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

# EVOLUTION AS A THEME IN ARTIFICIAL LIFE: THE GENESYS/TRACKER SYSTEM

David Jefferson
Robert Collins
Claus Cooper
Michael Dyer
Margot Flowers
Richard Korf
Charles Taylor
Alan Wang

December 1990
CSD-900047

# Evolution as a Theme in Artificial Life: The Genesys/Tracker System

David Jefferson, Robert Collins, Claus Cooper,
Michael Dyer, Margot Flowers, Richard Korf,
Charles Taylor, & Alan Wang

November 1990

Technical Report UCLA-AI-90-09

# Evolution as a Theme in Artificial Life: The Genesys/Tracker System[*]

David Jefferson, Robert Collins, Claus Cooper
Michael Dyer, Margot Flowers, Richard Korf
Charles Taylor, Alan Wang

UCLA
Los Angeles, California 90024

## Abstract

Direct, fine-grained simulation is a promising way of investigating and modeling natural evolution. In this paper we show how we can model a population of organisms as a population of computer programs, and how the evolutionarily significant activities of organisms (birth, interaction with the environment, migration, sexual reproduction with genetic mutation and recombination, and death) can all be represented by corresponding operations on programs. We illustrate these ideas in a system built for the Connection Machine called Genesys/Tracker, in which artificial "ants" evolve the ability to perform a complex task. In less than 100 generations a population of 64K "random" ants, represented either as finite state automata or as artificial neural nets, evolve the ability to traverse a winding broken "trail" in a rectilinear grid environment. Throughout this study we pay special attention to methodological issues, such as the avoidance of representational artifacts, and to biological verisimilitude.

## 1. Introduction

One of the major research themes of the UCLA Artificial Life group is the simulation of biological evolution, especially the exploration of its large scale behavior (macroevolution). Our simulations do not use differential equations or aggregated models, but are instead direct, fine-grained, "microanalytic" simulations of a large population in a complex environment over many generations. We believe that an organism can be simulated in a natural way by a computer program; just as an organism is born, moves, interacts with its environment, processes information, reproduces with variation, and dies, so too can a program initiate, migrate, interact with its environment, process information, produce modified copies of itself, and terminate. In our simulations we represent each individual organism separately as a computer program of some kind whose execution represents the sequence of significant events in the organism's life from birth to death. Each organism has a genotype (a bit string) and a phenotype (a program encoded in the bit string), both of which are generally unique in the population. Over many generations of replication, competition, and selection, we expect novel and superior forms to evolve.

Our long-term goal is to study natural evolution through artificial evolution. Today, biologists have only a few ways to study macroevolution: (a) the mathematics of population genetics,

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

(b) laboratory and field experiment, (c) examination of molecular relationships among modern species, and (d) examination of the fossil record. Each method has important limitations. Population genetics today usually cannot give sharp nonequilibrium results for systems with a large number of genes. Experiments are usually confined to small populations for a few generations (except for microorganisms), and usually cannot be perfectly controlled. The fossil record is notoriously biased and incomplete, containing primarily morphological information on those creatures whose life style and body type made them likely to be preserved. And, both molecular and fossil studies tell us about how natural evolution proceeded historically, as it was, but not so much about the principles of evolution as it might have been. We hope to add to this list another intellectual tool for the study of evolution, computer simulation. Just as viruses and bacteria can serve as models of the molecular biology of eucaryotic organisms, we believe that evolving populations of computer programs can act as simple, controllable, replicable models for large-scale evolutionary processes. We hope some day that biologists may use simulation to help resolve some of the outstanding foundational problems in evolution, including perhaps questions about modes of speciation, the evolution of cooperation, the unit(s) of selection, and the evolution of sex.

This paper has two parts. The first part is methodological; it sets forth our view of how evolution can profitably be studied by computer in new ways and with new kinds of models. We describe the strengths, and some of the limitations as well, of our paradigm.

The second part describes a particular system, called the Genesys, that we built to test the computational feasibility of this approach to studying evolution. We show how we have been able to evolve artificial "ants" that are capable of complex behavior, e.g. following a "broken trail" in a grid environment. Genesys is not *per se* an attempt to resolve any biological question. It is rather an ambitious exercise exploring computational and theoretical issues about biologically realistic evolution. Our experience with the most important of these issues is described below.

## 2. Methodological issues in artificial evolution

In attempting to mimic natural evolution many fundamental methodological questions arise. Just what is an organism, and how should it be represented computationally? How do genetic algorithms differ from biologically-motivated evolution? How can we be sure, when we purport to create something by evolution, that we have not somehow guided or forced that process by design choices or initial conditions? In this section discuss these questions, prior to describing the Genesys system in detail.

### *Can artificial evolution be open ended?*

In (Taylor, Jefferson, Turner and Goldman 1989) we reported on an earlier evolution simulator, called RAM in which each organism in the population is represented as a pair containing (a) a parameterized Lisp function, referred to as the organism's "behavior function", and (b) a sequence of parameter values to the behavior function, which act as the organism's genome. Each individual parameter value is like a "gene", and an organism inherits the parameter values of its parent(s), usually modified by mutation (and possibly recombination). All organisms (of the same "species") share a common behavior function, but differ in the genes.

We consider RAM to be a successful experimental system, and still use it today. (See (Gibson, Taylor, and Jefferson 1990), (Taylor, Muscatine and Jefferson 1989), (Taylor, Jefferson and Burla 1987)). It is definitely capable of illustrating evolution of organisms interacting in a common environment. But we are dissatisfied with some of RAM's limitations. First, it can support only small populations (a few hundred organisms) because of the large amount of memory it requires and because it runs on workstation-class machines; and second, while it is comparatively easy for RAM to simulate asexual reproduction, sexual reproduction and mate selection have to be specially programmed. But one of the most important deficiencies in our view is that the Lisp function that defines the behavior of an organism is not itself subject to mutation or recombination; only the genes (parameters) are. Furthermore, the genes have to be manipulated by mutation operators that are specific to the type and meaning of the gene. An integer gene, for example, might be incremented or decremented, with perhaps a clause preventing it from going negative; or a real-valued gene might be multiplied by a random value near 1.0.

This approach, however, seems ultimately too contrived and too distant from natural genetics. In natural life there are no integers and no reals in the genome. We do not have separate types of genes, each with its own specific type of mutation operator, and the boundaries between codon regions are not at all respected by the natural "genetic operators" of substitution, deletion, insertion, inversion, and crossover. Because in RAM all heritable information is contained in the genes, most of the information determining the structure and behavior of the organism (the behavior function and the Lisp interpreter) is static from generation to generation, and not subject to evolution. Since the modeler specifies in advance both the types of the genes and the mutation operators that apply to them, in effect he specifies a low-dimensional parameter space which is the universe of all genotypes. Evolution is then conducted as a genetic "search" within this space, and most of the structure and behavior of the organism is out of reach of evolution. Mathematically, the confinement of evolution to a small, closed parameter space does not resemble the thousands of degrees of freedom available in the evolution of even the tiniest natural genome. The kind of evolution exemplified by RAM is, unfortunately, not open ended.

In building our new system, Genesys, we wanted to construct a system that corrected these deficiencies. We wanted almost all of the logic of the organism's behavior to be subject to evolution, and we wanted the evolution to be based on genetic algorithms that are biologically defensible, with no knowledge of the environment or the evolutionary "goal" in any way embedded in the choice of genetic operators or any other aspect of the genetic algorithm. These constraints required us to change the representation of organisms and their genomes drastically, away from Lisp functions and typed parameter values, and toward a programming paradigm that would have a *small interpreter* (so that the complexity of an organism's behavior resides more in the program and less in the interpreter) and that would allow *the entire text of the program itself to be subject to evolution.* When representing organisms as programs for evolutionary studies, we believe that an encoded form of the program itself should be the genome.

## *What is an appropriate representation for organisms?*

Given that organisms are to be represented by programs, and that they must be subject to mutation and recombination, what kind of programs should they be? Should they be mathematical automata of some kind? Procedural programs? Rule systems? Logic circuits?

Logic programs? Constraint systems? Neural nets? And how should they be encoded into a genome, and what mutation and recombination operators should they be subject to? Should they be encoded as bit (or character) strings, with mutation and recombination at the bit (or character) level? Or should they perhaps be represented a flattened parse trees with mutation and recombination at the lexeme level? Data elements, such as integers, will inevitably be part of such a program. Should they be represented and mutated as atomic objects, with whole-integer operators, or as bit strings with bit flipping as the mutation operator? If the latter, should they be ones-complement, twos-complement, or perhaps gray code? We refer to all of these questions as the problem of the "representation of organisms", a problem we expect to grow in significance as research in artificial evolution continues.

From all these possibilities we chose two of programming paradigms to work with: finite state automata (FSAs) and recurrent artificial neural nets (ANNs). In both cases we chose to encode the entire "program" (FSA state transition table, or neural net weights) as a bit string and to subject it to bit-level mutation and recombination operations. We will describe them in more detail later in Sections 4 and 5; but here we only note that one of the major questions driving us in this research has been "Which of those two representations, FSAs or ANNs, is 'better' for evolution studies?". Some of us believed that the FSA's were superior, at least for the Tracker task, because that task is essentially finite and sequential, and any deterministic algorithm for it is formally equivalent to some finite automaton. Others of us felt that ANNs were probably superior; ANNs encoded as bit strings would seem to be inherently more robust and well-conditioned under mutation and recombination than the apparently more brittle and less redundant FSAs.

We put a lot of work and debate into trying to answer this question. It turned out that after going to some length to make the two representations as comparable as possible, empirical trials consistently showed that the FSA representation performed slightly better than the ANN representation on the Tracker task. However, we have not come to a full understanding of why this is so, and we do not necessarily believe the empirical result generalizes in any meaningful way to other tasks, or to larger organisms than the ones we are working with. We never did answer the question fully; probably neither representation dominates the other in all dimensions. In any case, we now believe it to be a false dichotomy, and not such an important question after all, as we will discuss in Section 11.

