Efficient Evaluation Functions for Multi-rover Systems

Adrian Agogino[1] and Kagan Tumer[2]

[1] University of California Santa Cruz, NASA Ames Research Center, Mailstop 269-3, Moffett Field CA 94035, USA,
adrian@email.arc.nasa.gov
[2] NASA Ames Research Center, Mailstop 269-3, Moffett Field CA 94035, USA,
kagan@email.arc.nasa.gov

Abstract. Evolutionary computation can successfully create control policies for single-agent continuous control problems. This paper extends single-agent evolutionary computation to multi-agent systems, where a large collection of agents strives to maximize a global fitness evaluation function that rates the performance of the entire system. This problem is solved in a distributed manner, where each agent evolves its own population of neural networks that are used as the control policies for the agent. Each agent evolves its population using its own agent-specific fitness evaluation function. We propose to create these agent-specific evaluation functions using the theory of collectives to avoid the coordination problem where each agent evolves neural networks that maximize its own fitness function, yet the system as a whole achieves low values of the global fitness function. Instead we will ensure that each fitness evaluation function is both "aligned" with the global evaluation function and is "learnable," i.e., the agents can readily see how their behavior affects their evaluation function. We then show how these agent-specific evaluation functions outperform global evaluation methods by up to 600% in a domain where a collection of rovers attempts to maximize the amount of information observed while navigating through a simulated environment.

1 Introduction

Evolutionary computation combined with neural networks can be very effective in finding solutions to continuous single-agent control tasks, such as pole balancing, robot navigation and rocket control [10,7,8]. The single-agent task of these evolutionary computation methods is to produce a highly fit neural network, which is used as the controller for the agent. Applying these evolutionary computation methods to certain multi-agent problems such as controlling constellations of satellites, constructing distributed algorithms and routing over a data network offer a promising approach to solving difficult, distributed control problems. Unfortunately the single-agent methods cannot be extended directly to a large multi-agent environment due to the large state-space and possible communication limitations. Instead we use an alternative approach of having

K. Deb et al. (Eds.): GECCO 2004, LNCS 3102, pp. 1–11, 2004.

each agent use its own evolutionary algorithm and attempt to maximize its own fitness evaluation function. For such a system to produce good global solutions, two fundamental issues have to be addressed: (i) ensuring that, as far as the provided global evaluation function is concerned, the agents do not work at cross-purposes (i.e., making sure that the private goals of the agents and the global goal are "aligned"); and (ii) ensuring that the agents' fitness evaluation functions are "learnable" (i.e., making sure the agents can readily see how their behavior affects their evaluation function). This paper provides a solution that satisfies both criteria in the problem of coordinating a collection of planetary exploration rovers based on continuous sensor inputs.

Current evolutionary computation methods address multi-agent systems in a number of different ways [2,9,1]. In [2], the algorithm takes advantage of a large number of agents to speed up the evolution process in domains where agents do not have the problem of working at cross-purposes. In [9] beliefs about about other agents are update through global and hand-tailored fitness functions. In addition ant colony algorithms [6] solve the coordination problem by utilizing "ant trails," providing good results in path-finding domains. Instead this paper presents a framework based on the theory of collectives that directs the evolutionary process so that agents do not work at cross-purposes, but still evolve quickly. This process is performed by giving each agent its own fitness evaluation function that is both aligned with the global evaluation function and as easy as possible for the agent to maximize. These agents can then use these evaluation functions in conjunction with the system designer's choice of evolutionary computation method. New evolutionary computation methods can replace the one used here without changing the evaluation functions, allowing the latest advances in evolutionary computation to be leveraged, without modifying the design of the overall system.

This paper will first give a brief overview of the theory of collectives in Section 2, showing how to derive agent evaluation functions that are both learnable and aligned with the global evaluation function. In Section 3 we discuss the "Rover Problem" testbed where a collection of planetary rovers use neural networks to determine their movements based on a continuous-valued array of sensor inputs. In Section 4 we compare the effectiveness of three different evaluation functions. We first use results from the previous section to derive fitness evaluation functions for the agents in a simple version of the Rover Problem in a static environment. Then we show how these methods perform in a more realistic domain with a changing environment. Results show up to a 600% increase in the performance of agents using agent-specific evaluation functions.

