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. I present a learning framework, the ABM Meta-modeling Framework (AMF), that can be used to predict and control system-level behaviors of agent-based models. Using 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 from the bottom up. At the core of these simulations are individual agents that interact locally with other agents and with the environment. All of the behaviors 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*.

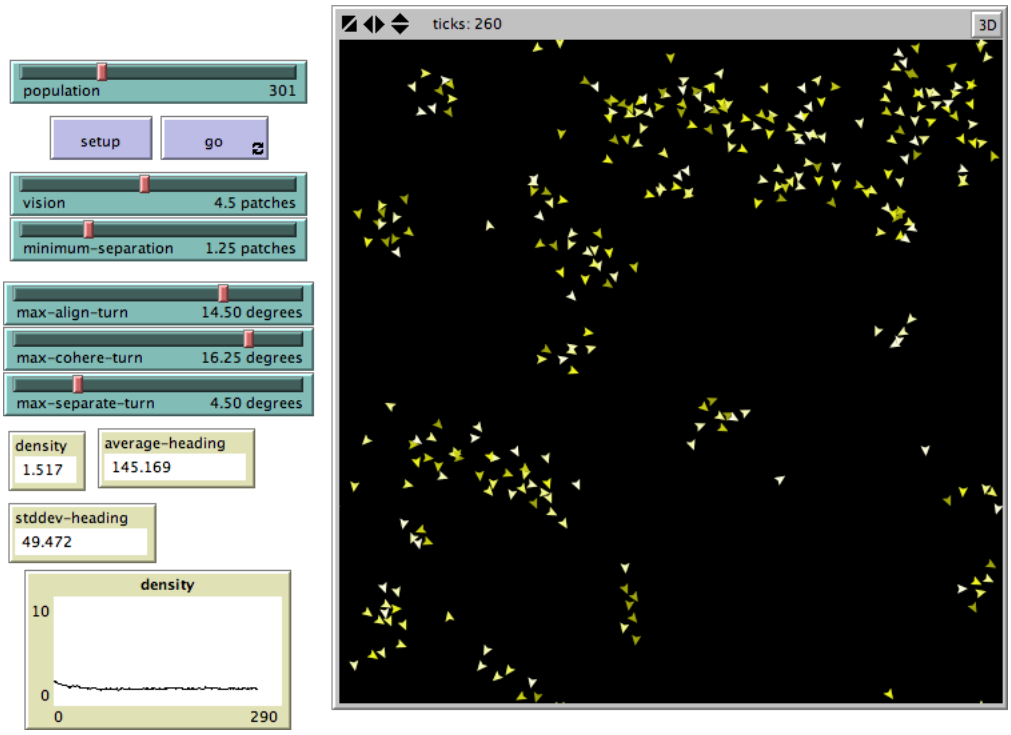


FIG. 1.1. A screenshot of NetLogo's graphical user interface while executing a flocking simulation.

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 studying agent-based models of lane formations in army ants were interested in the traffic patterns of the lanes, not in the individual behaviors of the ants (Couzin & Franks 2003). In other work, researchers studying locusts were interested in identifying the critical density at which locusts would begin to swarm and destroy crops (Buhl *et al.* 2006). Typically, scientists analyze ABMs by viewing visualizations of the environment or by gathering statistical data on the simulation. For instance, NetLogo, an agent-based modeling programming en-

vironment (Tisue & Wilensky 2004), provides 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 this property has changed over time. These tools can be used by researchers 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 major 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 these properties 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. Determining 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 using AMF, since researchers can interact 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.

1.2 Wolves, Sheep and Grass

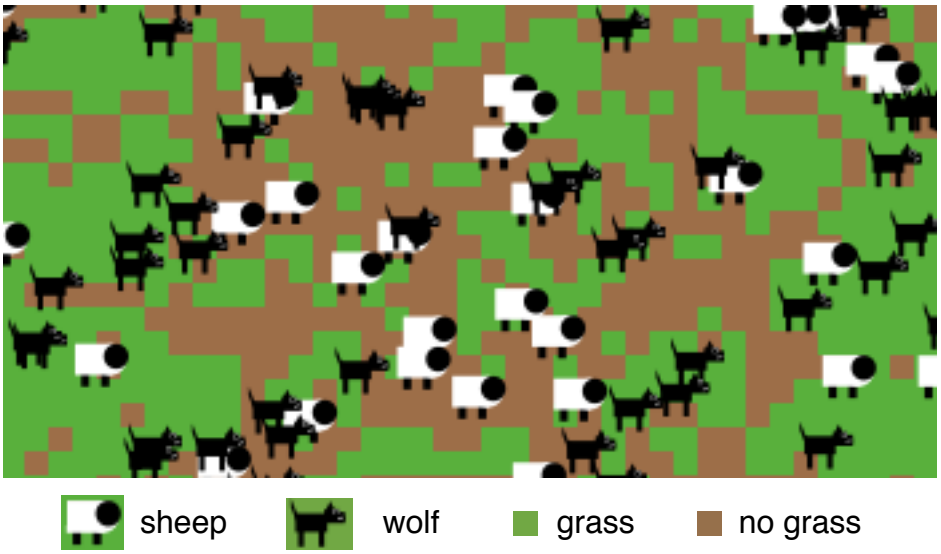


FIG. 1.2. A screen shot from NetLogo's Wolf Sheep Predation model.

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.

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

- Wolves and sheep die if they run out of energy.
- 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., the parent's energy divided by two).
- Grass regrows after *grass-regrowth-time* number of time steps.

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. That is, as more values for the sheep populations are sampled, the average approaches a single value, instead of oscillating like the instantaneous population. Another interesting feature is that some ecosystems fail: sheep and/or wolves go extinct.

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, as seen in 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.

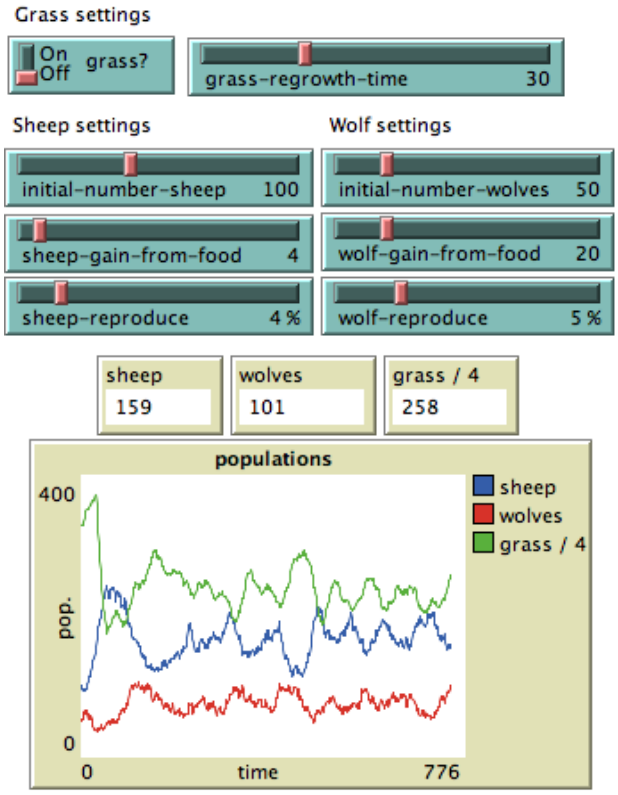


FIG. 1.3. The control and monitor interface for the Wolf Sheep Predation model.

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 (i.e., instantaneous population amounts divided by the number of instantaneous populations sampled). In Chapter 8 (Results), I will show that the intuitive disconnect in this domain can easily be solved by AMF.

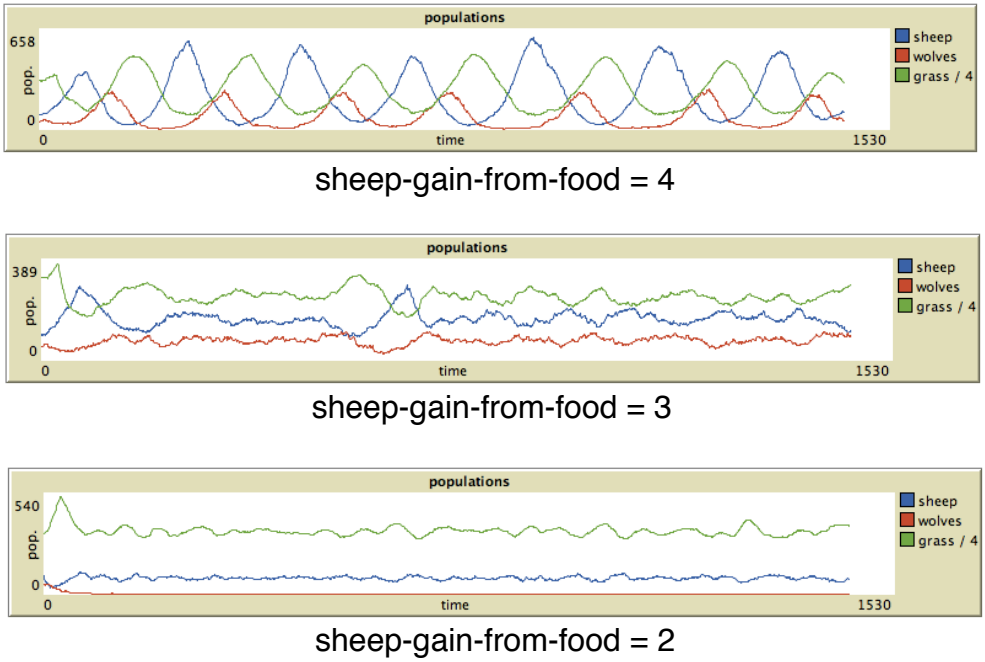


FIG. 1.4. Differences in populations based on changes of the *sheep-gain-from-food* parameter.

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 reverse-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 to define how to measure system-level properties of interest, to provide the ranges of parameters to be sampled, and to select a regression algorithm.

I will show in Chapter 8 that my framework is able to generate models of system-level behavior. For example, AMF is able to predict the average 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 desired system-level property outcome (the reverse-mapping problem).

A more comprehensive overview of AMF is provided in Chapter 2.

1.4 Summary of Contributions

The main contribution of 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 in which 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 serves as a proof-of-concept to show that my approach is feasible 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, describes 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 provides information about NetLogo and algorithms used by AMF. Chapter 4 discusses the numerous different ways that system-level properties of ABM can be defined. Chapter 5 enumerates the domains that AMF has been applied to, along with definitions of system-level properties. Chapters 6 and 7 discuss my solutions to the forward- and reverse-mapping problems. Chapter 8 evaluates AMF on a domain-by-domain basis and provides explicit examples of how AMF has been used. Chapter 9 compares and contrasts related approaches to AMF. Chapter 10 summarizes this dissertation, provides additional discussion and presents 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 for *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 the process of 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 that is needed for each new domain;
- Algorithm independent: any regression algorithm should be able to be applied with AMF;
- Accurate: AMF should generate accurate predictions and control suggestions; and
- Fast for the user – interactions with the models generated by AMF should require minimal computational time.

Domain independence is paramount because of the variety of ABMs. I have designed AMF in such a way that the same general approach would work for any ABM. Also, I strove to minimize the amount of configuration that is 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 for in-depth domain knowledge. To reinforce this claim, I have tested AMF on a number of diverse domains, using 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.[1] 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 used just as easily as current approaches.

[1] In general, the learning algorithms that will be discussed in this dissertation will satisfy the requirements for modeling most ABMs. 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.

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 more interaction with the user, increasing the user's effort. 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, because each step would have to run the ABM to calculate each 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). Figure 2.1 shows 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 external perspective, the framework only has a limited amout of input and output: AMF takes in observations of an agent-based model and provides interfaces to query for prediction and control. The user queries AMF to interact with the agent-based model in an intuitive way.

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, regardless of the configuration. This uniformity is what makes AMF a framework and not simply a collection of algorithms. For example, even though different regression algorithms can be used to learn the forward mapping, the forward mapping always provides predictions of how an ABM will behave, from the user's perspective.

2.2.1 Defining System-Level Behavior Properties

The 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 same 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 4.

For example, there are several system-level behavior properties that can be measured in the Wolf Sheep Predation model. Three of the most obvious properties 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.

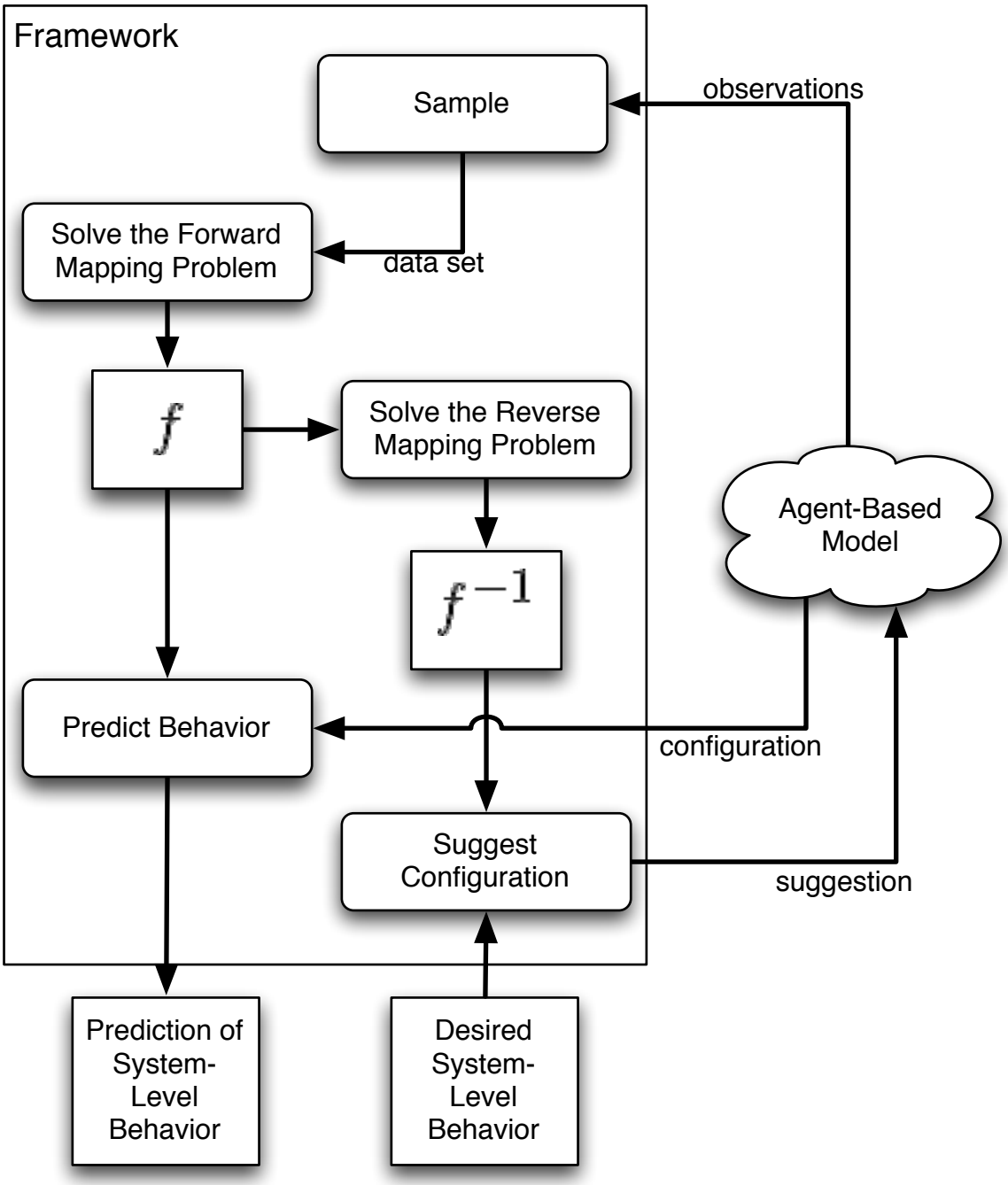


FIG. 2.1. An overview of the phases of AMF and how data flows among them.

