

Rule Abstraction:  
Understanding Emergent Behavior in Swarm Systems

Ph.D. Dissertation Proposal

Don Miner  
University of Maryland, Baltimore County

February 16, 2009

Advisor: Dr. Marie desJardins

Project website: <http://maple.cs.umbc.edu/~don/projects/SAF>

# Contents

|           |  |           |
|-----------|--|-----------|
| <b>1</b>  | <b>Introduction</b>                            | <b>3</b>  |
| <b>2</b>  | <b>Related Work</b>                            | <b>4</b>  |
| <b>3</b>  | <b>Rule Abstraction</b>                        | <b>6</b>  |
| <b>4</b>  | <b>Rule Hierarchies</b>                        | <b>7</b>  |
| <b>5</b>  | <b>Algorithms for learning</b>                 | <b>8</b>  |
| 5.1       | Hill climbing . . . . .                        | 9         |
| 5.2       | Simulated annealing . . . . .                  | 9         |
| 5.3       | Sample/interpolate . . . . .                   | 10        |
| 5.4       | Inverted decisions trees . . . . .             | 11        |
| 5.5       | Inverted kernel density estimation . . . . .   | 11        |
| 5.6       | Inverted k-nearest neighbor . . . . .          | 11        |
| <b>6</b>  | <b>Domains</b>                                 | <b>12</b> |
| 6.1       | Form Circle . . . . .                          | 12        |
| 6.2       | Boids . . . . .                                | 12        |
| 6.3       | Geometry . . . . .                             | 13        |
| 6.4       | Wireless Sensor Network Layout . . . . .       | 13        |
| 6.5       | Particle Swarm Optimization . . . . .          | 14        |
| <b>7</b>  | <b>Evaluation Criteria</b>                     | <b>15</b> |
| <b>8</b>  | <b>Preliminary Results</b>                     | <b>15</b> |
| 8.1       | Form Circle with Sample/interpolate . . . . .  | 16        |
| 8.2       | Boid Density with Sample/interpolate . . . . . | 18        |
| <b>9</b>  | <b>SwarmVis</b>                                | <b>21</b> |
| <b>10</b> | <b>Research Plan</b>                           | <b>22</b> |

# 1 Introduction

In this dissertation proposal, I present a framework for learning how swarm systems will behave, given their operating parameters. Frequently when studying swarm systems, one has an understanding of the low-level local behaviors as well as an understanding of the high-level emergent behaviors. What is not typically understood is how these levels relate to one another. The contribution of my work is a system for learning correlations between low-level local behaviors and high-level emergent behaviors.

Complex systems are comprised of a large number of individual and independent agents, operating autonomously. Agents in these systems observe their local environment and then behave according to some program. What makes these systems interesting is the emergent behavior that is resultant from the low-level interactions. Examples of complex systems in the physical world include ant colonies, bee swarms, multi-celled organisms, fluids, gasses, and street traffic. All of these systems consist of small agents (ants, bees, cells, particles, cars) that produce some high-level emergent behavior. Researchers have developed computational models of swarm systems, such as bird flocks [27], ants [2][6], locusts [4], and fish schools [24]. Also, naturally occurring systems have served as inspiration for many practical applications, such as optimization algorithms [7][9], computer graphics [25][26], data visualization [12][21][23], and multi-robot systems [13][14][16][22][30]. All of these systems are implemented at the *agent level* and have no intuitive connection to the high-level emergent properties of the system. The developer of the software must have an understanding of how the low-level parameters will affect the high-level properties in order to implement the system. Even so, from personal experience in working with these systems, tweaking the parameters (i.e., debugging the emergence) is a tedious process. This problem is even more confounded when the software is passed on to another developer: low-level parameter values are often obscure and seemingly arbitrary. For example, biologically accurate computational models of ants [6] and fish [24] have tens of parameters, all with different value ranges with short and vague descriptions. My goal is to make this process more intuitive by using machine learning and search techniques to learn correlations between low-level parameters and emergent properties of the system. This way, users and developers can interact with the system through a top-down interface, instead of manually tweaking low-level parameters. Similar to how a high-level programming language abstracts away unnecessary machine-level details, I plan to do the same with controlling swarm systems.

Rarely is there an obvious, explicit and mathematical understanding of how the parameters of the low-level workings affect emergent properties. For example, high-level emergent properties of gasses are easily identifiable: pressure, temperature, density, etc. These properties result from the interactions and individual behaviors of the particles in the system. This low-level behavior of the particles is dictated by rules of physics, each of which may contain a number of parameters and constants. However, determining a mathematical model of how the properties of the particles affect the high-level emergent properties is not trivial, even if one has an explicit model of the physics of the system. My goal is to use established and novel machine learning and search techniques to build models of the correlations between emergent behavior properties and the low-level parameters through a learning process named *Rule Abstraction*. This concept is illustrated in Figure 1 (left). Is it possible to apply machine learning or search algorithms to develop models of how the local interactions of

particles affect high-level emergent properties of a gas?

Another goal of this work is to demonstrate that emergent properties in some systems can be decomposed into a layer of lesser emergent properties. These lesser emergent properties are the result of some low-level behavioral properties of the lowest unit in the system (e.g., a particle in a gas, an ant in a colony, a neuron in a brain). Algorithms are applied to learn correlations between the low-level properties and the lesser emergent properties, and vice versa. This structure is illustrated in Figure 1 (right) and should remind the reader of a multi-layer neural network with an input layer, output layer and a hidden layer. This structure is called a *Rule Hierarchy*, which models emergent systems more intuitively and efficiently. Rule hierarchies can be extended to any number of layers.

Rule abstraction and rule hierarchies will be usable for a diverse set of applications. In general, rule abstraction can be used for *classification* of emergent systems and for providing *swarm-level control*. Classification of swarm systems is defined as predicting how the swarm system is expected to act without observing the system in action. For example, rule abstraction will be able to identify the density of a flock of agents, before the system is executed—just given the low-level parameters. Swarm-level control is defined as determining what low-level parameters the system should use to satisfy some set of desired high-level properties of the system. For example, rule abstraction will be able to determine what low-level parameters should be used, given a desired density value of a flock of agents.

The specific contributions of this work includes the concept of rule abstraction and the framework it provides. In addition, this work will enumerate several implementations of algorithms for rule abstraction and domains it has been applied to for use as examples in future work. In this proposal, I will formalize the rule abstraction and rule hierarchy mechanisms. I will provide a survey of learning and search algorithms that I have applied to learn correlations between high-level and low-level parameters. Then, I will discuss the several application domains which I propose to apply rule abstraction to. Finally, I conclude this document with a research plan.

## 2 Related Work

There has been work in developing “swarm-level control” for specific domains, in which researchers develop mathematical formulas [30] or describe swarm-level properties of systems at an abstract level [6][16]. All of these previous efforts required human intuition to develop the models. Rule abstraction is significantly different from these approaches because it provides a domain-independent learning framework that can provide mathematical models of abstract properties automatically. A human is only needed to set up the necessary pieces of the rule abstraction—the rest will be handled by the learning algorithms.

For example, in work by Spears et al. [30], the authors develop a physics-based control policy for a swarm of mobile robot agents. With this, the swarm-level behavior of the system can be described as a mathematical equation. This mathematical analysis of the system is useful and precise, but will not always extend to other domains. Also, some level of human intuition was required to develop these equations. Rule abstraction may not develop models that are as accurate as the theoretical behaviors developed by Spears et al., but it will do so autonomously with methods that can be extended to other domains.

