# Evolving Foraging Behaviors

Liviu Panait and Sean Luke
George Mason University
http://www.cs.gmu.edu/∼eclab

Insects are particularly good at cooperatively solving multiple complex tasks. Some such tasks, such a foraging for food far away from the nest or clustering objects into piles, can be solved through relatively simple behaviors in combination with communication mechanisms using pheromones. As task complexity increases, however, it may become difficult to determine the proper simple rules which yield the desired emergent cooperative behavior; or to know if any such rules exist at all. For such tasks, machine learning techniques like evolutionary computation (EC) may prove a valuable approach to searching the space of possible rule combinations.

As a first step towards this goal, we present a proof of concept experiment designed to show that learning techniques can successfully discover good performing foraging strategies which use only pheromone information. The significant contributions of this work are twofold. First, this paper presents a successful approach to learning both the pheromone-depositing and movement-decision-making aspects of an ant colony foraging strategy. Previous work has mainly described methods to learn only movement-decision-making. Second, this paper presents the first learning method which solely uses pheromone information rather than, as was done in previous work, providing hand-coded procedures for returning to the nest.

We performed four experiments to demonstrate the efficacy of EC to learning multi-pheromone problems. Our experimental substrate was a toroidal grid environment with a single nest and point food source, and between 50 and 500 ants. For each approach, we drew a sample of ten runs, and applied a Welch two-sample statistical test at 95%. In the experiments we used the ECJ[1] evolutionary computation system, and new multi-agent simulation library, MASON, introduced in another paper submitted to this workshop.

To evolve ant behaviors, we used strongly-typed genetic programming (GP) [Koza 1992][2]. In the common form of genetic programming, which we adopted, evolutionary individuals (candidate solutions) use a parse tree structure representation. Leaf nodes in the parse tree return parameters representing external state values for the ant. Internal nodes in the tree return functions applied to the return values of their subtrees. Crossover swaps subtrees among individuals. In strongly-typed GP, type constraints are placed on which nodes may be added as children of other nodes: we used strong typing

[1] http://www.cs.gmu.edu/∼eclab/

[2] Koza, J. 1992. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. MIT Press.

to enable a large set of available functions operating on vectors, scalars, and directional information. Even so, the representational complexity available to the GP learner was significantly less than that afforded in the hand-coded designs we discussed in another paper submitted to this workshop. Further EC details will appear in the workshop paper.

EC individuals consisted of two GP trees: the evaluation of one tree yielded the amount of pheromone to deposit, and the evaluation of the other tree yielded the direction to move. To test an EC individual, all ants would iteratively apply the individual's GP trees to their own local situations. The quality of an EC individual was the total amount of food brought back to the nest during its test period. Ants used two pheromones (the "food pheromone" and the "nest pheromone"). We suspected that in any ideal solution an ant would apply the same algorithm to forage as it would in carrying food back; but the particular pheromones used would be swapped (food for nest, and vice versa). This heuristic symmetry allowed us to simplify the state space, using two trees rather than four, and swapping the pheromone inputs and outputs based on the ant's current state.

Our first three experiments scaled in the number of ants (50, 50, 500), number of simulation time steps (501, 1001, 2501), and world size (10x10, 33x33, 100x100). In each case the EC populations converged rapidly to simple but reasonably high-performing ant foraging behaviors. Increasing the world size led to longer convergence times (from a mere two generations in the 10x10 case to ten generations on average in the 100x100 case). Interestingly, these behaviors were different in meaningful ways from one another, but all three learned behaviors yielded pheromone-depositing schemes similar to those used by hand in the companion paper at this workshop. This scheme, which "topped off" the pheromone value to the maximum of its current value and some desired value, considerably outperformed the more common scheme in the literature, namely simple addition.

This led to our fourth experiment, in which we selected the highest-performing behavior from each of the experiments, and compared the ten runs each of the three behaviors in all three experimental environments. The results of this experiment were interesting. In the 10x10 environment, all three behaviors produced statistically identical results. In the 33x33 environment, the 10x10 learned behavior fell short, bringing in only one ninth of the food of the other two. In the 100x100 environment, the 100x100 behavior was statistically significantly better than 33x33 environment, bringing in twice the total food. In this environment, the 10x10 behavior brought in almost nothing. In summary, we saw a total ordering among behaviors: the behaviors evolved on larger environments did as well as others on smaller environments, and outperformed them on larger environments.

For future work, we aim to scale this experiment to more complex behaviors, such as foraging multiple food sources that can become depleted with time, and additional tasks such as guarding the foraging trails from predators. However, even with a fast simulator, evolving solutions to these tasks may be daunting, requiring a large amount of computing power. To scale successfully, we will also need to examine approaches to speeding up the learning process: for example, adjusting the evolution procedure and representation, or "bootstrapping" complex behaviors by seeding them with decomposed, learned simpler behaviors.