

Predicting and Controlling System-Level Parameters of Multi-Agent Systems

Don Miner and Marie desJardins

University of Maryland, Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250

Abstract

Boid flocking is a system in which several individual agents follow three simple rules to generate swarm-level flocking behavior. To control this system, the user must adjust the agent program parameters, which indirectly modifies the flocking behavior. This is unintuitive because the properties of the flocking behavior are non-explicit in the agent program. In this paper, we discuss a domain-independent approach for detecting and controlling two emergent properties of boids: density and a qualitative threshold effect of swarming vs. flocking. Also, we discuss the possibility of applying this approach to detecting and controlling traffic jams in traffic simulations.

Introduction

A multi-agent system is comprised of several agents and exhibits some emergent system-level behavior. The behavior of an individual agent is typically well understood because its program has adjustable control parameters that directly modify its behaviors. What is typically not understood, however, is how the values of these agent-level control parameters affect the system-level behavior. Shaping the emergent behavior of a multi-agent system is typically reduced to a repetitive and time-consuming process of guess-and-check: select new parameter values and observe the result. The controller of the system is forced to use the agent-level parameters which only indirectly affect the system-level behavior. This process is unintuitive. Our goal is to develop a domain-independent approach for modeling how changes in agent-level control parameters affect the system as a whole, and using this model to enable a user to predict and control the behavior of the system *at the system level*. Our contribution is a learning framework for developing this understanding.

Our approach is to define system-level parameters that describe system-level properties of the system. These parameters can describe any measurable aspect of the observable state of the system, such as density, average velocity and cohesion. We claim that many system-level property values of interest can be defined in terms of agent-level parameter values. The problem of constructing a model of a system-level property, in terms of the agent-level parameter values,

Copyright © 2009, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.



Figure 1: A swarming boid system (left) and a flocking boid system (right).

is what we name the *forward-mapping problem*: that is, develop a functional (many-to-one) mapping F that maps the agent-level parameters $\mathbf{x} \in \mathbb{R}^n$ onto a system-level property $p \in \mathbb{R}$:

$$F : \mathbf{x} \rightarrow p.$$

This mapping can be used to predict what system-level behavior will result from a run of the system, given the agents' configuration vector. For example, this mapping can be used to predict whether a system will exhibit a threshold effect or what its spatial density will be. Learning this mapping is a relatively straightforward regression problem, although adequately sampling high-dimensional parameter spaces may be difficult. We explain how to develop the forward mapping later in this paper.

The other aspect of our approach is the *reverse-mapping problem*: mapping some system-level parameter $p \in \mathbb{R}$ onto a configuration vector $\mathbf{x} \in \mathbb{R}^n$:

$$F^{-1} : p \rightarrow \mathbf{x}.$$

Since F is many-to-one, this mapping is one-to-many. The reverse mapping can be used to control the behavior in terms of system-level parameters. Given some desired system-level property value, the reverse mapping returns a configuration vector that would cause this property to be observed. For example, F can be used to generate a system that exhibits a particular threshold effect or exhibits a particular density. The reverse-mapping problem can be solved either by optimization or by "inverted" regression. We will discuss both of these approaches in detail later in this paper.

In this paper, we present the results of applying our approach to the Reynolds boid flock system (Reynolds 1987), a system of agents that follow three simple rules that dictate movement. We define a forward mapping that predicts

the density of the flock as well as the qualitative behavior (whether it will “flock” like birds or “swarm” like bees). Flocking boids and swarming boids are shown in Figure 1. The difference between flocking and swarming is very visible, yet not directly obvious from looking at the configuration vector. Also, the transition between flocking and swarming is sudden: as the agent-level parameter values are changed, at some point a threshold is reached and a boid flock collapses into a swarm.

We also describe an automobile traffic simulation that we are currently working on because it exhibits traffic jams, an interesting threshold effect. We have also applied our approach to other domains, such as particle swarm optimization and wireless sensor network layout, but have not studied threshold effects in these domains.

Related Work

There has been work in developing “system-level control” for specific domains that require human intuition to develop the control models. Our approach is significantly different from these approaches because it provides a domain-independent framework that can provide models of system-level properties automatically. A human is only needed to set up the necessary components; the rest of the modeling process is handled by the learning algorithms.

For example, in work by Spears et al. (Spears et al. 2004), the authors develop a physics-based control policy for a swarm of mobile robot agents. With this policy, the system-level behavior can be described as a mathematical equation. This mathematical model of the system is useful and precise, but will not extend to other domains. Also, some level of human intuition was required to develop these equations. Our approach 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.