2 Multi-agent System Evaluation Functions

This section summarizes how to derive good evaluation functions, using the theory of collectives described by Wolpert and Tumer [11]. We first assume that there is a **global evaluation function**, $G(z)$, which is a function of all of the environmental variables and the actions of all the agents, z. The goal of the multi-

agent system is to maximize $G(z)$. However, the agents do not maximize $G(z)$ directly. Instead each agent, η, attempts to maximize its **private evaluation function** $g_\eta(z)$. The goal is to design $g(z)$s such that when all of the $g(z)$s are close to being maximized, $G(z)$ is also close to being maximized.

2.1 Factoredness and Learnability

For high values of the global evaluation function, G, to be achieved, the private evaluation functions need to have two properties, **factoredness** and **learnability**. First we want the private evaluation functions of each agent to be factored with respect to G, intuitively meaning that an action taken by an agent that improves its private evaluation function also improves the global evaluation function (i.e. G and g_η are aligned). Specifically when agent η takes an action that increases G then g_η should also increase. Formally an evaluation function g is **factored** when:

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

where $z_{-\eta}$ and $z'_{-\eta}$ contain the components of z and z' respectively, that are not influenced by agent η.

Second, we want the agents' private evaluation functions to have high **learnability**, intuitively meaning that an agent's evaluation function should be sensitive to its own actions and insensitive to actions of others. As a trivial example, any "team game" in which all the private functions equal G is factored [5]. However such systems often have low learnability, because in a large system an agent will have a difficult time discerning the effects of its actions on G. As a consequence, each η may have difficulty achieving high g_η in a team game. We call this signal/noise effect learnability:

$$\lambda_{\eta,g_\eta}(\zeta) \equiv \frac{\|\nabla_{\zeta_\eta} g_\eta(\zeta)\|}{\|\nabla_{\zeta_{-\eta}} g_\eta(\zeta)\|} \; . \tag{1}$$

Intuitively it shows the sensitivity of $g_\eta(z)$ to changes to η's actions, as opposed to changes to other agent's actions. So at a given state z, the higher the learnability, the more $g_\eta(z)$ depends on the move of agent η, i.e., the better the associated signal-to-noise ratio for η.

2.2 Difference Evaluation Functions

Consider **difference** evaluation functions, which are of the form:

$$D_\eta \equiv G(z) - G(z_{-\eta} + c_\eta) \tag{2}$$

where $z_{-\eta}$ contains all the variable not affected by agent η. All the components of z that are affected by agent η are replaced with the fixed constant c_η. Such difference evaluation functions are factored no matter what the choice of c_η, because the second term does not depend on η's actions [11]. Furthermore, they

usually have far better learnability than does a team game, because of the second term of D, which removes a lot of the effect of other agents (i.e., noise) from η's evaluation function. In many situations it is possible to use a c_η that is equivalent to taking agent η out of the system. Intuitively this causes the second term of the difference evaluation function to evaluate the fitness of the system without η and therefore D evaluates the agent's contribution to the global evaluation.

3 Continuous Rover Problem

In this section, we show how evolutionary computation with the difference evaluation function can be used effectively in the Rover Problem. In this problem, there is a set of rovers on a two dimensional plane, which are trying to observe points of interests (POIs). A POI has a fixed position on the plane and has a value associated with it. The observation information from observing a POI is inversely related to the distance the rover is from the POI. In this paper the distance metric will be the squared Euclidean norm, bounded by a minimum observation distance, d:[1]

$$\delta(x, y) = min\{\|x - y\|^2, d^2\} \ . \tag{3}$$

While any rover can observe any POI, as far as the global evaluation function is concerned, only the closest observation counts[2]. The global evaluation function for a trial is given by:

$$G = \sum_t \sum_i \frac{V_i}{\min_\eta \delta(L_i, L_{\eta,t})} \ , \tag{4}$$

where V_i is the value of POI i, L_i is the location of POI i and $L_{\eta,t}$ is the location of rover η at time t.

