JungleBoogie: A System for Studying Brain-Body Evolution of Virtual Creatures

Richard Giuly

College of Computing
Georgia Institute of Technology, Atlanta, GA 30332
rgiuly@fastmail.fm

This paper describes JungleBoogie, an extensible system that generates and optimizes the brain and body of virtual creatures. Evolution occurs in a virtual 3-D world that simulates the physics of motion. The creature bodies consist of block segments with connecting joints. In each creature, a neural network functions as a brain. Network outputs drive actuators that move joints. Virtual light sensors feed into network inputs. A user-specified evaluation function defines the fitness of an individual, and fitness determines the probability of reproduction. JungleBoogie evaluates creature fitness based on given criteria such as ability to push an object or follow a light source.

Introduction

Nature has generated many versatile, robust control systems with corresponding physical bodies, e.g. the brain and body of an ant. The purpose of this project is to mimic nature's evolutionary design process and create intelligent virtual creatures in a simulated physical environment. Evolved control systems could then be transferred to real robotic bodies.

Evolving real world skills like object manipulation requires evolution in an environment like the real world. JungleBoogie simulates the real world with a rigid body dynamics engine called breve [2]. Virtual creatures are evaluated based on their ability to sense and interact. JungleBoogie uses Newtonian physics so that the creatures will operate in the same logical world we do.

In the spirit of nature, JungleBoogie allows a creature's brain and body to evolve simultaneously. Coevolving the creature's brain and body helps keep complexity matched. That is, coevolution avoids generation of a body incapable of carrying out the brain's commands or a brain too clumsy to control an expressive body.

Previous work

JungleBoogie is based on the work of Karl Sims and Gregory Hornby. Sims created evolving creatures with sensors that could actively follow a goal object [4]. Hornby described parametric Lindenmayer-system encoding as a way to generate more “natural” structures [1]. JungleBoogie integrates the ideas of Sims, Hornby, and others into a single system that allows experimental evaluation of various evolutionary mechanisms. It is an open-source project that will make evolving brain-body research accessible to people without great programming overhead.

Software Features

Evaluating Creatures

JungleBoogie contains an extensible object-oriented system for evaluating population members based on a user specified fitness function and environment. It describes a generic evolutionary environment and fitness function in a class called EvaluatePopulation. This class contains the population of creatures and the framework of the genetic algorithm. By extending EvaluatePopulation, a programmer can define evolution scenarios, fitness functions, and initial environments. Environments may involve movable blocks or any other physical object supported by breve.

Brain and Body Representation

JungleBoogie encodes creature morphology in a parametric L-System grammar. The grammar deterministically expands into a string of commands that indicate how to build the body. This indirect representation creates fractal-like morphology with patterns, e.g. multiple identical limbs. Some commands are similar to logo turtle commands like up, down, left, and forward. Others commands add a joint and a body part at the current location, just as described in [1]. Creatures may have block body parts or wheel-shaped body parts.

Brains are also in encoded in a parametric L-System grammar. The commands operate on edges in the creature's neural net which is a directed graph. Exactly as in [1], commands can split an edge and add a new neuron, merge two neurons, set the edge weight, and set neuron parameters such as activation function.

Genetic Algorithm

Jungle Boogie generates a population, evaluates every member separately and then regenerates a new population. The probability of members reproducing is proportional to fitness. For crossover, JungleBoogie selects a random L-System production rule successor from one creature and copies it into the target creature. Point mutation involves changing, inserting, and deleting symbols in the L-System grammar that defines the creature. There are two modes of neural net mutation. The first uses the L-System grammar to generate a string of commands that create a network [1]. The second mode occurs optionally after the first and modifies the network by arbitrarily adding or deleting edges and nodes. The user specifies whether or not to enable the second mutation mode.

Actuation and Sensing

After interpreting the morphology description, JungleBoogie uses breve to create physical body segments connected with joints. Each time a joint is created, a corresponding actuator neuron is created and attached to the neural network. The output of the actuator neuron controls rotation speed about the joint.

