APPROVAL SHEET

Title of Thesis: A Framework for Predicting and Controlling System-Level Properties of Agent-Based Models

Name of Candidate: Donald P. Miner
PhD in Computer Science,
2010

Thesis and Abstract Approved: _____
Dr. Marie desJardins
Associate Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

Curriculum Vitae

Name: MY-FULL-NAME.

Permanent Address: MY-FULL-ADDRESS.

Degree and date to be conferred: DEGREE-NAME, GRADUATION-MONTH GRADUATION-YEAR.

Date of Birth: MY-BIRTHDATE.

Place of Birth: MY-PLACE-OF-BIRTH.

Secondary Education: MY-HIGH-SCHOOL, MY-HIGH-SCHOOLS-CITY, MY-HIGH-SCHOOLS-STATE.

Collegiate institutions attended:

University of Maryland Baltimore County, DEGREE-NAME MY-MAJOR, GRADUATION-YEAR.
MY-OTHER-DEGREES.

Major: MY-MAJOR.

Minor: MY-MINOR.

Professional publications:

FULL-CITATION-INFORMATION.
FULL-CITATION-INFORMATION.

Professional positions held:

EMPLOYMENT-INFO. (START-DATE – END-DATE).
EMPLOYMENT-INFO. (START-DATE – END-DATE).

ABSTRACT

Title of Thesis: A Framework for Predicting and Controlling System-Level Properties of Agent-Based Models

Donald P. Miner, PhD in Computer Science, 2010

Thesis directed by: Dr. Marie desJardins, Associate Professor
Department of Computer Science and
Electrical Engineering

This is the abstract. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

A Framework for Predicting and Controlling System-Level Properties of Agent-Based Models

by

Donald P. Miner

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science
2010

This is my dedication.

ACKNOWLEDGMENTS

These will be written out later:

- John Way: My excellent high school computer science teacher; he was the first teacher to truly inspire and make me interested in something
- Richard Chang: Unofficial undergrad advisor; Inspired me to work on hard problems with his classes/my undergrad thesis; Indirectly helped me want to pursue graduate school
- Marie: advisor; introducing me to MAS
- Bill Rand: introducing me to NetLogo
- Forrest Stonedahl: working on a similar problem; a conversation which was a turning point in my research focus to ABMs; introduced me to the term 'meta-model'
- Tim Oates: Suggestions dealing with the ML portion of my research
- Undergraduate Researchers (Peter, Kevin, Doug, Nathan?): Helping with researching new domains
- Marc Pickett: Good friend that is always willing to listen to research ideas; Helped me throughout grad school
- Senior grad students who helped me as a young grad student: Adam Anthony, Eric Eaton, Blaz Bulka
- Other supporting CS graduate students: Wes Griffin, Yasaman Haghpanah, Niels Kasch, James MacGlashan, JC Montminy, Sourav M, Patti Ordonez, Soumi Ray, and Brandon Wilson.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION AND MOTIVATION

The behavior of individual agents in an agent-based model (ABM) is typically well understood because the agent's program directly controls its local behaviors. What is typically not understood is how changing these programs' agent-level control parameters affect the observed system-level behaviors of the ABM. The aim of this dissertation is to provide researchers and users of ABMs insight into how these agent-level parameters affect system-level properties. In this dissertation, I discuss a learning framework named the ABM Meta-Modeling Framework (AMF) that I have developed that can be used to predict and control system-level behaviors of agent-based models. With this framework, users can interact with ABMs in terms of intuitive system-level concepts, instead of with agent-level controls that only indirectly affect system-level behaviors.

1.1 Agent-Based Models

Agent-based models are used by scientists to analyze system-level behaviors of complex systems by simulating the system bottom-up. At the bottom of these simulations are individual agents that locally interact with other agents and the environment. All the behavior in an ABM, from agent-level local interactions to system-level behaviors, emerge from these local interactions, which are governed by the individual *agent programs*. ABMs can

be used to understand how changes in individuals' *agent-level parameters* affect *system-level properties*.

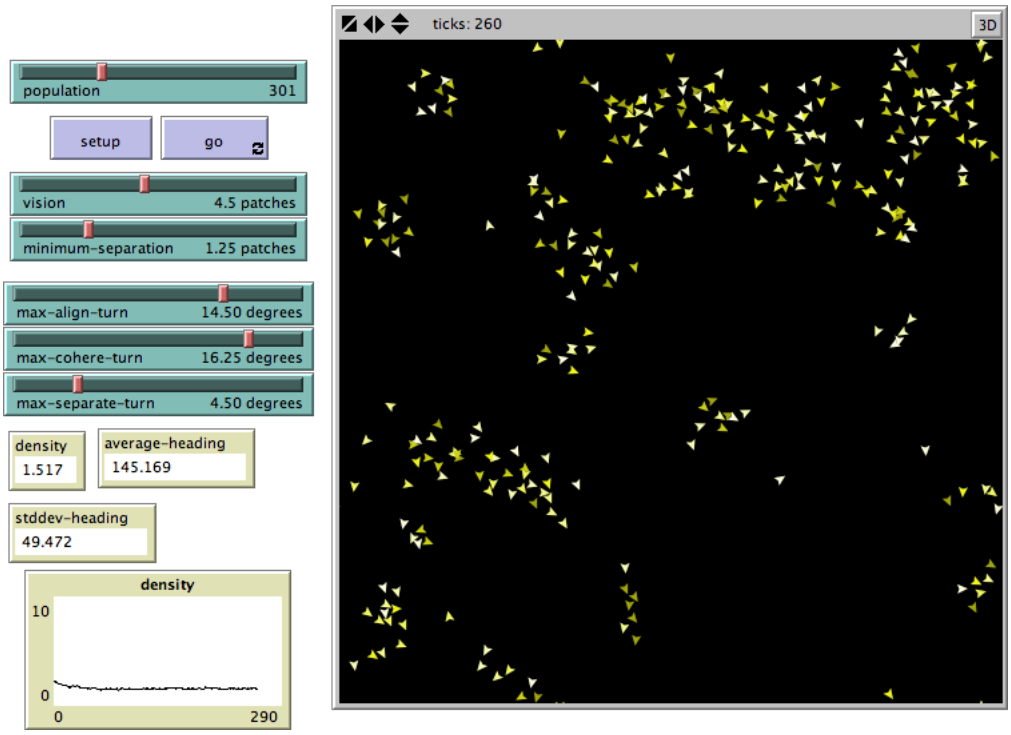


FIG. 1.1. A screenshot of NetLogo's graphical user interface while executing a flocking simulation.

Agent-level control parameters adjust the behaviors of agent-based programs. However, scientists are not typically interested in the local interactions between agents— they are interested in the resulting system-level behaviors that result. For example, researchers that studied agent-based models of lane formations in army ants were interested in the traffic patterns of the lanes, not the individual behaviors of the ants (Couzin & Franks 2003). In other work, researchers that studied locusts were interested in determining at what critical density locusts begin to swarm and destroy crops (Buhl *et al.* 2006). Typically, scientists analyze ABMs by viewing visualizations of the environment or gathering statistical data on the simulation. For instance, NetLogo, an agent-based modeling programming

environment (Tisue & Wilensky 2004), has monitors, plots and visualizations to convey system-level properties to the user. In Figure 1.1, monitors are displaying *density*, *average-heading* and *stddev-heading* statistics for a flocking domain. In addition, a plot of density shows how it has changed over time. These tools are used by a researcher to generate a mental model of how the agent-level control parameters of the flocking domain (the sliders seen in the user interface) affect these system-level properties.

Although using ABMs for researching agent-based systems has been proven useful in a number of domains, there is a glaring conceptual disconnect from the user's perspective, between the agent-level controls and the system-level properties. The classical ABM control method of adjusting agent-level properties is unintuitive because they only indirectly affect the system-level properties through emergence. With the current methodology, a simulation has to be executed in order to observe what the values of the system-level properties will be. A time consuming iterative process of guess-and-check is the only way to configure the system to have it exhibit a desired system-level behavior. A determination of what an ABM will do at a system-level, given only the agent-level parameters, is not possible with current software.

The main goal of the ABM Meta-Modeling Framework is to bridge the gap between agent-level parameters and system-level properties. AMF reduces the learning curve of an agent-based model since users are interacting with the system at the system-level, instead of at the agent-level. Qualitative analysis of an ABM's system-level properties will be a more efficient process since researchers deal with an abstraction of the system's controls. In addition, the models learned by AMF can be inspected to gather quantitative data about the correlations between system-level properties and agent-level parameters.

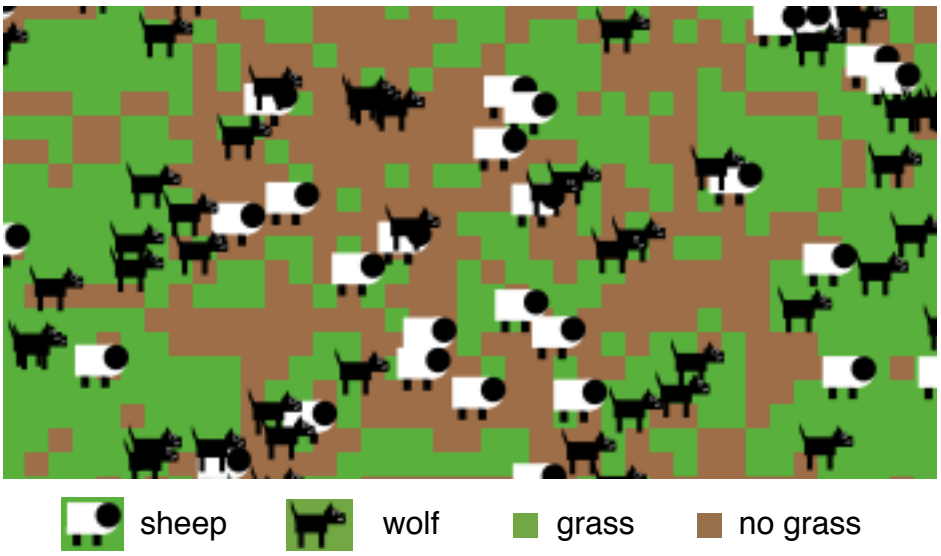


FIG. 1.2. A screen shot from NetLogo's Wolf Sheep Predation model.

1.2 Wolves, Sheep and Grass

Throughout this dissertation, I will use NetLogo's Wolf Sheep Predation model (Wilensky 1997e), which is bundled with NetLogo's standard Model Library,[1] as an example to explain concepts. A snapshot of its NetLogo visualization is shown in Figure 1.2. This multi-agent model simulates a food chain consisting of wolf agents, sheep agents and grass in a two-dimensional space. The model is controlled by seven agent-level control parameters, which directly affect the following agent behaviors:

- The system is initialized with *initial-number-sheep* sheep and *initial-number-wolves* wolves.
- Wolves and sheep move randomly though the space.
- Wolves and sheep die if they run out of energy.

[1] http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation

- Wolves eat sheep if they occupy the same space in the environment. Wolves gain *wolf-gain-from-food* units of energy from eating sheep. The sheep dies.
- Sheep eat grass if they are on a location of the environment that has grass. Sheep gain *sheep-gain-from-food* units of energy from eating grass. The grass dies in that grid location.
- Every time step, each sheep and each wolf has a chance (*sheep-reproduce* and *wolf-reproduce*) to reproduce asexually. Both the parent and the child split the parent's original energy evenly (i.e., parent's energy divided by two).
- Grass regrows after *grass-regrowth-time* number of time steps.

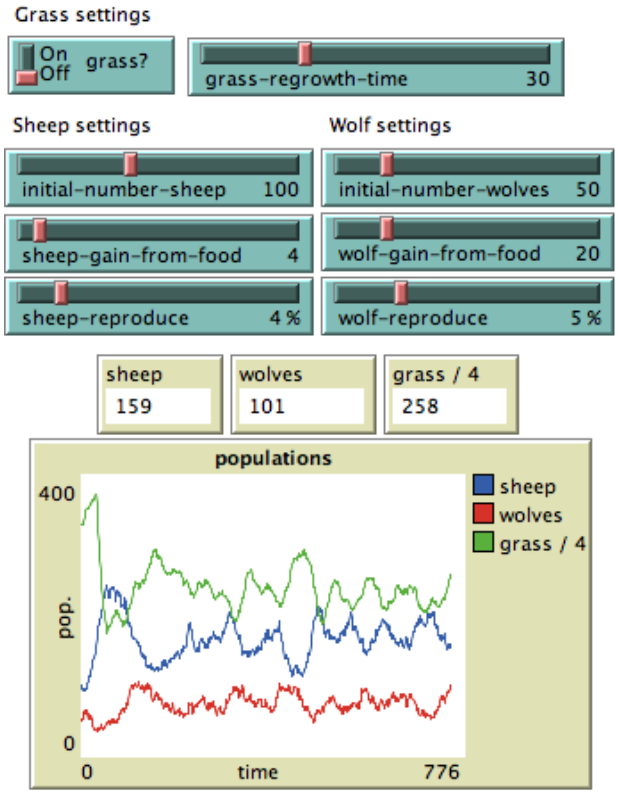


FIG. 1.3. The control and monitor interface for the Wolf Sheep Predation model.

The system-level concepts we are interested in are the number of sheep, the number of wolves and the number of grid locations containing grass. In NetLogo, these properties are displayed with monitors and a plot, as seen in Figure 1.3. The number of each population of agents may change continuously, but the average number of sheep converges. Another interesting feature is some ecosystems fail: either sheep or both sheep and wolves go extinct.

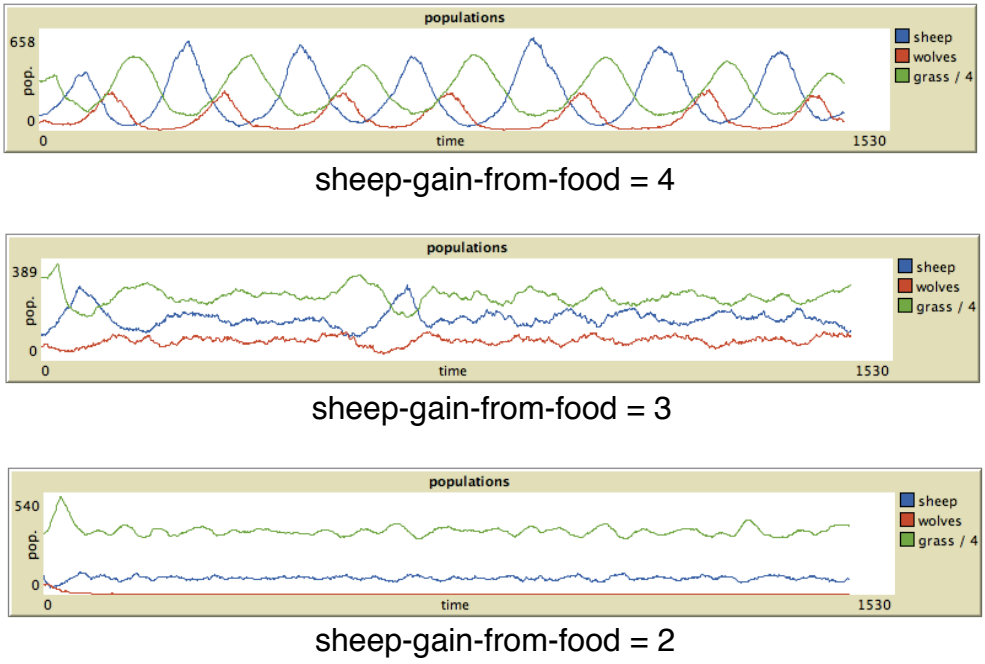


FIG. 1.4. Differences in populations based on changes of the *sheep-gain-from-food* parameter.