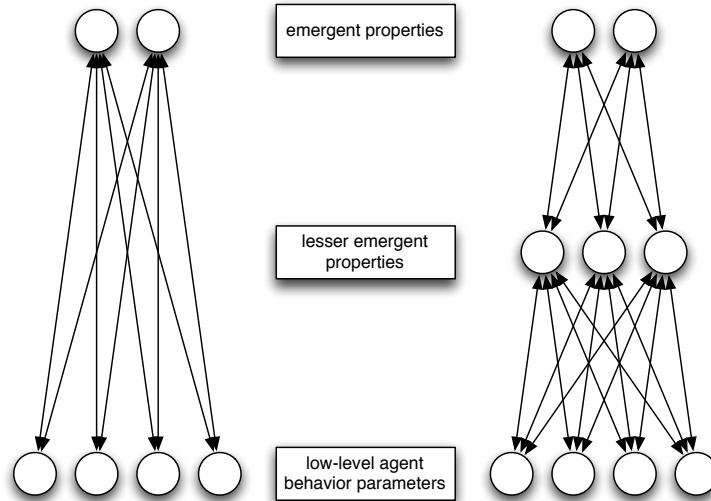


Figure 1: A rule abstraction correlates values of emergent properties with low-level agent behavioral parameters (left). Rule hierarchies are an extension of this idea, and are comprised of any number of rule abstraction layers (right). The lowest level (the low-level agent behavior parameters) are variables in the complex system software. All higher-level layers are abstract properties of the resultant emergent behavior. Each double headed arrow represents an individual “rule abstraction” process.

McLurkin has developed a library of swarm robot macro actions that provides an intuitive swarm-level interface to a swarm system [16]. These macro actions include follow the leader, form groups, disperse evenly. This work is another example of human intuition being used to develop swarm-level control of a specific system.

Lerman et al. discuss “macroscopic models” of swarm robot systems [13][14] and is similar to rule abstraction in motivation. In this work, the authors model agents as finite state automatons (FSA) and then describe the agents as Markov processes. With this approach, the macroscopic behavior of the swarm can be studied by analyzing the number of agents in any state at a particular time. Rule abstraction differs from this in two major ways. First, the authors do not discuss how inverted analysis could be applied: that is, given a value for the macroscopic behavior, generate parameters for a swarm with that property. However, this was probably not an explicit goal of their work. Second, the way the authors model agents is more restrictive than the model used in rule abstraction. Some swarm systems, such as boid flocks, behave based on a sum of forces, not as FSAs. Therefore, Lerman et al.’s work may not be able to intuitively model such a system. Meanwhile, rule abstraction’s view of an agent can represent an FSA: the parameters in a rule abstraction would include properties of each state of the FSA as well as transition probabilities. Therefore, I believe that rule abstraction is more general than the work by Lerman et al. and will be applicable in more domains.

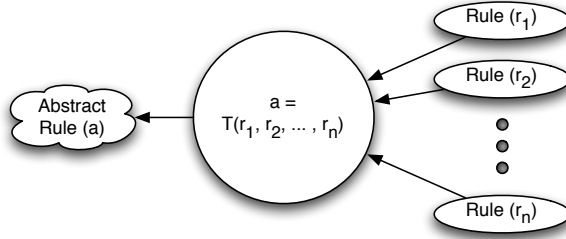


Figure 2: The mapping between low-level parameters and an abstract property can be represented as a mapping function  $T : \mathbf{r} \rightarrow a$ .

### 3 Rule Abstraction

Rule abstraction is the process of finding correlations between low-level rules and abstract properties in a complex system. In this section, this definition is expanded by explaining the framework’s components and the methodology for discovering these correlations.

Abstract properties are used to relate the agents’ low-level behavior to a single abstract parameter. An abstract property is represented as a function (often user-provided) that yields a quantitative value based on a swarm’s emergent behavior. For example, consider the Reynolds boid flock domain [27]; a system where agents flock like birds following three simple rules. The *density* property in this system in  $a$  can be described as the number of agents divided by the area covered by the swarm. The parameters of the low-level rules are values that modify the behavior of the rules. For example, an agent’s *center* rule in the boid flock domain generates a velocity vector towards the center of the swarm ( $c$ ) with a magnitude of  $r_{center} * distance(agent, c)$ . The parameter of the *center* low-level rule,  $r_{center}$ , adjusts the strength of the behavior. If the low-level parameters for rules  $r_1, r_2, \dots, r_k$  are denoted by a parameter vector  $\mathbf{r} = r_1, r_2, \dots, r_k$ , and the parameter of an abstract property of interest is denoted as  $a$ , then the correlation between the low-level and abstract parameters can be represented by a mapping function  $T : \mathbf{r} \rightarrow a$ , as illustrated in Figure 2. Then, to provide swarm-level control, the transfer function is inverted:  $T^{-1} : a \rightarrow \mathbf{r}$ .

For example, suppose a boid flock swarm with an arbitrary density of 5 boids per square inch is desired. The abstract property  $a$  in this situation is density, while the configuration vector  $\mathbf{r}$  specifies the parameters of the low-level rules (number of agents, avoidance, cohesion, and alignment). Rule abstraction generates a mapping  $T^{-1}$  such that the user specifies a desired value for  $a$  and retrieves a possible configuration of  $\mathbf{r}$ :

$$\{25, 1.34, 3.73, 112.3\} = T^{-1}(5).$$

Similarly, if the user wishes to know how a swarm will behave without running a simulation, the transfer function  $T$  may be used:

$$5 = T(\{25, 1.34, 3.73, 112.3\}).$$

In this example, the low-level rules to the abstract property have effectively been “abstracted”.

Learning these mapping functions is the fundamental challenge in generating abstract rules. Although  $T$  and  $T^{-1}$  could be manually specified, as was done by Spears et al. [30] for flocking and Lerman and Galstyan for foraging [13], a more practical approach would be to learn the mapping function by observing the swarm-level behavior in various conditions. The goal is to learn how changing the low-level parameters affects the value of the abstract property ( $T$ ) and vice-versa ( $T^{-1}$ ). To learn these mapping functions, an algorithm is applied to the rule abstraction framework to represent  $T^{-1}$ . This algorithm searches through the configuration space of  $\mathbf{r}$  to find a satisfiable value of an abstract property  $a$ . For example, a hill-climbing approach would see which modifications to  $\mathbf{r}$  can be made to improve the fitness by running simulations, then iteratively changing the current solution to the parameter values that yield the closest value to the desired abstract property value. The algorithms that have been used for rule abstraction are listed later in this document.

A challenging aspect of defining  $\mathbf{r} \leftarrow T^{-1}(a)$  is that the function may produce many-to-many mappings. That is, there may be several possible configuration vectors  $\mathbf{r}_k$  that generate the particular value  $a$  for some abstract property. For example, in the boid flock domain, the density can be increased to a specific amount by lowering the avoidance weight, increasing the cohesion weight, or one of several possible combinations of the two. Usually, the set of solutions  $\mathbf{S} : \{\mathbf{s}_1, \mathbf{s}_2, \dots\}$  can be expressed as a linear combination or as an exhaustive list. This problem can be handled after the algorithm finds  $\mathbf{S}$  by selecting a discrete point  $\mathbf{s}_k \in \mathbf{S}$  to be returned to the user. Several strategies are suggested for selecting this point, such as choosing the most central point to avoid borderline cases, choosing the point closest to the previous state to prevent any radical changes when modifying the behavior of an existing swarm, or simply choosing a random configuration. The best strategy may depend on the application for which rule abstraction is being used.

