 4-12. The hollow circle fitness metric was used to evaluate models. The discovered model is shown in table 4-10 below. The angle behavior causes the agents to form a circle; specifically, the agents circle in towards one another until physical limitations cause them to bunch up in a tight hollow circle shape. The sign of the angle behavior's offset controls which direction the agents rotate; making it positive would reverse the rotation direction. The speed behavior is unnecessary, because when it is removed the model remains fit. Like the snake shape behavior, this speed behavior has gotten past the simplification step due to the randomness inherent to fitness calculations based on simulations. Filter range sensitivity analysis reveals that the angle behavior's filter is unnecessary because the fitness of the model is fit, 0.9 to 1.0, for all filter

ranges. After some analysis the speed behavior and the angle behavior's filter can be removed. Although manual simplification is necessary for this model sensitivity analysis makes this process formulaic. Furthermore, the simplification of the polished model is still useful because it brought the complexity down from 9 to 4.

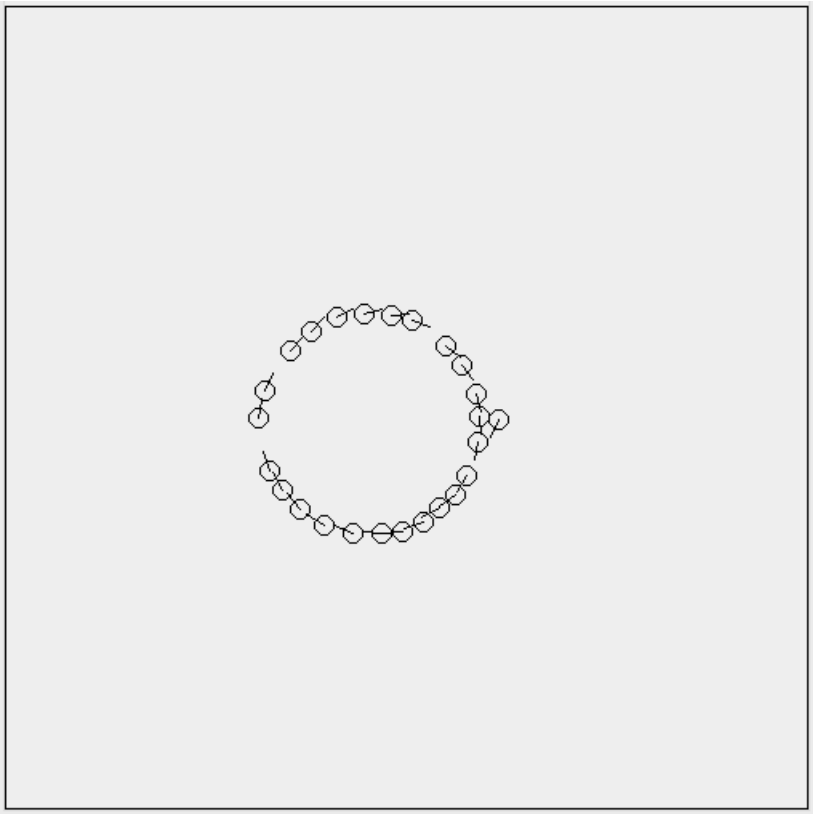

Figure 4-12 Hollow circle model

Table 4.10 Hollow circle model

Angle Behaviors				
ID	**1st Filter**	**Property to Extract**	**Offset**	**Interpreted Purpose**
1	difference between agent's angle to observer and agent's heading in [0.0, 108.0]	heading	-90.0	Circling
Speed Behaviors				
ID	**1st Filter**	**Property to Extract**	**Offset**	**Interpreted Purpose**
2	none	speed	0.8	unnecessary

The next step is to analyze the sensitivity of the angle behavior's offset given in figure 4-13 below. The speed behavior's offset will not be analyzed, because that behavior has already been determined to be extraneous. Figure 4-13 shows that the model is never fit when the absolute value of its offset is less than 90°. The model is fit when the absolute value of the offset is 90°, but as the absolute value of the offset increases the model tends to become less fit. The high peaks and troughs of fitness are caused by the randomness of the simulations.

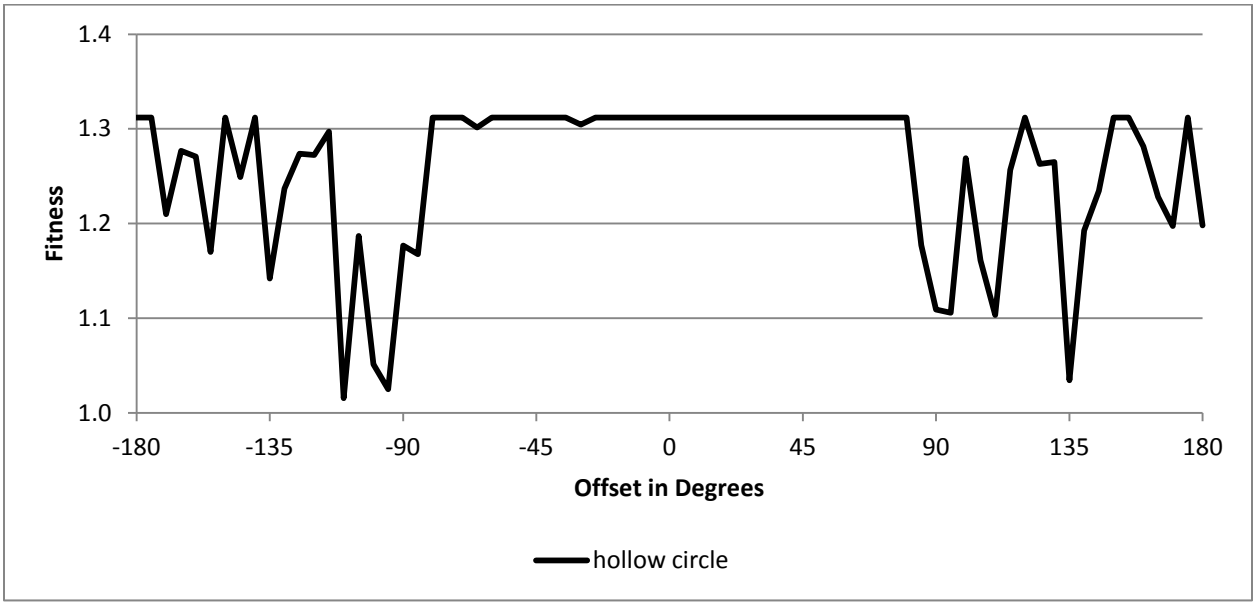

Figure 4-13 Hollow circle model's 1st behavior's offset sensitivity analysis

Unlike most models the combination method for the angle behavior is significant. Table 4-11 below shows that the model is only fit when the combination method is average. When the combination method is not average the agents do not spiral in to form a hollow circle shape.

Table 4.11 Hollow circle model's 1st behavior's combination method sensitivity analysis

angle behavior 1	
combination method	**Hollow circle**
average	1.1
random	1.3
closest	1.3
furthest	1.3

The model is robust to different numbers of agents and world sizes. When the number of agents is varied between 25 and 100 the fitness varies between 0.9 and 1. When the world size is varied between 300 and 1,000 the fitness remains below 1.1.

4.3 Controllability and compossability

This section presents results that demonstrate that models represented using the framework are controllable and composable; two framework evaluation metrics. A model is controllable if a modeler can determine how to modify an existing model into a related model that generates desired behavior that is similar but not identical to the unmodified model. Controllability is valuable because it makes it possible to re-use models in situations that are different from the ones in which they were learned. Compossability refers to model reuse and the ease with which behaviors can be combined. Compossability is valuable, because model reuse allows accumulated modeling knowledge to be utilized. In the first subsection a handcrafted personal space model is composed with the discovered snake shape model. This model is in turn modified (controlled) to adjust the personal space distance between agents. This illustrates that sometimes model compossability and controllability are used at the same time. In the last subsection a well known flocking model, Boids [3], is handcrafted and the obstacle behavior learned for the snake shape with obstacle avoidance model is added (composed) to it.

4.3.1 Adding Personal Space to Snake Model

To demonstrate that discovered models are composable and controllable, personal space behavior will be added to the discovered snake shape model and the amount of personal space will be controlled. If the personal space of agents is respected then there is a minimum distance between all agents; like the personal space between people. The speed behavior in the discovered

model was found to be unnecessary, so the snake shape model being used as a model base doesn't have it (it has 2 angle behaviors).

