Evolving Dynamical Neural Networks for Adaptive Behavior

Randall D. Beer
Department of Computer Engineering and Science
Department of Biology
Case Western Reserve University
Cleveland, OH 44106
beer@alpha.ces.cwru.edu

John C. Gallagher
Department of Computer Engineering and Science
Case Western Reserve University
Cleveland, OH 44106
johng@alpha.ces.cwru.edu

We would like the behavior of the artificial agents that we construct to be as well-adapted to their environments as natural animals are to theirs. Unfortunately, designing controllers with these properties is a very difficult task. In this article, we demonstrate that continuous-time recurrent neural networks are a viable mechanism for adaptive agent control and that the genetic algorithm can be used to evolve effective neural controllers. A significant advantage of this approach is that one need specify only a measure of an agent's overall performance rather than the precise motor output trajectories by which it is achieved. By manipulating the performance evaluation, one can place selective pressure on the development of controllers with desired properties. Several novel controllers have been evolved, including a chemotaxis controller that switches between different strategies depending on environmental conditions, and a locomotion controller that takes advantage of sensory feedback if available but that can operate in its absence if necessary.

Key Words: neural networks, genetic algorithms, motor control, chemotaxis, locomotion

Introduction

The natural behavior of animals is amazingly well adapted to the environments in which they are embedded. Through the process of evolution, an animal's responses come to "fit" the dynamical and statistical structure of its environment. To achieve this fit, animals are endowed with nervous systems whose dynamics are such that, when coupled with the dynamics of their bodies and environments, these animals can engage in the patterns of behavior necessary for their survival. Moreover, these biological control systems are remarkably versatile and robust. Insects, for example, can walk under a great variety of "normal" circumstances (e.g., over complex terrain, upside-down, etc.) that can pose qualitatively different control problems. In addition, because leg damage is not uncommon, insects can also cope with a variety of peripheral abnormalities such as sensory damage or leg amputations (Graham, 1985). In some cases, such as the loss of one or two legs, this compensation can involve a major reorganization of gait. Nevertheless, the structure of an insect's nervous system is such that the appropriate reorganization occurs immediately.

There is currently a great deal of interest in the design of artificial autonomous agents whose behavior exhibits a similar adaptive fit with the environments in which they must function (Beer, 1990; Maes, 1991; Meyer and Wilson, 1991; Langton et al., 1991). There is a growing suspicion that the traditional artificial intelligence (AI) approach to intelligent behavior, which involves explicit reasoning by manipulating symbolic representations of the world, is inadequate for systems that must operate in realistic environments (Winograd and Flores, 1986; Agre and Chapman, 1987; Beer, 1990; Brooks, 1991). In contrast, the adaptive behavior approach is more directly related to the ethology of simpler animals than it is to the human psychology that has driven most traditional AI and cognitive science research. It is characterized by the notion that a great deal of intelligent behavior arises from the interaction between an agent's internal control mechanisms and its external environment rather than from an agent's ability to reason explicitly with symbolic representations of its situation. The adaptive behavior approach is concerned with several questions: How much intelligent behavior can be understood in this fashion? What classes of control mechanisms are best suited to the generation of adaptive behavior? How can the appropriate control mechanisms be designed or automatically developed?

In this article, we will focus on the latter two questions. Specifically, we will examine the power of continuous-time recurrent neural networks as a mechanism for adaptive agent control, and we will explore the ability of the genetic algorithm to evolve such networks. After describing the basic framework, we illustrate its use to evolve neural controllers for two behaviors: chemotaxis and legged locomotion. We also attempt to gain some insight into how the genetic algorithm is searching the space of neural controllers. This article concludes with a discussion of the relative strengths and weaknesses of our approach as compared to other control mechanisms and approaches to their development.

2 Approach

For the purposes of this article, we assume that the dynamics of an agent's body and environment are fixed *a priori*. This is an obvious simplification of the biological situation, since natural bodies and nervous systems coevolve and realistic environments are rarely stationary. In addition, we assume the existence of some measure of the overall appropriateness of an agent's behavior. We will employ only those performance measures that do not include any direct specification of the detailed behavior required, because we are ultimately interested in environments in which the "correct" motor outputs may not be known. One extreme example of such a performance measure would be the average length of time the agent survives in the given environment.

2.1 Dynamical Neural Networks

Neural networks are parallel, distributed models of computation whose design is loosely based on the organization of natural nervous systems (Anderson and Rosenfeld, 1988). A neural network consists of a collection of model neurons interconnected in some fashion. Typically, a model neuron receives as input a weighted sum of the outputs of some subset of the other neurons. The details of a model neuron can vary along a number of dimensions, including whether its input-output behavior is static or time-dependent, and the nature of the nonlinearity in its response. In addition, neural models can be divided into discrete-time (those whose outputs change discontinuously in time) and continuous-time (those whose outputs change smoothly in time) models. Architectures of connectivity can be broadly classified as being either feedforward (an input layer of neurons connects to an output layer through one or more intermediate or hidden layers in a unidirectional fashion) or recurrent (feedback loops are allowed). Recurrent architectures may range from restricted classes of feedback to full interconnection between every neuron in the network. Regardless of the particular model and architecture chosen, neural networks are usually trained to perform a desired task by some general learning procedure.

The behavior of our agents is controlled by continuous-time, recurrent neural networks, which we will refer to as *dynamical neural networks*. The basic state equation for the *i*th model neuron is as follows:

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{N} w_{ji}\sigma_j(y_j) + I_i(t) \tag{1}$$

where y is sometimes interpreted as the mean membrane potential of the neuron; $\sigma_j(\xi) = (1+e^{(\theta_j-\xi)})^{-1}$ is a sigmoidal (S-shaped) function that can be interpreted as its short-term average firing frequency; θ is a bias term that controls the firing threshold; τ is a time constant associated with the passive properties of the cell membrane; w_{ji} represents the strength of the connection from the *j*th to the *i*th neuron; and $I_i(t)$ represents an external input to the network such as from a sensor. Models of this form were studied by Hopfield (1984) and have recently been employed by a number of others in both discrete and continuous form (for review, see Pearlmutter, 1990). It should be noted that we do not assume, as do some authors, zero diagonal symmetry of the weights (i.e., we do not assume that $w_{ij} = w_{ji}$ and $w_{ii} = 0$). Therefore, networks of model neurons described by Equation 1 admit of more complex dynamical behavior than fixed points.

Given a particular neural architecture, we wish to find settings of the thresholds and time constants of each neuron, and the weights of the connections between them, that produce the most appropriate behavior for a given body and environment. This is a parameter estimation problem with the goal of maximizing the performance measure (or minimizing an equivalent error measure). Toward this end, a number of generalizations of the backpropagation training algorithm for feedforward nets have been developed for recurrent neural networks. Pearlmutter (1989), for example, has developed a technique for training the outputs of continuous-time recurrent neural networks to arbitrary temporal state trajectories.

Unfortunately, techniques such as these require a desired state trajectory for comparison with the actual output trajectory of the network so that an error signal can be computed. In our case, however, no such desired state trajectory generally exists. Although more general parameter estimation techniques are available, most of these techniques can find only local extrema and they do not scale well with problem size. Therefore, they do not perform particularly well at finding good (if not near-global) maxima or minima for high-dimensional nonlinear systems containing many dozens of free parameters. For these reasons, we have employed genetic algorithms in the work described in this article.

2.2 Genetic Algorithms

A genetic algorithm (GA) is a search technique whose operation is loosely based on natural evolution (Holland, 1975; Goldberg, 1989). Genetic algorithms have proved to be a powerful technique for searching vast, complex, multimodal, noisy, discontinuous spaces. For our purposes, they offer the possibility of automatically producing controllers appropriate to a given environment without the need for explicit analysis or design.

The basic cycle of a GA operates as follows. The space to be searched is usually encoded as a binary string. An initially random population of such strings is maintained. At each iteration, the performance of each individual is evaluated. A new generation of individuals is then produced by applying a set of genetic operators to selected individuals from the previous generation. Individuals are selected for reproduction with a probability proportional to their fitness. The standard genetic operators are *mutation* (in which bits are randomly flipped) and *crossover* (in which portions of the genetic strings of two individuals are exchanged). Crossover is generally considered to be the principal search mechanism, with mutation relegated to a background operator whose only role is to maintain diversity in the population and to ensure that every point in the search space has some chance of being visited. By iterating the processes of selection, recombination, and mutation, the population accumulates information about the distribution of fitness in the search space. This information focuses subsequent search into fruitful subspaces.

There has recently been a great deal of interest in combining neural networks and GAs. Much of this activity has centered on feedforward neural networks (Belew, McInerney, and Schraudolph, 1991). There are four basic approaches: (1) using a GA to find appropriate connection weights instead of backpropagation; (2) using a GA to find good settings for backpropagation parameters (e.g., learning rate and momentum); (3) using a GA to find good initial weights for backpropagation; and (4) using a GA to find good feedforward architectures for backpropagation. There has also been some work using GAs to set the parameters of discrete-time recurrent neural networks (e.g., Wieland, 1990; de Garis, 1990; Werner and Dyer, 1991). To the best of our knowledge, GAs have not previously been applied to continuous-time recurrent neural networks.

