

Learning Non-Explicit Control Parameters of Self-Organizing Systems

Don Miner and Marie desJardins
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, Maryland, U.S.A.
don1@umbc.edu, mariedj@cs.umbc.edu

Abstract—Controlling self-organizing systems with given control parameters is often unintuitive and inefficient for the user. Typically, there is some emergent behavior that the user would like to produce that is not directly controllable. Our goal is to develop an autonomous and domain-independent learning framework for adding non-explicit control parameters to self-organizing systems that provide users with more intuitive real-time control of the system. These additional controls are created by using regression to learn a mapping between the explicit control parameters and the non-explicit control parameters.

I. INTRODUCTION

When interacting with self-organizing systems, one often 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 two levels relate to one another. The contribution introduced in this paper is a framework for learning correlations between agent-level parameters and group-level properties.

Rarely is there an obvious, explicit and mathematical understanding of how the low-level parameters of an agent affect the emergent properties of the self-organizing system as a whole. We refer to the low-level parameters of the agent program *Explicit Control Parameters* (ECPs), because the values are explicitly defined in the program. Using ECPs to control a system is typically an unintuitive way to create specific system-level behaviors. In contrast, we refer to the measurable emergent properties of the system *Non-Explicit Control Parameters* (NECPs), because the values are emergent and are not explicit in any part of the program. NECPs are more intuitive from a user’s perspective, since they describe what the user observes.

In this paper, we use the Reynolds boid flocking domain [1] as an example to demonstrate our framework. In the boids domain, agents follow three simple rules, which generate force vectors that are summed to guide the agents’ directions. At the swarm-level, a “flocking” behavior emerges. The parameters that adjust the strengths of the force vectors are the ECPs of this system. The particular NECP discussed here is the flock density (agents per unit area covered), but other NECPs for this domain are possible, such as coverage, swarm-level velocity, and smoothness of flow. Boid flocks of varying densities are shown in Figure 1.

Our framework uses supervised machine learning techniques (specifically, regression) to learn mappings of ECPs

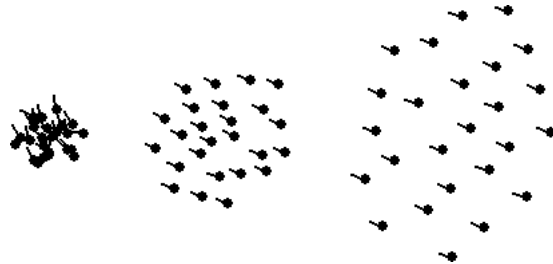


Fig. 1. Boid flocks with 25 members, shown in order of decreasing density (left to right). Each flock is running the same program, but with different control parameter values.

to NECPs and vice versa. Given the ECP values of a self-organizing system’s program, the values for the NECPs can be predicted with the ECP-to-NECP mapping. This is useful for determining what a system will do without having to actually run the system. Learning this mapping is a regression problem, where the ECPs are the independent variables and the NECPs are the dependent variables. Conversely, given some desired NECP values, the NECP-to-ECP mapping determines what ECP values should be used. This “reverse mapping” provides control of the system at a more intuitive level with a user-friendly interface, since users do not have to control the system in terms of the ECPs.

II. RELATED WORK

There has been work in developing system-level control for specific domains, in which researchers develop mathematical formulas [2] or describe systems at an abstract level [3][4]. All of these previous efforts required human intuition to develop the models. Our framework is significantly different from these approaches because it provides a domain-independent learning framework that can automatically produce models of NECPs. Human input is needed only to define the NECPs—the rest is handled by the learning algorithms.

For example, in work by Spears et al. [2], the authors develop a boids-like 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 analysis 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

always result in models that are as accurate as the theoretical behaviors developed by Spears et al. since some error may be introduced in the learning process. However, we do so autonomously with methods that can be extended to other domains.