The discovered snake shape model already forms the correct shape, so only the spacing between agents needs to be controlled. This is done by adding a speed behavior that causes an agent to slow down if it gets too close to an agent in front of it. Once the agent has backed off enough that the behavior no longer activates, then the default speed behavior takes over and the agent speeds up to maintain the new spacing. A handcrafted speed behavior was added to the discovered model as shown in table 4-12 below. The first filter in the speed behavior activates when there are agents within a 90° degree arc in front of the agent; agents change their speed with respect to agents they are following. The second filter in the speed behavior ensures that the behavior is only triggered when the agent being followed is close; within a distance of 24.

Table 4.12 Snake shape model with personal space preservation

Angle Behaviors					
ID	**1st Filter**	**2nd Filter**	**Property to Extract**	**Offset**	**Interpreted Purpose**
1	difference between observer's angle to agent and observer's heading in [0.0, 54.0]	None	angle between observer and observed boid	-10.0	cohesion
2	none	none	heading	10.0	alignment
Speed Behaviors					
ID	**1st Filter**	**2nd Filter**	**Property to Extract**	**Offset**	**Interpreted Purpose**
3	difference between observer's angle to agent and observer's heading in [0.0, 45.0]	distance to observer in [0.0, 24.0]	speed	-0.1	Slow down to spread out agents

To emphasize the spacing the world size has been increased from 500x500 to 800x800 in figures 4-14 and 4-15 below and the burnin time has been increased to 800 with a simulation time of 1,000. Figure 4-14, below, shows the snake shape model with a personal space distance of 24 (speed behavior's 2nd filter's range is [0, 24], which is 4 times the radius of an agent, so there should be roughly 1 agent diameter between agents, because distance is measured from the

centers of an agents. Personal space can be controlled by controlling the speed behavior's 2nd filter's range; in figure 4-15 it is changed to [0, 48], which results in 3 agent diameters worth of space between agents.

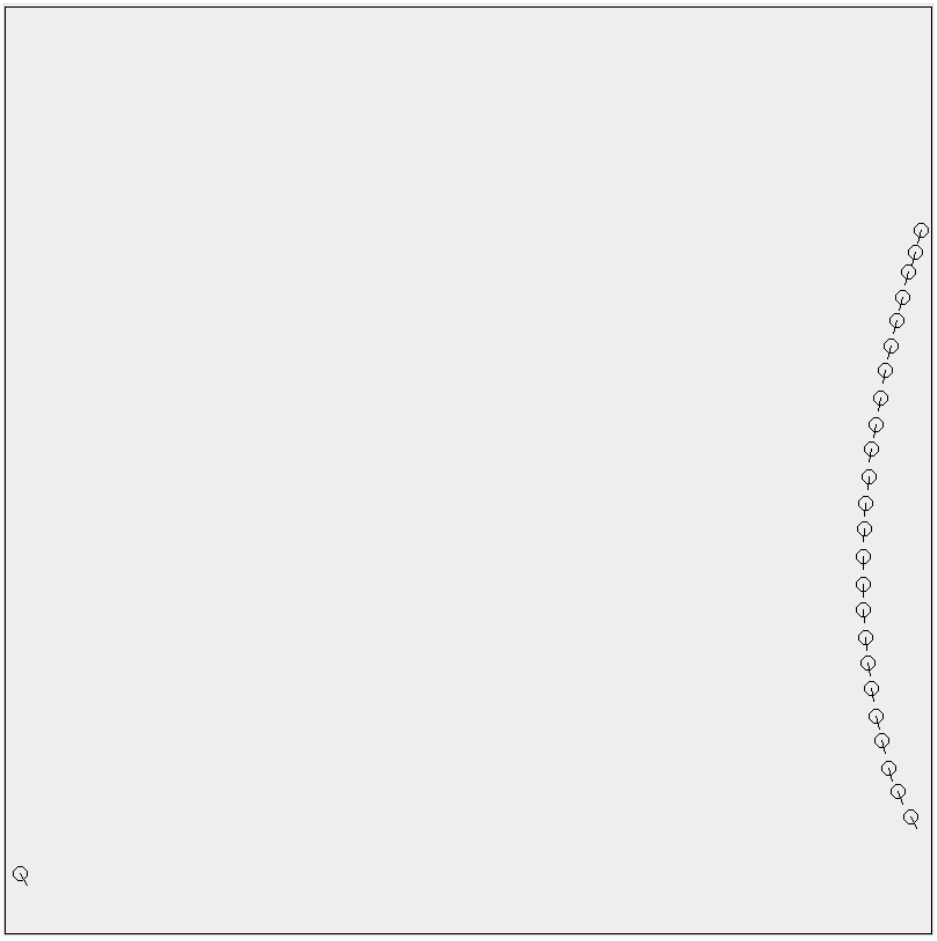

Figure 4-14 Snake shape model with personal space; personal space distance = 24, world size = 800x800

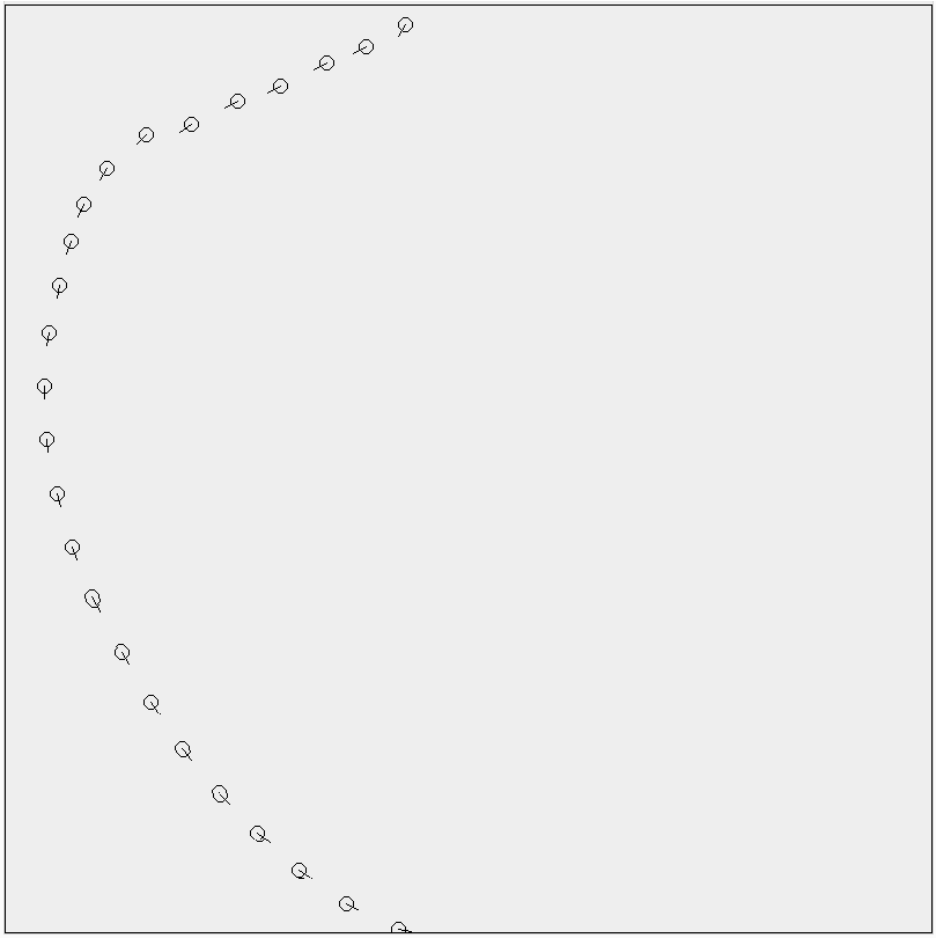

Figure 4-15 Snake shape model with personal space; personal space distance =48, world size 800x800

4.3.2 Modifying Boids Model

The boids model of flocking was first introduced by Reynolds in 1987 [3]. It is famous (or as famous as an academic paper can be) for being one of the earliest agent based models. Flocking refers to the schooling behavior of fish, the flocking of birds, or the movement of herds of animals. It showed how just 3 behaviors, separation, alignment, and cohesion, could replicate the behavior of flocks. Separation refers to individual flock members avoiding collisions with other agents. Alignment refers to each agent setting its heading as the average of its neighbors. Finally, cohesion refers to agents being attracted to one another, so that a group is formed. To

demonstrate that models within the model space are composable the boids model will be handcrafted and the discovered obstacle avoidance behavior from the snake shape with obstacle avoidance model will be added to it.