There are a number of ways to map a neural network onto a binary genome. The simplest method, and the one chosen here, is to discretize each parameter (threshold, time constant, or connection weight) and encode it in some fixed number of bits. The individual parameter encodings are then concatenated to produce the final genetic string. Typically, Gray-coding is used to ensure that small changes in the bit string correspond to small changes in the parameter value (Caruana and Schaffer, 1988). There are tradeoffs involved in such a mapping of real numbers to binary strings. Precision concerns dictate that each parameter be represented with as many bits as possible. On the other hand, allocating many bits to each parameter unnecessarily increases the size of the search space, because the least significant bits are generally less important early in the search, whereas the most significant bits are less important later in the search as the population begins to converge on a particular subspace.

In an attempt to resolve this dilemma, dynamical parameter encoding (DPE) has been introduced (Schraudolph and Belew, 1990). DPE is a technique whereby the actual mapping between a real parameter and a fixed set of bits is changed based on statistics gathered during the run. For example, if four bits of the genome are mapped to a parameter in the range $[-16, 16]$ but the population converges on values in the range $[0,16]$, then DPE will remap these four bits to this subrange, thereby doubling the precision. One problem with this zoom operation is that it is irreversible. Unless it is applied very conservatively, DPE can prematurely remove regions of the search space from further consideration. DPE is a relatively new technique, and there has been little practical experience with it as yet. However, DPE was employed in all experiments described in this article.

All experiments described herein were performed using a public-domain GA simulator known as *GAucsd* (version 1.1).[1] This GA simulator utilizes two-point crossover, Gray-coding of real parameters, and Baker's stochastic universal sampling selection procedure (Baker, 1987). It includes an implementation of DPE. The GAucsd parameter settings used in our experiments can be found in the Appendix.

3 Chemotaxis

Orientation to sensory stimuli is one of the most fundamental behavioral acts that an animal can perform. Many more sophisticated behaviors require that some object of interest first be brought into the appropriate spatial relationship with the animal. For example, feeding requires that an animal first align itself with some source of food. Because food sources often emit a characteristic odor that diffuses throughout the environment, chemical cues are used for this purpose by many animals. Orientation to a chemical stimulus is known as *chemotaxis*, and is the first behavior that we sought to evolve.

3.1 The Problem

In the specific problem that we considered, the agent is enclosed in a square box with a circular patch of food at its center. This food patch emits a chemical signal whose intensity falls off as the inverse square of the distance from the center of the patch. The intensity of the chemical signal within the environment varies five orders of magnitude from the center of the patch to the far corners of the box. To solve the general problem of chemotaxis, we wish the agent to find and remain in the vicinity of the food patch, starting from arbitrary locations and orientations within its environment.

To accomplish this task, the agent is endowed with a circular body (Figure 1). The agent possesses chemical sensors that can directly sense the intensity of the chemical signal at their location. These sensors are symmetrically placed about the center line of the body. In addition, the agent has two effectors located on opposite sides of its body. These effectors can apply forces that move the body forward and rotate it. In the simplified physics of this environment, the *velocity* of movement is proportional to the force applied.

The agent is controlled by a six-node fully interconnected dynamical neural network. Two neurons serve as sensory neurons and have an external input equal to the intensity of the chemical signal at their locations. Two neurons serve as motor

1 This software was originally named GENESIS, but its name has been changed to avoid confusion with several other similarly named systems. At the time of this writing, GAucsd is available by anonymous ftp from cs.ucsd.edu (132.239.51.3) in the pub/GAucsd directory.

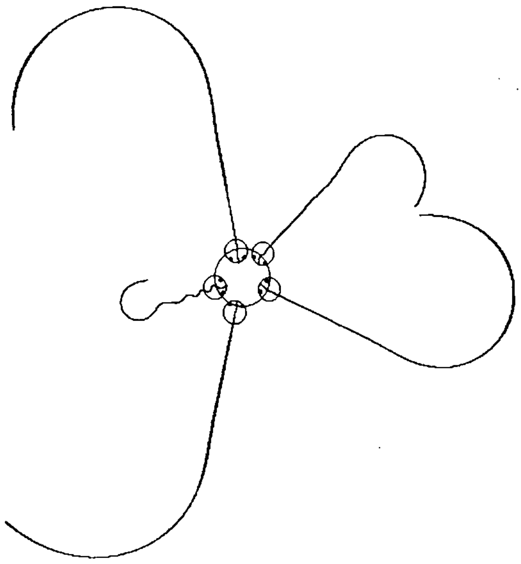


Figure 1
The behavior of a typical tropotactic agent. The paths followed by the agent from a number of different initial locations and orientations are shown.

neurons whose outputs drive the effectors. The remaining two neurons serve as interneurons whose role is unspecified. Due to the symmetrical nature of the problem, we impose bilateral symmetry on the network. There are thus 24 free parameters to be estimated (3 thresholds, 3 time constants, and 18 weights). Assuming that each real parameter is encoded in 4 bits (see Appendix), this gives us a genetic string 96 bits long.

To evaluate the performance of a given neural controller, the agent is started from a number of predetermined locations and orientations with randomized initial neuron states and is allowed to run for a fixed number of time-steps. At the end of each trial, the square of the distance between the agent and the center of the food patch is computed. The average of this value over a number of trials is then used as a measure of the controller's fitness, with lower values corresponding to better performances.

3.2 Results

Several dozen successful chemotactic agents were evolved. Almost all of these employed variations of the obvious strategy of turning toward the side of the stronger chemical signal while moving forward. This strategy is known as *tropotaxis* in the animal behavior literature (McFarland, 1981). Figure 1 illustrates the behavior of

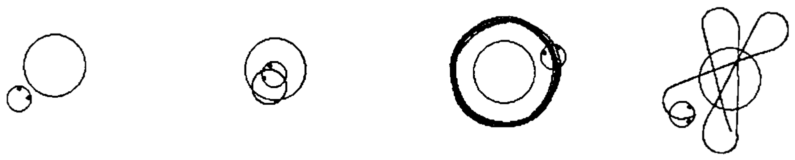


Figure 2
Examples of each of the four classes of near-patch behavior.

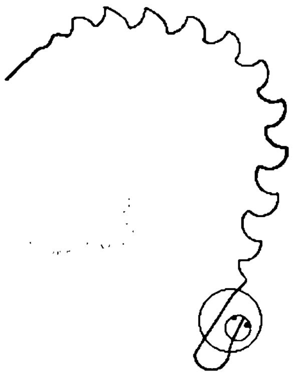


Figure 3
A chemotactic agent that initially uses klinotaxis but then switches to tropotaxis in the vicinity of the food patch.

one such agent. Note that this agent quickly finds the patch, regardless of the agent's initial location and orientation. These solutions varied in their sensitivity to the weaker signals found at large distances from the patch and their speed of movement. In addition, the exact behavior near the patch fell into four distinct classes: Some agents come to a complete stop, some move in a small circle, some orbit the patch at different distances, and some repeatedly cross the patch, "patrolling" it (Figure 2). Note that the performance measure we are using puts no particular selective pressure on the details of the agent's behavior near the patch, as long as it stays nearby.

In several runs, solutions emerged that were rather different in character from the simple tropotaxis just described. The dynamics of these controllers were such that they generated different strategies depending on their distance from the patch. The behavior of a typical example of this class of agents is illustrated in Figure 3. At large distances, the agent moved from side to side with oscillations that were biased toward the side on which the chemical signal was stronger, causing its path to curve toward the patch. This behavior is similar to *klinotaxis* (McFarland, 1981). Near the food patch, these oscillations ceased and the agent exhibited the more common tropotaxis described earlier.

Underlying the behavior of this agent is a neural controller with multiple modes of activity. However, the distinctions between these modes are somewhat obscured by the agent's movements, which affect the sensory signals that it receives. To more closely examine these modes, we clamped the sensory inputs to different values and examined the resulting network dynamics. When the sensory input signals were small in absolute magnitude and their difference was also relatively small, as would be the case at intermediate to large distances from the food patch, the network generated an oscillatory output in which the duration of the output burst on the side opposite the stronger signal was somewhat longer. This would tend to bias the agent's rhythmical movements toward the food patch. When the relative difference between the two sensory inputs was large, as would be the case near the patch, this oscillatory mode was no longer stable and the network entered a mode in which the motor output on the side opposite the stronger signal was continuously active. This would cause a steady turn in the direction of the patch. Finally, when the absolute magnitudes of the sensory inputs were large but their difference was zero, as would be the case when the sensors saturate within the patch, both motor outputs were continuously active, causing the agent to move forward in a straight line. Similar multifunctional neural circuits that can switch between distinct dynamical modes have been found in a number of invertebrates (e.g., Getting and Dekin, 1985; Selverston and Moulins, 1987).