Koza has described a fascinating system that takes a different point of view from the one we have adopted (Koza 1990). He executes genetic algorithms over Lisp programs, but with a different purposes and mechanisms than either RAM or Genesys. Like RAM, he uses Lisp programs as the representation of "organisms" (though he does not refer to them as organisms). But unlike RAM, where the genome is a list of parameters, he treats Lisp programs as an S-expressions and uses subexpression-oriented mutation and recombination operators on the text of the program itself. His goal it to engineer programs that can solve problems, and he has been able to essentially duplicate our work (among many other things) by evolving Lisp programs that perform the same Genesys/Tracker task that we will describe here. But the problem-solving flavor of his research, which studies essentially goal-directed evolution, is in contrast to our goal, which is the study of natural evolution, which is fundamentally not goal-directed. As a result, there is no need for him to choose genetic or evolutionary mechanisms that closely resemble natural ones, whereas we feel bound to justify our experimental designs biologically. Likewise, with an essentially engineering, nonbiological point of view, there is no need for him to pay the kind of attention to representational artificact that we do here.

*How can we control for experimenter bias and representational artifacts?*

One of the difficult problems that can arise in evolution studies is the "credit assignment" problem. When an evolutionary experiment succeeds in producing a population of organisms that exhibits a certain interesting behavior (e.g. trail-following), how can we decide the extent to which that behavior may have been built into the architecture of the organism or the structure of the initial population? How do we know that there is not a bias built into the mutation operators chosen, or the programming paradigm that the organisms are expressed in, etc? Has evolution really created something novel?

In Genesys we have taken a number of precautions and used a number of controls to ensure that bias and representational artifact are minimized in our results. First, as we just described, we use two distinct representations, finite automata and artificial neural nets, instead of just one. These two representations are so profoundly different that no artifactual result derived from an experiment with one would be naturally preserved in an experiment with the other. Because we succeed in evolving trail-following organisms in both cases, we feel confident that we are not being fooled by some subtle property of one or the other representation.

We guard against introducing bias through our choice of genetic algorithm and operators by fixing on one genetic algorithm for all experiments, even when we change representation, so that the genetic algorithm cannot be "tuned" to the representation or the experiment. In addition we always use bit strings to represent the genome of an organism, and we use bit-level genetic operators. Mutation is always simulated by inverting random bits, chosen with equal probability from anywhere in the genome, regardless of where the bit happens to fall with respect to the "meaningful" fields (codons) in the programs. Likewise, we permit crossover with equal probability between any two bits in the genome; we make no effort to guarantee that it occurs between two codons. These decisions, besides preventing bias, have the additional advantage of being biologically realistic, since point mutations and crossover can occur anywhere within any region of DNA, without regard to the role it may play in the resulting organism. And, by choosing the bit string and bit-level genetic operators we guarantee that the phenotype of an artificial organisms is a very distant and indirect function of the string of bits in its genotype, just as the phenotype of a natural organism is a very indirect function of the string of nucleotides. In both cases there is no simple way of predicting the change in phenotype that will be caused by a random mutation.

Finally, we further guard against introducing bias through our choice of initial population. We always start with a population of random organisms, i.e. those derived from an initial population of random bit strings, where each bit has an equal probability of being 0 or 1. Of course, all experiments are performed repeatedly with different random seeds. (It has recently been suggested that we might have tried other distributions, i.e. random bit strings that were 99% zeros. That would perhaps have guarded against the possibility that we had somehow "seeded" the population with so much initial variation that evolution proceeded at a faster than normal rate.)

*At what scale can artificial evolution of this kind be made to work?*

Even if the idea of evolving ANNs or FSAs encoded in bit strings is right in principle, we were concerned that it might not work on a computational scale that we could afford. In Genesys applied to the Tracker task, which we will describe shortly, we were attempting to run a genetic algorithm with genomes 450 bits long, and with genetic operators that did not reflect anything at all about the task. To our knowledge, very few genetic algorithm experiments had ever been attempted using such long genomes. Since the size of the space of all organisms grows exponentially with the length of the genome, perhaps an astronomical population size, or millions of generations, would be necessary before anything interesting happened. Furthermore, any genetic algorithm must be tuned with several of parameters (e.g. mutation rate, recombination rate, selection fraction, etc.) If our evolutions had failed, it could have been just because we were unsuccessful in finding the "right" combination of parameters for the scale at which we worked.

## 3. The Tracker task

With all of these methodological questions and considerations in mind we set about to see if we could indeed exhibit the evolution of some kind of complex behavior. Our intention was to see if it is possible to produce artificial organisms that exhibit a relatively complex behavior, and to do so entirely by evolutionary methods, with no built-in design bias, and with no learning. The evolutionary exercise we now describe here was inspired by the behavior of certain species of ants that lay down pheromone trails from a food site to their nest to aid in the process of collective foraging (see e.g. Holldobler and Wilson, 1990). The trails are "noisy", and fade with time as the pheromone odors disperse. This trail-following ability is quite remarkable, and clearly required the evolution of elaborate nervous system structures that are present in modern ants, but were not present in a sufficiently ancient ancestral population.

We designed a highly simplified task called *Tracker* that resembles ant trail-following at least superficially. The Tracker task requires an ant to follow a crooked, broken trail of black cells in a white toroidal grid. The trail has a series of turns, gaps, and jumps that get more difficult as it progresses. The behavior functions of the ants in the initial population were constructed by a random process, so that while they have built-in "eyes" to sense the environment and built-in "motor apparatus" to move around in it, the population clearly has no built-in bias toward the ability to coordinate sensory input with motor output in such a way as to follow trails. However after 100 generations of evolution a large fraction of the population (anywhere from 20% to 80%, depending on the genetic algorithm parameters) is nearly perfect at the task.

Genesys simulates a large population of organisms, each of which is represented as either a finite-state automaton (FSA) or an artificial neural net (ANN). That phenotype (the particular FSA or ANN) is encoded in a genotype represented as a bit string, and the population evolves using a genetic algorithm (Holland 1975) in which the score of the genotype, which determines the frequency of its representation in the next generation, is a function of how well the organism performs in its environment. Genesys runs on a 16K-processor Connection Machine (CM2) with 8K bytes of memory per processor (Hillis 1985), typically with a population of 64K organisms. Execution takes less than 30 seconds per generation.

Our experiments were conducted using the following task. An "ant" is placed in a two-dimensional grid at the beginning of the broken rectilinear trail shown in Figure 1. The grid is toroidal, so that the cells on the left edge are considered to be adjacent to those on the right edge, and the cells on the bottom edge are considered to be adjacent to those at the top. This particular trail, called the John Muir Trail (to distinguish it from other trails we have experimented with) is used throughout this paper. The ant's task is to move from cell to cell, traversing as much of the trail as possible in 200 time steps. Its success is measured by the number of distinct trail cells it traverses. It is free to wander off the trail and pick it up elsewhere, but that is not an efficient strategy; whenever an ant with such a strategy appears during evolution, its lineage presumably dies out.

An ant is considered to erase a cell of the trail as soon as it steps on it. This convention is used because we did not want to give ants "credit" for following the same piece of trail more than once (which would reward backtracking, circling in place, or just no-oping). Instead of incorporating into Tracker an algorithm to count the number of distinct trail cells the ant steps on, we found it computationally simpler just to erase the trail as it progressed.

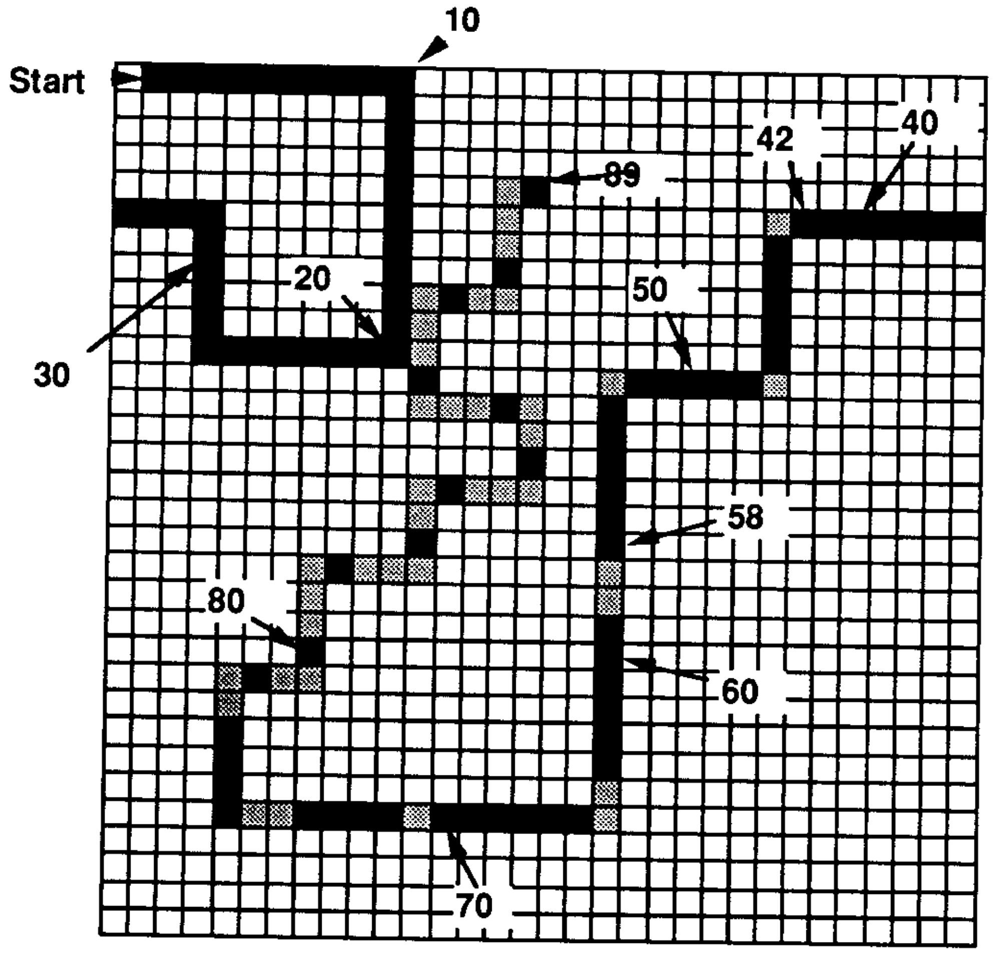


**Figure 1: The John Muir Trail in 32 x 32 toroidal grid. Scores for reaching various landmarks are indicated. (Gray cells are not part of the trail; they are visual aids to the reader to mark the fastest route of traversal, and appear white to the ants.)**

The John Muir Trail winds around the environment, easy at first, but presenting ever harder challenges. First it includes three straightaways ending in right turns, followed by a straightaway ending in a left turn. After a while, starting at trail step 42, it takes several turns where the corner cell is missing from the trail. At step 58 there is a straightaway double gap, and then the trail gets progressively more difficult, ending with a series of disconnected knight's moves and long knight's moves. These last ten steps on the trail are extremely difficult compared to the rest of the trail, because each of them requires at least three or four tempos to get to from the previous trail cell, and usually several more tempos must be used just searching for the continuation of the trail. The maximum possible score is 89, since that is the number of trail cells.

