Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

ANTFARM: TOWARDS SIMULATED EVOLUTION

Robert J. Collins
David R. Jefferson

December 1990
CSD-900048

AntFarm: Towards Simulated Evolution

Robert J. Collins
David R. Jefferson

November 1990

Technical Report UCLA-AI-90-10

AntFarm: Towards Simulated Evolution*

Robert J. Collins†
David R. Jefferson‡
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90024

Abstract

The most easily observed ant behavior is workers foraging for food. Foraging workers do not eat the food, but carry it back to the nest, where it is processed and consumed by all members of the colony. In many species, a high degree of coordination and cooperation between foragers is observed (usually mediated by pheromone communication).

We would like to understand more about the evolution of cooperative foraging. In this paper, we describe a computer program called AntFarm, that simulates the evolution of foraging strategies in colonies of artificial organisms that resemble ants. AntFarm is work in progress, and is being used to investigate issues surrounding simulated evolution of complex behaviors in complex environments, the evolution of cooperation among closely related individuals, and the evolution of chemical communication. We describe our genetic algorithm for simulating evolution. We also discuss the issue of the representation of artificial organisms, and empirically compare several ANN encodings based on their ability to evolve foraging behavior in AntFarm ants.

*To appear in J. D. Farmer, C. Langton, S. Rasmussen, and C. Taylor (Eds.), *Artificial Life II*, Addison–Wesley, in press.

†Electronic mail address: rjc@cs.ucla.edu.

‡Electronic mail address: jefferso@cs.ucla.edu.

1 Introduction

We are attempting to simulate the evolution of complex behavior (rather than physical morphology) in artificial organisms. In this paper, we consider the simulation of organisms that live and reproduce in relatively complex environments, with many sensors (external and internal), and many possible actions at each moment. In addition, the organisms possess internal memory, allowing their behavior to be history sensitive. In the course of its life each organism is born, makes thousands of decisions (eat, move, etc.), and eventually dies. The reproductive success of each organism is affected by its behavior throughout its lifetime.

We are particularly interested in the evolution of cooperative central place foraging in ants. AntFarm is a computer program that simulates an evolving population of ant colonies whose reproductive success is a function of the amount of food carried to their nest, producing a selection pressure favoring better foraging strategies. Each colony is made up of a small number of genetically identical ants, whose behavior is specified by an artificial neural network (ANN). In addition to the ability to sense and carry food, the ants can sense and drop pheromones (chemicals used by ants for communication).

AntFarm is work in progress. Eventually we will attempt to determine the conditions that are necessary for the evolution of cooperation (mediated by chemical communication) in central place foraging. Johnson, Hubbell, and Feener have developed a model of optimal central place foraging in ant colonies that predicts that the degree of cooperation should be a function primarily of the distribution of food in the environment [14]. While this model predicts when cooperation pays off, under what conditions an optimal strategy will actually evolve is an open question.

So far, we have completed the implementation of AntFarm, and are able to consistently evolve (solitary) foraging behavior. To get to this point, we have had to design both a genetic algorithm that closely resembles natural evolution, and a new ANN representation that is more suitable for evolutionary experiments than those used in previous studies. Our genetic algorithm uses local competition and mating, rather than the usual panmictic (random mating) scheme. In addition, we feel that a clear separation between the genetic algorithm and the simulated world/organisms is necessary to conduct unbiased evolutionary experiments. Hence our genetic operators are applied to structureless bit string chromosomes.

The need for an appropriate artificial organism representation has been a major obstacle, which we have recently overcome. A new ANN representation is necessary, because other behavior function representations either are not appropriate for biologically motivated simulated evolution, do not scale well to number of inputs/outputs required for AntFarm, or empirically are not capable of evolving foraging behavior in AntFarm. From this struggle, we have abstracted a number of properties that are necessary for organism representations that are to be used in simulated evolutionary experiments.

2 Microanalytic Simulation of Evolution

The computer simulation of evolving populations is important in the study of ecological, adaptive, and evolutionary systems [22]. Only the simplest genetic systems can be completely solved analytically, and evolutionary experiments in the laboratory or field are usually limited to at most a few dozen generations and are difficult to control and repeat. Simulated evolution makes it possible to study evolutionary systems over hundreds or even thousands of generations. By their very nature, computer simulations are easily repeated and varied, with all relevant parameters under the full control of the experimenter.

Most computer simulations in biology (including evolutionary simulations) are based on solving differential equations from mathematical models [20, 21]. Due to mathematical limitations, models of evolving systems are usually simple and unrealistic. Complex models that incorporate a large number of both intrinsic factors (e.g. the life history of the organisms) and extrinsic factors (e.g. weather, competitors, etc.) are more accurate and useful. Unfortunately, such complex evolutionary models are difficult or impossible to describe analytically.

Although most biological simulations are equation–based, simulations based on the observation that the execution of a computer program is very similar to the life of an organism have emerged in recent years [22, 23, 7, 4, 24, 13]. In such simulations, each organism is represented by a program, as are the various environmental processes: the population of executing programs simulates a population of living organisms and the environment. Rather than attempting to capture the complex global dynamics of the population and environment in a set of equations, only the local interactions between

the individual organisms and environmental factors are modeled. Based on these relatively simple local interactions, the complex global behavior of the population emerges.

This sort of "life–as–process" simulation is referred to as *microanalytic*, meaning that each individual organism and environmental effect is separately represented, and the biologically significant events in an organism's life are all separately simulated in detail [3]. Each organism in the population is represented as a program, and its life as a process: a detailed sequence of events, including its birth, its interactions with a dynamic environment (potentially including many of the other organisms in the population), its mating and reproduction (if any), and its death.

2.1 Biological Issues

While we cannot use simulated evolution to reconstruct an actual situation in the history of natural life, we can explore particular hypotheses, eliminating some and giving credence to others. Such simulations provide the researcher with an artificial world in which to perform evolutionary experiments that can be fully controlled and repeated, and can span a large number of generations. Simulated evolutionary experiments might someday be used to shed light on a number of open evolutionary problems, including

- modes of speciation (which of the many hypotheses are most likely, and in which sexual systems and ecological situations),
- the evolution of mutation and recombination rates,
- the evolution of information processing behavior (e.g. sensory–motor integration, communication, etc.),
- the evolution of sexual reproduction (i.e. why is it maintained in competition with asexual reproduction?),
- punctuated equilibria (i.e. is it true that most evolution occurs at speciation events, and not within species?),
- the dynamics of the evolution of predator–prey "arms races,"
- the influence host–parasite interaction on evolution rates,

- the stability of ecosystems,
- the evolution of evolutionarily stable strategies,
- sexual selection and the evolution of maladaptive characteristics, and
- the evolution of cooperation (especially among kin).