III. APPROACH

The learning framework involves learning two mappings, the ECP-to-NECP *forward mapping* and the NECP-to-ECP *reverse mapping*. The forward mapping defines a function f that approximately maps the vector of ECP values (denoted by \mathbf{E}) onto the vector of NECP values (denoted by \mathbf{N}):

$$\mathbf{N} \leftarrow f(\mathbf{E}).$$

The reverse mapping, on the other hand, is an optimization task, in which we must find values for ECPs that will cause the system to exhibit specified values of NECPs:

$$\text{minimize} |f(\mathbf{E}) - \mathbf{N}| \quad (1)$$

However, since this mapping is intended to be used for controlling systems in real time, we must create a faster representation of the reverse mapping than directly optimizing or searching over the search space at control time. To achieve constant-time lookup for the reverse mapping, we *invert* the forward mapping to define f^{-1} :

$$\mathbf{E} \leftarrow f^{-1}(\mathbf{N}) \quad (2)$$

For example, consider three ECPs of the boids domain: the number of agents, N ; the strength of avoidance, F_{avoid} (avoid others); and the strength of cohesion, $F_{cohesion}$ (tend towards center of the flock). Using these three parameters, we can define the mappings to and from the NECP of density ρ :

$$\rho \leftarrow f(N, F_{avoid}, F_{cohesion}) \quad (3)$$

$$(N, F_{avoid}, F_{cohesion}) \leftarrow f^{-1}(\rho) \quad (4)$$

Learning the forward mapping is a straightforward regression problem; any regression techniques could theoretically be used into this framework. On the other hand, learning the reverse mapping is an “inverted” regression problem. We cannot use straightforward regression because the mapping from NECPs to ECPs is one-to-many. For example, if we use k-nearest neighbor and specify a value for the NECP, the possible k nearest points will be scattered across the independent variable search space. When these values are averaged, the configuration vector will probably not result into behavior that produces the specified NECP value. This is because a wide range of ECP configurations satisfy the NECP configuration. Therefore, we must invert the regression problem: the model must determine the set of ECP parameter value vectors that would satisfy a specified NECP value. We assume that the mapping from ECPs to NECPs is many-to-one and the mapping from NECPs to ECPs is one-to-many. Thus, inverted regression returns a set of solutions, of which one must be selected to be applied to the system. We

have identified two general approaches for inverting regression models: enhanced model resolution and root finding.

Enhanced model resolution uses the learned regression model to supplement the training set with additional inferred observations. Then, when queried for a specific NECP configuration, the points in this augmented training set that are within a certain distance from the query point are returned. For example, suppose a user desires a boid flock of density $\rho = 5$. Our system will search through an enhanced data set and return all ECP configurations that would yield a predicted ρ . This approach is fast, but building the enhanced data set may be intractable in high-dimensional configuration spaces. This problem can be alleviated by enhancing only necessary portions of the data set, on a per query basis. Another challenge when using this approach is that adding new training instances may change the regression model and therefore may require recalculating the inferred points. This problem is especially severe when the training set is initially small or when we wish to use an on-line learning approach. In the boid density domain, we have achieved very precise results with this approach using k-nearest neighbor.

Root finding is usable with mathematical regressions, such as linear regression by using Newton’s method, gradient descent, or an other optimization algorithm to find a desirable ECP configuration. Queries using this approach are slower than with enhanced model resolution, but are more tractable in high-dimensional configuration spaces and in on-line learning, when new training points are added as the system runs.

IV. DOMAINS

Our framework is domain-independent and applicable in many different situations. Domains that we are using to test our framework include boid flocks, particle swarm optimization (PSO) [5], wireless sensor network (WSN) layout using boid-like rules, and traffic simulations.

For each of these domains, we are working to implement a system-level control interface. Users are able to control the system in terms of several intuitive NECPs:

- Boid flocks – density, flock velocity, internal chaos
- PSO – breadth vs. depth, clustering, convergence
- WSN layout – coverage, redundancy, connectivity
- Traffic simulations – traffic flow, variance in car velocity, frequency of traffic jams

REFERENCES

- [1] C. W. Reynolds, “Flocks, herds, and schools: A distributed behavioral model,” in *Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings)*, 1987, pp. 25–34.
- [2] W. Spears, D. Spears, J. Hamann, and R. Heil, “Distributed, physics-based control of swarms of vehicles,” *Autonomous Robots*, vol. 17, no. 2, pp. 137–162, 2004.
- [3] I. Couzin and N. Franks, “Self-organized lane formation and optimized traffic flow in army ants,” in *Proc. R. Soc. Lond. B*, vol. 270, 2003, pp. 139–146.
- [4] J. McLurkin, “Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots,” Ph.D. dissertation, Massachusetts Institute of Technology, 2004.
- [5] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, 1995.