# Distributed Evaluation Functions for Fault Tolerant Multi-Rover Systems

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

## ABSTRACT

The ability to evolve fault tolerant control strategies for large collections of agents is critical to the successful application of evolutionary strategies to domains where failures are common. Furthermore, while evolutionary algorithms have been highly successful in discovering single-agent control strategies, extending such algorithms to multi-agent domains has proven to be difficult. In this paper we present a method for shaping evaluation functions for agents that provide control strategies that are both tolerant to different types of failures and lead to coordinated behavior in a multi-agent setting. This method neither relies on a centralized strategy (susceptible to single points of failures) nor a distributed strategy where each agent uses a system wide evaluation function (severe credit assignment problem). In a multi-rover problem, we show that agents using our agent-specific evaluation perform up to 500% better than agents using the system evaluation. In addition we show that agents are still able to maintain a high level of performance when up to 60% of the agents fail due to actuator, communication or controller faults.

## Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Multiagent systems*

## General Terms

Algorithms, Performance

## Keywords

Multiagent Systems, Robust Optimization, Genetic Algorithms, Neural Networks

## 1. INTRODUCTION

Many important control tasks involve components that have high failure rates. These tasks are especially common in aerospace domains, where harsh environmental conditions stress hardware, and in defense domains, where adversaries try to destroy hardware. An evolutionary multi-agent system can be an excellent tool in overcoming failures, as agents that do not fail can evolve to overcome the difficulties imposed by agents that do fail. Adaptive multi-agent systems are naturally robust and reconfigurable. While evolutionary computation has shown to be successful in numerous single-agent control tasks, including pole balancing, robot navigation and rocket control, applying these single-agent methods to a multi-agent system in a robust way is often difficult [11, 7, 8, 3]. Instead of reducing the risk of failure, a poorly designed multi-agent system can increase the rate of failure over that of a single-agent system. For example evolutionary algorithms that are too centralized allow for single points of failure while evolutionary algorithms that cannot adapt quickly are vulnerable to any failure in the system. In this paper, we address these issues in a multi-rover domain by having each rover evolve its control strategy independently, using an evaluation function that allows it to adapt quickly. This framework is based on work previously described in [3] and [12] and is extended to the domain of rover failures.

The main issue that needs to be addressed in such a distributed evolutionary process is what evaluation function should be used by each rover. A natural choice is to have the rovers directly use the full system evaluation function that rates the performance of the entire multi-rover system. While as system designers, we ultimately wish to have the system evaluation maximized, having the rovers use the system evaluation directly could lead to difficult credit assignment problems when there are many rovers. When using the system evaluation, a rover could rate a good control policy poorly if other rovers were taking poor actions during the evaluation process. Correspondingly, a rover could rate a bad control policy highly if other rovers were taking good actions during the evaluation process. The difficulty with having rovers use the system evaluation, is that since each rover has so little individual impact on its evaluation function, it will take a long time to evolve an effective control strategy. To overcome this problem, this paper will have each rover use a rover-specific evaluation function. A rover's actions have greater impact on their rover-specific evaluation, which will be carefully designed so that when rovers evolve to maximize their rover-specific evaluation, they will also tend to maximize the system evaluation.

This paper shows how these rover-specific evaluations can be used effectively in a multi-rover environment when a majority of the rovers suffer from one of the following faults:


*GECCO'06,* July 8–12, 2006, Seattle, Washington, USA.

1. Rovers can have faults in actuators where they are unable to move.

2. Rovers can have faults in communication where they are unable to ascertain the locations of other rovers used to compute evaluation functions.

3. Rovers can have faulty controllers, where an overwhelming amount of noise is added to output of the controllers, causing the rovers to move almost at random.

We show that rovers using rover-specific evaluation functions can overcome severe fault scenarios, with little loss in performance.

In Section 2 we discuss the properties that evaluation functions should have in a multi-rover system and how to evolve rovers using evaluation functions possessing such properties along with a discussion of related work. In Section 3 we present the "Rover Problem" where a set of planetary rovers use neural networks to determine their movements based on a continuous-valued array of sensor inputs. In Section 5 we present the performance of the multi-rover system evolved using rover evaluation functions under actuator, communication and sensor faults. The results show that the effectiveness of the rovers in gathering information is 500% higher when they use properly derived rover fitness functions than when they use a system evaluation function. We conclude in Section 6 with a discussion of the implication of these results and their applicability to different domains.

## 2. MULTI-ROVER EVOLUTION

The most straightforward method to apply evolutionary computation to a multi-rover system is to treat the multi-rover system as a large single entity. Under this method there is a single population of control policies, where each control policy controls the entire multi-rover system. Unfortunately such a method is neither robust (i.e.,., single point of failure) nor efficient (i.e., extremely slow convergence). While ad-hoc redundancy facilities could be added, such an approach makes the search for a good solution even more cumbersome. The slow convergence problems of such systems are particularly devastating in our domain where rapid adaptation is needed when a significant percentage of the rovers fail.

