Artificial Life Through Evolutionary Computation

California Institute of Technology
Computing Beyond Silicon Summer School
August 17, 2002

Bryan Blumenkopf
Massachusetts Institute of Technology
bblumen@mit.edu

Ashley Holtgraver
Carnegie Mellon University
ashleyh@andrew.cmu.edu

INTRODUCTION

The following is a summary of work prepared over a two-week period as part of the Computing Beyond Silicon Summer School, offered by the California Institute of Technology during the summer of 2002. The objective was to prepare a pedagogical lecture on large-scale artificial life as an outgrowth of methods in the computational simulation of evolution, aimed toward an audience of graduate students and advanced undergraduate students in biology, chemistry, physics, engineering, and computer science.

OVERVIEW

Artificial life is loosely defined as the computational simulation of biological systems, both small (cells and cell networks, genetics, etc.) and large (organisms, populations, evolution). Here we focus on large-scale artificial life, in particular the modeling of virtual creatures through evolutionary methods.

Though other approaches are certainly possible, a powerful and well-studied method of simulating large-scale biological systems is *evolutionary computation*, drawing inspiration from Darwinian natural selection. Evolutionary computation simulates natural selection algorithmically, automatically "evolving" optimal solutions according to a prespecified measure of fitness. In particular we discuss the genetic algorithm as the primary basis of evolutionary computation.

Lastly, we show one method of evolving virtual organisms by summarizing the work of Karl Sims in his landmark 1994 publication on the subject.

EVOLUTIONARY COMPUTATION

Evolutionary computation (EC) is a general term for several computation techniques which are all based to some degree on the development of biological life in the natural world. Currently there exist several major evolutionary models.

The *genetic algorithm* (GA), by far the most common application of EC, is a model of machine learning taking inspiration from genetics and natural selection. In natural evolution, each species searches for beneficial adaptations (species optimizations), which arise through mutation and the chromosomal exchange and recombination of breeding. The two key axioms underlying the genetic algorithm are that complex nonbiological structures can be described by simple bit strings (analogous to the "genetic code" of chromosomes), and that these strings could be improved, according to a particular measure of fitness, by the application of simple transformation functions (just as living species may be "improved" through mating). This will be described in detail shortly.

Evolutionary strategies (ESs) simulate natural evolution similarly to the genetic algorithm. Like GAs, ESs are most powerful

while comparing populations of data, as opposed to individual samplings. Differences between the two lie in their application; ESs were designed to be applied to continuous parameter optimization problems seen in laboratory work, while the GA was used originally in integer optimization problems.

Evolutionary programming (EP) is a stochastic optimization function, similar in many ways to the GA. However, EP places emphasis on the behavioral link between parent and offspring, as opposed to the GA's attempt to model the exact code transition as seen in nature. EP follows a general process with obvious similarities to natural evolutionary progression. An initial population of trial solutions is selected at random from a coding scheme. A chosen mutation factor is applied to each solution, generating a new population. Because EP resembles biological evolution at the level of reproductive populations of species, and there is no genetic recombination between species, EP's transformations take place without *crossover*- combination of two parent member's genetic code. The offspring species' members are weighed for overall fitness; the best are kept while the rest are culled, and the algorithm repeats with the new, more fit population.

The *learning classifier system's* (LCS) purpose is to take in input and produce an output representing a classification of that input. They have undergone and continue to go through multiple minor changes of name and scope, but the enduring foundation originates in J.H. Holland's Adaptation In Natural and Artificial Systems, wherein he envisions a cognitive system capable of classifying and reacting appropriately to the events in its corresponding environment. This most obviously parallels the inherently intelligent behavior seen in all macro- and microscopic living creatures.

Though there are certainly other forms of evolutionary computation, the above offers a brief summary of the most established and useful evolutionary techniques. The genetic algorithm in particular lends itself well to the macroscopic forms of artificial life.

GENETIC ALGORITHMS

Genetic algorithms use the evolutionary process of natural selection as a metaphor for what is essentially a hill-climbing search without backup. GAs search for an optimum or global maximum among what is typically an enormous set of data by computationally modeling the alteration, recombination, and propagation of genes that forms the basis of biological evolution. To achieve this, certain complex biological details of evolution must be abstracted in favor of more relevant principles. Thus GAs share the following properties:

- GAs process an encoding of relevant parameters, typically represented as a DNA-like string.

- GAs search among a population (typically large) of individual state nodes.

- The optimality of each state node is evaluated according to a specified *fitness function*. This function determines, probabilistically, which nodes are propagated.

- Node propagation ("reproduction") is nondeterministic. This discourages the GA from mistaking local maxima for global maxima. Genetic splitting and pairing, as well as phenomena such as *crossover* and *mutation*, are modeled probabilistically.

Assuming that parameter encoding, population size, propagation iterations, genetic operators, and a fitness function have been chosen, the genetic algorithm proceeds as follows:

1) A population of given size is initialized.
2) For a specified number of generations:
   a) Assign each individual node a fitness level according to the fitness function.
   b) Probabilistically select a specified number of pairs of individuals according to fitness levels. Higher fitness levels increase an

individual's chance of being selected.

c) Apply the specified genetic operators to these chosen pairs to produce new individuals.

d) Randomly select individuals from the population. Replace them with the newly produced individuals.