At every time step, the rovers sense the world through eight continuous sensors. From a rover's point of view, the world is divided up into four quadrants relative to the rover's orientation, with two sensors per quadrant (see Figure 1). For each quadrant, the first sensor returns a function of the POIs in the quadrant at time t. Specifically the first sensor for quadrant q returns the sum of the values of the POIs in its quadrant divided by their squared distance to the rover:

$$s_{1,q,\eta,t} = \sum_{i \in I_q} \frac{V_i}{\delta(L_i, L_{\eta,t})} \tag{5}$$

[1] The square Euclidean norm is appropriate for many natural phenomenon, such as light and signal attenuation. However any other type of distance metric could also be used as required by the problem domain. The minimum distance is included to prevent singularities when a rover is very close to a POI

[2] Similar evaluation functions could also be made where there are many different levels of information gain depending on the position of the rover. For example 3-D imaging may utilize different images of the same object, taken by two different rovers.

where I_q is the set of observable POIs in quadrant q. The second sensor returns the sum of square distances from a rover to all the other rovers in the quadrant at time t:

$$s_{2,q,\eta,t} = \sum_{\eta' \in N_q} \frac{1}{\delta(L_{\eta'}, L_{\eta,t})} \tag{6}$$

where N_q is the set of rovers in quadrant q.

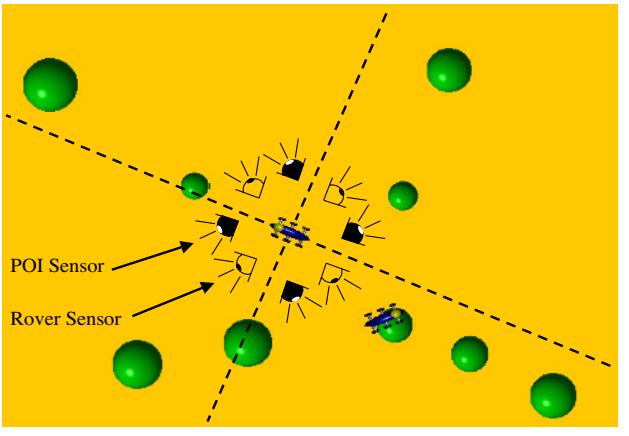

Fig. 1. Diagram of a Rover's Sensor Inputs. The world is broken up into four quadrants relative to rover's position. In each quadrant one sensor senses points of interests, while the other sensor senses other rovers.

With four quadrants and two sensors per quadrant, there are a total of eight continuous inputs. This eight dimensional sensor vector constitutes the state space for a rover. At each time step the rover uses its state to compute a two dimensional action. The action represents an x,y movement relative to the rover's location and orientation. The mapping from state to action is done with a multi-layer-perceptron (MLP), with 8 input units, 10 hidden units and 2 output units. The MLP uses a sigmoid activation function, therefore the outputs are limited to the range $(0, 1)$. The actions, dx and dy, are determined from substracing 0.5 from the output and multiplying by the maximum distance the rover can move in one time step: $dx = d(o_1 - 0.5)$ and $dy = d(o_2 - 0.5)$ where d is the maximum distance the rover can move in one time step, o_1 is the value of the first output unit, and o_2 is the value of the second output unit.

The MLP for a rover is chosen by a simple evolutionary algorithm. In this algorithm each rover has a population of MLPs. At the beginning of each trial, the rover selects the best MLP from its population 90% of the time and a random MLP from its population 10% of the time (ϵ-greedy selector). The selected MLP is then mutated by adding a value sampled from the Cauchy Distribution (with scale parameter equal to 0.3) to each weight, and is used for the entire trial. When the trial is complete, the MLP is evaluated by the rover's evaluation

function and inserted into the population. The worst performing member of the population is then deleted. While this algorithm is not sophisticated, it is effective if the evaluation function used by the agents is factored with G and highly learnable. The purpose of this work is to show gains due to principled selection of evaluation functions in a multi-agent system. We expect more advanced algorithms from evolutionary computation, used in conjunction with these same evaluation functions, to perform even better.

4 Results

The Rover Problem was tested in three different scenarios. There were ten rovers in the first two scenarios and thirty rovers in the third scenario. In each scenario, a trial consisted of 15 time steps, and each rover had a population of MLPs of size 10. The world was 100 units long and 115 units wide. All of the rovers started the trial near the center (65 units from the left boundary and 50 units from the top boundary). The maximum distance the rovers could move in one direction during a time step, d, was set to 10. The rovers could not move beyond the bounds of the world. The minimum distance, d, used to compute δ was equal to 5. In the first two scenarios, the environment was reset at the beginning of every trial. However the third scenario showed how learning in changing environments could be achieved by having the environment change at the beginning of each trial. Note that in all three scenarios other forms of continuous reinforcement learners could have been used instead of the evolutionary neural networks. However neural networks are ideal for this domain given the continuous inputs and bounded continuous outputs.