To allow for higher levels of robustness, instead of using a single population of system-wide control policies, we have each rover evolve its own control policy independently (see Figure 1). Since each agent has its own control policy, the structure of the control policy can be much simpler. In fact, in this paper each rover's control policy is a simple feed-forward neural network with one hidden layer. In this architecture, each rover has its own population of control strategies, that it uses to control its actions. After the rover has taken a sequence of actions, it can then evaluate the performance of a control policy based on how much it contributed to the full system. Since each rover evolves its own population it can be given any evaluation function. The key to making this process work effectively is having the rovers use well designed evaluation functions that are sensitive to the rovers actions, yet still aligned with the system evaluation. The following sections discuss critical properties of rover evaluation functions, and how to create a good evaluation function for this domain. This will lead to the definition of an evaluation function called the difference evaluation function (D). We will show in the experimental section this evaluation function has superior performance in our domain to a more traditional system evaluation (G).

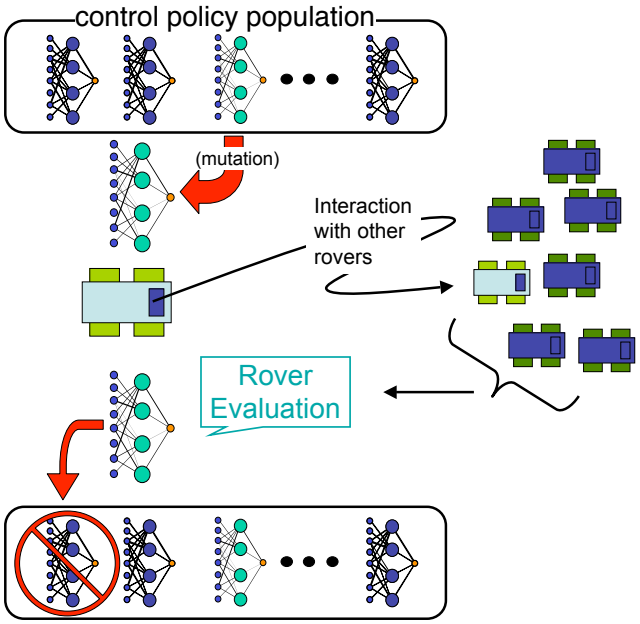


**Figure 1: Evolution Process for Single Rover. A rover chooses a control policy from its own population of control policies. It then uses it for control. After evaluating the control policy's effectiveness, the rover updates its population. In this paper, the control policies used are neural networks.**

### 2.1 Rover Evaluation Function Properties

In this section we illustrate some important properties of rover-specific evaluation functions based on work described in [15] and in the context of previous multi-rover control work described in [3] and [12]. Let the **system evaluation function** be given by $G(z)$, where $z$ is the state of the full system (e.g., the position of all the rovers in the system, along with their relevant internal parameters and the state of the environment). Let the **rover evaluation function** for rover $i$ be given by $g_i(z)$. First we want the private evaluation functions of each rover to have high *factoredness* with respect to $G$, intuitively meaning that an action taken by a rover that improves its private evaluation function also improves the system evaluation function (i.e. $G$ and $g_\eta$ are aligned). Formally, the degree of factoredness between $g_i$ and $G$ is given by:

$$\mathcal{F}_{g_i} = \frac{\int_z \int_{z'} u[(g_i(z) - g_i(z'))\ (G(z) - G(z'))] dz' dz}{\int_z \int_{z'} dz' dz}, \quad (1)$$

where $z'$ is a state which only differs from $z$ in the state of rover $i$, and $u[x]$ is the unit step function, equal to 1 when $x > 0$. Intuitively, a high degree of factoredness between $g_i$ and $G$ means that a rover evolved to maximize $g_i$ will also maximize $G$.

Second, the rover evaluation function must be more sensitive to changes in that rover's fitness than to changes in the fitness of all the other rovers. Formally we quantify the

*rover-sensitivity* of evaluation function $g_i$, at $z$ as:

$$\lambda_{i,g_i}(z) = E_{z'}\left[\frac{\|g_i(z) - g_i(z - z_i + z'_i)\|}{\|g_i(z) - g_i(z' - z'_i + z_i)\|}\right], \quad (2)$$

where $E_{z'}[\cdot]$ provides the expected value over possible values of $z'$, and $(z - z_i + z'_i)$ notation specifies the state vector where the components of rover $i$ have been removed from state $z$ and replaced by the components of rover $i$ from state $z'$. So at a given state $z$, the higher the rover-sensitivity, the more $g_i(z)$ depends on changes to the state of rover $i$, i.e., the better the associated signal-to-noise ratio for $i$. Intuitively then, higher rover-sensitivity means there is "cleaner" (e.g., less noisy) selective pressure on rover $i$. Ideally we want evaluation functions that are both factored and highly rover-sensitive (Figure 2).

