Avoiding Local Optima with Interactive Evolutionary Robotics

Josh C. Bongard
Dept. of Computer Science
University of Vermont
josh.bongard@uvm.edu

Paul Beliveau
Dept. of Computer Science
University of Vermont
paul.beliveau@uvm.edu

Gregory S. Hornby
Univ. of California Santa Cruz
NASA Ames Research Center
gregory.s.hornby@nasa.gov

ABSTRACT

The main bottleneck in evolutionary robotics has traditionally been the time required to evolve robot controllers. However with the continued acceleration in computational resources, the main bottleneck is now the time required for an investigator to create a robot simulator, a neural network, evolutionary algorithm and fitness function that together produce the desired behavior. Here we introduce a software framework that allows a user to conduct evolutionary robotics experiments without having to write any software themselves: the user defines the robot morphology, task environment and fitness function interactively; a neural network is constructed based on the robot's morphology; and an evolutionary algorithm optimizes desired behavior. We here show that this approach allows users to overcome one of the main limitations of evolutionary algorithms—recognizing and then preventing entrapment in local optima—in a continuous, code free manner.

Categories and Subject Descriptors

I.2.9 [**Computing Methodologies**]: Artificial Intelligence—*Robotics*

General Terms

Experimentation, Algorithms

Keywords

Evolutionary Robotics, Interactive Evolutionary Algorithms, Evolutionary Algorithms

1. INTRODUCTION

Typically in evolutionary robotics, the investigator takes considerable time to program a simulation, controller and evolutionary algorithm. After evolution commences she then alternates between short bursts of optimization and re- engineering of the fitness function. One approach to reduce the number of these design cycles is to use interactive evolution (e.g. [2] and [1]) in which the user rather than the computer determines which solutions breed and which are culled. Here we present a novel method of combining user input and evolutionary algorithms that does not require continuously re-programming a fitness function nor does it require the user to continuously supply preferences.

2. METHODS

Typically, an investigator directs an evolutionary algorithm to select for different robot behaviors by modifying a fitness function. For instance a fitness fitness that selects for locomotion in a legged robot may reward for displacement over a fixed time period. Changing the fitness function to reward for maximal vertical distance between the robot's feet and the ground will select for jumping. In the approach described here, the fitness remains fixed, but the user can direct evolution toward different behaviors by altering the task environment of the robot.

This is accomplished by allowing the user to interact directly with the physics-based simulator in which the robots are evolved. The user is presented with a simulated robot and a light-emitting object (shown as a red cube in Fig. 1)[1]. The user drags a copy of the robot to another position within the simulator, which indicates where the robot should move to by the end of the evaluation period. Each component in each copy of the robot contains a photosensor (the gray spheres embedded in the robots in Fig. 1) that registers light intensity.

The fitness function for all experiments reported here can then be defined as

$$f = 1 - \frac{\sum_{i=1}^{n}\sum_{j=1}^{t}|p_{ij}^{(s)} - p_{ij}^{(e)}|}{nt} \qquad (1)$$

where the robot is evaluated in the simulator for t time steps; the robot is constructed from n components; $p_{ij}^{(s)}$ is the value of the ith photosensor at time j for the robot at the start position; and $p_{ij}^{(e)}$ is the value of the ith photosensor at time j for the robot at the end position.

We define the photosensor values to range in $[0, 1]$ such that a sensor value of zero indicates the sensor is at or beyond some maximal distance from the light-emitting object, and a value of one indicates that the sensor is coincident with the object. This then constrains the fitness value f to also range in $[0, 1]$. A fitness value of zero means that the robot has remained at the start position or moved away from the end position. A fitness value of 1 indicates that the robot has moved instantaneously to the end position and remained there throughout the evaluation period. Higher fitness values indicate controllers that have moved the robot closer to

[1]Several videos that accompany this paper can be found at `bit.ly/IyN8qr`.