Each ant is in a sense-and-act loop, receiving one bit of sensory input per unit time about where the trail is, and deciding on two bits of action per unit time. At each time step it stands on one cell, facing one of the cardinal directions (N, S, E, W), and it can sense whether or not the cell just *ahead* of it is part of the trail (Figure 2). We designed the ant to sense the cell ahead, instead of the cell it is on, because trail-following strategies with this sensory arrangement are much simpler. After sensing the cell ahead of it, the ant must take one of four actions:

a) move forward one step;
b) turn right (without moving);
c) turn left (without moving); or
d) do nothing.

We chose this menu of moves also for simplicity. The choice to include a no-op was in anticipation that algorithms might more easily evolve for trail-following if it were possible for an ant to change its internal state without changing position or orientation on the grid. However, in practice we found that no-op moves were bred out of the best algorithms. Apparently the tempo that such a move consumes is too valuable to be wasted on an action that does not make progress along the trail, because difference of a single point of the score for an ant is often the difference between having progeny and having its lineage go extinct.

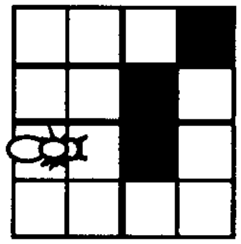


**Figure 2: An ant stands in one cell and sees only the cell ahead of it.**

In order for an ant to traverse the trail it must embody an algorithm that causes it to move forward when it sees trail on the cell ahead, and to search locally for the continuation of the trail when it does not. It is important to understand that the Tracker task does not require general trail-following ability; it only requires the ants to traverse *this particular trail*, starting from the NW corner facing East. Hence, we should not be surprised to see the evolution of ants with features adapted to the quirks of this particular trail.

## 4. The FSA Architecture

Figure 3 shows an example of a Finite State Automaton (FSA) that can perform the Tracker task. The circles represent states, and the arcs represent state transitions the ant undergoes as a function of its current state and its sensory input, each of which takes one time step of the 200 allowed. An arc is labelled with a pair of the form *s*/*a*, where *s* is a 1-bit sensory input indicating whether the ant "sees" a trail cell (1) or a non-trail cell (0); *a* indicates which of four actions should be taken: move forward (M), turn left without moving (L), turn right without moving (R), and do nothing (N). The FSAs employed in Genesys have up to 32 states.

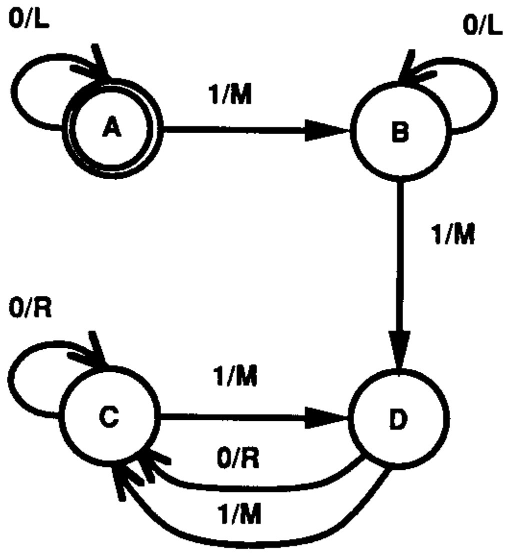


**Figure 3: The state transition diagram for a 4-state FSA that achieves a score of 42 on the Tracker task. State A is the start state.**

This example FSA is achieves a score of 42 on the John Muir Trail, and does so with some rather clumsy logic. On the first straightaway of the trail it walks along in the following sequence of states: ABDCDCDCDC. Thereafter, it never enters states A or B again, but negotiates all turns and straightaways going back and forth from states C to D. Right turns are made in one move; left turns are made using three right turns. At step 42, where there is the first break in the trail, the ant stops, spending the remainder of its life turning right in state C.

In order to be constructed by a genetic algorithm, the FSA must be encoded into a chromosome-like string of bits. The encoding used in Genesys is shown in Figure 4, where the FSA of Figure 3 is arrayed in a table in which each state, input, and move has an arbitrarily-assigned binary value. To define the FSA, it suffices to indicate the initial state (state A in this example, coded as 00 in binary), and to enumerate in canonical order the New State and Move columns of the State Transition Table (STT). The genome that encodes the FSA in Figure 3 is shown at the bottom of Figure 4. It is the concatenation of initial state and, in canonical order, the last two columns of the STT. Since Genesys FSAs have up to 32 states (requiring 5 bits to encode), and there are 64 lines in each STT, there are a total of 64*(5+2)+5 = 453 bits per genome.

| Old State | Input | New State | Action |
|---|---|---|---|
| 00 | 0 | 00 | 01 |
| 00 | 1 | 01 | 11 |
| 01 | 0 | 01 | 01 |
| 01 | 1 | 11 | 11 |
| 10 | 0 | 10 | 10 |
| 10 | 1 | 11 | 11 |
| 11 | 0 | 10 | 10 |
| 11 | 1 | 10 | 11 |

**Genome**

**00 0001 0111 0101 1111 1010 1111 1010 1011**

**Figure 4: The binary-coded State Transition Table (STT) for the FSA in Figure 3. The bit string at the bottom is the genome that encodes this organism. Only the initial state and the last two columns (enclosed in the heavy lines) are part of the genome. The first two columns need not be encoded because they are in canonical order.**

Actual FSAs that arise during evolution are likely, of course, to have fewer than 32 states, since there is no guarantee that all 32 distinct 5-bit state codes are present in the genome string; and even if they are, there is no guarantee that they are reachable from the start state. Some of the rows in the table for a particular FSA may correspond to states it can never enter, and thus represent "unreachable code". An FSA may have even fewer "reachable" states if we consider only those used while traversing the John Muir Trail.

## 5. The ANN architecture

To represent an ant using an ANN we chose a particular recurrent PDP architecture (Rumelhart and McClelland 1986, Elman 1988, Servan-Schreiber 1989) with standard sum-and-threshold logic in the connection topology shown in Figure 5. There are two input units with an activation of 1 or 0 according to whether the cell ahead of the ant is on the trail or not. One is activated when there is trail, and the other is activated when there is not. (This choice is historical; one input unit could have sufficed.)

Each input unit is connected to each of 5 hidden units and to each of 4 output units. The hidden units are fully connected among themselves, and also to each of the 4 output units. In each time step the net receives input from its sensor units, and the activation signals propagate once along every connection in the network. All of the units have Boolean thresholds except the output units. Output units have their inputs summed but not thresholded; instead the ant's next action is determined by which of the output units has the highest activation (with an arbitrary rule for breaking ties). Because this is a recurrent net with 5 Boolean hidden units, it can behave as though it has up to 5 bits of memory about its past history on the trail. Each ANN is completely deterministic; it does no learning in its lifetime.

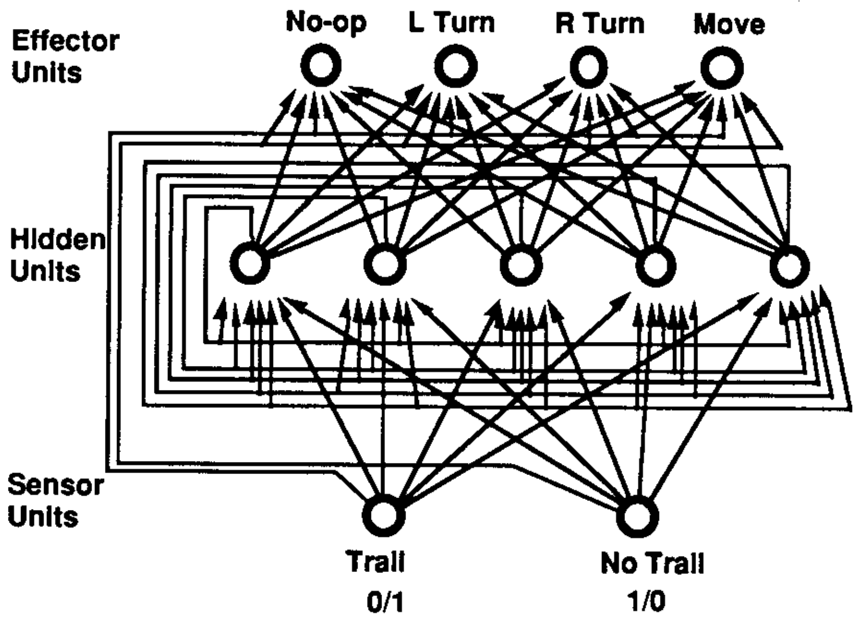


**Figure 5: Architecture of the recurrent ANN representing an ant.**

To specify an ANN for the trail-following task we must decide on the values of 63 weights, 5 thresholds, and 5 initial activations. In our model the unthresholded activations of the hidden units are 7 bits each; the weights are 6 bits (ones-complement), and the thresholds are 7 bits each (the high order bits of an implied 9-bit ones-complement number). The genome representing an ANN-ant is simply the concatenation of all of the weights, thresholds, and initial activations, which takes 63*6 +5*7 + 5*7 = 448 bits. The concatenation is done in a fixed order that was standardized early in the project.