## 4 Rule Hierarchies

Rule hierarchies are a natural extension of rule abstraction: abstract properties that have been developed for a system could be used as low-level properties in a higher-level rule abstraction. In this case, a rule abstraction hierarchy with three layers would be created. This procedure can be applied iteratively as long as higher levels of abstraction exist in the system. For example, in a particle swarm optimization domain (discussed in more detail later in this document), three layers could be implemented: (1) the lowest layer with the low-level parameters that dictate the behavior of the individual particles, (2) the mid-layer with swarm-level properties such as density, average velocity, and cohesiveness, and (3) the top layer with abstract concepts such as the running time of the algorithm and the goodness of the solution returned. Each correlation between each of these elements, in each of these layers, can be learned with algorithms, such as the ones proposed later in this document. Once the rule abstractions are completed, swarm-level control over the fitness parameters of a particle swarm optimization system would be possible. Also, given the low-level parameters, how well the system will perform could be determined without actually running the swarm system. In addition, more detailed, yet still intuitive, controls are provided by the mid-layer.

Rule hierarchies will be an extremely useful tool in modeling complex systems. Not only do they provide an intuitive model of a system, but they make modeling some systems

possible in the first place. For example, developing artificial general intelligence by simply applying rule abstraction to a collection of accurately modeled neurons with a top-level layer of cognition would not be a tractable problem. A multi-layered approach could make this goal achievable.<sup>1</sup>

Individual rule abstractions in a rule hierarchy can be learned in three different ways: (1) learn each individual rule abstraction as if each were an isolated problem, (2) learn the entire hierarchy simultaneously, or (3) some combination of the two. All of these have advantages and disadvantages. Mainly, if each individual rule abstraction is learned separately, different sets of observations for each may be needed. If learning is performed all at once, each observation will contribute to each rule abstraction. On the other hand, learning several correlations at once may introduce unexpected side effects. Local maxima and different learning rates could affect the end result and return a poorer solution. I plan on exploring all these options and gathering analytical results that compare and contrast the different strategies. Each approach will be compared to one another based on the learning times and accuracy of the model generated.

## 5 Algorithms for learning

Many search and machine learning techniques can be used for rule abstraction to model the transfer functions  $T$  and  $T^{-1}$ . Recall that the direct-mapping function  $T$  can be expressed as a supervised machine learning regression problem: given a configuration vector  $\mathbf{r}$ , determine the expected value of the abstract property  $a$ . The inverse-mapping function  $T^{-1}$  can be expressed as a search problem: given a desired abstract property value  $a$ , find a configuration vector  $\mathbf{r}$  that generates this behavior.

One special consideration that must be taken is the computational complexity of calculating the fitness or evaluation function when performing heuristic search, since the fitness is calculated by running a swarm simulation. For example, consider a genetic algorithm using a population of 1,000 with a swarm simulation that takes six seconds. Approximately 100 minutes would be required to execute a single generation. Search techniques that invoke the fitness function less often will behave faster in this situation, making some approaches faster than others. For example, many hill-climbing approaches (e.g., gradient ascent and simulated annealing) only store one solution at a time and take one path to the goal fitness. In contrast, population-based techniques (e.g., genetic algorithms and particle swarm optimization) call the fitness function several times per iteration and as a result may not run as fast.

Another way to overcome this computational problem is to use machine learning techniques to base predictions on previously observed states (i.e., existing training data). However, the common regression question that machine learning techniques answer is inverted. Instead of asking “what value does this observation result in?”, “what observations will yield this value?” is asked. For example, in a decision tree approach, what sequence of observations that will produce an output is determined, not vice versa. To do this, leaf nodes of the

---

<sup>1</sup>This particular problem of modeling the brain is discussed in more detail in a position statement developed by Miner, Pickett and desJardins [20].

decision tree that match the output desired by the user are found. Then, the set of ancestors of this leaf node is returned as the configuration vector  $\mathbf{r}$ .

Dealing with learning in a changing environment is an open problem with rule abstraction. Currently, all examples used to build models of complex systems are treated with the same weight. I plan to explore how each of the learning algorithms outlined here deal with this problem and determine if they could be extended to handle constantly changing environments to adapt models over time.

In the rest of this section, I enumerate the algorithms that I have used for rule abstraction to show that the framework is flexible enough to handle a wide range of swarm design problems.

## 5.1 Hill climbing

A straightforward way to compute  $T^{-1}(a)$  is with a search algorithm such as hill-climbing. This method iteratively adjusts a current configuration randomly, keeping the new configuration if the produced swarm is closer to the desired state. A series of iterations of this process being applied to the boid density domain is shown in Figure 3. Simulated annealing has also been used, which yields very similar results to stochastic hill-climbing, except that it does not get stuck in local minima, if there are any.

This approach suffers from slow running times if the swarm system takes a significant amount of time to execute. For example, stochastic hill-climbing takes approximately one minute<sup>2</sup> to compute the desired configuration shown in Figure 3 after iterating ten times. This would not be fast enough for real-time applications. For these reasons, other search techniques that make many calls to the fitness function, such as particle swarm optimization and genetic algorithms, do not work well in this situation. Machine learning techniques, which use previously observed configurations, have quicker query times than search techniques.

Hill-climbing approaches typically work well if there are relatively few local maxima. Fortunately, many common swarm domains have abstract behavior values that change monotonically with changes in the parameter settings. For example, increasing the avoidance parameter in a boid flock decreases density.

## 5.2 Simulated annealing

Simulated annealing [10] has been found to require more iterations than stochastic hill climbing at reaching a goal abstract property in the form circle domain and the boid density domain. This is because the benefit of avoiding local maxima that simulated annealing provides is unneeded with these two domains, which have monotonically increasing abstract property values.

I expect that this algorithm may find more use in more complex domains. However, simulated annealing suffers from the problems that other search techniques have: searching is typically too slow to be useful in real-time applications.

---

<sup>2</sup>Running times were measured on a 2.2 GHz Intel Core 2 Duo.

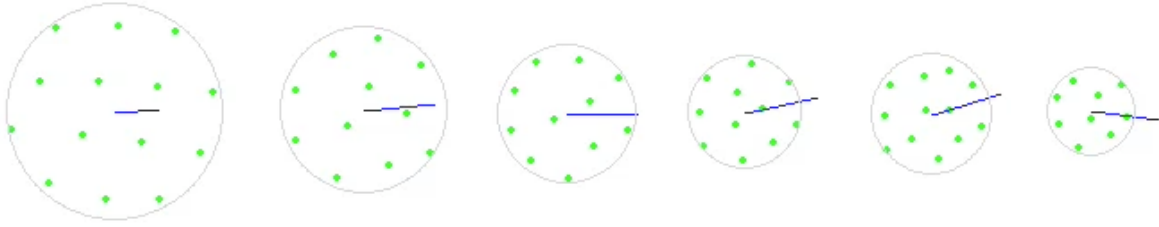


Figure 3: Stochastic hill-climbing progressively searches for a desired density by starting at a low density then moving towards a higher density. In this image, the progression from an initial configuration to a final desirable configuration is shown. The bounding circle is the area covered by the swarm.

### 5.3 Sample/interpolate

A basic technique that has been developed for use in rule abstraction combines sampling and inverted interpolation. Before the rule abstraction application is queried, the program is supplied with a sufficiently detailed sampling of the configuration space. This data set is generated by systematically sampling configurations and recording the resulting abstract property values. Then, interpolation is used to generate a continuous map through these sample points.