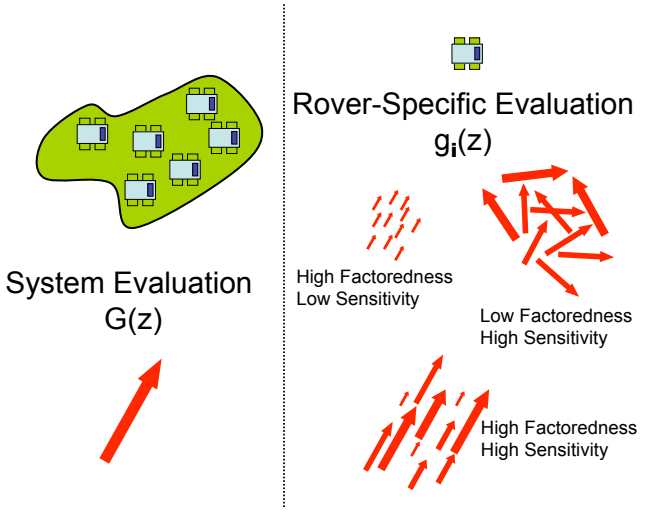


**Figure 2: Properties of Rover-Specific Evaluations. Direction of an arrow represents the goal of evaluation. Size of an arrow represents sensitivity of the evaluation to rover's action. As a system designer we are concerned with maximizing the system evaluation function (left). For rovers to be able to effectively maximize system evaluation, their evaluations should be aligned with the system evaluation (high factoredness) and supply a strong signal (high sensitivity).**

As an example, consider the case where the rover evaluation function of each rover is set to the system evaluation function, meaning that each rover is evaluated based on the fitness of the full system (e.g., approach 2 discussed in Section 2). Such a system will be fully factored by the definition of Equation 1. However, the rover fitness functions will have low rover-sensitivity (the fitness of each rover depends on the fitness of all other rovers).

## 2.2 Difference Evaluation Functions

Let us now focus on improving the rover-sensitivity of the evaluation functions. To that end, consider **difference** evaluation functions [15], which are of the form:

$$D_i \equiv G(z) - G(z_{-i} + c_i), \quad (3)$$

where $z_{-i}$ contains all the states on which rover $i$ has no effect, and $c_i$ is a fixed vector. In other words, all the components of $z$ that are affected by rover $i$ are replaced with the fixed vector $c_i$. Such difference evaluation functions are fully factored no matter what the choice of $c_i$, because the second term does not depend on $i$'s states [15] (e.g., $D$ and $G$ will have the same derivative with respect to $z_i$). Furthermore, they usually have far better rover-sensitivity than does a system evaluation function, because the second term of D removes some of the effect of other rovers (i.e., noise) from $i$'s evaluation function. In many situations it is possible to use a $c_i$ that is equivalent to taking rover $i$ out of the system. Intuitively this causes the second term of the difference evaluation function to evaluate the fitness of the system without $i$ and therefore D evaluates the rover's contribution to the system evaluation.

Though for linear evaluation functions $D_i$ simply cancels out the effect of other rovers in computing rover $i$'s evaluation function, its applicability is not restricted to such functions. In fact, it can be applied to any linear or non-linear global evaluation function. However, its effectiveness is dependent on the domain and the interaction among the rover evaluation functions. At best, it fully cancels the effect of all other rovers. At worst, it reduces to the system evaluation function, unable to remove any terms (e.g., when $z_{-i}$ is empty, meaning that rover $i$ effects all states). In most real world applications, it falls somewhere in between, and has been successfully used in many domains including rover coordination, satellite control, data routing, job scheduling and congestion games [3, 13, 15]. Also note that the computation of $D_i$ is a "virtual" operation in that rover $i$ computes the impact of its not being in the system. There is no need to re-evolve the system for each rover to compute its $D_i$, and computationally it is often easier to compute than the system evaluation function [13]. Indeed in the problem presented in this paper, for rover $i$, $D_i$ is easier to compute than $G$ (see details in Section 5).

## 2.3 Related Work

Evolutionary computation has a long history of success in single-agent and multi-agent control problems [14, 9, 6, 2, 1]. Advances in evolutionary computation methods in single-agent domains tend to result from improvements in search methods. In [9] this is accomplished through fuzzy rules in a helicopter control problem, while in [14] cellular encoding is used to improve performance on pole-balancing control. Similarly [6] addresses planetary rover control by having genetic algorithms search through a space of plans generated from a planning algorithm.

Many advances in evolutionary computation for multi-agent control have been accomplished through the use of domain specific fitness functions. Ant colony algorithms [5] solve the coordination problem by utilizing "ant trails" that provide implicit fitness functions resulting in good performance in path-finding domains. In [2], the algorithm takes advantage of a large number of agents to speed up the evolution process in certain domains, but uses greedy fitness functions that are not generally factored. Also outside of evolutionary computation, coordination between a set of mobile robots has been accomplished through the use of hand-tailored rewards designed to prevent greedy behavior [10]. While highly successful in many domains all of these methods differ from the methods used in this paper in that they lack a general framework for efficient evolution in multi-agent systems.

