Efficient Credit Assignment through Evaluation Function Decomposition

Adrian Agogino
UC Santa Cruz, NASA Ames Research Center
Mailstop 269-3
Moffett Field, CA 94035, USA
adrian@email.arc.nasa.gov

Kagan Tumer
NASA Ames Research Center
Mailstop 269-4
Moffett Field, CA 94035, USA
ktumer@mail.arc.nasa.gov

Risto Miikkulainen
Computer Sciences Dept., The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-1188, USA
risto@cs.utexas.edu

ABSTRACT

Evolutionary methods are powerful tools in discovering solutions for difficult continuous tasks. When such a solution is encoded over multiple genes, a genetic algorithm faces the difficult credit assignment problem of evaluating how a single gene in a chromosome contributes to the full solution. Typically a single evaluation function is used for the entire chromosome, implicitly giving each gene in the chromosome the same evaluation. This method is inefficient because a gene will get credit for the contribution of all the other genes as well. Accurately measuring the fitness of individual genes in such a large search space requires many trials. This paper instead proposes turning this single complex search problem into a multi-agent search problem, where each agent has the simpler task of discovering a suitable gene. Gene-specific evaluation functions can then be created that have better theoretical properties than a single evaluation function over all genes. This method is tested in the difficult double-pole balancing problem, showing that agents using gene-specific evaluation functions can create a successful control policy in 20% fewer trials than the best existing genetic algorithms. The method is extended to more distributed problems, achieving 95% performance gains over tradition methods in the multi-rover domain.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Multiagent systems*; I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Connectionism and neural nets*

General Terms

Algorithms, Performance

Keywords

Multiagent Systems, Genetic Algorithms, Neural Networks

GECCO'05, June 25–29, 2005, Washington, DC, USA.

1. INTRODUCTION

A critical step in a genetic algorithm's (GA) discovery process is the fitness evaluation of a chromosome. As an example consider how to evaluate a chromosome used to control a planetary rover. This evaluation can be done by measuring the performance of the rover over a series of trials. For example, suppose a chromosome, C_1, was used 100 times to control the rover, and the rover crashed 50 times. Then a chromosome, C_2, was used 100 times to control the rover, and the rover crashed 30 times. We can say from these tests that chromosome C_2 has a higher fitness than chromosome C_1. However, how can a single gene be evaluated? Suppose a gene, G_1, was part of 100 different chromosomes that were tested, and the rover crashed 50 times. In addition a gene, G_2, was part of a new set of 100 different chromosomes that were tested, and the rover crashed 30 times. Can we say that gene G_2 has a higher fitness than G_1? Not with the same confidence as saying C_2 has higher fitness than C_1, particularly if the chromosome consists of many genes. This is because on average, the choice of gene G_1 or G_2 will likely have an impact of $\frac{1}{n}$ on the evaluation where n is the number of genes. When there are many genes, there is a significant chance that gene G_1 was just unlucky and had been tested with chromosomes consisting of many highly unfit genes. To find out the impact of G_1 requires many more evaluations than to find out the impact of an entire chromosome. The difficulty here arises from using rover crashing, which is inherently a function of an entire chromosome, as evaluation for a single gene.

This paper shows that it is often possible to evaluate genes individually by using evaluation functions designed for self-organizing multi-agent systems: The task of each agent is to discover a highly fit gene, and these genes are put together to form a chromosome. These agents can be evolved more simply than a full genetic algorithm since they are finding only a single gene and can often use simple evolutionary algorithms without crossover. While each agent makes independent choices on how to choose a gene to best maximize its evaluation function, the choices of the genes are strongly coupled through the evaluation function. Implicitly, each agent's choice depends the gene choices of the other agents.

For these gene evaluation functions to be most effective, a system needs to have a certain amount of stability so that the fitness of an agent's choice of gene is unlikely to be completely different from trial to trial. A simple example where this is true is a resource summation problem, where each

gene defines a bit and the goal of the problem is to have the bits sum to a value within a fixed range. At the beginning of the evolutionary process, each gene evaluation function is not very accurate, since it does not know the bit-choices of all the other agents. However, once the bit choices of the agents begin to stabilize, the gene evaluation function can give a good signal to the agent for its choice of bit.

Note that when this problem is changed to a parity problem, then this process is not possible since any change in any bit affects the evaluation of all the other bits and the system can never converge. However, parity problems (including XOR problem) are not representative of most real-world control domains which have more in common with the summation problem [3]. This paper presents two such problems, where even though the effects of each gene are coupled, the multi-agent system can effectively produce a set of genes that lead to high evaluation without diverging or falling into local minimum.

Section 2 explains this multi-agent system in more detail, showing how it maps to a genome and how genes are generated. Section 3 summarizes principles of multi-agent evaluation functions that can be used to evaluate genes. Section 4 discusses issues with computing these multi-agent evaluation functions in Markov Decision Processes. Section 5 discusses what neural network controller works best with gene-specific evaluations depending on how coupled the inputs are. Section 6 shows how a multi-agent system can be used in domains with highly coupled state variables by discovering weights to a radial basis function network that solves the difficult double-pole-balancing problem more quickly than the best existing genetic algorithms. Then Section 7 shows how a multi-agent system is effective in a domain with more loosely coupled state variables by discovering a superior multi-layer perceptron used to control a set of rovers.