2.2.2 Sampling

The first phase is *sampling*. In this phase, numerous observations are made of 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 resulting system-level behaviors, for each observation. This process can be very time-consuming, depending 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 fine-grained sampling 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[2] (Wilensky 1997d) requires about thirty seconds[3] to generate. A large sample of 50,000 observation of a Reynolds boid flock (Reynolds 1987) took approximately four days.

The user is required enough understanding of the domain to specify to AMF which ranges of values should be sampled. AMF is not able to automatically infer which value ranges are interesting, so these must be explicitly defined.

This sampling process is easily parallelizable. Since each experiment is independent of the others, 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 completed, the results can be merged into one data set.

[2] See Chapter 8 for detailed results.

[3] Most experiments are executed on a 3.0 Ghz Pentium 4 running Arch Linux.

For the purposes 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 intelligent sampling strategies could improve the performance of the framework, and such strategies could be the focus future work.

2.2.3 The Forward-Mapping Problem

The *forward-mapping problem* is to 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 $\hat{\mathbf{y}}$:

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

The set of values $\mathbf{x}$ consists of all the agent-level parameters (independent variables) in the data set provided by the sampling phase. 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 first "initialized," if necessary. For example, linear regression would need to solve the least-squares problem. Meanwhile, an algorithm like k-nearest neighbor would not necessarily need to initialize anything because it scans the data set per query. Then, the forward mapping can be used to predict values for configurations that have not been sampled.

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 represent *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.

2.2.4 The Reverse-Mapping Problem

The *reverse-mapping problem* is to produce a mapping $f^{-1} : \mathbb{R} \to \hat{S}$ 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 $\mathbf{y}$):

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

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 that f^{-1} does not, in general, describe a functional mapping: 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, using a novel method that I developed called *simplical complex inversion*. This approach has the benefit of using the forward mapping to develop the reverse mapping, so

no additional inputs from the user or additional data samples are needed.

The reverse mapping can be used to suggest a system configuration that will exhibit a specific set of system-level properties. I call this process *control* of the ABM, since it is controlling the ABM at the 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 single 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.

2.2.5 Summary of Configuration Points

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

- A list of agent-level control parameters, and the ranges within which they should be sampled.
- A list of system-level behavior properties and a process for measuring them.

In addition, the user may configure the following if they wish to modify the default behavior of AMF:

- The sampling strategy (default: random sampling).
- The regression algorithm to be used for the forward mapping (defaults: k-nearest neighbor, locally weighted linear regression, or nonlinear regression).

- 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, avoiding specific implementation details 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 in Chapter 8.

I have tested AMF with NetLogo[4] domains, however, technically any agent-based modeling system would suffice. 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 nuances, 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 a Java program by retrieving each of these values individually, then performing statistics on them. Performing the statistics calculations in Java has the benefit of not having to change

[4]More about NetLogo is discussed in more detail in Chapter 3.

the NetLogo ABM source code.

During sampling, 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 B.1.

The regression algorithms used with AMF must conform to a standard API[5] and are passed as arguments into the forward mapping, 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 this section, 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

[5]The specification for the standard regression algorithm interface is given in Appendix A.1.

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 requirement, since automatically detecting configuration points and having a computer determine the 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 it conforms to the standard API. A simple Python wrapper can be written to adapt an 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 depend on the running time of the regression algorithms, but typically 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 the ABM Meta-modeling Framework. In contrast to Chapter 9, the previous research presented in this chapter do not share the same motivation as AMF, but provide a foundation upon which AMF builds.

AMF uses preexisting research in two major areas: agent-based modeling and regression. Section 3.1 discusses what an ABM is, discusses some examples of ABMs, and lists some existing multi-agent software frameworks. Section 3.2 defines regression, outlines favorable and unfavorable properties, and describes a number of regression algorithms that have been incorporated into AMF.

3.1 Agent-Based Modeling

Agent-based models are computer simulations that are implemented from an agent perspective. Agents in ABMs are typically(Epstein 1999):

- bounded by a limited global view;
- only able to perform local interactions, affecting their local environment and neighboring agents;

- autonomous (i.e., there is no top-down control); and
- heterogeneous (i.e., agents within the system can have different properties) .

This paradigm stand sin contrast 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 provides 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 provide a theory for why that behavior emerges. Agent-based models are particularly well-suited to answer the Generativist's question.

Implementing systems as an ABM can 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 parameterized 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 is 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 the 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 dramatically. Agent-based models remedy this difficulty 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 from trying to realistically model a multi-agent system.

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. These examples will illustrate the uses for ABMs and how they can be used to answer research questions.

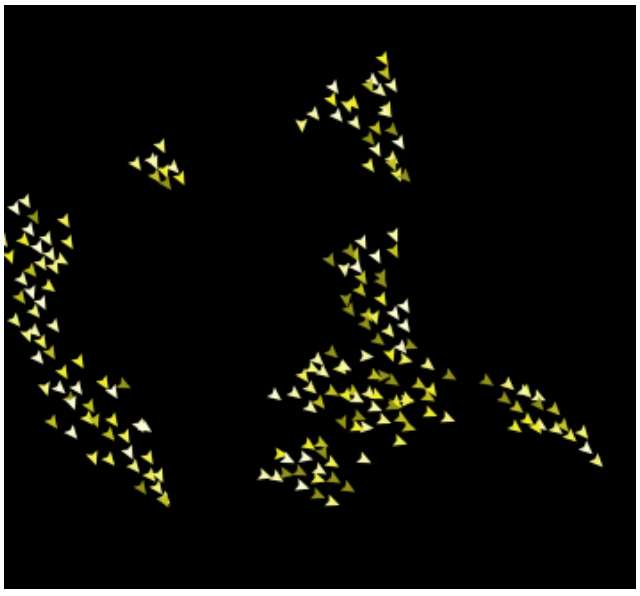


FIG. 3.1. A boid flock moving through a two-dimensional space.

Reynolds Boid Flocking One of the first popular ABMs was the boid flock (Reynolds 1987; 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 in 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 would therefore be 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 forms two-way highways to reduce the number of head-on collisions while ants are

searching for food and returning with 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 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 and group 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 have only 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 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 the critical density at which 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 the comparison this particular type of ABM experimentation and the approach taken by AMF is provided in Section 9.1.

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 such projects aim to create an accurate model of human interactions with each other 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 has either a positive sentiment or a negative sentiment towards the missile facility. Each agents can change its sentiment towards PMRF either by interacting with another agent that has a different sentiment or by experiencing a local event. The model simulates local events and agent interactions and displays how either positive or negative sentiment propagates accross the island. With this model, the authors were able to model hypothetical situations and how they affect 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 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 among 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 whether they 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 do not 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 blocks 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 that 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). More recent 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 tool kits, which model time as discrete events and have grid world environments. In addition, some physics models, such as gravity and object collision resolution with friction, are built-in features of every simulation. Breve's OpenGL display engine

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

allows for easy-to-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 freely available multi-agent simulation toolkit developed at George Mason University.[4] MASON takes a different approach than previous multi-agent tool kits in that it strives to be minimalist and efficient for up to a million agents (Luke *et al.* 2005). It is meant to be run 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 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 using a separate visualization that binds to the simulation. This is a different paradigm than other tool kits like NetLogo that are tightly coupled with the visualization.