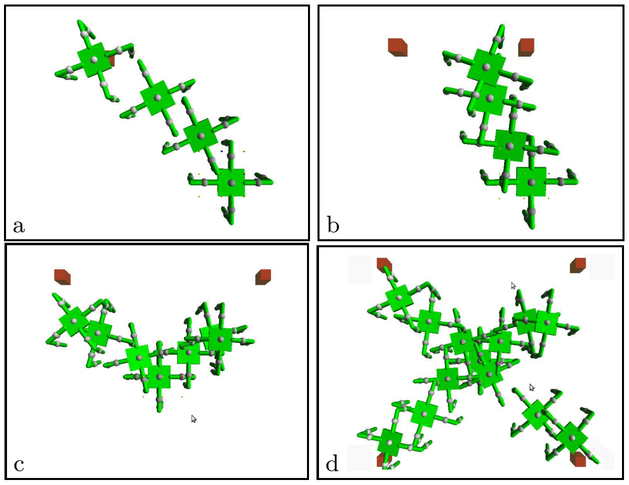

Figure 1: Interactive application of robot shaping.

the light-emitting object—or moved the robot more rapidly to the same position—compared to controllers with lower fitness values.

Using this framework the user can interactively create different environments that select for different behaviors: placing the target object at the top of a flight of stairs selects for climbing; suspending the robot and the target object above the ground and creating rungs between the two will select for brachiation (see the accompanying videos).

Users may also elect to incorporate robot shaping [3] while evolving controllers: controllers may initially be evolved in a single environment and, after success, evolved in multiple environments. This is shown in the locomotion example reported in Fig. 1. Because the robot and the light-emitting object are on the ground (the end position of the robot is not shown for clarity) and there are no intervening obstacles, this task environment selects for locomotion toward the object (Fig. 1a). Later, the user may create a second task environment (illustrated by the two light-emitting objects in Fig. 1b) at which point controllers are evaluated against both environments and the fitness of a controller is now computed as

$$F = \frac{\sum_{i=1}^{k} f_i}{k} \tag{2}$$

where the controller is evaluated in k task environments and f_i denotes fitness in the ith environment (Eqn. 1).

3. RESULTS

To test robot shaping using the system, locomotion toward the light-emitting object was first selected for (Fig. 1a).

Once successful locomotion was achieved, a second task environment was added (Fig. 1b) by copying the original task environment—the start-position robot, the end-position robot and the object—and moving the object in the second environment slightly to the right, along with the end-position robot (not shown). After a short period of evolution the controllers became mired in a local optimum: an easy solution for evolution to find is to ignore the photosensors and instead produce an effectively open-loop controller that causes the robot to locomote in the same manner in both environments. This is illustrated in Fig. 1b by the fact that there is only one locomotion trajectory even though the controller was evaluated twice.

This local optimum was removed by dragging the light-emitting object in the second environment further to the right. This reduced the fitness of controllers that ignore the photosensors and allowed for the evolution of controllers that use the photosensors to alter the robot's trajectory toward the object. Thus after a short subsequent period of evolution a controller was discovered that allowed successful travel toward both placements of the light-emitting object (Fig. 1c). A third task environment was created and evolution continued until success was achieved, and finally a fourth environment was constructed and a controller evolved that succeeded in all four environments (Fig. 1d).

This experiment illustrates how a user can lead optimization out of local optima by altering the robot's task environment rather than altering the fitness function. The ability to construct different robot morphologies was added to the system, and a second robot was constructed with the ability to brachiate. This robot also became mired in a local optimum: it found a way to swing up between the rungs and 'walk' over the top of them. The user interactively guided evolution out of this optimum by placing barriers above the rungs, thus forcing the robot to evolve the ability to swing from one rung to the next (see the accompanying videos).

4. CONCLUSIONS

Code-free robotics promises to provide a novel entry point for students interested in the field. We have tested the system with undergraduate students who have had 14 weeks of formal instruction in evolutionary robotics. Many of them were able to construct a robot morphology and task environment, and evolve successful behaviors for them in a 50 minute period. In future work we wish to expand the social aspect of the system. It may be that multiple users collaborating on the same robot design—or alternatively forking off novel designs of their own—may uncover more local optima than an equal number of users working independently.

Acknowledgments

This work was supported by Defense Advanced Research Program Agency Grant W911NF-1-11-0076 and National Science Foundation Grant PECASE-0953837.

5. REFERENCES

[1] J. Clune and H. Lipson. Evolving three-dimensional objects with a generative encoding inspired by developmental biology. In *Proceedings of the European Conference on Artificial Life*, pages 144–148, 2011.

[2] R. Dawkins. *The Blind Watchmaker*. Harlow Longman, 1986.

[3] M. Dorigo and M. Colombetti. Robot Shaping: Developing Autonomous Agents Through Learning. *Artificial Intelligence*, 71(2):321–370, 1994.