After working with this ABM for some time, a user will begin to realize that changes in the control parameters will yield different types of behavior. For example, by setting *sheep-gain-from-food* to 2, 3, and then 4, major differences in system-level behavior are apparent by viewing the graphs in Figure 1.4. When the value of *sheep-gain-from-food* is 4, the system rhythmically exhibits major changes in all three agent populations. When the

value is 2 or 3, the population remains relatively stable, but the average population values are different. When the value is low enough (e.g., 2) the wolves go extinct.

The Wolf Sheep Predation model is a good example of the intuitive disconnect between agent-level parameters and system-level properties. There is no clear *explicit* relationship between the controls presented in the user interface and the resulting system-level properties. An experienced user may have a qualitative understanding of the correlations, but would not be able to predict quantitative concepts, such as the average number of sheep after 2000 time steps. In Chapter 8: Results, I will show that the intuitive disconnect in this domain can easily be solved by AMF.

1.3 Overview of the ABM Meta-Modeling Framework

The foundation of this work is framing the problem of building a meta-model of an ABM as two sub-problems: the *forward-mapping problem* and the *reverse-mapping problem*. In Chapter 6: The Forward-Mapping Problem, I will discuss how AMF maps given values of the agent-level parameters to expected system-level property values with standard regression approaches. In Chapter 7: The Reverse-Mapping Problem, I will discuss how AMF maps a set of desired system-level property values to a set of agent-level parameters that would generate this behavior. My general approach to solving the reverse-mapping problem is to interpolate configurations using the forward mapping to approximate a smooth and continuous surface. This interpolated surface represents the space of configurations that would satisfy the system-level requirements set out by the user. Also, in Chapter 7, I will discuss alternative methods for solving the revers-mapping problem.

AMF is simple and has only a few configuration points. This allows researchers to focus on the analysis of the system, instead of on the details of AMF. The framework consists of three major steps: sampling, solving the forward-mapping problem and solving

the reverse-mapping problem. In these three steps, the only configurations the user must perform are: define how to measure system-level properties of interest, provide the ranges of parameters to be sampled, and plug in a regression algorithm.

I will show in Chapter 8: Results that my framework is able to generate models of system-level behavior. For example, AMF is able to predict the number of sheep and wolves in the Wolf Sheep Predation model, given the configuration parameter values (the forward-mapping problem). Also, AMF is able to make suggestions for the values for the control parameters, given the the desired system-level property outcome (the reverse-mapping problem).

A more comprehensive overview of AMF is provided in Chapter 2: The ABM Meta-Modeling Framework.

1.4 Summary of Contributions

My main contribution presented in this dissertation is an in-depth analysis of meta-models of agent-based models. This analysis includes a discussion of methods for using regression to build models of the correlations between agent-level parameters and system-level properties. In addition, this dissertation contains a survey of ways that that meta-models can be used to inspect system-level behaviors of agent-based models.

The ABM Meta-Modeling Framework encapsulates my methodology for building meta-models of ABMs. The implementation of AMF as software serves as a proof-of-concept to show that my approach is implementable and applicable to a variety of domains. The software itself is a contribution, since it is available to be used by researchers interested in building meta-models of NetLogo ABMs. The design of the general framework is a contribution as well, since it could be implemented to interact with other agent-based modeling systems similar to NetLogo, or totally independent agent-based models.

1.5 Dissertation Organization

This dissertation is divided into nine chapters, including this one. Chapter 2: The ABM Meta-Modeling Framework explains each framework component in detail, explains how a new user would tailor AMF to a new ABM, discusses implementation details and gives an introduction to the forward- and reverse-mapping problems. Chapter 3: Background provides information about NetLogo and algorithms used by AMF. Chapter 4: Related Work compares and contrasts approaches similar to AMF in motivation, with AMF. Chapter 5: Defining System-Level Properties discusses the numerous different ways that system-level properties of ABM can be defined. Chapters 6 and 7 discuss my solutions to the forward- and reverse-mapping problems. Chapter 8: Results evaluates AMF on a domain-by-domain basis and provides explicit examples of how AMF has been used. Chapter 9: Conclusions and Future Work summarizes this dissertation, provides additional thoughts I have regarding this work and possible directions for future work.

Chapter 2

THE ABM META-MODELING FRAMEWORK

The ABM Meta-Modeling Framework builds meta-models that map the values of agent-level control parameter values to system-level property values, and vice versa. Learning these mappings are separate problems, which I call the *forward-mapping problem* and the *reverse-mapping problem*. To solve these problems, a user of AMF "plugs in" a regression algorithm of their choice. The framework uses this regression algorithm to learn the mappings and then provide the user with interfaces to query them. Once the mappings are learned, the user can query either for *prediction* or *control*. A prediction query uses the forward mapping to determine values for the expected values for system-level properties, given the system's configuration. A control query uses the reverse mapping to suggest values for agent-level parameters, given desired values for system-level properties. Most of the implementation details and inner workings of AMF are abstracted away from the user, who only has to attend to a limited number of configuration points.

In this chapter, I will discuss the design goals of AMF, the framework structure, software implementation details, and how well AMF conforms to the design goals.

2.1 Design Goals of The ABM Meta-Modeling Framework

My specific goal in designing AMF was to make controlling and interacting with agent-based models more intuitive. In addition to this central goal, AMF strives to be:

- Domain independent – the design of AMF should minimize the amount of configuration for each new domain,
- Algorithm independent – any regression algorithm should be able to be plugged into AMF,
- Accurate – AMF should generate accurate predictions and control suggestions,
- Fast for the user – interactions with the models generated by AMF should require minimal computational time.

Domain independence is paramount because of the variety in which ABMs come. I have designed AMF such that the same general approach would work for any ABM. Also, I strove to minimize the amount of configuration needed to apply AMF to a new domain. These constraints I have set on the design make AMF broadly applicable to a number of domains, without the need of in-depth domain knowledge. To reinforce this claim, I have tested AMF on a number of diverse domains and used the same general approach for each.

Algorithm independence in a learning framework is important because different algorithms may be more effective for modeling different agent-based models. In general, the learning algorithms that will be discussed in this dissertation will satisfy the requirements for modeling most agent-based models. However, an in depth analysis of which types of algorithms should be used for different classes of ABMs is outside the scope of this dissertation research. In addition, algorithm independence allows AMF to scale with new advances in machine learning research, since future state-of-the-art regression algorithms can be plugged in just as easily as current approaches.

Accuracy and fast user response time appear to be obvious design goals. However, achieving these goals require sacrifices in performance in other portions of the framework. AMF requires a significant amount of computational time to sample different configurations of the target ABM. These large training sets can be used to build static meta-models of ABMs that are both accurate and fast to query. In contrast, an active learning approach would be able to learn models faster, but would require interaction with the user, increasing the amount interaction. Likewise, optimization approaches (e.g., hill climbing) could be used to generate arbitrarily accurate results, but typically require numerous iterations and would significantly increase the response time for a user's query. This is because each step would have to run the ABM to calculate the fitness score, which could take several seconds. In summary, I am making the assumption that users studying ABMs with AMF are more interested in achieving more accurate results for their research and interacting with the models quickly, than spending less time sampling.

2.2 Framework Structure

The framework is split into several phases: sampling, solving the forward-mapping problem, solving the reverse-mapping problem, querying for prediction, and querying for control (i.e., suggesting a configuration). I explain with Figure 2.1 how the different phases interact with one another. Sampling makes observations from the actual ABM and then feeds the newly generated data set to the forward-mapping solver. The result of solving the forward mapping is a function f, which is used to predict behavior and to develop the reverse mapping f^{-1}. From an exterior perspective, the framework only has a limited about of input and output: AMF takes in observations of an agent-based model and then provides interfaces to query for prediction and control. These queries interact with the user, as well as the agent-based model.

Many configuration points exist, which allow users of AMF to modify the behavior of the framework. However, the individual phases take the same output and provide the same output, no matter the configuration. This is what makes AMF a framework and not a collection of algorithms. For example, even though different regression algorithms can be used to learn the forward mapping, the forward mapping still provides predictions of how an ABM will behave, from the user's perspective.

2.2.1 Defining System-Level Behavior Properties

System-level properties of an ABM must be defined by the user of AMF. The measurement of a system-level property is a statistical or mathematical calculation based on the state of the ABM over some period of time (or "ticks"). The one assumption made by AMF is that the measurement is *stable*, meaning that as the same configuration is sampled repeatedly, the measurement's value should vary minimally. One way to measure stability is to calculate the standard deviation of measured values of several runs of the sa e system. The need for this assumption, ways to conform to it, and a more detailed explanation of how to define system-level properties is provided in Chapter 5: Defining System-Level Behaviors.

For example, there are several system-level behavior properties we measure in the Wolf Sheep Predation model. Three of the most obvious are the average number of sheep, average number of wolves, and average amount of grass over a significant number of ticks. These are measured by individually summing the number of sheep, wolves, and grass living at each time step and dividing by the number of ticks. Although the number of sheep and wolves change rhythmically, the average values typically converge to a single value after about 10,000 ticks.

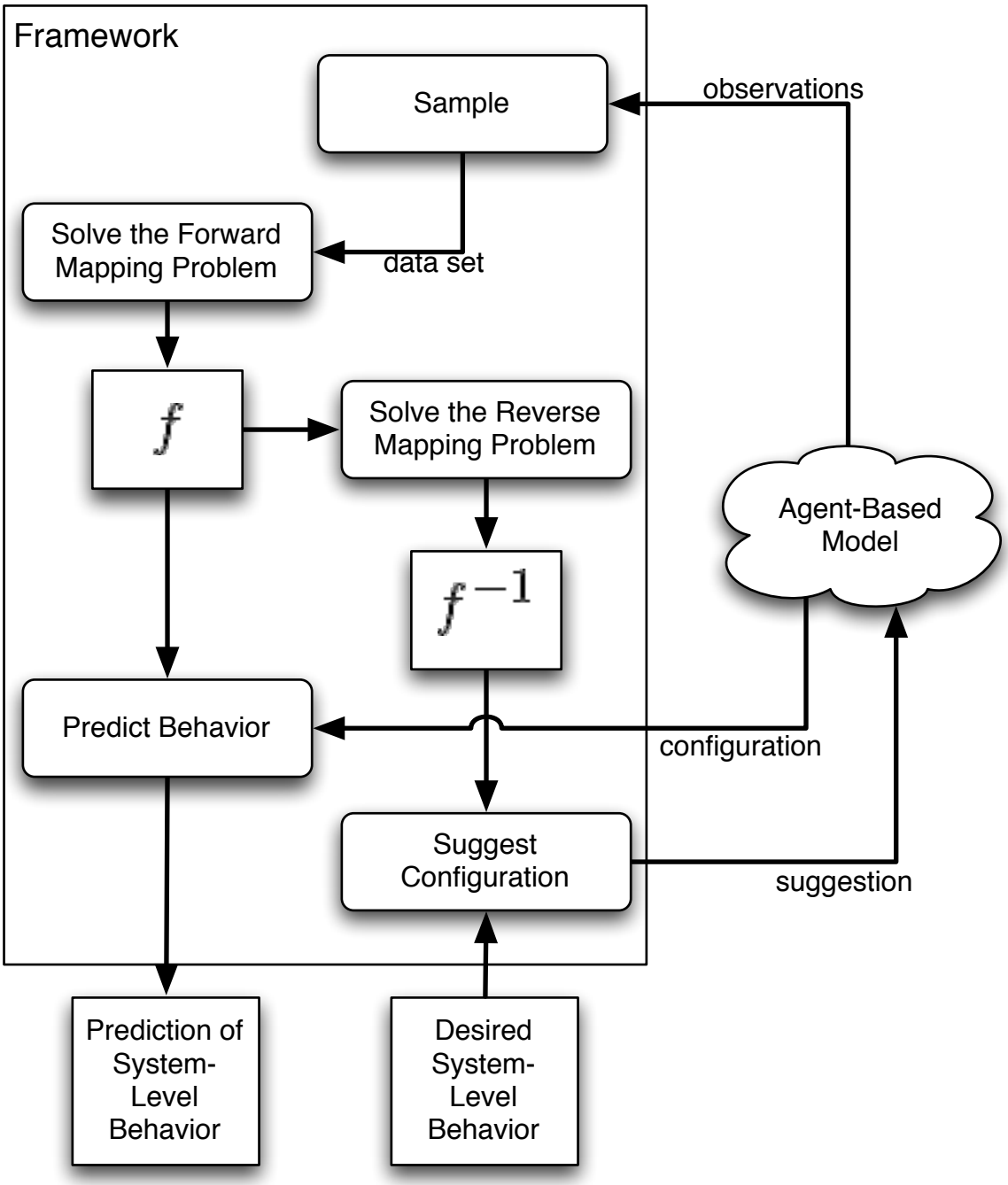


FIG. 2.1. An overview of the phases of AMF and how data flows between them.

2.2.2 Sampling

The first phase is *sampling*. In this phase, numerous observations are made on different configurations of an agent-based model. A data set is generated that contains the independent variables (the agent-level control parameter values) and the resultant system-level behaviors that are measured, for each observation. This process can be significantly time consuming, dependent on several factors:

- Execution time of the model – the models will be executed numerous times, so the longer a model takes to execute, the longer the whole set of experiments will take to execute,
- Granularity – more detailed samplings will take more time, since more points must be sampled,
- Dimensionality – more agent-level control parameters result in a larger search space and naturally more points to sample,

A 120 observation sample of the NetLogo *Fires* model[1] (Wilensky 1997d), requires about thirty seconds[2] to generate. Meanwhile, a large sample of 50,000 observation of a Reynolds boid flock (Reynolds 1987) took approximately four days.

The user is required to have a minimal amount of domain knowledge to specify to AMF which ranges of values should be sampled. AMF is not able to automatically infer which value ranges are interesting, and thus must be explicitly defined.

This sampling process is easily parallelizable. Since each experiment is independent of one another, the set of all experiments can be segmented among a number of systems and processors to significantly reduce the computation time. Once all of the experiments are done executing, the results can be merged into one data set.