NetLogo NetLogo is a relatively new ABM system, which was started in 1999 at 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 ]
```

Other than turtles, there is another built-in agent type: the “patch.” Patches are organized as a grid in a two-dimensional environment to which agents are confined. 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, monitorsm and the domain visualization. Users define controls that 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 to query current variable values. This capability 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, an experienced programmer can learn to write a domain similar to the Wolf Sheep Predation model in less than an hour. 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 as a built-in system-dependent tool. This property 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 provides the foundation for 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 and 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 keeps the form of the function $f(\cdot)$ fixed, but changes parameters to this function, instead. 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 high-dimensional. I also needed algorithms that could model nonlinear correlations, since behavior spaces are diverse 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 smoothness property 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.

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

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. The 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 to the shape of the behavior space. Therefore, KNN requires little to no configuration or tailoring to a particular domain.

However, when implemented naively, the running-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.

Selecting k is an important step, as different values of k will fit the data better than others. If k is too low, the regression will be overfit since it is ignoring many of the points. If k is too large, the regression will be overgeneralized and provide a mapping that is too

smooth. Unfortunately, the nature of the data and the size of the data set determine which k is best. To determine a proper k, I use the standard technique of using 10-fold validation to determine which in a set of candidate k values results in the least average error.

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 inflection points one side of where the k-nearest neighbors will include a particular data point, whereas on the other side it will not. At this separation, there will be a discontinuity 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 $W(\mathbf{x}_i, \mathbf{x}_j)$. which 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 a 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 $\mathbf{x}$s 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 that other points have on 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), whereas, if the smoothing parameter is too high, some features of the curve may be eliminated.

LOESS is computationally expensive in comparison to other approaches, 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. Caching these values provides a quick response time to users wanting to infer $\hat{y}$ values. This property satisfies one of the design criteria of AMF: the regression algorithm should be fast for the user.

3.2.3 Nonlinear Regression

Nonlinear regression (NLR) is a parametric approach to regression. In this approach, a parametric model is provided by the user that represents the relationships in the data, in general. This is typically done based on some sort of intuition provided by 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 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 for each target domain.

Chapter 4

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 an agent-based model. These properties are typically non-explicit, and dependent in complex ways on the explicit parameters, 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 which features should 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 is

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.

4.1 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.

If this assumption does not hold, regression algorithms will bias their predictions. As more points are sampled, the regression algorithm's 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.

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]The Fires model is discussed in more detail in Section 5.1.

zero population. When the wolf population does stabilize, it generally converges upon the same population value, $\bar{w}$. However, the average of successive runs will result in a bimodal distribution, with a single spike at zero and a tight distribution of values around this stable population size, as shown in Figure 4.1. 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 population measure can be decomposed into two components: the average populations in the case where the wolves become extinct and the case when they do not. The average number of wolves $\bar{w}$ can be then be posed as:

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

where $extinct$ is true when the wolves became extinct, $\mathrm{P}(extinct)$ is the probability that the wolves become extinct, and $(\bar{w}|\neg extinct)$ is the average number of wolves, given that they do not become 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 in some cases an experiment will 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 are 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 $\mathrm{P}(extinct)$, one can divide the number of samples that ex-

hibited zero wolves by the number of total samples for that configuration. To calculate the value of $(\bar{w}|\neg extinct)$, one can average the population size of the wolves in the samples in which they did not go extinct.

The histogram in Figure 4.1 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 the 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, since it does not tell us about the stable wolf population size, or how often the system will exhibit zero wolves. In fact, the actual number of wolves is *never* close to $\bar{w}$. 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 more accurately than $\bar{w}$.

In many 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 are on a boundary of a threshold effect can exhibit erratic behavior,as in 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.

[2]The parameters were 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%.

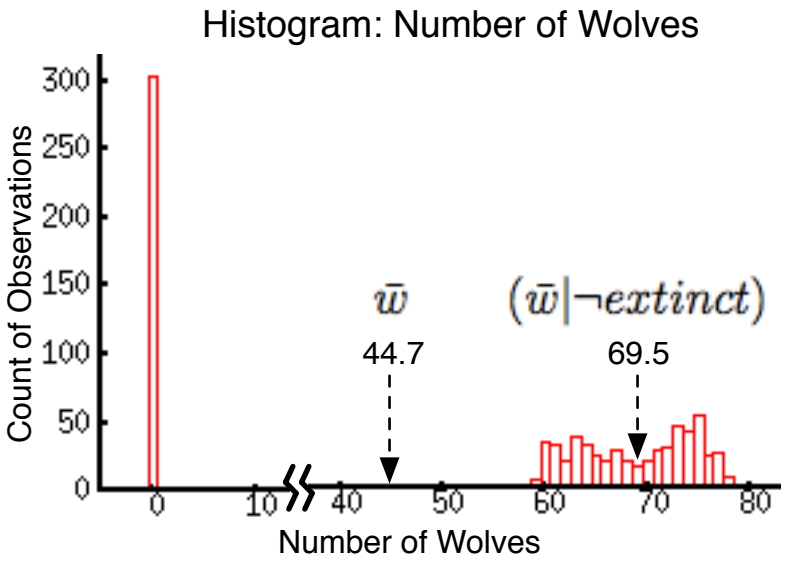


FIG. 4.1. A histogram of the number of wolves after many successive runs shows that the average number of wolves is a poor measure the system behavior.

4.2 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.

4.2.1 Average of a Value

One of the simplest measurements is the average value of a property over the life of the system, which 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 delayed 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

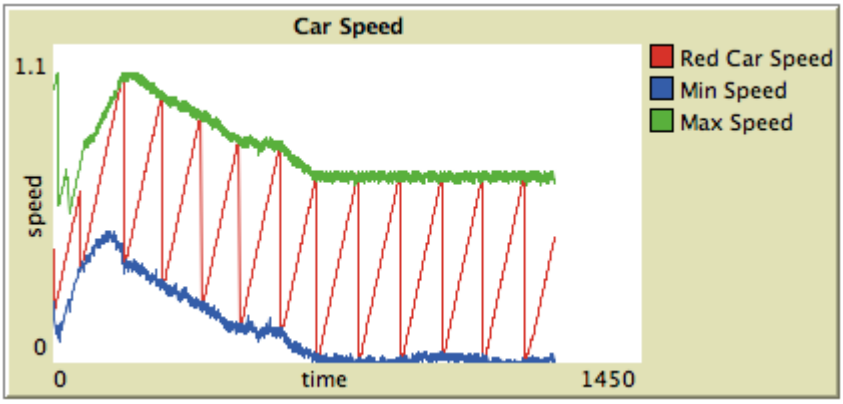


FIG. 4.2. The behavior of the NetLogo Traffic Basic ABM converging after about 650 time steps.

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 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 changing significantly and start measuring from there.

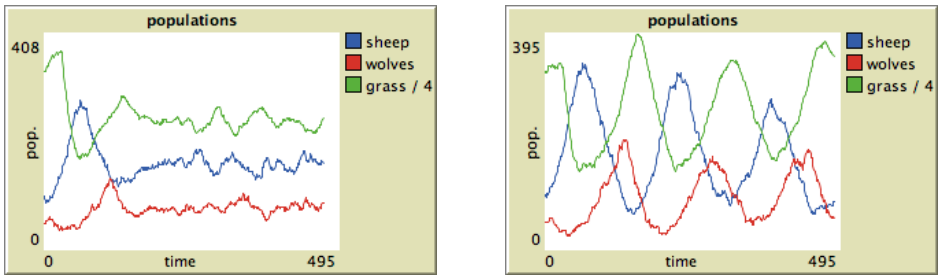
In Figure 4.2 (a plot from the NetLogo Traffic Basic ABM), the maximum speed and minimum speed of the vehicles converge to .65 and 0, respectively. However, before time step 650, the behavior was quite different than the eventual convergent behavior. If each time step (0 to 1250) were averaged, the maximum speed value would be biased to be larger than .65. This number has some 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. Instead, the aver-

age value should 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 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.

4.2.2 Variance of a Value

The variance of a value over the life of a system is a useful metric for determining how "stable" the property is. If the property changes significantly from one time step to another, the variance will be higher than that of a property in which the value remains stable. Figure 4.3 shows a comparison between two situations in which the variance of values are different.



(a) (b)

FIG. 4.3. 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 converges on a single value as more time steps are sampled, for the same reasons that the average metric should converge.

4.2.3 Probability of a Threshold Effect

Threshold effects are binary properties that may or may not appear in a run of 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 vary in which behavior side of the threshold they will converge to. For example, in the Wolf Sheep Predation ABM, wolves may or may not be extinct when the system converges. 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 outcomes 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 system-level property 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.

4.2.4 Measuring a Value That Changes Over Time

The previous measurements assume that the behavior will converge over time, which is not the case for all behaviors. Also, there are situations in which the average value or

the variance of a value does not convey enough information. In particular, properties that change over time may 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. Predicting each behavior 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 of the N parameters. The value for each parameter is predicted by the corresponding mapping, given the system's agent-level configuration. These predicted parameters can be plugged into the parametric model to generate a curve that represents the behavior over time.

This approach can be applied to a number of phenomena, such as the ones illustrated in Figure 4.4. In Figure 4.4(a), the animal populations in the Wolf Sheep Predation model vary rhythmically and could be modeled with sine curves. The parameters of these behaviors modify the magnitude, height, and frequency of the sine curves. In Figure 4.4(b) (a plot from the NetLogo AIDS model (Wilensky 1997a)), the *HIV-* (people without HIV) curve decreases and *HIV+* increases in a sigmoid pattern. The parameters of these two sigmoid behaviors modify the shape of the sigmoid function. The *HIV?* (people who 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 Figure 4.4(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

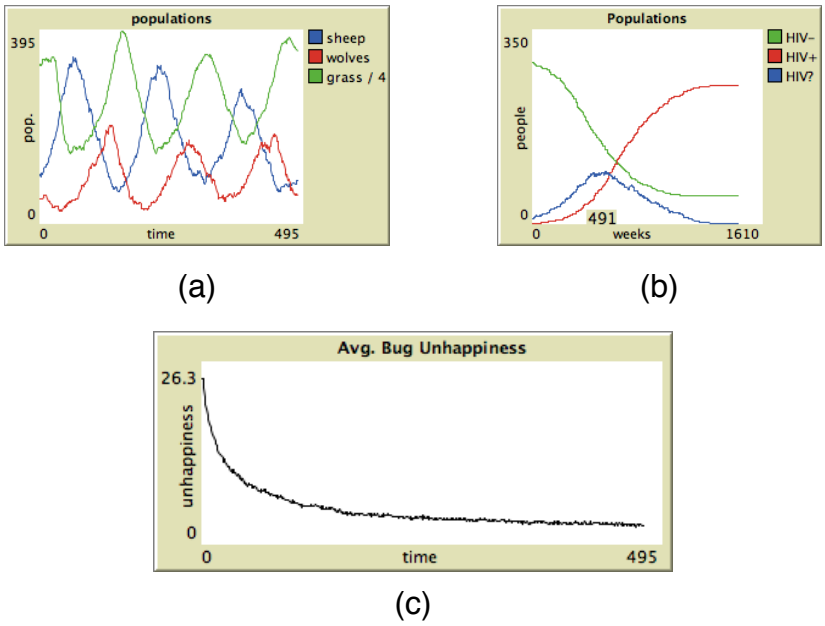


FIG. 4.4. Different types of behaviors that change over time in NetLogo ABMs.

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 speed of the red car,[3] as illustrated in Figure 4.2. 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

[3]Nothing is special about the red car in the Traffic Simple situation. It is singled out to show the traffic pattern from the perspective of an individual.

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 observations, the information necessary 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.

4.3 Sampling

The sampling process may begin once the system-level properties are defined. The AMF sampling process consists of two major steps: (1) retrieve raw data (i.e., observed agent behaviors each time step) from the simulation; and (2) use that data to compute the

values of the system-level properties (e.g., average and variance of specific behaviors). Instead of performing the calculations while sampling, I prefer a method of computing the actual statistics from a raw data set. This process permits new statistics to 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 the side of the data set to be changed 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 assume AMF work with randomly distributed data. More advanced sampling techniques could be used in future work (see Subsection 10.1.1).

Sampling the same point repeatedly, 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, or
- Generate more accurate individual points.

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

4.4 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 four distinct steps:

1. Identify, by name, which NetLogo properties need to be extracted from the simulation in order to calculate the system-level properties, and specify how often (i.e., time steps passed) each property should be measured.

2. Specify all ABM configuration parameters by name, along with the 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[4] 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 of the user provided information is passed to this program in a text configuration file. A program that generates values in an identical format to 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 the standard output (stdout). The data is tab-delimited to separate items within entries; each entry is delimited by a newline. The entries are recorded in the order specified in the configuration file, so the configuration file can be used to give each column an identifiable name. The standard 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 and returns aggregate or expanded statistics, per row. The output of the

[4]The name "map" is to signify similarity to LISP's map function.

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

Chapter 5

APPLICATION DOMAINS

Four target domains are used to demonstrate AMF in action, showing the effectiveness of AMF in a variety of different situations. In Chapter 8: Results, experiments are performed on each of these ABMs to evaluate AMF. No modifications to AMF have to be performed to make AMF compatible with these domains, which shows that AMF is domain independent. In each of the domains enumerated in this chapter, I give a brief overview of what the ABM represents, what configuration parameters are available, and what system-level properties I have defined.

5.1 NetLogo Fires

The NetLogo Fires ABM is the simplest ABM used in this dissertation. In this ABM, trees are distributed throughout a 250×250 grid world. A fire is started in the left column, after which all trees in this column burn. Each burning tree then burns its adjacent trees (up, down, left, and right). This process continues until no more trees are adjacent to burning trees.