A few experiments were also run using a body containing only a single chemical sensor. Chemotaxis with a single sensor is more difficult than with bilaterally paired sensors because the agent has no direct measure of the chemical gradient. Instead, a single-sensor agent must use more indirect means to find the patch. One possibility would be to move from side to side in the manner of klinotaxis, using subsequent sensory signals to estimate the chemical gradient in time rather than space. However, this type of solution was not observed. Instead, in a number of experiments utilizing a variety of five- and six-neuron architectures, we found that the best solution was to move in large loops throughout the environment, decreasing the radius of the loop as the strength of the chemical signal increases and eventually coming to a stop when the signal becomes sufficiently strong near the patch. In the environment that we have considered here, this turns out to be a reasonably effective strategy.

A number of observations were made concerning the effects of details of the performance evaluation on the character of the evolved controllers. For example, the length of time that agents were allowed to move before their distance from the patch was evaluated could affect the speed of movement of the resulting solutions.

Shorter evaluations tended to evolve agents that moved much more quickly than those that were allowed a longer period of time before they were evaluated. In addition, evaluating an agent's performance using points at greater distances from the patch tended to evolve agents that were more sensitive to weak signals than using nearby points as the basis for evaluation. Finally, averaging fewer than three evaluation points tended to produce agents that only passed by the patch at the time of evaluation when started from the given evaluation points, rather than solving the general problem of chemotaxis.

4 An Examination of the GA Search

To gain some insight into the way in which the GA is searching the network parameter space for good controllers, we compared the performance of a series of GA experiments using different combinations of genetic operators. A population size of 500 was used in all experiments. Though successful chemotaxis controllers could be evolved with populations as small as 100, premature convergence to nonsolutions was much more common than at larger population sizes. Except as indicated in the text, the complete GAucsd parameter settings for these experiments can be found in the Appendix.

The progression of average population performance for three typical runs using the GAucsd default parameters for crossover and mutation are shown in Figure 4A. Recall that the GA is minimizing the square of the final distance between the agent and the food patch, so lower numbers correspond to better performance. Note that all three plots are very similar, consisting of an initial drop that levels off near 20,000, followed by a subsequent drop and leveling off near 1000 as the population begins to converge. The progression of best individual performance (Figure 4B) is likewise similar across the runs. In all three cases, the best solution found had an error of approximately 100 or less, implying that, on average, these agents are found within 10 pixels of the center of the food patch at the time of evaluation.

It is interesting to compare these plots to those obtained using a simple random sampling of the parameter space. In this search technique, we simply take a number of independent random samples of the parameter space equivalent to the total number of evaluations that the GA performs. For ease of comparison, we divided these samples into groups of 500. Because the samples are independent, no net improvement takes place and the "population" performance oscillates around the average value of the error measure in the parameter space (Figure 5A). The performance of the best individual found so far does improve with time, eventually reaching a value of slightly more than 1000 after some 15,000 samples have been taken (Figure 5B). The fact that simple random sampling found a solution with this low an error (considering

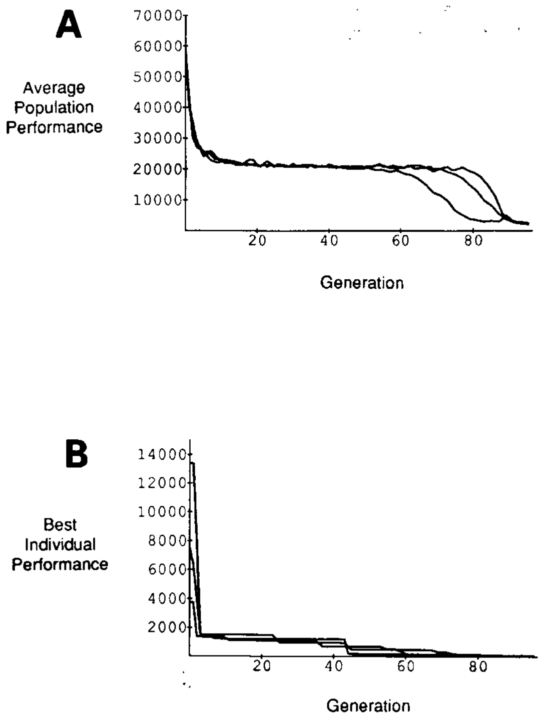


Figure 4
Performance of the standard GA for three typical runs. (A) Average population performance. (B) Best individual performance.

that the average solution appears to have a performance of approximately 65,000) suggests that "decent" solutions are not uncommon in this space. However, the performance of this solution is still more than an order of magnitude worse than the solutions found by the GA. It is also worth noting that using other common measures of search performance such as on-line performance (a running average of the population performance) or off-line performance (a running average of the best individual performances), the GA performs significantly better than random sampling.

We next examined the role that selection plays in the GA search by running a GA with mutation and crossover disabled (Figure 6). Selection operates by copying individuals from the previous to the next generation with a probability proportional to their fitness. Since no new individuals are being generated after the initially random population, selection quickly converges. Interestingly, the shape of the average population performance curve for a selection-only search is very similar to the initial portion of the average population performance curve for the standard GA (see Figure 4), suggesting that the initial stage of the search is being driven primarily by selection. Note that the best individual performance still improves somewhat with time, even though no new individuals are being generated after the first random population (Figure 6B). This is because the performance of the entire population is being evaluated each generation. Because each evaluation occurs with randomized initial neuron states and some initial conditions lead to better performance than others, slightly better performances are occasionally found.

To search a particular space effectively with crossover, it should be the case that the location of parameters on the genome be related to their degree of coupling, with highly interdependent parameters located as close together as possible. Unfortunately, there is no reason whatsoever to believe that this condition is satisfied by our simple encoding of network parameters. In general, all of the parameters of a high-dimensional nonlinear dynamical system can be coupled in very complex ways. Two additional problems with this encoding have to do with the properties of neural networks themselves (Belew et al., 1991). First, neural network solutions are generally far from unique in terms of the network parameters. One would expect that crossing the parameter sets of two good but different solutions would produce offspring whose fitness has little, if anything, to do with the fitness of its parents. A second problem arises from the fact that one may obtain fully isomorphic networks that nevertheless

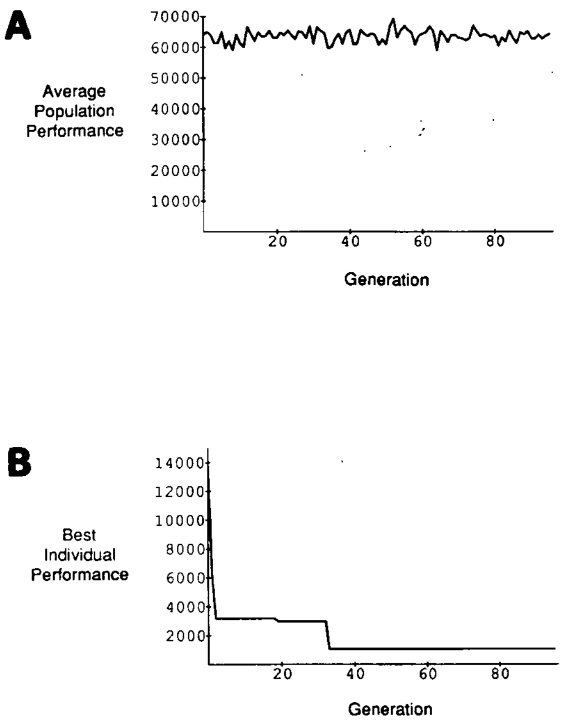


Figure 5
Performance of random sampling. (A) Average population performance. (B) Best individual performance.

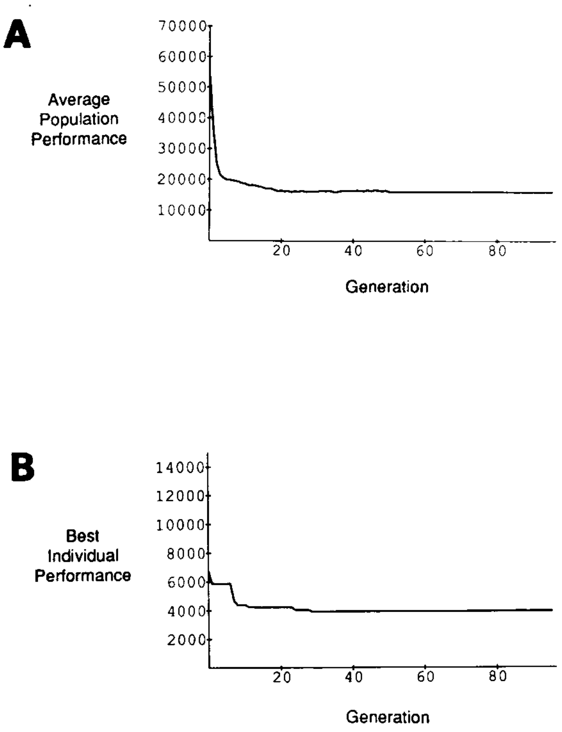


Figure 6
Performance of a selection-only GA search. (A) Average population performance. (B) Best individual performance.

still have rather different genome representations simply by permuting the roles of the interneurons. Such considerations have led some researchers interested in the application of GAs to neural networks to forego crossover altogether (e.g., de Garis, 1990), whereas others have still found standard crossover to be a useful search operator (e.g., Whitley and Hanson, 1989; Wieland, 1990; Werner and Dyer, 1991).