## 6. Comparison of FSA and ANN representations

Since for a long time we were concerned with deciding which of the two representations, ANNs or FSAs, is "better", we went to some length to assure that the FSA and ANN representations of ants are as similar as possible in order to make comparisons meaningful. First, we made the two representations approximately equally powerful computationally. We used FSAs that have up to 32 states, the equivalent of up to 5 bits of memory; likewise we used ANNs with 5 recurrent hidden units, each of which passes on 1 bit of information, so that they have the equivalent of up to 5 bits of memory as well. Both representations express deterministic algorithms. And in both cases an ant's "algorithm" is fixed from birth, so apart from the fact that it can gather a few bits of information about the trail, nothing resembling "learning" takes place during an ant's lifetime.

Second, we made sure that the genomes had similar length: 453 bits for FSAs compared to 448 bits for ANNs. This is important because, other things being equal, we would expect evolution to take longer with a longer genome than with a shorter one because the space being searched by the genetic algorithm grows exponentially with genome length. If these lengths were substantially different, then the size of the space to be searched by genetic algorithm could have been the dominating performance factor making one representation appear to be superior to the other.

Because the ANN's are deterministic, with no more than 5 bits of memory, each of them is behaviorally isomorphic to one of the 32-state FSA organisms. (This fact was very useful in our study because, once we created a tool to translate an ANN to an equivalent FSA, all of our other tools for manipulating FSAs, e.g. for animation or state minimization, applied equally well to ANNs.) However, the reverse is not true; not every FSA can be translated into one of our ANNs because there is a great deal of redundancy in the ANN encoding, and in particular FSA-ants with the largest numbers of states are underrepresented among the ANN-ants. Hence, we must conclude that our ANN-ants, as a class, are somewhat narrower and less powerful computationally than our FSA-ants (though this is difficult to quantify) despite our attempts to make them comparable. There presumably exist tasks, defined by a trail and time limit, that can be accomplished by some FSA-ant, but not by any ANN-ant with the same amount of memory.

Many other ways of encodings the FSAs or ANNs into a string could have been used. We could have placed the rows of the FSA table right to left in the string, or placed the bits of the integers in reverse order, or stored the table columnwise, or used gray codes for the integers instead of binary, etc. We could have reordered the weights in the ANN encoding, or chosen different field widths (e.g. 5-bit weights instead of 6). Technically each of these represents a distinct "representation of organisms". The actual encodings we chose were the only ones we tried, though we have no reason to believe it matters very much. Although there are undoubtedly linkage effects in the particular canonical encodings we chose, we have not explored the consequences of other encodings.

## 7. How difficult is the Tracker task, and how difficult is it to design an ant for the Tracker task?

The John Muir trail was carefully designed so that wherever a straightaway segment of the trail ends, its continuation always begins somewhere in a three-cell-wide extension of the straightaway. The hand-constructed 5-state automaton shown in Figure 6 can traverse the entire trail. Its strategy is to move forward whenever it sees a cell of trail, but when it comes to a point where it does not see the trail in the next cell ahead, it turns right (without moving) and checks for a trail cell there. If it finds one, it moves ahead and continues, but if not, it turns right again. It will turn right a total of four times looking a trail cell. After that, it is facing the original direction, and will move forward anyway, even though there is no trail cell, and will again to search the four directions. We chose a trail that could be traversed this way to be certain that at least some small FSA existed that could perform respectably at the task. There is no reason to suppose, however, that this particular FSA, or any one isomorphic to it, ever actually evolved. And if it did it could not have survived ultimately because, although it can indeed traverse the entire trail, it requires 314 time steps to do so; it only gets a score of 81 in 200 time steps.

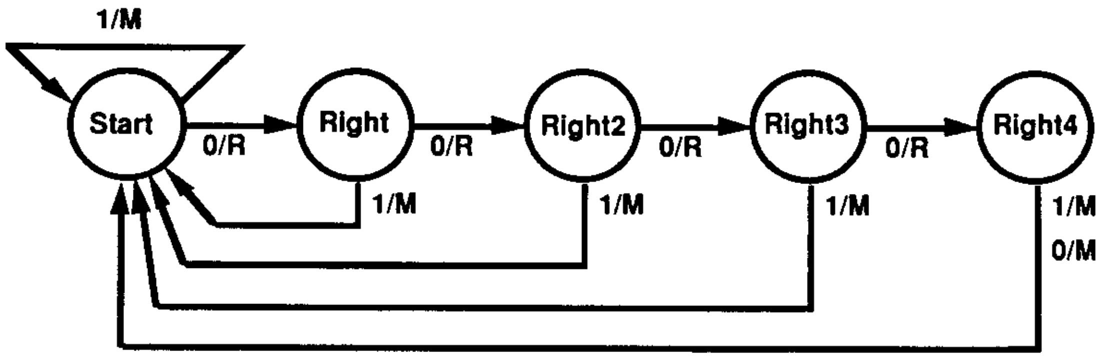


**Figure 6: Five state FSA that can traverse the trail in 314 steps, and gets a score of 81 in 200 time steps.**

Through evolution we were able to find an FSA with 13 states (the equivalent of less than 4 bits of memory) that can get a perfect score of 89 (see ahead in Figure 10).

One can further analyze the difficulty of the task by asking how many bits of information are needed to "memorize" the trail perfectly. If we encode a forward move by a 0, a left turn by the two-bit combination 10, and a right turn by the combination 11, then the entire trail can be unambiguously encoded as a concatenation of "moves" needed to traverse it. Since there are 89 trail cells and 38 non-trail cells also have to be traversed, and since there are 19 turns, then the entire trail can be encoded in 89 + 38 + 2*19 = 165 bits. Our organisms are specified by 450 bits, so it might appear that the could simply evolve to memorize the trail. However, this analysis overlooks that fact that the trail could have be made 100 times longer, and the organisms given 100 times as much time to traverse them. The "information content" of such a trail would be many times larger, but the same 450-bit FSAs and ANNs would be able to successfully traverse it. There is a fundamental difference between evolving programs to traverse a trail, and evolving compact data representations of the trail.

We have come to believe that the Tracker task itself is difficult, but not profoundly so, since it can be accomplished by a modest-sized FSA. But just how difficult is the problem of *designing* or *discovering* a high-scoring organism? That is a completely different issue. Here we are concerned not with searching the environment that the organisms live in, but the searching the space of all organisms. Since the Tracker task is irregular, it does not yield much to analysis. There is no linearity property, for example, to suggest that it is somehow twice as "difficult" to get a score of 80 as a score of 40. Presumably a skilled human could design an 89-scoring FSA for the Tracker task in an hour or two, and an ANN in somewhat more time, but that is not much of a handle on the difficulty of the design task.

To attempt a quantitative calibration of the design difficulty we decided to sample the space of all genomes to discover what fraction of them coded for organisms that got each score. We chose a random sample of over $1.3 \times 10^9$ organisms, of both the FSA and ANN type, and scored each one on the Tracker task to produce the results in Table 1. (In the interest of space we reproduce here only the most interesting regions of the distribution.) The first column shows the score achieved by the animals in the sample. Columns 2-3 show the sample of FSAs. Column 2 shows the exact number of FSAs from the sample that achieved a particular score. Column 3 shows what fraction that represents of the total sample. Columns 4-5 are similar to 2-3, but refer to the ANN representation. The sample was chosen by creating random bit strings of the appropriate length (roughly 450 bits each), translating each string into the appropriate organism, and then running the organism on the Tracker task as usual to tally its score. Since there are $2^{450}$ @ $10^{134}$ possible genomes, our sample of $\sim 10^9$ is an extremely small fraction of the total, but is still large enough to reliably sample all but the high end of the distributions.

| Score | Number of FSAs | Fraction of FSAs | Number of ANNs | Fraction of ANNs |
|---|---|---|---|---|
| 0 | 557,258,091 | .413 | 932,924,367 | .683 |
| 1 | 283,207,491 | .210 | 122,460,544 | .0896 |
| 2 | 132,410,979 | .0981 | 6,275,613 | .00459 |
| 3 | 63,272,402 | .0469 | 944,545 | 6.91e-04 |
| 4 | 31,819,298 | .0236 | 152,981 | 1.12e-04 |
| 5 | 18,120,801 | .0134 | 345,287 | 2.53e-04 |
| 6 | 12,985,610 | .00962 | 20,361 | 1.49e-05 |
| 7 | 10,063,926 | .00746 | 357,592 | 2.62e-04 |
| 8 | 9,603,340 | .00712 | 379,791 | 2.78e-04 |
| 9 | 8,472,800 | .00628 | 38,531 | 2.82e-05 |
| 10 | 97,907,623 | .0726 | 202,234,068 | .1481 |
| ... | ... | ... | ... | ... |
| 40 | 522,627 | 3.87e-04 | 119,459 | 8.74e-05 |
| 41 | 500,301 | 3.71e-04 | 896,204 | 6.56e-04 |
| 42 | 12,110,275 | .00897 | 93,179,890 | .0682 |
| 43 | 369,162 | 2.74e-04 | 7,127 | 5.21e-06 |
| 44 | 170,037 | 1.26e-04 | 22,279 | 1.63e-05 |
| 45 | 110,622 | 8.20e-05 | 2,436 | 1.78e-06 |
| ... | ... | ... | ... | ... |
| 55 | 9,505 | 7.04e-06 | 365 | 2.67e-07 |
| 56 | 9,684 | 7.18e-06 | 288 | 2.11e-07 |
| 57 | 10,218 | 7.57e-06 | 6,715 | 4.91e-06 |
| 58 | 125,296 | 9.28e-05 | 3,875 | 2.84e-06 |
| 59 | 5,488 | 4.07e-06 | 249 | 1.82e-07 |
| 60 | 842 | 6.24e-07 | 2 | 1.46e-09 |
| ... | ... | ... | ... | ... |
| 75 | 255 | 1.89e-07 | 3 | 2.19e-09 |
| 76 | 51 | 3.78e-08 | 8 | 5.85e-09 |
| 77 | 45 | 3.33e-08 | 0 | 0.00e+00 |
| 78 | 354 | 2.62e-07 | 9 | 6.58e-09 |
| 79 | 188 | 1.39e-07 | 19 | 1.39e-08 |
| 80 | 64 | 4.74e-08 | 19 | 1.39e-08 |
| 81 | 10 | 7.41e-09 | 21 | 1.54e-08 |
| 82 | 0 | 0 | 1 | 7.32e-10 |
| Totals | 1,349,517,312 | 1.00 | 1,366,818,816 | 1.00 |