The configuration space is one-dimensional and the system-level property space is one dimensional. The configuration parameter for this domain is the density ρ of the forest, which in our experiments I range from 45 % to 75%. This parameter changes the proba-

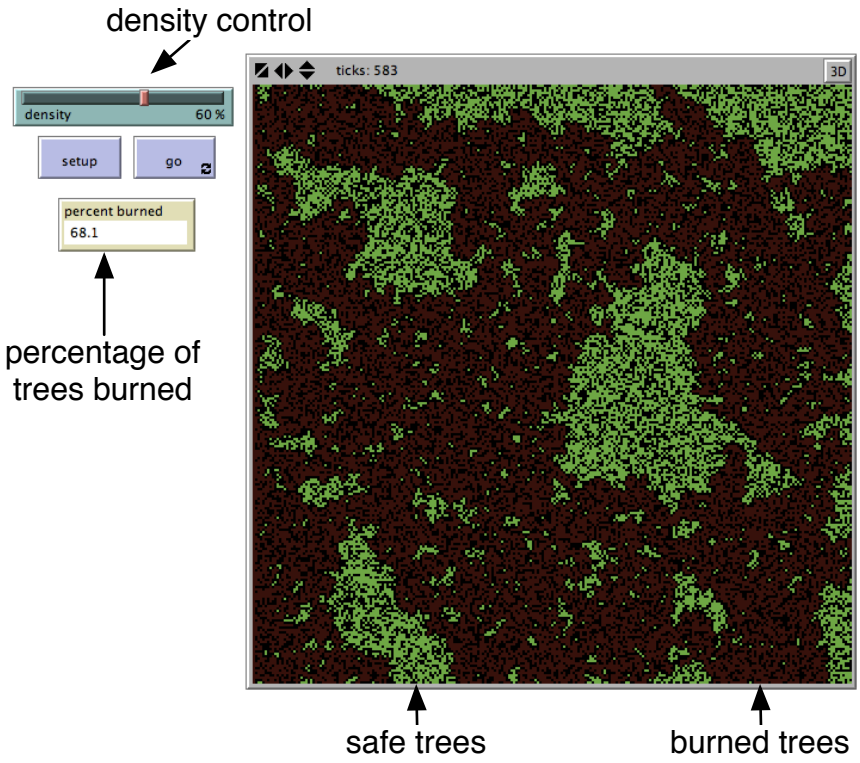


FIG. 5.1. Screen capture of the Fires NetLogo ABM.

bility that a grid point contains a tree: during the initialization step, each grid point has a tree placed with probability ρ. The system-level property is the percentage of trees burned down, which is measured by dividing the number of trees that were burned by the total number of trees originally in the system.

The Fires ABM is particularly interesting because it exhibits a threshold effect. There is a sharp transition from small amounts of the forest burning down and the entire forest burning down when the forest density increases past 58%. The behavior space is not linear and exhibits high variance around 58%, making this ABM moderately challenging, regardless of it only having one configuration parameter.

5.2 NetLogo Flocking

The NetLogo Flocking ABM is a classic agent-based model based on rules devised by Reynolds (Reynolds 1987). As in the the original work, the emergent flocking behavior results from the summation of the following forces:

- *avoidance* – repels agents that are too close to one another,
- *center* – attract agents towards the center of the flock,
- *align* – steer towards the average heading.

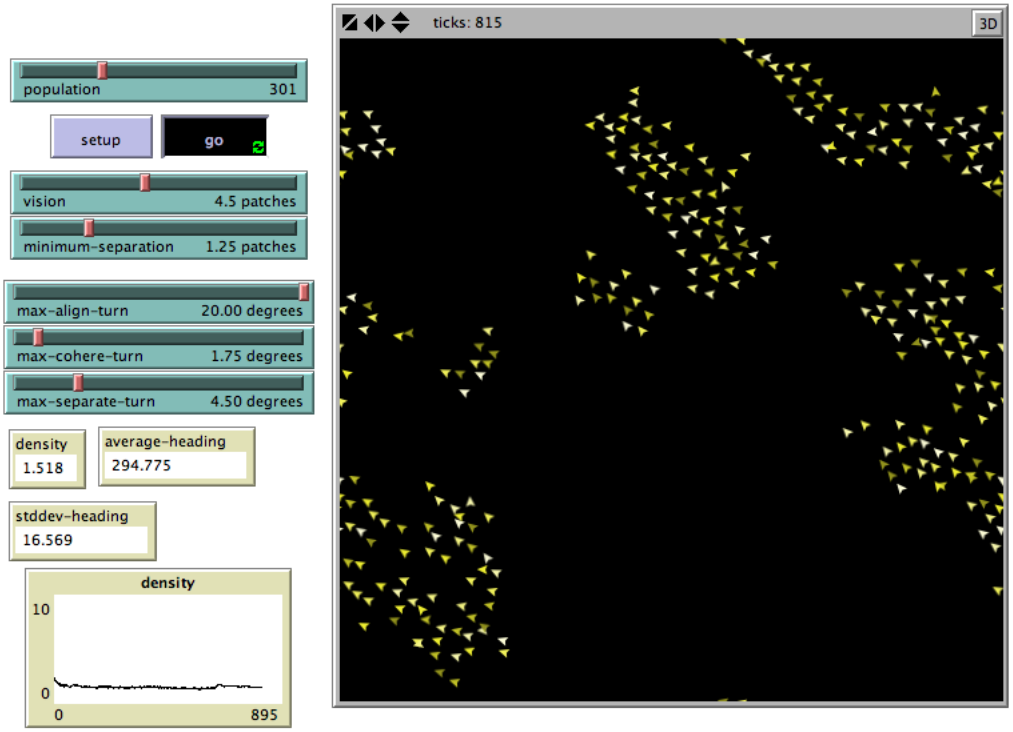


FIG. 5.2. Screen capture of the NetLogo Flocking ABM.

The NetLogo implementation of flocking follows the original definition closely. There are six configuration parameters, as seen in Figure 5.2, however, only three are used:

- *max-align-turn* – the maximum amount of degrees an agent will turn to align its heading with its neighbors,
- *max-cohere-turn* – the maximum amount of degrees an agent will turn towards the center of the flock,
- *max-separate-turn* – the maximum amount of degrees an agent will turn away if another agent is within *minimum-separation*.

The other three configuration parameters available in the user interface (*population*, *vision*, and *minimum-separation*) are excluded because they do not affect the system-level properties as directly as the three outlined above. The values of these parameters are kept constant throughout the experimentation performed for this thesis.

Two system-level properties are analyzed for this system:

- *spread* – measures how spread out the agents are.

  This metric is calculated as the average distance from each agent to its nearest neighbor. This calculation is an approximation to the true density, as it does not directly measure agents per patch. Approximating is necessary because determining the area that an individual flock covers is not trivial and sometimes subjective. In addition, a single instance could have several individual flocks, making this computation even more obscure. The distance to the average neighbor captures the information that density conveys: as the distance increases, the agents are more spread out, and thus less dense.

- *stddev-heading* – measures the standard deviation of the agent headings.

  This metric is used to determine how much the agents in the ABM agree on the direction they are heading. Configurations with agents that flock smoothly in one direction

FIG. 5.3. A swarming boid system (left) and a flocking boid system (right).

typically have a very low *stddev-heading* value. Meanwhile, more chaotic configurations will have a relatively high *stddev-heading* value. This contrast is illustrated in Figure 5.3.

The Flocking ABM has the role in testing AMF that it is less challenging than Wolf Sheep Predation, but more complex than Fires. The system-level properties change in a more monotonic and predictable manner than the other domains, but still remains sufficiently complex. The behaviors are not linear and the configuration space is three-dimensional, making this problem still nontrivial.

5.3 NetLogo AIDS

The AIDS ABM is a model of how AIDS spreads in humans. There are a number of agents in a two-dimensional world that move around randomly. Agents that collide have a chance to couple, and thten have a chance to spread AIDS if one partner has AIDS (*HIV+* and one does not (*HIV-*). Individuals that know they have the disease will not spread it to

people without it. Only individuals that have AIDS but do not know (*HIV?*) can spread the disease. Eventually, once there are no more people unknowingly spreading AIDS, the population is segregated into people who have AIDS and know it, and people that do not have it. A screen capture of the domain running in NetLogo is shown in Figure 5.4.

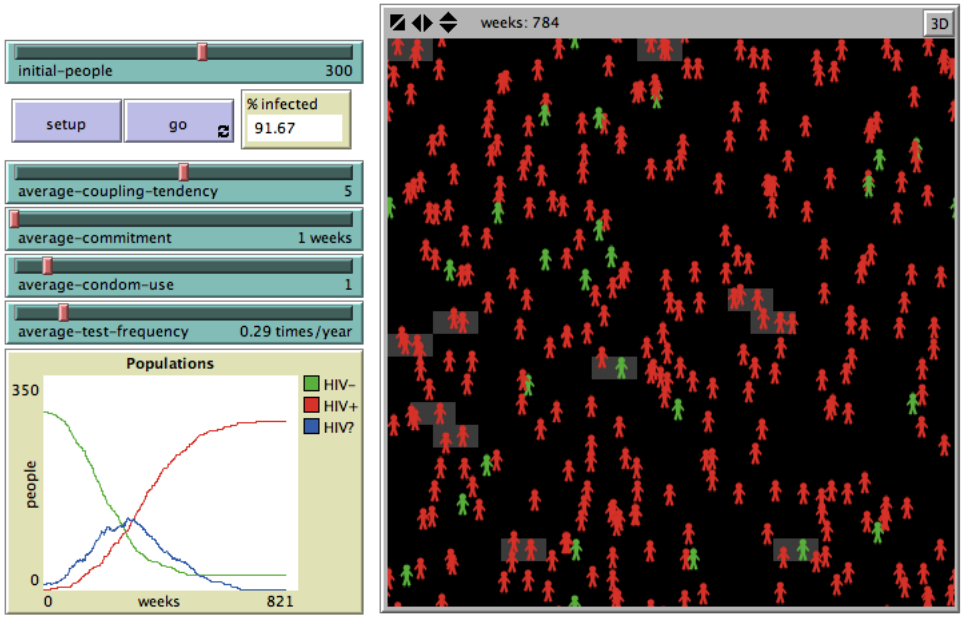


FIG. 5.4. Screen capture of the NetLogo AIDS ABM.

There are five configuration parameters available in the NetLogo implementation of this ABM. However only two are used for the intents of this dissertation:

- *average-condom-use* – The probability that one of the two individuals will insist on using a condom.

  The usage of a condom prevents the spread of the disease. Therefore, when this value is lower, the disease will spread to more people. When this value is higher, the opposite happens. The difference between a high value and a low value is illustrated in Figure 5.5.

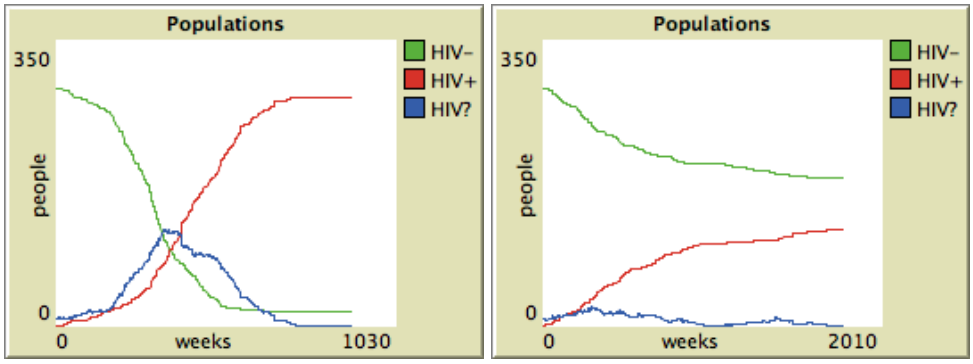


FIG. 5.5. Plot of a AIDS ABM run with low (left) and high (right) *average-condom-use* (notice the difference in the scale of weeks).

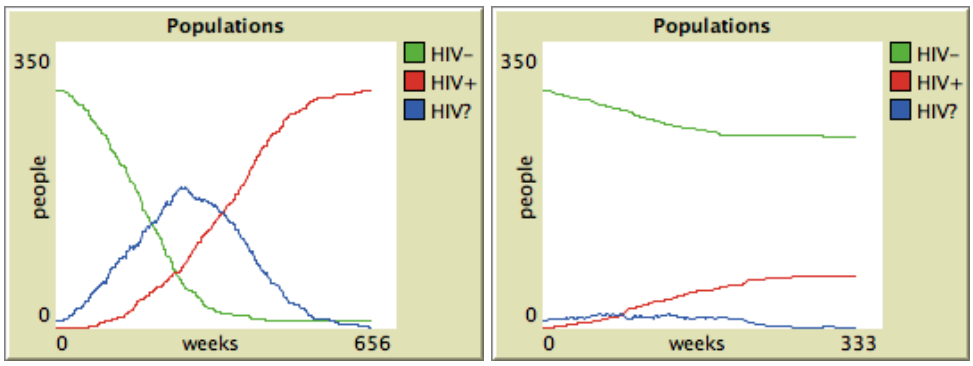


FIG. 5.6. Plot of a AIDS ABM run with low (left) and high (right) *average-test-frequency* (notice the difference in the scale of weeks).

- *average-test-frequency* – The frequency that an individual will get tested for AIDS, per year.

  Getting tested decreases the population of the individuals that have AIDS but do not know it, directly reducing the number of agents capable of spreading the disease. This parameter has as similar effect as *average-condom-use*, as seen in Figure 5.6.

The other parameters, *initial-people*, *average-coupling-tendency* (chance a couple will form), and *average-commitment* (how long a couple stays together) are not used. I chose not to incorporate these in my experiments because they change the system-level behavior in a similar way to the two parameters that I do use. In addition, condom usage and test

frequency may be more easily affected by public policy than coupling tendency and how long a couple are committed for.

The AIDS domain is used to specifically demonstrate the usage of parametric models to describe system-level behaviors over time. For this domain, I will use AMF to model the plots of the populations, such as the ones in Figures 5.5 and 5.6. Nonlinear regression is used to perform curve fitting on representative parametric models, using training instances to learn the parameters. Thus, the forward mapping maps the two configuration parameters to predicted values of the parameters of the model. The appearance of the plots can then be predicted with the forward mapping.

A base parametric model that can fit the data well is required. For this domain, the curves appear to be sigmoid functions, so I will use a generalized logistic function:

$$P(t) = c_1 + \frac{c_2 - c_1}{1 + e^{-c_3(t-c_4)}}.$$

The parameters represent the following:

- c_1 – the lesser asymptote,
- c_2 – the greater asymptote,
- c_3 – the growth rate, and
- c_4 – the location of the inflection point.

The step of performing the nonlinear regression to model the curve is done after the raw data sampling and before solving the forward mapping. Each instance of scatter plot data has nonlinear regression applied to it to determine the four parameters of the logistic function. The parameters are used as the new data set, instead of the original scatter plot data.

5.4 NetLogo Wolf Sheep Predation

The NetLogo Wolf Sheep Predation ABM has been the running example throughout this dissertation. In this ABM, there are three major entities: wolves, sheep and grass. Wolves eat sheep and sheep eat grass. The grass naturally regrows, meanwhile wolves and sheep occasionally reproduce. Every time step sheep and wolves lose some energy and die naturally if their energy reaches zero. A screen capture of the Wolf Sheep Predation ABM in NetLogo is shown in Figure 5.7. The configuration controls are in the top left, the monitors are in the bottom left and the visualization of the domain is on the right.

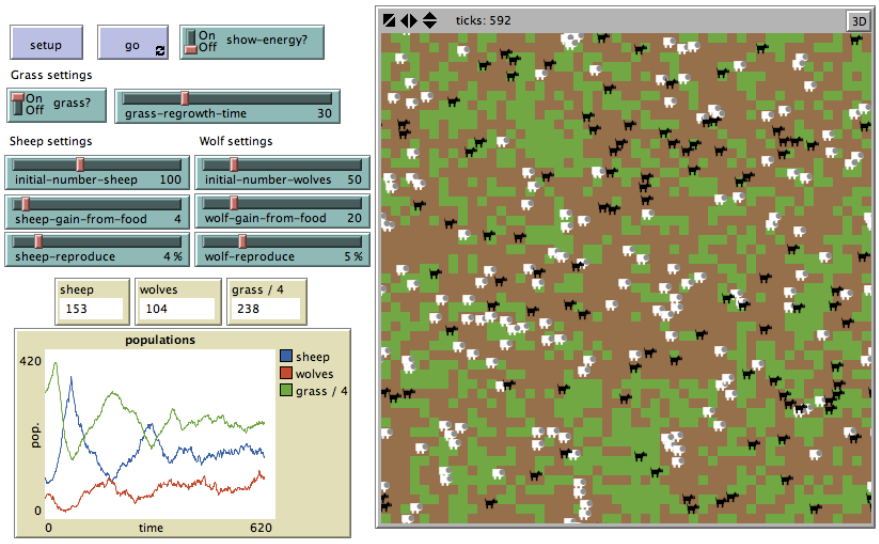


FIG. 5.7. Screen capture of the Wolf Sheep Predation ABM.

The populations of a running Wolf Sheep Predation system are constantly changing. Therefore, measuring properties is not as straightforward as in the Fires ABM. Changes in populations can range from small oscillations to high magnitude oscillations, depending on the configuration parameters. Quantitatively capturing these properties is challenging, but is possible with AMF. A plot of a relatively normal (not too stable and not too high-magnitude) population is shown in Figure 5.8.

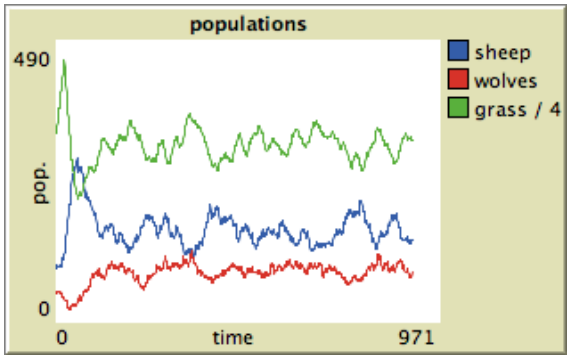


FIG. 5.8. Plot of a stable populations in a Wolf Sheep Predation system.

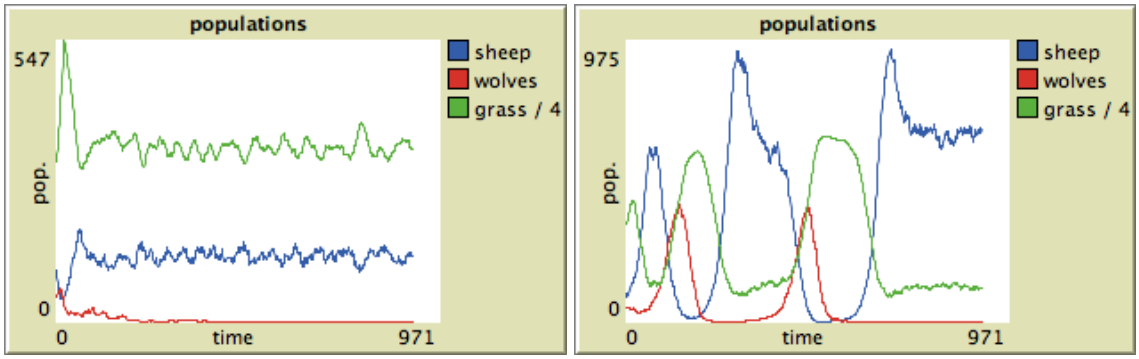


FIG. 5.9. Plot of a Wolf Sheep Predation run with low (left) and high (right) *sheep-gain-from-food*.

The configuration space is five-dimensional, making this domain nontrivial to analyze. The configuration parameters, along with some general observations, are as follows:

- *sheep-gain-from-food* – the amount of energy sheep gain from eating grass.

  This parameter has interesting effects on the system. The effect of changing this parameter from the stable norm is shown in Figure 5.9.

  At lower values, the population of the sheep has difficulty not going extinct. This almost always results in the wolf population going extinct.

  At higher values, sheep live longer and quickly become overpopulated, which causes the grass to become a scarce resource. With little grass left, the sheep begin to die

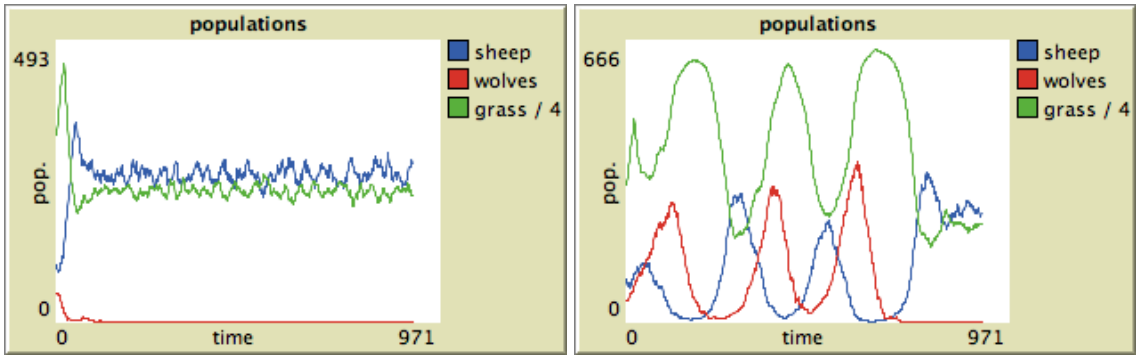


FIG. 5.10. Plot of a Wolf Sheep Predation run with low (left) and high (right) *wolf-gain-from-food*.

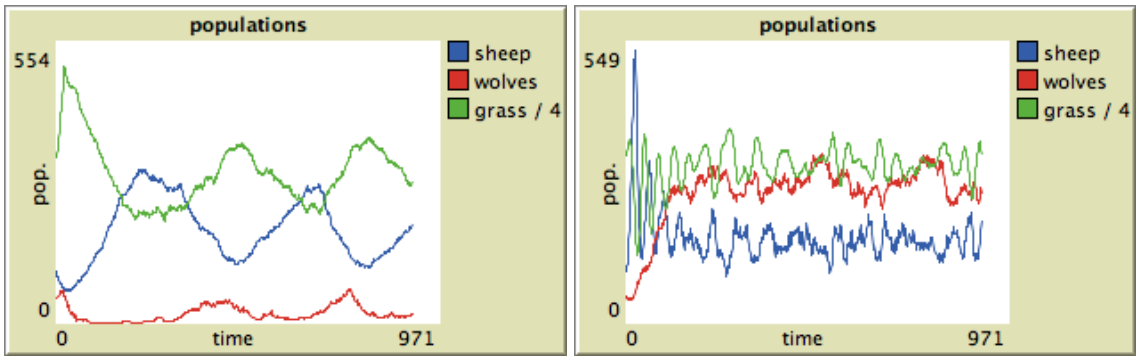


FIG. 5.11. Plot of a Wolf Sheep Predation run with low (left) and high (right) *sheep-reproduce*.

off. Once enough sheep have died off, the grass begins to regrow and the sheep begin to reproduce. This results in sharp increases and decreases in sheep population as the sheep population oscillates between overpopulated and underpopulated. When the oscillations are of high enough magnitude, the wolves have a high chance of becoming extinct while the sheep population is very low.

- *wolf-gain-from-food* – the amount of energy wolves gain from eating a sheep.

  This parameter affects the system in a similar way as *sheep-gain-from-food*. Lower values cause the wolves to go extinct, while higher values cause more drastic oscilla-

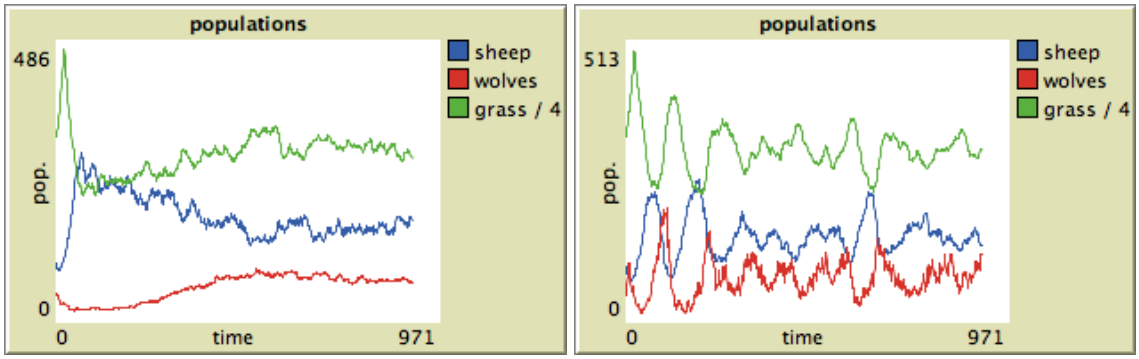


FIG. 5.12. Plot of a Wolf Sheep Predation run with low (left) and high (right) *wolf-reproduce*.

tions in wolf and sheep populations. The effect of changing this parameter from the stable norm is shown in Figure 5.10.

- *sheep-reproduce* – the percent chance that a sheep will reproduce, per time step.

  This parameter has the interesting property that when it is increased, the wolf population increases, but the sheep population remains relatively constant. When *sheep-reproduce* is too low, the sheep population has difficulties sustaining its numbers, which causes the wolf population to be very low. The effect of changing this parameter from the stable norm is shown in Figure 5.11.

- *wolf-reproduce* – the percent chance that a wolf will reproduce, per time step.

  This parameter has relatively little effect on the system, in comparison to the other parameters. The populations oscillate more when the reproduction rate is higher and more stable with it lower. The effect of changing this parameter from the stable norm is shown in Figure 5.12.

- *grass-regrowth-time* – the amount of time steps a patch of grass takes to regrow.

  This is perhaps the strangest behaving configuration parameter. At higher values, not

enough grass grows to sustain a large sheep population, which causes the wolves to go extinct. At lower values, sheep rarely every die, until they become overpopulated and eat all the grass. At this point, there is a mass extinction, which causes the wolves to typically go extinct. This parameter is similar to that of *sheep-gain-from-food* in terms of system-level behavior.

Each of these parameters individually affect the system state. Even more diverse behaviors can be observed by adjusting several parameters at once.

Several system-level properties have been developed for this domain. To sample many of these properties and to provide more consistent results, each random configuration is sampled a number of times and averaged.

- *sheep-extinct?* and *wolves-extinct?* – the probability that the sheep and wolf populations will reach zero, respectively.

  The probability of sheep and wolf extinction is measured by sampling the same configuration numerous times and calculating the percentage of instances they went extinct. Most often with a specific configuration, the sheep or wolves always go extinct or always do not go extinct. However, there are borderline configurations in which the sheep or wolves will go extinct while the system is stabilizing into a rhythm. This metric is useful in distinguishing between these three situations. Wolves will always go extinct if there are no sheep, but sheep can exist without wolves. Therefore, the value of *wolves-extinct* is always higher than *sheep-extinct?*.

- *sheep-average* and *wolf-average* – the average sheep and wolves given they did not go extinct, respectively.

  The average sheep and wolf populations are calculated by averaging the populations at every time step in instances in which they did not go extinct. Not including extinct