We have focused our attention on a smaller question: the evolution of central place foraging strategies in ants. We are exploring the evolution of the use of chemical communication and cooperative foraging within ant colonies.

The dominant insects throughout the world are the ants. All ant species have eusocial societies, characterized by overlapping generations, care of young by adults, and adults divided into reproductive (kings and queens) and nonreproductive (workers) castes. Ants live in colonies ranging in size from a few individuals to more than 20 million, all with a high degree of organization. Nearly all communication between ants is either tactile, visual, or chemical. Large–scale coordination is achieved through the use of pheromones.

Each individual ant is relatively small and simple, typically performing only 20 to 42 distinct behaviors [12]; yet the emergent behavior of the colony as a whole is amazingly complex. In many contexts, myrmecologists treat the whole colony as a single *superorganism.* The unparalleled success of these superorganisms in all parts of the world (perhaps as many as 20,000 species [12]) speaks well for the strength and versatility of the eusocial colony.

Although we are strongly motivated by the example of real ants, we do not feel bound to model them exactly. Our goal is to use AntFarm to verify that approximately optimal cooperative foraging behavior consistent with a model proposed by Johnson et al. [14] can evolve by natural selection, and to explore the conditions favorable to its evolution.

Optimal Central Place Foraging

Central place foraging consists of two phases: the search for food and its recovery to a central location [19]. Much of the cost of foraging is associated with search [6, 17], but all of the payoff is from recovery, which consists primarily of transportation of the food to the nest. Foraging strategies that minimize search time will clearly be advantageous.

The Johnson, Hubbell, and Feener model of central place foraging in eusocial insects is fairly complex and the details are beyond the scope of this paper, but it predicts the effect of the size of food patches on the number of foraging workers and the style of foraging that is used. In species that feed on small patches of food, a small number of workers, each foraging alone is optimal. Recruitment of nestmates to help recover the food does not pay off because the food patches are small. In this model, the search for food (in the absence of recruitment) is assumed to be a random walk beginning at the nest so that the area around the nest is searched many times by different foragers. As the number of foragers increases, the amount of additional area searched decreases. The diminishing returns for additional foragers results in a small foraging force being optimal.

Species that feed on large patches of food should have a large foraging force, with heavy reliance on recruitment. When a patch is too large for the discovering ant to harvest alone, it pays to recruit (rather than rely on rediscovery by other foragers). Recruitment of nestmates to help harvest a known food source can nearly eliminate search costs. With reduced search costs, the diminishing returns for additional workers is not such an important factor, resulting in a large foraging force being optimal.

In real ants, recruitment to harvest food resources takes many different forms [12]. In the simplest case, a second ant is physically led to the food in a process called *tandem running.* More common is *group* recruitment, which uses a short–lived pheromone trail to bring up to a few dozen workers to the food source. The most impressive form is called *mass* recruitment. In mass recruitment, a relatively fixed, long–lived pheromone trail leads hundreds or thousands of workers to the food source. The trail is reinforced by each successful worker. Mass recruitment is used only in species that forage for food that is found in very large clumps.

The Genetic Algorithm

The biological focus of the AntFarm experiments requires us to closely model the process of natural evolution. Genetic algorithms are loosely based on natural evolution, and have been used by computer scientists and engineers as an optimization method for more than 25 years [11]. Unfortunately, traditional genetic algorithms are not well suited for simulated evolution.

Genetic algorithms are typically used to search for good solutions for

complex optimization problems [9], i.e. a string of function parameters that (more or less) optimizes a particular function. A genetic algorithm evolves a population of these strings (chromosomes) by assigning each a score (fitness value), based on the quality of the solution. The likelihood that a particular string will be chosen for mating is a function of its score and the rest of the population. The key to the genetic search is that those chosen to mate reproduce with variation.

The mechanics of genetic algorithms are relatively simple, consisting of four basic parts:

1. assigning fitness scores,
2. selection and mating,
3. recombination, and
4. mutation.

The assignment of fitness scores is wholly dependent on the particular application. New populations are created by the repetition of these steps. Because selection is biased towards strings with higher scores, the populations typically achieve higher and higher scores as generations pass. In this section, we briefly describe our genetic algorithm and informally compare and contrast it with traditional genetic algorithms and natural evolution systems.

In most genetic algorithms, there is only one gender, so that any individual can mate (sexually) with any other individual (although simple extensions allow for two or more genders). The parents of the next generation are selected probabilistically based on their score (defined by the *objective* or *fitness function*) and the scores of all the other members of the population. Let f_i be the fitness score of string i, and N be the number of strings in the population. The probability that string i is chosen to be a parent is usually defined to be something like

$$P(i) = \frac{f_i}{\sum_{j=0}^{N-1} f_j}, \tag{1}$$

and the strings are randomly paired according to this distribution for sexual mating.

Although this panmictic (random global mating) scheme is simple and widely used in genetic algorithms, it is a poor model of real evolution. One of the basic assumptions of Wright's shifting balance theory of evolution is that spatial structure exists in large populations [25, 27, 28, 29, 30, 5]. The structure is in the form of *demes* [8], or semi–isolated subpopulations, with thorough gene mixing within a deme, but restricted gene flow between demes. One way that demes can form in a continuous population and environment is *isolation by distance*: the probability that a given parent will produce an offspring at a given location is a function of the geographical distance between the parent and offspring locations.

To simulate isolation by distance in the selection and mating process, we place the artificial organisms on a toroidal, 2–dimensional grid, with one organism per grid location. Selection and mating take place locally on this grid, with each individual competing and mating with its nearby neighbors. In his quantitative analysis of isolation by distance, Wright assumes a normal distribution for parent–offspring distances [28, p. 303]

> Normal distributions of parents relative to offspring are to be expected if dispersion occurs by a long succession of random movements...

In our genetic algorithm, the parents are chosen during short random walks that begin at the offspring location, one parent per walk. The highest scoring individual encountered during the random walk is chosen as the parent (breaking ties in favor of those encountered later in the walk). The parents are chosen with replacement, so it possible for the same high–scoring individual to be encountered during both random walks, in which case it would act as both parents for the offspring.

In genetic algorithms used for optimization, it is common to exploit problem specific or representation specific information in the implementation of the genetic operators in an effort to speed the search. However, to create unbiased and realistic evolutionary experiments, it is necessary to avoid building the experimenter's preconceptions into the simulation. Therefore, we require a clear separation between the genetic algorithm and the simulated organisms/environment [3].