**Table 1: Interesting parts of the score distribution for FSA and ANN organisms**

A glance at Table 1 reveals a number of interesting features. For example, 41% of all FSAs, and 68% of all ANNs receive a score of exactly zero! 90% of both FSAs and ANNs get a score of 10 or less, meaning they cannot make it past the first right turn in the trail. (Of course a small positive score can also be achieved by striking off in an odd direction and "accidentally" touching a few cells of trail.) By referring to the trail in Figure 1 we can see that a score of more than 32 indicates the ability to make the first left turn; a score of more than 42 shows that the animal can get past the first left turn where the corner cell is missing, and a score of more than 58 shows it can get past the first straightaway double gap (always assuming the organism is following the trail in the obvious order). About 1 in 100,000 FSAs can get past 58, while about 1 in 3,000,000 ANNs can do so. Put another way, in a sample of size 65,536 (the initial generation in most of our evolutionary experiments) there is more than a 55% probability of encountering at least one FSA that can traverse the trail past the double gap, while there is only a 2% probability that there will be such an organism in the initial generation of an ANN run.

The highest FSA score discovered in our sample was 81, of which there were 10 instances. The highest score found among ANNs was 82, of which there was only one. In our entire sample we never found an FSA that scored higher than the simple one in Figure 6 would score in the 200 time units. We should note that we typically evolve an 81-scoring FSA within 20 generations with a population of 65,536, i.e. after having generated at most 20*65536 @ $10^6$ ants, whereas if we extrapolate from the observed frequency in the sample (which is dangerous since we are extrapolating from the long tail of the distribution), we would expect about one FSA-ant in $10^8$ to achieve 81. We can only presume that FSA ants that score in the upper 80's are *much* rarer, though there is no feasible way to estimate their frequency by sampling.

In Figure 7 we see the frequency distribution for FSA organisms plotted on a logarithmic scale. There is a clear trend toward exponentially decreasing numbers of organisms as the score increases; but what is most interesting is the pattern of departures from the trend. There are a number of scores, such as 10, 32, 42, 47, 52, 58, 64 and 70, where a sharply larger number of organisms (by a factor of five or more) are clustered than at either neighboring score. A glance back at Figure 1 will show that those scores exactly mark the most challenging features of the John Muir Trail, e.g. the first right turn, the first left turn, the first left-turn-with-gap, the first right-turn-with-gap, the double gap, the first knight's move, etc.

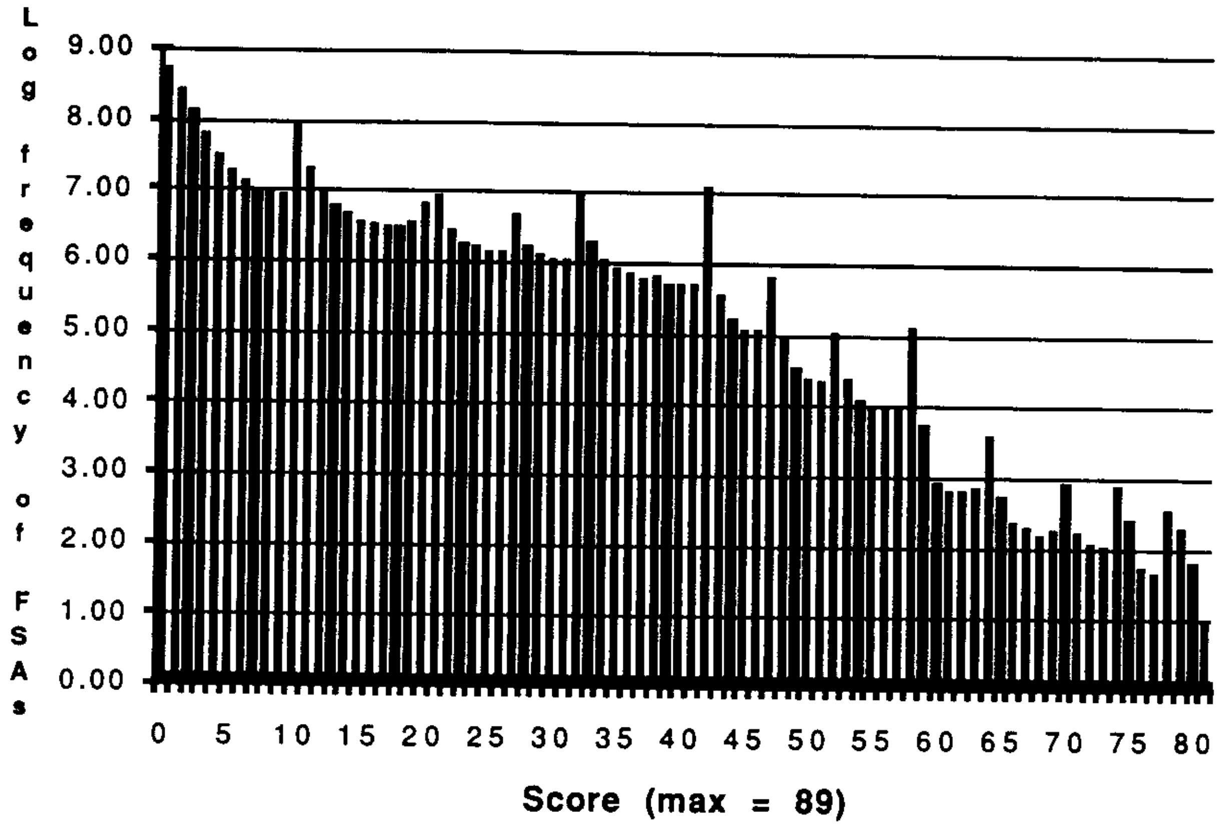


**Figure 7: Log (base 10) of the number of FSA ants in the sample as a function of their score on the John Muir Trail.**

In Figure 8 we see a similar logarithmic plot of the frequency in our sample of the ANN-ants. While there is general similarity to the FSA plot, some of the contrasts are striking. From spikes in this distribution it is still easy to pick out most of the features of the John Muir Trail at scores 10, 27, 32, 42, 47 and 64, but the ratios by which these spikes overshadow the neighboring scores are much larger than for the FSA plot. One curious feature is that there are more ANN organisms that score 57 than 58! We do not know why this should be so, but it is possible that there is a particularly common erroneous path through the trail that happens to achieve that score; we have no other explanation. Another striking difference between the FSAs and the ANNs is that from scores 60 to 89 the total number of ANN ants in our sample (183) is 61 times smaller than the total number of FSA ants (11152). This might suggest that high-scoring ANNs are rarer than high-scoring FSAs. Of course, the ANN sample may be unreliable at such high scores, but probably not since it still seems possible to pick out the expected spikes at scores 64, 70 and 74.