We next examined the role of crossover in the GA search by disabling mutation. Two of three runs evolved solutions, whereas the third prematurely converged to a nonsolution (Figure 7). This behavior is consistent with the standard interpretation of GA search because no mutation is available to ensure that every point in the search space has some probability of being visited and to maintain diversity in the population. Of all the experiments described in this section, these are the most similar to the average population performance and best individual performance curves exhibited by the standard GA (see Figure 4), except that the final convergence is somewhat noisier. This similarity suggests that crossover is playing the major role, despite the fact that the representation we are using does not seem to be particularly amenable to crossover. Though the long plateau near 20,000 suggests that crossover may be doing little more than random sampling for much of the search, the fact that usually it eventually does find considerably better solutions than random sampling suggests that it is exploiting at least some structure of the search space to focus its sampling.

Mutation is normally considered to be a background operator in the GA literature. To examine the role of this operator, we ran a final series of GAs with crossover disabled (Figure 8). When the default mutation rate for GAucsd is used (probability of mutation, $\mu = 0.0002$ per bit), mutation is indeed a very poor search mechanism. Its performance is very similar to that seen using only selection (see Figure 6), as might be expected for such a low probability of mutation. On the other hand, using a much larger mutation rate ($\mu = 0.08$) results in behavior that is similar to simple random sampling (see Figure 5). However, an intermediate mutation rate ($\mu = 0.008$) produces a very effective search. It is interesting to note that, for the almost 100-bit genetic string that we are employing here, these mutation rates correspond respectively to an average of roughly 0.02, 8, and 0.8 mutations per genetic string. Thus the most successful mutation rate corresponds to mutating slightly less than 1 bit per individual on average. From these results, we would have to conclude that the combination of selection and mutation alone can be a very effective search procedure. A number of other investigators have recently argued that mutation's

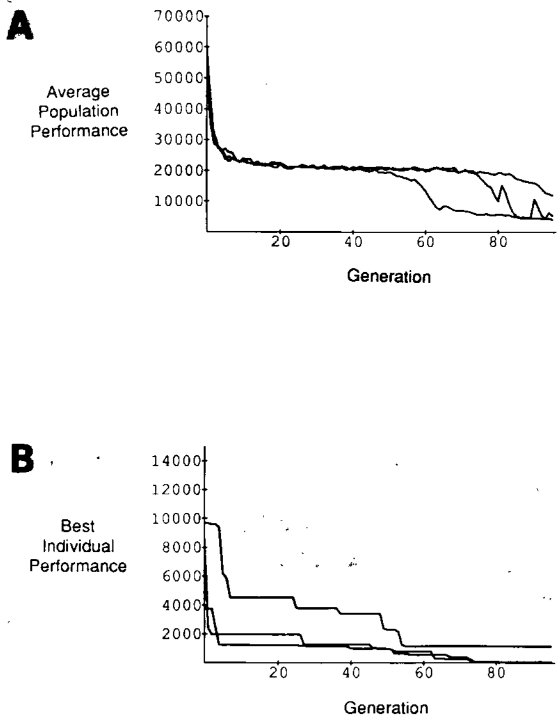


Figure 7
Performance of a crossover-only GA search for three typical runs. (A) Average population performance. (B) Best individual performance.

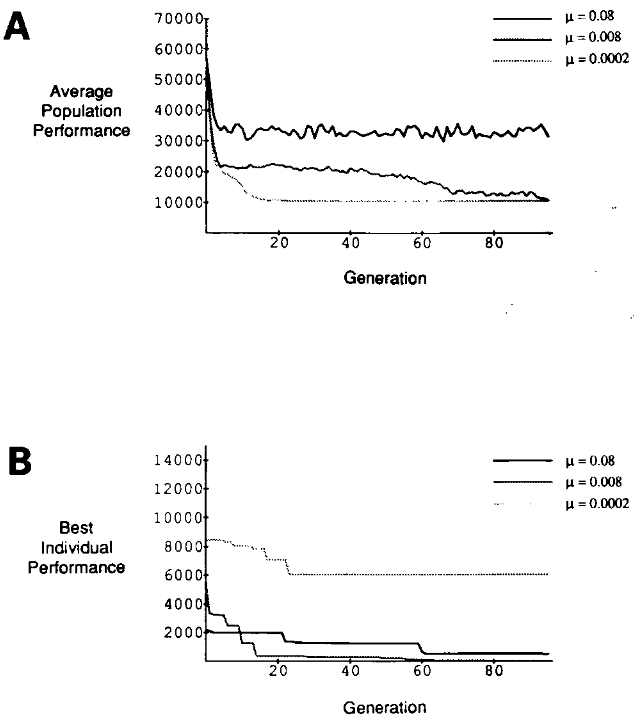


Figure 8
Performance of a mutation-only GA search for three different mutation rates.
(A) Average population performance. (B) Best individual performance.

status as a background operator in GAs needs to be reconsidered (e.g., Schaffer et al., 1989; Fogel and Atmar, 1990).

Locomotion

Like orientation, locomotion is another fundamental behavior for an autonomous agent. Even many orientation behaviors depend on some means by which an animal's body can be moved along a desired path. In the case of legged animals, a locomotion system must simultaneously solve the two tightly coupled problems of support and progression. Because legged locomotion is in many ways a more difficult control problem than chemotaxis, we next explored the ability of a GA to evolve effective controllers for this behavior.

5.1 The Problem

The agent possesses an insect-like body that is an elaboration of a model we have employed in previous work (Figure 9; Beer, 1990). There are six legs, each containing

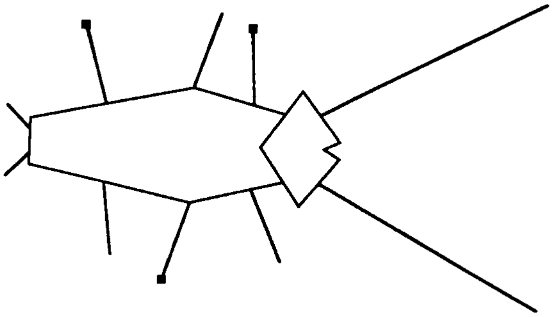


Figure 9
Simulated insect body model.

a foot that may be either up or down. When its foot is down, a leg provides support and can generate forces that move the body (a stance phase). When its foot is up, a leg is incapable of providing support, and any forces it generates serve only to swing that leg (a swing phase). The insect can move only when it is statically stable (i.e., when the polygon of support formed by the feet that are down contains its center of mass). When the body is stably supported, the forces exerted by each stancing leg are summed to produce a resultant force that translates the body under Newtonian dynamics. In addition, a velocity decay term was added to prevent the insect from coasting indefinitely after a single stance. Whenever the insect falls, its velocity drops immediately to zero.

Each leg contains three effectors: One governs the state of the foot, and the other two generate clockwise and counterclockwise torques about the leg's single joint. The intent of opposing torques is to model the antagonistic muscles that are ubiquitous in natural limb control. These forward and backward torques are summed to produce a resultant torque about the leg's joint. Depending on the state of the foot, this resultant torque either causes the leg to swing or generates a force on the body. Each leg has a limited range of angular motion. A supporting leg may stretch outside of this range, but it can apply forces only within these limits. In addition, each leg possesses a single sensor that measures its angular position.

Each leg is controlled by a five-neuron fully interconnected dynamical neural network (Figure 10A). Three of these neurons serve as the motor neurons, governing the state of the foot, the forward swing torque, and the backward swing torque, respectively. The other two neurons serve as interneurons whose role is unspecified. Each neuron also receives an additional weighted input from the leg's angle sensor (i.e., $I(t)$ in Equation 1 is of the form $w_s\theta(t)$, where $\theta(t)$ is the leg angle in radians and