Many interpolation techniques can be used for sample/interpolate; however, the resulting models must be invertible. Interpolation, without any modification, provides the mapping  $T(\mathbf{r})$  of the rule abstraction. However, this function is inverted to produce  $T^{-1}(a)$ . This can be done in either of two ways: (1) algebraically solving for  $\mathbf{r}$  in the interpolation equations, or (2) interpolating discrete points to increase the resolution of the data set. The first option is the cleanest and represents the solution space in a functional way. Solutions are often returned as linear combinations, allowing the user to choose the desired point analytically. The second option is less precise than the first, since its accuracy is bound to its resolution.

Two-dimensional triangular linear interpolation with a Delaunay triangulation of the data set has been used for sample/interpolate. This method is useful when data points are added “on the fly,” since the Delaunay triangulation and the interpolation can be updated incrementally. Unfortunately, this approach is only applicable in two-dimensional domains. A more general approach, for any number of dimensions is to sample evenly, then apply linear interpolation (e.g., bilinear interpolation for two dimensions, trilinear interpolation for three dimensions) to the points. This has the disadvantage that points cannot be easily added to the data set; however, it does simplify the interpolation process.

An advantage of this class of techniques is that queries take nominally constant time to return a result. However, the preprocessing step to construct the interpolation can be expensive. In multi-dimensional, complex domains, where the swarm simulation takes several seconds (or longer) to execute, this sampling becomes intractable. Also, these prior built models may become obsolete if the domain changes, since they are static. Any changes that are made to the domain will require the data to be resampled. This is a problem in domains where the conditions may change outside of the scope of the swarm system parameters.

## 5.4 Inverted decisions trees

In a decision tree approach for use with rule abstraction, the sequence of observations that will produce an output are returned, not a classification. To do this, leaf nodes of the decision tree that match the output desired by the user are found, then, the configuration vector  $\mathbf{r}$  as the set of ancestors of this leaf node is generated.

This approach would be useful in domains that have discrete valued parameters. Swarm systems with this property are rare and I do not plan on implementing any domains with this property. However, continuous valued parameters may be discretized to a particular resolution.

This algorithm will be implemented at the very least because is a superb example of inverted machine learning techniques.

## 5.5 Inverted kernel density estimation

Similar to decisions trees, kernel density estimation takes in a configuration vector  $\mathbf{r}$  and returns an expected value for  $a$ . This process is inverted such that kernel density estimation is passed the value for  $a$  and a list of possible values for  $\mathbf{r}$  that could generate this behavior is returned.

Kernel density estimation is very mathematical in nature. That is, the process of interpolating  $a$  is defined as a mathematical equation. Therefore, I am optimistic in being able to “invert” these equations algebraically.

## 5.6 Inverted k-nearest neighbor

The structure of k-nearest neighbor is very similar to kernel density estimation. However, I am skeptical that a straightforward mathematical inversion is possible. I cannot find a elegant mathematical equation that represents the choosing of the  $k$  nearest points.

One approach being considered is to generate a segregated mapping of the space, in a similar vein as Delaunay triangulation. These fragment will have the following property: any two points that lie in a fragment, will share the same  $k$ -nearest neighbors. When the user makes a query, the list of spaces will be searched and fragment with the closest  $k$ -nearest neighbor value match will be returned. This approach will be very dependent on the resolution and distribution of the sample points.

Another approach would be to simply “enhance” the resolution of the space by using k-nearest neighbor. For example, if the user submitted a query, a previously observed configuration with the closest abstract property value could be returned. If the resolution of this space is enhanced by applying k-nearest neighbor to several random or evenly distributed points over the space, these new points could be returned as well. This approach can be used with many different algorithms listed in this document, such as kernel density estimation and sample/interpolate.

## 6 Domains

Rule abstraction and rule hierarchies will be tested in the context of a wide variety of complex systems. One of my goals is to demonstrate that rule abstraction is domain independent and flexible enough to handle different system properties. Also, some domains are included for a particular purpose or outlining a particular feature of rule abstraction. Additional domains may be considered if the feature set of this work cannot be fully tested in the ones enumerated in this section.

### 6.1 Form Circle

In the Form Circle domain, the agents align themselves around a circle’s circumference, forming one unfilled circle. The sum of forces from the three low-level rules and the number of agents contribute to this emergent behavior:

1. *Avoidance* repels agents  $agent_x$  and  $agent_y$  away from one another with force:

$$r_{avoidance} / distance(agent_x, agent_y)^2$$

2. *Circle* moves an agent towards the perceived existing circle. The agent calculates the average distance from its local neighbors to the center ( $distance_{avg}$ ) and conforms to this distance by moving towards the center with force:

$$r_{circle} * (distance(agent_x, center) - distance_{avg})$$

Note that if  $distance(agent_x, center)$  is greater than  $distance_{avg}$ , the agent will move away from the center because the difference produces a negative magnitude.

3. *Center* attracts an agent towards the center of the environment with force:

$$r_{center} * distance(agent_x, center)$$

This rule is kept relatively weak and contributes little to the final force.

4. The number of agents  $N$  is treated as a low-level parameter. Although  $N$  does not directly affect any individual’s behavior, the number of agents does affect the swarm behavior.

This domain is considered to be a “toy domain”: it has no purpose other than being a simple test for rule abstraction techniques. The swarm does not inhibit any chaotic behavior and the configuration space is monotonic and smooth.

### 6.2 Boids

In the boid swarm domain, agents follow the traditional Reynolds boid rules [27] that generate flocking behavior. As in the the original work, this emergent behavior results from the summation of the following forces:

1. *Avoidance* repels agents  $agent_x$  and  $agent_y$  away from one another with force:

$$r_{avoidance}/distance(agent_x, agent_y)^2$$

2. *Center* attracts an agent towards the center of the flock with force:

$$r_{center} * distance(agent_x, center)$$

3. *Align* steers agents towards the average heading ( $\vec{v}_{avg}$ ) of neighboring agents. This is done by applying a velocity vector to the current agent's velocity  $v_x$ :

$$\vec{v}_x = \vec{v}_x + (\vec{v}_{avg} - \vec{v}_x) * r_{align}$$

In our experiments, two abstract properties are focused on: *density* and *chaos*. Density measures how spread the agents are. Chaos represents how much internal motion in the swarm there is, relative to the swarm as a whole. If the average velocity of the agents is greater than the velocity of the swarm as a whole, there is extra motion occurring that is not contributing to the swarm velocity. For example, a flock of geese have a lower chaos value than a bee swarm.

### 6.3 Geometry

The “geometry domain” has been an interesting and useful toy domain for rule abstraction. In this domain, agents self-organize into geometric shapes such as circles (described in the form circle domain section) and polygons in both two and three dimensions.

This domain is particularly interesting because it has two classes of agents when forming polygons: corner agents and edge agents. Agents select with a very simple election algorithm which agents are to be corners. Then, the different types of agents follow a unique set of rules which self-organize into the shapes. Triangles and squares have been formed in many unique ways and polygons of higher degree. Also, tetrahedrons have been created in three-dimensional space by applying very similar rules to form triangles in two-dimensions.

This domain will be used to develop a strategy for applying rule abstraction to discrete valued parameters. Discrete values require a different kind of search and in many situations may be easier since the space is smaller. However, it does introduce its own challenges since interpolation techniques that would return real numbers (e.g., k-nearest neighbor and sample/interpolate) cannot be used here.

Also, there are many different rules available in this domain for forming shapes: ranging from detrimental, to useless, to important. This domain will be used to demonstrate that rule abstraction can decide which rules are important and which are not, given a body of rules.

