Efficient Evaluation Functions for Evolving Coordination

A. Agogino Adrian.K.Agogino@nasa.gov
UCSC-NASA Ames Research Center, Mailstop 269-3, Moffett Field, CA 94035, USA

K. Tumer kagan.tumer@oregonstate.edu
Oregon State University, 204 Rogers Hall, Corvallis, OR 97331, USA

Abstract

This paper presents fitness evaluation functions that efficiently evolve coordination in large multi-component systems. In particular, we focus on evolving distributed control policies that are applicable to dynamic and stochastic environments. While it is appealing to evolve such policies directly for an entire system, the search space is prohibitively large in most cases to allow such an approach to provide satisfactory results. Instead, we present an approach based on evolving system components individually where each component aims to maximize its own fitness function. Though this approach sidesteps the exploding state space concern, it introduces two new issues: (1) how to create component evaluation functions that are aligned with the global evaluation function; and (2) how to create component evaluation functions that are sensitive to the fitness changes of that component, while remaining relatively insensitive to the fitness changes of other components in the system. If the first issue is not addressed, the resulting system becomes uncoordinated; if the second issue is not addressed, the evolutionary process becomes either slow to converge or worse, incapable of converging to good solutions.

This paper shows how to construct evaluation functions that promote coordination by satisfying these two properties. We apply these evaluation functions to the distributed control problem of coordinating multiple rovers to maximize aggregate information collected. We focus on environments that are highly dynamic (changing points of interest), noisy (sensor and actuator faults), and communication limited (both for observation of other rovers and points of interest) forcing the rovers to evolve generalized solutions. On this difficult coordination problem, the control policy evolved using aligned and component-sensitive evaluation functions outperforms global evaluation functions by up to 400%. More notably, the performance improvements increase when the problems become more difficult (larger, noisier, less communication). In addition we provide an analysis of the results by quantifying the two characteristics (alignment and sensitivity discussed above) leading to a systematic study of the presented fitness functions.

Keywords

Evolution strategies, distributed control, fitness evaluation.

1 Introduction

In many continuous control tasks such as pole balancing, robot navigation, and rocket control, using evolutionary computation methods to develop controllers based on neural networks has provided successful results as shown in Stanley and Miikkulainen (2002); Floreano and Mondada (1994); Miglino et al. (1995); and Gomez and

Miikkulainen (1999, 2003). Extending those successes to distributed domains such as coordinating multiple robots, controlling constellations of satellites, and micro device control promises significant application opportunities as discussed in Wu et al. (1999); Martinoli et al. (1999); Tumer and Wolpert (2004a,b); and Agogino and Tumer (2004a). The goal in such distributed control tasks is to evolve a large set of separate components that collectively strive to maximize a global evaluation function.

Approaching the design of a complex, multicomponent system directly by an evolutionary algorithm (e.g., having a population of system controllers and having the evolutionary operators work directly on the system controllers to produce a solution with high global fitness) is appealing but is often impractical or impossible. The search space for such an approach is simply too large for all but the simplest problems (Tumer and Wolpert, 2004b). A more promising solution is to evolve control policies for individual components by having each of them use their own fitness evaluation function. The key issue in such an approach is to ensure that each component's fitness evaluation function possesses the following two properties: (1) it is aligned with the global evaluation function, ensuring that components maximizing their own fitness do not hinder one another and hurt the fitness of the full system; and (2) it is sensitive to the fitness of the component, ensuring that it provides the right selective pressure on the component (i.e., it limits the impact of other components in the fitness evaluation function).

In this paper we focus on evolving a set of data gathering rovers that attempt to maximize the aggregate information collected by all the rovers. This problem possesses a number of properties that make it both a very interesting problem and a particularly challenging one for evolutionary algorithms:

1. The environment is dynamic, meaning that the conditions under which the rovers evolve changes with time. The rovers need to evolve general control policies, rather than specific policies tuned to their immediate environment.

2. The sensors are noisy, meaning that the signals they receive from the environment are not reliable. Each rover needs to demonstrate that its control policies are not sensitive to such fluctuations in sensor readings.

3. The rovers have limited sensing capabilities, meaning that their representation of their surroundings is restricted. The rovers need to formulate policies that satisfy the global evaluation function based on limited, local information.

4. The number of rovers in the system is large, meaning that the rovers cannot be fully aware of the actions of all the other rovers. The rovers need to decouple the impact of other rovers' actions from their evaluation functions.