The selection phase of the genetic algorithm produces a pair of strings (chromosomes) for each offspring that is to be produced for the next generation. Recombination mixes the genetic information of the parents when

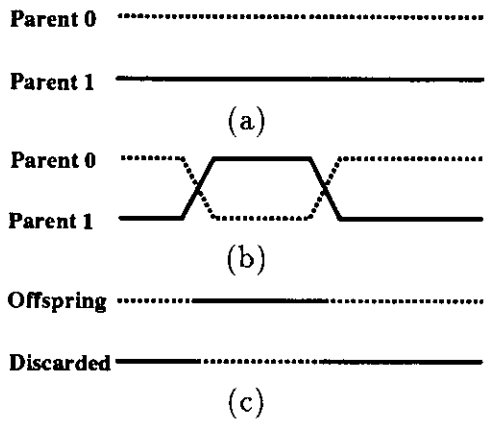


Figure 1: A two-point reciprocal recombination. (a) The parent chromosomes are aligned. (b) At a random point, the chromosomes cross. (c) The chromosomes are cut and rejoined at the crossover point, resulting in new gene combinations. One of the chromosomes specifies the offspring, and the other is discarded.

producing offspring, so an offspring chromosome contains some of the genetic information from each parent. We only consider *reciprocal recombination*, where equivalent length strings are exchanged (Figure 1). The model of recombination that we use in our genetic algorithm begins with an alignment of the pair of chromosomes (Figure 1a). At some random point (or points), the chromosomes cross (Figure 1b), then the chromosomes are cut and rejoined at the crossover point(s) (Figure 1c).

Our model of crossover is not typical of most genetic algorithms because it operates on the chromosome as a bit string, rather than, for example, a list of parameters. It is defined completely independently of what or how the genetic information is encoded in the chromosome, which is biologically realistic.

Our recombination operation produces two haploid chromosomes. One of these (chosen randomly) is discarded, and the other is retained for use as the offspring chromosome. In practice, we only explicitly generate one of the two chromosomes.

After the process of recombination, the genetic algorithm mutates the new chromosome, producing the final version that describes the offspring. Many classes of mutation appear in natural genetic systems, including base

substitution, deletion, frame–shift, insertion, inversion, translocation, duplication, etc. Although most of these types of mutation can make sense in the context of a genetic algorithm, we only consider base substitutions, i.e. the substitution of one nucleotide for another. We simulate a mutation by flipping a bit (change 0 to 1, or 1 to 0) in the bit string chromosome. Like the recombination operation, this formulation of mutation differs from most genetic algorithm implementations in that the mutations make small changes in a structureless bit string, rather than making small changes to a problem–specific parameter.

2.2 Computational Issues

Microanalytic evolutionary experiments are computationally large in many dimensions (including population size, number of generations, size of the genome, size of the behavior function, size of the sensory/effector/memory apparatus, size of the environment, etc.). Until recently, these experiments were not computationally feasible, and even today parallel computation is required for AntFarm experiments.

The panmictic selection and mating scheme of typical genetic algorithms is not very well suited for a massively parallel implementation, because the survival and mating success of each individual involves global knowledge of the population (Equation 1). The local competition/mating scheme that our genetic algorithm is fully distributed, requiring only local information, is both biologically more realistic and well–suited for a massively parallel implementation.

3 AntFarm

AntFarm is a microanalytic simulation that evolves group foraging behavior in colonies of ant–like organisms. The AntFarm evolution is driven by the genetic algorithm described above, operating at the level of colonies (superorganisms), not individual ants. The actions (determined by the ant's behavior function) of all of the ants in a colony contribute to its fitness. Each colony has a single chromosome that codes for the behavior functions of all of its ants (all members of a colony are identical, although each ant receives different sensory input, so they behave differently). Fitness is based primarily on

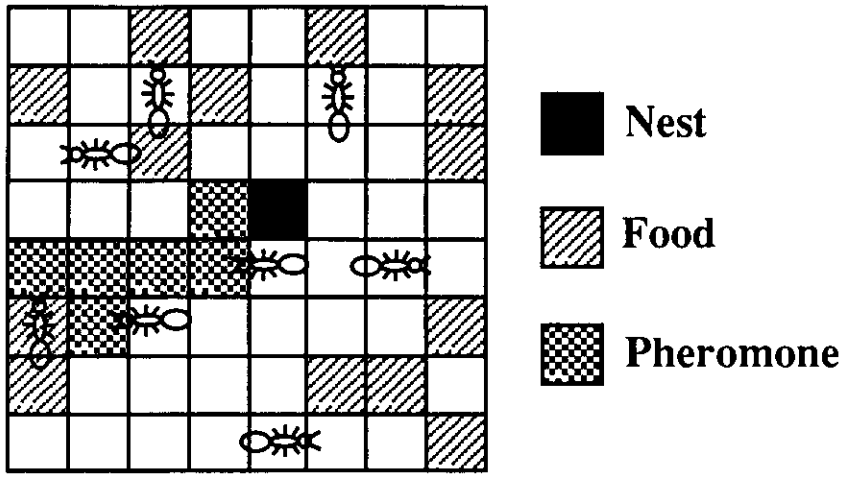


Figure 2: The AntFarm environment contains a nest, food, pheromone, and ants. At the beginning of each generation, all the ants are in the nest, food is distributed in the environment, and not pheromones are present. The actual environment is 16 x 16.

the number of pieces of food carried into the nest, so better foraging means a higher score and greater reproductive success, causing selection pressure for better central place foraging strategies. The initial population consists of randomly generated chromosomes.

AntFarm evolves a population of 16,384 colonies, with 128 ants per colony, for a total of more than two million ants. Each colony lives in its own separate 16 x 16 grid environment, where each location contains some number of ants along with information about the presence or absence of a nest, the amount of food, and the amount of the pheromone ("odor") at that location (Figure 2). Any pheromones that are dropped by the ants slowly diffuse and eventually disappear. The nest of each colony is located at the center of its environment, and the colony's genetic information is represented by a 25,590 bit haploid chromosome.

Each generation begins with each ant in its nest and its memory initialized to zero. All ants live throughout the entire generation. A score is calculated for each colony based primarily on the amount of food deposited in the colony's nest in 100 time steps, although the "metabolic" costs of ant movement, pheromone production, etc. are also taken into account. Each

unit of food is worth 1000 points, each unit of pheromone dropped by an ant costs 0.1, and each other action (move or pickup/drop food) costs 0.1. The inclusion of metabolism in the score results in selection pressure towards more streamlined foraging strategies. During reproduction, both crossovers and mutations occur at a rate of about 0.0001 per bit (about 2.6 mutations and crossovers per colony each generation), which has been shown empirically to be satisfactory.