The handcrafted boids model is given in table 4-13 below and the simulated model is shown in figure 4-16 below. To give the agents time to form a group the burnin time was increased to 800 (from 400). The first angle behavior causes agents to turn away from other agents when they get within a distance of 24 (equal to 2 agent diameters). The second angle behavior causes an agent to head towards the average position of agents that are in front of it; within a 180° arc (within 90° of the agent's heading). Finally, the last behavior aligns the agents by setting an agent's heading to the average heading of its neighbors.

Table 4.13 Handcrafted boids model

Angle Behaviors					
ID	**1st Filter**	**Property to Extract**	**Combination method**	**Offset**	**Interpreted Purpose**
1	distance to observer in [0.0, 24.0]	position	closest	180.0°	separation
2	difference between observer's angle to agent and observer's heading in [0.0, 90.0]	position	average	0°	cohesion
3	none	heading	average	0°	alignment

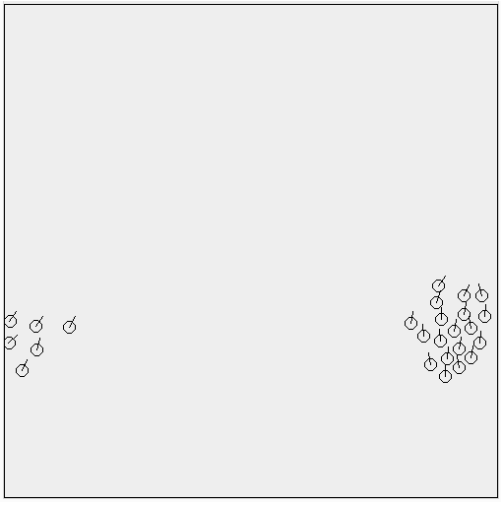

Figure 4-16 Handcrafted boids model

To demonstrate compossability the discovered obstacle avoidance behavior from the snake shape with obstacle avoidance model is added to the handcrafted boids model. The resulting model is shown in figure 4-17 below. Without the obstacle avoidance behavior the boids model by itself has an average of 3.78 obstacle collisions per iteration. When the obstacle avoidance behavior is added the average number of obstacle collisions per iteration drops to 0.72. Both of these averages were averaged over 100 simulations, where each experiment was 500 iterations long with a burnin of 0 (obstacle collisions measured from start of experiment).

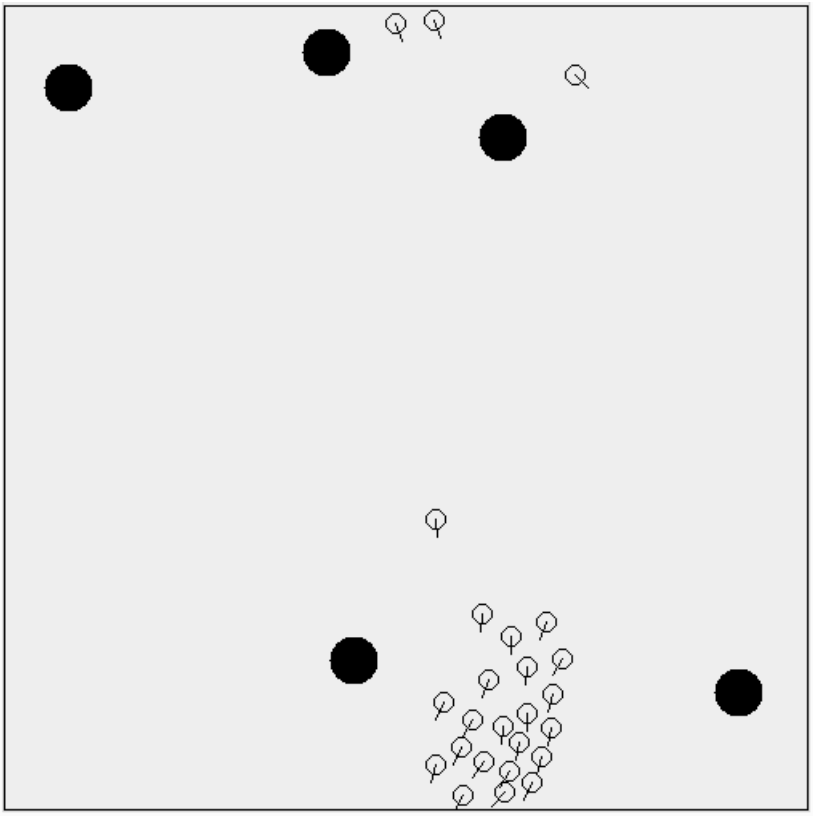

Figure 4-17 Handcrafted boids model with added obstacle avoidance behavior taken from the discovered snake shape with obstacle avoidance model

In this chapter a variety of models were discovered, analyzed, and re-used in various ways. In the next chapter the framework will be extended to take advantage of incomplete knowledge and a lane formation model will be discovered.

5 INCORPORATING KNOWLEDGE INTO DATA-DRIVEN MODELING THROUGH THE USE OF SYSTEM ENTITY STRUCTURES (SES'S)

This chapter describes how to extend the data-driven modeling framework presented in chapter 3 to incorporate incomplete knowledge. In chapter 3, the framework assumes that the modeler has no knowledge of how a mobile agent system produces emergent behavior. However, frequently the modeler has some idea about how the real system works, but does not have a complete picture. Therefore the modeler, using knowledge driven modeling techniques, builds the part of the system they understand and builds the rest of the system through trial and error (although they might not mention this). In this chapter the framework presented in chapter 3 is extend through a modeling technique that uses System Entity Structures (SESs) to incorporate incomplete knowledge. SES makes it possible to formally specify the parts of the system that are understood and the parts that are unknown (and will make up the search space).

5.1 Introduction to System Entity Structures (SESs)

The concept of a System Entity Structure (SES) dates to 1990 [2]; however, its application to this framework is novel. A SES formally defines all the possible ways that entities can be arranged in a system being modeled. For example, if the system to model is a car, than the entities could be different types of engines, different types of hubcaps, different types of interior layouts, etc. An SES for a car is given in figure 5-1 below. The PES in the figure will be discussed shortly.

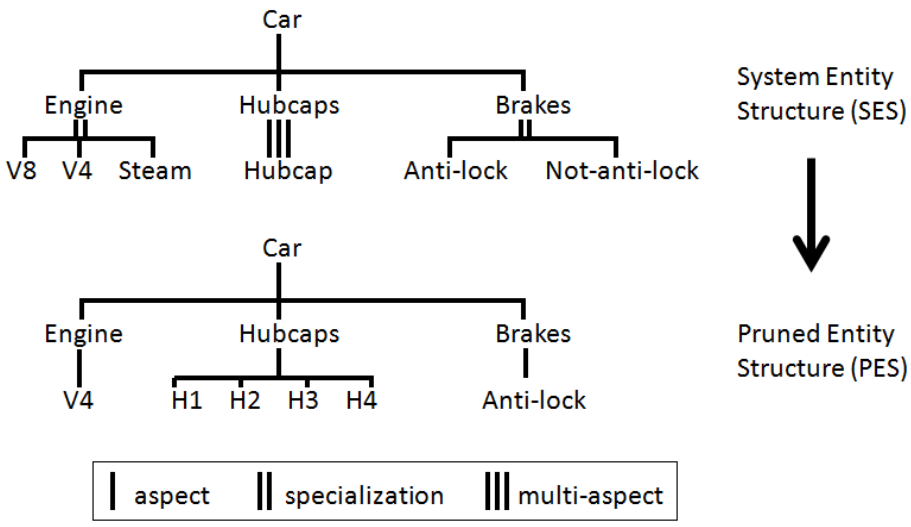

Figure 5-1 Converting car SES into car PES

The SES given above represents all the possible constructions of the car model. Real cars have many more parts, but the car being modeled is at a level of abstraction where the modeler only cares about the engine, hubcaps, and brakes. Single lines indicate the decomposition of an entity into its aspects. The engine, hubcaps, and anti-lock brakes are all aspects of the car entity. Double lines, called specialization, indicate that one entity must be chosen from many options. For example, the engine can either be a V8, V6, or steam (for the adventurous). Similarly, the brakes can be either anti-lock or not-anti-lock. Triple lines, called multi-aspect, indicate that the number of entities is variable. In the figure above, the number of hubcaps can vary based on how many have currently been stolen, but not yet replaced. A requirement of multi-aspects is that the model be able to meaningfully run with a variable number of entities. In the car example, a car without hubcaps can be simulated (but may cease to serve as a status symbol). The car SES is simple, but SES trees can be as broad and deep as desired.