## 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[1]. In this problem, there is a collection of rovers on a two dimensional plane, which is trying to observe points of interests (POIs). Each POI has a value associated with it and each observation of a POI yields an observation value 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, $\delta_{min}$:[2]

$$\delta(x, y) = \max\{\|x - y\|^2, \delta_{min}^2\} \ . \tag{4}$$

The goal of the system designer is to have rover try to observe valuable POIs at a close range as computed by the system evaluation.

### 3.1 System Evaluation

Formally the system evaluation function is given by:

$$G = \sum_t \sum_j \frac{V_j}{\min_i \delta(L_j, L_{i,t})} \ , \tag{5}$$

where $V_j$ is the value of POI $j$, $L_j$ is the location of POI $j$ and $L_{i,t}$ is the location of rover $i$ at time $t$. Intuitively, while any rover can observe any POI, as far as the global evaluation function is concerned, only the closest observation matters[3]. This is the function we as a system designer wish the rovers to maximize.

### 3.2 Rover Capabilities

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 3). 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 and scaled by the angle between the POI and the center of the quadrant:

$$s_{1,q,j,t} = \sum_{j \in J_q} \frac{V_j}{\delta(L_j, L_{i,t})} \left(1 - \frac{|\theta_{j,q}|}{45^\circ}\right) \ , \tag{6}$$

where $J_q$ is the set of observable POIs in quadrant $q$ and $|\theta_{j,q}|$ is the magnitude of the angle between POI $j$ and the center of the quadrant. The second sensor returns the sum of square distances from a rover to all the other rovers in the quadrant at time $t$ scaled by the angle:

$$s_{2,q,i,t} = \sum_{i' \in N_q} \frac{1}{\delta(L_{i'}, L_{i,t})} \left(1 - \frac{|\theta_{i',q}|}{45^\circ}\right) \ , \tag{7}$$

where $N_q$ is the set of rovers in quadrant $q$ and $|\theta_{i',q}|$ is the magnitude of the angle between rover $i'$ and the center of the quadrant.

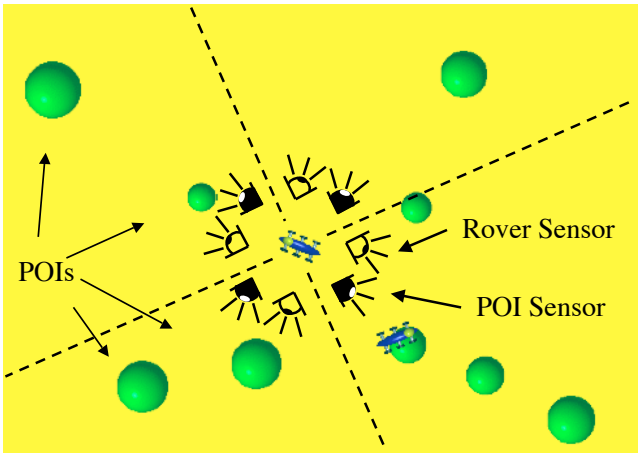


**Figure 3: Diagram of a Rover's Sensor Inputs. The world is broken up into four quadrants relative to rover's position. In each quadrant one sensor senses points of interests, while the other sensor senses other rovers.**

The sensor space is broken down into four regions to facilitate the input-output mapping. There is a trade-off between the granularity of the regions and the dimensionality of the input space. In some domains the tradeoffs may be such that it is preferable to have more or fewer than four sensor regions. Also, even though this paper assumes that there are actually two sensors present in each region at all times, in real problems there may be only two sensors on the rover, and they do a sensor sweep at 90 degree increments at the beginning of every time step.

### 3.3 Rover Control Strategies

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 output. This output represents the $x, y$ movement relative to the rover's location and orientation.

The mapping from rover state to rover output is done through a Multi Layer Perceptron (MLP), with eight input units, ten hidden units and two output units [4]. The MLP uses a sigmoid activation function, therefore the outputs are limited to the range $(0, 1)$. The actual rover motions $dx$ and $dy$, are determined by normalizing and scaling the MLP output by the maximum distance the rover can move in one time step. More precisely, we have:

$$dx = 2d_{max}(o_1 - 0.5)$$
$$dy = 2d_{max}(o_2 - 0.5) \ ,$$

where $d_{max}$ 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.

[1]This problem was first presented in [3].

[2]The square Euclidean norm is appropriate for many natural phenomena, 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.

[3]Similar evaluation functions could also be used 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.

[4]Note that other forms of continuous reinforcement learners could also be used instead of evolutionary neural networks. However neural networks are ideal for this domain given the continuous inputs and bounded continuous outputs.