At the beginning of each generation, the environment is reinitialized so that no pheromone is present and food is placed in a new configuration from a fixed probability distribution. The food pattern seen by each colony in a single generation is identical so that no colony has a chance advantage.

In each of the 100 time steps, the ant's sensory inputs (and 21 bits of internal memory) are processed by its *behavior function* (represented as an ANN), producing a set of actions to perform (Figure 3). An ant has a 3 x 3 sensory array centered on its current location that can sense

- the presence of food,
- the presence of a nest, and
- the amount of pheromone.

In addition, each ant can sense

- whether or not it is carrying food,
- the correct direction to its nest (a *compass* sensor), and
- 4 bits of random input.

In any time step, an ant can decided to do any or all of the following

- move to any of the eight neighboring locations,
- pick up a unit of food (although it can carry a maximum of one unit of food),
- drop a unit of food, and
- drop from 0 to 64 units of the pheromone.

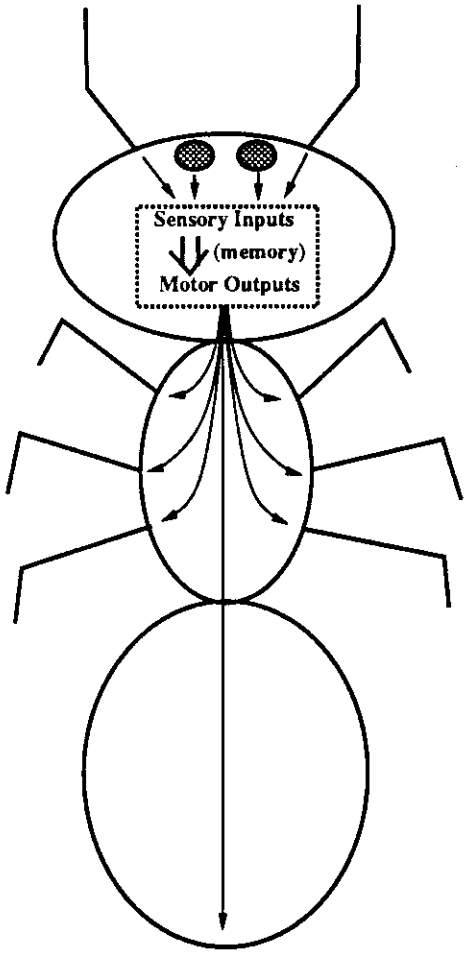


Figure 3: The internals of an AntFarm ant. The box represents the behavior function which is computed by an ANN. The behavior function receives sensory inputs and its internal state (memory), and produces motor outputs. Among other things, the motor outputs control locomotion and the production of the pheromone (from the tip of the abdomen, although in AntFarm we do not simulate the morphology of the ant).

We chose not to try to evolve both foraging search strategies and strategies for navigating back to the nest. Real ants typically use elaborate techniques for navigation [12], often involving memorizing landmarks, calculating average angle of the sun during foraging, etc. We provide the ants with a special sense organ (the "compass") that performs the task of navigation, although the ants still must evolve behavior to interpret and use the compass correctly.

The AntFarm simulation is implemented on a Connection Machine [10], a massively parallel supercomputer, consisting of up to 65,536 1–bit processing elements. AntFarm is written in C++ [18] and uses the CM++ [1] interface to the Connection Machine.

3.1 Comparison of AntFarm to Genesys/Tracker

AntFarm is a direct descendent of the Tracker task studied on the Genesys system [13]. Genesys/Tracker is also a massively parallel microanalytic evolutionary simulation, evolving simple organisms that can follow a noisy, broken trail. The behavior of the Genesys organisms is produced by either an FSA or a 3–layer fully connected recurrent ANN.

The main differences between Genesys/Tracker and AntFarm are a result of the biologically motivated task (central place foraging) of AntFarm. Since AntFarm is trying to model natural evolution, it is implemented with a more realistic genetic algorithm (local competition and mating, rather than global competition and random mating). In addition, the simulated organisms are more complex in many dimensions (summarized in Table 4).

3.2 Representation of the Behavior Function

The ANN organism representation that was used in Genesys [13] encodes the network as the concatenation of the binary integer weight (connection strength) values. The strength of each connection is under genetic control, but not the connectivity pattern itself. The connectivity of the network is statically defined, and the weight values are placed in the bit string chromosome in a canonical order.

For reasons described in Section 4, we have departed from the ANN encoding used in Genesys, and we have designed a new way to encode an ANN that places the connectivity pattern of the network under genetic control [2]. Our new encoding consists of K *connection descriptors*; each consists of three

Dimension	AntFarm	Genesys/Tracker
Population	16,384 colonies 2,097,152 ants	65,536 ants
Info/Environment Location	32 bits	1 bit
Level of Selection	Colony	Individual
Sensory Input/Time Step	$\sim$ 200 bits	1 bit
Effector Outputs/Time Step	13 bits	2 bits
Internal Memory (max)	21 bits	5 bits
Genome Size	25,590 bits	450 bits

Figure 4: A comparison of AntFarm to Genesys/Tracker. The AntFarm simulation is larger and more complex in many dimensions.

parts: the indices of the units that are to be connected (*From* unit, *To* unit) and the weight (strength) of the connection (Figure 5). Certain units are designated as inputs and outputs, and the rest are hidden units, which can serve as memory for the organism. The genotype is the concatenation of the bit representation of the K connection descriptors (as two's complement binary integers).

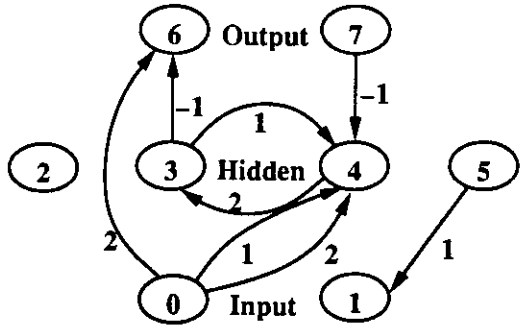
To convert a set of inputs to a set of outputs (behavior), we transmit one signal across each connection in the network. This consists of adding the product of the *From* unit activation and the weight to the *To* unit accumulator. After all K signals have been transmitted, each accumulator is converted to a Boolean value (positive sums to 1; negative or zero sums to 0) and assigned to the corresponding activation. The output unit activations specify the chosen behavior, and the hidden unit activations describe the memory state of the organism.

All possible connection descriptors are legal, including recurrent connections and multiple connections between pairs of units. Connections leading *From* an output unit or *To* an input unit have no effect on the output of the ANN.