### 6.4 Wireless Sensor Network Layout

Due to the multi-agent nature of wireless sensor networks (WSNs), rule abstraction could be applied. The goal was to measure three basic abstract characteristics of a WSN (coverage, connectivity, and density) by using four low-level swarm rules (avoid closeness, move

to center, and stay in borders), hopefully revealing a relationship between the two. A network/swarm simulator that enabled us to manipulate the layout of a simple WSN in a 2-dimensional Euclidean grid was developed. The simulator treats the network as a swarm in the background; each sensor represents a swarm agent that moves/acts according to the four rules above. By running thousands of simulations, ranging over different combinations of the rule weights and network configurations, the results showed general relationships between the rule weights and the WSN abstract characteristics do in fact exist. However, the other direction of rule abstraction has not yet been implemented: given a desired abstract level property of the WSN, what configuration should be used to generate this behavior?

This work demonstrates the viability of swarm rule abstraction beyond simpler domains (e.g. shape formation) as a means of providing swarm-level control of a swarm, as opposed to inefficient low-level control via manual rule weight manipulation (you can expand on this obviously)

There is still work to do in the WSN domain. The experiments were limited to a homogeneous network in a featureless 2D environment. New work involving heterogeneous networks, obstacles, 3D environments, and more advanced rules has yet to be investigated. The immediate goal is to make structural improvements to the simulator.

The majority of the work in this domain, including implementation, experiments and documentation were performed by Peter Hamilton and is outlined in more detail in his undergraduate thesis [8].

## 6.5 Particle Swarm Optimization

Particle swarm optimization (PSO) is a swarm intelligence technique for finding a solution to optimization problems in a multi-dimensional, continuous search space [9]. Basically, PSO has a multitude of agents that “swarm” around good solutions, hoping to find better solutions. Agents in PSO are in predetermined “neighborhoods” in which all members of the same neighborhood share the neighborhood’s highest fitness found. Each agent keeps track of its personal highest fitness found, its neighborhood’s highest fitness found and the global highest fitness found. Then, an agent moves towards each of these maxima with a force of predetermined strength. Additional parameters specify the number of agents and how much momentum agents have. Momentum is defined as how much of the agent’s velocity vector in the previous time step is carried into the next time step.

The motivation for applying rule abstraction to this domain is that when particle swarm optimization is used, typically there are no guidelines for determining appropriate parameter settings for the following: the number of neighborhoods, the size of the neighborhoods, the constants in the force equation (personal best factor, neighborhood best factor, global best factor, momentum), the initial spread of the agents, and the initial velocity. Rule abstraction can automatically optimize the performance of a particle swarm optimization algorithm by adding several abstract properties: density, center, average velocity, total velocity, etc. For example, by controlling the density of the swarm, one can control the breadth of the search. With a lower density, more of the search space will be explored with less detail. With a higher density, less of the search space will be explored, but with higher detail. In effect, density could be a crude control for breadth versus depth. More observations into the behavior of a PSO algorithm would bring more abstract properties.

Once all these abstract properties are built with some sampling or learning technique, two abstract properties on top of these mid-level properties are now built. These higher-order abstract properties represent what is being optimized: the goodness of solutions and the speed at which they are determined. Rule abstraction then finds a correlation between the abstract properties listed above and the performance of the algorithm, allowing users of PSO to deliberate on the speed versus accuracy tradeoff through intuitive abstract properties, instead of manipulating low-level parameters directly.

This process of optimizing swarm intelligence behavior can be applied to other population-based optimization algorithms, such as ant colony optimization [7] or genetic algorithms.

## 7 Evaluation Criteria

The effectiveness of rule abstraction must be evaluated at two levels:

- Measure the effectiveness of rule abstraction in learning accurate correlations between low-level parameters
- Measure the effectiveness of rule abstraction in providing an intuitive swarm-level interface to users

Evaluating the effectiveness of rule abstraction in learning accurate correlations can be measured in terms of error rates in predictions. This can be measured with cross-validation in data sets by learning the model of the system with a subset of the data, making predictions about the rest of the data, then measuring the difference as error. This analysis will be applied to all algorithms implemented for this work and will be compared. Since many of our algorithms are commonly used, the purpose of this exercise is not to prove the effectiveness of these algorithms but to compare them amongst themselves in the context of rule abstraction. Analysis of this type has been done for sample/interpolate in the form circle and boid density domain, and is detailed in the Preliminary Results section of this document.

Evaluating the effectiveness of rule abstraction in providing an intuitive swarm-level interface to users may require a user study. The hypothesis that users interacting with abstract rules instead of low-level parameters should be tested to show that it is true. Assuming that it is true, a quantitative analysis of the results will show how much better a swarm-level interface is. A user study may not be necessary if interfaces developed with rule abstraction give instantaneous results, as obviously this will be more efficient than tweaking parameters.

## 8 Preliminary Results

Experiments have been ran for rule abstraction using stochastic hill-climbing and sample/interpolate. Hill-climbing has not been analyzed enough to provide quantitative results (this is a goal for the near future), but some videos of the process have been generated.<sup>3</sup> Sample/interpolate has been extensively analyzed in the form circle and boid density domains.

---

<sup>3</sup>A video showing a boid flock converging to a particular density is available online in the video section of the website: <http://maple.cs.umbc.edu/~don/projects/SAF/videos/>

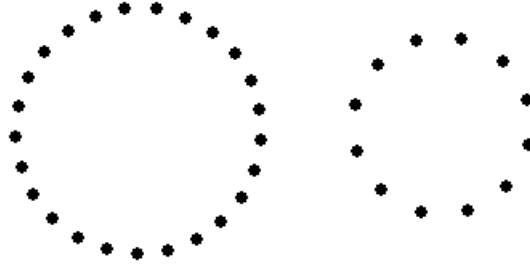


Figure 4: Two circles with different radii. The length of the radius is completely determined by the underlying rule parameters. It is never explicitly specified in the rules.

It has been found to be extremely accurate, but the pre-sampling phase takes a considerable amount of time. More details of the experiments are in the following two subsections.

Eventually, results will be gathered that test rule abstraction using every algorithm in every domain.

## 8.1 Form Circle with Sample/interpolate

The Form Circle domain is used to demonstrate that rule abstraction, using sample/interpolate, can achieve accurate results with negligible query times for both classification and swarm-level control. Form Circle is an intentionally basic because it was one of the first experiments exploring the possibilities of rule abstraction. The domain does not inhibit any chaotic behavior and the configuration space is monotonic and smooth.

For simplicity’s sake, only the parameter of the avoidance rule  $r_{avoidance}$  is varied, which repels agents from one another, and  $N$ , the number of agents. Therefore, there are two low-level rule parameters:  $N$  and  $r_{avoidance}$  and the abstract property modeled is the radius of the circle  $C_r$ . Circles of different radii are shown in Figure 4. This property is interesting because radius is trivial to compute when viewing the swarm in action, but is not immediately obvious from the values of the low-level rules.

Using the sample/interpolate method, 21 values for the number of agents  $N$  were sampled, evenly spaced from 3 to 24 and 49 values for  $r_{avoidance}$ , evenly spaced from 150 to 7350. Therefore, a total of 1029 data points were sampled. At each point, the radius formed by the agents was recorded. The data is very smooth because there is negligible variance in the radius over several runs with the same parameter. Given this data set, a user can begin submitting queries.

Bilinear interpolation was used to determine the radius of a circle given the two variable parameters. The “square” (since the data is uniformly distributed) that the data point fits into is found and the four corner points are used to interpolate a value for radius. This process is illustrated in Figure 5: a selection of sampled data points are shown on the left along with a high-resolution plot of interpolated points on the right.