4.1 Rover Evaluation Function

In each of the three scenarios three different evaluation functions were tested. The first evaluation function was the global evaluation function (G):

$$G = \sum_t \sum_i \frac{V_i}{\min_\eta \delta(L_i, L_{\eta,t})} \tag{7}$$

The second evaluation function was the "perfectly learnable" evaluation function (P):

$$P_\eta = \sum_t \sum_i \frac{V_i}{\delta(L_i, L_{\eta,t})} \tag{8}$$

Note that the P evaluation function is equivalent to the global evaluation funtion when there is only one rover. It also has infinite learnability in the way it is defined in Section 2, since the P evaluation function for a rover is not affected by the actions of the other rovers. However the P evaluation function is not factored. Intuitively P and G offer opposite benefits, since G is by definition factored, but has poor learnability. The final evaluation function is the difference evaluation

function. It does not have as high learnability as P, but is still factored like G. For the rover problem, the difference evaluation function, D, is defined as:

$$D_\eta = \sum_t \left[\sum_i \frac{V_i}{\min_{\eta'} \delta(L_i, L_{\eta',t})} - \sum_i \frac{V_i}{\min_{\eta' \neq \eta} \delta(L_i, L_{\eta,t})} \right]$$
$$= \sum_t \sum_i I_{i,\eta,t}(z) \frac{V_i}{\delta(L_i, L_{\eta,t})}$$

where $I_{i,\eta,t}(z)$ is an indicator function, returning one if and only if η is the closest rover to L_i at time t. The second term of the D is equal to the value of all the information collected if rover η were not in the system. Note that for all time steps where η is not the closest rover to any POI, the subtraction leaves zero. The difference evaluation can be computed easily as long as η knows the position and distance of the closest rover to each POI it can see. If η cannot see a POI then it is not the closest rover to it. In the simplified form, this is a very intuitive evaluation function yet it was generated mechanically from the general form of the difference evaluation function [11]. In this simplified domain we could expect a hand-crafted evaluation function to be similar. However the difference evaluation function can still be used in more complex domains with a less tractable form of the global utility, even when it is difficult to generate and evaluate hand-crafted solution. Even in domains where an intuitive feel is lacking, the difference evaluation function will be provably factored and learnable.

4.2 Learning in Static Environment

The first experiment was performed using ten rovers and a set of POIs that remained fixed for all trials (see Figure 2). The POIs were placed in such a way as to create the potential of congestion problems around one highly valued POI, testing the cooperation level among the rovers. This was achieved by creating a grid of 15 POIs, with value 3.0, to the left of the rovers' starting location, and a high valued POI of value 10.0 to the right of the rovers' starting location. There was also a POI of value 10.0, which was ten units to the left of the rovers' starting location. System performance was measured by how well the rovers were able to maximize the global evaluation function (even though from a rover's point of view, it is trying to maximize its private evaluation function).

Results from Figure 3 (left) show that the rovers using D performed the best, by a wide margin. Early in training, rovers using P performed better than rovers using G. However since the learning curve of these rovers using P remained flat, while the ones using G increased, the rovers using G eventually overtook the ones using P. The error bars (smaller than the symbols) show that these results are statistically significant. In essence agents using P converge quickly to a poor solution, while agents using G move slowly towards a good solution, while agents using D converge rapidly to a good solution. This phenomenon is explained by factoredness and learnability. The P evaluation function is highly learnable since it is only affected by the moves of a single rover. However since

P is not factored, a rover that maximizes it occasionally takes actions that hurt its fitness with respect to the global evaluation. In contrast rovers using G learn slowly, since the global evaluation is effected by the actions of all the other rovers.