The use of connection descriptors gives this encoding some interesting properties. A mutation might change the value of a particular connection weight, or it can move a connection within the network. A crossover can result in the appearance of a connection that neither parent possesses.

The most important property introduced by this new ANN encoding is

From	To	Weight
5	1	1
3	4	1
0	6	2
7	4	-1
0	4	2
4	3	2
3	6	-1
0	4	1



Genotype

10100100101110001100011001011110011100010001010001101001111011100010000 1

Figure 5: The connection descriptors (left), the network (right) and the genotype (bottom) of an ANN encoded with connection descriptors. Each descriptor specifies the pair of units that it connects (*From* and *To* columns), and the strength (*Weight*) of the connection (each of these three fields is 3 bits wide in this example). Note that some units have no connections associated with them, some have no out-going connections, some pairs of units are connected by multiple connections, and recurrent connections are allowed.

the unconstrained and heritable connectivity pattern in the ANN. This freedom is achieved by placing the location and strength of connections under the control of evolution. Another potentially important property of this representation is the position independence of the connection descriptors, which means that a connection descriptor has the same effect no matter where it lies on the chromosome. This allows linkage patterns between functionally related units to evolve. Organisms built with this type of network are competitive with human–designed neural architectures that possess many more connections (see Section 4). Our current encoding is limited in that the number of neurons and connections are not under genetic control.

Here are the details of the AntFarm ANN behavior function. These are features that are available, but particular organisms may "use" (have connected) many fewer:

- Input Units
  - 9 units for pheromone density
  - 9 binary units for presence of food
  - 9 binary units for presence of a nest
  - 4 binary units for compass (an optimal path to the nest)
  - 4 binary units for random noise
  - 1 binary unit for whether or not it is carrying food
- Hidden Units
  - 21 binary units for memory
- Output Units
  - 4 binary units for direction to move
  - 1 binary unit to pick up food
  - 1 binary unit to drop food
  - 1 unit to indicate number of units of pheromone to drop

The whole neural network consists of 64 neural units and 1709 connections. The connection weights are encoded in 3 bits and the *From* and *To* each in 6 bits, so the network is specified by 25,590 bits of genome.

4 Representational Issues

One of the most difficult problems we have encountered thus far has been the search for an appropriate artificial organism representation. Although many organism–based evolutionary simulations have been run, most of the problems and models have been very simple. We encountered serious problems when we attempted to scale the representations to the complexity of the AntFarm organisms.

The representation of an artificial organism in a microanalytic simulation consists of the following parts:

- genotype: a bit string that encodes the behavior function;
- development function: the mapping that decodes the genotype to produce the behavior function;
- behavior function: the program that maps sensory inputs and the memory state into a new memory state and effector outputs;
- interpreter: used to execute organism behavior functions.

In AntFarm, the development function and interpreter are fixed for all organisms and for all time; they are not subject to evolution. The genotype, of course, differs from animal to animal, but is static throughout the life of the organism. The behavior function also does not change during the life of an ant; there is no "learning" protocol: the weights and connectivity are static. Complex, history–sensitive behavior can be realized through the use of the 21 bits of internal memory (over 2 million possible memory states), especially in conjunction with feed–back connections in the ANN.

We have surveyed a variety of animal representations that have been used in simple evolutionary simulations (e.g. parameterized functions [22], Lisp S–expressions [16], finite state automata [13], rule systems (e.g. classifier systems) [9], etc.). Unfortunately, none of these representations is appropriate for simulated evolution with the environment/organism complexity of AntFarm [3]. Each of them either scales exponentially in size with the number sensors/effectors (and thus require too much computer memory for use with AntFarm), or inherently requires too much knowledge specific to the artificial world/task to be built in. The inclusion of task–specific information

in the organism representation opens the door for systematic biases in our evolutionary experiments, so these representation schemes must be avoided.

The most promising representation that we have examined is based on the ANN programming paradigm: ANNs grow slowly as the number of inputs and outputs increase, their internal computations are simple and fast, they are easily encoded in a bit string, and mutations and crossovers in this bit string representation usually cause little or no change in the function that is computed.

4.1 Artificial Neural Networks

We began our work with AntFarm using an ANN organism representation with fully connected layers and recurrent connections, like we used in Genesys [13]. With this representation, we were unable to get even simple non–cooperative foraging to evolve. We then tried multi–layer feed forward ANN networks, and again we failed to evolve foraging behavior.

Our next step was to construct an ANN encoding with as much knowledge of the foraging problem as necessary to get the evolution of foraging behavior. Our aim was to understand why the ANN representations we had used successfully in simpler problems failed in AntFarm.

The foraging task is made up of two separate sub–tasks: searching for food, and returning the food to the nest. The ant can determine which sub–task to perform based on the "carrying food" binary input. Each of these two sub–tasks are separately rather simple. While searching for food, an ant should pay attention to the food sensors, maybe the pheromone sensors (if cooperative foraging is used), maybe the compass and nest sensors (it might want to move away from the nest area), and maybe the random sensors (so a pseudo–random search can be used). While transporting food to the nest, the most important sensor is the compass input; all others can be ignored.

To apply this knowledge of the dual nature of the foraging task, we constructed an ANN behavior function that consists of two fully connected, recurrent networks. One of these networks is invoked when the ant is not carrying food (search), and the other is invoked when the ant is carrying food (transport). We found that ants with behavior functions based on this dual–ANN encoding quickly and consistently evolve (non–cooperative) foraging behavior. This suggests that the problem with the other ANN encodings was that they have difficulty evolving discrete behavior (where a small

change in the inputs leads to a large change in behavior). These representations "generalize," so small changes in the inputs are smoothed away, making the evolution of discrete behavior unlikely.

Although we were able to evolve foraging behavior, we still had a serious problem: the dual–ANN representation requires a huge amount of task–specific information. This could bias the evolutionary outcomes of our experiments in subtle (or obvious) ways, which is unacceptable.

To avoid this problem, we have designed an encoding scheme based on *connection descriptors* (described in Section 3.2), which we have adopted for use in AntFarm. This decision is based on the fact that the connection descriptor encoding does not allow or require knowledge of the task, and an empirical study (presented below) that shows it is able to evolve foraging behaviors that are as successful as that produced by the human–designed dual–network behavior function.