Using sample/interpolate, the values necessary to create a circle of a desired radius are also determinable. Gradient lines for the target radius are created in each square and pieced together so that the user may select one point as the result of the query. This process is illustrated in Figure 6.

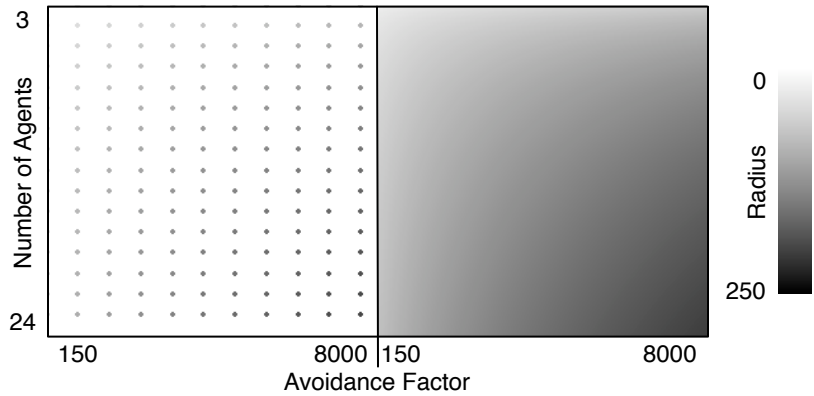


Figure 5: These sample points are used as the corners of boxes to create a piecewise gradient function, which is shown in the right plot. This gradient is used to determine desired radius values, as seen in Figure 6.

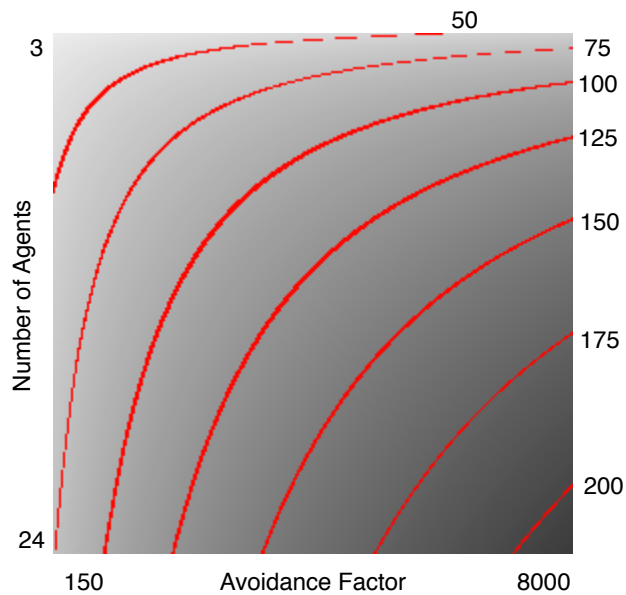


Figure 6: The continuous radius gradient is used to find desired radius values. For example, gradient lines for radii of 50 to 200 are shown as cuts in the image. For example, if a user desired a circle of radius 150 with 20 agents, the transfer function determines an avoidance factor of 3500.

| Sample Size | Average Error (Pixels) |
|-------------|------------------------|
| 100         | 5.636                  |
| 150         | 3.531                  |
| 200         | 2.338                  |
| 250         | 2.012                  |
| 300         | 0.971                  |
| 350         | 0.863                  |
| 400         | 0.427                  |
| 450         | 0.329                  |
| 500         | 0.235                  |

Figure 7: The average error of two thousand random direct mapping queries, varied by the sample data set size in the circle’s radius domain.



Figure 8: Boid flocks with 25 members shown in order of decreasing density (left to right).

This method has demonstrated that it works well by applying the abstract property to two thousand random direct mapping queries and recording the error. The error in the prediction is determined by comparing the predicted value and the actual value after running the swarm with the query’s low-level rule parameters. With radii in this sample ranging from 20 to 210 pixels, and using a sample size of 500, the mapping yielded an average error of .23 pixels over the entire query test set. Also, as expected, decreasing the sample size increases the error of the predictions. These results are shown in Figure 7.

## 8.2 Boid Density with Sample/interpolate

In this experiment, the align factor  $r_{align}$  is maintained at a constant value of .05. Therefore, there are three low-level parameters:  $r_{avoidance}$ ,  $r_{center}$ , and the number of agents  $N$ . The abstract property in that is measured in this experiment is the density  $\rho$  of the swarm. Boid swarms of varying density are shown in Figure 8. The density is measured by dividing the number of agents by the area covered by the swarm:  $\rho = \frac{N}{\pi C_r^2}$ , where  $C_r$  is the radius of a bounding circle. To find the area covered by the swarm, the area of the circle with the center at the average location of the agents and the radius  $C_r$  that would contain all agents is calculated. Therefore, the area covered is greater if the flock is more spread out. The bounding circle can be seen in a screenshot of the simulation software in Figure 9.

This domain is more complex than the circle-radius domain because there are three

low-level parameters instead of two, which results in a three-dimensional search space. Although this difference may appear to make rule abstraction more difficult, the same sample/interpolate method was applied in this domain as the circle-radius domain. Sampling occurs on the three-dimensional space; trilinear interpolation is used instead of bilinear interpolation; and queries are adjusted to handle one more possible constraint.

First, evenly spaced combinations of the low-level parameters ( $r_{avoidance}$ ,  $r_{center}$ , and  $N$ ) are sampled. All 21780 combinations of the following low-level values were sampled: 20 values of  $N$  from 10 to 29, 33 values of  $r_{avoidance}$  from 25 to 825, and 33 values of  $r_{center}$  from .03 to .99. For each of these data points, the average density of a simulated boid flock over time was recorded. To reduce error in the sampling, each data point is the average of five simulations. The sampled points in this data set have an average density of 0.011, ranging from 0.001 to .05 with a standard deviation  $\sigma = .0074$ .

From this data, 19456 adjacent 8-point “cubes” were created. For experimental purposes, the number of cubes can be reduced by removing a portion of the sample if the evenly spaced property is maintained. For each cube, the maximum and minimum densities over all the 8 points are calculated and stored to quickly decide which cubes contain the queried density value.

For this domain, the software responds to all types of queries (both direct mapping and reverse mapping) in the same manner. First, the query is formulated as a hypercube. For example, from the query  $\{\rho = 0.115; r_{center} < .3; N = 25\}$  (“For 25 agents and a  $r_{center}$  factor of less than .3, what combination of parameters will yield a density of 0.115?”) a two-dimensional square representing varying values  $r_{center}$  and  $r_{avoidance}$  is created, because  $N$  is fixed. Then, the relevant cubes as the ones that intersect with this query are identified. Effectively, cubes that cannot interpolate the value  $\rho$  are filtered out. These remaining cubes are used to generate the gradient returned to the user.

Trilinear interpolation is used to interpolate several evenly spaced points inside each relevant cube. Then, the sub-cubes that can interpolate the desired value of density are selected. This process may be repeated several times to increase the resolution of the gradient. When enough iterations of this process have been completed or a certain error bound has been reached, points representing the relevant sub-cubes are returned to the user. A density gradient is shown in Figure 10. Note that a direct mapping query (e.g.,  $\{r_{center} = .3, r_{avoidance} = 180, N = 25\}$ ) is a special case and is handled as a zero-dimensional hypercube. This single point will only intersect with one cube and will be interpolated at the query point to retrieve a single value for  $\rho$ .