3) Return the individual with the highest fitness level as the output.

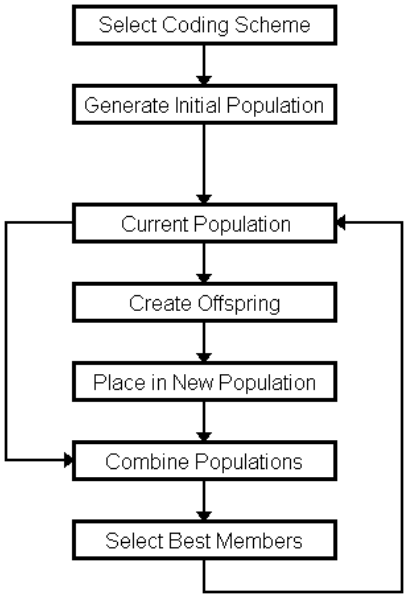

Figure 1. Graphical outline of the genetic algorithm.
From http://members.aol.com/btluke/evprog.htm

A careful choice of genetic operators can improve the efficiency of the genetic algorithm or enable it to find otherwise inaccessible solutions.

Crossover switches two subsequences of two parent strings; the goal is to place two fit sequences on the same string. Subsequences are selected probabilistically.

124423 | 424314224 331412424314224
- - >
331412 | 311132314 124423311132314

Mutation introduces "genetic diversity" into the population by randomly altering one character of an individual string. Mutation provides a way to help the GA avoid the situation in which the system fixates on a local maximum after repeatedly propagating a particular character.

1441**2**3142132 - - > 1441**4**3142132

Inversion reverses the order of a particular subsequence.

314**121313**3141 - - > 314**313121**3141

EVOLVING VIRTUAL CREATURES

An important means of simulating the evolutionary development of large-scale organisms was described by Karl Sims in the landmark 1994 publication, "*Evolving Virtual Creatures*." Sims employed genetic algorithms to traverse a directed graph bound by the constraints of a modeled physical world and ultimately create a physically feasible computational creature optimally designed to accomplish a particular task; the following briefly summarizes the premise of his work.

Each node in Sims's directed graph represents a specific physical part – such as a head, arm, or hand -- in an analogous 3D physical structure corresponding to a particular graph traversal *(see Figure 2)*. A virtual "creature" is generated by traversing the directed graph; recursion limits and constraints on the placement and physicality of parts are defined by variables intrinsic to each node and internodal connection.

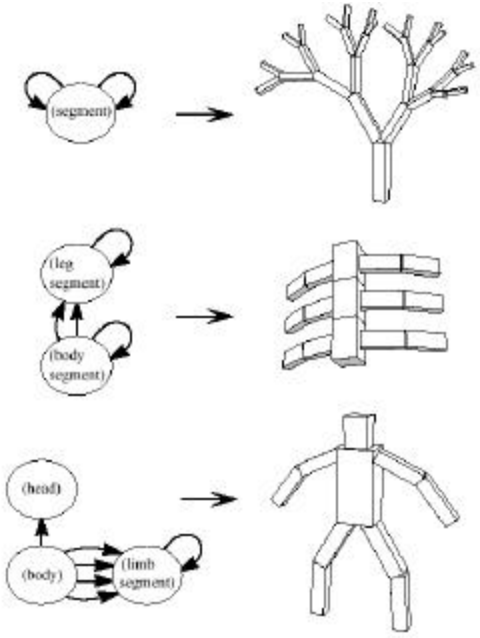

Figure 2. Correspondence between directed graph and 3D morphology in Sims's study. From Karl Sims, Evolving Virtual Creatures.

Behavior of the creatures in this simulation is controlled by a dynamic "brain," accepting sensor values (such as joint angles, surface contact, or luminance values) from the physical environment and from virtual sensors on the morphological parts and outputting effector values as forces or torques applied to specific parts of a creature. These sensor and effector values are represented as positive or negative scalar quantities.

Contained within each graph node are internal "neural nodes," assembled as an artificial neural network. This "neural structure" is assembled with the physical morphology of each creature. This network gives each creature a sense of internal state, enabling probabilistic or arbitrary behavior rather than the deterministic behavior that would result from purely sensory response. Figures 3 and 4 show an example of a descriptive graph and the resulting physical creature.

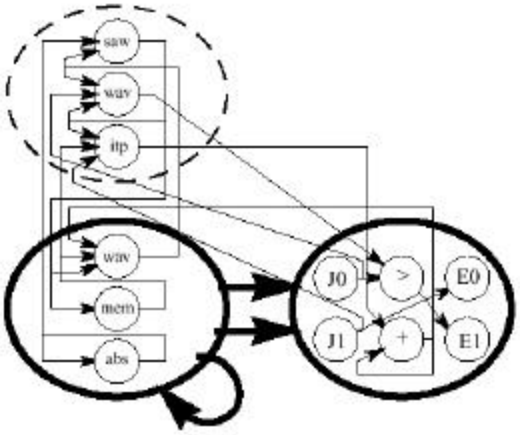

Figure 3. Example of physical structure and underlying neural structure. From Karl Sims, Evolving Virtual Creatures.