A SES is a formal tree like definition for the different ways a system can be constructed. A SES is pruned to create a system model called a Pruned Entity Structure (PES) as shown in

figure 5-1 above. Whereas a SES is a definition of a set of possible models, a PES is a specific model. The PES in figure 5-1 represents a car model with a V4 engine, all 4 hubcaps, and anti-lock brakes. The PES is technically not a model itself, but rather a definition for how entities, taken from a model database, can be combined to create a model. In figure 5-2 below, the entity model database and model synthesis components are fixed, so the conversion from a specific PES to a model is well defined. Therefore, in this chapter the terms "PES" and "model" are used synonymously.

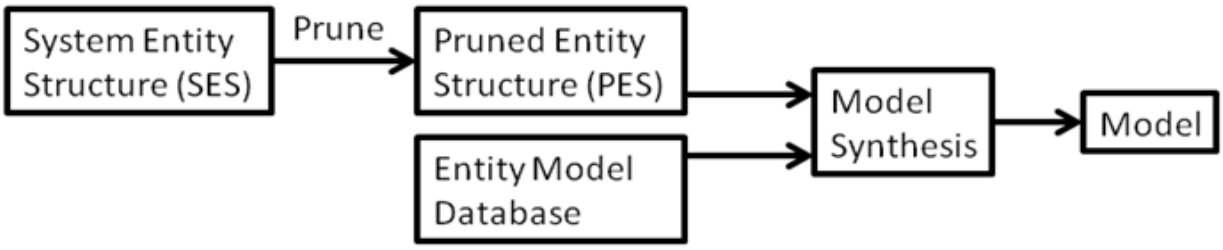

Figure 5-2 Generating a model from a SES and entity model database

The modeler determines how to prune the SES. One option is to randomly prune the SES, which essentially randomly samples models from the SES, which may be useful if the SES defines more models than can be simulated (some SESs might have millions of distinct PESs). Another strategy is to methodically prune the SES multiple times until all PESs that are possible have been generated. In the car example in figure 5-1 above, the SES is simple enough that it can be exhaustively pruned. The final option is a guided SES pruning strategy where the goal is to find a pruned model which exhibits some properties when simulated; it is this approach that the framework SES extension uses.

5.2 Using SES to Discover Mobile Agent Models

The framework uses SES to formulize how partial knowledge is incorporated into the model discovery process. The first step is to create a general framework SES, which represents

how the framework structures mobile agent models. This general SES is shown in figure 5-3 below. Each model has one or more behavior groups. The number of behavior groups is not fixed because the number or properties that can be modified by behavior groups is not yet fixed. Each behavior group can have multiple behaviors. Like the behavior groups, the number of behaviors is not fixed. Each behavior can be specialized into one of many possible behaviors.

The general framework SES is partially pruned to create a specific mobile agent SES. Partial pruning refers to when a SES is pruned to create a simpler SES (as measured by the number of possible prunings). The mobile agent SES represents how mobile agents are currently represented in the framework. Mobile agents have a behavior group for the angle agent property and another behavior group for the speed agent property. Each of these behavior groups can have 1 or more behaviors. Each behavior can be specialized into one of many possible behaviors.

The mobile agent SES is pruned into the model space search PES. When the GA searches through the model space it only considers models of a certain size, this PES represents that size. Each model has angle and speed behavior groups. Each of these groups has a set number of angle and speed behaviors. Each behavior entity in the model space search PES, for the rest of this chapter called "PES" for convenience, is actually a behavior restriction entity. A behavior restriction entity defines a set of behaviors that satisfy those restrictions. By restricting a behavior a modeler can insert incomplete knowledge into the search process. For example, if a modeler expects that one behavior is an avoidance behavior, then the offset could be restricted to 180° (along with other possible restrictions). A model generated from the PES is a valid if it conforms to all behavior restrictions in the PES. It is possible that there are no angle or speed behaviors and that the behavior groups are empty. In this case the default hardcoded angle and

speed behaviors are used to determine behavior. Furthermore, if a behavior is completely restricted, then this is the same as including a fixed behavior.

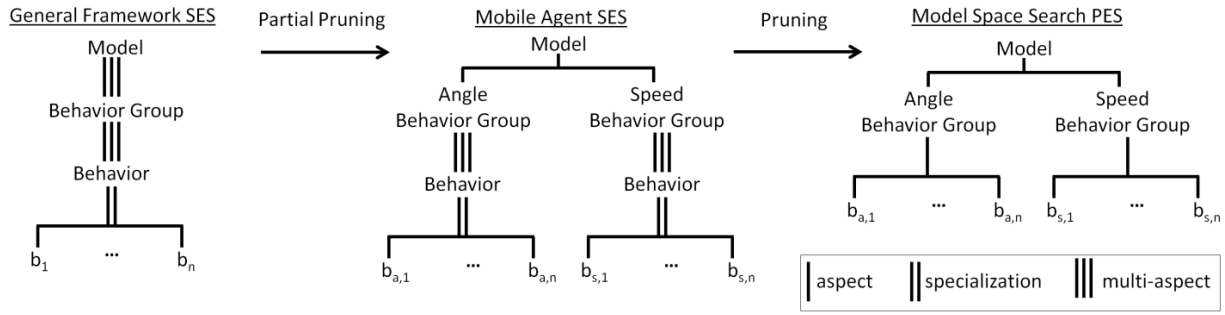

Figure 5-3 Progressing from general framework SES to model space search PES

A behavior restriction, $\boldsymbol{b_r}$, has a similar form to a behavior:

$\boldsymbol{b_r = <C_r, T_r, F_r, Extract_r, Combine_r, Offset_r>}$, *where*
$\boldsymbol{C_r}$ *is the set of allowed categories*
$\boldsymbol{T_r}$ *is the set of allowed types*
$\boldsymbol{F_r}$ *is the set of allowed filters*
$\boldsymbol{Extract_r}$ *is the set of allowed extractable properties*
$\boldsymbol{Combine_r}$ *is the set of allowed combination methods*
$\boldsymbol{Offset_r}$ *is the set of allowed offsets*

Notice that there are some dependencies here. For example, certain categories and types have certain possible filters and certain extractable properties only allow certain combine and offset options. This isn't an issue, because these restrictions are defined by the modeler who can take into account these limitations. Behavior restrictions give the modeler lots of flexibility in deciding what incomplete knowledge to include.

The model space search PES is used to extend the GA search method described in chapter 3. Anytime the GA creates a random model, that model must conform to the behavior restrictions in the PES. Any time a model is mutated, the resulting mutation must result in a model valid with respect to the PES. Finally, the GA crossover step must also result in valid models. Lastly, the GA gradually adds behaviors to the model as part of the search process;

instead of adding a behavior each time it adds a behavior restriction (from which a behavior is sampled).

5.3 SES Applied to Lane Formation

When two groups of people approach one another in a narrow hallway from opposite directions they spontaneously and without verbal communication form lanes to move quickly [1]. In this experiment SES is used to discover a model which replicates this behavior. Although the behavior is known to exist, how it occurs is still not fully understood; a perfect chance to test the framework's ability to handle incomplete knowledge.