We have empirically compared four ANN–based behavior functions in AntFarm: a network specified by connection descriptors (ANN1), a three layer recurrent network (ANN2), a feed–forward network (ANN3), and a behavior function made up of two recurrent networks (ANN4). ANN4 invokes one of the networks when the ant is carrying food and the other when it is not. Comparison with ANN4 allows us to see how well the other representations are able to evolve behavior for two different tasks in one network. In all four behavior functions, all weights are encoded as 3–bit signed integers and all initial activations of the hidden units are initialized to 0 at the beginning of the generation. The connectivity of the ANN2, ANN3, and ANN4 networks is shown in Figure 6. Table 1 summarizes the main parameters of the ANNs. In an attempt to make the comparison fair, we made the different networks approximately the same size (although ANN1 requires more bits to encode it, but it also has far few connections).

To perform this study, we set the AntFarm parameters as follows. The population consists of 16,384 colonies, each of which contains four identical ants. Each colony forages in its own 16 x 16 environment. The initial food distribution for each colony in each generation is always the same: one unit of food in each location, except for locations on a straight (horizontal, vertical, or diagonal) line with the nest (for a total of 196 units of food). We chose this food distribution because foraging that only requires walking in a cardinal direction from the nest would involve only a few neurons. Each run is 500 generations long, each lasting 50 time steps. Both the mutation and

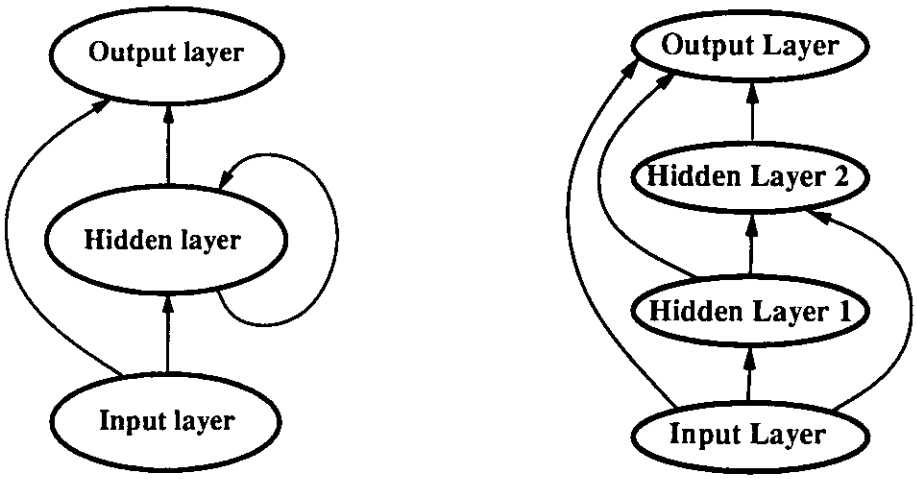


Figure 6: The architecture of the networks in ANN2, ANN3, and ANN4. Each arc indicates that the layers are fully connected. ANN2 and ANN4 (left) have a fully recurrent hidden layer. In ANN3, each layer is fully connected to all "forward" layers. ANN3 has five hidden layers (although only 2 are shown here).

ANN	Number of Connections	Hidden Layers x units/layer	Bits of Memory	Chromosome length (bits)
ANN1	682	varies	21	10240
ANN2	2652	1x32	32	7956
ANN3	2612	5x8	0	7836
ANN4	2×1325=2650	2x18	18	7950

Table 1: A summary of the ANN behavior functions, including the number of connections, arrangement of hidden units into layers (layers x units/layer), the number of bits of memory, and the number of bits in the chromosome.

BM	Search/Transport	Mean	Max
1	random/random	1.07	6
2	random/compass	15.07	21
3	random+food/compass	20.82	25

Table 2: The foraging task is treated as two separate tasks: search for food and transport of the food back to the nest. Random indicates that only the random inputs are used, compass indicates that the inputs pointing the way to the nest are used, and food indicates that the food sensors are used. Mean and Max refer to the amount of food recovered in the population of 16,384 colonies.

recombination rates are set at 0.0001 per bit.

How can we tell how well a population is foraging? It is clearly impossible for four ants to carry all 196 units of food in the environment to the nest in only 50 time steps, so how much food can we expect them to recover? We have hand-coded three simple behavior functions (in C++) to serve as foraging benchmarks (Table 2). BM1 forages (both the search phase and recovery phase) using only a random walk. BM2 searches for food with a random walk (ignoring the food sensors) and carries food to the nest by following the compass (the input that always points to the nest). BM3 improves on BM2 by using the the food sensors while searching. These benchmarks provide an absolute measure of foraging efficiency.

The results are summarized in Figure 7. The dual-network representation (ANN4) out-performs all of the other representations, foraging nearly as effectively as an algorithm that uses the food and compass sensors perfectly. It is not surprising that ANN4 performs the best, since it has a great deal of information about the task built into the representation. The ANN representation based on connection descriptors (ANN1, a scaled down version of the encoding used in AntFarm) is nearly as successful, even though it has been provided with no task-specific information. The other two representations with no built-in knowledge perform rather badly—they are not good at changing their behavior based on whether or not they are carrying food, although the recurrent network (ANN2) does better than the feed forward network (ANN3).