2. MULTI-AGENT SYSTEM STRUCTURE

This paper proposes creating high-fitness chromosomes that have n genes, using a multi-agent system with n agents. Each agent is mapped to a single gene-position and is responsible for creating a gene for a single position on the chromosome. Each agent stores a population of genes and uses a simple evolutionary algorithm to improve the fitness of the population over a series of trials, as shown in Figure 1.

At the beginning of every trial, each agent chooses a gene from its population. All their genes are then concatenated to form a chromosome, defining a phenotype such as a neural network. The trial is conducted by using this neural network as the controller for a Markov Decision Process, until the process terminates. Information from the trial is then used to evaluate each gene, using evaluation techniques from multi-agent systems (Section 3). Each agent then uses its evaluation to modify its population using an evolutionary algorithm.

This paper will use a simple evolutionary algorithm that removes the worst performing gene from the population at the end of the trial and replaces it with a mutated version of the best gene. In this architecture even this simple evolutionary method was found to perform well in the test domains and allows this paper to focus on evaluation functions instead of the sophistication of evolutionary algorithms. Note however that almost any evolutionary algorithm can be used in this system. Each agent can even use its own set of parameters or its own search algorithm.

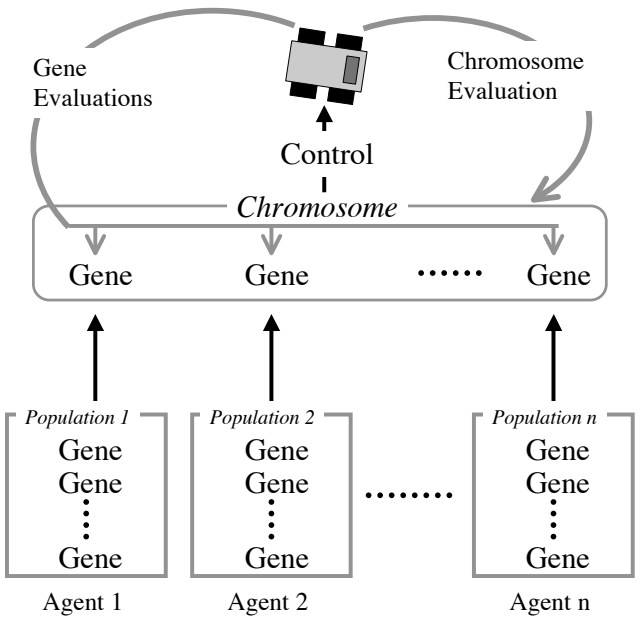


Figure 1: Multi-agent System Producing Chromosomes for a Control Problem. At the beginning of a trial, each agent chooses a gene, which encodes a part of the solution (i.e. the weights for a single hidden node of a neural network). The choice of all the agents forms a chromosome representing an entire solution. This solution (i.e neural network) is then used as a controller. Standard genetic algorithms will evaluate entire chromosome after trial. Agents instead evaluate only the contribution of their gene, leveraging evaluation methods used in self-organizing multi-agent systems. Gene evaluation can be more efficient allowing a solution to be found in fewer trials.

3. MULTI-AGENT SYSTEM EVALUATION FUNCTIONS

Finding a fitness evaluation function for a single gene that can be used by the agents described in Section 2 is a difficult problem. Interestingly, this same credit assignment problem is found in multi-agent systems: how to give credit to an individual agent's action when the multi-agent task depends on the actions of all the agents. This section will outline a solution to this problem coming from the theory of collectives [16], in the specific context of the multi-agent system in Section 2.

Mathematically, the goal of a genetic algorithm is to maximize a global fitness evaluation function $G(z)$, which is a function of a chromosome z. This chromosome is broken down into n genes:

$$z = (z_1, z_2, ..., z_n) \; . \tag{1}$$

Maximizing $G(z)$ is also the goal of the multi-agent system. However each agent will not try to maximize $G(z)$ directly. Instead, each agent maximizes its gene evaluation function

$g_i(z)$, where i is the index specifying the agent. Note that while $g_i(z)$ is used to evaluate a single gene it is still a function of *all* the genes. This property couples the gene agents, enabling a global solution.

3.1 Factoredness and Learnability

For the multi-agent system to achieve high values of the global evaluation function G, the gene evaluation functions need to have two properties, called **factoredness** and **learnability** [14]. First the gene evaluation functions of each agent should be factored with respect to G, intuitively meaning that when an agent choses a gene that improves its gene evaluation function, this choice also improves the global evaluation function (i.e. G and g_i are aligned). Also when an agent choses a gene that reduces G, it should also reduce g_i. Formally an evaluation function g_i is factored when:

$$g_i(z) \geq g_i(z') \Leftrightarrow G(z) \geq G(z') \quad \forall z, z' \ s.t. \ z_{-i} = z'_{-i} \ ,$$

where z_{-i} and z'_{-i} contain the genes not chosen by agent i. In game theory language, the Nash equilibria of a factored system are local maxima of G [14, 11]. In addition to this desirable equilibrium behavior, factored systems also automatically provide appropriate off-equilibrium incentives to the agents.