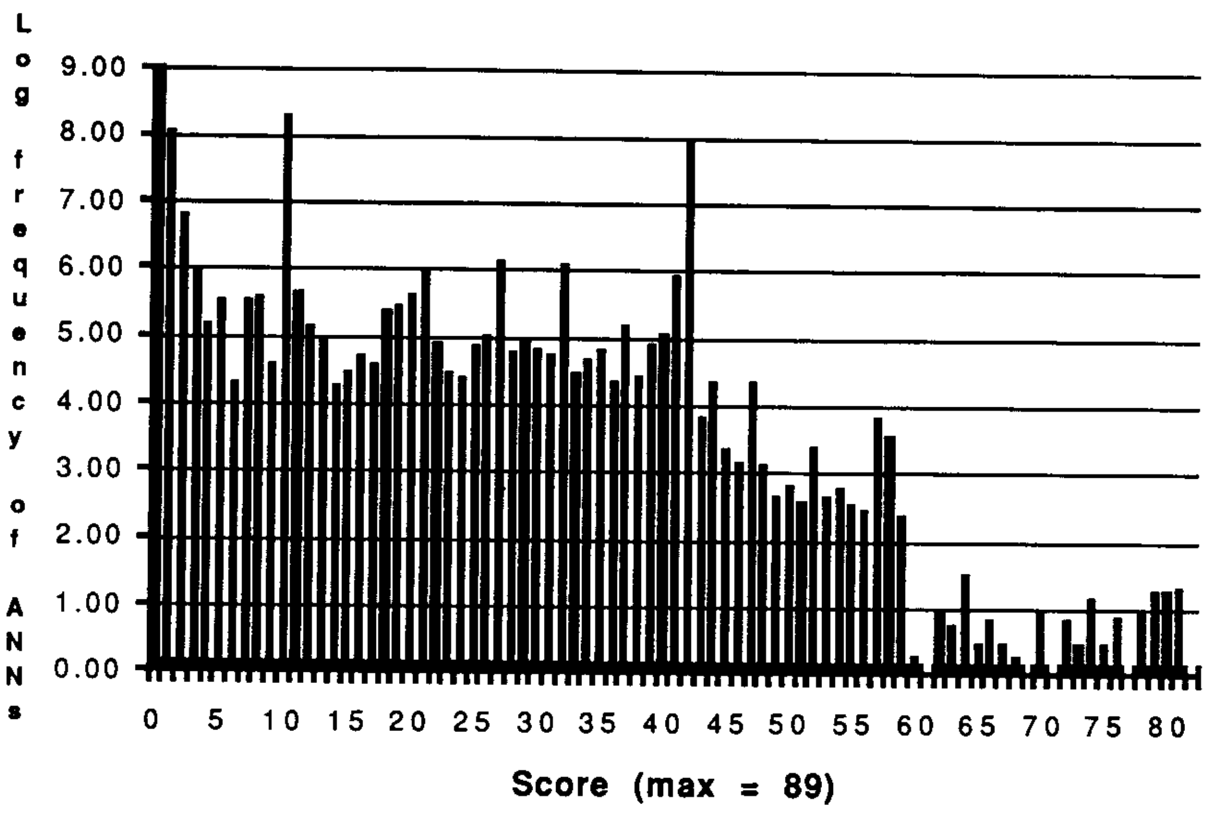


**Figure 8: Log (base 10) of the number of ANN ants as a function of their score on the John Muir Trail.**

From the study of these sample statistics we have come to view the problem of designing an organism for the Tracker task as being not very difficult up to score 81. However, beyond the score of 81 the design task becomes considerably more difficult, not because the John Muir Trail is difficult to traverse *per se*, but because it is difficult to do so in only 200 time steps. It must take over one third of the time to traverse the last eight steps of the trail. Hence, we should expect evolution to proceed in such a way that at the beginning there is competition to evolve logic that is basically competent on the trail; in later generations the competition should be for refinements that take advantage of particular features of the trail to save time (steps).

## 8. The genetic algorithm

The genetic algorithm we used is reasonably standard (Goldberg 1989), though to our knowledge GA's are rarely attempted on bit strings of this length. We begin with a population of 65,536 strings of random bits, each of which is either 448 or 453 bits long (depending on the representation to be used). During a generation each genome is decoded into either an FSA or an ANN, and all 64K ants are executed for 200 time units (in parallel) on separate copies of the trail. At the end of each generation all of the ants are scored as to their success on the trail. Those ants scoring highest of their generation (either the top 1% or 10%) are selected for breeding, and all others are discarded. Ties are broken arbitrarily. The new generation is produced by the following procedure.

a) **Mating**: 65,536 pairs of genomes are chosen at random (with replacement) from the selected fraction. No preference is given to those ants scoring higher than others within the selected fraction.

b) **Recombination**: From each pair a single bit string is constructed by random crossover. The crossover probability is typically between 0.5% and 1.0% per bit, so the mean number of crossovers per ant per generation is 2.25 to 4.5. Crossovers are performed without regard to the boundaries of semantic units in the bit string (e.g. state fields in the FSA genome, or weight fields in the ANN genome).

c) **Point mutation**: The recombined string is mutated by random bit-flip operations, again at a rate anywhere from 0.1% to 1% per bit. The mean number of mutations is thus also 0.225 to 2.25 per ant per generation.

Recombination and mutation are illustrated schematically in Figure 9, where two parent bits strings .

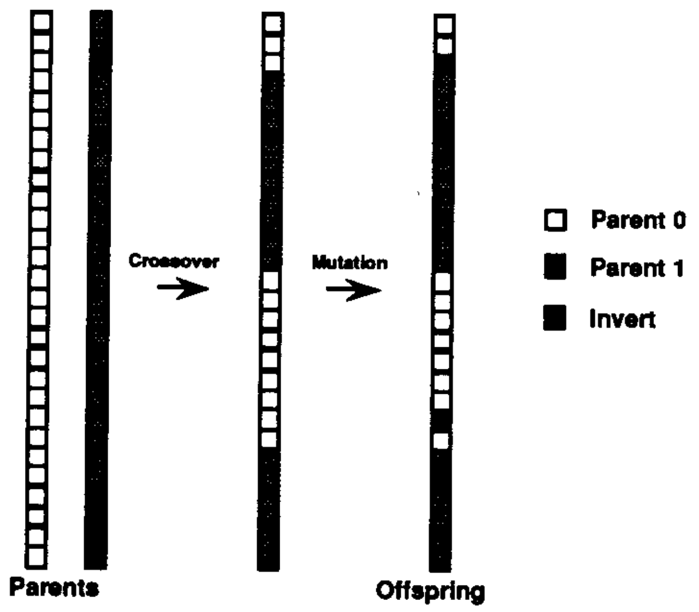


**Figure 9: Two parents produce one offspring by a two-stage process of random crossover followed by random point mutation.**

We performed numerous sensitivity studies to determine that the selection, mutation, and recombination parameters were reasonable and effective. These rates are in line with those reported in (Goldberg and Holland 1988) for other genetic algorithms. The qualitative character of the results are extremely robust over wide variations of the evolutionary parameters. Subsequent analysis showed, however, that mutation rates 100-fold smaller would have been just as effective, and in some ways preferable.

## 9. Results of the Tracker evolutions

Figure 10 shows Champ-0, the highest-scoring ant in generation 0 of one run of Genesys with the FSA representation. Since no evolution has taken place, this is just the highest scorer in a particular random sample of 65,536 FSAs. The diagram shown here has been simplified in two ways in order to expose the underlying algorithm: (a) states and transitions that are coded for by the ant's chromosome but are unreachable on the John Muir Trail have been removed; (b) the result was then processed by a state-minimization procedure to remove redundant logic; and (c) the states were renumbered from 0 to 16.

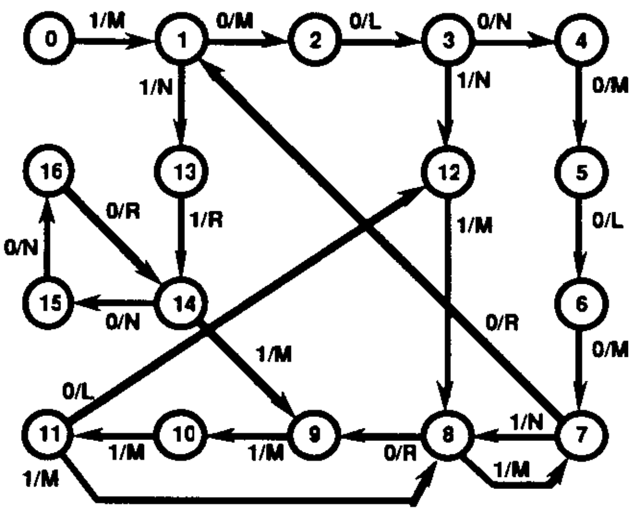


**Figure 10: Champ-0, the champion FSA in generation 0, which scores 58.**

Champ-0 gets a score of 58 on the John Muir Trail. It has great difficulty getting around the first three right turns of the trail, arriving at them in state 7 and taking 10 state transitions in each case (1,13,14,15,16,14,15,16,14) before completing the turn in state 9. It has even greater difficulty making the left turn at trail step 32, requiring two executions of the cycle (8,1,2,3,4,5,6,7) before using the right turn sequence (1,13,...) above. On the straight-away segments of the trail it takes two transitions per step around the 7-8 loop in the diagram, which is a very time-inefficient way to travel. Curiously, Champ-0 makes the left turn where the corner cell is missing at step 42, more efficiently than it makes the earlier left turn at step 32 where the corner is present! Arriving at the corner in state 8, one trip around the cycle (1,2,3,4,5,6,7,8) makes the turn. Champ-0 runs out of time when it gets to the double gap at trail step 58.

In Figure 11 we show Champ-100, an FSA-ant that is the product of 100 generations of evolution starting from the population of which Champ-0 was the champion. The evolution was conducted with a mutation rate of 0.5% per bit, a recombination rate of 0.5% per bit, and a selection fraction of 1% per generation. Its logic has been simplified in the same way Champ-0's was. Although Champ-100 is descended from the same population that included Champ-0, there is no reason to believe Champ-0 is actually a genetic ancestor of Champ-100, and in fact it is rather unlikely.

Champ-100 gets a perfect score of 89, demonstrating that it is indeed possible to traverse this trail in 200 steps. In fact, Champ-100 takes exactly 200 time steps to finish the trail. Since the scoring function does not give any extra credit for reaching the end sooner, there is no selective pressure for any further improvement in performance.

We can observe a number of fascinating features in Champ-100's logic. First, it traverses the straightaway portions of the trail by staying in state 0 or in state 9, thereby taking one unit of time per trail step rather than two. Second, it makes a right turn from state 0 using the (1,12,0) cycle, a far more efficient mechanism than Champ-0 used, though not optimal. Third, the left turn at trail step 32 is accomplished by three right turns. We deliberately placed three right turns in the trail before the first left turn precisely to see if there would be an evolutionary bias toward more efficient right-turning. In this case, apparently, there was.

The state sequence (9,10,11) is extremely versatile. It is used in four critical places along the trail: at step 42 (left turn with missing corner), at steps 58 and 74 (straightaway double gap) and at step 64 (forward right knight's jump). Likewise, the sequences (12,7,8,3,4,0) and (6,4,0) are used repeatedly to traverse the "stepping stone" part of the trail from step 78 to 89. Such efficient logic seems exquisitely adapted to the features of this particular trail, and suggests that evolution has had the effect of "compiling" knowledge of this environment into the structure of the organism.