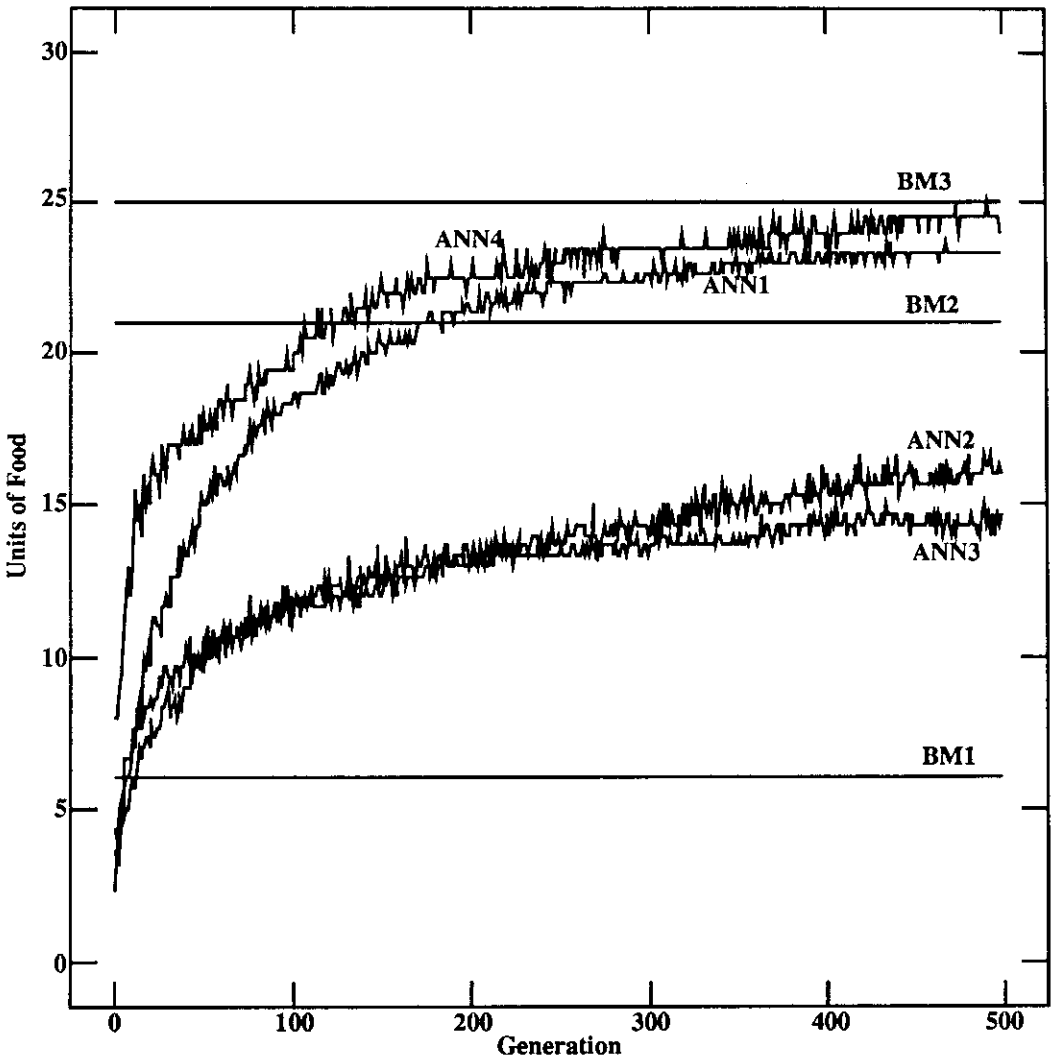


Figure 7: The maximum units of food brought back to the nest in a population of 16,384 colonies across 500 generations. Each curve is the average of three runs, differing only in the initial random seed. These simulations required 8 days of Connection Machine computation (on 8K processors).

4.2 Properties of Representations

From our exploration of potential artificial organism representations, we have abstracted a number of properties that we believe are necessary for simulated evolution experiments:

1. (approximate) *closure* of the set of legal genotypes under the action genetic operators;

2. *smoothness* of the phenotype under the action genetic operators, i.e. the behavior function should tend to change smoothly as the genotype is changed by mutation and recombination;

3. the ability to *scale* to large behavior functions, i.e. those that can handle large amounts of input and output data without undue increase in genome size;

4. the ability to *evolve* phenotypes that exhibit *both continuous and discrete behaviors* as a function of their inputs; and

5. a *uniform computational model*, i.e. the programming paradigm in which the behavior functions are expressed should not contain features that include any kind of explicit or implicit knowledge of the environment, nor bias toward a particular evolutionary trajectory.

Closure (1) and smoothness (2) are properties of the development function and the genetic operators. Scaling (3), the ability to evolve continuous and discrete behavior (4), and uniformity (5) are all properties of the computational model of the behavior function.

The property of syntactic closure constrains the development function (the encoding of the behavior function into the genotype). To be syntactically closed (approximately), the genetic operators must always (or almost always) produce genomes that translate into syntactically legal behavior functions. An evolutionary system will not work if a single mutation or recombination is likely to transform a genotype that encodes a perfectly good behavior function program into a program that is syntactically illegal.

The smoothness property requires that most changes to the genotype due to the application of genetic operators result in small changes in the behavior function. For example, a mutation in an ant should usually have

a small affect on its foraging algorithm. Of course, it need not *always* cause a small change; some small changes will be fatal, and a few small changes may cause profound but beneficial effects. Still, evolution cannot work if the phenotype space is not relatively smooth as a function of genotype. The encoding and mapping functions should be smooth not only under mutation, but also under recombination, implying that functionally related genes should usually be inherited as a unit (strongly linked). The smoothness property has the effect of requiring the "adaptive landscape" [26, 15] to be correlated with respect to the genetic operators. Evolution can successfully searches the space of possible organisms only in correlated adaptive landscapes.

Smoothness is an extension of the closure property. Not only is it required that a legal program be (usually) transformed into another legal program by the genetic operators, but also that it is (usually) transformed into one that is semantically similar to the original.

The scaling properties of a representation are of extreme practical importance because we must be able to store large populations in a reasonable amount of computer memory. Scaling refers to the rate at which the size of the representation grows as a function of the number of inputs, outputs, and bits of internal memory. The size of the representation includes the number of bits in the genotype, the number of bits required to store the decoded behavior function, the amount of time to translate from the genotype to the behavior function, and the amount of time to run a set of inputs through the behavior function to produce the outputs. We are interested in organisms with dozens or hundreds of inputs, outputs, and bits of internal memory.

In AntFarm, foraging requires a combination of both continuous and discrete behaviors (which is probably necessary in any simulation that hopes to evolve complex behavior). Roughly speaking, a behavior function is producing continuous behavior when a small change in the inputs to the function results in a small change in the outputs, and discrete behavior when a small change in the inputs results in a large change in the outputs. In AntFarm, the foraging behaviors that we have evolved consist of two modes: search and transport. Within each mode, the behavior appears to be continuous, but the transition from one mode to the other is determined by the "carrying food" input bit (a very discrete change in behavior). Most or perhaps all complex behaviors will involve the combination of different modes of behavior, and thus require the evolution of discrete behavior.

The final property is that the computational model of the representation should be uniform. In particular, it must be able to describe all desired behavior functions *without designing features of the problem or possible solutions into the representation.* If we are trying to shed light on a biologically motivated hypothesis, the results will be invalid if we bias the organisms toward (or away from) some evolutionary path. The computational model must not have knowledge of the problem or possible solutions embedded in it.

5 Future Work

The AntFarm project is work in progress. We have designed a genetic algorithm that closely models real evolution, designed an appropriate behavior function, found empirically good values for various parameters (crossover and mutation rates, etc.), and evolved colonies that successfully forage for food.

So far, we have not observed the evolution of cooperative foraging. In the AntFarm model, the use of pheromone trails to lead other workers to a large pile of food is quite complex. Because pheromone trails are nondirectional, following a pheromone trail involves the combined information of the pheromone and compass inputs. Simply walking uphill in the pheromone density does not work, because the pheromone trail will be faintest nearest to the food, due to diffusion. A reasonable trail-following strategy is to move to the location that is furthest from the nest and contains some of the pheromone. This will keep the ant on the trail and moving toward the food source.