[1] See Chapter 8 for detailed results

[2] Most experiments are executed on a 2.2GHz Intel Core 2 Duo running Mac OS X

For the scope of this dissertation, I limit AMF to use a simple random sampling method (i.e., randomly select points within a specified range) or a systematic sampling method (i.e., given ranges, sample evenly spaced points). I acknowledge that an intelligent sampling strategy can be improve the performance of many machine learning techniques, however, none are used for the sake of focusing on other portions of the framework. In future work, different sampling techniques' effect on accuracy and speed could be measured.

2.2.3 The Forward-Mapping Problem

Develop a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that maps a provided configuration vector $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ to several system-level behavior properties $\hat{\mathbf{y}}$:

$$\hat{\mathbf{y}} \leftarrow f(\mathbf{x})$$

The set of values $\mathbf{x}$ consists of all the agent-level parameters (independent variables) in the data set provided by the sample step. An individual mapping is learned for each system-level behavior measurement provided by the user.

This problem is solved with straightforward regression, such as k-nearest neighbor. In this phase of AMF, the regression algorithm is "initialized," if necessary. For example, linear regression would need to solve the least-squares problem. Meanwhile, an algorithm like k-nearest neighbor would not need to initialize anything.

Once this phase is completed, the user is presented with f, an interface to the mapping built by the regression algorithm. The forward mapping is primarily used to *predict* what the system-level property values will be, given the system's configuration. For example, a learned mapping could be used to determine the average number of sheep given a configuration vector, without having to run the system. Some sample predictions, given particular system configurations, are shown in Table 2.1. The values for a system configuration rep-

resent *grass-regrowth-time*, *sheep-gain-from-food*, *wolf-gain-from-food*, *sheep-reproduce*, and *wolf-reproduce*, respectively.

Table 2.1. Sample Predictions of Behavior in the Wolf Sheep Predation Model

Configuration	Average # Sheep	Average # Wolves	Average # Grass
$(30, 4, 20, 4, 5)$	162.8	76.1	964.8
$(30, 3, 26, 7, 5)$	122.8	90.4	1135.2
$(14, 3, 26, 7, 5)$	144.3	163.8	1621.4
$(5, 3, 17, 7, 5)$	946.9	3.4	1065.4

A more in depth definition of the forward-mapping problem, specific examples of regression methods used in AMF, and the role of regression is given in Chapter 6: The Forward-Mapping Problem.

2.2.4 The Reverse-Mapping Problem

Produce a mapping $f^{-1} : \mathbb{R} \rightarrow \hat{S}$ from a given system-level behavior properties $\mathbf{y}$ to a set of configurations $\hat{S} = \{\hat{\mathbf{x}} | f(\hat{\mathbf{x}}) = \mathbf{y}\}$ (i.e., S is the set of behaviors that will produce behavior $\mathbf{y}$):

$$\hat{S} \leftarrow f^{-1}(\mathbf{y})$$

The problem of developing the mapping f^{-1} is what I call an "inverted regression problem" and has many unique challenges. The main challenge is f^{-1} does not describe a functional mapping. This is because f^{-1} describes a one-to-many relationship, since f is many-to-one. Therefore, AMF cannot use standard regression techniques to solve this problem. Instead, the default behavior of AMF is to approximate the inverse of the forward mapping, with a method that I developed. This approach has the benefit of using the forward mapping to develop the reverse mapping, so no additional input from the user

or data set are needed.

The reverse mapping can be used to suggest a system configuration that will exhibit a specific system-level properties. I call this process *control* of the ABM, since it is controlling the ABM at a system-level by suggesting configurations. For example, the reverse mapping could be used to suggest a configuration of $(30, 4, 20, 4, 5)$ for a desired system-level property of 162.8 average sheep (see Table 2.1). However, using the reverse mapping is more involved than using the forward mapping, because f^{-1} returns a set of possible solutions, not a singular suggestion. If the mapping is to be used for control, any configuration in this set will satisfy the system-level requirements. Therefore, an additional step must be taken to extract a point from the set if it is to be used to control an ABM.

The implementation details of developing reverse mappings and how to use the reverse mappings are discussed in Chapter 7: The Reverse-Mapping Problem.

2.2.5 Summary of Configuration Points

The following is a summary of configuration points discussed in this section. For each new domain, the user must specify the following:

- A list of agent-level control parameters, and within what ranges they should be sampled,
- A list of system-level behavior properties and the way to measure them.

In addition, the user may configure the following if they wish to stray from the defaults:

- The sampling strategy (default: random sampling),
- The regression algorithm to be used for the forward mapping (default: k-nearest neighbor),

- The method for the reverse mapping (default: approximate the inverse of the forward mapping).

2.3 Software Implementation Details

So far, I have discussed AMF abstractly and have avoided specific implementation details. This is because the framework is a methodology and could be reimplemented in a number of different ways or to work with a number of different ABM simulation systems. In this section, I discuss the details of my proof-of-concept implementation that was used to run many of the experiments documented Chapter 8: Results.

My implementation works directly with NetLogo[3] to sample, predict and control NetLogo ABMs. All interactions with NetLogo, such as running experiments, retrieving a current configuration or pushing a suggested configuration are handled with Java through NetLogo's Java API. The rest of the framework is implemented in Python. Each step of AMF returns its result as a file, so that it can be passed to the next step or saved for later use. For example, the forward mapping writes the model to a file so that it can be used by both the prediction script and the reverse mapping script.

Since each domain has different variable names and nuaces, the user must implement a Java program that interacts with NetLogo's API. An abstract base class for interaction is provided to guide the development of this program. Next, the user lists the configuration parameters and the ranges of these values so that the sampling can begin. Measuring the system-level parameters can either be calculated within NetLogo or in Java. For example, I modified the standard Wolf Sheep Predation model to keep track of the number of sheep at each tick. Then, to extract the value, NetLogo's API is used to retrieve the sum of the sheep divided by the number of ticks. A similar calculation could be performed within

[3]More about NetLogo is covered in Chapter 3: Background.

Java program by retrieving each piece individually, then performing statistics on them. Performing the statistics in Java have the benefit of not having to change the NetLogo ABM source code. The data set is written to a file, row by row. Therefore, the results of several instances of the sampling, executing on different machines, can be easily concatenated. An example of a sampling program for Wolf Sheep predation is given in Appendix A.1.

The regression algorithms used with AMF must conform to a standard API[4] and are passed as arguments into the forward, reverse mapping and prediction scripts. Therefore, no configuration of these core scripts is needed, since they are "pluggable."

An overarching "master script," written in Python, runs all steps automatically, limiting the amount of direct interaction with the framework software.

In addition to the core framework software, I have developed a toolkit that queries the models generated by the forward- and reverse-mapping models. The core tools include:

- Query the forward mapping for a prediction by passing in a configuration,
- Query the reverse mapping for a set of possible solutions by passing in a desired system-level behavior configuration,
- Visualize the mappings.

2.4 Analysis of AMF vs. the Design Goals

In summary, I align the actual implementation of AMF with the design goals.

Domain independence— My framework's implementation is domain independent, to an extent. AMF reduces the forward-mapping problem to a classical regression problem of learning the correlation between the agent-level parameters and the system-level properties. My reverse-mapping problem solution is also domain independent because it interacts

[4]The specification for the standard regression algorithm interface is given in Appendix B.

exclusively with the forward mapping, which is domain independent.. However, my implementation of AMF still requires the user to specify the agent-level parameters and how to measure the system-level properties. This is a reasonable, since automatically detecting configuration points and having a computer determine behaviors of interest in an ABM would be a challenging unsupervised learning problem.

Algorithm independence— Any regression algorithm can be plugged into my implementation, as long as they conform to the standard API. A simple Python wrapper can be written to adapt an already existing third party regression algorithm to work with my implementation of AMF. My default process of inverted regression requires nothing special of the regression algorithm, because the inversion learning process uses the algorithm's standard forward-mapping behavior.

Accuracy— No extra error is incurred by AMF, itself; the accuracy of AMF depends on the regression algorithm used and the amount of time spent sampling. If AMF is not providing accurate results with state of the art regression algorithms, either the correlations cannot be learned with current technology or not enough time has been spent sampling.

Fast for the user— Most computation time is spent sampling and learning the models. These operations are offline and do not affect the response time of real-time interaction with the user. The response time of the forward mapping and the reverse mapping are proportional to the running time of the regression algorithms, bur are often require less than a few seconds.

Chapter 3

BACKGROUND

The purpose of this chapter is to provide necessary background information for understanding concepts related to ABM Meta-Modeling Framework. In contrast to Chapter 4: Related Work, the previous research presented in this chapter do not share my motivation in AMF, but are a foundations for which AMF is built upon.

AMF uses preexisting research in three major areas: agent-based modeling, regression and surface-to-surface intersection. The framework is targeted at predicting and controlling behavior in an agent-based model (ABM). The next section discusses what an ABM is, discusses some examples of ABMs, and lists some existing multi-agent software frameworks. In the following section, I define regression, outline favorable and unfavorable properties, and outline a number of regression algorithms that have been used by AMF. Finally, I discuss surface-to-surface intersection, which is used for solving the reverse-mapping problem.

3.1 Agent-Based Modeling

Agent-based models or computer simulations that are mainly implemented from the agent perspective. Agents in ABMs are typically:

- bounded by a limited global view,

- perform local interactions, affecting their local environment and neighboring agents,
- are autonomous (i.e., not following some top-down control), and
- heterogeneous (i.e., agents within the system can have different properties) (Epstein 1999).

This paradigm is opposed to implementing the system from the "observer" perspective, in which an overarching control system dictates what agents are to do. For example, in NetLogo, an agent moves forward two units with the following command:

```
> fd 2
```

Notice that this code is agnostic to the global direction of the agent or the position of the agent in the environment. The code simply tells the agent to move forward two units. In contrast, if this were to done from the observer perspective in NetLogo, the following code would be necessary:

```
> set [xcor] of turtle 0
([xcor] of turtle 0 + 2 * cos([heading] of turtle 0))
> set [ycor] of turtle 0
([ycor] of turtle 0 + 2 * sin([heading] of turtle 0))
```

This code computes the change in the x-coordinate and the y-coordinate, given that the agent should move 2 units. Then, it adds the result to the agent's original x- and y-coordinates. Finally, it sets this turtle's x- and y-coordinates to the new x- and y-coordinates. This process is cumbersome. The agent-based property of models allows for an intuitive way to build multi-agent systems.

One of the common uses of ABMs is to discover which local interactions generate a given emergent behavior of a system through experimentation. The research question is posed well by Epstein as the *Generativist's Question*:

> How could the decentralized local interactions of heterogeneous autonomous agents generate the given regularity?

To answer this question, Epstein then poses the *Generativist's Experiment*:

> Situate an initial population of autonomous heterogeneous agents in a relevant spatial environment; allow them to interact according to simple local rules, and thereby generate–or "grow"–the macroscopic regularity from the bottom up. (Epstein 1999)

If a model accurately generates the emergent behavior of the target system, then that ABM could explain why that behavior emerges. Agent-based models are particularly well-suited to answer the Generativist's question.

Implementing systems as an ABM could be more natural than other approaches when the system cannot be defined in an aggregate manner or when individual behavior is complex (Bonabeau 2002). ABMs are often compared to equation-based modeling (EBM), in which the model is a parametrized system of equations that describe system-level behavior (i.e., the observer perspective). For example, a supply chain management system could be modeled as an ABM or an EBM (Parunak, Savit, & Riolo 1998). In the ABM proposed by Parunak et al., individual agents represent different companies that trade with one another. In contrast, the EBM is a series of ordinary differential equations describing the input and output of different components of the supply network. The authors argue that in this situation, an ABM would be more appropriate.

Modeling certain types of behavior, such as changes in state, erratic behavior and local interactions, can prove to be difficult with EBMs. In my NetLogo example above where the agent moves forward, a relatively simple task of adjusting the position of an agent proved to be overly complicated. When more detailed operations need to be performed, the complexity of the observer-perspective model increases drastically. Agent-based models

remedy this by simply changing the context of the programming.

The concept of ABMs can be extended to other uses beyond modeling. Many swarm intelligence techniques, such as particle swarm optimization (Kennedy & Eberhart 1995) and ant colony optimization (Dorigo 2004), use a decentralized agent-based approach to solve optimization problems. Although they are inspired by naturally occurring phenomena, their purpose is entirely separate.

3.1.1 Examples of Agent-Based Models

ABMs have been used to model a variety of different systems. In this subsection, I outline a number of agent-based models that have been developed by other researchers in the past. These examples will illustrate uses for ABMs and how they can be used to answer research questions.

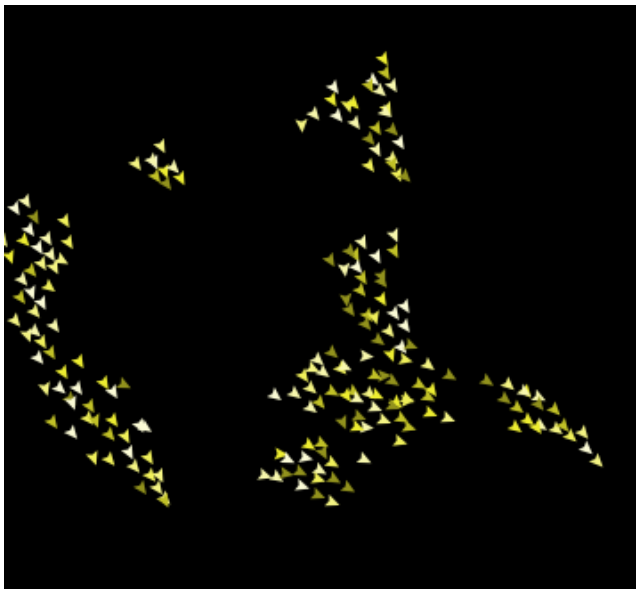


FIG. 3.1. A boid flock moving through a two-dimensional space.

Reynolds Boid Flocking One of the first popular ABMs was the boid flock (Reynolds 1987)(Reynolds 1999), in which agents move through an environment in a similar way to how birds flock. A screen shot from NetLogo's boid implementation (Wilensky 1997c) is shown in Figure 3.1. Agents follow three simple rules:

- Move away from other agents to avoid collisions,
- Align to move in the same direction as nearby agents, and
- Move towards the central position of local flockmates.

These simple agent-level behaviors result in the elegant flocking behavior observed from a top-down view of the system. This system naturally lends itself to being implemented as an agent-based model because the agents act autonomously and have local interactions with only their neighbors. The flocking behavior is emergent and therefore would be more difficult to program directly (e.g., with equation-based modeling).

Boid flocking has been used in a number of applications, such as visualizing time-varying data (Moere 2004), clustering documents (Cui, Gao, & Potok 2006), controlling unmanned air vehicles (Crowther & Riviere 2003) and art (Boyd, Hushlak, & Jacob 2004).