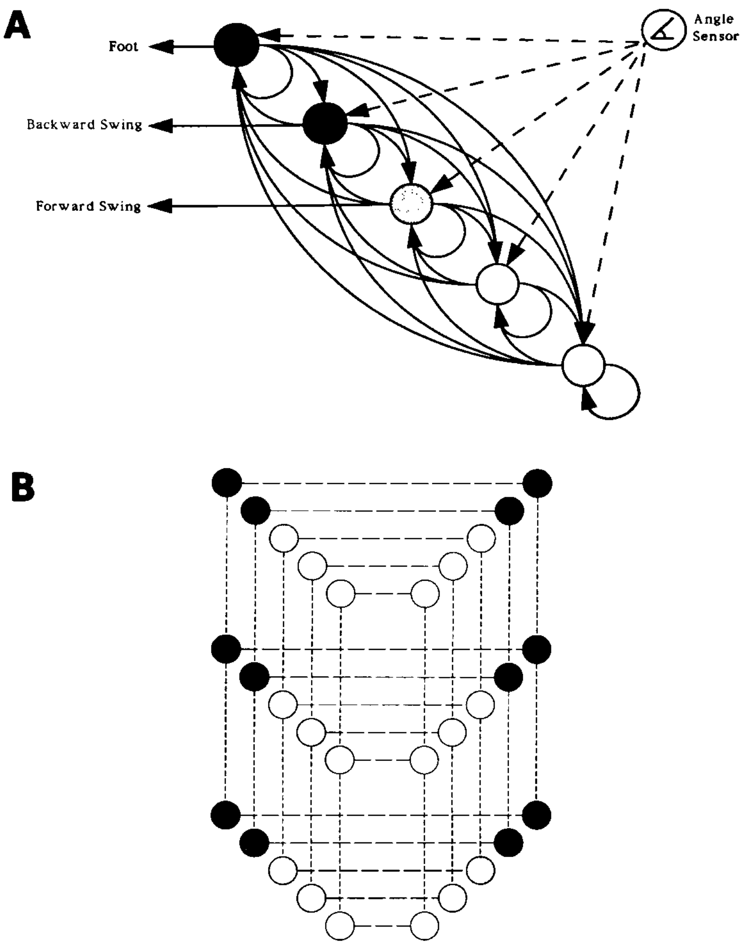


Figure 10
Locomotion controller network architecture. (A) Architecture of a single leg controller. (B) Coupling between leg controllers. Identically shaded neurons have identical thresholds and time constants. Dashed lines represent crossbody connections, and dotted lines represent intersegmental connections. Connections between identically shaded neurons with the same line style have identical weights. All coupling connections are symmetrical.

w_S is the weight of this sensory input). There are thus 40 free parameters (5 thresholds, 5 time constants, 25 connection weights, and 5 sensor weights) to be estimated in a single leg controller, requiring a genetic string 160 bits long for representation.

Though in principle it should be possible to form a locomotion controller by fully interconnecting six leg controllers, this strategy is simply not feasible in practice because such a network would have 990 parameters, resulting in a 3960-bit genetic string! For this reason, we made two simplifying assumptions to reduce this architecture to a reasonable size (Figure 10B): (1) We assumed that each leg controller neuron connects only to the corresponding neuron in each of the leg controllers adjacent to it. Thus the neurons are much more tightly coupled within a leg controller than between two leg controllers; (2) We assumed that the locomotion controller exhibits left-right and front-back symmetries.

Assumption 2 implies that the leg control problem is identical for the two front and two back legs. We also assumed that the leg control problem for the middle legs was identical to that of the other legs. Therefore, only one set of leg controller parameters needs to be encoded on the genetic string; these parameters are then copied to each of the six legs. The symmetry assumptions also imply that the coupling connections are symmetrical: the weights of the connections between any pair of coupled neurons are the same in both directions. Finally, these symmetries imply that the crossbody connections between the front, middle, and back leg controllers are identical. The intersegmental connections along each side of the body are likewise identical. These simplifying assumptions leave us with a network architecture with 50 free parameters (40 leg controller parameters, 5 crossbody connection weights, and 5 intersegmental connection weights), resulting in a genetic string 200 bits long. Thus the full locomotion problem amounts to a 6-input, 18-output nonlinear control problem with 50 free parameters.

This architecture is a generalization of one that has been proposed as a model for the neural circuitry underlying cockroach locomotion (Pearson, Fourtner, and Wong, 1973; Pearson, 1976). We have previously explored the properties of an extended version of Pearson's model in simulation and found it capable of robustly producing a continuous range of statically stable hexapod gaits (Beer, Chiel, and Sterling, 1989; Chiel and Beer, 1989; Beer, 1990). A significant difference between this model and the present architecture is that our implementation of Pearson's model utilized rhythmical pacemaker neurons as essential network elements, whereas the model neurons employed here are not intrinsically oscillatory.

The performance of a given locomotion controller was evaluated by randomizing the insect's initial leg positions and neuron states and then allowing it to run for a fixed amount of time. The performance measure to be maximized was the forward distance that the insect travels in this fixed amount of time. This is equivalent to minimizing the squared difference between the actual position of the insect at the end of an evaluation and the position it would have attained if it had been continuously traveling at its maximum velocity. Because the insect can make forward progress only when it is statically stable, the GA must evolve a neural controller that generates 18 motor outputs with the appropriate amplitude, phase, and duration such that the insect remains stable at all times and moves forward as quickly as possible.

5.2 Single Leg Controller Results

The most basic component of walking is the rhythmical motions of the individual legs. The creation of a dynamical neural network capable of producing the appropriate three motor outputs for leg control is itself a nontrivial task. Therefore, we first examined whether we could evolve leg control networks. A successful leg controller must have the following properties: (1) The activation of the backward swing and foot motor neurons must be in phase with each other and out of phase with the activation of the forward swing motor neuron; (2) It must rhythmically oscillate between these two states; (3) The duration of these phases must be matched to the range of motion of the leg. Note that, for the purposes of evolving single leg controllers, we must modify the notion of static stability so that a one-legged insect is stably supported whenever its single foot is down and unstable otherwise. Without this modification, a one-legged insect would be unable to move.

Five different leg controllers were evolved under the conditions just described. In each case, leg controllers passed through three distinct stages. First, insects appeared that planted their foot, pushed the body as far forward as possible, and then stopped. These controllers thus exhibited roughly the proper phasing of the motor outputs. Next, insects appeared that were capable of rhythmically lifting their leg and swinging it forward. However, these stepping movements were suboptimal because they did not utilize the leg's full range of motion. In the final stage, the controllers were fine-tuned to the leg's range of motion.

The behavior of one evolved leg controller over a number of steps is shown in Figure 11A. In each step, the velocity steadily increases and then drops abruptly when the insect lifts its single leg and falls. Significant insight into the operation of these

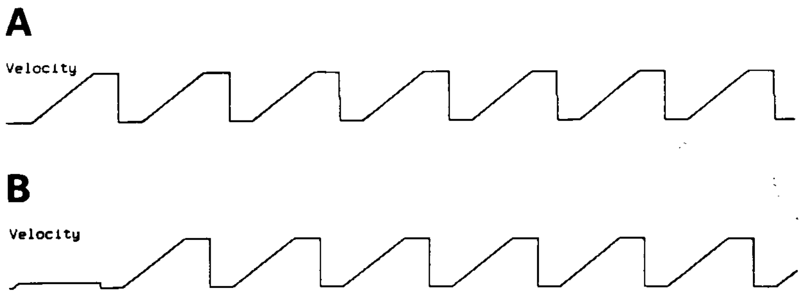


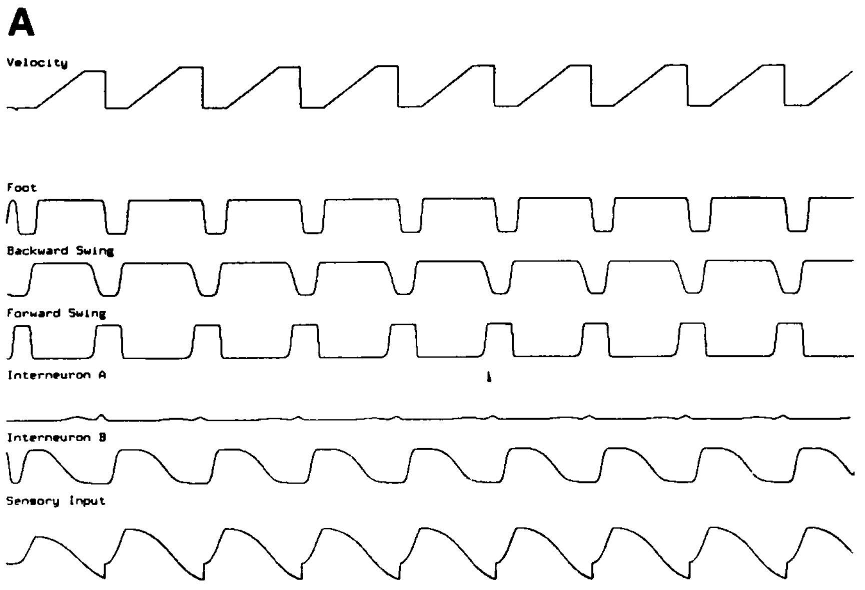
Figure 11
Behavior of evolved leg controllers. The forward velocity of the body is shown for each controller. (A) A reflexive pattern generator. (B) A central pattern generator. In both cases, the leg was initialized at 95 percent of its full backward position.

circuits can be gained by observing the effects of removing the sensory input signal. Under these conditions, we found that the controller generates only a single step; this lesion has abolished the circuit's ability to lift the leg rhythmically. Similar results were obtained with the other leg control networks. These leg controllers are thus wholly dependent on sensory feedback for their operation. Since these networks were evolved with access to a sensory signal, and the genetic algorithm is known to be very opportunistic, it is not surprising that solutions are found which take full advantage of this sensory information. In contrast, most biological control circuits can tolerate the loss of some sensory feedback, making them robust to sensory damage.