Lerman et al. discuss “macroscopic models” of swarm robot systems (Lerman and Galstyan 2002)(Lerman, Martinoli, and Galstyan 2005) and is similar to our approach in motivation. The authors model agents as finite state automata (FSAs) 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. Our approach differs from this work 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. Second, the way in which the authors model agents is more restrictive than the models used in our approach. Some 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, our approach’s view of an agent can represent an FSA: the agent-level parameters would be the properties of each state of the FSA as well as the transition probabilities. Therefore, we believe that our approach is more general than the work by Lerman et al. and is applicable in more domains.

Defining System-Level Parameters

Our approach consists of learning two mappings: the forward mapping and the reverse mapping. Given a data sample, learning the forward mapping is a straightforward regression problem. An initial data set can be built by systematic sampling different configurations and may be augmented with new data points in real time as the system is running. We have applied k-nearest neighbor, support vector regression (Smola and Schölkopf 2004), linear regression and nonlinear regression to this problem. For example, in the boid flocking domain, thousands of different configurations were sampled. For each execution, the density and the flocking behavior were recorded. With this data set, we used regression to learn the correlation between agent-level configuration parameters and the system-level properties (density and flocking behavior).

On the other hand, learning the reverse mapping is an “inverted” regression problem. Inverted regression can be seen as an optimization problem: adjust the agent-level parameter values to minimize the difference between the desired system-level property value and the actual system-level property value. Standard optimization techniques such as stochastic hill climbing or simulated annealing (Kirkpatrick 1984) can be used for this purpose. However, optimization algorithms require several iterations of guess-and-check. By inverting the mappings learned for the forward mapping, we achieve constant-time lookup to determine what configuration should be used to generate a desirable behavior.

Inverting regression can be done in many ways. In our work, we typically use one of two approaches. The first is to use the forward mapping to generate additional points in the data set. This effectively increases the granularity of the discrete data set. When a digital picture is scaled up, interpolation (i.e., bilinear interpolation) is used to fill in missing pixels to make the image appear larger. In effect we are: “scaling up” the data set. Any method used for learning the forward mapping can be used to enhance the data set in this way. With this enhanced data set, we can select points that are very close to the desired configuration.

Another way to invert the learned mapping is to solve an equation that represents the forward mapping. For example, suppose that we have found a linear mapping $F(\mathbf{x}) \rightarrow p$ that describes the relationship between the configuration vector \mathbf{x} and system-level property value p . Determining a configuration that results in a value of $p = 5$ is equivalent to solving the equation $F(\mathbf{x}) = 5$. This solution can be found in a number of ways, such as root finding.

Boid Flocking Domain

We use the boid flocking domain (Reynolds 1987) to demonstrate the effectiveness of our approach. Our motivation for using boids is to demonstrate a proof of concept of our approach in a relatively simple domain. In the boids domain, agents follow three simple rules, which generate force vectors that are summed to guide the agents’ directions. When all agents follow these rules, flocking behavior emerges. These rules are:

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 :

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

The three parameters that adjust the magnitude of these vectors ($r_{avoidance}, r_{center}, r_{align}$) and the number of agents N , are the agent-level control parameters of this system. Thus, the configuration vector is defined as

$$\mathbf{x} = [r_{avoidance}, r_{center}, r_{align}, N].$$

We consider two system-level properties: density $\rho \in \mathbb{R}$ and flocking behavior $b \in \{flocking, swarming\}$. Density measures how close together agents are and is calculated by dividing the area covered by the entire flock, divided by the number of member agents. To detect whether the system is flocking or swarming, the ‘‘internal velocity’’ is calculated as:

$$v_{internal} = \sum_{i=0}^N (|v_i| - |v_{center}|).$$

This internal velocity is the average velocity of the agents, relative to the velocity of the center of the flock. This number can be interpreted as the total agent velocity that does not contribute to the flock’s average velocity. This value is very close to zero when the boids are flocking, and becomes significantly larger when it is swarming. To assign a *flocking* or *swarming* value, a threshold k is chosen such that $b = flocking$ if $v_{internal} < k$ and $b = swarming$ if $v_{internal} > k$.

Given the configuration vector, the values for density and flocking can be determined by solving the forward-mapping problem. A data set was generated by running thousands of instances of differently configured boid flocks and recording the observed density and flocking behavior. Density is a real value, while flocking is a binary value. Therefore, to map density, we are restricted to regression approaches, such as k-nearest neighbor, linear regression and nonlinear regression. Meanwhile, since flocking is a binary value, classification techniques can be used to predict, such as support vector machines or logistic regression.

Our results show that density behaves as expected: density increases as r_{center} is increased and $r_{avoidance}$ is decreased. The number of agents and r_{align} has little effect on the density. K-nearest neighbor has been the most accurate of the approaches, achieving relative error of .5%. A graph showing the density of our boid flocks given two agent-level parameters is shown in Figure 2.