Boids has remained one of my core inspirations for developing AMF. The system is simple, yet exhibits a number of seemingly unpredictable system-level behaviors.

Social Insect Behavior A number of different computational biology projects have aimed at modeling insect societies. In modeling these systems, researchers are able to gain insights into how individual agent behaviors affect the system-wide emergent behavior. These models are typically based on observations of the real insects.

An agent-based model approach has been used to model how army ants form traffic lanes (Couzin & Franks 2003). The particular species of army ants discussed in this work

form two-way highways to reduce the amount of head-on collisions while ants are searching for food and returning food. The authors show how the turning rates and perception of the individual ants affect the efficiency of the ants. They found that when the turning rate is too high, ants are too willing to steer off course and intersect the path of other ants. On the other hand, when the turning rate is too low, ants will not adjust their heading to avoid head on collisions. By tuning their model to optimize the highway performance, they reached an accurate model of the traffic flow in army ants. The result of this research is an accurate model that describes the individual behavior of ants in a traffic situation.

With a different species of ants, researchers were able to produce an agent-based model that simulates the process of an ant colony that collectively selects a location for a new nest (Pratt *et al.* 2005). This process is interesting because an entire ant colony converges on one location, even though many of the ants only have scouted one nest site. Agents are implemented as state machines, in which agents are either exploring for a site, assessing a site, canvassing a site or committed to a site. In each of these phases, the agent performs different actions and at any time may reject the current site and begin exploring once again. Over time, all ants converge on a single site as a nest. With this model, the authors showed that a colony-level decision can be made by ants following agent-based rules.

An ABM has been developed to simulate swarming locusts (Buhl *et al.* 2006). Locusts have an interesting property in that when they are isolated, they tend to not stray from their current location. However, when locusts are among several other locusts, they begin to "march," travelling from one area to another, consuming everything in their path. The authors were particularly interested in determining at what density the locusts begin to march. By changing the number of simulated locusts in a confined space, they were able to determine at what critical density they would march. This simulation approach is far more practical than experimenting with real locusts. The authors suggest that their models

could be used to predict when locust swarms will occur to help warn farmers to protect their crops.

These particular studies in insect behavior illustrate the need for a system like AMF. Researchers in this field are trying to build theories of how an agent's programming can affect system-level properties. The authors perform empirical studies to answer their research questions and test their hypotheses. However, this approach lacks a robust mathematical understanding of the behavior correlations. Researchers would be able to answer their questions through analytical means provided by a domain-independent approach like AMF to reduce the amount of time spent tailoring specific experiments. More on this comparison this particular type of ABM experimentation and the approach taken by AMF is provided in Section 4.1.

3.1.2 Models of Human Societies

Although human behavior is more complicated than insect behavior, local human interactions can often be generalized to build accurate models of a subset of human society. Most of these aim to create an accurate as possible model of the human interactions with others in order to predict some society-level behavior.

Agent-based models have been used to determine how locals would react to an incident at the Pacific Missile Range Facility (PMRF) in Hawaii (Zanbaka, HandUber, & Saunders-Newton 2009). The researchers used census data to represent each islander as an individual agent that either has a positive sentiment or a negative sentiment towards the missile facility. Agents can either change its sentiment towards PMRF by either interacting with another agent that has a different sentiment or experiencing a local event. The model simulates local events and agent interactions and displays how either positive or negative sentiment propagates throughout the island. With this model, the authors were able to model hypothetical situations and how they effect the sentiment on the island.

EpiSims is an agent-based simulation tool that models disease outbreaks (Eubank *et al.* 2004). It uses estimates of how diseases are transmitted and how humans interact based on census and land-use data to realistically simulate the human society. The system simulates individuals going to work and shopping, which exposes them to the disease and exposes the disease to others. With this system, researchers are able to predict the effectiveness of mass and targeted vaccination strategies, given a particular community.

STREETS is an agent-based model of pedestrian traffic (Schelhorn *et al.* 1999). Pedestrian traffic is affected by two major aspects: the layout of the street network and the location of attractions. Instead of trying to analyze the behavior of the system from this data alone, STREETS simulates people to determine a number of properties of pedestrian traffic patterns. STREETS initializes the system with a statistically accurate distribution of individuals across the environment. Next, the agent-based model simulates the movement of agents from their arrival point to and through the urban center. Agents visit buildings and attractions and walk between them. Researchers use this tool to observe a top-level view that can be used to analyze the effectiveness of the urban center layout. Hypothetical modifications to the environment can be performed in simulation to determine if it would improve the traffic situation.

In these simulations, one of the main goals is to model human society so that hypothetical changes can be made to see the changes in the behavior of the system. This way, the researchers don't have to use real humans in social experiments. AMF could be helpful in answering this type of research problem. A desired system-level property could be used to determine suitable configurations by solving the reverse-mapping problem.

3.1.3 Multi-Agent Software Frameworks

Different implementations of different multi-agent systems have many similarities. To reduce the amount of "boiler plate" code for each new multi-agent system, a number of

unique multi-agent software frameworks have been developed in the past two decades. In this section, I survey the most popular of these framewworks. AMF could be configured to analyze ABMs implemented in any of these frameworks.

The Swarm Simulation System Swarm is one of the original agent-based modeling platforms, originally developed at the Santa Fe Institute in the 1990s. Swarm is currently supported and hosted[1] by an independent organization called the Swarm Development Group. The developers' motivation were to enable researchers to focus less on implementation and more on actual experimentation (Minar *et al.* 1996). To achieve this goal, they implemented a number of libraries in Objective-C and Java that aid in the creation of "swarm" objects, the building block of a Swarm simulation. The swarm objects manage the agents and handle the time schedulers that queue the order of agent actions. Also, agents can be nested as hierarchies in swarm objects to allow for heterogeneous collections of agents. Swarm is a discrete time simulation, which means time progresses as actions occur.

Unfortunately, programming in Swarm can require a significant amount of programming overhead, since the libraries are not a single integrated application (Klein 2003). Younger software frameworks attempt to remedy this situation by providing a more cohesive library of tools.

Repast The Recursive Porous Agent Simulation Toolkit (Repast) is a comprehensive agent-based modeling toolkit, focusing on modeling social interactions (Collier 2003). The toolkit is freely available to download.[2] Repast is fully object oriented and attempts to be as platform independent as possible, supporting programming in Java, Python, Visual Basic.Net, and more.

[1]The Swarm website: http://www.swarm.org/
[2]The Repast website: http://repast.sourceforge.net/

Repast's feature set is quite comprehensive and extensible. Repast's features are split into six modules (North, Collier, & Vos 2006):

1. The engine module – controls the agents, environment and scheduler,
2. The logging module – records execution results,
3. The interactive run module – manages user interaction with the model,
4. the batch run module – allows the user to design a series of simulations to be executed in succession,
5. the adaptive behaviors module – provides built-in adaptive agent behaviors that use techniques such as genetic algorithms and neural networks,
6. the domains module – helps define environments, such as social systems, geographic information systems, and computational game theory.

All these modules function together to provide a modular programming environment for a diverse set of multi-agent systems.

Breve Breve is a unique 3D simulation environment that focuses on decentralized systems and artificial life (Klein 2003). Like most other simulation environments, Breve is freely available.[3] The authors of Breve tout that it aims to provide a platform for physically realistic 3D models. The models simulate continuous time, and continuous space, unlike most other toolkits which model time as discrete events and have grid world environments. In addition, some physics such as gravity and object collision resolution with friction are built-in features of every simulation. Breve's OpenGL display engine allows for easy to

[3]Breve website: http://www.spiderland.org/

implement 3D-accelerated graphics. Users of Breve must use a custom object-oriented language called Steve.

One of the motivating applications for building Breve was Sims' (1994) evolved 3D creatures (Klein 2003). In this project, creatures composed of several blocks and joints compete in a game to be closest to a ball. The features of Breve naturally fit to this domain, since it has physically realistic servos and objects that can interact with one another. Due to these features, Breve is an interesting system for modeling individual mobile agents, as well as multiple mobile agents.

MASON MASON is a multi-agent simulation toolkit developed at George Mason University and is freely available.[4] MASON takes a different approach than previous multi-agent toolkits in that it strives to be minimalist and efficient for up to a million agents (Luke *et al.* 2005). It is meant to be ran on a number of back-end computation servers in parallel, without visualization.

MASON does not have any domain-dependent tools or built-in environments like Breve or Repast, leaving most to the user to implement their environments in Java. Although MASON is quite minimalist, it is designed to be extensible so that it can be used as a foundation for new simulation systems.

MASON simulations can be interacted with a separate visualization that binds to the simulation. This is a different paradigm than other toolkits like NetLogo that are tightly coupled with the visualization.

NetLogo NetLogo is a relatively new ABM system, which was started in 1999 out of Northwestern University as a derivative of StarLogo (Tisue & Wilensky 2004). The software is free to download,[5] but not open source. As the name suggests, the language

[4]MASON website: http://cs.gmu.edu/ eclab/projects/mason/
[5]NetLogo website: http://ccl.northwestern.edu/netlogo/

used by NetLogo is a derivative of Logo, a Lisp-like language. An artifact of this language is that agents are referred to as "turtles" in NetLogo.

NetLogo's language is tailored to the agent-based model paradigm. Particularly, a program can change context to an individual and execute code from its point of view. Also, populations of agents can execute these individual actions concurrently. For example, the code to ask all the agents to move forward one unit, then turn right would be:

```
ask turtles [ fd 1 rt 90 ]
```

The other built-in agent type is the "patch." Patches are organized as a grid in a two-dimensional environment that agents are confined to. Users can interact with patches from their context, like the turtles. Also, it is easy to retrieve turtles that are in contact with a patch and to retrieve which patch a turtle is on, from their respective contexts. This makes turtle interactions with the environment simple to implement.

NetLogo also includes a built-in user interface and user interface editor. The interface consists of controls, monitors and the domain visualization. Users define controls which bind to global variables and buttons that bind to function calls. In addition, users can add monitors and plots, which show a variable's value or plot a variable's value over time. These are useful tools in conveying information that the domain visualization cannot. NetLogo can be run "headless" to facilitate parallel execution of models or to reduce run-time.

NetLogo comes with a number of extensions, most notably BehaviorSpace and HubNet. BehaviorSpace is a tool for designing a series of systematic experiments with different system configuration parameters. BehaviorSpace performs a user-defined measurement value after each individual execution and reports the result to a data set in the form of a spreadsheet. HubNet is a server/client tool that allows several users of a system to interact with a NetLogo ABM at the same time, which is useful for classroom instruction.

NetLogo can be interacted with from an external Java API. A NetLogo "workspace"

is instantiated as an object and can be used to send commands and query current variable values. This is useful for an number of tasks. First, sometimes NetLogo's language is not expressive enough and some may find programming in Java more familiar. In addition, external Java libraries (such as machine learning libraries) can be used without modification by integrating with NetLogo from its Java API. Also, a sequential experiment system can be implemented in Java if a user would like to run experiments without BehaviorSpace.

An extensive model library with over 140 sample models is bundled with the NetLogo distribution. These already existing models serve as excellent examples for new models as well as starting points for modified models. Most of the domains discussed later in this dissertation are from this model library.

NetLogo is my ABM system of choice for this dissertation research for a number of reasons. First, NetLogo's learning curve is surprisingly short. From personal experience, a new user can learn to write a domain similar to the Wolf Sheep Predation model in less than a couple hours. In addition, the documentation[6] is well organized and detailed. Second, the model library provides several models that are interesting and easy to work with. Since each of these models are implemented in NetLogo, I was able to implement AMF so that it interacts with each ABM in a similar way. Other multi-agent system toolkits provide more flexibility in implementation, which would make identifying agent-level control parameters and system-level properties more difficult. Third, although NetLogo's platform is well contained, the interactions possible with the Java API make almost anything possible. I was able to implement my framework as an external modular application that interfaces with NetLogo, instead of a built-in system dependent tool. This makes my research easily extensible to other ABM systems in the future.

[6]The NetLogo documentation is available at http://ccl.northwestern.edu/netlogo/docs/

3.2 Regression

Regression is an integral part of this dissertation research, as it is the foundation of solving the forward- and reverse-mapping problems. Regression techniques are used to predict values of dependent variables, given values of the independent variables. The typical approach is to use a model developed from a sample training set to infer new values that of configurations that have not been sampled. Also, regression can be used to smooth the natural error in sampling.

Linear regression is perhaps the simplest form of regression. It fits a line to represent the correlation between one independent variable with one dependent variable, taking the form of:

$$\mathrm{y} = \mathrm{x} + b.$$

There exists a closed-form solution to determine m and b in order to minimize the sum of squares between the training data and the curve.

Unfortunately, linear regression is limited in that it only models linear relationships, and is not able to model nonlinear ones, such as rhythmic oscillations, or quadratic correlations. Two paths are generally taken to address this concern. The first is to use the kernel trick (Muller *et al.* 2001) to map nonlinear data to be linear, but higher dimensional. Then, once the data is projected into a space in which it is linear, linear methods (e.g., linear regression, perceptrons (Minsky & Papert), or support vector regression(Smola & Schölkopf 2004)) can be used.

The second approach is to use nonlinear methods, which relax linear regression to learning a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that satisfies the following equality:

$$y_i = f(\mathbf{x}_i) + \varepsilon_i.$$

In this equation, y_i is a dependent variable and $\mathbf{x}_i$ is the vector of independent variables associated with y_i. Also, there is an assumption of some normally distributed and independent error with each point, denoted by ε_i. The process of *nonparametric* regression is to estimate this function directly (Fox 2002).

Alternatively, a *parametric* regression model has the form of the function $f(\cdot)$ relatively fixed. Instead of learning $f(\cdot)$ directly, a vector of model parameters β are optimized to fit the data (Fox 2002):

$$y_i = f(\beta, \mathbf{x}) + \varepsilon_i.$$

For example, $f(\cdot)$ could model two-dimensional sinusoidal data with the model:

$$f(\beta, (x_1, x_2)) = \beta_1 + \beta_2 \sin(x_1 - \beta_3) + \beta_4 \sin(x_2 - \beta_5)$$

Typically, the value of the least squares metric is minimized to fit the curve to the data.

While developing the ABM Meta-Modeling Framework, I needed regression algorithms that satisfied a number of properties. The regression algorithm should scale well with dimensionality, since agent-based model behavior spaces (i.e., the space of agent-level parameters and system-level properties) are typically highly dimensional. I also needed algorithms that could model nonlinear correlations, since behavior spaces are expansive and rarely maintain linearity throughout the entire space. However, in the ABMs I have experimented with, all spaces are very smooth locally. That is, there is little variance in values between nearby locations in the behavior space. Even in the situation where there is variance, the error is typically evenly distributed and cancels out if the same configuration is sampled a number of times. Therefore, nonlinear parametric regression can fit to the overall behavior space, without having to worry about fine grained variations in the data. Also, the mappings built by the regression models should be smooth. This is important for the tractability of finding the reverse mapping, since bumpy surfaces will naturally yield more

intersections than smooth ones. Therefore, regression methods that smooth the data, such as LOESS, are effective in this situation.