populations in the average is important because these instances will introduce significant amounts of bias. This metric describes how large the population can expected to be. Along with *sheep-variance* and *wolf-variance*, the averages describes the nature of the population and how it changes.

- *sheep-variance* and *wolf-variance* – the variance of the sheep and wolf population over the course of a run.

  The variance is calculated with the same data as the average. This metric describes how much the population changes from time step to time step, since the population is typically rhythmically changing.

The Wolf Sheep Predation ABM is interesting because its behaviors are not linear and are not monotonic. That is, as many of the configuration parameters increase, the resulting system-level behavior does not change in a linear manner. This property of the behavior space makes many of the system-level properties difficult to predict. In addition, there are five configuration parameters, which makes the configuration space five-dimensional. The system-level property space is up to six dimensions. Handling dimensionality up to eleven is a challenging task for any approach that is attempting to analyze system-level behavior.

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 (as opposed to directly building a bidirectional mapping from the data).

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.

This chapter details aspects of the forward-mapping problem and the solution that is used in AMF.

6.1 Definition of the Problem

The forward mapping is a functional mapping that maps *configuration space* to *system-level property space*. The configuration space is made up of the dimensions representing all the input configuration parameters of the agent-based model that could affect the system-level behavior. 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 provide user access to the configuration dimensions in the configuration space.

The system-level property space is made up of the dimensions representing all of the user-defined system-level properties that are being measured. Typically, these propertyies are 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 though it is impossible to have .6 of a wolf.

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

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

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 prop-

erties $\hat{\mathbf{y}}$."

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

Table 6.1. Outline of Wolf Sheep Predation Behavior Space

Configuration Space	
grass-regrowth-time	Number of time steps it takes for grass to 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 that 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

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. The most important aspect for any approach to solving the forward-mapping problem is that querying for a prediction must be faster than sampling the system directly and predictions should be as accurate as possible.

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 as training data.

Different regression approaches use this data in different ways. Approaches like k-nearest neighbor (kNN) do not need to perform any pre-processing and instead uses the entire data set for each query. Meanwhile, parametric approaches like nonlinear regression (NLR) train a compact model, using a limited number of parameters to represent the entire data set. In general, the more time that is spent pre-processing the data, less time is 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, and 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. 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 10.1.2.

Linear scaling is an optional step in AMF that can inserted between the sampling phase and the forward-mapping phase. When AMF is queried, the points are rescaled to their original values.

6.2.2 Handling Non-Continuous Configuration Spaces

Ordinal (discrete) configuration parameters are handled as a special case in AMF. The sampling program is told whether or not each parameter is ordinal or real-valued. The random selection of data points will abide by this restraint by randomly selecting integers for ordinal parameters, instead of real values

The best way to handle an ordinal parameter in learning the forward mapping depends on the nature of the variable. If the parameter has a wide rage, so that thousands of different values are possible, or if 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 method has the advantage of providing a common representation across all parameters. 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 method provides more accurate results at the cost of higher computation requirements. When there are numerous different discrete parameters, however, the number of subspaces grows quickly, making this approach intractable.

6.2.3 Handling Multi-Variate Forward Mappings

Although multiple system-level properties can be measured at once, AMF represents each system-level property as an independent single-property forward-mapping problem. The reformulation of the original definition of the forward mapping is as follows:

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

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

$$\{f_0(\mathbf{x}), f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})\} \rightarrow \hat{\mathbf{y}}$$

This approach assumes that the system-level properties are independent of one another. If this assumption does not hold, the predictions may not be accurate. This problem is outside the scope of the implementation of AMF; the test domains that have been used with AMF, the assumption appears to hold.

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 information on how to build this library in order to properly interface with AMF is provided in Appendix A.1.

Training During training phase of solving the forward-mapping problem, any necessary pre-processing and/or model construction is performed. This process is implemented as a Python function called *train*. Most regression algorithms will store models or meta-data in memory 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 the optimal parameters for the specified parametric model by using optimization; the parameters for this model are stored.

Each of these approaches requires varying amounts of computational time, but needs to be executed only once for a given data set.

Predicting The prediction phase of solving the forward mapping problem is invoked whenever a query is submitted. This process is implemented as a Python program called *predict*, but is mostly driven by the user-provided regression module. The *predict* function in AMF uses the models generated by *train* to predict values for individual system-level properties.

As in the training phase, the predicting phase is algorithm-dependent, however, the API users interact with is the same. kNN will iterate over each input point to find the nearest neighbors, then average the values. LOESS will interpolate between nearby smoothed training data points. NLR applies the parametric model using the input configuration values. Each of these approaches requires varying amounts of computational time, as with the training step. Since, this process is executed once per query, efficiency is an important factor.

6.3 Using Forward Mapping to Solve the Reverse-Mapping Problem

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. AMF uses the forward mapping to solve the reverse-mapping problem, using one of many of the approaches discussed in Chapter 7. All of these approaches use the forward mapping instead of directly sampling from the ABM for two reasons. First, querying models of the behavior space is faster than interacting with the ABM directly. Second, the regression models smooth the error over the entire space, effectively eliminating natural sampling noise, which can produce more accurate and consistent results.

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

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 cause the system to 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, $\hat{\mathbf{S}}$ is the set of all the points that would predict $\mathbf{y}$ in the forward mapping.

The major difficulty in solving the reverse-mapping problem is that the mapping is one-to-many because the forward mapping is built with regression (many-to-one). Therefore, querying a reverse mapping returns a solution space, not a singular solution. For this reason, AMF strives to return the *space of configurations* that will exhibit a given system-level property in an ABM, not individual configurations.

This general method of developing a reverse mapping can be contrasted with standard optimization methods. Finding a set of configuration parameters that satisfies a system-level property, by definition, is an optimization problem. However, instead of a single point being returned as in most optimization methods, a reverse mapping returns the space of all points that would satisfy the configuration. This has a number of benefits, such as being

able to visualize and inspect the entire solution space. Analysis of how the solution space changes with respect to the configuration parameters can be performed. Different solutions that produce similar results can be selected from a number of good solutions based on some exterior preference on configurations. That is, although all points in a solution space should be satisfactory, some points may be more desirable for a number of reasons, such as points with lower expected variance.

In this chapter, I give details of how AMF provides solutions to this problem, in general, then discusses actual implementation of the solution. Throughout this chapter, I use the NetLogo Fires ABM[1] as an example. The Fires domain is simple and has solution spaces that are easy to visualize. Other domains, such as Wolf Sheep Predation, have too many dimensions to make graphing feasible. The arguments made with the aid of the Fires domain scale to larger dimensions.

7.1 The AMF Approach: Simplical Complex Inversion

The AMF reverse-mapping approach is a novel approach I developed, named *Simplical Complex Inversion* (SCI). A simplical complex is a set of adjacent simplexes (multi-dimensional triangles) that together segregate a space. In abstract terms, SCI builds an invertible approximation of the forward mapping in the form of a simplical complex. This apprach is used because regression is not easily invertible. I have found that inverting even the simplest approaches such as K-nearest neighbor and LOESS are either difficult to implement or computationally expensive (Section 7.3). By approximating a representation of the forward mapping that is invertible, solving the reverse-mapping problem becomes computationally tractable. These direct inversion approaches are generally not preferred over SCI, since they are dependent on the forward mapping approach used and SCI is ap-

[1] More on the Fires model is discussed in Section 5.1.

plicable to any forward-mapping technique. *SCI is building an invertible approximation of a trained forward mapping.* This is in contrast to building a reverse mapping directly from the training data.

As with the forward-mapping solution, the system-level property $\mathbf{y}$ 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 of 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.

Simplical Complex Inversion has three phases. First, regression is used to infer the system-level property values for several "knots"[2] in the configuration space. The locations

[2]The term "knot" is taken from Multilinear Interpolation(Davies 1997) (section 9.4).

of these knots are used to segregate the configuration space into a number of simplexes. This segregated configuration space is referred to as a *simplical complex* (Munkres 1993). Next, the intersection between each of these simplexes and a plane representing the target system-level property is found. That is, SCI derives the equation of the hyperplane through each simplex that satisfies the target system-level property. A piecewise linear represention the configurations that would satisfy the target system-level property is pieced together from all the intersections. Figure 7.1 illustrates the SCI process; the rest of this subsection dscusses these steps in detail.

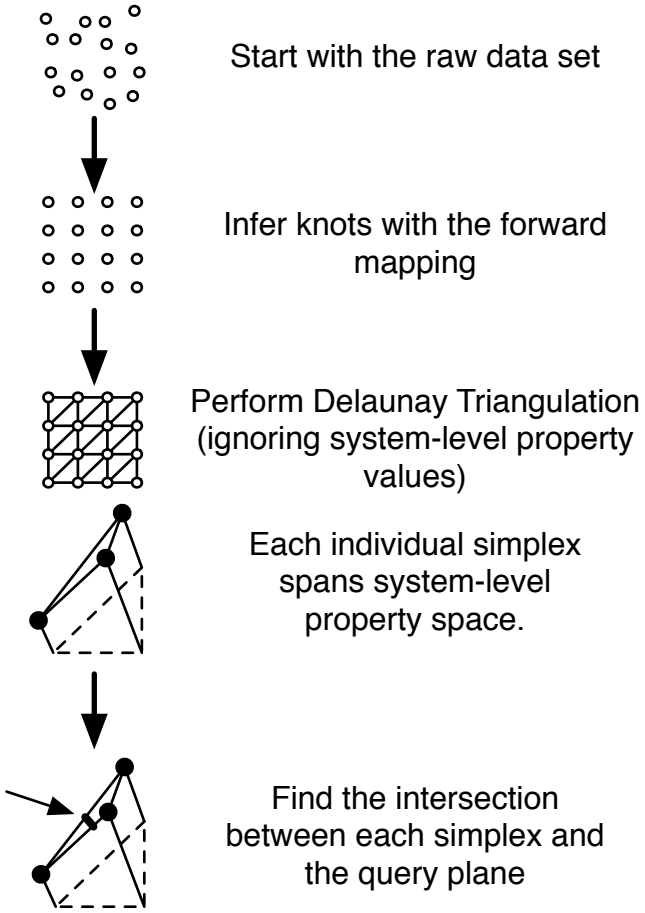


FIG. 7.1. The flow of stages in the Simplical Complex Inversion approach to the reverse-mapping problem in a two-dimensional configuration space.

I claim that SCI is a general approach to the reverse-mapping problem that is applicable in all domains in which the forward-mapping problem is solved accurately. Also, SCI provides more useful information than optimization, since SCI returns the *set* of all configurations. These hyperplanes give a more continuous and complete view of the nature of a specific system-level property value than individual points.

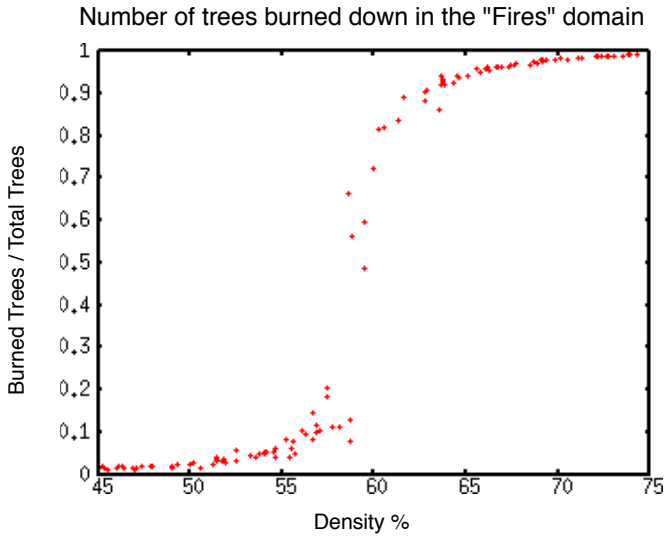


FIG. 7.2. The raw data set, containing 120 points, from the Fires domain.

7.1.1 Segregating the Configuration Space

The reason for using regression as the first step is that the raw data in many ABMs will exhibit variance. For example, raw data from the Fires ABM is shown in Figure 7.2. If this data were to be used with interpolation, the curve for the behavior space would be very erratic. This is a problem for SCI because erratic curves will result in a number of inaccurate intersections. A linear interpolation performed on the raw Fires data to generate a behavior space curve is shown in Figure 7.3. From this figure, the curve appears very erratic and would provide poor results with SCI.

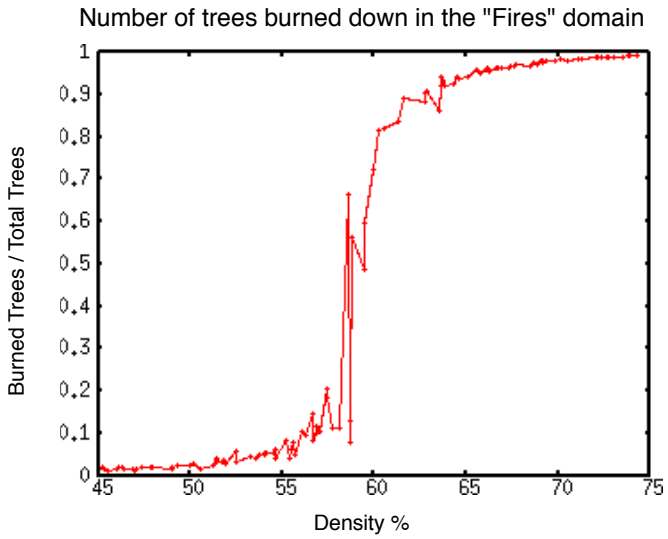


FIG. 7.3. Linear interpolation performed on the raw data set (see Figure 7.2) from the Fires domain.