## 3.4 Rover Selection

The MLP for a rover is selected using an evolutionary algorithm as highlighted in approaches two and three in Section 2. In this case, each rover has a population of MLPs. At every Nth time step (N set to 15 in these experiments), the rover uses $\epsilon$-greedy selection ($\epsilon = 0.1$) to determine which MLP it will use (e.g., it it selects the best MLP from its population with 90% probability and a random MLP from its population with 10% probability). 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 those N steps. At the end of those N steps, the MLP's performance is evaluated by the rover's evaluation function and re-inserted into its population of MLPs, at which time, the poorest performing member of the population is deleted. Both the system evaluation for system performance and rover evaluation for MLP selection is computed using an N-step window, meaning that the rovers only receive an evaluation after N steps.

While this is not a sophisticated evolutionary algorithm, it is ideal in this work since our purpose is to demonstrate the impact of principled evaluation functions selection on the performance of a multi-rover system. Even so, this algorithm has shown to be effective when the evaluation function used by the rovers is factored with $G$ and has high rover-sensitivity. We expect more advanced evolutionary computation algorithms used in conjunction with these same evaluation functions to improve the performance of the system further.

## 3.5 Evolving Control Strategies in Multi-Rover Systems

The key to success in evolving control strategies for multi-rover systems is to determine the correct rover evaluation functions. In this work we test two different evaluation function for rover selection. The first evaluation function is the system evaluation function (G), which when implemented results in approach two discussed in Section 2:

$$G = \sum_t \sum_j \frac{V_j}{\max_i \delta(L_j, L_{i,t})} \ . \tag{8}$$

The second evaluation function is the difference evaluation function. This evaluation is factored with respect to the system evaluation function, but has much high rover sensitivity. For the rover problem, the difference evaluation function, D, becomes:

$$\begin{aligned} D_i(L) = &\quad G(L) - G(L - L_i) \\ = &\quad \textstyle\sum_t \sum_j I_{j,i,t}(z) \left[ \frac{V_j}{\delta(L_j, L_{i,t})} - \frac{V_j}{\delta(L_j, L_{k_j,t})} \right] \ , \end{aligned}$$

where $k_j$ is the second closest rover to POI $j$ and $I_{j,i,t}(z)$ is an indicator function, returning one if and only if rover $i$ is the closest rover to POI $j$ at time $t$. The second term of $D$ is equal to the value of all the information that would have been collected if rover $i$ were not in the system. Note that for all time steps where $i$ is not the closest rover to any POI, the subtraction leaves zero. As mentioned in Section 2.2, the difference evaluation computation requires that rover $i$ know the position and distance of the closest rover to each POI it can see. In that regard, the $D$ evaluation function requires knowledge about the position of fewer rovers than the system evaluation function $G$.

# 4. SIMULATION SETUP

We performed extensive simulations to test the effectiveness of the two rover evaluation functions under three different kinds of failure scenarios as well as under a scenario where there were no failures. The simulations were designed to illustrate the relative effectiveness of the evaluation functions when there are many agents, and to show the ability of the distributed evolutionary system to reconfigure itself in the event of failures.

## 4.1 Failure Modes

In the simulation we test the following three types of faults:

- The first fault is actuator failure, which we simulate as having a faulty rover being unable to move after the fault has occurred. This tests the ability of the distributed system to recover when not all the members of the system continue to operate. This recovery capability is especially important in increasing the longevity of space science missions, where scientists want to keep the mission operating as long as valuable data is being produced. While high failure rates are common within the nominal life of a space science mission, as the mission is extended beyond its nominal life, the failure rate of components increases further. The ability to overcome these failures increases the likelihood of success within its nominal life and is critical in extending the duration of missions to their maximum potential.

- The second fault is communication failure where rovers are unable to ascertain the locations of other rovers used to compute evaluation functions. This fault represents a form of equipment failure common in exploration and high-risk domains.

- The third fault is control failure, where an overwhelming amount of noise is added to the output of the controllers, causing the rovers to move almost at random. This fault simulates a failure in the control computer or a fault in a lower level control system. These types of faults are particularly devastating as they can cause rovers to take actions that are destructive.

## 4.2 Environmental Setup

In these experiments, each rover had a population of MLPs of size 10. The world was 75 units long and 75 units wide. All of the rovers started the experiment at the center of the world. In all experiments there were 30 rovers in the simulations. The maximum distance the rovers could move in one direction during a time step, $d_{max}$, was set to 3. The rovers could not move beyond the bounds of the world. The minimum observation distance, $\delta_{min}$, was equal to 5.