1.1 Related Work

Evolutionary computation has been used in a wide variety of domains differing along many attributes. It has been used historically in single-component domains, but has expanded to multicomponent ones. Also, within multicomponent domains it has been used with cooperative components as well as with local greedy components. More closely related to this paper, evolutionary computation has also been used with multirobotic domains. This section gives a brief summary of prior use of evolutionary computation among these different domains.

Evolutionary computation has a long history of success in single-component (single-agent, single-robot, etc.) control problems. Examples of such successes are described

in Whitley et al. (1995); Hoffmann et al. (1999); Farritor and Dubowsky (2002); Agogino et al. (2000); Lamma et al. (2001); Gomez and Miikkulainen (2003). Improvements in search methods are the driving force of advances in single-component evolutionary control algorithms. Hoffmann et al. (1999) describes fuzzy rules in a helicopter control problem that provide such an advance, as does Whitley et al. (1995), which describes a cellular encoding on pole-balancing control. Similarly, Farritor and Dubowsky (2002) show how searching through a space of plans generated from a planning algorithm with a genetic algorithm provides good control policies in a planetary rover control problem. In addition Gomez and Miikkulainen (2003) describes how the use of "subpopulations" used with the ESP (enforced subpopulations) algorithm can be effective in controlling rockets.

Evolutionary computation has also been applied to multi-component domains, especially in the field of collaborative coevolution (Panait et al., 2006; Hoen and de Jong, 2004; Sen et al., 2003; Mundhe and Sen, 2000; Potter and de Jong, 2000). In collaborative coevolution, agents evolve their own populations and collaborate to form a good global solution. Research in this field involves making this process more efficient and more likely to reach good global solutions. In Panait et al. (2006) this is done by having agents try to maximize a global evaluation function through a process of finding good collaborators that avoid suboptimal equilibria. In Hoen and de Jong (2004) the coevolution process is sped up through the shaping of the fitness function in a similar manner to that used in this paper. Similarly, in Mundhe and Sen (2000) evaluation functions are scaled to prevent a "tragedy of the commons" where agents evolve to over-compete for a common resource.

Outside of cooperative coevolution, evolutionary algorithms have also been used in multi-component systems. Ant colony algorithms solve the coordination problem by utilizing "ant trails" that provide implicit fitness functions resulting in good performance in path-finding domains as described in Dorigo and Stützle (2004). In Agogino et al. (2000) it is shown that in some domains, a large number of agents using greedy nonfactored utilities can be used to speed up the evolution process. In Lamma et al. (2001) it is shown that beliefs about other agents can also be used to improve the control policies through using global and hand-tailored fitness functions. In the field of game theory, evolutionary computation has been used in the prisoner's dilemma problem for two-agent and N-agents respectively as shown in Harrald and Fogel (1996) and Yao and Darwen (1994). Also, by combining the fields of reinforcement learning and genetic algorithms, Mikami et al. (1996) presented a method to improve performance in a multicomponent system.

Evolution has also been used to coordinate robots in multirobotic domains. In Agah and Bekey (1996) genetic algorithms were used to control colonies of robots by promoting coordination by manipulating the "likes" and "dislikes" of each robot. Evolutionary methods have also been used in the domain of scheduling constellations of satellites as shown in Globus et al. (2003). Finally, outside evolutionary computation (but using related evaluation functions), Mataric (1998) describes how coordination between a set of mobile robots can be accomplished through the use of domain specific rewards designed to prevent greedy behavior.

1.2 Contributions of this Paper

This paper builds upon earlier work by the authors and presents a comprehensive study of the impact of using principled evaluation function selection for evolving coordinated multirover systems. This concept and the particular rover domain was first introduced

in Agogino and Tumer (2004a). That work explored the basic concept in an "offline" manner where the rovers went through repeated "episodes" (multiple trials) in training where they performed the same task. This manner of learning was appropriate for domains where rovers could be trained ahead of time and then deployed in a similar domain to perform a specific set of tasks. We subsequently explored the use of "online" training where the rovers evolved through one continuous task, and briefly explored scaling and communication issues in Agogino et al. (2005) and Tumer and Agogino (2006a). In addition, extensions of the above work to scaling and dynamic environments were presented in Tumer and Agogino (2006a,b, 2007). Finally, the impact of failing rovers on system level coordination was investigated in Agogino and Tumer (2006).