In addition to being factored, the agents' gene evaluation functions should be highly learnable, intuitively meaning that they should be sensitive to the agent's own choice of gene and insensitive to the choices of other agents. For a given chromosome z, the higher the learnability, the more $g_i(z)$ depends on the gene choice of agent i, i.e., the better the associated signal-to-noise ratio for agent i. Note that the learnability of $g_i(z)$ system is not directly related to epistasis (i.e. multiple genes controlling one phenotype), since the system designer is free to choose a $g_i(z)$ that is not directly related to a phenotype. However, typically it is easier to create a $g_i(z)$ that is factored and highly learnable in genomes that exhibit low epistasis.

When the same global evaluation G is used for all the gene evaluations, each gene evaluation is by definition factored. This use of the global evaluation is common in many multi-agent systems [14]. In addition most genetic algorithms either explicitly or implicitly use this global evaluation to evaluate genes. However in a large system, an agent will have a difficult time discerning the effects of its choice of gene on G, since G is a function of the actions (i.e. gene choices) of all of the agents in the system. As a consequence, each agent i has difficulty achieving high g_i (i.e. G has low learnability).

3.2 Difference Evaluation Functions

It is desirable for an agent's gene evaluation function to be factored and to have high learnability. Factoredness assures that an agent will try to produce genes that maximize the fitness of the chromosome. Having a highly learnable evaluation function will reduce the number of trials that are needed for an agent to achieve this high level of fitness. One evaluation function that is factored, yet still highly learnable (unlike G) is the difference evaluation function, defined as follows:

$$D_i(z) = G(z) - G(z_{-i} + c_i) \ , \tag{2}$$

where z_{-i} contains all the genes chosen by agents other than agent i. The gene z_i chosen by agent i is replaced with the fixed constant c_i. Such difference evaluation functions are factored no matter what the choice of c_i because the second term does not depend on agent i's choice of a gene [14, 15]. Furthermore, they usually have far better learnability than does $G(z)$ because the second term of D_i removes a lot of the effect of other agents (i.e., noise) from agent i's evaluation function [14]. This evaluation function has proven effective in many multi-agent system domains including network routing, job scheduling and control [16, 14]. Because it is effective, this paper will focus on using the difference evaluation, D_i, as the gene evaluation g_i.

While this paper focuses on finding chromosomes that encodes neural networks as an example, a multi-agent system using the difference gene evaluation can be used to find almost any type of chromosome that is separated into genes. However, in many domains where the functionality is less distributed among the genes, D_i can be difficult to evaluate and great care has to be given to approximate D_i in such a way that it retains its high learnability. Section 4 discusses the issues with computing D_i, when the chromosome describes a controller used in a Markov Decision Process where there can be stronger coupling between genes. Addressing these issues are needed to apply D_i to such domains as pole balancing.

4. DIFFERENCE EVALUATION FOR MDPS

A Markov Decision Process (MDP) represents an important class of control problems, where a decision maker bases its action on its current state without a need to know its previous actions or states [9]. The problems of pole balancing, robot navigation and rover control all can be represented as MDPs. In this paper the decision maker in the domain is called an MDP-agent (not to be confused with an agent in the multi-agent system that selects genes) and uses a neural network to map states into actions. A neural network is used as genetic algorithms combined with neural networks have been shown be effective in finding solutions to continuous control tasks, such as pole balancing, robot navigation, rocket control and rover control [13, 10, 6, 4, 5].

At every time step in an MDP control problem, the state of the MDP-agent is fed into the input of its neural network, and the action of the agent is determined from the output of the neural network. After the MDP-agent takes an action it receives a reward and moves to a different state. Both the reward and the new state are functions of the action and the previous state.

At the beginning of a trial, the MDP-agent starts in a start-state and over the course of the trial takes T actions, receives T rewards and enter T states. The goal of the system is to maximize the sum of rewards received during a trial. Given the agent's start-state, this sum of rewards completely depends on the neural network it uses. The remainder of this section will show how a multi-agent system can be used to produce a neural network that performs well in an MDP.

Since the goal of the MDP is to maximize the sum of rewards received during a trial, this sum is used as the global fitness evaluation function for the multi-agent system:

$$G(z) = \sum_t R_t(z) \ , \tag{3}$$

where $R_t(z)$ is the reward received at time step t, and z is the chromosome defining the neural network used by the MDP-

agent. Note that this equation assumes a fixed start-state, which enables each reward to be represented as a function of a chromosome. Given the global fitness function, the difference gene evaluation function is:

$$\begin{aligned} D_i(z) &= G(z) - G(z_{-i} + c_i) \\ &= \sum_t R_t(z) - \sum_t R_t(z_{-i} + c_i) \ . \end{aligned} \quad (4)$$

This is the function that each agent uses to evaluate the gene it chose at the beginning of the trial.