In the experiments the environment was dynamic, meaning that the POI locations and values changed with time. There were as many POIs as rovers, and the value of each POI was set to between three and five using a uniformly random distribution. In these experiments, each POI disappeared with probability 2.5%, and another one appeared with the same probability at 15 time step intervals. The locations of new POIs is not related to the old ones. A new POI is placed randomly within the bounds of the world using a uniform distribution. Because the experiments were run for 3000 time steps, the initial and final environments

had little similarities. All results were averaged over at least one hundred independent trials. For each experiment and trial the weights of the neural network were initialized to random using the Cauchy distribution (parameter of 0.5).

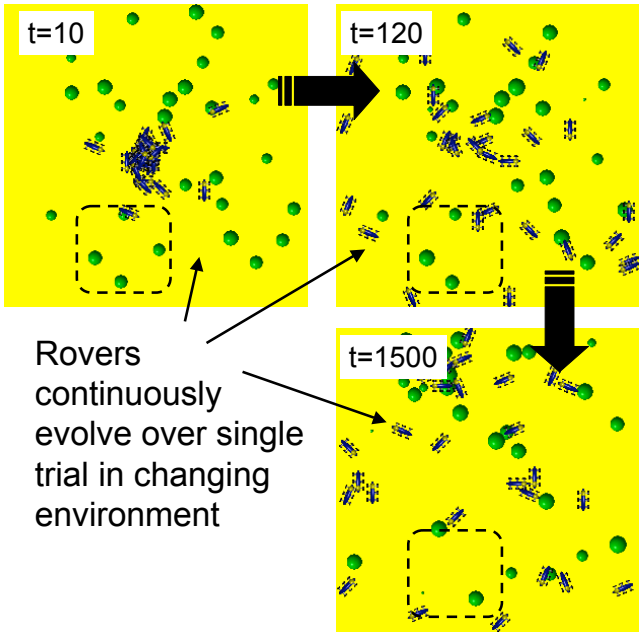


**Figure 4: Sample POI Placement. Left: Environment at time = 15. Middle: Environment at time = 150. Right: Environment at time = 1500. POI placement changes significantly during trial.**

The dynamic environment experiments used here test the ability of the evolutionary system to produce rover control policies that can be generalized from one set of POIs to another, regardless of how significantly the environment changes. Figures 4 shows an instance of change in the environment throughout a simulation. The final POI set is not particularly close to the initial POI set and the rovers are forced to focus on the sensor input-output mappings rather than focus on regions in the $(x, y)$ plane.

# 5. RESULTS

Using the rover simulation we produced experimental results for rovers performing under four scenarios, in which rovers have faults in three of the scenarios. In all experiments performance was measured using the rovers' system-wide performance as computed by the system evaluation function, regardless of the evaluation functions the individual rovers actually used. The performance measures were all relative to the performance of a set of rovers that used random evaluations functions (computed using a uniform distributed between 0.0 and 1.0). Values were obtained by subtracting the performance results of the random rovers from the performance of the rovers we were testing. The error bars were computing using the root mean square of the sigma values for the rovers using the random evaluations and the rovers using the evaluation functions we were testing. All experiments were done over at least 50 independent trials to gain statistical significance.

## 5.1 Evolution with no Failures

The first set of experiments tested the performance of the two evaluation functions in a dynamic environment for 30 rovers, where none of the rovers had failures. Figure 5 shows the performance of each evaluation function. Independent of the evaluation function the rovers actually used, performance was measured using the system evaluation function. Here rovers evolving the system evaluation function directly performed poorly, and could not even achieve statistically significant performance beyond random levels until 1000 time steps. In contrast rovers using the difference evaluation learned quickly and achieved a final performance level more than five times higher than the agents using the system evaluation.

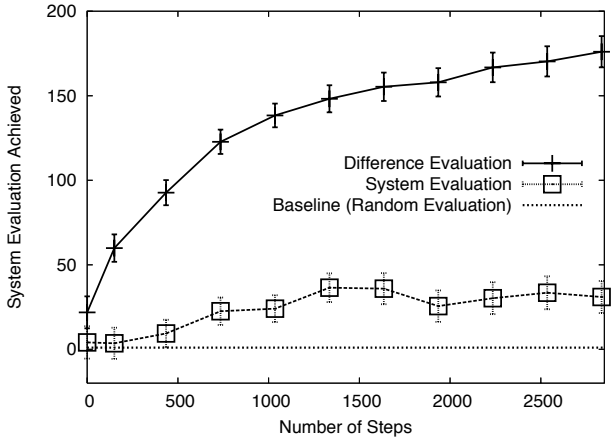


**Figure 5: Performance of a 30-rover system for two evaluation functions in fault-free environment. Difference evaluation function provides the best system-wide performance because it is rover-sensitive.**

The evolution of this system demonstrates the importance of having evaluation functions that are rover-sensitive. Having an evaluation function that is highly rover-sensitive speeds up learning since a rover needs to take fewer actions in order to discern which actions lead to high evaluation and which actions lead to low evaluation. In this domain rovers that use $G$ are hurt by the evaluation function's low rover-sensitivity and learn slowly. Since the fitness of each rover depends on the state of all of the other rovers, the noise in the system overwhelms the evaluation function. Rovers using the system evaluation have difficulty figuring out which actions are bad and which actions are good. In contrast the $D$ evaluation has high rover-sensitivity. As a consequence, it continues to improve well into the simulation as the fitness signal the rovers receive are not swamped by the states of the other rovers in the system.