5.3.1 Experimental Setup

The starting position of the agents and the dimensions of the hallway were replicated from [1], where a lane formation experiment was conducted with real people. One group of 27 agents starts in the left start zone and travels towards the right goal zone. Another group of 27 agents starts in the right start zone and travels towards the left goal zone. Each agent is assigned one of two types that correspond to the group it is a member of. All agents that start on the left have the same type. Similarly, all agents that start on the right have the same type. Each group of agents gets its own set of behaviors; however, these behaviors are identical except that all references to a specific group are swapped. For example, a behavior that causes an agent to avoid the other group will identify the other group by its type which will vary depending upon the type of the agent (the behavior belongs to). Models are evaluated based on how quickly agents reach their goal zone, how direct a path they take to the goal zone, and how evenly they spread out in the hallway measurement area to use all of the available hallway width. The agents can perceive the goal zones, but not the hallway measurement area zone or the impermeable walls that border all 4 edges of the world. The length of the entire hallway is 41 meters and its width is 3 meters.

The hallway measurement area is 14 meters long. By comparison agents take up a circular space 0.4m (~1.3 ft) in diameter. Note that the units of distance in the discovered model are in pixels and there are 30 pixels to a meter. The start positions of agents can be seen in frame A of figure 5-5 below. The start zone is split into 9 starting lines and each line has 3 agents randomly placed along it. The placement of agents was randomized for each simulation. The simulation length is 500 and the burnin time is 0.

Figure 5-4 Hallway zones used in lane formation model search

5.3.1.1 Fitness Metrics

A composite fitness function composed of three metrics is used to represent the lane formation behavior. The hallway travel time metric evaluates how quickly agents reach their goal zone. The hallway movement direction metric measures the degree to which agents are heading towards the goal zone. Finally, the hallway density metric measures the extent to which agents spread out to take advantage of the full width of the hallway. Before the hallway density metric was added each group of agents would form a single snake shaped line and these two lines would then pass by one another on their way to their goal zone. Although this behavior was fit for the travel time and movement direction metrics it did not look right because it was not utilizing the entire hallway width, thus the hallway density metric was added. This story illustrates how the framework can be used in an iterative modeling process.

5.3.1.1.1 Hallway Travel Time metric

This metric measures how quickly agents move from the start zone to the opposite goal zone. People naturally form lanes because it helps them (and the crowd) move faster. This metric

returns the number of agents not in the goal zone at the current iteration, where the goal zone is opposite the start zone; the agents that start in the east move to the west goal zone and vice versa. The travel time metric scale is 30, which means that if over the entire 500 timesteps in the simulation the agents are in their goal zone (54 - 30)/54 = 44% of the time or more than the model is fit. This could never be 100% because it takes agents some time to travel between their start position and goal zone.

5.3.1.1.2 Hallway Movement Direction Metric

In the lane formation scenario agents are trying to get from one side of the hallway to the other. This metric measures whether agents are heading towards their assigned goal region, by comparing their heading to the angle to the goal region as shown below:

$$metric\ value\ returned = \frac{\sum_{i}^{agents} |diff(angle\ to\ goal\ region\ for\ agent\ i, agent_{i,heading})|}{number\ of\ agents}$$

The hallway movement direction metric scale is 15°, which means that if on average agents are within 15° of heading directly towards the goal region they are fit with respect to this metric. Agents need to deviate from a direct path towards the goal region to avoid other agents.

5.3.1.1.3 Hallway Density Metric

The hallway density metric ensures that when agents travel down the hallway they spread out to use all of the available space. The hallway (not including the start zones) is split into 3 zones parallel to the movement direction. Each zone is in turn split into 3 sub-zones perpendicular to the movement direction. One way to think of it is that each zone represents a cross section of a hallway and the subzones in a zone are used to calculate how evenly spread agents are within a zone. In figure 5-5 panel A subzones that belong to the same zone are different shades of the same color. In lane formation figures where the subzones are not visible they have been hidden to make the figures more legible, but they are still used for fitness

calculations. These zones are used for fitness calculations, but are not entities in the environment; unlike the start, goal, and wall zones. A density metric is calculated for each zone (equation 1 below), then the sum of the zone density metrics is returned as the value of the hallway density metric (equation 2 below). The density metric measures how evenly agents within a zone are spread across the subzones; the more even the better. In equation 1, $\frac{\sum_i^{z_k} s_i}{|z_k|}$, represents how many agents would be in a subzone if all agents within a zone were evenly distributed between the subzones; the ideal distribution. If there are 3 or fewer agents in a zone, then the hallway density metric returns 0, because there aren't enough agents to measure density. Zero is returned, so that fitness is not penalized if there are no agents in a zone. This means that the model is perfectly fit with respect to the hallway density metric if agents never leave their start zones and enter the hallway; however, this is not a problem because the travel time metric will identify this model as unfit. Equation 1 measures the total deviation from the ideal distribution. Although there are three zones and 3 subzones per zone, other numbers of zones and subzones are possible. However, this combination of zones and subzones was found to work well in practice. The hallway density metric scale is 1.0, which means that each zone on average can deviate from the ideal density by at most 0.33.

Let $Z = \{z_1, z_2, z_3\}$, where: Z is the set of zones
$z_k = \{s_1, s_2, s_3\}$, where: s_i is the number of agents in the i'th subzone of zone k

$$density\ of\ k'th\ hallway\ zone = d_k = \sum_i^{z_k} abs(\frac{\sum_i^{z_k} s_i}{|z_k|} - s_i) \quad (1)$$

$$hallway\ density\ metric = \sum_k^{Z} d_k \quad (2)$$

5.3.1.2 Behavior restrictions used in lane formation model search

When deciding on what the behavior restrictions should be for the lane formation model search the question asked was: what are we certain agents will want to do in some form? There are 4 behavior restrictions: 1) avoid other agents, 2) avoid wall, 3) avoid all agents, and 4) head

towards the goal region. Agents want to avoid other agents, so they do not run into them. Agents want to avoid the wall, so they do not get stuck on it. Agents might also want to avoid other agents that are going in the same direction; this is done with a behavior that avoids all agents. Finally, agents definitely need a behavior guiding them towards the goal region.

The four behavior restrictions are given in table 5-1 below. The avoid other agents behavior is restricted to the other group of agents. The position is extracted and the closest combination method is used to select the closest agent from which the observing agent turns 180° away from. The filter is left undefined. In the avoid wall behavior restriction the category is zone and the type is wall. If the type were "any" then this could result in this behavior doing something in response to either goal zone, which is not desirable. Either avoiding the closest collision point or the closet point seems plausible, so both of these are included as possible properties to extract. The avoid all agents behavior restriction is similar to the avoid other agents restriction except that the type is set to all, so that the observing agent avoid all other agents. The goal zone behavior, 4th behavior restriction, is set as a fixed behavior, which is represented using a complete restriction.

Table 5.1 Behavior restrictions

ID	purpose	Categories (C_r)	Types (T_r)	Filters (F_r)	Properties to extract ($Extract_r$)	Combination methods ($Combine_r$)	Offsets ($Offset_r$)
1	Avoid agents from other group	Agent	Other group	Any filter allowed	Position	Closest	180°
2	Avoid wall	Zone	wall	Any filter allowed	Collision point, closest point	closest	180°
3	Avoid all agents	Agent	all	Any filter allowed	position	closest	180°
4	Head towards goal	Zone	Goal zone	No filters allowed	closest point	closest	0°

5.3.2 Analyzing the Discovered Lane Formation Model

The discovered lane formation model is shown in figure 5-5 below and defined in table 5-2. Frame A in figure 5-5 shows the starting position of the agents and the zones and subzones used in the hallway density fitness metric (this is hidden in subsequent frames for visibility). Frames B through F show the simulation at progressively later points in time. Notice in frame B that the agents at the head of their respective groups have started to maneuver slightly (notice their headings). The agents behind the vanguard agents can't see the other group yet, so they have not started to maneuver yet. Frame C is chaotic, it isn't clear that the model is going to work. By frame D the agents have formed lanes. In frame E the lanes start to disengage. Finally, in frame F the agents have reached their goal zones.