50,000 random direct queries to test the accuracy of rule abstraction in this domain were submitted. The random parameter value combinations are contained within the sampled data set’s bounds. The actual density provided by the boid simulation is compared to each predicted value for density and recorded the error. This experiment was repeated several times after reducing the number of cubes by varying amounts. The results of these experiments are shown in Figure 11. From the quantitative results, I conclude that a large number of cubes (around twenty thousand) provides more than sufficient accuracy in direct mapping queries relative to the standard deviation of density values ( $\sigma = .0074$ ). Also, as expected, decreasing the number of sample cubes increases the average error significantly.

By inspecting density gradients some interesting features from the data can be inferred. The most important observation is that the number of agents  $N$  has very little effect on the

```
#2 num_boids=23 match=0.05 avoid=635.759148072
center=0.18 t=145 density=0.0039178 .
```

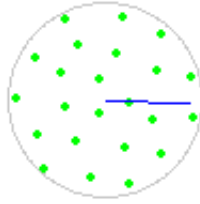


Figure 9: A screenshot of a boid flock simulation. Statistics and parameters are shown at the top of the screen. The boid agents are shown as dots, along with the bounding circle (used to calculate density) and a line indicating the average direction of the flock

### Gradient for Density=.0115

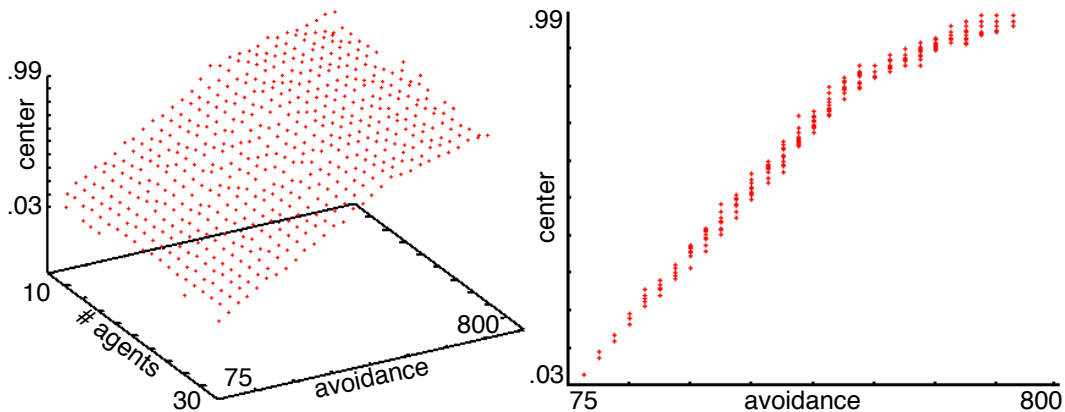


Figure 10: Two different views of the gradient for  $density = .0115$ . On the left, points show samples of the gradient in respect to the three varied parameters:  $center$ ,  $avoidance$  and  $number\ of\ agents$ . From this graph, it is seen that the number of agents has a minimal effect on the swarm density and that the two important parameters are  $avoidance$  and  $center$ . On the right, the number of agents dimension ignored and all the points in respect to  $center$  versus  $avoidance$  are plotted. The data shows a mostly linear correlation, except for an inflection point around  $center = .8$ .

| Sample Cubes | Average Error $e$ |
|--------------|-------------------|
| 19456        | 0.0004995         |
| 2560         | 0.0025855         |
| 320          | 0.0041827         |
| 75           | 0.0058983         |
| 8            | 0.0063675         |

Figure 11: The average error of 50,000 queries to determine density.

density. Intuitively, this is because the size of the bounding circle grows linearly with the number of agents while the low-level rules *avoidance* and *center* enforce a uniform spreading of agents. This fact is illustrated by the right graph in Figure 10.

Also, an inflection point appears in every density gradient observed, which can be seen in Figure 10. To determine what causes this, I examined some of the simulations around this boundary. I discovered that this inflection point represents the threshold at which the *center* rule overcomes the *avoidance* and *match* rules and causes the agents to cyclically move through the center of the swarm. That is, the avoidance rule is not preventing the agents from moving in and out through the center, causing a behavior more similar to swarming bees than flocking boids.

## 9 SwarmVis

In development of the boids domain and the geometry domain, Niels Kasch and I developed an application to visualize complex systems in three dimensions. This application has been particularly useful in analyzing the behavior of swarms.

Visualizing swarm systems effectively is a non-trivial task, partially due to the large number of individual agents involved. The high density and chaotic motions of agents found in typical swarms amplify the complexity of swarm visualization. Effective swarm visualizations are needed to gain insight into local (agent-level) and global (swarm-level) behavior. A well defined set of visualization techniques, in combination with an interactive exploratory tool, provides the facilities to explore these systems at the depth necessary to completely understand them.

A side project to my work in complex systems is *SwarmVis*, a toolkit for visualizing swarms, that goes beyond the simple plotting method to increase the expressiveness of swarm visualizations.<sup>4</sup> The toolkit aims to provide visualization techniques that allow researchers to interactively investigate a swarm’s interactions, fine-grained movements, and behavior. *SwarmVis*’ still images and animations convey agent-level information such as velocity and direction, as well as swarm-level information such as structure, direction and rotation. Figure 12 (right) displays a swarm system with *SwarmVis*’ trails visualization. In this image, direction is expressed by the history of an agent’s past positions. *SwarmVis* provides a collection of visualizations, such as trails, tracks, velocity coloring, group coloring and animation.

---

<sup>4</sup>*SwarmVis* is available for download from the *SwarmVis* project site: <http://code.google.com/p/swarm-vis/>

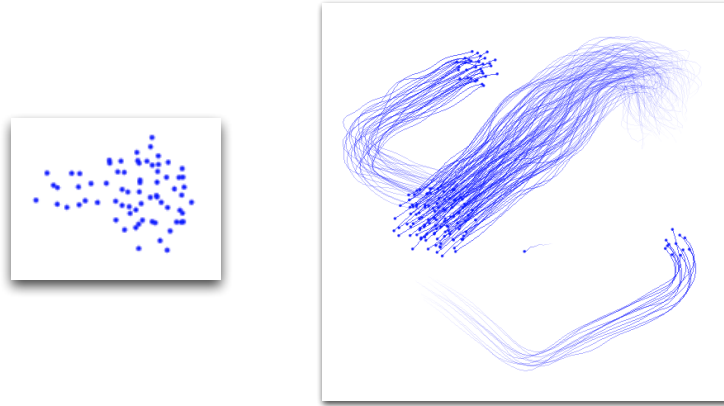


Figure 12: (Left) A basic visualization of a boid flock with agents are plotted as points in space. The image only conveys positional and limited structural information. (Right) A visualization of the boid flock using SwarmVis. Agent position is augmented by a trail of significant length (right). Important properties such as direction, velocity, structure, rotation, and previous positions are all naturally conveyed.

Not much work has been done that specifically addresses the problem of creating effective swarm visualizations. Most swarm visualizations in the literature are the result of swarm intelligence research. Typically, the literature does not discuss the details or effectiveness of such visualizations. Comprehensive swarm system frameworks [15][11] have been created in the past. Their priority is the actual simulation of swarms, not the swarm’s visual representation. Some researchers have implemented visualization techniques for specific swarm domains, such as the particle swarm optimization algorithm [28] and boid flocking [27]. There have been several projects that used a swarm system paradigm to visualize specific information or domains, such as data variations [21], art [3], evolutionary algorithms [32][31], flow [12][17], and source code commits [23]. This work is inspired by the lack of a coherent collection of effective visualization techniques for swarm systems.

More details on SwarmVis are available in my paper written for Dr. Rheingans’ Data Visualization class (Fall 2008) [19].