*[**TODO**: an image showing graphs of the fires, boids, and sheep/wolf predation behavior spaces (need to make wolf/sheep data set to do this)]*

In the course of testing AMF, I used four regression techniques: k-nearest neighbor, LOESS, least-squares nonlinear regression, and Multilinear Interpolation.

3.2.1 K-Nearest Neighbor Regression

K-nearest neighbor (KNN), in general, is the process of selecting the k nearest points to some point $\mathbf{x}$. These points can then be used for classification or regression. In the simplest case, each neighbor's y value is averaged to infer a value $\hat{y}$ for the given point $\mathbf{x}$. KNN is easy to implement and accurate with large data sets. Accuracy of this regression can be improved by weighting points by distance or by using a secondary regression algorithm (e.g., linear regression) on the neighbors.

We use KNN in our results as a baseline, since it is easy to implement and accurate. In addition, it is nonparametric, which makes it agnostic the the shape of the behavior space. Therefore, KNN requires little to no configuration or tailoring to a particular domain.

However, when implemented naively, the run-time scales linearly with the size of the data set because the process of finding the nearest neighbors requires a scan of all data points. This search time can be reduced with the implementation of faster searching techniques, such as locality sensitive hashing (Gionis, Indyk, & Motwani 1999), among other methods. Even with faster neighbor locating techniques, some sort of search is required for each query to KNN.

The spaces generated by KNN are not continuous (particularly if the neighbors are not weighted). This is because similar points will have the same neighbors and, in effect, have the same predicted value. There are separations in which on one side the k-nearest

neighbors will include a particular data point, meanwhile on the other side it will not. At this separation, a noncontinuous jump will occur in predicted values. This can be a problem for solving the reverse-mapping problem because the intersection between hyperplanes generated by KNN will not intersect smoothly. This can be remedied by using KNN in conjunction with multilinear interpolation.

3.2.2 Robust Locally Weighted Regression and Smoothing Scatterplots (LOESS)

Robust Locally Weighted Regression and Smoothing Scatterplots (LOESS) is a mature nonparametric regression technique for fitting curves to data (Cleveland 1979)(Cleveland & Devlin 1988). Similar to k-nearest neighbor, LOESS uses local points to infer values of new data points. LOESS also iteratively smooths the regression curve, allowing for variable magnitudes of smoothing. This is important for particularly noisy data sets. The role of LOESS in the testing of AMF is that it is more robust than KNN, yet still nonparametric.

Abstractly, LOESS fits a new $\hat{y}_i$ to every $\mathbf{x}_i$ to replace the original noisy y_i. Then, LOESS iteratively smooths each $\hat{y}_i$ to reduce variability. Specifically, LOESS follows the following steps:

1. Initialization step – for each $\mathbf{x}_i$, weight all other points x_j, where $j = 1...n$, based on a weighting function $W(\mathbf{x}_i, \mathbf{x}_j)$. W weights closer points higher. Perform weighted linear regression to infer $\hat{y}_i$.

2. Smoothing step – For each $\mathbf{x}_i$, weight all other points based on w_i, as well as a new weight function δ, which returns magnitude inversely proportional to $y_i - \hat{y}_i$ (i.e., the larger the distance, the lesser the weight). Perform weighted linear regression to infer $\hat{y}'_i$.

3. Set each $\hat{y}_i$ to $\hat{y}'_i$.

4. Repeat the the smoothing step as many times as necessary.

Values in between xs can be now be interpolated.

LOESS can be customized in a number of ways. The different weighting functions can be changed to change the amount of influence other points give to the linear regression step. Typically, these weighting functions have a "smoothing" parameter, which increases the influence of further away points, effectively reducing the strong influence of local points. Adjusting the smoothing parameter is important because if it is too low, the regression curve will be erratic (i.e., overfit), meanwhile, if the smoothing parameter is too high, some features of the curve may be eliminated.

LOESS is relatively computationally expensive in comparison to other approaches. This is because weights are applied and points are readjusted iteratively a number of times. However, the model can be built offline and then stored. Once the $\hat{y}_i$ values are inferred, they can be used to interpolate new points (similar to multilinear regression) in response to queries. This provides a quick response time to users wanting to infer $\hat{y}$ values. This fits one of the design criteria of AMF: the regression algorithm should be fast for the user.

The *R* statistical package[7] has a built in LOESS function.

3.2.3 Nonlinear Regression

Nonlinear regression (NLR) is a parametric approach to regression. With this approach, a parametric model is provided by the user that represents the relationships in the data, in general. This is typically done with some sort of intuition on behalf of the user. For example, the data may appear sinusoidal or quadratic, in which case a model involving sine curves or quadratic polynomials, respectively, would be appropriate. Similarly to how linear regression will not model nonlinear relationships well, certain nonlinear models will

[7] R website: http://www.r-project.org/

not model other nonlinear relationships. Therefore, accuracy is largely dependent on the selection of the correct model.

Optimization techniques, such as Gauss-Newton or Levenberg-Marquardt (Mor 1977), are used to adjust nonlinear model parameters in order to minimize the least squares metric (Gallant 1975). These processes iteratively move models towards the best fit to the data.

Nonlinear regression is of particular interest in my research because it is analytically invertible, since trained NLR models are already in the form of a function. Unfortunately, since NLR is parametric, it requires a significant amount of configuration between different target domains.

3.2.4 Multilinear Interpolation

Multilinear interpolation is a dynamic approach that uses multi-dimensional interpolation between "knots" to generate a smooth surface across a space of any dimension (Davies 1997). Knots are sampled data points, scattered across the behavior space in a regular fashion such that the knots, when connected, form hypercubes. When a point $\mathbf{x}$ is queried, multidimensional interpolation is performed using the corners of the hypercube to infer the value of $\hat{y}$.

The major downside to multilinear interpolation is that the sampling needs to be systematic and evenly spaced. To remedy this situation, another regression algorithm is used to compile a set of evenly spaced points. These evenly spaced inferred points area then passed to a traditional multilinear interpolation approach.

In this dissertation, I typically use multilinear interpolation as a supplement to other regression approaches. Any irregularly sampled data set can be converted to a evenly spaced one by inferring knot locations with another regression technique. Once this intermediary inferred data set is in place, standard multilinear interpolation can be used.

3.3 Surface Intersection

The surface-to-surface intersection (SSI) problem is the problem of finding the intersection between two surfaces. This is a difficult problem due to the variety of shapes and configurations of high-dimensional spaces. Surface-to-surface intersection algorithms for solving these problems can take a number of forms: lattice evolution methods, marching methods, subdivision methods, and analytic methods (Patrikalakis *et al.* 1993).

TODO: [Give specific examples of SSI algorithms; a more comprehensive literature survey of SSI is needed] (Huber & Barth. 1999)

SSI serves an important purpose in solving the reverse-mapping problem. First, SSI is used to find the space of configurations that would satisfy a desired system parameter. To apply SSI approaches, we frame the problem of solving for $f^{-1}(y_k) = \hat{\mathbf{x}}_k$ (i.e., find $\hat{\mathbf{x}}_k$, given y_k) as an SSI problem. This is done by finding the intersection $\vec{w} : \vec{u} \bigcap \vec{v}$ between the surfaces $\vec{u} : y = y_k$ and $\vec{v} : y = f(\mathbf{x})$. In this case, f is the forward mapping space. The surface $\vec{w}$ can be used to extract valid configurations that will produce the behavior $\hat{y}$. This approach can be expanded to satisfy a number of system-level properties at once. For example, in the Wolf Sheep Predation model, a user might want to find configurations of a system that typically averages 100 sheep and 80 wolves. Intricacies and extensions to this approach are discussed in Chapter 7: The Reverse-Mapping Problem.

Chapter 4

RELATED WORK

4.1 Experimentation in ABMs

Experimental platform for messing around with ABMs: (Bourjot – "A platform for the analysis of artificial self-organized systems" 2004) (relevant?)

ABMs have been used to study behaviors in biological systems... Domain specific work interested in system-level behavior: ant lane formation (Couzin & Franks 2003), marching locusts (Buhl *et al.* 2006), fish schools (Parrish, Viscido, & Grunbaum 2002). Most of these studies are qualitative in nature.

particle swarm optimization: emperical study (Shi & Eberhart 1998); more general approach: (Van den Bergh & Engelbrecht 2006)

4.2 Prediction of System-Level Behavior

physics-based control policy (Spears *et al.* 2004) – similar to our approach, but the models are tightly coupled to the domain. the system was designed with system-level models in mind. Algebraic inversion for inverse mapping is nice. Inspires the nonlinear regression approach in AMF.

Macroscopic models of swarm robot systems (Lerman & Galstyan 2002)(Lerman, Martinoli, & Galstyan 2005) – similar in motivation, but models the system in a more

specific way (FSAs). Specific to systems in which agents can be modeled as FSAs. Our approach is more general, since it just looks at parameters.

4.3 Inversion of Neural Networks

Inversion of neural networks:

- (A Linden and J Kindermann "Inversion of multilayer nets" 1989 – optimization problem solved by gradient descent)
- (Bao-Liang Lu "Inverting Feedforward Neural Networks using Linear and Nonlinear programming" 1990 – formulate the inverse problem as a nonlinear programming problem, a separable programming problem or a linear programming problem)
- (S. Lee and R.M. Kill "Inverse Mapping of continuous functions using local and global information" 1989 – iterative update towards a good solution)
- (Michael I. Jordan work with robot arm "Forward Models: supervised learning with a distal teacher" – asks the question, what configuration of the robot arm will yield this behavior?)

All of these are optimization techniques. They do not return an actual mapping. Also, some of the techniques are restricted to neural networks (and are thus not algorithm-independent).

Chapter 5

DEFINING SYSTEM-LEVEL PROPERTIES

System-level properties play a central role in this dissertation. A system-level property is a mathematical measurement of some non-explicit concept of interest in a agent-based model. These properties are typically non-explicit, which means that inferring their values from just the given agent-level configuration parameters is difficult.

Identifying which system-level properties of an ABM to analyze is the first step of using the ABM Meta-Modeling Framework. Deciding on which features to be measured is a task for the user of the framework and will be influenced by what that user finds interesting or what that user needs to analyzed. Each system-level parameter is defined as a mathematical measurement of the behavior in terms of observable features of the ABM. For example, the average number of sheep in the Wolf Sheep Predation model is a system-level property of the system that is calculated by averaging the number of sheep measured in each time step over the lifetime of the system. The framework utilizes the user provided mathematical definition to create a data set that is used in the forward mapping.

In this chapter, I discuss the problem of defining system-level properties for use in AMF. First, the important "stability assumption" and the consequences of this assumption are explained. Next, a comprehensive list of common system-level property classes are outlined. This list should be useful for users of AMF in defining their own system-level

properties. Then, the process in which AMF uses to sample an ABM is explained. Finally, software implementation details are given for this particular step of AMF.

5.1 The Stability Assumption

Before delving into the topic of defining system-level properties, I first discuss the "stability assumption." Understanding this assumption and its consequences is important in defining system-level properties so that they will be measured accurately by AMF.

AMF assumes that the naturally occurring error for sampled values of a system-level property must have the following feature: the expected mean and expected median of the behaviors should be identical. In other words, the errors should be expected to be evenly distributed around some value $\bar{y}$, in both magnitude and quantity. Errors following a normal distribution will have this property. For example, the percentage of trees burned down in the Fires[1] model will vary from run to run, but for the most part will be centered around one value for a particular configuration.

This assumption is important because if this assumption does not hold, regression algorithms will converge to predict with bias. As more points are sampled, the regression algorithm predictions will converge on the value provided by the mean of measured values. However, the median may be a better predictor of the expected behavior if the errors are not normally distributed. This problem is best explained with an example.

A good example of

In the Wolf Sheep Predation domain, a simple system-level measurement is the average number of wolves after a thousand time steps. This number can be misleading, because certain configurations will *sometimes* result in the wolves going extinct (i.e., zero wolves). Other times, the same configurations will result in the wolves converging to a stable non-

[1] More on the Fires model is discussed in Section 8.1.1

zero population. When the wolf population does stabilize, it converges upon generally the same population value, $\bar{w}$. However, the average of successive runs will result in a biased value between zero and this stable population size. Unfortunately, this average is not useful in any way, since it does not tell how often the wolves will go extinct or what their population will be, should it be the case that the population stabilizes.

To remedy this problem, the average can be decomposed into two components: the case of going extinct and the case of not. The average number of wolves $\bar{w}$ can be decomposed as:

$$\bar{w} = \mathrm{P}(extinct) * (\bar{w}|extinct) + (1 - \mathrm{P}(extinct)) * (\bar{w}|\neg extinct)$$

where $extinct$ is true when the wolves went extinct, $\mathrm{P}(extinct)$ is the probability wolves go extinct, and $(\bar{w}|\neg extinct)$ is the average number of wolves, given they did not go extinct. The value of $(\bar{w}|extinct)$ is zero, so the left hand side of the above equation is zero. Therefore, the equation can be simplified to:

$$\bar{w} = (1 - \mathrm{P}(extinct)) * (\bar{w}|\neg extinct)$$

As stated earlier, $\bar{w}$ is biased because sometimes an experiment with return zero and sometimes it will return $(\bar{w}|\neg extinct)$. However, the values $\mathrm{P}(extinct)$ and $(\bar{w}|\neg extinct)$ are useful in describing the system's behavior and are more stable (and accurate) than just using $\bar{w}$. The system-level properties that should be measured for the Wolf Sheep Predation model is not the average number of wolves, but the average number of wolves given that they did not go extinct, and the probability of the wolves going extinct.

To calculate the value of $\mathrm{P}(extinct)$, divide the number of samples that exhibited zero wolves by the number of total samples for that configuration. To calculate the value of

$(\bar{w}|\neg extinct)$, average the population size of the wolves, only in samples in which they did not go extinct.

The histogram in Figure 5.1 illustrates why $\bar{w}$ is a poor metric for this domain. I measured the average number of wolves post-convergence in 850 runs of the same configuration[2] of the Wolf Sheep Predation model. In this experiment, 303 of the 850 samples resulted in wolves going extinct. The other 547 samples had their wolf populations stabilize. The average number of wolves over all samples $\bar{w}$ is 44.69. The value 44.69 has little meaning in this domain, as it does not tell us about the stable wolf population size, or how often the system will exhibit zero wolves. The estimated probability of extinction is $P(extinct) \approx 303/850 \approx .356$. The average number of wolves in a stable population is $(\bar{w}|\neg extinct) = \sum(w_i)/547 \approx 69.45$. These two values explain the the behavior of this particular Wolf Sheep Predation model instance better than $\bar{w}$.