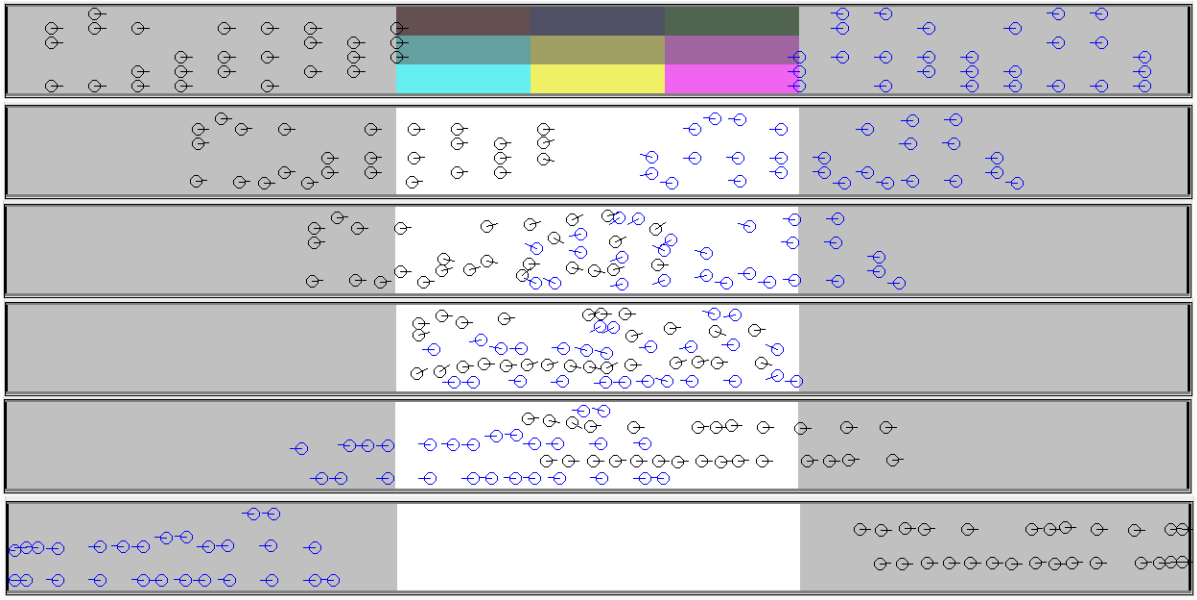

Figure 5-5 Lane formation model simulation. Panel A shows the zone and subzones used in the hallway density metric, which is hidden in panels B through F

The discovered lane formation model is defined in table 5-2 below. The behavior restrictions specified 4 behaviors (3 restricted behaviors, and 1 given behavior), but the fittest model has only 3. The avoid all agents behavior restriction turned out to not be necessary. This

isn't entirely unexpected, because all of the agents of the same type are heading in the same direction and it is unlikely that agents need to avoid agents going in the same direction. This behavior restriction was included just in case. The results show that even when behavior restrictions are mistakenly included the search method (and simplification step) is robust enough to eliminate them. The filters in all behavior restrictions were left undefined and it is here were the analysis will now focus.

Table 5.2 Discovered lane formation model

Angle Behaviors						
ID	**observe**	**1st Filter**	**2nd Filter**	**Property to Extract**	**Offset**	**Interpreted Purpose**
1	wall zone	distance to closest point in [0.0, 10.125]	none	closest point	180.0°	avoid wall
2	Other agent group	agent's perpendicular distance from observer's heading line in [0.0, 16.875]	difference between the observer's heading and the direction to the agent in [0.0°, 90.0°]	position	180.0°	avoid agents from other group
3	goal zone	None	none	closest point	0.0°	head towards goal

The first behavior causes agents to avoid the closest point on a wall zone if that zone is within a minimum distance of 10.125 (which at 30 pixels to a meter comes out to about 1/3 off a meter). Table 5-3 below shows that when the filter range is too large, travel time fitness drops substantially because agents jam up and can't reach the goal zone. This occurs because the agents are now avoiding distant walls instead of just focusing on avoiding closer agents. Note that the exact filter range in the model doesn't exist in table 5-3 (and other filter range tables), because of different levels of discretization, nevertheless the difference is small enough to not impact the interpretation of the results.

Table 5.3 Lane formation model's 1st behavior's filter range analysis for travel time metric

		Lower bound of filter range									
		0	6.8	13.5	20.3	27	33.8	40.5	47.3	54	60.8
Upper bound of filter range	67.5	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.3	1.3
	60.8	1.4	1.4	1.4	1.4	1.3	1.3	1.4	1.3	1.2	
	54	1.4	1.4	1.4	1.4	1.3	1.3	1.2	1.1		
	47.3	1.4	1.4	1.4	1.4	1.3	1.3	1.1			
	40.5	1.4	1.4	1.4	1.4	1.3	1.3				
	33.8	1.4	1.4	1.4	1.3	1.2					
	27	1.3	1.3	1.2	1.2						
	20.3	1.0	1.1	1.1							
	13.5	0.9	1.0								
	6.8	1.0									

The second behavior cause agents to avoid colliding with agents heading in the opposite direction (and having the opposite type). The first filter measures how close agents are to colliding with other agents (of the opposite type). Tables 5-4 and 5-5 below show the filter range analysis for the travel time and hallway density metrics. Table 5-4 suggests that maybe this filter is not needed at all, because when the filter range is at its maximum (and hence not doing any filtering), [0,67.5], the model is fit with respect to the travel time metric. However, the max filter range is unfit for the hallway density metric as table 5-5 shows. Thus at the max filter range the agents reach the goal zone, but underutilize the available width of the hallway. When using tables 5-4 and 5-5 at the same time we can see that there is a very small area centered on the discovered filter range in which this filter is fit for both metrics.

The second filter of the second behavior causes agents to only avoid agents that are in front of them; 180° arc. This is needed because without this agents will try and avoid agents that have already passed them.

Table 5.4 Lane formation model's 2nd behavior's 1st filter's range analysis for travel time metric

		Lower bound of filter range									
		0	6.8	13.5	20.3	27	33.8	40.5	47.3	54	60.8
Upper bound of filter range	67.5	1.1	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4
	60.8	1.1	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	
	54	1.1	1.3	1.4	1.4	1.4	1.4	1.4	1.4		
	47.3	1.1	1.4	1.4	1.4	1.4	1.4	1.4			
	40.5	1.1	1.3	1.4	1.4	1.4	1.4				
	33.8	1.1	1.3	1.4	1.4	1.4					
	27	1.0	1.2	1.3	1.4						
	20.3	1.0	1.2	1.4							
	13.5	1.0	1.3								
	6.8	1.4									

Table 5.5 Lane formation model's 2nd behavior's 1st filter's range analysis for hallway density metric

		Lower bound of filter range									
		0	6.8	13.5	20.3	27	33.8	40.5	47.3	54	60.8
Upper bound of filter range	67.5	1.4	1.2	1.2	1.1	1.3	1.1	1.2	1.1	1.0	1.1
	60.8	1.4	1.2	1.2	1.1	1.1	1.0	1.1	1.0	1.0	
	54	1.3	1.1	1.1	1.1	1.0	1.1	1.0	1.0		
	47.3	1.2	1.0	1.0	1.0	1.1	1.0	1.0			
	40.5	1.2	1.0	1.1	1.1	1.0	1.0				
	33.8	1.3	1.2	1.0	1.1	1.0					
	27	1.0	1.0	1.1	1.0						
	20.3	1.0	1.0	1.0							
	13.5	0.9	1.1								
	6.8	0.9									

The combination methods were fixed by the behavior restrictions; now is the time to evaluate these choices. Table 5-6 shows that for the most part the combination method does not matter. Note that the combination method couldn't possibly matter for the third behavior, heading towards the goal zone, because each agent has only 1 goal zone. It is included in the table for completeness. In the first behavior the combination method doesn't matter, because at most only 1 wall zone will satisfy this behaviors filter at a time. The combination method for the second

behavior is the only one that matters. Agents must select the closest agent (that passes the filters) in the second behavior or it doesn't work.