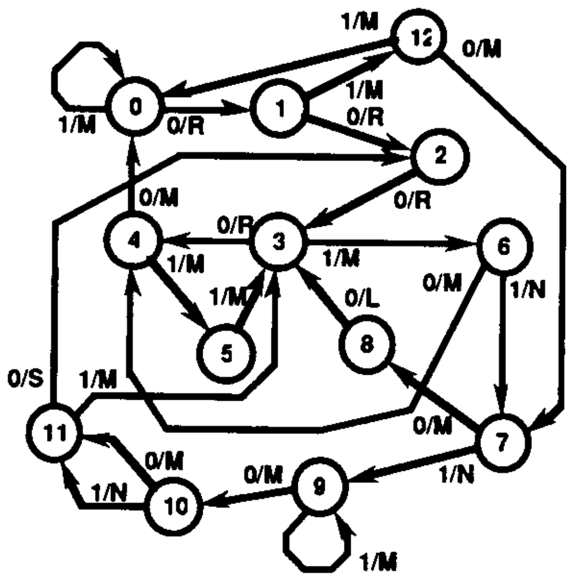


**Figure 11: Champ-100, the champion FSA in generation 100, which scores 89.**

In Figures 12 and 13 we chart the evolutionary progress of the entire population in the Tracker task, one chart for FSA ants and one for ANN ants. These are runs typical of many we have made, and are chosen for presentation here because their parameters are identical. In both cases the mutation and recombination rates are 1% per bit per generation, and the selection fraction is 5% per generation. Both runs evolve a population of 65,536 for 100 generations. Each took about one hour on a 16K-processor Connection Machine (CM2).

Figure 12 is a graph of the progress of an evolution with FSA ants. The horizontal axis is the generation number, and the vertical axis is the score on the Tracker task. Four population statistics are plotted as a function of generation:

(a) the maximum score achieved by any ant in the generation
(b) the mode (most frequent) score
(c) the mean score
(d) the standard deviation of the scores

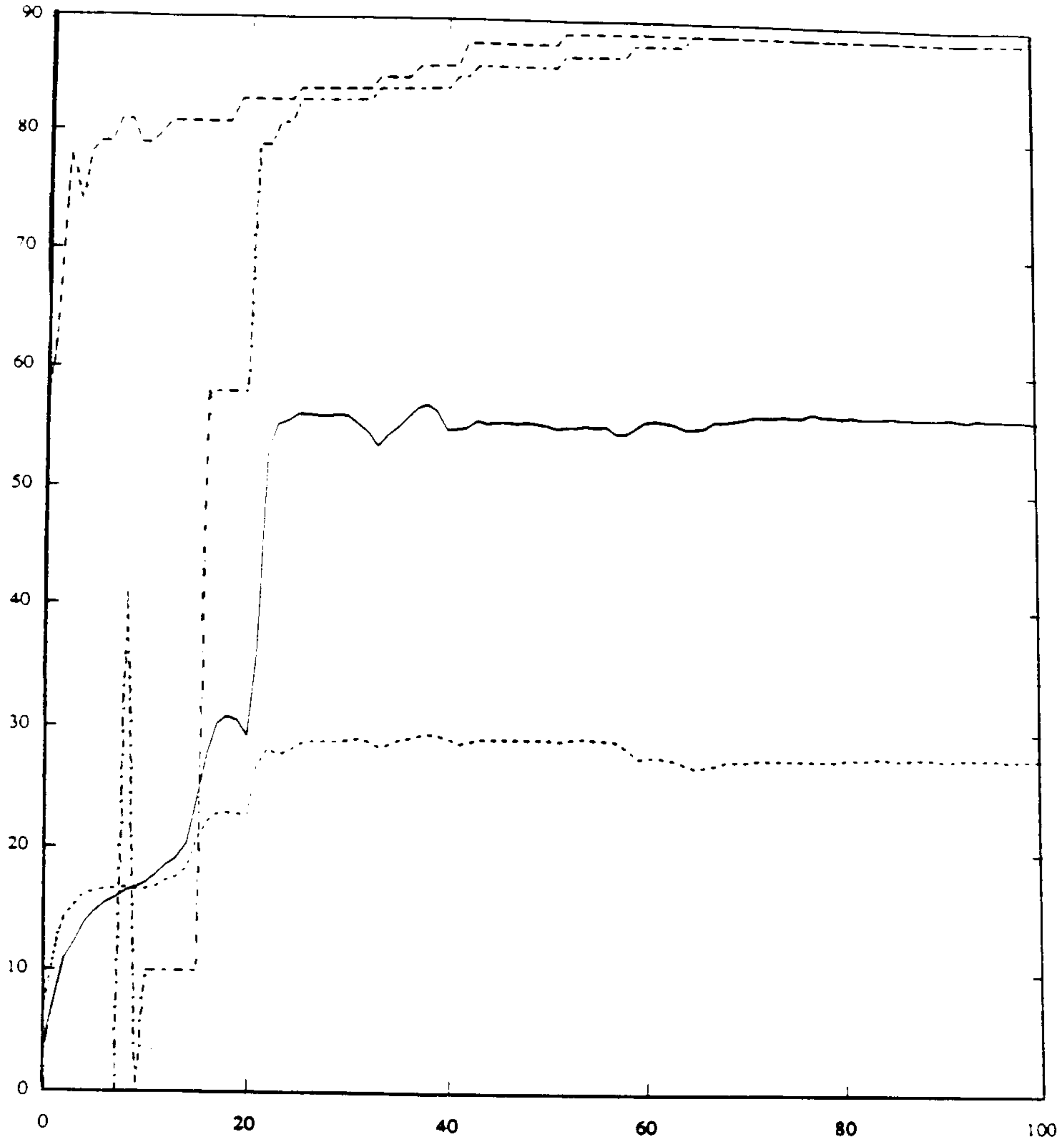
Initially the mean score was somewhere around 3, but the maximum score was 58, consistent with the statistics given in Section 7. After 15 generations of selection the maximum score is already in the 80's, and by generation 52 perfect-scoring ants are always present in the population, and are the most common score by generation 70. However, the mean score never approaches the maximum score, hovering around 56 for the last 75 generations, and there is always a high standard deviation. The high variance and the great difference between the mean and the max scores is, we have learned, a consequence of the relatively high mutation rate of 1% per bit. We can get the same max score in the same number of generations with a mutation rate two orders of magnitude smaller, and then the mean score reaches the 80's. With a high mutation rate shown here, a large fraction of successful parents have offspring that are destroyed by mutation and hence bring down the average score.

Figure 13 is a similar graph except that the ants are represented as artificial neural nets. All evolution parameters are the same as in Figure 12. We see similar features in Figure 13, i.e. the maximum score starts low (this time at 46) and quickly reaches the 80's (by generation 18). It then takes a long time to reach 89 (generation 94). As in Figure 12, there is a large gap between the maximum score and the mean score, and for the same reasons. Comparing Figure 12 to Figure 13 we see that evolution proceeds somewhat more slowly in the ANN representation than in the FSA representation, for reasons unknown. This result was consistent over hundreds of executions with many different parameters.

## 10. Discussion and conclusions

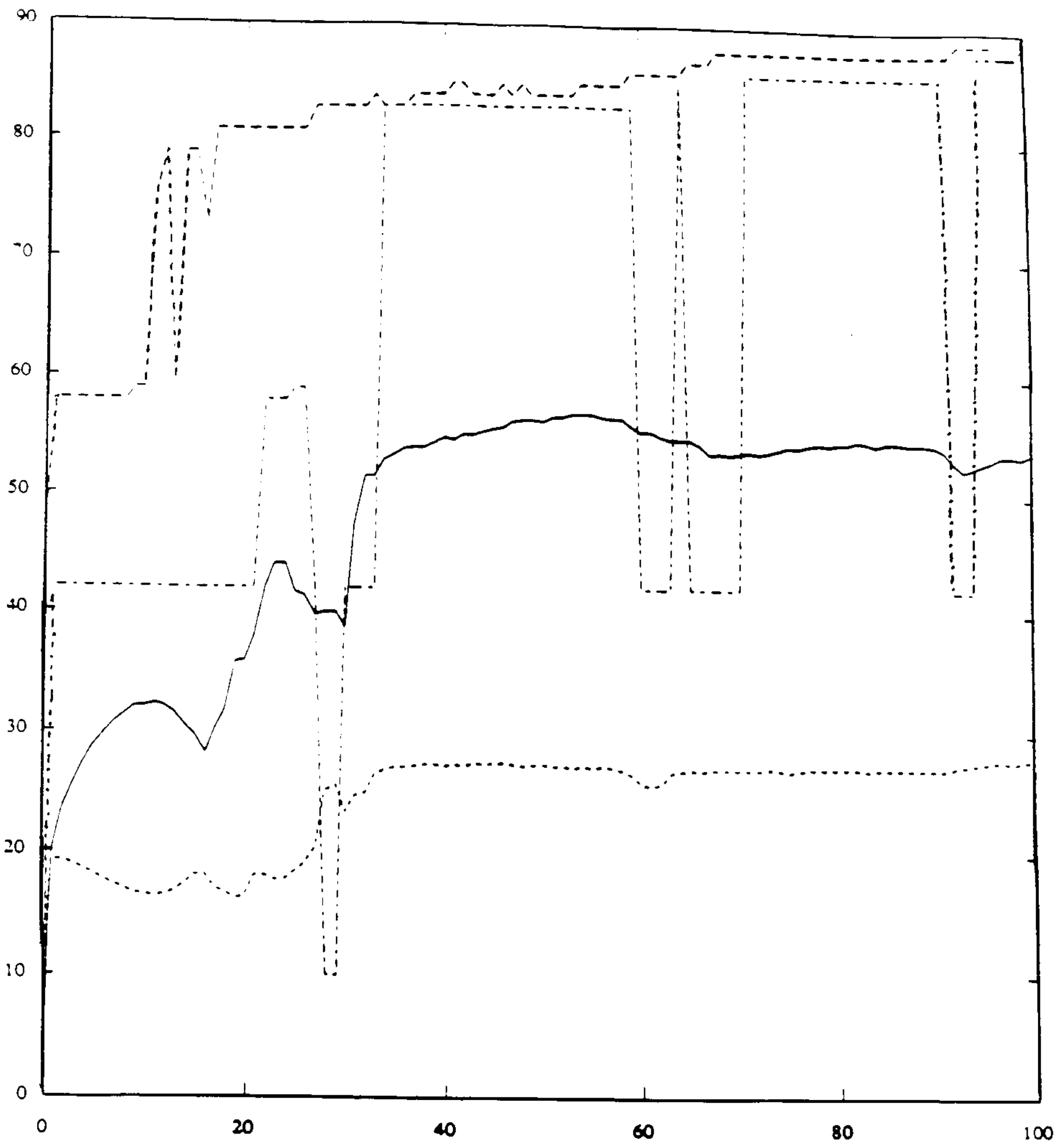
This research clearly shows, as we had hoped, that it is computationally feasible to produce artificial organisms that can exhibit complex behavior, and to produce them by evolution. And evolution can act on the entire text of the program representing the organisms, at least with populations of 64K organisms and genomes of 450 bits. In that sense the exercise was a success.