When there is a closed form mathematical formula representing the global evaluation as a function of genes, then computing the difference evaluation can the simple. In fact computing the difference evaluation is often easier than computing the global evaluation since many of the terms in the global evaluation can cancel out with the substraction. However, in most complex domains there is no explicit formula for $G(z)$ as a function of the chromosome. In these cases the global evaluation is typically computed as a function of states (e.g. the rover is in a crashed state therefore give it low evaluation) or measured directly from the environment. With no explicit formula for $G(z)$, computing the second term of the difference gene evaluation, $G(z_{-i} + c_i)$, may be difficult. Recall from Section 3 that $G(z_{-i}+c_i)$ returns what the global evaluation would be if agent i's choice of gene were changed to an arbitrary gene c_i. Without knowing the function form of $G(z)$, in general, computing $G(z_{-i} + c_i)$ would necessitate running an entire trial using the chromosome $z_{-i} + c_i$. This computation would have to be done for every agent i. While computationally difficult in simulated environments, the computation of $G(z_{-i} + c_i)$ would often be completely impractical in real environments. For example consider the rover-control domain where a rover is controlled by a neural network defined by 100 genes. After a single rover test, the rover would have to be tested 100 more times just to compute the gene evaluation functions for the initial test.

To overcome the difficulties in computing $G(z_{-i} + c_i)$, an estimate can be made by determining which rewards received during a trial were affected by agent i's choice of gene. Recall that $G(z_{-i}+c_i)$ is a sum of rewards: $\sum_t R_t(z_{-i}+c_i)$. Also the reward for time step t is a function of the action at time step t and all the previous actions (while the reward is Markovian, being a function of only the current action and state, the current state is implicitly a function of all the pervious actions taken to get there). Since the actions are the output of the neural network, they are functions of the chromosome. Therefore a reward $R_t(z_{-i}, c_i)$ can be represented as:

$$R_t(z_{-i} + c_i) = R_t(a_1(z_{-i} + c_i), \cdots, a_t(z_{-i} + c_i)) \ , \quad (5)$$

where $a_t(z_{-i}, c_i)$ is the action taken at time t. Therefore if agent i's choice of gene does not significantly affect any action before time t, it should not significantly affect any reward before time t. Let the level of how much a gene affects an action be formally defined as the *action sensitivity* at time t :

$$L_{i,t}(z) = \frac{\delta a_t(z_{-i} + z_i)}{\delta z_i} \ , \quad (6)$$

In addition define T_i as the first time step in which $L_{i,t}(z)$ is greater than a threshold τ. For all $t < T_i$, the choice of z_i has little influence on the MDP-agent's moves and therefore the MDP-agent's rewards. The values of $R_t(z_{-i}, c_i)$ can then be approximated as $R_t(z)$ for all $t < T_i$. For time steps after T_i, the value of $R_t(z_{-i}, c_i)$ is unknown. As a first approximation, these unknown reward values can be set to zero. An agents difference gene evaluation function $D_i(z)$ can then be approximated as follows:

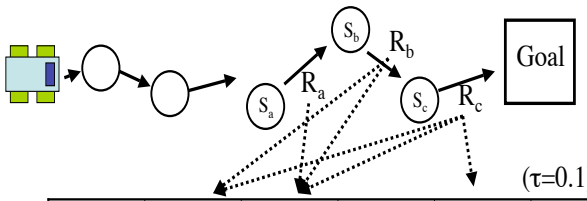
$$\begin{aligned} \hat{D}_i(z) &= \sum_t R_t(z) - \sum_{t<T_i} R_t(z) \quad (7) \\ &= \sum_{t \geq T_i} R_t(z) \ . \quad (8) \end{aligned}$$

Note that $\hat{D}_i(z)$ is only an approximation to $D_i(z)$ since the unknown rewards in the second term of $D_i(z)$ are set to zero. While this may be a rough approximation, $\hat{D}_i(z)$ is still factored because its approximation of the second term is not affected by the actions of agent i. Instead the approximation potentially reduces the evaluation function's learnability, causing the system to converge more slowly. However, the results show in Sections 6 and 7 show that the use of $\hat{D}_i(z)$ still leads to high performance.

Figure 2 shows how action sensitivity can be used. In this figure only certain genes influence a rover's action at any given state. Therefore genes that do not influence the actions of the rover at or before that state do not need to be given credit for rewards received at that state. This reward structure can be used to improve the learnability of gene evaluations. Note that the setting of the threshold τ moves the tradeoff between factoredness and learnability. When τ is very small, the difference evaluation function is almost always factored since it includes all the rewards an agent influences by even the smallest amount. However it has low learnability since T_i is close to zero making the difference evaluation almost the same as the global evaluation. For example in 2 if τ were set to 0.0005 instead of 0.1 than all of the genes would get credit for most of the rewards. In contrast, when τ is large, the difference evaluation is very learnable since many of the rewards are removed from the evaluation, but it can be very far from being factored since the agent could have a significant influence over many of the removed rewards. For example in 2 if τ were set to 0.9 instead of 0.1 only Gene 2 would get credit for any of the rewards even though Genes 1 and 4 had significant contrabutions. In addition to the setting of τ, the value of T_η is also highly dependent on the type of neural network used. This issue is explored in the next section.