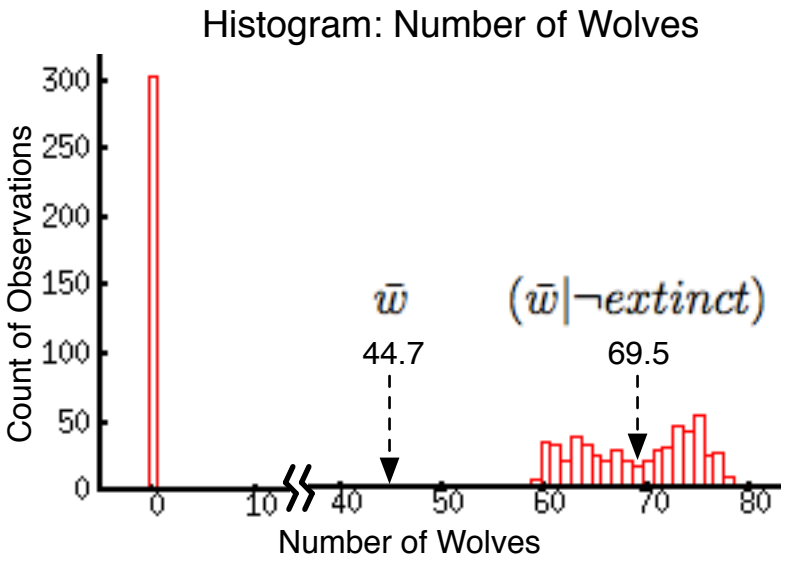


FIG. 5.1. A histogram of the number of wolves after many successive runs shows that the average number of wolves is a poor number to use to describe the behavior.

[2]grass-regrowth-time = 22, initial-number-sheep = 100, initial-number-wolves = 50, sheep-gain-from-food = 4.7, wolf-gain-from-food = 20, sheep-reproduce = 3%, wolf-reproduce = 4%

In most situations, non-stable system-level properties are the result of some sort of *threshold effect*. Threshold effects are sudden changes in behavior, given a small change in the configuration. Threshold effects are also sometimes referred to as *tipping points*. In ABMs that incorporate random behavior, configurations that lay on a boundary of a threshold effect can exhibit erratic behavior, which was the case with the Wolf Sheep Predation example. Typically, this problem can be overcome by decomposing the property into several sub-properties that describe the threshold effect. This approach is what was needed to provide useful and accurate information in the Wolf Sheep Predation example.

5.2 Common Classes of System-Level Properties

Through the course of identifying a number of system-level properties in a variety of domains, I have found that most properties fit into one of these general classes.

5.2.1 Average of a Value

One of the simplest measurements is the average value of a property over the life of the system. This is useful for measuring behavior that converges over time. To measure this property, the property is measured every time step up until a predetermined stopping point, and then averaged.

A few modifications to this simple approach are possible. First, the recording of values can be ignored for a number of time steps to allow the system to converge. If convergence is ignored, the values recorded before convergence may bias the results in an unexpected way. To remedy this, only the points post-convergence are used for the average. The next step is to determine what this "post-convergence" happens. The simplest method is to determine a duration of time in which most model will have converged. Another naive method is to just sample for a very long time. As time passes, the average will converge

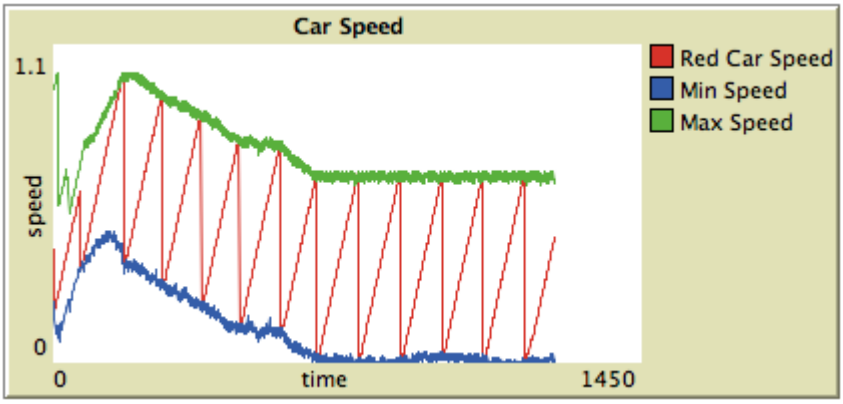


FIG. 5.2. The behavior of the NetLogo Traffic Basic ABM converging after about 650 time steps.

to the convergent value, since more values representing the convergent behavior will factor into the average. A more advanced technique would be to detect when the system values have stopped drastically changing and start measuring from there.

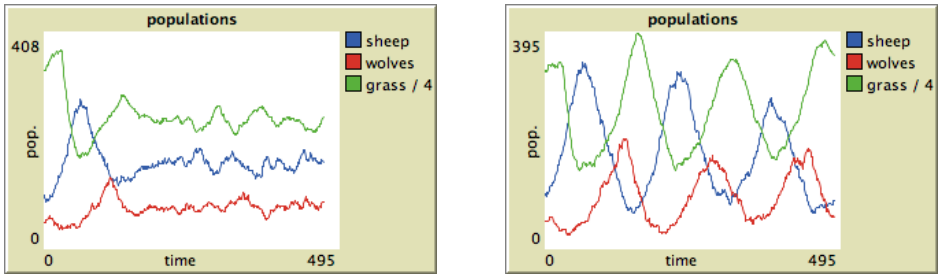
In Figure 5.2 (a plot from the NetLogo Traffic Basic ABM), the maximum speed and minimum speed of the vehicles converge to .65 and 0, respectively. However, before time step 650, the behavior was quite different than the eventual convergent behavior. If each time step (0 to 1250) were averaged, the value would be biased to be larger than .65. This number has meaning, but not the "average top speed" that was expected. The average should be comprised of values after time step 650 to match the actual measurement with the expectations.

This class of measurement should not be used on properties that diverge (i.e., fail to converge on a single value over time). In divergent cases, the average will continue to increase or decrease as more time steps are used in the measurement. However, it should be the case that the average value converges as more time steps are measured. The value produced by the forward and reverse mappings will be inaccurate and meaningless if this property does not hold. This is because the value would be dependent on how many time

steps the system was sampled, and would have very little to do with the value of the behavior itself.

5.2.2 Variance of a Value

The variance of a value over the life of a system is a useful metric for determining how "stable" the behavior is. If the behavior changes wildly from time step to time step, the variance will be relatively high to one in which the value remains stable. See Figure 5.3 for a comparison between two situations in which the variance of values are different.



(a) (b)

FIG. 5.3. Two different behaviors in the NetLogo Wolf Sheep Predation ABM with similar average sheep, wolves and grass values. The two can be distinguished by the variance: the values in (a) have a lower variance than the ones in (b).

Similar to averaging system-level properties over time, only values after convergence should be used to calculate the variance. This must be done to avoid biasing the variance of the behavior after it has converged. Taking this problem into consideration is particularly important in this for the variance metric because systems' behaviors typically vary significantly before convergence.

Also similar to the average value metric, this variance metric should not be used on behaviors that diverge. As a divergent property value continues to increase or decrease, the variance will increase. The variance should converge on a single value as more time steps

are sampled, for the same reasons that the average metric should converge.

5.2.3 Probability of a Threshold Effect

Threshold effects are binary properties that may or may not happen in a system. These tipping points divide the behavior space into configurations that exhibit the threshold effect and ones that do not. Configurations near the threshold will often times vary between which behavior side of the threshold it will converge on, run after run. For example, in the Wolf Sheep Predation ABM, wolves may or may not go extinct while the system is approaching convergence. With some configurations, the wolves will always go extinct, but in others the wolves never go extinct. Configurations on the boundary between wolves going extinct and not will exhibit both of these in different runs due to the inherent randomness in the system.

One particularly useful metric for analyzing threshold effects is the probability that a certain configuration will exhibit the threshold effect. For example, in the Wolf Sheep Predation ABM, a percentage could represent the probability that the configuration will result in the wolves going extinct. To measure this property, several experiments need to be executed for a single configuration. The probability is estimated by calculating the proportion of iterations that had no wolves, to the total number of experiments. The value of this probability should converge as more experiments for a single configuration are executed.

5.2.4 Measuring a Value That Changes Over Time

The previous measurements assume that the behavior will converge over time. Also, there are situation in which the average value or the variance of a value does not convey enough information. There are properties that change over time that do not fit into the above classes.

So far, all properties have been scalar values. The framework works only with scalar

values and is unable to naturally predict how a behavior will change over time. To circumvent this issue, the system-level properties could be parameters of a parametric model that represents a behavior that changes over time. This is essentially a nonlinear regression problem, in which the independent variable is time and the dependent variable is the property.

The parameters of the behavior's parametric model are learned in the forward mapping process as individual prediction problems. N different forward mappings will be learned for each N parameter. The value for each parameter is predicted with each mapping, given the configuration. These parameters are plugged into the parametric model to generate a curve that represents the behavior over time.

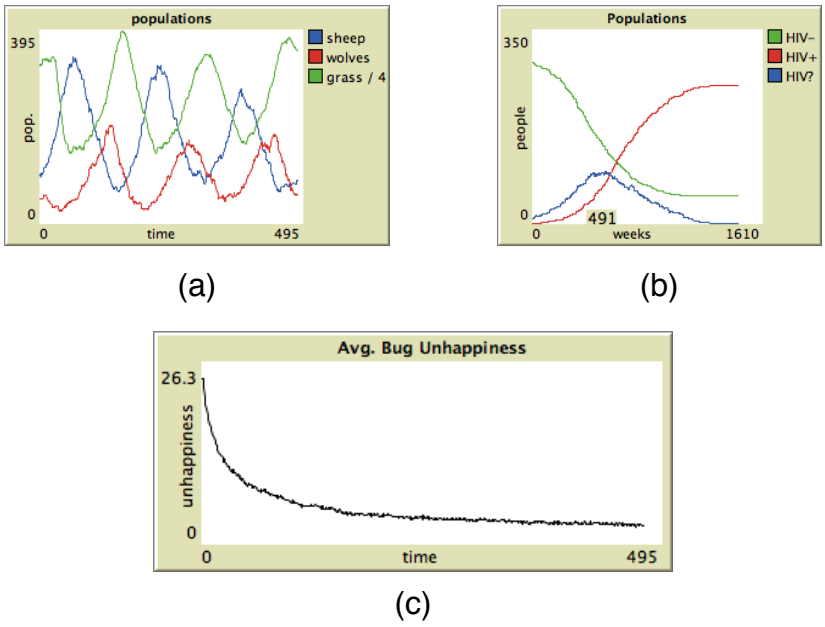


FIG. 5.4. Different types of behaviors that change over time in NetLogo ABMs.

This approach can be applied to a number of phenomena, such as the ones illustrated in Figure 5.4. In *(a)*, the values of animal populations in the Wolf Sheep Predation model vary rhythmically and could modeled with sine curves. The parameters of these behav-

iors modify the magnitude, height and frequency of the sine curves. In *(b)* (a plot from the NetLogo AIDS model (Wilensky 1997a)), the *HIV-* (people without HIV) curve decreases and the *HIV+* increases with following a sigmoid. The parameters of these two sigmoid behaviors modify the shape of the sigmoid. The *HIV?* (people that have HIV but do not know it) value appears to follow a curve that is similar to a probability distribution. However, it is simpler to model this property in terms of the other two: the HIV- and HIV+ values subtracted from the total population. In *(c)* (a plot from the NetLogo Firebugs model (Wilensky 1997b)), the behavior converges to zero over time. The rate of convergence (i.e., the slope of the curve) would be the parameter for the model of this behavior. This general approach is applicable to any property that can be described by a parametric regression model.

These system parameters are measured during sampling by performing parametric nonlinear regression for each experiment to determine good values of the parameters. These parameters are the system-level properties for AMF, but can be interpreted by the user as the parameters to the regression model.

Custom models can be developed for particular domains. Consider the NetLogo Traffic Simple simulation, in which cars move in a single lane. Traffic jams appear in waves, which is illustrated by the movement of the red car in Figure 5.2. We would like to develop a mapping to represent the velocity of the red car. From visual observation, it appears that the red car has a constant minimum speed and a maximum speed. It is known that the acceleration is linear from the programming of the model. Also, the red car follows a rhythmic behavior: the car reaches a certain speed, then stops. With these, all that is needed to create a wave-like parametric model that describes this behavior is available. The following variables are used in the parametric wave model:

- Let A be the amplitude of the wave (i.e., the maximum velocity minus the minimum velocity)

- Let T be the period of the wave (i.e., the time between the occurrence of a maximum velocity and the occurrence of a minimum velocity)
- Let h be the height of the wave (i.e., the minimum speed)

The behavior within each period is linear, so it can be described as a simple linear model:

$$speed(t) = \frac{A}{T}\, t + h$$

Since this behavior is periodic, modulus T is used to reset the behavior back to the minimum spread.

$$speed(t) = \frac{A}{T}\, (t \bmod T) + h$$

The forward-mapping problem would involve predicting values for $A, T,$ and h, given the configuration of the system. For each experiment, nonlinear regression is used to find good values for $A, T,$ and h, which are recorded as the dependent variables.

5.3 Sampling

The sampling process may begin once system-level properties are defined. The AMF sampling process consists of two major steps: first, retrieve raw data (i.e., just simple property values at each time step) from the simulation, then, use that data to compute the values of the system-level properties (e.g., average and variance). Instead of performing the calculations while sampling, I prefer a method of computing the actual statistics from a raw data set. This way, new statistics could be measured using the raw data at a later point in time without having to re-sample the system.

The AMF uses an evenly distributed random sampling technique. Ranges for parameters are passed to the sampling program and random values within these ranges are generated for each experimental configuration. This technique is flexible because it allows

me to change the size of the data set dynamically for testing purposes. Also, new points can be sampled to increase the accuracy of the data set in a particular area. The initial regression algorithms I have selected to work with AMF work with randomly distributed data. More advanced sampling techniques could be used in future work. See Subsection 9.1.1 for a more in depth discussion of this possible extension to AMF.

Sampling the same point, instead of generating a new point per experiment, has several uses. Different samplings of the same point could be used to:

- Measure the variance of a variable between samples to determine its stability,
- Determine the probability of a threshold effect being exhibited,
- Smooth the data set by reducing natural error,
- Generate more accurate individual points, and more.

For these reasons, I suggest sampling each randomly selected point several times, instead of sampling a wider variety of points once.