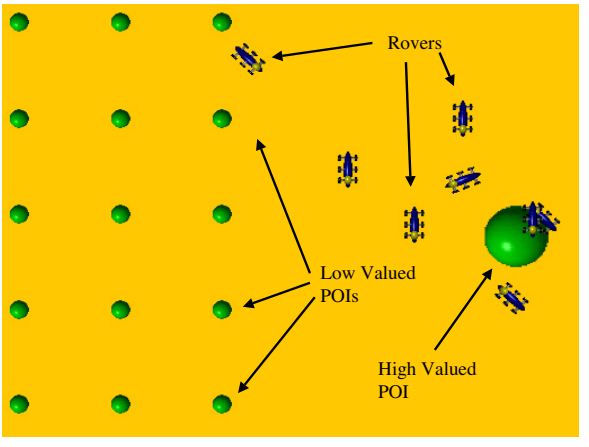

Fig. 2. Diagram of Static Environment. Points of interests are at fixed locations for every trial.

The second experiment is similar to the first one except that the value of a POI goes down each time it is observed by the closest agent. This reduction in value models a domain where an agent receives less useful new information with each successive observation. This is a harder problem than the previous one, since the values of the POIs change at every time step during a trial. Figure 3 (right) verifies that this domain is more difficult as rovers using all three evaluation function performed worse than in the pervious domain. However rovers using D, still performe significantly better than rovers using the other evaluation functions. Note also that rovers using G suffer the most in this domain since they were cut off at a steeper portion in their learning curve than the agents using the other evaluation function. By the time the trial had ended, the rovers using G had just begun to learn their domain because of G's low learnability.

4.3 Learning in Changing Environment

In the first two experiments, the environment was returned to its starting state at the beginning of each trial. Therefore the rovers learned specific control policies for a specific configuration of POIs. This type of learning is most useful when the rovers learn on a simulated environment and are then deployed to an environment closely matching the simulation. However it is often desirable for the rovers to be able to learn a control policy after they are deployed in their environment. In such a scenario the rovers generalize what they learned in the parts of the environment they were first deployed in, to other parts of the environment. The

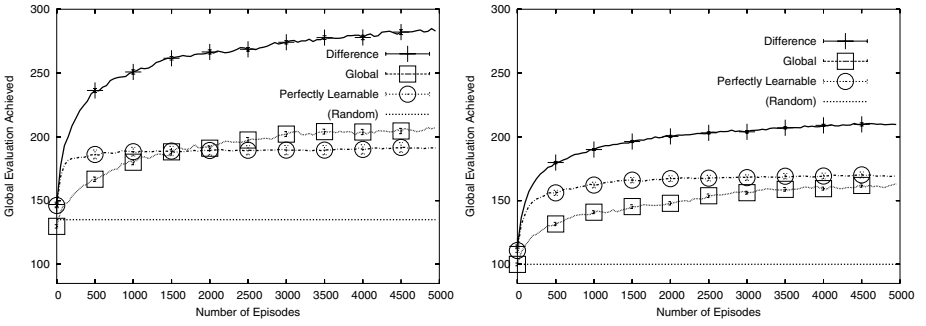

Fig. 3. Results for Three Different Evaluation Functions in Static Environment. Points of interests are at fixed locations for every trial. Right Figure: Results when POI values constant for duration of trial. Left Figure: Results when POI values decrease as they are observed. Difference evaluation is superior since it is both factored and learnable.

last experiment tests the rovers' ability to generalize what they learned in early environmental conditions to later environmental conditions.

In the last experiment there were thirty rovers and thirty POIs. POI locations were set randomly at the beginning of each trial using a uniform distribution within a 70 by 70 unit square centered on the rovers starting location (see Figure 4). The value of the POIs were set to random values uniformly chosen between one and ten. Changing locations at each trial forced the rovers to create a general solution, based on their sensor inputs, since each new trial was different from all of the trials they had previously seen. This type of problem is common in real world domains, where the rovers typically learn in a simulator and later have to apply their learning to the environment in which they are deployed. Note that learning in this scenario does not depend on the use of multiple trials as the rovers can continuously learn, and generalize from their past experience.

Figure 5 shows that rovers using D performed best in this scenario. Rovers using D were effective in generalizing the knowledge gained from exploring previous POI configurations and applying that knowledge to new POI configurations. In contrast, rovers using the P evaluation were especially ineffective in this scenario. We attribute this to the congested nature of the problem, where the rovers competed rather than cooperating with each other. Since a rover's P evaluation only returns the value of what that rover observes, a rover using the P evaluation tends to move towards the highest valued POI in its area. However all the other rovers in that vicinity are also moving towards the same high-valued POI, and thus many other POIs are not properly observed.