5. NEURAL NETS FOR MDP CONTROL

In the discussion so far, the controller has been assumed to be a neural network, but its type has not been specified. However, the type of network influences the value of T_η, the first time step that an agent's choice of gene significantly affects the output of the neural network. The value of T_η is important because it is used to estimate $D_\eta(z)$. If T_η tends to be close to the first time step for most agents, then the value the second term of $D_\eta(z)$ will be close to zero and $D_\eta(z)$ will essentially be the global evaluation function. While $D_\eta(z)$ would still be factored, none of the learnability advantages of gene evaluation functions will be achieved. This section compares Multi-Layer Perceptrons and Radial Basis Function Networks with respect to their effect on the value of $T(\eta)$.



(τ=0.1)

State/Gene Action Sensitivity

	Gene 1	Gene 2	Gene 3	Gene 4	Gene 5
S_a	0.001	**0.998**	0.001	0.011	0.003
S_b	**0.421**	**0.745**	0.001	0.020	0.003
S_c	0.001	0.002	0.000	**0.890**	0.023

Figure 2: Action Sensitivity. A rover with a policy defined by 5 genes moves through different states to a reach a goal. Action sensitivities seen in the final three states shown in the table. Only Gene 2 influences rover's action in state S_a above the threshold τ so it is the only gene that gets credit for reward R_a. Genes 1 and 2 influence rover's action in state S_b so both get credit for R_b. Gene 4 influences rover's action in state S_c so it gets credit for R_c. Since the actions taken at states S_a and S_b are responsible for the rover entering state S_c, the reward R_c must also be propagated back to Gene 1 and 2 for their contributions.

5.1 Multi-Layer Perceptron

Consider a two-layer Multi-Layer Perceptron (MLP) [7] where each gene, z_η, determines one of the network weights. At each time step the current state is fed into the input layer of the MLP, and the action for the agent in that state is taken from the output layer. For MLPs with sigmoid activation functions, the output of the network is:

$$a(s) = g(\sum_i w_i \phi_i(s)) \ , \tag{9}$$

where $\phi_i(s)$ is the evaluation for hidden unit i and $g(x)$ is the sigmoid function, $\frac{1}{1+e^{-x}}$. The action sensitivity for a time t is therefore:

$$\begin{aligned} L_{\eta,t}(z) &= \frac{\delta a_t(z_{-\eta}, z_\eta)}{\delta z_\eta} \\ &= g(\sum_i w_i\phi_i(s_t))(1 - g(\sum_i w_i\phi_i(s_t)))\frac{\delta \sum_i w_i\phi_i(s_t)}{\delta w_\eta} \\ &= g(\sum_i w_i\phi_i(s_t))(1 - g(\sum_i w_i\phi_i(s_t)))\phi_\eta(s_t) \ . \end{aligned}$$

Note that the action sensitivity will be low either when the network is saturated or the activation of the hidden unit is low. If the network is saturated, little information is going to be gained from the trial, and some external mechanism will have to be applied to get the system out of saturation. If the input layer is fully connected to the hidden layer, the output of a hidden unit will rarely be very low, so that the first time an agent's action has significant impact on the output of the network is likely to be very early in the trial. Therefore the value of T_η is likely to be close to zero for most agents and $D_\eta(z)$ will have nearly as low learnability as the global evaluation function. However, in many domains such as distributed rover exploration, a loosely connected MLP can be used where the action sensitivity of a hidden unit is high in only a few states. In these domains the gene evaluation function, $D_\eta(z)$, is likely to have high learnability when used these MLPs.

5.2 Radial Basis Function Networks

In domains with highly coupled state variables, such as pole balancing, it is difficult to construct an MLP where hidden nodes only react to small regions of the state space. In such domains radial basis function network (RBFN) can be used as an alternative, since they naturally have hidden nodes that react to small portions of the state space [3]. Like the MLP, the state is fed into the input layer of the RBFN and the action is determined by the output of the RBFN. Consider a standard RBFN with n bases with fixed width d. The output of the RBFN is a linear sum of the basis activations:

$$a(s) = \sum_\eta w_i \phi_\eta(s) \ , \tag{10}$$

where $\phi_\eta(s)$ is the basis function and weight w_η is the action of agent η. For RBFNs, the action sensitivity at time t is simply equal to $\phi_\eta(s_t)$, the activation of the basis function. RBFNs typically use gaussian activation functions of the form:

$$\phi_\eta(s) = e^{\frac{1}{2}(s-c_\eta)^2/d^2} \ , \tag{11}$$

where c_η is the centroid of the basis function. Due to the localized nature of this type of activation function, one can expect that the value of $L_{\eta,t}$ will be very low for most states. Only states that are close to the centroid will produce significant activation. Therefore T_η will be equal to the time step that the MDP entered a state that was close to the centroid ϕ_η. In many cases T_η will not be close to zero and the value of $D_\eta(z)$ will be significantly more learnable than the global evaluation function. This increased learnability arises from the rewards that were not influence by agent η's choice of gene being removed from $D_\eta(z)$.

6. DOUBLE POLE BALANCING

This paper shows results for two very different domains: double pole balancing and rover exploration. In the first domain the state variables are highly coupled: the value of a single state variable has little meaning out of the context of the other state variable. Despite the coupled state variable, this section shows how gene-specific evaluations are effective when used with RBFNs. This method is then compared against the best known evolutionary and genetic algorithms that have been previously applied to this problem.