In this work, we provide a comprehensive study of shaping evaluation functions for evolving coordinated rover teams operating in dynamic and noisy environments. We extend the results above by providing more detailed scaling results as well as two types of restrictions on rover sensing capabilities (rovers cannot see/communicate with other rovers and rovers cannot see points of interest). In addition, we provide three sets of new experiments:

1. We explore letting the rovers share a population of control strategies. We investigate the impact of this approach in terms of speed of convergence and solution quality and compare it to approaches where each rover keeps its own population of control strategies.

2. We provide a quantification of the relationship between a rover's evaluation function and the system level evaluation function. This concept first explored for visualizing reinforcement learning rewards was introduced in Agogino and Tumer (2005). We use this method for evaluation function analysis and gain insight into how to modify evaluation functions in the presence of limited information.

3. We provide a new dynamic domain simulating surveillance problems where the points of interest continually move. This provides a test on whether the evaluation functions can learn patterns at a higher level than the time frames in which they perform their navigation tasks.

This paper is organized as follows. In Section 2.1 we present the properties needed for good component-specific evaluation functions, and how those properties lead to good system level control policies. In Section 3 we present the "Rover Problem" where a system of multiple planetary rovers use neural networks to determine their movements based on a continuous-valued array of sensor inputs. In Section 4 we describe the experimental setup. In Section 5 we present extensive simulation results in a wide array of environments, types of evolutionary algorithms, and POI distribution. In Section 6 we provide a detailed analysis of the different fitness functions and explicitly evaluate their factoredness and sensitivity. These results show that we have found a good predictor of system performance. Finally in Section 7 we discuss the implication of these results and their applicability to different domains.

2 Evolving Multicomponent Systems

Designing control policies for systems with many components through evolution is often approached in one of the following three ways (also shown in figure 1):

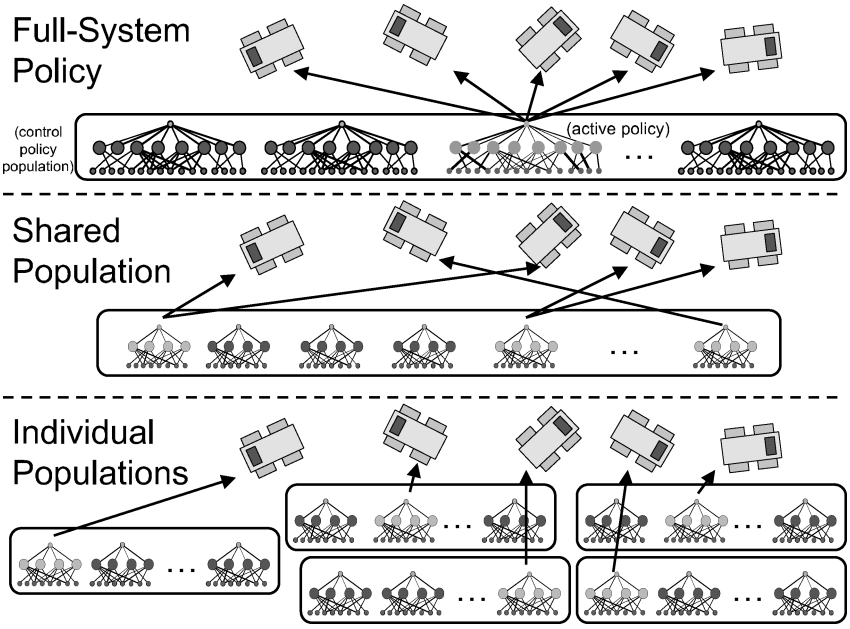

Figure 1: Three policy control methods. Upper diagram: Each control policy controls all rovers. Middle: Each rover chooses its own control policy from single population of controllers. Lower: Each rover has its own population of controllers.

1. One can have a single population of full-system control policies. During training a policy is chosen from the population and this policy explicitly controls the individual actions of *all the components* (note that this policy is extremely complex as each component may be performing different actions). In this approach a single genotype encodes the (possibly different) control strategies of all the components. A standard evolutionary algorithm is then used to select a full-system control policy that best satisfies a predetermined global evaluation function.

2. One can have a single shared population of individual component control policies. During training, each component independently selects a control policy for itself from the shared populations (components may therefore have different policies). In this approach, one genotype encodes the control strategy of a single component, but there is a single population of control strategies from which the components select their genotype. An evolutionary algorithm then evolves the shared population to create control policies that best maximize the global evaluation.