5.4 Implementation Details

The AMF implementation for defining system-level properties and performing sampling is designed to require a minimal amount of user programming. The sampling process consists of a number of distinct steps:

1. Identify, by name, which NetLogo properties need to be extracted from the simulation in order to calculate the system-level properties; Also, the user specifies how often (i.e., time steps passed) each property should be measured.
2. Specify all ABM configuration parameters by name, along with minimum and maximum values that random configurations will lie within;

3. Run the sampling application to generate the raw data set;

4. Pass the raw data set through the map program, which converts the raw entries into a more compact data set that contains the desired system-level behaviors and relevant configuration parameters.

The sampling program is written in Java so that it can interface with NetLogo's Java API, which is used to run the ABMs. All the user provided information is passed to this program in a text configuration file. The details of this configuration file and how to write one is covered in Appendix *[TODO]*. A program which emits values with an identical format as the default sampling program could replace the default sampling program. Replacing the default sampling program would be useful for taking more control of the sampling process, for instance to implement an advanced sampling strategy.

The sampling program prints the raw data to standard out. The data is tab delimited to separate items within entries and entries are delimited by newlines. Each entry is organized in the order specified in the configuration file, so the configuration file can be used to give the different columns identifiable names. The standard out output could be redirected to store data in a file or piped to the next step.

In the final step of sampling, the "map" Python program is used to parse the raw data and output a data set that contains the system-level properties. The map script parses the data for each entry. The parse returns a hash table object (Python dictionary) per entry, in which the keys are the names of the raw data items and the keys are the values, as a string. The map script is passed a user-created Python module that is used to compute the values of the system-level properties from the raw data. This user-created Python module contains a list of functions that each returns a value for a different system-level property. Map returns new data sets to individual files (one per system-level property), containing the configuration parameters and the system-level property. More details on how to create this

Python module is covered in Appendix *[TODO]*. The name “map” was chosen because it is a similar process to functional programming’s *map* function.

The output of the map function is then passed to the forward-mapping problem solver, which is covered in the next chapter.

Chapter 6

THE FORWARD-MAPPING PROBLEM

The *forward-mapping problem* is the problem of developing a mapping from an ABM's configuration space to the user-defined system-level property space. The forward-mapping problem is the first half of building a complete meta-model of an agent-based model (the other half being the reverse-mapping problem). Learning a meta-model is made simpler by splitting the problem into two sub-problems, in comparison to directly building a bidirectional mapping from data.

This chapter details aspects of the forward-mapping problem and my proposed solution that is used in AMF. In addition, I suggest a comprehensive list of evaluation criteria for all solutions to the forward-mapping problem.

6.1 Definition of the Problem

The forward-mapping is a functional mapping that maps *configuration space* to *system-level property space*. Let $\mathbf{y}$ be a point in system-level property space and $\mathbf{x}$ be a configuration. The forward mapping $f$is defined as the mapping that satisfies the following:

$$f(\mathbf{x}) \rightarrow \mathbf{y}$$

This mapping is used to answer the query "given $\mathbf{x}$, approximate the the system-level properties $\hat{\mathbf{y}}$.

$$f(\mathbf{x}) \approx \hat{\mathbf{y}}$$

The space that is the combination of the configuration and the system-level property spaces is the *behavior space*.

Configuration space is comprised of dimensions representing all configuration parameters of the agent-based model that could affect system-level behavior. Configuration parameter values can be discrete values (i.e., integers) or real values. This space is n-dimensional, where n is the number of configuration parameters. For NetLogo ABMs, user interface elements such as sliders, text boxes and switches represent the configuration dimensions in the configuration space.

System-level property space is comprised of dimensions representing all user-defined system-level properties that are being measured. Typically, these values are always real valued. Since many system-level property metrics are statistical in nature, real values are used even when a discrete value is expected. For example, a expected wolf population of 70.6 still has statistical meaning, even if having .6 of a wolf is impossible. Dimensions could be used to represent monitors or features of graphical plots, but, in general, system-level properties are user-defined statistical metrics.

An outline of the configuration and system-level property space for the Wolf Sheep Predation ABM are shown in Table 6.1. In the Wolf Sheep Predation ABM, five configuration parameters and five system-level properties are considered. Both of these spaces are five-dimensional and thus the behavior space is ten-dimensional.

A solution to the forward-mapping problem must be able to handle configuration spaces with many dimensions. Also, the methods must be able to perform accurately in both continuous and discontinuous behavior spaces. Several evaluation criteria that are

Table 6.1. Outline of Wolf Sheep Predation Behavior Space

Configuration Space	
grass-regrowth-time	Number of time steps it takes for grass regrow
sheep-gain-from-food	Energy gained by a sheep when it eats grass
wolf-gain-from-food	Energy gained by a wolf when it eats a sheep
sheep-reproduce	Probability that a sheep reproduces every time step
wolf-reproduce	Probability that a wolf reproduces every time step
System-Level Property Space	
wolf-extinction	Probability the wolves will go extinct
wolf-population	Average wolf population, should it stabilize
sheep-population	Average number of sheep
wolf-variance	Variance of the wolf population
sheep-variance	Variance of the sheep population

used to analyze the effectiveness and efficiency of forward-mapping problem solutions are provided later in this chapter in Section 6.4.

6.2 The AMF Approach

Regression is the default approach taken by AMF to solve the forward-mapping problem because it fits this problem naturally: the configuration parameters are the independent variables and the system-level properties are the dependent variables. The sampling phase of AMF provides a data set with measurements from numerous and diverse configurations. Each entry in the data set is a *(configuration, system-level measurements)* pair. These individual observations are passed to the regression algorithm to base its predictions off of.

Different regression approaches use this data in different ways. Approaches like k-nearest neighbor (kNN) do no pre-processing and uses the entire data set for each query. Meanwhile, parametric approaches like nonlinear regression (NLR) train a compact model and uses a limited number of parameters to represent the entire data set. In general, more time spent pre-processing equates to less time spent querying. Amortized over a large

number of queries, offline computation to speed up online queries is a worthwhile trade-off.

6.2.1 Scaling

Calculating the distance between two observations is an important part of many non-parametric regression techniques. Distance is used in kNN to determine which points will be factored in to determine the output value. Distance is also used in LOESS to weight closer points higher (hence, the "locally-weighted" part of the name).

Different configuration parameters have different ranges and different meanings. Therefore, performing a simple Euclidean distance on these parameters to determine distances between instances will typically not yield favorable results. For example, consider *sheep-reproduce*, a percentage value in the Wolf Sheep Predation ABM, which is sampled from .01 to .08. Another parameter, *grass-regrowth*, is sampled from 5 to 30. The Euclidean distance metric treats all dimensions as equal, so closeness in the larger-ranged dimensions will be given lower weight than closeness in the smaller ranged dimension. For example, values of .01 and .03 could be proportionally similar to 6 and 7, yet the Euclidean distances are very different: .02 and 1. This skew in weight can make dimensions less important, regardless of their actual importance. Properly scaling the data set rectifies this problem.

Simply scaling each dimension linearly such that the minimum and maximum values of each dimension are zero and one, respectively, is typically sufficient. The analysis of the effect of more advanced sampling techniques is left as future work and is discussed more in Section 9.1.2.

Scaling is an optional step in AMF that is inserted between the sampling phase and the forward-mapping phase. Parameters for the minimum and maximum values are passed to the scale program, which scales data points linearly, accordingly. When AMF is queried,

points are rescaled their original values.

6.2.2 Handling Non-Continuous Configuration Spaces

Discrete value configuration parameters are handled as a special case in AMF. In designing AMF, I did not feel the need to handle discretely valued parameters as a typical use case since they are rather rare. Therefore, to streamline and simplify the processing of real-valued parameters, discrete valued parameters are handled differently in sampling and learning the forward mapping.

The sampling program is told whether or not each parameter is a discrete integer or a real value. The random selection of data points will abide by this restraint by randomly selecting integers for discrete valued parameters, instead of real values

The best way to handle a discretely valued parameter in learning the forward mapping depends on the nature of the variable. In the case where the parameter is sampled such that thousands of possible values are possible or the number has little effect on the system, it may suffice to treat the parameter as a real value. In many cases, the error incurred by making this assumption is negligible. This has the advantage of not having to treat the discrete values as special. Another approach would be to split the configuration space into separate configuration subspaces, with each subspace having a uniform value for the discrete value. This provides more accurate results at the cost of higher computation requirements. When there are numerous different discrete parameters, the number of subspaces grows quickly, making this approach intractable.

Another problem caused by discrete values is the reverse mapping returns a real values for each of the parameters. The approaches used in AMF for the reverse mapping are not compatible with discrete values. The mappings will return impossible configurations as possible solutions to the reverse mapping problem. AMF leaves circumventing this problem to the configuration selection phase to select discretely valued configurations. This

way, the reverse-mapping problem can be solved the same way for both real and discretely valued parameters.

6.2.3 Handling Multi-Variate Forward Mappings

In the problem definition, I mention that a number of system-level properties can be measured at once. AMF splits each system-level property into individual single-property forward-mapping problems. The reformulation the original definition of the forward mapping is as follows:

$$f_i(\mathbf{x}) \rightarrow y_i$$

where $i = \{0, 1, \ldots, |\mathbf{y}|\}$. To answer the query for $\hat{\mathbf{y}}$, given $\mathbf{x}$, the sub-mappings are individually queried and collected:

$$\{f_0(\mathbf{x}), f_1(\mathbf{x}), \ldots, f_{|\mathbf{y}|}(\mathbf{x})\} \approx \hat{\mathbf{y}}$$

This approach assumes that the system-level properties are not correlated (i.e., independent). By assuming independence, the predictions may not be as accurate as they could be. This problem is outside the scope of the implementation of AMF, as it has not been a problem for any of the test domains that have been used with AMF.

6.2.4 Implementation Details

The AMF regression solution to the forward mapping is split into two distinct steps: training and predicting. The training component builds the model and/or pre-processes the data as necessary. The predicting component allows the user and other framework components (e.g., the reverse mapping) to query the regression model. This process is driven by a user-provided regression library that is plugged into the framework. More on

how to build this library in order to properly interface with AMF is provided in Appendix ??.

Training The training portion of solving the forward mapping problem is where any necessary pre-processing and/or model construction is performed. This process is implemented as a Python program called *train*. *train* takes the training data set as input and uses this data to prepare and set up the specific regression approach passed in by the user. Most regression algorithms will write models or meta-data to the local file system for later use. Basic kNN does nothing since the queries use the entire data set. LOESS performs the iterative locally weighted smoothing step to build y' values, which are written to a file. Nonlinear regression learns optimal parameters for the parametric model given by using optimization. The parameters to the model are written to a file. Multi-linear interpolation approaches use this phase to develop an evenly spaced data set with another regression approach. Each of these approaches require varying amounts of computational time. This step only needs to be executed only once for a given data set.

Predicting The prediction portion of solving the forward mapping problem is the action performed when a query is submitted. This process is implemented as a Python program called *predict*, but is mostly driven by the user-provided regression module. *predict* uses the models generated by *train* to predict the value for individual system-level properties.

Like the training phase, the predicting phase changes from algorithm to algorithm. kNN will iterate through every point to find the nearest neighbors, then average the values. LOESS will interpolate between nearby smoothed data set points. NLR simply plugs in the configuration values into the parametric model. MLI finds which bin the configuration lies within and then interpolates the values from the corners. Each of these approaches require

varying amounts of computational time, like the training step. However, this process is executed once per individual query, making efficiency and important factor.

6.3 Using Forward Mappings

The forward mapping has two main uses: it is used to predict the behavior of a system, given the configuration, and it is used by AMF to solve the reverse mapping problem.

The forward mapping can be used to predict the behavior of a system without having to actually run it. This is useful because running an ABM may require too much time for hypothetical experiments. Also, in ABMs that have noisy behavior, several iterations of sampling may have to be performed to get an accurate average value. This makes sampling the original ABM even more computationally intensive. Interacting with the forward mapping is more convenient than interacting with the ABM to verify hypotheses about system-level behavior. Also, exploration of the behavior space is more efficient when interacting with a model instead of running an ABM numerous times.

AMF uses the forward mapping to solve the reverse mapping. A number of approaches are discussed in the next chapter, Chapter 7: The Reverse-Mapping problem. Most of these approaches use the forward mapping instead of directly sampling from the ABM for two reasons. First, querying meta-models is faster than interacting with the ABM. Second, the regression models smooth the error over the entire space, effectively eliminating natural sampling noise. This can produce more accurate and consistent results.

A secondary use for the forward mapping is for visualizing the behavior space. Plotting programs could be passed a number of inferred points to show the general shape of the mapping. The forward mapping can be used to generate a number of points, which are then plotted to show the space. Also, parametric regression approaches such as NLR could be graphed directly and compared to the training data set.

6.4 Solution Evaluation Criteria

A number of algorithms and approaches can be used to solve the forward-mapping problem. Some approaches work better in different domains. Therefore, evaluation of the forward mapping is important in determining which approach would work best.

There are a number of evaluation criteria proposed in this section. Not one algorithm I have tested with AMF has dominated the others in performance. Some are more accurate but require more query time. Some are fast for querying but slow for training. There are a number of these trade-offs that can affect the choice of which algorithm is "best" for a situation.

A number of algorithms plugged into the general AMF forward-mapping problem solver are evaluated with the criteria outlined in this chapter. The results are given in Chapter 8: Results.

6.4.1 Time Required for Training

The time required for training is listed as not very important in the AMF design goals. However, intractable training problems should be avoided.

There are two matters of importance when measuring the time required for training a model. First, how long does it take? Second, how does the length of time training takes correlate to the size of the data set?

This is measured by training a number of different mappings with different sized data sets. The training time for each of these is measured and compared.

6.4.2 Time Required for Querying

The time required for querying is noted as important in the design goals because a quick average query time is important for user interaction. Also, querying is important for

solving the reverse-mapping problem, as numerous queries are performed. If the query time is slow, the performance of the reverse mapping solver will significantly suffer. The time required for querying the forward mapping should always be less than the amount of time it takes to sample the ABM directly. If querying the forward mapping takes longer, there is no point in this process as the user could just sample the ABM for faster and perhaps more accurate results.

Evaluation is performed by by submitting a large number of random queries to the regression algorithm. The average response time and standard deviation of response times are measured. The amount of time it takes to query the ABM directly is used as a baseline to show the improvement provided by AMF.

With some regression algorithms, the size of the data set may affect the average query time. The relationship between data set size and query time is also evaluated to determine at which point direct domain sampling would be preferred. For example, kNN will query slower when the data set is larger because it needs to iterate through more points per query. At a certain point, traversing the entire data set will take more time than simply sampling the ABM directly. Identifying this point for relevant algorithms is important for determining an appropriate data set size or determining whether or not an algorithm is appropriate for a particular ABM.

6.4.3 Accuracy of the Forward Mapping

The accuracy of a forward mapping is perhaps the most important of all evaluation criteria. If the mapping is not very accurate, sampling the ABM directly would be more useful, even if it takes more time.