6.1 Problem Description

In this problem there is a cart that can move along one axis (Figure 3). Two poles of different lengths are attached to the cart, and can pivot at the attachment point. The controller can apply a positive or negative force to the cart. The goal of the controller is to keep the two poles from falling while keeping the position of the cart within fixed bounds. The state space consists of six values: the position and velocity of the cart, and the angles and angular velocities of the two poles. At each time step a reward of 1 is received. The trial ends when either the angle of either pole or the cart position goes outside of bounds. Note that these state

variables are highly coupled. For instance knowing the velocity of the cart has little value if it is not known if the pole is falling to the left or to the right.

The learning algorithms were evaluated based on the number of trials that needed to be completed before a solution could be found that balanced the poles for 50,000 time steps [1]. In this particular problem, the length of one of the poles was one meter and the other was one tenth of a meter. The time resolution was 20 milliseconds. For all five algorithms, the same code base was used to simulate the pole (code can be downloaded from: "www.cs.utexas.edu/ users/ nn/ downloads/ software/ javaesp.1.0.tar.Z").

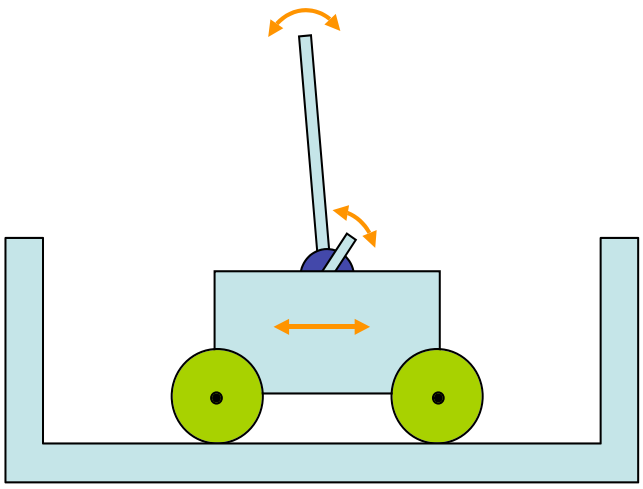


Figure 3: The Double Pole Balancing Problem. A cart with two poles can move with one degree of freedom and each pole can rotate with one degree of freedom. Since the poles have different lengths, they respond differently to a force on the card and both poles can be balanced indefinitely.

The controller was an RBFN with six input units and one output unit. The basis functions were added dynamically to cover the input space. When a state was entered where the activation of all of the bases was less than 0.1, a new basis was added, centered in that state. In a typical problem several hundred basis functions were created. At every time step the six values of the state space were fed into the RBFN and its output determined the force applied to the cart. The controller RBFN was encoded by a chromosome produced by a multi-agent system. In one set of experiments the difference evaluation function, D_i, was used by the agents to evaluate their choice of gene. The action sensitivity τ used in computing D_i was set to 0.1. In a second set of experiments the global fitness evaluation was used by the agents to evaluate their choice of gene.

Based on the evaluation function, the agents made their choice of gene using a very simple evolutionary algorithm, based on a population of ten weights. Each agent starts with a random population based on identical distributions, but through time each population converges to a different distribution. At the beginning of a trial, an agent would select the most fit weight 90% of the time and a random weight 10% of the time. At the end of the trial, it would evaluate its choice of weight based on its gene evaluation function. It would then remove the worst performing weight from its population and replace it with a mutated copy of the best performing weight. The mutation was done using the Cauchy Distribution (with scale parameter equal to 0.3). With time each population tends to slowly converge to a set of similar genes. This convergence allows self-organization to take place: while the agents make diverse choices of genes in early trials, they can later begin to refine their choices based on the choices of the other agents.

[1]Experimental results show that if the poles can be balanced for 50,000 time steps they can be balanced indefinitely. Results for any value above 10,000 time steps are similar.

Algorithm	Average Trials	Deviation in Mean
SANE	12,600	
ESP	3,800	
NEAT	3,578	257
RBFN (G)	4,025	178
RBFN (D_i)	2,815	91

Table 1: Effectiveness of Gene-evaluations in Double Pole Balancing Problem. The multi-agent system evolving RBFN controllers and using D_i for gene evaluations finds a solution in 20% fewer trials than best previously existing algorithm. All differences from RBFN (D_i) are statistically significant assuming unreported variances are similar ($p < 0.005$).

6.2 Results

The results averaged over for 400 runs are shown in Table 1. The RBFN using a global evaluation function performs almost as well as the two existing high performance algorithms, ESP and NEAT [6, 13]. This is to be expected since these algorithms have some features in common. Similar to ESP, the multi-agent system evolves separate "subpopulations." It is also related to the speciation in NEAT, since each agent evolves specialized populations.

However, the results for using D_i are significantly better than for G. This increased performance can be expected since the D_i for an agent eliminates the reward values that the agent could not possibly influence, therefore giving it a cleaner signal. When G is used, each basis function gets credit for every single reward received during a trial. Even if a basis function does not influence a single action in a trial, using G will give it credit for all the rewards when it should receive credit for none. In addition even when a basis function does influence an action it should not receive credit for rewards that happened before it had any influence. This poor credit assignment of G adds noise to the system, making it difficult to discern how a well a particular gene choice is performing. The use of D_i eliminates all of the rewards that a gene choice could not influence, reducing the noise in the evaluation, allowing the system to converge in a fewer number of trials.