3. One can evolve individual components separately, generating a different control policy for each component. In this case, each component has its own population of policies. In this approach, one genotype encodes the control strategy of one component and each component keeps an independent population of control strategies. The evolutionary algorithms are then used to select a control policy based on how a given component using that control policy satisfies an evaluation function. This approach can be further divided into two categories according to the nature of that evaluation function:

(a) Each component is evaluated based on *the same global fitness function.*
(b) Each component is evaluated based on *a component-specific evaluation function tuned to the fitness of that component.*

The first method presents a computationally daunting task in all but the simplest problems. In this approach one is looking specifically for the best "collective" action. Finding good control strategies is difficult enough for single controllers, but the search space becomes prohibitively large when they are concatenated into an "individual" representing the full system. Even if there were components with good control policies present in the system, there would be no mechanism for isolating and selecting them when the system to which they belong performs poorly. As a consequence, this approach has serious scaling issues, particularly in large continuous domains.

The second method avoids the problem of having controllers with huge state and action spaces and often works well when all the components do similar things (Agogino et al., 2000). This method can lead to fast convergence since the shared population of controllers is updated by every component in the system. However, since there is a single population of controllers for all of the components, each controller has to be generalized enough to handle the needs of a variety of different components. This need for generalization could necessitate the need for more complex controllers and could increase convergence times. We provide a discussion on this approach and present brief results in Section 5.5.

The third method addresses part of the generalization problem in that each component evolves its own population of controllers (Panait et al., 2006). Figure 2 shows the general principle behind this approach where each component evolves its own control policy from its own population of policies. However, this method introduces a new problem: What is the selective pressure on each component? In approach 3(a) all the components receive the same evaluation values based on the global evaluation of the entire system. This approach encourages components to evolve in such a way as to try to maximize the global evaluation, but also results in a component's evolution being guided by the fitness of all the other components. When there are few components, this method provides good solutions, but as the number of components increases, it becomes increasingly difficult to isolate the impact of a component on the system. For instance, in a domain with 100 rovers acting simultaneously, a rover that is performing well may receive a poor evaluation if many other rovers happen to be performing poorly at the same time. As a consequence, this approach, though preferable to the first two approaches in many ways, suffers from a signal-to-noise problem where the selective "signal" for a component is drowned by the noise of all the other components' impact on the evaluation function.

In approach 3(b), on the other hand, the use of a component specific evaluation eliminates this signal-to-noise problem. This approach enables us to create fitness evaluation functions that are more tailored to a specific component, but introduces a new twist to the problem: How does one ensure that the specialized component evaluation functions are aligned with the global evaluation function? In other words, how do we ensure that a system in which all the components evolve according to their own fitness functions has a high global fitness? In this paper we focus primarily on approaches 3(a) and 3(b), discuss what the desirable properties of component evaluation functions are, and show how such evaluation functions provide good distributed control strategies.

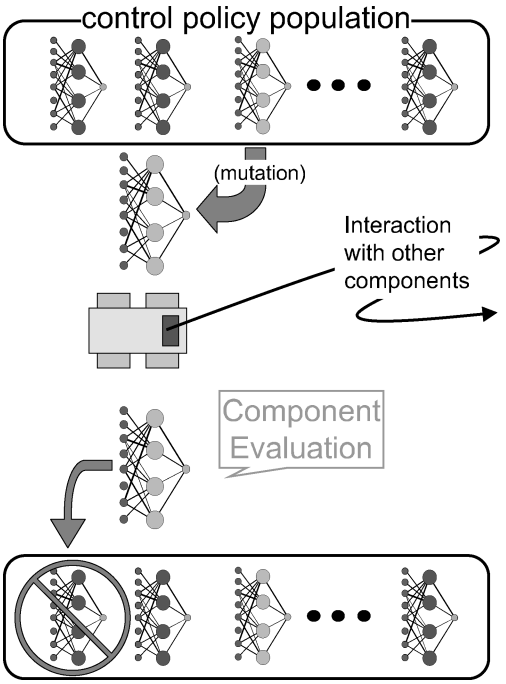

Figure 2: Evolution process for single component. A rover (a single component of a larger system) chooses a control policy from its own population of control policies. It then uses it for control. After evaluating the control policy's effectiveness, the rover updates its population.

2.1 Properties of Component Evaluation Functions

Component-specific evaluations need to relate to the global evaluation function in a particular way for a full system to evolve properly. Fortunately by looking at two particular properties of this relation, we can create component-specific evaluations that will lead to high global performance. This analysis is based on work on "collectives" (also dubbed COllective INtelligence, or COIN) which principally applies to utility-based agents (Wolpert and Tumer, 2001; Tumer and Wolpert, 2004b). Here we apply this theory to the derivation of evaluation functions for evolving a team of coordinated agents/components. We will now formally define these properties.