## 5.2 Evolution under Mobility Failures

The second set of experiments tested the performance of the two evaluation functions in a dynamic environment when rovers have mobility failures. In this scenario at time step 1500, 60% of the rovers stopped moving and were unable to move for the rest of the trial. However, the immobile rovers were still able to observe POIs. In this scenario the challenge of the mobile rovers was to observe POIs that were

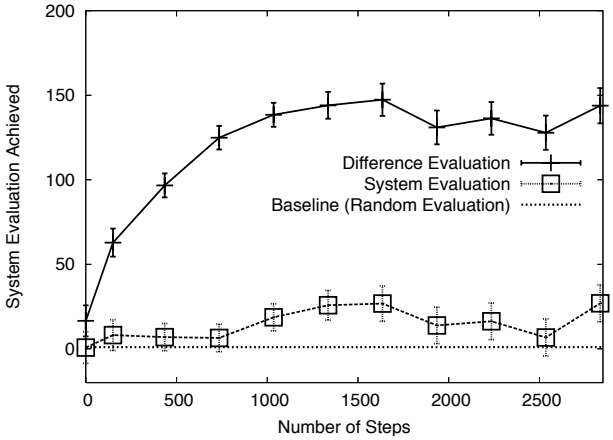


**Figure 6: Performance of a 30-rover system for two evaluation functions when 60% of the rovers become immobile after 1500 time steps (though immobile rovers can still make observations).**

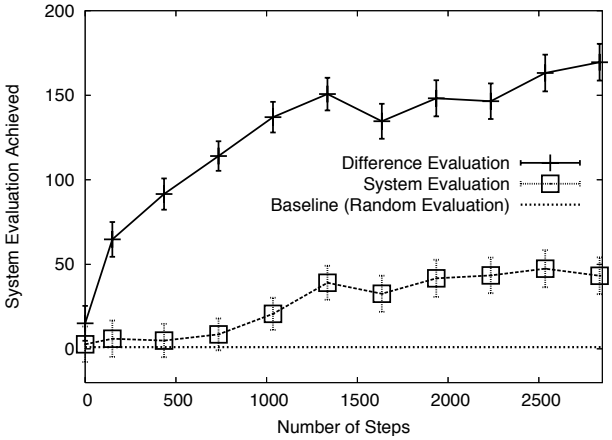


**Figure 7: Performance of a 30-rover system for two evaluation functions when 60% of the rovers are unable to communicate with other rovers in order to compute evaluation functions. Rover-sensitive evaluations are superior, allowing rovers to quickly adapt to failure.**

out of range of the immobile ones. Figure 6 shows the performance of both evaluation functions in this scenario. Agents using the difference evaluation proved that they could handle the faults gracefully and maintain a performance level that was only slightly degraded over when there were no faults. These agents show that they could form a dynamically reconfigurable system. In contrast agents using the system evaluation performed poorly and were barely able to do better than random.

## 5.3 Evolution with Communication Failures

The third set of experiments tested the performance of the two evaluation functions in a dynamic environment where 60% of the rovers lost most of their ability to communicate with other rovers after time step 1500. These failed rovers were only aware of other rovers when they were within a radius of 4 units from their current location. This amounted to the rovers being able to communicate with only 1% of the grid. This loss of communication made it difficult for the rovers to compute their evaluation functions. Figure 7 shows the performance of the two evaluation function under this failure scenario. While the communication failure initially lowered the system performance, the rovers using difference evaluations were able to overcome this setback and ended up achieving a performance level higher than before the failure. However, rovers using the system evaluation, as before, were never able to achieve satisfactory performance levels.

## 5.4 Evolution under Controller Failures

The fourth set of experiments tested the performance of both evaluation functions in a dynamic environment where 60% of the rovers had significant control failures at time step 1500. This failure was modeled by adding 200% noise to the output of the rovers' neural network controller. As the output of the controllers for non-failing rovers was bounded between 0.0 and 1.0, the failure was modeled by adding a random value to this output sampled from the uniform distribution between 0.0 and 2.0. The resulting noise caused failed rovers to take almost random actions. However, note that even rovers having control failures are still observing POIs and contributing to the system utility. Figure 8 shows the performance of the evaluations under this failure condition.