7. ROVER EXPLORATION PROBLEM

This section shows how gene-specific evaluations are also effective in the rover exploration problem, a domain where the state variables are not nearly as coupled as in the double-pole balancing problem. Simple canonical evolutionary algorithms are used to highlight the importance of the evaluation function and to show that genes can be evaluated separately without suffering from convergence to local minima.

7.1 Problem Description

This section summarizes the distributed rover exploration problem described in detail in [1]. In this problem, a set of ten rovers explores a finite two dimensional world, trying to observe interesting rocks distributed throughout the domain. The global evaluation function for a trial is given by:

$$G = \sum_t \sum_i \frac{V_i}{\min_\eta \delta(L_i, L_{\eta,t})} , \quad (12)$$

where V_i is the value of rock i, L_i is the location of rock i and $L_{\eta,t}$ is the location of rover η at time t, and $\delta(x, y)$ is the euclidean distance[2]. To maximize the global evaluation, rovers should tend to navigate towards rocks with high values, but they should also avoid congestion, since having multiple rovers observe the rock will not increase the global evaluation.

Each rover has eight input sensors divided up into four quadrants. In each quadrant there is one sensor detecting other rovers in that quadrant and one sensor detecting rocks in that quadrant. Each rover is controlled through a two dimensional number determining the direction and magnitude in which the rover will move during the current time step. The set of rovers are controlled with a single one hundred hidden-node multi-layer-perceptron with eighty inputs and twenty outputs, corresponding to the inputs and outputs of the ten rovers. This network is loosely connected, with the two outputs for each rover being only dependent on the eight inputs for the rover (the network has 10% the number of connections as a fully connected network). One gene determines all of the weights between the layers of the eight inputs, ten hidden units and two outputs corresponding to a rover. Here the output of the neural network is a linear combination of gene choices. However, the gene choices are still coupled since in general the global evaluation will not be a linear combination of the gene choices.

The performance of five different evolutionary and GA methods was tested in this domain. In all of the methods, a trial starts with the best member of a population being chosen with 90% probability and a random member being chosen with 10% probability. This member of the population is then evaluated during the trial. Weight mutation is performed by adding a value sampled from the Cauchy Distribution (with scale parameter equal to 0.3). Specifics of the methods are as follows:

1. **EC** : Traditional evolutionary computation without crossover using the global evaluation. At the end of the trial, the worst member of the population is replaced with a mutated version of the best member.

2. **GA**: Traditional evolutionary computation with point crossover using the global evaluation. At the end of the trial the worst member of the population is replaced with a combination of mutated versions of the best member and second best member of the population.

3. **G Agents**: A ten-agent multi-agent system where each agent is responsible for choosing a gene based on the global evaluation. Each agent makes its choice of gene using an evolutionary algorithm without crossover.

4. **D Agents**: A ten-agent multi-agent system where each agent is responsible for choosing a gene based on the *difference evaluation*. Each agent makes its choice of gene using an evolutionary algorithm without crossover.

5. **D Agents HC**: Same as **D Agents**, but employs strict monotonic hill-climbing to make its choice of gene. This method is used to test if agents choosing their genes through evolutionary methods can overcome local minimum that cause monotonic hill-climbers to perform poorly.

Each algorithm used a population of size ten. For "EC" and "GA" a member of a population consisted of the full one hundred hidden unit neural network. Instead for "G Agents", "D Agents" and "D Agents HC" each agent had a population of ten hidden nodes.

[2]When the distance become close, $\delta(x, y)$ is set to a constant to avoid singularities

7.2 Results

Results averaged over 200 runs show that the three methods using the global evaluation function all perform about the same (see Figure 4). Traditional GAs with crossover, traditional GAs without crossover and agent based GAs using G, all evolve very slowly. This result can be expected since the global evaluation does a poor job in assigning credit to the individual genes causing all evolutionary algorithms to suffer. Even though each gene is contributing to only a small portion of the global evaluation, it is receiving full credit for the evaluation. If a gene has high fitness it may be given a low evaluation if the other genes in the system have low fitness. This is true even if the other genes have only slightly lower fitness on average since the impact the single gene is small. In contrast, when agents use the difference evaluation, they evolve very quickly. Their evaluation function provides a clean signal indicating how effective an agent's choice of gene is.

While each agent is choosing genes for one part of the chromosome, agents using factored evaluation functions still perform global search. They all attempt to maximize an evaluation that is a function of all the genes. In addition when they use evolutionary algorithms with populations they can break out of local minima and can find globally high performance solutions. This conclusion is confirmed when comparing their performance to that of the monotonic hill climbers. The hill climbing agents cannot climb out of local minima and Figure 4 shows that their performance is significantly worse than for agents using evolutionary algorithms with the same evaluation function.