In what follows, we will denote the global evaluation function by $G(z)$, where z is the state of the full system (e.g., the position of all the components in the system, along with their relevant internal parameters and the state of the environment). We will then denote the component evaluation function for component i by $g_i(z)$.

2.1.1 Factoredness for Fitness Functions

First we want the component evaluation functions of each component to have high *factoredness* with respect to G, intuitively meaning that an action taken by a component that improves its own evaluation function also improves the global evaluation function (i.e., G and g_i are aligned). Formally, the degree of factoredness between g_i and G is

given by (Tumer and Wolpert, 2004b):

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

where z' is a state that only differs from z in the state of component 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 component evolved to maximize g_i will also maximize G.

2.1.2 Sensitivity of Fitness Functions

Second, the component evaluation function must be more sensitive to changes in that component's fitness than to changes in the fitness of other components in the system (Agogino and Tumer, 2004b; Wolpert and Tumer, 2001). Formally we can quantify the *sensitivity* of evaluation function g_i, at z as:

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

where $E_{z'}[\cdot]$ is the expected value taken over z', and $(z - z_i + z'_i)$ notation specifies the state vector where the vector-elements corresponding to component i have been removed from state z and replaced by the vector elements of component i from state z'. The numerator of the fraction represents component i's influence on its evaluation function and the denominator represents all the other components' influence on component i's evaluation. So at a given state z, the higher the sensitivity, the more $g_i(z)$ depends on changes to the state of i, that is, the better the associated signal-to-noise ratio for i. Intuitively then, higher component-sensitivity means there is "cleaner" (e.g., less noisy) selective pressure on i.

As an example, consider the case where the component evaluation function of each component is set to the global evaluation function, meaning that each component is evaluated based on the fitness of the full system (e.g., approach 3(a) discussed in Section 2). Such a system will be fully factored by the definition of Equation 1. However, the component fitness functions will have low sensitivity (the fitness of each component depends on the fitness of all the other components).

2.2 Difference Evaluation Functions

Let us now focus on improving the sensitivity of the evaluation functions. To that end, consider difference evaluation functions, previously shown in Wolpert and Tumer (2001), which are of the form:

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

where z_{-i} contains all the states on which component i has no effect, and c_i is a fixed vector. In other words, all the vector-elements of z that are affected by component i are replaced with the fixed values from 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 as shown in Wolpert and Tumer (2001) (e.g., D and G will have the same derivative with respect to z_i). Furthermore, they usually have far better sensitivity

than does a global evaluation function, because the second term of D removes some of the effect of other components (i.e., noise) from i's evaluation function. In many situations it is possible to use a c_i that is equivalent to taking component 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 component's contribution to the global evaluation.

While it is the case that for linear evaluation functions, D_i simply cancels out the effect of other components in computing component i's evaluation function, its applicability is not restricted to such functions. In fact, it can be applied to any linear or nonlinear global evaluation function. However, its effectiveness is dependent on the domain and the interaction among the component evaluation functions. At best, it fully cancels the effect of all other components. At worst, it reduces to the global evaluation function, unable to remove any terms (e.g., when z_{-i} is empty, meaning that component i affects all states). In most real world applications, it falls somewhere in between, and has been successfully used in many domains including agent coordination, satellite control, data routing, job scheduling, and congestion games as described in Agogino and Tumer (2004a); Tumer and Wolpert (2000); and Wolpert and Tumer (2001). Also note that the computation of D_i is a "virtual" operation in that component i computes the impact of its not being in the system. There is no need to re-evolve the system for each component to compute its D_i, and computationally it is often easier to compute than the global evaluation function. Indeed in the problem presented in this paper, for component i, D_i is easier to compute than G is (see discussion in Section 3.4).

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 (Agogino and Tumer, 2004a). The "full system" in this formulation is the set of rovers, while the components of the system are the individual rovers. In this problem, multiple rovers are trying to observe points of interest (POIs) on a two-dimensional plane. 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,[1] $\delta_{\min}$:

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

where x and y are location coordinates. The global evaluation function is given by:

$$G = \sum_{t=1}^{\tau} \sum_{j} \frac{V_j}{\min_i \delta(L_j, L_{i,t})}, \tag{5}$$

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. Here the global evaluation is a sum of single-time-step evaluation taken over

[1]The squared 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.