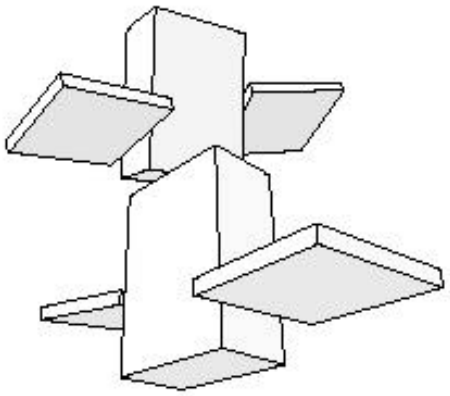

Figure 4. Corresponding physical morphology. From Karl Sims, Evolving Virtual Creatures.

The above outlines a basic description of the framework that Sims established for generating both the morphology and the control mechanisms for virtual "block creatures." An indefinite number of possible creatures exist; these creatures could be generated automatically and then "evolved" using genetic algorithm methods as described earlier. Sims placed these creatures in a simulated physical environment, and matched their performance on certain tasks – walking, jumping, swimming, following – to determine a measurement of fitness. The genetic algorithm then proceeded to mutate internal node characteristics and connection parameters, and to add or delete nodes and connections of existing creatures to create

a successive generation. Sexual reproduction was modeled by combining the nodal structures of two existing creatures. The end result of any particular iteration of this process was an automatically generated creature nearing an optimal level of fitness. Some resulting creatures of Sims's work are shown in Figures 5, 6, and 7.

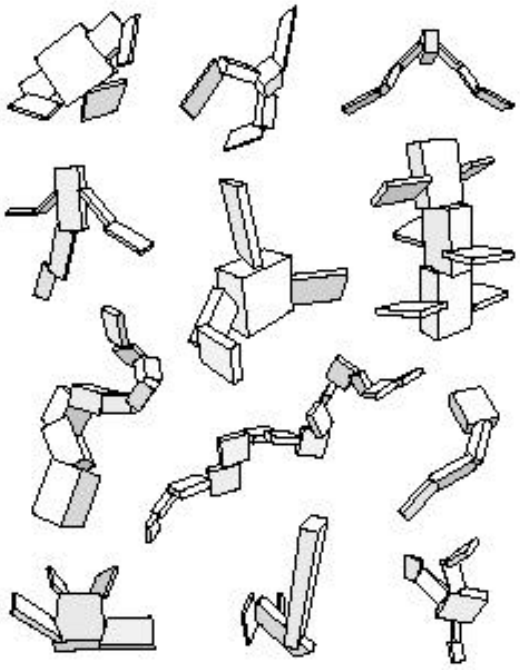

Figure 5. Example creatures evolved for optimal swimming. From Karl Sims, Evolving Virtual Creatures.

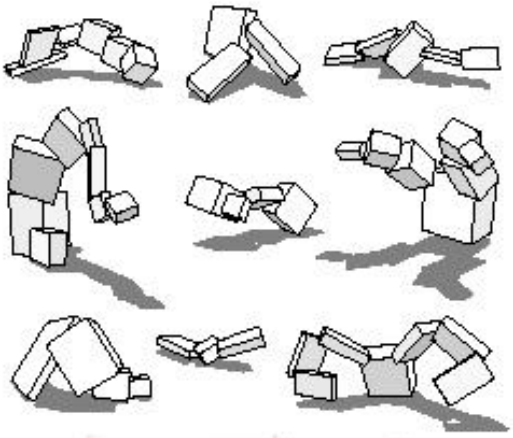

Figure 6. Example creatures evolved for optimal walking. From Karl Sims, Evolving Virtual Creatures.

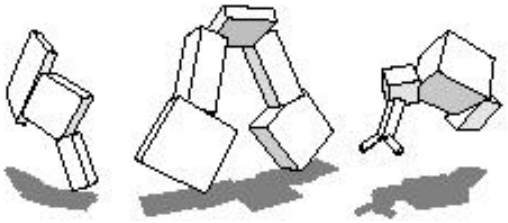

Figure 7. Example creatures evolved for optimal jumping. From Karl Sims, Evolving Virtual Creatures.

The results of Sims's work demonstrate that clever application of methods of evolutionary computation hold great potential for the autonomous generation of computational simulations of large-scale organisms.

REFERENCES

Adami, Christoph. (1998). *Introduction to Artificial Life*. New York: Springer-Verlag.

Holland, J.H. (1992). *Adaptation in Natural and Artificial Systems.* Cambridge, MA: MIT Press.

Sims, Karl. *Evolving Virtual Creatures. Computer Graphics*, Annual Conference Series, (SIGGRAPH '94 Proceedings), July 1994, pp.15-24.

Sims, Karl. *Evolving 3D Morphology and Behavior by Competition. Artificial Life IV Proceedings,* ed. by R. Brooks and P. Maes, MIT Press, 1994, pp.28-39.