Table 5.6 Lane formation model combination method analysis

behavior 1				
combination method	**total Fitness**	**travel time**	**hallway density**	**direction**
average	0.8	0.9	0.9	1.0
random	0.9	1.0	0.9	1.0
closest	0.9	1.0	0.9	1.0
furthest	0.8	0.9	0.9	1.0
behavior 2				
average	1.4	1.3	1.0	1.0
random	1.4	1.4	1.0	1.0
closest	0.9	1.0	0.9	1.0
furthest	1.5	1.4	1.0	1.0
behavior 3				
average	0.9	1.0	0.9	1.0
random	0.9	1.0	0.9	1.0
closest	0.9	1.0	0.9	1.0
furthest	0.8	0.9	0.9	1.0

The final step is to evaluate the sensitivity of the model to various offset values. Since the travel time is the most important metric I will focus on that. The original offset for the first behavior was 180° and as figure 5-6 shows as long as the absolute value of the offset is about 90° or larger the model is fit. This is consistent with results from chapter 4 that show that avoidance behaviors aren't very sensitive to the exact offset value. The second behavior also has an offset of 180° and it is also fit so long as the absolute value of the offset is greater than 90°. The third behavior's original offset was 0°. Unlike the other two behaviors, this behavior's fitness is more narrowly focused around the original offset value.

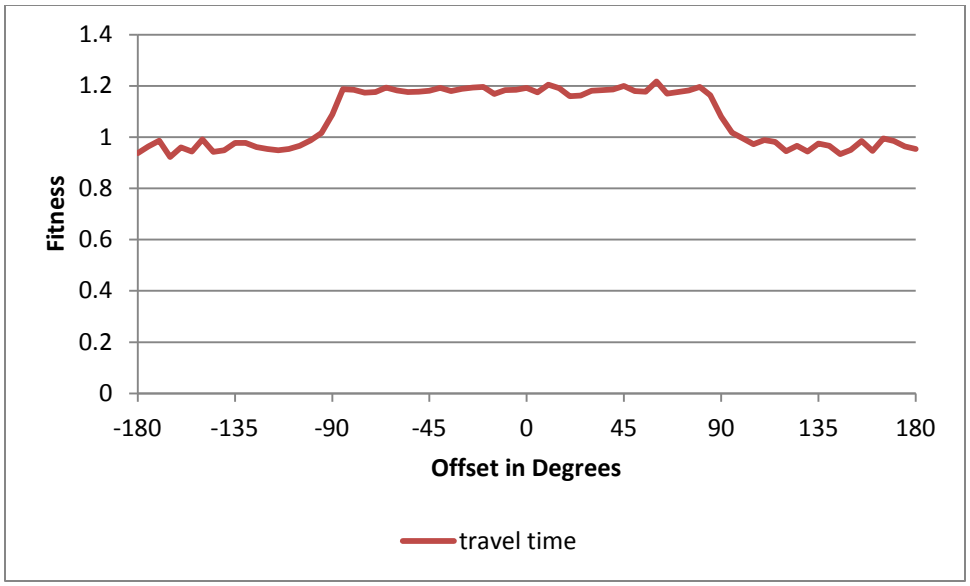

Figure 5-6 Lane formation model's 1st behavior's offset sensitivity analysis

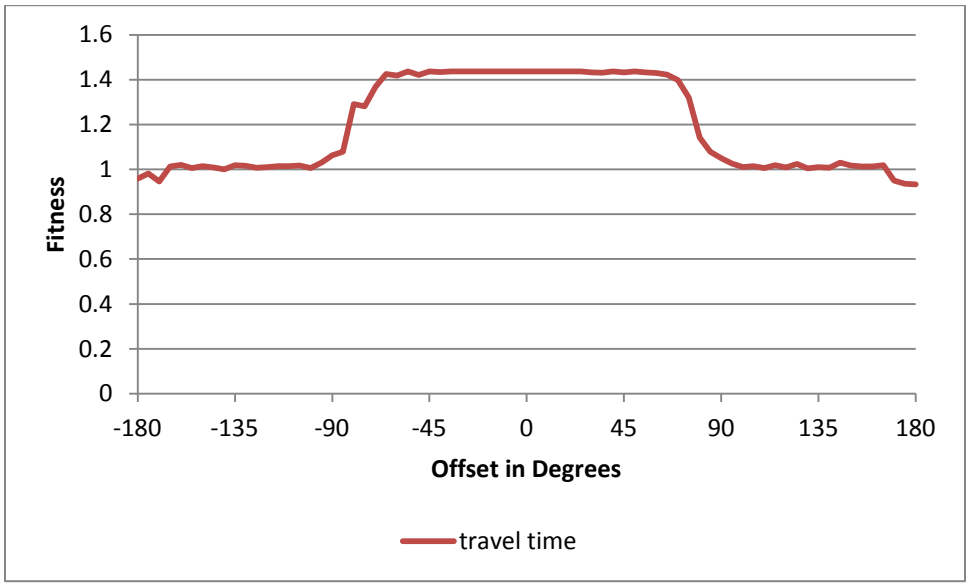

Figure 5-7 Lane formation model's 2nd behavior's offset sensitivity analysis

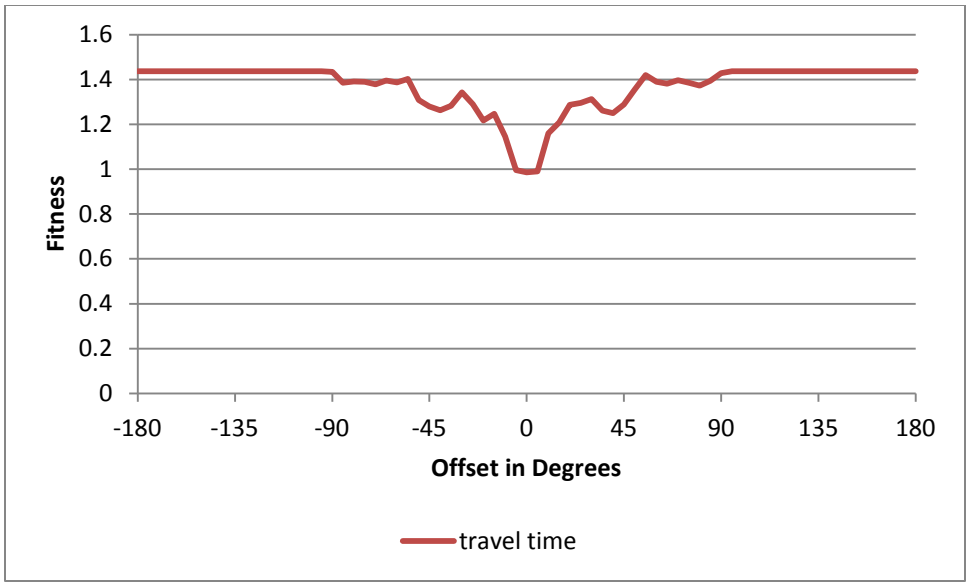

Figure 5-8 Lane formation model's 3rd behavior's offset sensitivity analysis

This chapter demonstrates that it is possible to use SES to incorporate incomplete knowledge into the model search process. Furthermore, this chapter shows how the framework can be used to discover models that exhibit complex behavior such as the lane formation model presented.

6 CONCLUSION

The core contribution of the framework is that it can automatically discover comprehensible mobile agent models using incomplete (or no) knowledge. Data-driven modeling methods are automatic, but the resulting predictive models provide no insight into how a system works, because they are incomprehensible. Knowledge-driven models are easy to interpret, but they must be handcrafted and the resulting models are biased. This framework is the first step towards a modeling method that has the comprehensibility of knowledge-driven modeling and the automation and lack of bias of data-driven modeling.

The results demonstrate that it is possible to discover, understand, and modify a variety of models. The formal analysis of discovered models is just as important as their discovery. Only well understood models can be re-used and modified. Of particular significance is the discovered lane formation model because it demonstrates searching for a model using incomplete knowledge, which has the potential to drastically reduce the model search time.

The next step is to focus on a specific application of mobile agent modeling. It is only by moving from a general mobile agent modeling framework to a specific application that the modeling method can be improved by revealing shortcomings. One of the biggest applications for my framework is swarm robots. The framework currently discovers behaviors that control virtual agents; however, this could easily be extended to discover behaviors used to control real agents. Simulations could be used to test and discover agent behaviors before the behavior software is uploaded to real robots. Simulations are already used to design swarm robot behaviors; however, the current behavior design process is ad-hoc and trial-and-error based. In the remainder of this chapter I will focus on how my framework could be applied to the swarm robot domain. In the process of doing this I will also touch on ways in which my framework could be improved.