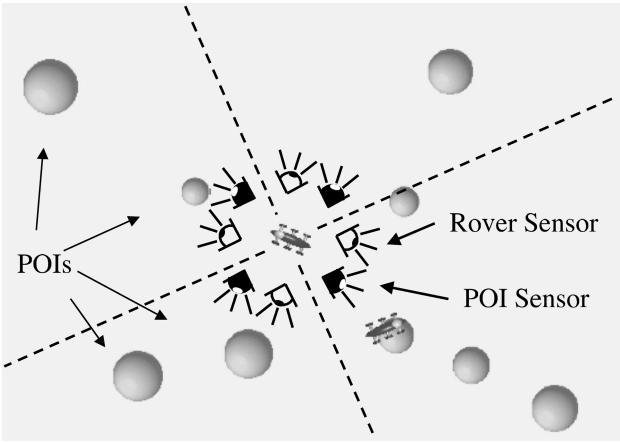

Figure 3: Diagram of a rover's sensor inputs. The world is broken up into four quadrants relative to the rover's position. In each quadrant one sensor senses points of interest, while the other sensor senses other rovers.

a fixed number (τ) of time steps. Intuitively, while any rover can observe any POI at every time step, as far as the global evaluation function is concerned, only the closest observation matters.[2] In our experiments, the global evaluation is computed over a time window of size τ.

3.1 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. This value is then scaled by the angle between the line from the rover's location to the POI and the line from the rover's location to center of the quadrant:

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

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 squared distances from a rover to all the other rovers in the quadrant at time t scaled

[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 3D imaging may utilize different images of the same object, taken by two different rovers.

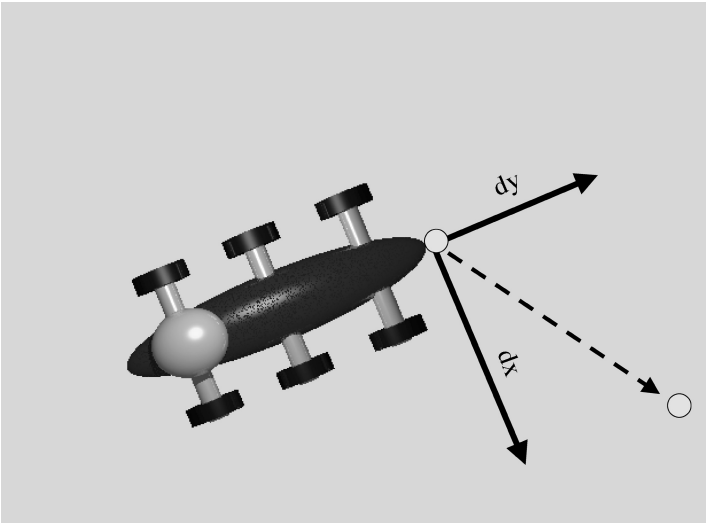

Figure 4: Diagram of a rover's movement. At each time step the rover has two continuous outputs $(\mathrm{d}x, \mathrm{d}y)$ giving the magnitude of the motion in a two-dimensional plane relative to the rover's orientation.

by the angle:

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

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. The term $1 - \frac{|\theta_{i',q}|}{45}$ smoothes the transition between quadrants, resulting in slightly higher performance than if the term were not there, but is not necessary for rovers to perform well.

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 trade-offs 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 two sensors present in each region at all times, this setup can also be implemented with only two sensors which perform a sweep at 90° increments every time step.

3.2 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. Figure 4 displays the orientation of a rover's output space.

The mapping from rover state to rover output is done through a Multi Layer Perceptron (MLP), with eight input units, 10 hidden units, and two output units (Haykin,

1994).[3] The MLP uses a sigmoid activation function, therefore the outputs are limited to the range (0, 1). The actual rover motions $\mathrm{d}x$ and $\mathrm{d}y$ 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:

$$\mathrm{d}x = 2d_{\max}(o_1 - 0.5)$$
$$\mathrm{d}y = 2d_{\max}(o_2 - 0.5),$$

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.