To measure the accuracy of predictions, I use cross validation. To do this, I randomly split the original training set into two sets, a smaller training set and a validation set. The training set is used by the regression algorithm. Then, for every entry in the validation

set, the true value originally sampled is compared to the predicted value, to measure error. This process is repeated a number of times by selecting different subsets of the data set for the training set and the validation set. All errors collected can then be used for a number of statistics that describe the accuracy of the approach. The average error and standard deviation of errors provide information on the expected range of performance. These errors are compared to the natural standard deviation observed from repeated samples of the true ABM.

Visualizing the distribution of errors over the entire behavior space is a useful tool in determining which portions of the space are more tumultuous. These areas can be considered more chaotic than other portions of the space.

The relationship between accuracy and data set size is also an interesting metric. This metric can provide a guideline for the size of a data set, given that a particular approach will be used. Also, different approaches can be compared by how efficiently they use the data: better algorithms will be more accurate with less data.

6.5 Summary

The forward-mapping problem is the problem of developing a mapping that predicts system-level behavior, given the system configuration parameter values. The solution to this problem has a central role in the framework, as it provides the ability for users to predict behavior and is used to solve the reverse-mapping problem.

My solution to the forward-mapping problem is to use regression to learn the correlations between the configuration parameters (independent variables) and the system-level properties (dependent variables), individually. Different regression algorithms, in the form of a Python module library, can be plugged into AMF seamlessly. This allows for a great amount of flexibility in approaches that can be used to sample ABMs. AMF scales with new

technology as new regression techniques are developed since they can simply be plugged in by the user.

I have proposed a number of different metrics for evaluating the current forward-mapping approaches, as well as future approaches that users may use. The most important goals for any approach to solving the forward-mapping problem is: querying for a prediction must be faster than sampling the system directly and predictions should be as accurate as possible.

Chapter 7

THE REVERSE MAPPING PROBLEM

The reverse-mapping problem is the problem of defining a function f^{-1} that maps a system-level configuration $\mathbf{y}$ onto a solution space $\hat{\mathbf{S}}$ that represents all possible configurations that would have the system exhibit $\mathbf{y}$:

$$f^{-1}(\mathbf{y}) \rightarrow \hat{\mathbf{S}}.$$

Every configuration $\hat{\mathbf{x}} \in \hat{\mathbf{S}}$ should satisfy the constraint $f(\hat{\mathbf{x}}) \approx \mathbf{y}$, where f is the forward mapping. That is, $\hat{\mathbf{S}}$ contains all the points that would predict $\mathbf{y}$ in the forward mapping.

In this chapter, I give details of how AMF approaches the solution to this problem, in general, then delves deeper into actual implementations of the solution. Then, I discuss evaluation criteria for implementations of reverse-mapping problem solutions.

7.1 The AMF Approach

The general approach taken by AMF for solving the reverse-mapping problem is similar to the AMF forward-mapping solution approach. First, the different system-level properties are split into sub-problems. Next, solution spaces for each sub-problem are constructed with a forward-mapping inversion technique. Finally, each solution subspace is

recombined to produce the space of configurations that will produce the provided system-level property.

Much like the forward-mapping solution, $\mathbf{y}$ is split into subspaces to simplify the problem:

$$\mathbf{y} = \{y_0, y_1, \ldots, y_{|\mathbf{y}|}\}.$$

The solution space is split up in a similar manner:

$$\hat{\mathbf{S}} = \{\hat{S}_0, \hat{S}_1, \ldots, \hat{S}_{|\mathbf{S}|}\}.$$

By performing this split, each forward mapping can be inverted independently:

$$f_i^{-1}(y_i) \rightarrow \hat{S}_i.$$

Recombining the individual $\hat{S}_i$ into $\hat{\mathbf{S}}$ is not as simple as recombining individual $\hat{y}_i$ into $\hat{\mathbf{y}}$ in the forward-mapping solution. Since each solution space represents which configurations satisfy a particular system-level requirement, the solution space $\hat{\mathbf{S}}$ is the intersection of all these spaces:

$$\hat{\mathbf{S}} = \hat{S}_0 \cap \hat{S}_1 \cap \ldots \cap \hat{S}_{|\mathbf{S}|}.$$

All points at the intersection of these spaces should satisfy all system-level properties at once.

The nature of the solution spaces vary from approach to approach. In the approaches used by AMF, they are represented as a collection of discrete subspaces, discrete exemplar points, sets of functions, or linear combinations.

7.2 Approaches

I have devised and evaluated a number of different approaches for solving the reverse-mapping problem in this dissertation research. This section enumerates these approaches. Although each the approaches are quite different, they all *invert* a solution to the forward-mapping problem. That is, they use the forward mapping to develop the reverse mapping, instead of learning the reverse mapping directly. This has the benefit that the reverse-mapping solutions are agnostic to the regression method used to learn the forward mapping.

Each approach is follows a different technical process, but all query the forward mapping for points instead of using the original data set. Most approaches conform to the assumption stated in Chapter 6: The Forward-Mapping Problem that all the system-level properties can be analyzed independently. Every approach has a different method for intersecting the different solution spaces to find a solution space that satisfies all system-level properties at once.

Each approach has different computational and space complexity. The expectations for each method and the situations in which they are optimal vary from domain to domain and from query to query.

The first two approaches discussed in this section, *Thresholding* and *Regression / Interpolation / Intersection*, conform to the theoretical framework outlined above. Two other approaches are labeled as alternatives, as they do not follow the framework exactly, but perform a similar task. In the final subsection, I discuss the special case in which the system-level property is a binary value (i.e., the forward mapping is classification). Throughout this section, I use the NetLogo Fires ABM[1] as an example. The Fires domain is simple, but has solution spaces that are easy to visualize. Other domains, such as Wolf Sheep Predation, have to many dimensions to make graphing possible. The arguments made with the

[1] More on the Fires model is discussed in Section 8.1.1

aid of the Fires domain scale to larger dimensions.

7.2.1 Thresholding

7.2.2 Regression/Interpolation/Intersection

Regression

Interpolation

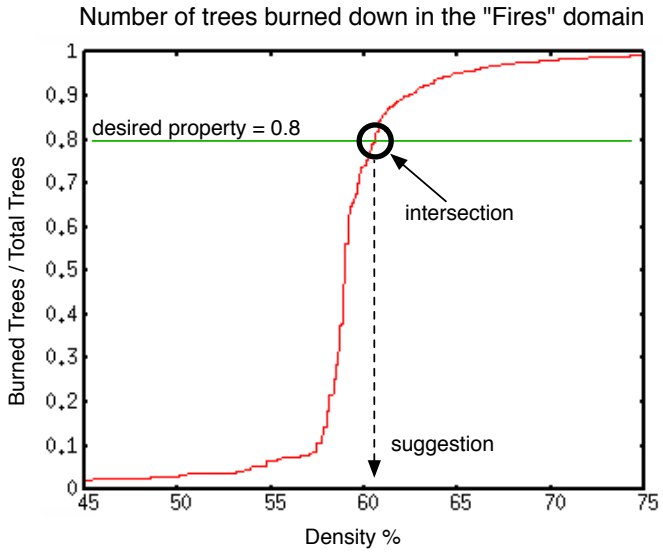


FIG. 7.1. An illustration of using surface-to-surface intersection to solve the reverse-mapping problem in the NetLogo Fires domain.

Intersection

Chapter 8

RESULTS

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

8.1 Domains

8.1.1 NetLogo Fires Domain

Chapter 9

CONCLUSIONS AND FUTURE WORK

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

9.1 Future Work

9.1.1 Advanced Sampling Techniques

9.1.2 Advanced Scaling Techniques

Appendix A

CODE SAMPLES

A bunch of code samples will be put here. These will serve as examples to show how to do things or to show how easy they are

A.1 Sample Java/NetLogo Sampling Program

This code example will show how to write a sampler for AMF.

Appendix B

INTERFACE SPECIFICATION FOR REGRESSION ALGORITHMS

I will explicitely detail the interface that a regression algorithm to be plugged into AMF needs to have.

REFERENCES

[1] Bonabeau, E. 2002. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences of the United States of America* 99(Suppl 3):7280.

[2] Boyd, J.; Hushlak, G.; and Jacob, C. 2004. SwarmArt: Interactive art from swarm intelligence. In *Proceedings of the 12th Annual ACM International Conference on Multimedia*, 628–635.

[3] Buhl, J.; Sumpter, D.; Couzin, I.; Hale, J.; Despland, E.; Miller, E.; and Simpson, S. 2006. From Disorder to Order in Marching Locusts. *Science* 312(5778):1402–1406.

[4] Cleveland, W., and Devlin, S. 1988. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American Statistical Association* 83(403):596–610.

[5] Cleveland, W. 1979. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association* 829–836.

[6] Collier, N. 2003. Repast: An extensible framework for agent simulation. *The University of Chicago's Social Science Research*.

[7] Couzin, I., and Franks, N. 2003. Self-organized lane formation and optimized traffic flow in army ants. In *Proceedings of the Royal Society of London, Series B*, volume 270, 139–146.

[8] Crowther, B., and Riviere, X. 2003. Flocking of autonomous unmanned air vehicles. *Aeronautical Journal* 107(1068):99–109.

[9] Cui, X.; Gao, J.; and Potok, T. 2006. A flocking based algorithm for document clustering analysis. *Journal of systems architecture* 52(8-9):505–515.

[10] Davies, S. 1997. Multidimensional triangulation and interpolation for reinforcement learning. *Advances in neural information processing systems* 1005–1011.

[11] Dorigo, M. 2004. *Ant Colony Optimization*. MIT Press.

[12] Epstein, J. 1999. Agent-based computational models and generative social science. *Complexity* 4(5):41–60.

[13] Eubank, S.; Guclu, H.; Anil Kumar, V.; Marathe, M.; Srinivasan, A.; Toroczkai, Z.; and Wang, N. 2004. Modelling disease outbreaks in realistic urban social networks. *Nature* 429(6988):180–184.

[14] Fox, J. 2002. *An R and S-Plus companion to applied regression (Appendix)*. Sage Publications.

[15] Gallant, A. 1975. Nonlinear regression. *The American Statistician* 29(2):73–81.

[16] Gionis, A.; Indyk, P.; and Motwani, R. 1999. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*.

[17] Huber, E., and Barth., W. 1999. Surface-to-surface intersection with complete and guaranteed results. *Developments in Reliable Computing* 185–198.

[18] Kennedy, J., and Eberhart, R. 1995. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*.

[19] Klein, J. 2003. Breve: a 3D environment for the simulation of decentralized systems and artificial life. In *Proceedings of the Eighth International Conference on Artificial Life*, 329–334. MIT Press.

[20] Lerman, K., and Galstyan, A. 2002. Mathematical model of foraging in a group of robots: effect of interference. *Autonomous Robots* 13(2):127–141.

[21] Lerman, K.; Martinoli, A.; and Galstyan, A. 2005. A review of probabilistic macroscopic models for swarm robotic systems. In *Swarm Robotics Workshop: State-of-the-art Survey*, 143–152. Springer.

[22] Luke, S.; Cioffi-Revilla, C.; Panait, L.; Sullivan, K.; and Balan, G. 2005. Mason: A multiagent simulation environment. *Simulation* 81(7):517–527.

[23] Minar, N.; Burkhart, R.; Langton, C.; and Askenazi, M. 1996. The swarm simulation system: A toolkit for building multi-agent simulations.

[24] Minsky, M., and Papert, S. Perceptrons: expanded edition. *MIT Press, Cambridge MA* 19882:20.

[25] Moere, A. 2004. Time-varying data visualization using information flocking boids. In *Proceedings of the 2004 IEEE Symposium on Information Visualization*, 97–104.

[26] Mor, J. 1977. The Levenberg-Marquardt algorithm: implementation and theory. *Lecture notes in mathematics* 630:105–116.

[27] Muller, K.; Mika, S.; Ratsch, G.; Tsuda, K.; and Scholkopf, B. 2001. An introduction to kernel-based learning algorithms. *IEEE transactions on neural networks* 12(2):181–201.

[28] North, M.; Collier, N.; and Vos, J. 2006. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 16(1):25.

[29] Parrish, J.; Viscido, S.; and Grunbaum, D. 2002. Self-organized fish schools: an examination of emergent properties. *Biological Bulletin, Marine Biological Laboratory, Woods Hole* 202(3):296–305.

[30] Parunak, H.; Savit, R.; and Riolo, R. 1998. Agent-Based Modeling vs Equation-Based Modeling: A Case Study and Users' Guide. *Lecture Notes in Computer Science* 1534:10–25.

[31] Patrikalakis, N.; Maekawa, T.; Ko, K.; and Mukundan, H. 1993. Surface-to-surface intersections. *IEEE Computer Graphics and Applications* 13(1):89–95.

[32] Pratt, S.; Sumpter, D.; Mallon, E.; and Franks, N. 2005. An agent-based model of collective nest choice by the ant Temnothorax albipennis. *Animal Behaviour* 70(5):1023–1036.

[33] Reynolds, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings)*, 25–34.

[34] Reynolds, C. 1999. Steering behaviors for autonomous characters. In *Game Developers Conference*, volume 1999.

[35] Schelhorn, T.; O'Sullivan, D.; Haklay, M.; and Thurstain-Goodwin, M. 1999. STREETS: an agent-based pedestrian model. In *Proceedings of Computers in Urban Planning and Urban Management*.

[36] Shi, Y., and Eberhart, R. 1998. Parameter selection in particle swarm optimization. *Lecture notes in computer science* 591–600.

[37] Smola, A., and Schölkopf, B. 2004. A tutorial on support vector regression. *Statistics and Computing* 14(3):199–222.

[38] Spears, W.; Spears, D.; Hamann, J.; and Heil, R. 2004. Distributed, physics-based control of swarms of vehicles. *Autonomous Robots* 17(2):137–162.

[39] Tisue, S., and Wilensky, U. 2004. NetLogo: A simple environment for modeling complexity. In *International Conference on Complex Systems*, 16–21.

[40] Van den Bergh, F., and Engelbrecht, A. 2006. A study of particle swarm optimization particle trajectories. *Information Sciences* 176(8):937–971.

[41] Wilensky, U. 1997a. NetLogo AIDS model. Technical report, Northwestern University.

[42] Wilensky, U. 1997b. NetLogo Firebugs model. Technical report, Northwestern University.

[43] Wilensky, U. 1997c. NetLogo Flocking model. Technical report, Northwestern University.

[44] Wilensky, U. 1997d. NetLogo Traffic Basic model. Technical report, Northwestern University.

[45] Wilensky, U. 1997e. NetLogo Wolf Sheep Predation model. Technical report, Northwestern University.

[46] Zanbaka, C.; HandUber, J.; and Saunders-Newton, D. 2009. Modeling and Simulating Community Sentiments and Interactions at the Pacific Missle Range Facility.