In another series of experiments, we sought to evolve leg controllers that could operate in the absence of sensory feedback by forcing the output of the angle sensor to zero during all performance evaluations. Such motor control circuits are known as *central pattern generators* (Delcomyn, 1980). These controllers passed through three evolutionary stages similar to those previously described. In four of five experiments, central pattern generators (CPGs) were successfully evolved. The behavior of one such network over a number of steps is shown in Figure 11B. Even though the individual neurons are not intrinsically oscillatory, this network as a whole is capable of generating the necessary rhythmical control signals.

One disadvantage of these pure CPGs is that, while they can produce rhythmical stepping in the absence of sensory feedback, they are incapable of taking advantage of sensory information when it is available to fine-tune their outputs. An example of this problem can be seen in the first step taken by the CPG (see Figure 11B). The first velocity ramp is truncated because, due to the initial leg angle, the leg has quickly reached its backward angle limit and is no longer able to apply force. However, the CPG cannot know this since it receives no sensory information about the state of the leg, so it continues to attempt to apply a force. For this reason, the average performance of CPG leg controllers is consistently worse than the performance of those leg controllers that utilize sensory feedback.

In contrast, natural nervous systems are designed to take full advantage of any available sensory feedback, since such information is normally available, but to operate (often in a somewhat degraded fashion) in the unusual circumstances of its absence. We next sought to evolve leg controllers with similar properties using a composite performance measure given by averaging the performance of a controller with and without sensors. Five such mixed controllers were evolved. Again, leg controllers passed through three stages similar to those described earlier, though the boundaries were less distinct. As can be seen in Figure 12, a mixed controller performs very well with its sensor intact but is also capable of rhythmically stepping when the sensor is removed, though not as efficiently. Note that the role of interneuron A is significantly increased when the sensory input is removed.



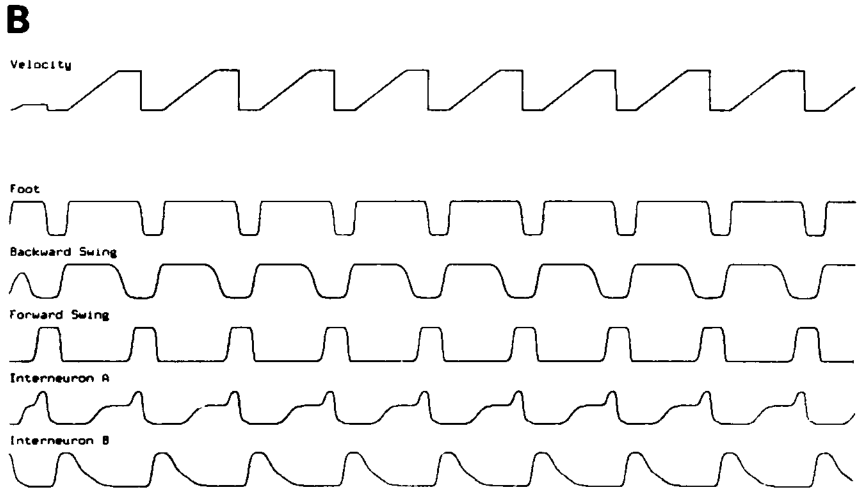


Figure 12
Activity of a typical mixed leg controller (A) with and (B) without its sensor. As in the previous figure, the leg was initialized to 95 percent of its full backward position.

Given these properties, it would be interesting to know how these mixed controllers work. In an attempt to learn this, we examined the operation of a number of different leg controllers and their responses to various lesions. Unfortunately, because these fully interconnected networks were evolved rather than designed, only a few general conclusions could be drawn. First, the interneurons appear to be crucial to the ability of the leg controllers to generate oscillatory outputs intrinsically. Second, leg controllers with access to a sensory signal seem to require fewer interneurons to operate than those without sensors. Finally, the more interneurons that a leg controller has available, the cleaner are its motor outputs (i.e., sharper transitions between stance and swing and better phasing). Thus it appears that, in addition to generating oscillations, the interneurons play an important role in shaping motor outputs.

5.3 Locomotion Controller Results

Once we knew that successful leg controllers could be evolved, we turned to the full locomotion problem. Here the GA must evolve dynamical neural networks that not only produce the appropriate leg control signals but also coordinate the movements of the individual leg controllers in such a way that the insect is continuously supported. Somewhat surprisingly, this problem turned out to be no more difficult than evolving single leg controllers, and full locomotion controllers were evolved under a variety of different conditions.

We successfully evolved full locomotion controllers both with sensors (four trials) and without (five trials). As for single leg controllers, removal of the sensors from a locomotion controller evolved with access to a sensory signal resulted in severe degradation of performance, and rhythmical stepping was often abolished. We were also able to evolve mixed locomotion controllers by averaging the performance of an insect with and without its sensors (two trials). As expected, the locomotion of a mixed controller is generally better with its sensors than without (Figure 13). With its sensors intact, this mixed controller exhibits a higher stepping frequency and cleaner phasing than it does without its sensors. However, even without its sensors, the performance of this controller is still quite good.

In all 11 trials, the GA converged on controllers that generated a pattern of leg movements known as the *tripod gait*. In this gait, the front and back legs on each side of the body swing in phase with the middle leg on the opposite side and exactly out of phase with the other tripod. It should be noted that, because we are optimizing total distance traveled in a fixed amount of time, our performance measure puts pressure not just on the development of locomotion controllers per se but on the development of locomotion controllers that cause the insect to move as quickly as possible. The tripod gait is ubiquitous among fast-walking insects (Graham, 1985).

In all 11 trials, the insects passed through four distinct evolutionary stages. In the first stage, insects appeared that moved forward by planting all six feet and pushing until they fell. Next, insects evolved the ability to swing their legs rhythmically in an uncoordinated fashion. These insects made regular forward progress but fell often. In the third stage, insects utilizing statically stable gaits appeared, but their coordination

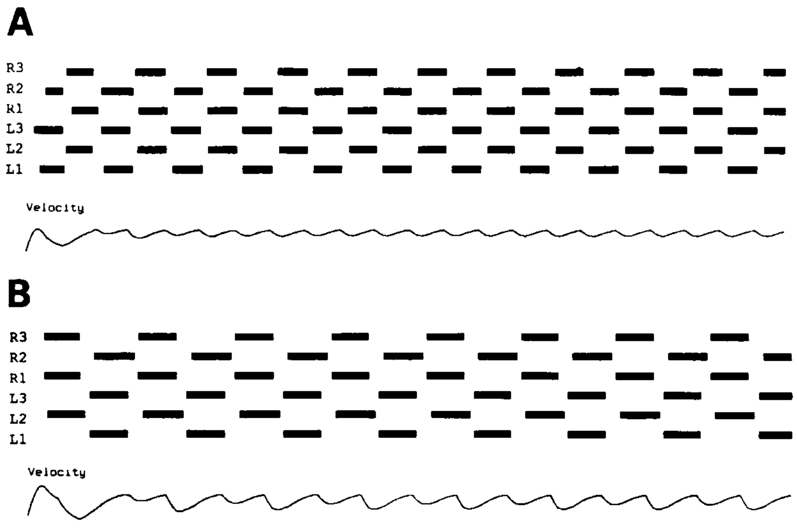


Figure 13
Behavior of a mixed locomotion controller (A) with and (B) without its sensors. The legs are labeled *L* for left and *R* for right and numbered from 1 to 3 starting from the front of the insect. Black bars denote the swing phase of a leg, and the space between bars represents a stance phase.

was still suboptimal. In the fourth and final stage, the efficiency of locomotion slowly improved.

Instead of evolving solutions from scratch, one can take a modular approach. A neural network that solves a complex problem can be evolved by incrementally evolving subnetworks that solve simpler subproblems and then integrating them (de Garis, 1990). The principal advantage of the modular approach is that the evolution is much faster because simpler problems are being solved at each step. To explore this approach, we took six copies of working single leg controllers and evolved only the coupling connections between them on the full locomotion problem. Although adequate solutions were always found, their performance was generally not as good as those locomotion controllers that were evolved from scratch. The modular locomotion controllers tended to have longer transients before the tripod gait was established and were generally less efficient. This is probably due to the fact that these leg controllers were not originally evolved with the full locomotion problem in mind and are therefore less amenable to coupling.

Interestingly, the converse also appears to be true. A leg control network that is evolved in the context of other leg controllers on the full locomotion problem does not, in general, make a very good stand-alone leg controller. In the full locomotion controller experiments we have described here, we found that, unless they are coupled to other leg controllers, the individual leg control networks exhibit inappropriate phasing of motor outputs, and some cannot even oscillate in isolation.