3.3 Rover Selection

The MLP for a rover is trained using an evolutionary algorithm as summarized in figure 2, where the control policies are MLPs. Each rover has a population of MLPs. At each τ time step (where τ is set to 15 in these experiments), the rover uses epsilon-greedy selection ($\epsilon = 0.1$) to determine which MLP it will use (i.e., 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 τ steps. At the end of those τ steps, the MLP's performance is evaluated by the rover's evaluation function and reinserted into its population of MLPs, at which time, the poorest performing member of the population is deleted. Both the global evaluation for system performance and rover evaluation for MLP selection are computed using a τ-step window, meaning that the rovers only receive an evaluation after τ steps. The pseudocode for this process is as follows:

1. At t=0 initialize MLP population
2. Pick an MLP using epsilon-greedy alg ($\epsilon = 0.1$)
3. Randomly modify MLP (mutation)
4. Use MLP to control rover for 15 steps
5. **Evaluate MLP performance (i.e., compute G, P_i, D_i as shown in Section 3.4)**
6. Reinsert MLP into pool
7. Delete worst MLP from pool
8. Go to step 2

The main emphasis of this paper is on step 5, and how the proper balance between factoredness and sensitivity in the evaluation function promotes better and faster evolution of the system. While the rest of the evolutionary algorithm is not sophisticated, it is ideal in this work since our purpose is to demonstrate the impact of principled evaluation function selection on the performance of a distributed system. Even so, this

[3] 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.

algorithm has been shown to be effective if the evaluation function used by the rovers is factored with G and has high sensitivity. More advanced algorithms from evolutionary computation, used in conjunction with these same evaluation functions, could improve the global performance further.

3.4 Evolving Distributed Control Strategies

The key to success in this approach is to determine the correct rover evaluation functions. In this work we test three different evaluation functions for rover control, which are the global evaluation function, a perfectly rover-sensitive evaluation function, and the difference evaluation function. In this section, we will present the formulation of the three evaluation functions (following some notational details to make the presentation more precise).

All evaluation functions are a sum of single-time-step evaluations computed over τ time steps. They are computed over a window of size τ every τth time step. The locations of POIs are constant across the τ time steps of an evaluation. All evaluation functions used are a function of the location table L which contains the locations of the POIs and the rovers for time steps from 1 to τ. We use the notation $L_{i,t}$ to refer to the location of rover i at time step t. We use the notation L_i to refer to the set of locations of rover i across all τ time steps. Also, we use the notation L_j to refer to the location of POI j.

- The first evaluation function is the global evaluation function (G), which when implemented results in approach 3(a) discussed in Section 2:

$$G(L) = \sum_{t=1}^{\tau} \sum_{j} \frac{V_j}{\min_i \delta(L_j, L_{i,t})}. \tag{8}$$

- The second evaluation function is the "perfectly rover-sensitive" evaluation function (P), which follows one instantiation of approach 3(b) discussed in Section 2:

$$P_i(L) = \sum_{t=1}^{\tau} \sum_{j} \frac{V_j}{\delta(L_j, L_{i,t})}. \tag{9}$$

  The P evaluation function is equivalent to the global evaluation function in the single rover problem. In a multi-rover setting, it has infinite sensitivity (in the way sensitivity is defined in Section 2). This is because the P evaluation function for a rover is not affected by the states of the other rovers, and thus the denominator of Equation 2 is zero. However, the P evaluation function is not factored. Intuitively, P and G offer opposite benefits, since G is by definition factored, but has poor sensitivity.

- The final evaluation function is the difference evaluation function which also is an instantiation of approach 3(b) discussed in Section 2: It does not have as high sensitivity as P, but is still factored with G:

$$\begin{aligned} D_i(L) = &\quad G(L) - G(L - L_i) \\ &= \sum_{t=1}^{\tau} \sum_{j} I_{j,i,t}(z) \left[\frac{V_j}{\delta(L_j, L_{i,t})} - \frac{V_j}{\delta(L_j, L_{k_j,t})} \right], \end{aligned}$$

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 the D is equal to the value of all the information 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 in this case is easier to compute as long as rover i knows the position and distance of the closest rover to each POI it can see. In that regard it requires knowledge about the position of fewer rovers than if it were to use the global evaluation function.

4 Simulation Description

In this section we provide a description of the simulation used in this paper, including the rover characteristics, the environment specification, and the experimental setup. In our experiments, a set of rovers started at the center of the domain world and navigated the domain, observing POIs throughout a single trial. The environment slowly changed throughout the trial, but was never reset at any time. At the beginning of the trial their control policies were random, so they had to evolve control policies that allow them to effectively observe POIs in a way that led to high global evaluation.