## 10 Research Plan

To reiterate, these are some of the tasks that have been already accomplished:

1. Implemented the boids domain, the form circle domain and the geometry domain.
2. Implemented rule abstractions for the boid domain and the form circle domain.
3. Implemented the wireless sensor network domain and applied some rule abstractions.
4. Implemented rule abstraction correlation learning algorithms: hill-climbing, simulated annealing, sample/interpolate.

I propose the following tasks to complete the dissertation research, which I expect to be completed within the next year:

1. Formally implement rule abstractions for the geometry domain since, currently, I have only generated informal results. Quantitative results showing that rule abstraction can decide how many corners an object should have will demonstrate rule abstraction's usage on discrete parameters. Also, experimental results that show how important rules are selected and useless or detrimental results are discarded.
2. Implement and analyze a rule hierarchy for particle swarm optimization (with undergraduate Kevin Winner). The results of this experiment will show rule abstraction can be used to optimize population-based search techniques. Also, the differences between a straightforward rule abstraction and a rule hierarchy will be explored. This is expected to be completed by Fall 2009.
3. Implement several rule abstractions in a realistic wireless sensor network domain (with undergraduate Peter Hamilton). This domain shows that rule abstraction can be used in domains that may not be a straightforward swarm system. This is expected to be completed by Fall 2009.
4. Implement, analyze and compare additional learning algorithms k-nearest neighbor, kernel density estimation, decision trees, and perhaps more. These algorithms have been selected because they appear to fit well with the rule abstraction framework. If these algorithms do not provide favorable results, trying some different approaches will be considered. This is expected to be completed by Spring 2009.
5. Gather analytical results of different point in solution set selection techniques ( $\mathbf{s}_k \in \mathbf{S}$ ). I would like to have a decisive strategy for choosing point selection techniques for different kinds of problems and situations.
6. Encapsulate the rule abstraction work into an intuitive software package. The software package should be usable by researchers interested in swarm systems. The algorithms implemented will be included, and users will be able to add their own. Additionally, the domains implemented will be provided as examples to users. This is the final result of this work and is expected to be completed by early 2010.

## References

- [1] BABAOGU, O., MELING, H., AND MONTRESOR, A. Anthill: A framework for the development of agent-based peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems* (2002).
- [2] BECKERS, R., DENEUBOURG, J., AND GOSS, S. Trails and U-turns in the selection of a path by the ant *Lasius niger*. *Journal of theoretical biology* 159, 4 (1992), 397–415.
- [3] BOYD, J., HUSHLAK, G., AND JACOB, C. SwarmArt: Interactive art from swarm intelligence. In *Proceedings of the 12th Annual ACM International Conference on Multimedia* (2004), pp. 628–635.

- [4] BUHL, J., SUMPTER, D., COUZIN, I., HALE, J., DESPLAND, E., MILLER, E., AND SIMPSON, S. From Disorder to Order in Marching Locusts, 2006.
- [5] COLLIER, N. Repast: An extensible framework for agent simulation. *The University of Chicago's Social Science Research* (2003).
- [6] COUZIN, I., AND FRANKS, N. Self-organized lane formation and optimized traffic flow in army ants. In *Proc. R. Soc. Lond. B* (2003), vol. 270, pp. 139–146.
- [7] DORIGO, M. *Ant Colony Optimization*. MIT Press, 2004.
- [8] HAMILTON, P. Applying swarm rule abstraction to a wireless sensor network domain. *UMBC Undergraduate Honors Thesis* (2008).
- [9] KENNEDY, J., AND EBERHART, R. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks* (1995).
- [10] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Optimization by Simulated Annealing. *Science, Number 4598 220, 4598* (May 1983), 671–680.
- [11] KLEIN, J. Breve: a 3D environment for the simulation of decentralized systems and artificial life. In *Proceedings of the Eighth International Conference on Artificial Life* (2003), MIT Press, pp. 329–334.
- [12] KRUGER, J., KIPFER, P., KONDRATIEVA, P., AND WESTERMANN, R. A Particle System for Interactive Visualization of 3D Flows. *IEEE Transactions on Visualization and Computer Graphics* 11, 6 (2005), 744–756.
- [13] LERMAN, K., AND GALSTYAN, A. Mathematical model of foraging in a group of robots: effect of interference. *Autonomous Robots* 13, 2 (2002), 127–141.
- [14] LERMAN, K., MARTINOLI, A., AND GALSTYAN, A. A review of probabilistic macroscopic models for swarm robotic systems. In *Swarm Robotics Workshop: State-of-the-art Survey* (2005), Springer, pp. 143–152.
- [15] LUKE, S., CIOFFI-REVILLA, C., PANAIT, L., SULLIVAN, K., AND BALAN, G. Mason: A multiagent simulation environment. *Simulation* 81, 7 (2005), 517–527.
- [16] MCLURKIN, J. *Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots*. PhD thesis, Massachusetts Institute of Technology, 2004.
- [17] MERZKIRCH, W. *Flow Visualization Second Edition*. Academic Press Inc. (London) Ltd., 1987.
- [18] MINAR, N. *The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations*. Santa Fe Institute, 1996.
- [19] MINER, D., AND KASCH, N. SwarmVis: a tool for visualizing swarm systems. *UMBC Computer Science 636: Data Visualization* (2008).

- [20] MINER, D., PICKETT, M., AND DESJARDINS, M. Understanding the brain's emergent properties. In *Proceedings of the Second Conference on Artificial General Intelligence* (2009).
- [21] MOERE, A. Time-varying data visualization using information flocking boids. In *Proceedings of the 2004 IEEE Symposium on Information Visualization* (2004), pp. 97–104.
- [22] MONDADA, F., PETTINARO, G., GUIGNARD, A., KWEE, I., FLOREANO, D., DENEUBOURG, J., NOLFI, S., GAMBARDILLA, L., AND DORIGO, M. Swarm-Bot: A new distributed robotic concept. *Autonomous Robots* 17, 2 (2004), 193–221.
- [23] OGAWA, M. Code swarm, Website last accessed September 2008. <http://vis.cs.ucdavis.edu/ogawa/codeswarm/>.
- [24] PARRISH, J., VISCIDO, S., AND GRUNBAUM, D. Self-organized fish schools: an examination of emergent properties. *Biological Bulletin, Marine Biological Laboratory, Woods Hole* 202, 3 (2002), 296–305.
- [25] REYNOLDS, C. Steering behaviors for autonomous characters. In *Game Developers Conference* (1999), vol. 1999.
- [26] REYNOLDS, C. Interaction with groups of autonomous characters. In *Game Developers Conference* (2000), pp. 449–460.
- [27] REYNOLDS, C. W. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings)* (1987), pp. 25–34.
- [28] SECREST, B., AND LAMONT, G. Visualizing particle swarm optimization - gaussian particle swarm optimization. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium* (April 2003), pp. 198–204.
- [29] SIKORA, R., AND SHAW, M. A multi-agent framework for the coordination and integration of information systems. *Management Science* 44, 11 (1998), 65–78.
- [30] SPEARS, W., SPEARS, D., HAMANN, J., AND HEIL, R. Distributed, physics-based control of swarms of vehicles. *Autonomous Robots* 17, 2 (2004), 137–162.
- [31] SPECTOR, L., AND KLEIN, J. Evolutionary dynamics discovered via visualization in the breve simulation environment. In *Workshop Proceedings of Artificial Life VIII* (2002), pp. 163–170.
- [32] SPECTOR, L., KLEIN, J., PERRY, C., AND FEINSTEIN, M. Emergence of Collective Behavior in Evolving Populations of Flying Agents. *Genetic Programming and Evolvable Machines* 6, 1 (2005), 111–125.