A robotic swarm is made up of many robots (10's to 1,000's) that work together to accomplish a goal. By spatially distributing themselves a swarm of cheap robots can accomplish goals, such as surveillance and search and rescue, that a larger more expensive robot would have difficulty doing. Swarm robots are frequently envisioned as flying because this allows swarms to cover larger areas. Flying robots don't get in the way of people to the same degree that ground robots do. Additionally, flying robots are less likely to get stuck on some environmental feature. Robotic swarms are potentially more robust than larger, more complex, single robots because if

one swarm member breaks down the swarm can frequently still complete the task with the remaining members.

For robotic swarms to become a reality advances need to be made in multiple areas. First, the physical body of swarm robots needs to be perfected both to increase endurance and physical robustness. Second, the computer vision algorithms used by swarm robots must be both robust and capable of running in real time on cheap hardware that doesn't use a lot of power. Computer vision is needed for navigation and identifying objects of interest. The final challenge is coordinating the actions of a robotic swarm; it is here that my framework can be applied.

Robots in a swarm must coordinate with one another using some algorithm. It isn't practical to individually control each robot. Each swarm member has only a local view of the environment and frequently can't communicate with every swarm member. My framework models autonomous agents that only have a local view of the environment and that collectively perform some action; it's a perfect fit for swarm robots. The concept of observation in the framework could be extended to include two way communication.

Swarm robot communication is non-trivial. Swarm robot communication is limited by environmental features that absorb wireless signals forcing them to have to communicate over a mesh network where moving robots are the nodes. In the adversarial environment that military swarm robots will operate in the future, communication may be intentionally limited to reduce the chance of detection. The enemy may also jam communication signals, which while maybe not cutting off communication completely, may reduce bandwidth and transmission distance. Furthermore, in an adversarial environment robotic swarms have to handle the sudden appearance of signal jamming and take advantage of its sudden disappearance if the source is destroyed (or is having its Windows operating system forcibly updated). Although the

framework needs to be expanded to include the concept of wireless communication, the model discovery approach used by the framework will still work in this expanded model space.

The framework is the best approach to programming the interactions between swarm members. Currently swarm behaviors are handcrafted. This strategy may work for designing behaviors to achieve simple tasks, but this approach is unlikely to scale to designing swarms to complete multiple, at times competing, goals at once. Another issue with handcrafting swarm behavior is that it cannot handle rapidly changing swarm behavior requirements. If the requirements for swarm behavior change, then when the handcrafting software development method is used there will be a long delay before a new version of the software can be deployed. The slight increase in development cost may not matter much to the military, but the delay could have battlefield implications. On future battlefields were both sides have swarms of robots the ability to rapidly adapt a swarm's behavior could be a crucial advantage. My framework speeds up the design of swarm behavior software.

The next step for the development of the framework is to apply it to the military swarm robot domain. This is preferred over incremental theoretical development, because only the harsh realities of a real application can identify with certainty the parts of a framework that need improvement. The focus should be on military and not civilian swarms. One reason is that the military, especially through the Defense Advanced Research Projects Agency (DARPA), can provide more funding. The adversarial environment that the military's robotic swarms will have to operate in makes it harder to design swarm member behaviors, so they would benefit more than civilian swarms from the framework. It is possible to work on military robotic swarms even before Pentagon funding is received by working on the wireless communication issue. I firmly believe that swarm robots is the best application for my research.

REFERENCES

[**1**] Feliciani, Claudio, and Katsuhiro Nishinari. "Empirical analysis of the lane formation process in bidirectional pedestrian flow." *Physical Review E* 94.3 (2016): 032304.

[**2**] Kim, Tag Gon, et al. "System entity structuring and model base management." *IEEE transactions on systems, man, and cybernetics* 20.5 (1990): 1013-1024.

[**3**] Reynolds, Craig W. "Flocks, herds and schools: A distributed behavioral model." *ACM SIGGRAPH computer graphics* 21.4 (1987): 25-34.

[**4**] Pan, Xiaoshan, et al. "A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations." *Ai & Society* 22.2 (2007): 113-132.

[**5**] Luo, Linbo, et al. "Agent-based human behavior modeling for crowd simulation." *Computer Animation and Virtual Worlds* 19.3-4 (2008): 271-281.

[**6**] Guériau, Maxime, et al. "Multi-Agent Dynamic Coupling for Cooperative Vehicles Modeling." *AAAI*. 2015.

[**7**] Brambilla, Manuele, et al. "Swarm robotics: a review from the swarm engineering perspective." *Swarm Intelligence* 7.1 (2013): 1-41.

[**8**] Ren, Chuanjun, Chenghui Yang, and Shiyao Jin. "Agent-based modeling and simulation on emergency evacuation." *Complex Sciences* (2009): 1451-1461.

[**9**] Arel, Itamar, et al. "Reinforcement learning-based multi-agent system for network traffic signal control." *IET Intelligent Transport Systems* 4.2 (2010): 128-135.

[**10**] Lee, Kang Hoon, et al. "Group behavior from video: a data-driven approach to crowd simulation." *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 2007.

[**11**] Boltes, Maik, et al. "Automatic extraction of pedestrian trajectories from video recordings." *Pedestrian and Evacuation Dynamics 2008* (2010): 43-54.

[**12**] Gharavi, Hamid, and Srikanta P. Kumar, eds. "Special issue on sensor networks and applications." (2003): 1151-1256.

[**13**] Siegel, Mel. "The sense-think-act paradigm revisited." *Robotic Sensing, 2003. ROSE'03. 1st International Workshop on*. IEEE, 2003.

[**14**] Srinivas, Mandavilli, and Lalit M. Patnaik. "Genetic algorithms: A survey." *computer* 27.6 (1994): 17-26.

[**15**] Zhong, Jinghui, et al. "Learning behavior patterns from video: a data-driven framework for agent-based crowd modeling." *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[**16**] Zhong, Jinghui, et al. "Learning behavior patterns from video: a data-driven framework for agent-based crowd modeling." *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[**17**] Patil, Sachin, et al. "Directing crowd simulations using navigation fields." *IEEE transactions on visualization and computer graphics* 17.2 (2011): 244-254.

[**18**] Tang, Wenwu, and David A. Bennett. "Agent-based Modeling of Animal Movement: A Review." *Geography Compass* 4.7 (2010): 682-700.

[**19**] Dia, Hussein. "An agent-based approach to modelling driver route choice behaviour under the influence of real-time information." *Transportation Research Part C: Emerging Technologies* 10.5 (2002): 331-349.

[**20**] Kumar, Sujai, and Sugata Mitra. "Self-organizing traffic at a malfunctioning intersection." *Journal of Artificial Societies and Social Simulation* 9.4 (2006).

[**21**] Liu, Shaobo, et al. "An agent-based microscopic pedestrian flow simulation model for pedestrian traffic problems." *IEEE Transactions on Intelligent Transportation Systems* 15.3 (2014): 992-1001.

[**22**] Georgiadis, Patroklos, Dimitrios Vlachos, and Eleftherios Iakovou. "A system dynamics modeling framework for the strategic supply chain management of food chains." *Journal of food engineering* 70.3 (2005): 351-364.

[**23**] Hu, Xiaolin, and Nicholas Keller. "Agent-Based Modeling and Simulation of Child Maltreatment and Child Maltreatment Prevention." *Journal of Artificial Societies and Social Simulation* 18.3 (2015): 6.

[**24**] Torralba, Antonio, Rob Fergus, and William T. Freeman. "80 million tiny images: A large data set for nonparametric object and scene recognition." *IEEE transactions on pattern analysis and machine intelligence* 30.11 (2008): 1958-1970.

[**25**] Banea, Carmen, et al. "Multilingual subjectivity analysis using machine translation." *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2008.

[**26**] Assured Autonomy, DARPA, Solicitation Number: HR001117S0045, Presolicitation

[**27**] Evtimov, Ivan, et al. "Robust Physical-World Attacks on Deep Learning Models." (2017).

[**28**] Hamacher, Horst W., and Stevanus A. Tjandra. "Mathematical modelling of evacuation problems: A state of art." (2001).

[**29**] Ferrara, Emilio, et al. "The rise of social bots." *Communications of the ACM* 59.7 (2016): 96-104.

[**30**] Sugiyama, Yuki, et al. "Traffic jams without bottlenecks—experimental evidence for the physical mechanism of the formation of a jam." *New journal of physics* 10.3 (2008): 033001.

[**31**] Tesla legal document, https://www.tesla.com/about/legal