Rather than decide which of the two representations was "better" for the study, we concluded that working with two representations was extremely important for completely different methodological reasons. First, it tends to eliminate the potential problem that we might observe an evolutionary phenomenon which is merely an artifact of the representation, and not a genuine feature of evolution. If we perform each experiment many times, with different representations and different environments, etc., and if we observe the same evolutionary phenomenon in all cases, then we can be more certain that whatever phenomenon we observe is a property of evolution itself, and not an artifact of the representation. Second, it focuses attention on a fundamental issue that all similar studies must face in the future: just what is a good computational representation of living organisms? We expect that this issue will continue to be of fundamental importance in future artificial life studies.



| | | | |
|---|---|---|---|
| Date: Apr 26, 1989 | Chromosome Length: 520 | Score | |
| Execution Time: 0:53:11 | Fraction Selected: 0.050000 | Mean | ——— |
| Population: 65536 | Mutation Rate: 0.010000 | Std. Dev. | -------- |
| Random Seed: 8276342 | Crossover Rate: 0.010000 | Maximum | ------ |
| Iterations/Generation: 200 | Decision Maker: fsa | Mode 1 | -·-·-·- |
| Environment: 32 x 32 | FSA States: 5 | | |

Figure 12



| | | |
|---|---|---|
| Date: Apr 26, 1989 | Chromosome Length: 520 | Score |
| Execution Time: 1:5:22 | Fraction Selected: 0.050000 | Mean ——— |
| Population: 65536 | Mutation Rate: 0.010000 | Std. Dev. -------- |
| Random Seed: 8276342 | Crossover Rate: 0.010000 | Maximum ------ |
| Iterations/Generation: 200 | Decision Maker: net | Mode 1 -·-·-·- |
| Environment: 32 x 32 | Hidden Units: 5 | |

Figure 13

As a result of this study we have identified a number of important properties that we believe a programming paradigm must have to be suitable as a representation for organisms in biologically motivated studies. Among them the following:

(a) The paradigm must be what we refer to *computationally complete*, i.e. it must be possible for any bounded-memory (finite state) algorithm to be encoded in it, provided that the size of the representation is large enough. If such organisms are intended for an infinite environment, and have the ability to move around and read and write into it, then the paradigm is Turing-equivalent.

(b) It should specify a *simple, uniform model of computation* so the interpreter can be small, and so there is minimal chance to bias the system by choosing a programming model that happens to rich in the very operators or strategies needed to adapt to a particular environment. Furthermore, if the goal is to have a simple model for studying complex natural systems, it is important that organism programs should be manipulable and understandable.

(c) It should be *syntactically closed* (or nearly so) under the genetic operators. Mutation and recombination operators must not (usually) transform legal programs to illegal ones. In practice, we have designed our representations so that all bit strings (of the appropriate length) encode for some legal program. With this approach we get the added benefit that we can start evolution with a population of "random" organisms by just producing a set of random bit strings, a simple and indisputable way of avoiding bias in the initial population from which evolution starts.

(d) It should be *well-conditioned* under the genetic operators. This requirement is not very formally defined, but essentially requires that "small" mutational changes in the program should (usually) cause "small" changes in its behavior, and that a crossover of two parent programs should usually produce an offspring program whose behavior is in some sense a "mixture" of the parents' behavior. These requirements are probably necessary for evolution to have a chance to succeed in realistic adaptive landscapes. Of course, occasional jumps or discontinuities can be tolerated, and are likely even necessary.

(e) It should specify *one time unit* of the organism's life. This basically means that one execution of an organism's program should (a) accept input from its sense organs (both external and internal), (b) possibly change its internal state, and (c) possibly take one or more actions through effectors. An organism's life, then, is the iterated execution of its behavior program.

(f) It must *scale well*. This means that the size and time complexity of an organism's program is a modestly-growing function of the size of the input to or output from it. (Finite state automata (FSAs), at least as we formulate them in Section 4, fail this test. It takes a transition table with $2^{m+n}$ rows to specify an FSA with $m$ bits of state and $n$ bits of input, so if the entire transition table is encoded in the genome, then the genome length grows exponentially with the size of the organism's sensory apparatus, which is unacceptable for all but the smallest $m$ and $n$.)

In RAM (Taylor et al, 1989) both organisms and the environment are represented as programs. The behavior and evolution of the population in the environment is simulated by the co-execution of all of the organism programs and environment programs. One of the good things about this design choice is that it allows the environment to be just as "active" as the organisms are, and it makes a statement that the line between organism and environment is often arbitrary. It also allows each organism to be part of the environment of the others, so that competitive and cooperative relationships can evolve. Organisms are not of different fundamental stuff than the environment: all are represented as programs, and have a symmetric, coequal status in the simulation. But, however attractive this point of view is from a modeling perspective, it does not properly capture the true relationship between organisms and their environment in natural life. For in RAM, organism and environment programs are defined separately, interacting as though each is "outside" the other, whereas in natural life organisms are definitely "inside" the environment, and part of it. This distinction between environment and organism is manifest in the fact that while organisms "move" and "reproduce", these actions are treated as primitive, and do not occur as the result of any lower-level processes. The organism's behavior function says "move" and, *as if by magic*, the location coordinates of the organism change; the behavior function says "reproduce", and *as if by magic* a full copy of the organism (complete with modified genes), appears in no time on a neighboring location. Organisms in RAM do not really have a simulated "physical" body whose natural activity produces motion and reproduction. In natural life, however, both "environment" and "organism" are epiphenomena arising from the same physics below.

Genesys has these limitations of RAM and others as well. While the organisms are represented as programs in Genesys, the environment is almost completely static.

Theoretically it would be extremely fruitful to correct this modeling deficiency, but we know of no way to do that and still retain the principle that an organism is a program. The only really satisfactory way we know to remove the separation between organism and environment is to invent an artificial physics, such as a huge cellular automaton, in which both the organisms and the environmental processes are represented as interacting patterns of activity in the same playing field. For the size of population and the complexity of simulation that we envision today, that is computationally out of the question.

As a result, in Genesys, and all succeeding systems that we envision, we will continue to simulate each organism by a program. We will continue to simulate not the organisms themselves, but their *behavior*, in their environments. We may, at a later stage, endow organisms with a "body", e.g. arms and legs, so that the primitive operations are reduced from "move organism" to "move leg". But for the foreseeable future in our research there will always be a level at which the organism "interfaces" with the environment, or "communicates" with it, instead of participating in it.

## Acknowledgements

This research was funded in part by a grant from the W. M. Keck Foundation and by National Science Foundation Biological Facilities grant number BBS 87 14206. We also thank Jon Bentley for extremely detailed correction and comments.

## Bibliography

Collins, Robert J. and Jefferson, David R. (1990) AntFarm: Towards simulated evolution, submitted to *Artificial Life II*, Langton, C. G., Farmer, D., Rasmussen, S. and Taylor, C. (eds.), Addison-Wesley

Elman, J. L. (1988) Finding structure in time. CRL Tech. Rep. 8801. Center for Research in Language. University of California, San Diego

Gibson, Robert, Taylor, Charles, E., and Jefferson, David R. (1990) "Lek formation by female choice: a simulation study", *Journal of the International Society for Behavioral Ecology*, 1(1), 36-42.

Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley

Goldberg, D. E. and Holland , J. H. (eds.) (1988). *Machine Learning*. Vol. 3, Nos. 2-3, Special Issue on Genetic Algorithms, Kluwer Academic Publishers

Hillis, W. D. (1985) *The Connection Machine*. MIT Press, Cambridge, Mass.

Holland, John (1975) *Adaptation in Natural and Artificial Systems*, The University of Michigan Press

Holldobler, Bert and Wilson, Edward O. (1990) *The Ants*, Harvard University Press

Koza, John R. (1990) "Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems", Stanford University Computer Science Dept. TR No. STAN-CS-90-1314, June, 1990

Langton, C. G. (ed.) (1989) *Artificial Life*, Addison-Wesley

Rumelhart, D. E. and McClelland, J. L. (eds.) (1986) *Parallel Distributed Processing*, Vols. 1 and 2. Bradford Books/MIT Press.

Servan-Schreiber, D., Cleeremans, A. and McClelland , J. L. (1989) *Learning Sequential Structure in Simple Recurrent Networks*. In D. S. Touretzky (ed.) Advances in Neural Information Processing Systems 1, pp. 643-652

Taylor, Charles E., Jefferson, David R., Turner, Scott, and Goldman, Seth (1989) "RAM: Artificial life for the exploration of complex biological systems", in Langton, Chris (ed.) *Artificial Life*, Addison-Wesley

Taylor, Charles E., Muscatine, L., and Jefferson, David (1989) "Maintenance and Breakdown of the Hydra-Chlorella Symbiosis: A Computer Model", *Proceedings of the Royal Society, London*, Series B, B238, pp. 277-289

Taylor, Charles E., Jefferson, David R., and Burla, Hans, (1987) "Habitat-dependent dispersal of *Drosophila obscura* and *D. subobscura*", *Genetica Iberica*, 39, p. 547