The behavior space is naturally much smoother when mapped to the regression space. The first step of SCI is to use the forward mapping (i.e., regression) to infer knots in the space, which are evenly distributed across the range of sampled values. The granularity of the knots is specified by the user. These knots cover the space and provide reference points for the interpolation. To generate these points, the forward mapping is queried by passing a complete vector of configuration parameters. An inferred system-level property value is returned. Each data knot in an n-dimensional[3] configuration space is represented by a vector of configuration space parameter values ($\mathbf{x}$) and the system-level property value (y):

$$< x_1, x_2, ..., x_n, y > .$$

The configuration space is then segregated with Delaunay triangulation(Delaunay 1934) into adjacent simplexes (simplical complex). These simplexes have $n + 1$ corners

[3] n refers to the dimensionality of the configuration space for the rest of this chapter.

(e.g., they are triangles if $n = 2$ and lines if $n = 1$) and spans $(n + 1)$-dimensional space. Note that the behavior space resides in $(n + 1)$-dimensional space, but the simplexes are n-dimensional objects (e.g., a tetrahedron in four-dimensional space if $n = 3$ space or a line segment in two-dimensional space if $n = 1$). This is because the system-level property y is not taken into consideration for the the Delaunay triangulation. The relationships between these objects' dimensionality and different dimensional configuration spaces are outlined in Table 7.1. For example, in the Fires ABM (1-dimensional configuration space), the space is segregated into line segments that reside in two-dimensional space.

Table 7.1. Dimensionality of Objects for Simplex Inversion

Configuration Space Dimensionality	Behavior Space Dimensionality	Simplex	Intersection
1	2	line	point
2	3	triangle	line
3	4	tetrahedron	plane
4	5	pentachoron	3D hyperplane
5	6	5-simplex	4D hyperplane
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	$n + 1$	n-simplex	$(n - 1)$D hyperplane

If the configuration points were randomly scattered, a standard algorithm for computing the Delaunay triangulation could be used. However, since SCI controls how configuration points are distributed, it can organize the points in such a way that a solution to the Delaunay triangulation can be computed implicitly. In general, SCI will evenly space the sample points as a grid and split each hypercube into two simplexes. The Delaunay triangulation is trivial to compute with points organized in this way. For example, in Figure 7.1, the space is segregated into a number of adjacent triangles that satisfy the requirements for being a Delaunay triangulation. To segregate the space into a Delaunay triangulation from a regular grid, each hypercube is split into two simplexes. The two simplexes are defined

as the simplex defined with all the corners except for the corner closest to the origin, and as the simplex defined with all the corners except for the corner furthest from the origin. For example, with a square, two triangles are formed with two opposite corners as the right angles for their respective triangles.

The equations of the simplexes can be defined in terms of the hyperplane in which they reside. The $n+1$ points define a n-dimensional hyperplane. If a point is within all the facets and lies on the hyperplane, the point is on the simplex. For example, in the Fires ABM, the two points defining a line segment reside on a line that extends infinitely in both directions. Meanwhile, the facets are defined by the points individually. If a behavior space point lies on the line and resides between the two facet points, the point is a plausible configration that would generate the desired system-level behavior. This explanation expands to two-dimensional configuration spaces, which has triangles as simplexes and line as facets. If a point lies on the plane defined by the the points, and is within the boundaries of the line segment facets, the point is realistic.

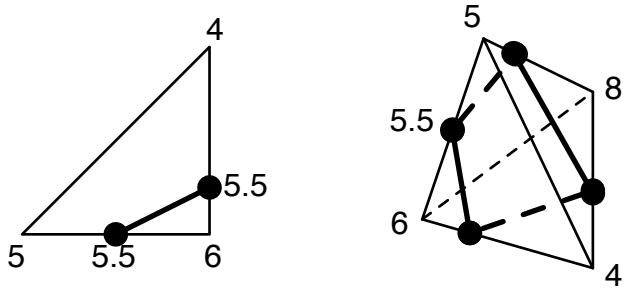


FIG. 7.4. A triangle from a two-dimensional configuration space (left) and a tetrahedron from a three-dimensional configuration space (right) are intersected with the plane $y = 5.5$. The y-values for each corner are given.

7.1.2 Simplex Intersection

Recall that the final goal of this process is to find the equation of the intersection between all the simplexes and a system-level property hyperplane. Therefore, no complicated technique needs to be performed to calculate the actual equation of the simplex or the facets. Edge intersections are used to approximate the intersection through the simplex. The process to find the intersection between the system-level property hyperplane and a simplex is as follows:

1. Pair off each corner with every other corner, in every possible combination. These point pairs define the edges of the simplex.

2. Determine which edges intersect with the hyperplane. An edge intersects with the hyperplane if the target system-level property value is between the y-values for the two points that define the edge. From the intermediate-value theorem, the plane must intersect with this line segment at some point, at least once.

3. Use linear interpolation to determine at which point on the edge line segment the intersection occurs. For example, suppose that the corners have y-values of 2 and 4 and the target system-level property value is 3. The location of the intersection is halfway between the corners.

4. The set of all edge intersection points lie on the hyperplane that intersects the simplex. Use these points to define the intersecting hyperplane.

This approach circumvents the need for modeling the actual simplexes as equations, which can be challenging to implement. Two illustrative examples of this intersection method are shown in Figure 7.4. In the example on the right-hand side in Figure 7.4, the two-dimensional plane intersects the tetrahedron in four places. However, only three of these

points are needed to define the plane because the fourth point is coplanar and thus redundant. This illustrates the fact that not every edge has to be checked for intersection, only enough points to define the intersecting hyperplane. Since the intersecting hyperplane is $(n-1)$D, n points are needed. SCI uses the following standard representation for a plane, where $\mathbf{C}$ is a vector of constants and $\mathbf{x}$ is the vector of configuration parameters and system-level property:

$$\mathbf{C} \cdot \mathbf{x} + D = 0$$

$$C_0 x_0 + C_1 x_0 + ... + C_n x_n + C_{n+1} x_{n+1} + D = 0.$$

This approach is very general and will work in most cases; however, when the plane intersects with either a corner point or an edge only, no plane can be defined because not enough non-colinear points can be found. To circumvent this problem, SCI considers the boundaries of the simplex as not part of the simplex. Thus, a point that intersects with only the corner or an edge will not be considered in the final piecewise intersection.

7.1.3 Recombination of Solution Spaces

Once solution spaces have been found for each of the desired system-level properties, they are recombined into one solution space. This final solution space will be a piecewise hyperplane function, consisting of points that exhibit the specified system-level property values. Since the individual reverse mappings are hyperplanes, the recombination is the intersection of all these hyperplanes.

First, SCI verifies if the hyperplanes intersect within the simplex. Two hyperplanes intersect if the points on one hyperplane are separated by the other hyperplane. If the points are separated, then there must be an intersection. An illustration of points being separated by a hyperplane, and not, is shown in Figure 7.5. This intersection check is performed by plugging in values for the edge intersections of one hyperplane into the equation of

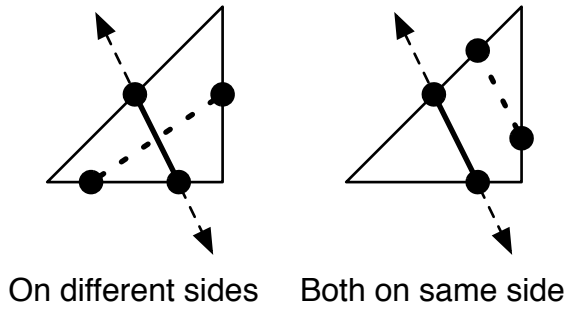


FIG. 7.5. Illustration of two recombination scenarios: intersecting (left) and not intersecting (right).

other hyperplane. Then, the nature of the inequality (greater-than or less-than) for each point specifies which side the edge intersection is on. If one point is on the other side of another point, the planes intersect. This calculation is easy to perform since SCI has the edge intersection points and the equation of the hyperplanes already computed. Also, this process can be repeated for any number of hyperplanes by performing this test for each possible pair of hyperplanes. However, it may be the case that the hyperplanes do not intersect in the same places, even if all of the hyperplanes intersect, pairwise.

To find the intersection, a system of linear equations is set up in matrix form:

$$\begin{bmatrix} C_{0,0} & C_{0,1} & \dots & C_{0,n+1} & D_0 \\ C_{1,0} & C_{1,1} & \dots & C_{1,n+1} & D_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ C_{m,0} & C_{m,1} & \dots & C_{m,n+1} & D_m \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n+1} \end{bmatrix} = \mathbf{0}$$

To solve this system of equations, SCI reduces the coefficient matrix to reduced row echelon form (RREF). What remains is a system of reduced linear equations that are more concise in defining the intersection. From the check performed as the first step, any intersectios returned by the solution of this system of equations is guaranteed to be within the

simplex.

The RREF can reveal that the system is either overspecified or underspecified. A system of equations is overspecified if there is a row in RREF that is all zeros, except for the last column value $D \neq 0$:

$$\begin{bmatrix} 0 & 0 & ... & 0 & D \end{bmatrix}.$$

This row translates to $0 = D$, which is not true. That is, there is no possible configuration of $\mathbf{x}$ that will satisfy this specific equation. Thus, the set of system-level properties the user has provided is not able to be created with this ABM. This is typical if the user has specified too many requirements of the system-level properties. In this case, the user should remove some less important system-level property requirements and allow the system to suggest values for them.

RREF reveals that the system is underspecified if there are any rows with multiple non-zero values in it, other than D. For example, consider the row pulled from a matrix in RREF:

$$\begin{bmatrix} 0 & 1 & 2 & 0 & 5 \end{bmatrix}.$$

This row translates to $x_1 + 2x_2 = 5$. Therefore, x_1 and x_2 are free variables, but are still constrained by this equation. An infinite number of solutions can satisfy the system-level properties in this case. This is not a problem like being overspecified, since there are a number of possible solutions available to the user.

7.1.4 Granularity

SCI can be configured by adjusting the granularity of the knots. SCI approximates an invertible continuous representation of the forward mapping. Therefore, as the granularity of SCI increases, it models the forward mapping more accurately. Thus, the granularity should be as high as computationally acceptable.

Granularity can be adjusted dynamically on a query-by-query basis. Individual simplexes may be separated into several subsimplexes. These new simplexes are composed of the original simplex's corners, in addition to new corners. The new corners have their system-level property value inferred from the forward mapping. Once these new simplexes have been formed, nothing special has to be performed to accommodate them in the general SCI algorithm. In addition, even if the granularity has been increased for a specific query, the finer grained simplexes can remain to increase the accuracy of future queries.

Dynamic adjustment of the granularity process can be performed automatically to increase the accuracy of simplexes that contain significant amounts of variability between the system-level property values at the corners. Increasing the granularity will reduce the variance, since points closer to one another will have increasingly similar values.

The effect of the granularity on accuracy is dependent on the nature of the forward-mapping regression method used. If the forward-mapping produces overfit or otherwise inaccurate predictions, the reverse-mapping produces will be inaccurate as well. This difficulty should be dealt with in the forward-mapping solving step, not the reverse-mapping solving step. This is because the purpose of SCI is to model the forward-mapping, not the domain.

An empirical analysis of the relationship between granularity and accuracy is provided in Chapter 8. The accuracy is measured as the approximate total difference between the continuous forward-mapping function and the continuous simplical complex. In addition, suggestions made from SCI are validated against an actual system run of the target ABM. The error between the desired system-level property values and the actual system-level property values are measured. This measurement is redundant, since the error between the actual run and the reverse-mapping solution should be the sum of the forward mapping error and the reverse mapping error (in respect to the forward mapping).

7.1.5 Example: Fires ABM

Figure 7.6 illustrates the progression of SCI from step to step when used for the Fires ABM. First, the forward-mapping problem is solved, with any regression method. Then, the knots are formed in evenly spaced intervals: in this case intervals of 1% tree density (see top right image in Figure 7.6). The simplexes in the case of the Fires ABM are line segments, since the configuration space is one-dimensional. Therefore, linear interpolation between the points produces the simplexes (see bottom left image in Figure 7.6). Each segment is also the only edge of the simplex.

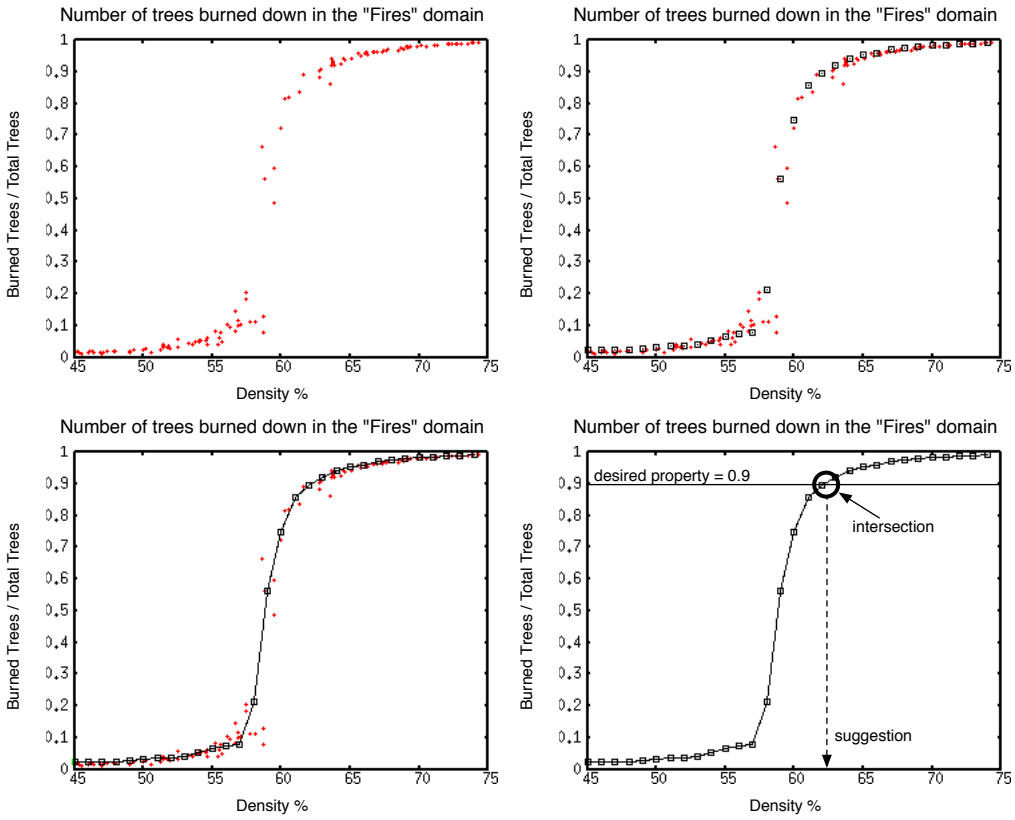


FIG. 7.6. The progression of SCI in the Fires ABM from raw data set (top left), to intersection (bottom right).

Next, SCI finds the intersection between the behavior space and a desired system-

level property behavior $b = Burned\ Trees/Total\ Trees = 0.9$ (see bottom right image in Figure 7.6). This is done by checking each simplex to see if 0.9 is within the range of the minimum value and maximum value for b in each simplex. The only segment that intersects the 0.9 plane is the one between 63% and 64%. The intersection between this segment and the line is calculated, which in this case is approximately at $Density = 63.1\%$.