5.4 Related Work

Maes and Brooks (1990) have designed an algorithm that allows a hexapod robot to learn to walk by trial and error. The algorithm receives a binary positive feedback signal from a trailing wheel that turns when the robot moves forward and a binary negative feedback signal from touch sensors on the underside of the robot, which are triggered when the robot falls down. Only the coordination between legs was learned; the individual leg movements were controlled by hand-designed finite-state machines (Brooks, 1989). The swing of each leg is triggered by a conjunction of conditions on the state (up/down/don't care) of the other legs. The learning algorithm operates by exhaustively monitoring the correlations between the actions taken and the feedback received and uses this information to update iteratively the condition vectors of each leg. The end result of this process is that the robot learns to walk with a tripod gait.

The main difference between this work and our approach is that Maes and Brooks have built a great deal more *a priori* structure into their system, including preexisting leg controllers, the fact that the state of the other legs is the important quantity to monitor, and the explicit negative feedback signal generated by falling down. In contrast, we evolve leg controllers as well as their coordination, do not specify what information is to be used in coordinating the legs, and do not include an explicit negative term for falling down in our evaluation function (falling down is only implicitly bad insofar as it affects the ability of the insect to move forward due to the physics of its body).

In other related work, de Garis (1990) has used a genetic algorithm to evolve a neural network for controlling the walking of a pair of two-joint stick legs. At each time-step, the lowest foot was taken to be "down" and the legs were treated as a manipulator rooted at this point. In this model, locomotion is purely kinematic, and no stability problem has to be solved for the legs to walk. The legs were controlled by a 12-node fully-interconnected discrete-time neural network. All neurons were either input or output units; there were no interneurons. The input to the network consisted of the angles and angular velocities of each of the four joints. The outputs of the network were interpreted as angular accelerations. The rightward velocity of the legs was used as the performance measure to be maximized. Network weights were encoded in a fixed number of bits as magnitude plus a sign bit; no Gray-coding was employed. The GA search was driven entirely by mutation; no crossover was used. Using this approach, de Garis was able to evolve successful locomotion controllers. However, good solutions could be obtained only by utilizing an error measure that explicitly computed the difference between the actual and desired output trajectories of the network.

Discussion

In this article, we have shown that continuous-time, recurrent neural networks are a viable control mechanism for adaptive behavior. Relatively small networks were able to solve the problems of chemotaxis and six-legged locomotion and, for the latter behavior, they could do so in a way that was tolerant of sensory damage. In addition, we found examples of single networks that were capable of generating multiple dynamical modes of activity, all of which were behaviorally relevant. It will be interesting to see how well these encouraging results generalize to more complex behaviors.

A number of other control mechanisms for adaptive behavior are being explored by other authors, including condition-action rules (Wilson, 1986; Booker, 1988), combinational logic (Agre and Chapman, 1987; Kaelbling and Rosenschein, 1990), spreading activation networks (Travers, 1988; Maes, 1989), finite-state machines (Brooks, 1986; Collins and Jefferson, 1991), and discrete-time neural networks of both the feedforward and recurrent variety (Collins and Jefferson, 1991; Werner and Dyer, 1991). It is far too early to determine which of these is the "best" substrate for adaptive behavior, and it is possible that no clear winner will ever emerge. However, it is worth pointing out some of the advantages and disadvantages of dynamical neural networks over these other classes of control mechanisms.

A significant problem with some of the other control mechanisms currently being explored is that they are purely functional in nature. The response of the agent at each instant is determined solely by the environmental stimuli it encounters *at that instant.* This is undesirable because such agents are constantly pushed around by their environment. They cannot take any initiative in their interactions and therefore exhibit no true autonomy. This problem is intrinsic to some of the architectures (e.g., feedforward neural networks), but it is merely a frequently taken option with the others. In contrast, because dynamical neural networks maintain state, their responses to identical environmental stimuli can differ at different times. Their activity exhibits a certain "inertia" independent of their immediate environmental context. Internal state is what allows these networks to oscillate in the absence of sensory feedback, for example. Note that this state does not necessarily correspond to a *representation* of anything in particular, any more than the concentrations of reactants in an industrial fractionation column serve to represent anything about the company outside. Both systems simply have time-dependent input-output behavior appropriate to their respective tasks.

A related advantage of dynamical neural networks, particularly for problems involving motor control, is their ability to generate temporally continuous responses (Pearlmutter, 1990). While in principle the state of such mechanisms as finite-state machines and discrete-time neural networks can switch between arbitrary points in their state spaces from one discrete step to the next, continuous-time networks smoothly change their state through time. Note, however, that the behavior of dynamical neural networks is not limited to smooth trajectories, as is illustrated by the sudden switches between klinotaxis and tropotaxis in some of the chemotaxis controllers as sensory stimuli are smoothly varied.

A final advantage of dynamical neural networks is that they have the potential for being directly related to natural nervous systems. The networks described in this article exhibit such biological characteristics as central pattern generation modulated by sensory feedback and discrete switches between distinct modes of activity. By analyzing these networks and examining the conditions under which they evolve, we may be able to gain insight into the operation of their biological analogs. Because the basic form of the neural model is correct, this neurobiological connection could be made much stronger simply by replacing the linear summation and sigmoidal activation function in Equation 1 with more biologically realistic terms.

Dynamical neural networks have some disadvantages as well. Because of their distributed and dynamical nature, these networks can be more difficult to understand and, therefore, to design than some of the other mechanisms. It is also unclear how well dynamical neural networks can cope with discontinuous tasks, such as sequential decision making, though the switches between tropotaxis and klinotaxis are an encouraging step in this direction. More generally, the representational power of continuous-time recurrent neural networks for arbitrary dynamics is currently an open question.

Given a particular choice of control mechanism, a variety of approaches can be taken to developing a controller for a particular task. Within AI, a common approach is to design the required controller by hand (e.g., Brooks, 1986; Agre and Chapman, 1987; Maes, 1989; Beer, 1990). Unfortunately, design methodologies require an analysis of the possible agent-environment interactions that can occur before an appropriate design can be synthesized. Not only can this analysis be very difficult and time-consuming to perform, but it is potentially open to many of the same problems that have plagued more traditional AI approaches to autonomous agent control, because its success is dependent on the ability of the designer to consider all of the possible contingencies. This is simply not practical in realistic environments.

Rather than design an autonomous agent controller by hand, one can attempt to develop the appropriate control mechanisms automatically. Various learning and evolutionary algorithms for this purpose have received a significant amount of attention in recent years. The GA in particular has been widely applied to the problem of developing adaptive agent controllers (e.g., Wilson, 1986; Booker, 1988; de Garis, 1990; Collins and Jefferson, 1991; Werner and Dyer, 1991).

In this article, we have demonstrated that the standard genetic algorithm (GA) can be used to evolve good dynamical neural network controllers for adaptive behavior. This approach has an important advantage over existing neural network training methods in that one need not know the "correct" motor outputs in order to produce an effective neural controller. Instead, one can achieve a kind of design by specification. By manipulating the details of performance evaluation, one can put selective pressure on the development of controllers with desired properties, such as the mixed sensory- central leg controllers. We have also demonstrated that crossover is reasonably effective even when using the naive genetic encoding of network parameters that we employed here. Finally, we have shown that the combination of mutation and selection can be an effective search procedure in its own right when the mutation rate is set appropriately.

Like a number of other authors, we have found that the performance of genetic search on neural network spaces does not scale well with problem size using the naive encoding we employed here (Werner and Dyer, 1991; Collins and Jefferson, 1991). This is hardly surprising given the way that the problems with this encoding (which were discussed earlier) are magnified by longer genetic strings. We have found that some surprisingly sophisticated dynamical behavior can be achieved using relatively small networks whose parameters can be encoded in fewer than 200 bits, and that somewhat larger networks can be encoded in a similar number of bits through the use of symmetries. Larger population sizes can also be used to postpone the problem. However, there is a distinct limit to the size of the network that can currently be evolved using this approach. A great deal of experimentation may be needed to simplify a problem of interest to a point where it can be solved by this approach without trivializing the original problem. For this reason, the exploration of alternate encodings is currently a very active area of research. Ideally, an encoding should allow not only network parameters but also network architecture to be searched. One promising approach is to utilize a simplified model of the developmental processes that construct natural nervous systems. A few tentative steps in this direction have already been taken (Harp, Samad, and Guha, 1989; Miller, Todd, and Hegde, 1989).

Even with better network encodings, evolutionary search processes are not without their limitations. Like any sampling technique, their performance is limited by the statistical properties of the underlying search space. If networks with a desired property are extremely unlikely, or even impossible, with a given neural model and network architecture, then no practical amount of genetic search will succeed in producing one. A better understanding of the generic dynamical properties of continuous-time recurrent neural networks and the relationship of these properties to network architecture would obviously be extremely useful.

One final observation we should make is that, while we have not specified the actual motor outputs that a network should produce in the work described in this article, we have fixed *a priori* the behavior that the agent is to perform (e.g., chemotaxis or locomotion). In natural evolution, no such explicit behavioral specification is provided. Indeed, because a natural agent's environment includes many other agents that are simultaneously evolving, the relationship of a given behavior to reproductive success may itself change over time (i.e., fitness is something intrinsic to natural environments rather than being extrinsically specified). Some promising work on the use of intrinsic fitness in simulated environments has already appeared (e.g., Rizki and Conrad, 1986; Packard, 1988; Werner and Dyer, 1991), and this would seem to be an important direction for future research.