5 Conclusion

This paper has shown that even simple evolutionary algorithms can be used in complex multi-agent systems, if the proper evaluation functions are used. In

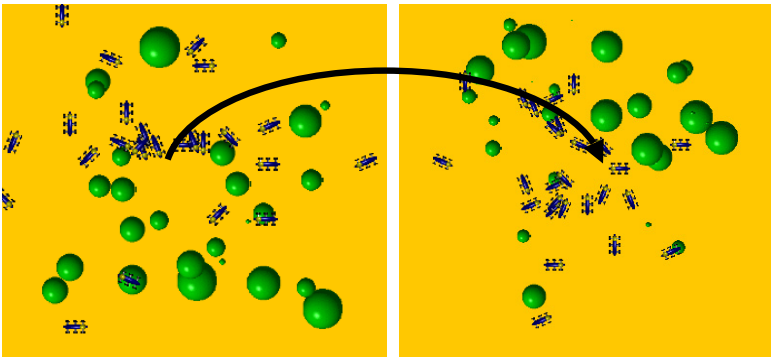

Fig. 4. Changing Environment. POIs are placed at random locations at the beginning of each trial. Rovers have to generalize their knowledge from one trial to the next.

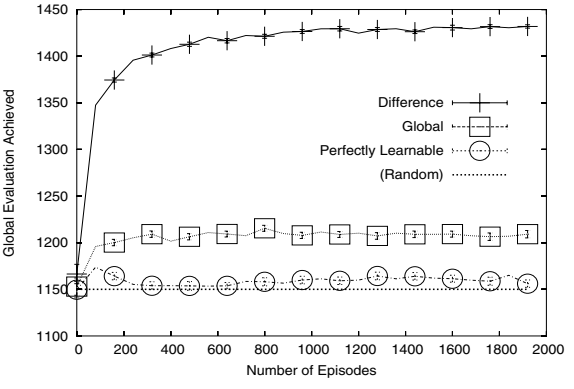

Fig. 5. Results in Changing Environment. Difference evaluation is superior since it is both factored and learnable.

simple continuous problems, the neural network based rovers using the difference evaluation function D, derived from the theory of collectives, were able to achieve high levels of performance because the evaluation function was both factored and highly learnable. These rovers performed 300% better (over random rovers) than rovers using the non-factored perfectly learnable utility and more than 200% better than rovers using the hard to learn global evaluations. These results were even more pronounced in a more difficult domain with a changing environment where rovers using the difference evaluation performed up to 600% better than rovers using global evaluations. These rovers were still able to learn quickly even though they had to generalize a solution learned in earlier environmental configurations to new environmental configurations. These results show the power of using factored and learnable fitness evaluation functions, which allow evolutionary computation methods to be successfully applied to large distributed systems.

References

1. Arvin Agah and George A. Bekey. A genetic algorithm-based controller for decentralized multi-agent robotic systems. In *In Proc. of the IEEE International Conference of Evolutionary Computing*, Nagoya, Japan, 1996.
2. A. Agogino, K. Stanley, and R. Miikkulainen. Online interactive neuro-evolution. *Neural Processing Letters*, 11:29–38, 2000.
3. T. Balch. Behavioral diversity as multiagent cooperation. In *Proc. of SPIE '99 Workshop on Multiagent Systems*, Boston, MA, 1999.
4. G. Baldassarre, S. Nolfi, and D. Parisi. Evolving mobile robots able to display collective behavior. *Artificial Life*, pages 9: 255–267, 2003.
5. R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems - 8*, pages 1017–1023. MIT Press, 1996.
6. M. Dorigo and L. M. Gambardella. Ant colony systems: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
7. 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.
8. 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.
9. E. Lamma, F. Riguzzi, and L.ÊM. Pereira. Belief revision by multi-agent genetic search. In *In Proc. of the 2nd International Workshop on Computational Logic for Multi-Agent Systems*, Paphos, Cyprus, December 2001.
10. 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.
11. D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279, 2001.