We found that very low values for $r_{avoidance}$, very high values for r_{center} , and very low values for r_{align} cause the swarming behavior. Intuitively, the desire to go towards the

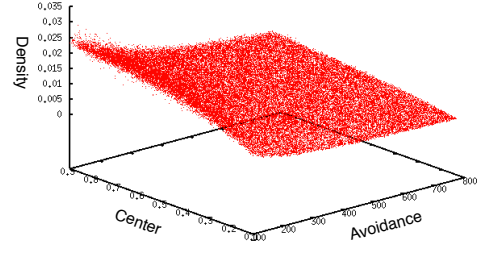


Figure 2: This graph shows the density of the boid flock given r_{center} and $r_{avoidance}$. Density is highest when r_{center} is high and $r_{avoidance}$ is low.

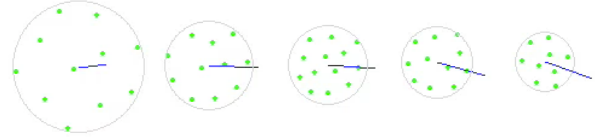


Figure 3: A progression from a lower density to a higher density (left to right) generated by hill climbing. In practice, the initial configuration would be much closer to the target density than in this example.

center of the flock overcomes the alignment and avoidance desires, causing agents to repeatedly move through and past the center of the flock then be pulled back towards the center. With a higher avoidance and alignment, they are forced into a flocking behavior. We hypothesize that the reason this behavior appears to be a threshold effect (i.e., the transition from flocking to swarming is quite sudden), is the flock breaks the equilibrium of outward forces and inward forces. Once the agents start moving through the center all at once, the avoidance forces dominate and the flock becomes very chaotic.

To develop the density reverse mapping (i.e., the mapping that returns a configuration vector, given the density), we have experimented with two approaches: stochastic hill climbing (optimization) and nearest neighbor (i.e., returning training set instances that have densities that are similar to the desired density). Both approaches return results that are as accurate as the forward mappings. A sample progression from a lower density to a higher density is shown in Figure 3.

Automobile Traffic Simulation

We are currently applying our approach to detect an interesting threshold effect in automobile traffic simulations: the appearance of traffic jams. Some traffic situations will never exhibit a traffic jam. As more cars are added to the system, the entire system stops in a sudden gridlock situation. We wish to apply our approach to detect whether or not a traffic configuration, consisting of a specified number of cars, number of trucks, imposed speed limit, and other param-

ter values, will result in a free-flowing traffic pattern. To predict the behavior of a traffic pattern, we build a forward mapping, correlating the agent-level parameter values to the binary system-level property of whether or not the system exhibits traffic jams or not. The construction of this mapping is similar to detecting whether boids swarm or flock.

Knowing whether or not a system will have traffic jams provides only limited information. It would also be useful to predict the severity of the traffic jams and how frequently they will occur. We believe that our approach can learn the mappings from the agent-level parameters to these system-level properties. The challenge in this problem is that severity and frequency of traffic jams are scalar factors of a functional model of the traffic system behavior. That is, it will be necessary to learn the correlations between agent-level parameter values to parameters of a system-level measurement over time. For detecting traffic jams in a traffic simulation, the most suitable model for the behavior over time may be a repeating pattern, such as a sine wave. We believe it is possible to not only infer singular scalar values, as was done in the boids domain, but also to infer graphs of behavior over time.

Our approach should also be able to generate traffic situations with specified desired system-level features by using a reverse mapping. For example, the model could be used to generate a traffic simulation that exhibits a traffic jam of moderate severity every 7 seconds.

By November 2009 (the time of the symposium), we expect to have positive results showing that our approach is applicable in a traffic simulation domain to detect the threshold effect of traffic jams.

Conclusion

We have presented a domain-independent approach for predicting and controlling the behavior of a multi-agent system. This approach can be used to detect threshold effects in several domains, such as the boid flocking domain and traffic simulation. However, learning the forward and reverse mappings depend on the agent-level parameters are highly correlated with the emergent behavior.

Generating instances of simulations with a desired emergent behavior is a powerful tool. For the boids domain and the traffic domain, manually adjusting agent-level control parameters to achieve a particular result is a time-consuming task. Our approach lessens the time required by the use to achieve necessary results.

References

- Kirkpatrick, S. 1984. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics* 34(5):975–986.
- Lerman, K., and Galstyan, A. 2002. Mathematical model of foraging in a group of robots: effect of interference. *Autonomous Robots* 13(2):127–141.
- 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.

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

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

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