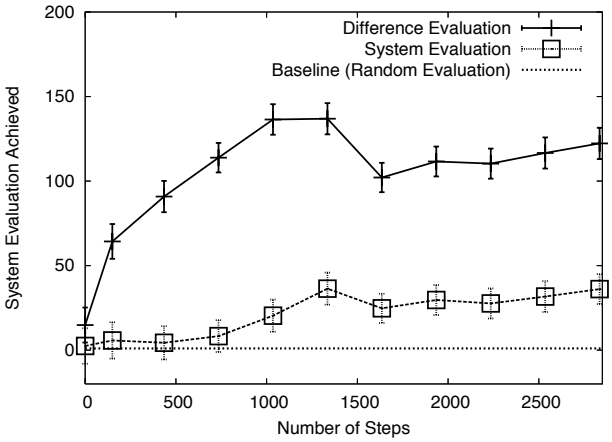


**Figure 8: Performance of a 30-rover system for two evaluation functions when at time step 1500, 60% of the rovers have control failures. Rover-sensitive evaluations are superior, allowing rovers to quickly adapt to failure.**

The failure caused a significant performance hit at time step 1500. Because in this case the failed rovers were taking nearly random actions, this was a significantly more complex control problem. Rovers using the difference evaluations were able to re-evolve and improve performance after the failure. However, in this instance, agents evolved with the difference evaluation were not able to achieve as high as performance after the failures than right before the failures. This was not an unexpected result, since a rover with a noisy controller not only failed to contribute to the system goals, but could prevent other rovers from achieving their tasks.

While the difference evaluation effectively removed a lot of noise from other agents, it could help when noise was being added to the end of the control process.

Figure 9 illustrates the different effects the different faults had on the multi-rover systems. While rovers using the system evaluation function performed poorly in general, no particular fault stands out statistically as being more problematic than others. For the difference evaluation though, failures involving noisy control were clearly more difficult to overcome.

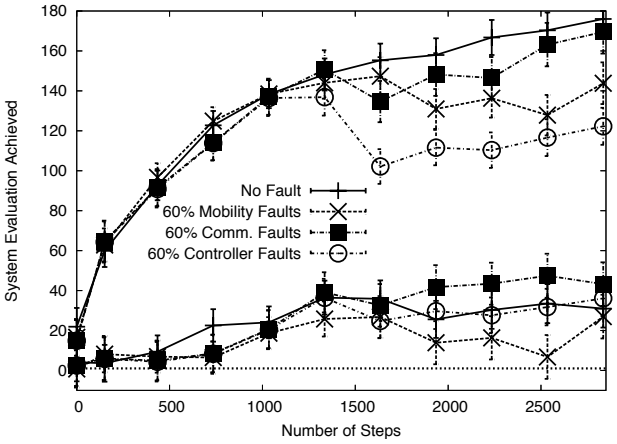


**Figure 9: Performance of a 30-rover system under different failure scenarios. Upper plots correspond to rovers using difference evaluations. Lower plots correspond to rovers using system evaluation.**

## 6. DISCUSSION

This paper shows that distributed evolutionary computation is a powerful tool in creating robust control policies for large systems. The key to making evolutionary computation effective is to use evaluation functions that are aligned with the system evaluation, yet more sensitive to the actions of individual controllers. This paper shows that one such evaluation is the *difference evaluation*, D. Systems of rovers evolved using D perform at much higher performance levels than systems of rovers evolved directly with the system evaluation. More importantly systems using D continue to achieve high performance levels when a majority of the rovers in the system experience various types of failures. These results show that evolutionary computation combined with the difference evaluation could be very successful in providing the robustness needed in high risk domains, including many aerospace and defense applications.

The results also show that the system more readily handles certain types of failures. These differences in failure responses can help mission planners decide what risks are acceptable to take for a particular mission. For example, having some rovers take actions that may cause them to loose communications is an acceptable risk, since other rovers can adapt to this loss. However having a rover take an action that may cause loss in control is much riskier, since it is harder for other rovers to adapt to such a failure. Knowledge of these tradeoffs can allow mission designers to make tradeoffs that optimize scientific payoff for long duration missions in challenging environments.

## 7. REFERENCES


[1] A. Agah and G. 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] 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, June 2004.

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

[5] 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.

[6] 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.

[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 (GECCO 2003)*, Chicago, Illinois, 2003.

[9] F. Hoffmann, T.-J. Koo, and O. Shakernia. Evolutionary design of a helicopter autopilot. In *Advances in Soft Computing - Engineering Design and Manufacturing, Part 3: Intelligent Control*, pages 201–214, 1999.

[10] M. J. Mataric. Coordination and learning in multi-robot systems. In *IEEE Intelligent Systems*, pages 6–8, March 1998.

[11] 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.

[12] K. Tumer and A. Agogino. Coordinating multi-rover systems: Evaluation functions for dynamic and noisy environments. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, Washington, DC, June 2005.

[13] K. Tumer and D. H. Wolpert. Collective intelligence and Braess' paradox. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 104–109, Austin, TX, 2000.

[14] D. Whitley, F. Gruau, and L. Pyeatt. Cellular encoding applied to neurocontrol. In *International Conference on Genetic Algorithms*, 1995.

[15] D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279, 2001.