Appendix: Simulation Parameters

All network parameters were encoded in four bits, with time constants in the range $[0.5, 5]$, thresholds in the range $[-4, 4]$, and connection weights in the range $[-16, 16]$. Unless otherwise stated in the text, the following GAucsd parameters were employed.

Chemotaxis Experiments

```
      Total trials  =  48,000
   Population size  =  500
  Structure length  =  96
    Crossover rate  =  0.600000
     Mutation rate  =  0.000200
    Generation gap  =  1.000000
    Scaling window  =  -1
  Structures saved  =  49
 Max gens w/o eval  =  2
           Options  =  cela
      Maximum bias  =  0.990000
   Max convergence  =  0
    Conv threshold  =  0.900000
 DPE time constant  =  25
     Sigma scaling  =  2.000000
```

Locomotion Experiments

Parameters are the same as above except for the following:

```
      Total trials  =  80,000–100,000
  Structure length  =  160–200
     Mutation rate  =  0.000100
 DPE time constant  =  50
```

Acknowledgments

We thank Shawn Lockery for pointing out connections between the evolved strategies of the chemotactic agents and the animal behavior literature. We would also like to thank Leon Sterling, Hillel Chiel, and Leslie Picardo for their comments on an earlier draft of this article. This work was supported by grant N00014-90-J-1545 (to R. D. B.) from the Office of Naval Research. Additional support was provided by the Howard Hughes Medical Institute and the Cleveland Advanced Manufacturing Program through the Center for Automation and Intelligent Systems Research.

References

Agre, P., & Chapman, D. (1987). Pengi: an implementation of a theory of activity. *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 268–272. San Mateo, CA: Morgan Kaufmann.

Anderson, J. A., & Rosenfeld, E. (Eds.). (1988). *Neurocomputing: Foundations of research.* Cambridge, MA: The MIT Press.

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 14–21). Hillsdale, NJ: Lawrence Erlbaum Associates.

Beer, R. D. (1990). *Intelligence as adaptive behavior: An experiment in computational neuroethology.* New York: Academic Press.

Beer, R. D., Chiel, H. J., & Sterling, L. S. (1989). Heterogeneous neural networks for adaptive behavior in dynamic environments. In D. S. Touretzky (Ed.), *Advances in neural information processing systems*, Vol. 1. San Mateo, CA: Morgan Kaufmann.

Belew, R. K., McInerney, J., & Schraudolph, N. N. (1991). Evolving networks: Using the genetic algorithm with connectionist learning. In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen (Eds.), *Artificial life II.* Reading, MA: Addison-Wesley.

Booker, L. B. (1988). Classifier systems that learn internal world models. *Machine Learning, 3*, 161–192.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation, RA-2*, 14–23.

Brooks, R. A. (1989). A robot that walks: Emergent behaviors from a carefully evolved network. *Neural Computation, 1*, 253–262.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence, 47,* 139–159.

Caruana, R., & Schaffer, J. D. (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Proceedings of the Fifth International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Chiel, H. J., & Beer, R. D. (1989). A lesion study of a heterogeneous neural network for hexapod locomotion. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 407–414). IEEE.

Collins, R. J., & Jefferson, D. R. (1991). Representations for artificial organisms. In J. A. Meyer and S. W. Wilson (Eds.), *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior.* Cambridge, MA: The MIT Press.

de Garis, H. (1990). Genetic programming: Building artificial nervous systems using genetically programmed neural network modules. In B. W. Porter & R. J. Mooney (Eds.), *Proceedings of the Seventh International Conference on Machine Learning* (pp. 132–139). San Mateo, CA: Morgan Kaufmann.

Delcomyn, F. (1980). Neural basis of rhythmic behavior in animals. *Science, 210,* 492–498.

Fogel, D. B., & Atmar, J. W. (1990). Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics, 63,* 111–114.

Getting, P. A, & Dekin, M. S. (1985). *Tritonia* swimming: A model system for integration within rhythmic motor systems. In A. I. Selverston (Ed.), *Model neural networks and behavior.* New York: Plenum Press.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Graham, D. (1985). Pattern and control of walking in insects. *Advances in Insect Physiology, 18,* 31–140.

Harp, S. A., Samad, T., & Guha, A. (1989). Towards the genetic synthesis of neural networks. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 360–369). San Mateo, CA: Morgan Kaufmann.

Holland, J. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

Hopfield, J. J. (1984). Neurons with graded response properties have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences of the United States of America, 81,* 3088–3092.

Kaelbling, L., & Rosenschein, S. (1990). Action and planning in embedded agents. *Journal of Robotics and Autonomous Systems, 6,* 35–48.

Langton, C. G., Taylor, C., Farmer, J. D., & Rasmussen, S. (Eds.). (1991). *Artificial Life II.* Reading, MA: Addison-Wesley.

Maes, P. (1989). The dynamics of action selection. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 991–997). San Mateo, CA: Morgan Kaufmann.

Maes, P., & Brooks, R. A. (1990). Learning to coordinate behaviors. *Proceedings of the Eighth National Conference on Artificial Intelligence* [AAAI-90] (pp. 796–802). San Mateo, CA: Morgan Kaufmann.

Maes, P. (Ed.). (1991). *Designing autonomous agents.* Cambridge, MA: The MIT Press.

McFarland, D. (1981). *The Oxford companion to animal behavior.* Oxford, Engl.: Oxford University Press.

Meyer, J. A., & Wilson, S. W. (Eds.). (1991). *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior.* Cambridge, MA: The MIT Press.

Miller, G. F., Todd, P. M., & Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 379–384). San Mateo, CA: Morgan Kaufmann.

Packard, N. H. (1988). Intrinsic adaptation in a simple model for evolution. In C. Langton (Ed.), *Artificial life.* Reading, MA: Addison-Wesley.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation, 1,* 263–269.

Pearlmutter, B. A. (1990). *Dynamic recurrent neural networks* (Technical Report CMU-CS-90-196). Pittsburgh: School of Computer Science, Carnegie Mellon University.

Pearson, K. G. (1976). The control of walking. *Scientific American, 235,* 72–86.

Pearson, K. G., Fourtner, C. R., & Wong, R. K. (1973). Nervous control of walking in the cockroach. In R. B. Stein, K. G. Pearson, R. S. Smith, & J. B. Redford (Eds.), *Control of posture and locomotion.* New York: Plenum Press.

Rizki, M. M., & Conrad, M. (1986). Computing the theory of evolution. *Physica D, 22,* 83–99.

Schaffer, J. D., Caruana, R. A., Eschelman, L. J., & Das, R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 51–60). San Mateo, CA: Morgan Kaufmann.

Schraudolph, N. N., & Belew, R. K. (1990). *Dynamic parameter encoding for genetic algorithms* (Technical Report CS 90-175). San Diego: Cognitive Computer Science Research Group, Computer Science and Engineering Dept., University of California at San Diego.

Selverston, A. I., & Moulins, M. (Eds.). (1987). *The crustacean stomatogastric system.* Berlin: Springer-Verlag.

Travers, M. (1988). Animal construction kits. In C. Langton (Ed.), *Artificial life.* Reading, MA: Addison-Wesley.

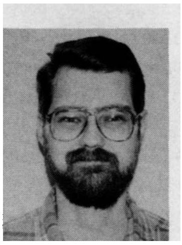
Werner, G. M., & Dyer, M. G. (1991). Evolution of communication in artificial organisms. In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen (Eds.), *Artificial life II.* Reading, MA: Addison-Wesley.

Whitley, D., & Hanson, T. (1989). Optimizing neural networks using faster, more accurate genetic search. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 391–396). San Mateo, CA: Morgan Kaufmann.

Wieland, A. P. (1990). Evolving controls for unstable systems. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, & G. E. Hinton (Eds.), *Connectionist models: Proceedings of the 1990 Summer School* (pp. 91–102). San Mateo, CA: Morgan Kaufmann.

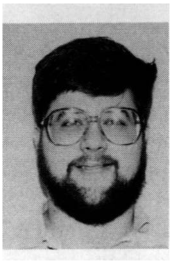
Wilson, S. W. (1986). Knowledge growth in an artificial animal. In K. S. Narendra (Ed.), *Adaptive and learning systems.* New York: Plenum Press.
Winograd, T., & Flores, F. (1986). *Understanding computers and cognition.* Norwood, NJ: Ablex.

About the Authors



Randall D. Beer

Randall D. Beer received his Ph.D. in computer science from Case Western Reserve University in 1989. He is currently an assistant professor at CWRU in the departments of computer engineering and science, and biology. He is interested in the neural control of natural animal behavior and the application of biological control principles to problems in autonomous agent control.



John C. Gallagher

John C. Gallagher received his M.S. in computer science from Case Western Reserve University in 1991, and currently is pursuing his Ph.D. in the same department. He is interested in the application of genetic algorithms to the design of neural network controllers.