Although the model of Johnson et al. indicates under what circumstances cooperation is the optimal strategy, we do not know under what circumstances cooperative strategies will *evolve.* It may be that cooperative foraging is unlikely to evolve in any static environment. We may have to vary the environment, slowly making foraging more difficult.

We plan to perform a systematic study of the effect of food distribution on the evolution of foraging strategies, testing the model of Johnson et al. [14]. It will be interesting to see how our artificial evolution differs from biological theory. We are interested in exactly how and when the pheromones are used to communicate information about the food distribution.

We are also interested in the evolution of foraging strategies that are

strongly affected by competition. We might find strategies that utilize exploitation or direct interference. A possible strategy for better exploitation of resources under stiff competition would be to forage further from the nest first, beating the neighboring colonies to that food, resulting in a larger foraging area for the colony. Interference strategies might involve disrupting communication of neighboring colonies, either by overwriting existing pheromone trails, or by laying misleading trails. In order to investigate this area, we will run AntFarm in a mode where a single large environment is shared by all colonies during foraging.

Acknowledgments

We owe thanks to many people who have contributed to ideas and comments to AntFarm, especially Doyne Farmer and Chris Langton. We also acknowledge the contributions of Michael Dyer, Don Feener, Danny Hillis, Andrew Kahng, Adam King, John Lighton, Joe Pemberton, Chuck Taylor, and Greg Werner. This work is supported in part by W. M. Keck Foundation grant number W880615, University of California Los Alamos National Laboratory award number CNLS/89-427, and University of California Los Alamos National Laboratory award number UC-90-4-A-88.

References

[1] Robert J. Collins. CM++: A C++ interface to the Connection Machine. In *Proceedings of the Symposium on Object Oriented Programming Emphasizing Practical Applications*, September 1990.

[2] Robert J. Collins and David R. Jefferson. An artificial neural representation for artificial organisms. In Reinhard Männer and David E. Goldberg, editors, *Proceedings of Parallel Problem Solving from Nature*. Springer–Verlag, (in press).

[3] Robert J. Collins and David R. Jefferson. Representations for artificial organisms. In Jean-Arcady Meyer and Stewart Wilson, editors, *Proceedings of Simulation of Adaptive Behavior*. The MIT Press/Bradford Books, (in press).

[4] Robert N. Coulson, Joseph Folse, and Douglas K. Loh. Artificial Intelligence and natural resource management. *Science*, 237:262–267, July 1987.

[5] James F. Crow. *Basic Concepts in Population, Quantitative, and Evolutionary Genetics.* W. H. Freeman and Company, New York, 1986.

[6] Jennifer H. Fewell. Energetic and time costs of foraging in harvester ants, *pogonomyrmex occidentalis. Behav. Ecol. Sociobiol.*, 22:401–408, 1988.

[7] John Fry, Charles E. Taylor, and U. Devgan. An expert system for mosquito control in Orange County California. *Bulletin of the Society of Vector Ecology*, 14(2):237–246, 1989.

[8] J. S. L. Gilmour and J. W. Gregor. Demes: A suggested new terminology. *Nature*, 144:333, 1939.

[9] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison–Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[10] W. Daniel Hillis. *The Connection Machine.* The MIT Press, Cambridge, Massachusetts, 1985.

[11] John H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

[12] Bert Hölldobler and Edward O. Wilson. *The Ants.* Harvard University Press, 1990.

[13] David Jefferson, Robert Collins, Claus Cooper, Michael Dyer, Margot Flowers, Richard Korf, Charles Taylor, and Alan Wang. The Genesys System: Evolution as a theme in artificial life. In Christopher Langton, J. Doyne Farmer, Steen Rasmussen, and Charles Taylor, editors, *Artificial Life II.* Addison–Wesley Publishing Company, (in press).

[14] Leslie K. Johnson, Stephen P. Hubbell, and Donald H. Feener, Jr. Defense of food supply by eusocial colonies. *Amer. Zool.*, 27:347–358, 1987.

[15] Stuart Kauffman and Simon Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128:11–45, 1987.

[16] John R. Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical report, Department of Computer Science, Stanford University, 1990.

[17] John R. B. Lighton. Energetics of foraging and recruitment in the giant tropical ant *paraponera clavata* (hymenoptera: Formicidae). (unpublished manuscript).

[18] Bjarne Stroustrup. *The C++ Programming Language.* Addison–Wesley Publishing Company, 1986.

[19] John H. Sudd and Nigel R Franks. *The Behavioural Ecology of Ants.* Chapman & Hall, New York, 1987.

[20] Gordon L. Swartzman and Stephen P. Kaluzny. *Ecological Simulation Primer.* Macmillan Publishing Company, 1987.

[21] Charles E. Taylor. Evolution of resistance to insecticides: The role of mathematical models and computer simulations. In George P. Georghiou and Tetsuo Saito, editors, *Pest Resistance to Pesticides.* Plenum Press, 1983.

[22] Charles E. Taylor, David R. Jefferson, Scott R. Turner, and Seth R. Goldman. RAM: Artificial life for the exploration of complex biological systems. In Christopher G. Langton, editor, *Artificial Life*, pages 275–295. Santa Fe Institute, Addison–Wesley Publishing Company, Inc., 1989.

[23] Charles E. Taylor, L. Muscatine, and David R. Jefferson. Maintenance and breakdown of the *hydra–chlorella* symbiosis: A computer model. *Proceedings of the Royal Society of London*, 238:277–289, 1989.

[24] Gregory M. Werner and Michael G. Dyer. Evolution of communication in artificial organisms. In Christopher Langton, J. Doyne Farmer, Steen Rasmussen, and Charles Taylor, editors, *Artificial Life II.* Addison–Wesley Publishing Company, (in press).

[25] Sewall Wright. Evolution in Mendelian populations. *Genetics*, 16:97–159, 1931.

[26] Sewall Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the Sixth International Congress of Genetics*, volume 1, pages 356–366, 1932.

[27] Sewall Wright. *Evolution and the Genetics of Populations. Volume 1: Genetic and Biometric Foundations.* University of Chicago Press, 1968.

[28] Sewall Wright. *Evolution and the Genetics of Populations. Volume 2: The Theory of Gene Frequencies.* University of Chicago Press, 1969.

[29] Sewall Wright. *Evolution and the Genetics of Populations. Volume 3: Experimental Results and Evolutionary Deductions.* University of Chicago Press, 1977.

[30] Sewall Wright. *Evolution and the Genetics of Populations. Volume 4: Variability Within and Among Natural Populations.* University of Chicago Press, 1978.