Therefore, setting $Density = 63.1\%$ will be expected to yield 90% of the trees burned down. In higher-dimensional domains, the number of points that will satisfy the system-level property will be infinite instead of just one point, since they lie on a continuous hyperplane.

7.1.6 Special Cases

In this section, I discuss two special cases that can simplify the SCI process significantly. Users of AMF can see significant improvements in computation time if either of these cases can be identified as a property of a domain.

Classification System-level properties that are binary (positive and negative) in nature (i.e., the forward-mapping is classification, not regression) can be treated differently. SCI handles binary-valued system-level properties by labeling each simplex as either:

- positive – corners are all positive instances,
- negative – corners are all negative instances, or
- inconclusive – corners are a mixture of both positive and negative instances.

Then, once a query is submitted with binary values as part of the requirement, SCI removes all simplexes that do not match that binary value. For example, if a query contains a positive binary value, all negative and inconclusive simplexes are removed from the search.

Inconclusive simplexes are removed since there are probably a number of simplexes which will satisfy the system-level property constraint. This is possible because these simplexes will not be part of the solution, even if other system-level properties could be satisfied by configurations within these simplexes. This shortcut can significantly reduce computation time if a majority of the space is removed.

Inconclusive simplexes are situated on the boundary of positive and negative instances in the space. Therefore, the granularity of the inconclusive simplexes can be increased by separating them (Subsection 7.1.4) in order to reduce the amount of space occupied by inconclusive simplexes.

One-to-One Mapping With some properties, the forward mapping may define a one-to-one mapping. If this is the case, then regression can be used for the reverse mapping by training on the system-level property to infer values for the configuration parameters. The regression will return the single configuration that satisfies the system-level property, since it is one-to-one. Therefore, if a user is working with a one-to-one mapping, this should be the only property being analyzed. Only one point is returned, so adding additional constraints will typically over specify the system of equations and not return anything.

The percentage of trees burned down in the Fires ABM is an example of a one-to-one relationship, since the percentage of trees burned down increases monotonically with the density of the trees.

7.2 Alternative Approach I: Optimization

In previous work, researchers typically treat the problem of finding a configuration that will satisfy some system-level constraint as an optimization problem. In this type of approach, an optimization meta-heuristic such as simulated annealing or genetic algo-

rithms can be used to incrementally adjust the system configuration until the system-level properties match the final goal. These approaches use a concept of fitness that measures the actual outcome with the expected outcome. ABMs that exhibit closer outcomes are weighted higher so that the algorithm gradually converges on a solution.

The main reason that this approach is a poor choice for the reverse-mapping problem is that sampling the ABM on every iteration can be very slow. To calculate the fitness of a particular configuration, the ABM must be executed and measured. Running the Wolf Sheep Predation model takes at least one second per run. Every generation of a genetic algorithm could have to execute an ABM several hundred times, which case, each generation could take several minutes to compute.

A hybrid approach is possible that significantly reduces the online running time of optimization. Instead of directly sampling the ABM, the optimization technique can use the forward mapping to infer the outcome of a configuration instead of executing it. This approach also has the benefit that the regression smooths the predictions, which in many cases will give more consistent results than querying an ABM directly. This approach is effectively searching through the regression space created by the forward mapping, which is an approximation of the true behavior space.

7.3 Alternative Approach II: Functional Inversion

Some regression approaches can be directly inverted. This has the benefit that no approximation, such as SCI, has to be performed, however, a functional inversion is algorithm-dependent, meaning a new method must be implemented for each regression.

I will provide functional inversions of kNN and NLR, as examples. kNN is conceptually and computationally difficult to invert. The "forward mapping" of kNN is: What is the average of the values for the k nearest point to configuration vector $\mathbf{c}$? The inverse of

this statement is: What space covered by k neighboring points would produce the average p? One way to compute this, would be to generate a structure similar to a Voronoi diagram that segregates the space into sections. Every point within a section has the same k-nearest neighbors, which is iterated over to find averages that are close enough. This search can be computationally intensive as the number of sections can be large with large high values for k. Also, since this space is discontinuous, finding a desired average may be impossible. With SCI, continuous transitions are inserted between the discontinuous spaces, allowing intersections to be inferred, even if no average exists.

Nonlinear regression, on the other hand, is rather simple to invert with algebra, with certain parametric models. Basically, inverting NLR reduces to satisfying the following equation:

$$f(\mathbf{x}, \beta) - y = 0,$$

where β are the learned model parameters and y is the desired system-level property value. This equation can be solved with a number of numerical root finding methods. Inverting nonlinear regression, in particular, is useful because it provides an elegant solution in the form of mathematical equations, instead of the piecewise ones provided by SCI.

7.4 Using Reverse Mappings

The main purpose of generating a reverse mapping is to gain understanding of the space of configurations that generate a particular system-level behavior. The relationships between configuration parameters can be inferred by analyzing these spaces. For example, when one configuration parameter value is reduced and the other is increased a specific amount, the resulting system-level behavior may be the same. The equations returned by SCI provide this mathematical representation in a piecewise way. This deeper understanding of how a system-level property can be produced is more faster and potentially more

valuable than having a researcher manually study a domain to gain understanding of how the configurations affect system-level behavior.

The reverse mapping can be used to suggest a configuration when a user wishes to see a particular behavior exhibited in the ABM. For example, suppose the user of AMF wants to determine what an execution of a Fire ABM that burns down 80% of the trees looks like. SCI can suggest what density of trees should be used. Typically, the user would have to go through an iterative process of guessing a possible configuration, then executing the ABM to determine if the configuration generated the behavior or not. This is effectively manual hill climbing, and can be unintuitive, difficult and tedious.

To suggest a configuration with AMF, the user queries the reverse mapping with the desired system-level property values. SCI returns the space of configurations that would satisfy this query. Any point in this space will be satisfactory; however the user must select one. Whether this point is selected arbitrarily or not is up to the user. To select an arbitrary point, a random simplex is chosen. Then, a set of values that will satisfy the system of equations associated with that simplex are derived. Free variables have their values randomly selected, within the range of the simplex. The fixed variables are fixed, so their values are already given. Together, these points provide the user with a suitable point in the solution space to configure the target ABM.

Certain points in the solution space can be considered better than others, even if they should produce similar output. Typically, areas of the space that have minimal variance are safer choices for configuration suggestions, since the expected error is less. As an aside, the action of removing high variance areas of the space is possible by adding a binary valued system-level property that separates the space into “high variance” and “low variance” spaces, given a specified threshold. This approach has the benefit that the selection of configurations with low variance is performed inside the framework, instead of by the user.

7.5 Summary

The reverse-mapping problem is the problem of finding a one-to-many mapping that maps system-level property values to the set of ABM configurations that would exhibit that behavior. AMF uses an algorithm I devised, named SCI, that builds an *invertible* approximation of the forward mapping. This approximated mapping can be inverted and thus return the inverse of the forward mapping. An alternative to this approach is optimization, which is used as a baseline for SCI's performance. Results detailing the computational time required offline, the computational time required per query, and the accuracy of the reverse mappings developed by AMF are discussed in Chapter 8.

Chapter 8

RESULTS

In this chapter, I discuss experiments that test AMF for accuracy and computational efficiency. These experiments also serve as examples of AMF being applied to a variety of ABMs. In the next section, the different evaluation criteria are outlined. Each of these metrics will be applied to a number of domains using a number of different techniques for the forward mapping. The four domains used for the sake of experimentation were outlined in Chapter 5. Each of these domains have unique properties that provide unique challenges to show the versatility of AMF. The different regression methods for the forward mapping are outlined in Section 8.2. Finally, in Section 8.3, the statistical results are discussed.

8.1 Evaluation

Evaluating AMF is important because it shows that the framework is a practical and useful tool for studying ABMs. Evaluation is split into two processes: evaluating the forward mapping solution and evaluating the reverse mapping solution. Accuracy of these mappings in modeling the behavior space of the target ABM is paramount in importance. If the forward and reverse mappings are not accurate, there would be no reason to use this framework for investigating behaviors of ABMs. Computation time is also an important factor in evaluating AMF. Interactions with the user need to be quick in order to make us-

ing AMF more convenient than manually inspecting the target ABM. The response time of when a user interacts with AMF is the *online* time and is far more important than the *offline* time, the one-time computation requirement to learn the forward or reverse mapping.

8.1.1 Forward Mapping

Accuracy Accuracy is the measurement of how closely the forward mapping represents the true behavior space of a target ABM. To measure this, the difference (error) ε between the predictions for a system level property $\hat{y}$ that the forward mapping produces and the actual value y for a sampled point: $\varepsilon = |\hat{y} - y|$. This measurement is performed many times for a single domain to produce the median error and the the upper and lower quartiles of the errors.

The errors are gathered by performing cross validation. Two subsets of data points are randomly pulled from the master set of samples to create a training set and a validation set. The forward mapping is built by using the training set, and then checked with the validation set.

The error is calculated for varying size data sets to show the relationship between error and data set size.

Offline Computation Time Offline computation time is the average amount of system time the forward mapping uses to train a model. This is measured as the amount of time the pre-processing takes, given a training set. The size of the training set is varied to determine how forward mapping training time is correlated to training set size. Each forward mapping method is compared to the others.

Online Computation Time Online computation time is the average amount of time AMF takes to return the results of a user-submitted query. This is measured by performing

a large number of queries and calculating the average run time. The size of the training set can affect the amount of time a query will take with some algorithms. In these cases, the size of the training set is varied to determine how query time correlates to training set size.

8.1.2 Reverse Mapping

Accuracy vs. Forward Mapping Accuracy of the reverse mapping is measured in two ways. The first is how accurately the reverse mapping models the forward mapping. Recall that AMF solves the reverse mapping by building an invertible approximation of the forward mapping. This metric determines if this approach is doing what is expected to do. The effect that granularity has on this accuracy is also measured by learning the same mapping with increasingly higher granularity.

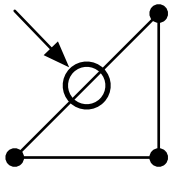


FIG. 8.1. The most erroneous spot in a right-angled simplex.

The error is presented as an "average worst-case scenario". The area of the reverse mapping space that is be the most erroneous is typically be the one furthest from the knots (corners of the simplex). An illustration of this point is presented in Figure 8.1. The point is halfway between all corners, except for the corner at the right angle. This point is the most erroneous if the forward mapping is monotonically changing from one side of the simplex to the other (i.e., there are no local maxima or minima within the space). Note that the reverse mapping will be identical to the forward mapping at the knots, since the forward mapping was used to infer their values.