JungleBoogie allows for multiple types of emitters and detectors analogous to light sensors sensitive to different colors. Every body segment is created with a detector for each color of light. A creature can optionally have a set of sensors that remain in fixed positions relative to its main body segment.

Implementation

JungleBoogie runs within breve [2], a user-friendly programming environment that provides dynamic physics simulation, OpenGL display, and GUI interfacing. JungleBoogie's object oriented code is extensible: new evolutionary strategies can be tested, and different evaluation functions can be added. The graphical interface displays creatures during evaluation and optionally displays the state of the neural net during evaluation. JungleBoogie is open source: code is available at www.jungleboogie.org.

Experiments

Three subclasses of EvaluatePopulation were created: EvaluatePuser, EvaluateSeek, and EvaluateDoubleSeek. Each subclass defined a different kind of fitness: pushing, seeking, and bi-directional seeking. Single creature evaluations occurred over a period of ten seconds. On each machine, the population's size was 30. Eight separate populations were evaluated on eight different microprocessors. About 24 hours of evolution was usually enough to produce interesting results.

To evaluate pushing, fitness was proportional to the average velocity of a block initially lying next to the creature. Some of the EvaluatePusher creatures simply hit the block one time with a baseball-bat appendage. One creature evolved locomotion and a hook shaped appendage that pushed the block efficiently.

To evaluate seeking, fitness was proportional to velocity in one specific direction, toward a fixed target. The target was always in the same place. Simple creatures evolved, some with only three segments that could move to the target.

For Bi-directional seeking, JungleBoogie evaluated each creature twice, one time where it had to seek a target behind it and another where it had to seek a target in front. The total fitness was the sum of the average velocities toward the correct target. Proximity sensors fed the neural net with input about distance from the center of each body segment to the target. Evolution successfully found a way to use proximity sensor data to tell whether to go back or forward: one resulting creature was able to seek the target object while a human tester moved the object arbitrarily in either direction. Videos of evolved creatures can be viewed at www.jungleboogie.org.

Future Work

The experimental fitness functions of seeking and pushing were purposely simple enough to be solvable in a short time, but future work should explore more complex fitness functions. There is no reason for intelligence to evolve when all a creature has to do is move fast toward a target. One way to increase difficulty is with competition: evaluation of multiple interacting creatures will be an important addition to JungleBoogie's features. Competing, coevolving creatures automatically increase the difficulty of tasks incrementally as they improve. For example, consider the evaluation function of final score in a soccer game against other creatures. As opponents evolve better strategies, difficulty increases.

An alternative way of representing the brain is with procedural code rather than a neural net. This alternate encoding would harness the power of Genetic Programming. It would allow the brain to build up from evolved building blocks, automatically defined functions [3].

Creatures could have more sensing ability like contact sensors, joint position sensors, and vision. Vision sensors, available in breve, would allow the creature to operate more like insects do in real life. Vision sensors create an image from the creature's point of view. If all the numeric color values of pixels in the image were fed to the creature's neural net, it could evolve a system for processing visual information.

Acknowledgements. I am grateful to Dr. Tucker Balch of the Georgia Institute of Technology for setting project milestones, giving helpful advice, and providing necessary computing equipment.

References

1. Hornby, G. S. and Pollack, J. B. Body-brain coevolution using L-systems as a generative encoding. In *Genetic and Evolutionary Computation Conference*, pages 868-875, 2001.
2. Klein, J. 2002. breve: a 3-D simulation environment for the simulation of decentralized systems and artificial life. In *Proceedings of Artificial Life VIII, the 8th International Conference on the Simulation and Synthesis of Living Systems*. The MIT Press.
3. Koza, J. R. 1994. *Genetic Programming II.* MIT Press.
4. Sims, K. Evolving Virtual Creatures. In *SIGGRAPH 94 Conference Proceedings*, Annual Conference Series, pages 15-22, 1994.