8. DISCUSSION AND FUTURE WORK

The results show that a multi-agent system can discover superior solution, especially for control problems in continuous domains. Even though each agent is extremely simple, the multi-agent system collectively converges on a complex global solution. However, the multi-agent system can be improved in several ways. For instance, in the experiments, each agent used a very simple evolutionary algorithm to choose a gene. Almost any evolutionary algorithm could be used instead, including genetic algorithms with crossover. Each agent could even adapt its own learning parameters individually (paralleling Evolutionary Strategies, but with agent-specific evaluations) [2]. Also the gene evaluation

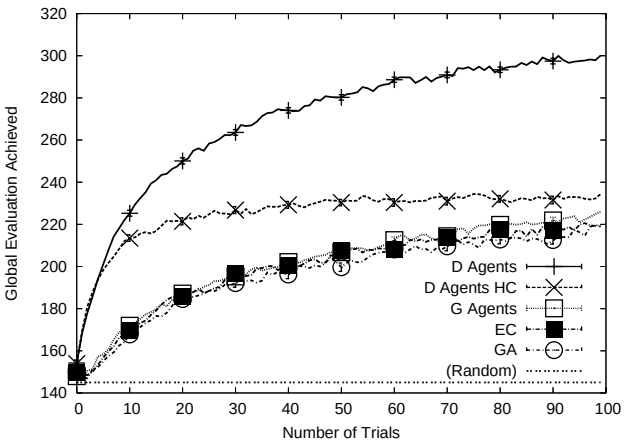


Figure 4: Exploration Rover Problem. All methods using global evaluation perform the same. The use of the difference evaluation to evaluate individual genes performs substantially better. Further, population based methods can break out of local minima that simple hill-climbers cannot. Therefore, the population based method using the difference evaluation performs 95% better (over random) than the others in this taks. Difference is statistically significant with $p < 0.001$ (error bars present but smaller than symbols).

function, D_i could be estimated more accurately. With better estimation, the learnability of D_i could be improved even in domains with highly coupled variables where MLPs are used as controllers. In addition to improved fitness estimation, state estimations would allow this multi-agent solution to be extended to POMDPs, where some of the state variables used to estimate the evaluations are not observable.

Another important area of research is how to optimize the granularity of the system, i.e. how many variables should each gene/agent control. Intuitively the variables within a gene should be tightly coupled, while the variables between genes should be loosely coupled. In this paper, the granularity was set by hand, i.e. one gene corresponded to one rover. Automating this process may lead to finding superior granularities, while reducing the burden on the system designer.

9. CONCLUSION

Many single-agent problems that can be solved by genetic algorithms, can also be solved by a multi-agent system, where each agent focuses on the simpler problem of producing a single gene. Instead of utilizing recombination to search for a good chromosome, the multi-agent approach has a large advantage in that each agent can use its own evaluation function to evaluate a single gene independently. Even though each agent begins identically, through their gene evaluation functions they self-organize to produce a set of compatible genes that combine to form a global solution. This paper shows how evaluation functions known to be effective in multi-agent problems can be used to evaluate genes as well. In the difficult double-pole-balancing problem having highly coupled state variables, the multi-agent system can produce a solution using 20% fewer trials than the best previously existing method. The multi-agent approach performs even better in domains with loosely coupled state variables achieving a performance gain of 95% (as compared to random rovers) over standard genetic and evolutionary algorithms when using gene-specific difference evaluations.

10. REFERENCES

[1] A. Agogino and K. Tumer. Efficient evaluation functions for multi-rover systems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, pages 1–12, Seattle, WA, 2004.

[2] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies: A comprehensive introduction. *Natural Computing*, 1:3–52, 2002.

[3] Christopher M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1996.

[4] S. Farritor and S. Dubowsky. Planning methodology for planetary robotic exploration. In *ASME Journal of Dynamic Systems, Measurement and Control*, volume 124, pages 4: 698–701, 2002.

[5] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *Proc. of Conf. on Simulation of Adaptive Behavior*, 1994.

[6] F. Gomez and R. Miikkulainen. Active guidance for a finless rocket through neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Chicago, Illinois, 2003.

[7] Simon Haykin. *Neural Networks A Comprehensive Foundation*. Macmillan College Publishing Company, New York, 1994.

[8] P. Hoen and E.D. de Jong. Evolutionary multi-agent systems. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature PPSN-04*, pages 872–881, 2004.

[9] T. M. Mitchell. *Machine Learning*. WCB/McGraw-Hill, Boston, MA, 1997.

[10] David Moriarty and Risto Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5:373–399, 2002.

[11] J. F. Nash. Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(48-49), 1950.

[12] G. Baldassarre S. and Nolfi D. Parisi. Evolving mobile robots able to display collective behavior. *Artificial Life*, 9:255–267, 2003.

[13] K. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, San Francisco, CA, 2002.

[14] K. Tumer and D. Wolpert, editors. *Collectives and the Design of Complex Systems*. Springer, New York, 2004.

[15] K. Tumer and D. Wolpert. A survey of collectives. In *Collectives and the Design of Complex Systems*, pages 1,42. Springer, 2004.

[16] D. H. Wolpert, K. Tumer, and J. Frank. Using collective intelligence to route internet traffic. In *Advances in Neural Information Processing Systems - 11*, pages 952–958. MIT Press, 1999.