Calculating the approximate upper bound on error, within a simplex, involves interpo-

lating the system-level property value at the most erroneous point. This is done by average the system-level property values of all corners not at the right angle. Also, the location of the most erroneous point is queried for prediction through the forward mapping. The difference between the interpolated value on the simplex and the actual predicted value using the forward mapping constitutes the error. This metric is applied for every simplex and then averaged. Thus, the metric measures the average error over all the worst-case scenarios for each simplex. Also, the distribution of these errors is recorded to convey how consistent the reverse mapping is.

Accuracy vs. Agent-Based Model Run The main goal of the reverse mapping is to be able to suggest configurations that would generate a specified behavior. This approach is different from the previous metric because it measures the difference between the reverse mapping and the true behavior space.

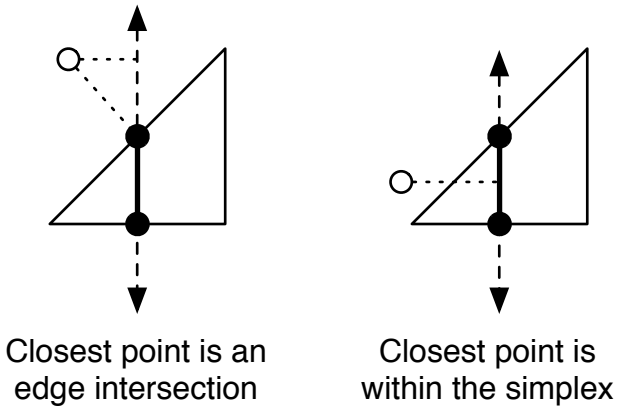


FIG. 8.2. Two cases for the closest point to a simplex intersection.

Error is measured by generating a random configuration point and sampling it from the agent-based model. Then, the system-level property that was measured is passed to AMF to generate a reverse-mapping solution. The error is the distance from the original

configuration and the reverse-mapping solution space. This value is calculated by adhering to the following procedure, per simplex:

1. Determine the distance from the original point to the intersecting hyperplane.
2. If the shortest line from the hyperplane to the point intersects the hyperplane *within* the simplex, then the distance from the hyperplane is the distance from the intersection to the point.
3. Otherwise, the closest point is one of the edge intersections.

The two different cases (closest to an edge (left) and closest to the hyperplane (right)) are illustrated in Figure 8.2. The minimum of all of these closest simplex points is taken as the distance from the actual system-level property value to the piecewise curve generated by the reverse mapping.

Several random configurations are sampled in this way. The average and standard deviation of these errors are calculated to convey the accuracy of the reverse mapping, as well as the consistency of the reverse mapping.

Offline and Online Computation Time Offline computation time and online computation time is measured the same way for the reverse mapping as the forward mapping. The amount of time AMF takes to prepare the reverse mapping is the offline computation time, and the amount of time required to produce a reverse mapping solution is the online computation time.

8.2 Forward Mapping Methods

For the experiments in this chapter, the three algorithms described in Chapter 3 are used for the forward mapping. In the following outline, I indicate specific configuration parameters or implementation details of importance.

- k-nearest neighbor (KNN)
  - Custom implementation with Python.
  - 10-fold validation is used to determine the best k out of a set of possible k values.
- robust locally weighted regression and smoothing scatterplots (LOESS)
  - Custom implementation with Python.
  - A good window is selected by myself for each domain. Changing this configuration parameter per sample size does not affect accuracy, as it does with kNN.
  - Performs one smoothing operation. I found that increasing the number of smoothing operations past one provided limited improvement in accuracy and occasionally worsening accuracy. Finding an optimal smoothing with cross validation, per training set, would be computationally expensive.
- nonlinear regression (NLR)
  - Implemented in Python and uses the SciPy[1] package to optimize the parametric model with least-squares.

8.3 Experiments

First, I will discuss results from the Fires domain together, which provides an extended example that explains the methodology and results for each of the seven experiments. The rest of the experiments for the other domains are grouped together by experiment, so that the effectiveness of the different components of AMF are evaluated independently.

[1]SciPy is a freely available Python module that has a number of useful computational tools. The SciPy website is: http://www.scipy.org/

8.3.1 Fires Domain Experiments

The NetLogo Fires domain is an excellent example for showing how AMF works, as the domain is relatively simple and easy to visualize. Plots of the behavior space for this domain are shown in Figure 7.6.

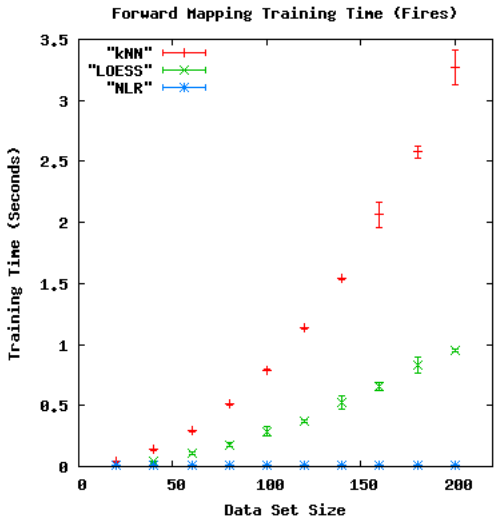


FIG. 8.3. The average length of time required by the forward mapping to train. The error bars show the 95% confidence interval for the mean.

Forward Mapping Training Time Different training data set sizes, ranging from 20 to 200 (with a step of 20), were used to measure the amount of computation time the forward mapping requires to train a model. Each of the three regressions were retrained twenty times per training data set size. The results are plotted in Figure 8.3. This plot shows that kNN grows faster than the other two approaches with increasing data set sizes. This is because the 10-fold validation for picking k requires a significant amount of time. Meanwhile, LOESS grows at a linear rate. The training time for NLR is insignificant in comparison to the other approaches. Least-squares optimization is a mature subject area, and SciPy uses some of the most advanced methods to make the optimization quick.

Even the slowest of these regressions, kNN with 200 training instances, runs in only 3.25 seconds, which is acceptable for a one-time process.

The error bars show that there is little variability in training time, meaning that the approaches are consistent in the amount of time they take.

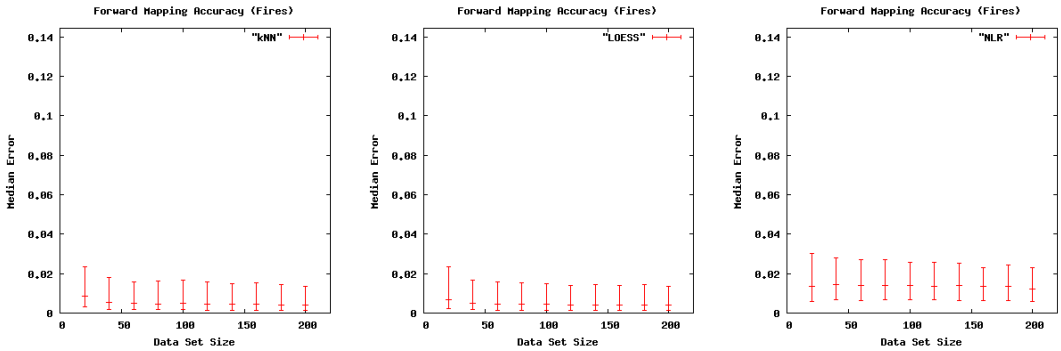


FIG. 8.4. ...

Forward Mapping Accuracy Different training data set sizes, ranging from 20 to 200 (with a step of 20), were used to train a model. Fifteen different training data sets were produced per data set size. For each of these trained models, 2500 points from a validation set were used to evaluate the accuracy of the forward mapping, in respect to values sampled directly from ABMs. The configuration of each validation instance was passed into the forward mapping to retrieve a predicted value. This predicted value was then compared to the sampled value from the validation instance to produce an error value.

The accuracy of the forward mapping is very high, for all three approaches: all three approaches predict the amount of trees burned down within an accuracy of 1%. The median error, along with the upper quartile and lower quartile are plotted in Figure 8.4.

This is impressive considering to the amount of variability from sample to sample of the Fires domain. Both KNN and LOESS perform better with more data points, however after a sample size of 80, the accuracy does not appear to get affected very much. Overall,

KNN had a lower median error than LOESS, but not by a statistically significant amount. NLR performed worse than the other two approaches and experienced more variance in its errors. Interestingly, the accuracy of NLR only slightly improved from a data sample size of 20 to a data sample size of 200. Since NLR is a parametric approach, the optimal model for a small number of points will be very similar to the optimal model for a large number of points from the same distribution. Meanwhile, locally weighted nonparametric approaches have the ability to fit sections of the behavior space better as more points are provided. This trade off is demonstrated in the accuracy results.

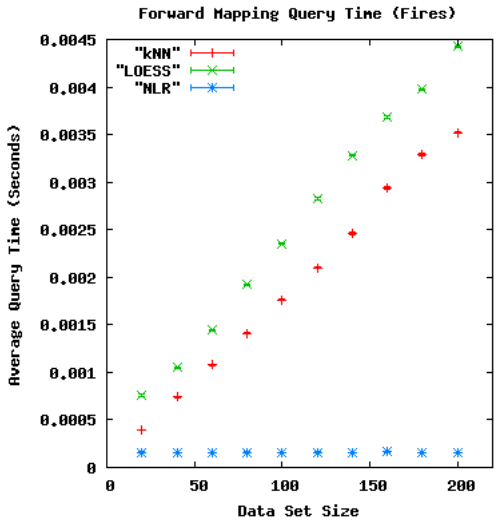


FIG. 8.5. ...

Forward Mapping Query Time Measurements of the query time were performed at the same time as the accuracy measurements. A time was measured for every prediction call to the forward mappings.

Queries to the forward mappings for the three approaches are fast. The longest time for a query to execute was .0045 seconds for LOESS using a data set of size 200. This amount of time is instantaneous to a human user and thus satisfies the requirement that the forward

mapping is fast for the user. As expected, data set size has no effect on the computation time of NLR because the model equation is always the same number of parameters. Also, as expected, LOESS and kNN query time grows linearly with the size of the data set. For every new point added, an additional point must be checked to see if it is local to the passed in configuration, or not.

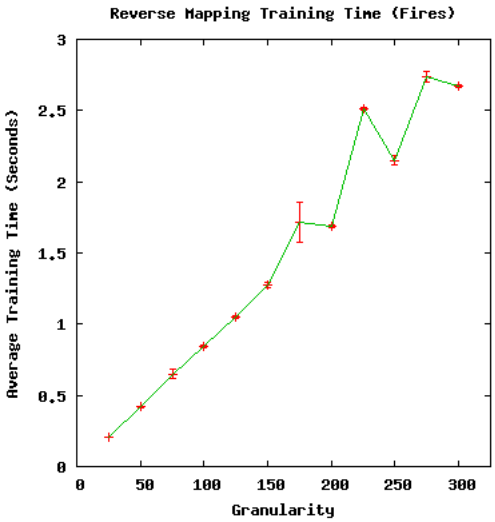


FIG. 8.6. ...

Reverse Mapping Training Time The most accurate forward mapping trained in the previous experiments was used for the reverse mapping experiments. In this case, it was kNN with a training set of 200. Training time was measured for reverse mappings training on granularities of 25 to 300 (step 25). A granularity of N means the configuration space was split into N simplexes, per dimension. The test was repeated fifteen times per granularity.

The training time increased linearly with an increase in granularity. With a granularity of three hundred, the training time was under three seconds. With a granularity of twenty-five, the training time was just under a quarter of a second.

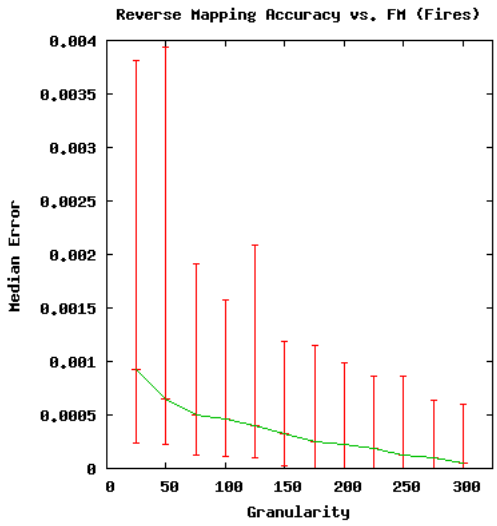


FIG. 8.7. ...

Reverse Mapping Accuracy The reverse mapping accuracy is measured in two ways. First, I measured the median error of the reverse mapping in respect to the predicted values of the forward mapping. This metric demonstrates the accuracy of fifteen different reverse mappings per granularity, as an approximation of the forward mapping. This metric is measured by taking each simplex in the reverse mapping and selecting the furthest point from the corners. Each of these points are then passed to the forward mapping to provide a prediction, which is then compared to the interpolated value within the simplex. The difference between these two denote the error of the values within the simplex, in respect to the forward mapping it is approximating.

Figure 8.7 shows the effect of granularity on median error. The error steadily decreases with higher granularity, eventually approaching 100% for about half of the queries. Also, the error is extremely low in all cases, never exceeding half a percent of trees burned down. The reverse mappings trained with high granularities were often accurate within a tenth of a percent of trees burned down.

The second way the accuracy of the reverse mapping is measured is how distant the

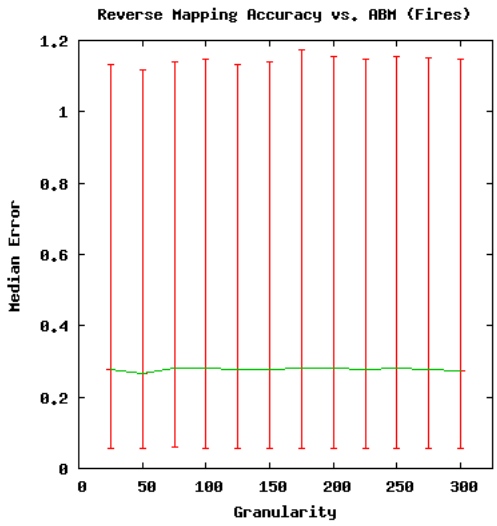


FIG. 8.8. ...

piecewise curve returned by the reverse mapping is from the actual system-level property configuration. Recall that the reverse mapping returns a set of intersections (however, in the case of Fires typically only one, since the mapping is one-to-one). This metric intends to measure the distance from a configuration suggested by the reverse mapping, and the actual configuration from the validation set. Each of the 2500 instances from the validation set were used for each of the reverse mappings that were trained.

The results in Figure 8.8 show that the median error remains fairly constant, regardless of granularity. The reason for this is that the error incurred by the approximation of the forward mapping is insignificant in comparison to the error of kNN. Since the same kNN forward mapping was used for all granularities, the error incurred by the reverse mapping is not noticeable or statistically significant.

Reverse Mapping Query Time The reverse mapping query time was measured along with the accuracies of the reverse mapping, in respect to the validation set.

Overall, the average reverse mapping query computation time is almost instantaneous,

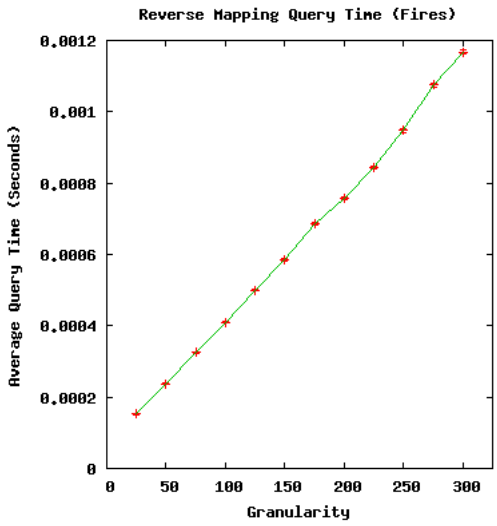


FIG. 8.9. ...

taking about 0.0012 seconds with a granularity of 300, as seen in Figure 8.9. The average query time increased linearly with the granularity because each additional simplex must be checked for intersections.

8.3.2 Forward Mapping

Accuracy

Offline Computation Time

Online Computation Time

8.3.3 Reverse Mapping

Accuracy vs. Forward Mapping

Accuracy vs. System Run

Offline Computation Time

Online Computation Time

Chapter 9

RELATED WORK

9.1 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)

9.2 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.

9.3 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).

9.4 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 are 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 an 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.

Chapter 10

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.

10.1 Future Work

10.1.1 Advanced Sampling Techniques

10.1.2 Advanced Scaling Techniques

Appendix A

USER GUIDE FOR THE AMF SOFTWARE

A.1 Regression Plug-In Interface

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

Appendix B

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

B.1 Sample Java/NetLogo Sampling Program

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

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] Delaunay, B. 1934. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* 7:793–800.

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

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

[14] 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.

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

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

[17] 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*.

[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] Munkres, J. 1993. Simplicial complexes and simplicial maps. *Elements of Algebraic Topology* 7–14.

[29] 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.

[30] 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.

[31] 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.

[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.