4.1 Environment Specification

In these experiments 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. The domain was dynamic, where POIs appeared and disappeared throughout the course of the trial. This dynamic simulates the effect of collecting an exhaustive amount of information about a point of interest so that it becomes irrelevant, and discovering new points of interest that need to be observed. Each POI had a probability of 2.5% of disappearing each $\tau = 15$ time steps. Also, a new POI had a probability of 2.5% of appearing at $\tau = 15$ time step intervals. Because the experiments were run for 4,000 time steps, the initial and final environments had little similarity. Figure 5 shows changes to the environment throughout a sample simulation. Though the POI sets at $t = 10$ and $t = 120$ share many similarities, the POI set at $t = 1{,}500$ is entirely different from the POI set at $t = 10$. As a consequence, the rovers need to evolve efficient policies that are solely based on their sensor inputs and not on the specific configuration of the POIs.

4.2 Rover Specification

In all these experiments, each rover had a population of MLPs of size 10. The world was 70 units long and 70 units wide, unless otherwise stated (as in the scaling experiments of Section 5.2). All of the rovers started the experiment at the center of the world unless otherwise stated (as in the population sharing experiments of Section 5.5). There were 30 rovers and 30 POIs in the simulations, unless otherwise stated (as in the scaling experiments of Section 5.2). The maximum distance the rovers could move in one direction during a time step, $d_{\max}$, was set to three. The rovers could not move beyond the bounds of the world. The minimum observation distance, $\delta_{\min}$, was equal to five.

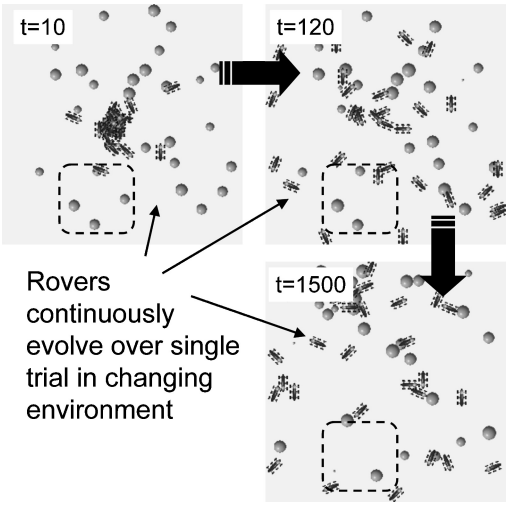

Figure 5: Sample POI placement. Left top: Environment at time = 10. Right top: Environment at time = 120. Bottom: Environment at time = 1,500. Environment at time step 10 is similar to environment at time step 120, but significantly different from the environment at time step 1,500. Rovers evolve continuously in a single dynamic environment. Environment and rover locations are never reset. Rovers must be able to use their control policies evolved from an earlier time step, in future changed environments.

4.3 Statistical Significance of Results

All results presented were averaged over 100 independent trials (except for the 70 rover domains, where the results were averaged over 30 trials). In all cases, performance was measured by full system fitness, regardless of the evaluation function used to evolve the system. On all the performance graphs, a performance level of 1.0 represents the case in which all rovers were optimally positioned on top of a POI. Note that in practice a performance level of 1.0 was not possible since the environment kept changing, and it took time for rovers to reposition themselves.

5 Experimental Results

We performed an extensive set of simulations to test the effectiveness of the different rover evaluation functions under a wide variety of environmental conditions and rover capabilities. All the environments were highly dynamic, so rovers had to learn generalized policies that worked under a wide set of configurations. Our experiments were broken down into the following categories:

1. Performance of rovers with noise-free sensors.
2. Scaling properties for rovers with noise-free sensors.
3. Performance of rovers with noisy sensors.
4. Performance of rovers with noisy sensors and limited communication/sensing with:

(a) restrictions on rover to rover communication; and
(b) restrictions on both rover to rover communication and POI sensing.

5. Performance of rovers sharing population of control policies.

6. Performance of rovers in a surveillance environment where POIs move, where:

(a) POIs move more slowly than rovers; and
(b) POIs move faster than rovers.

These sets of experiments were performed online in a dynamic environment. The rovers evolved throughout a single trial, forming control policies online to meet the needs of the environment. This contrasts to many multi-trial or "episodic" methods of evolution where evolution is performed in simulation with the simulated environment being reset at the beginning of each trial. The online approach used in this paper is more widely applicable as it allows for evolution to occur in rovers that are actively deployed in real-world environments that cannot be reset. It can also be used in augmentation to a multi-trial approach where initial policies are evolved offline and then refined online to correct for errors in the original simulation. This augmentation is especially powerful with the presented controllers that map sensor inputs directly to control outputs. This mapping forces the rovers to develop general control policies, rather than learn a specific mapping from fixed x, y locations to outputs. As a consequence, the rovers evolve policies that are applicable to different environments and can be perfect complements to policies evolved offline.

5.1 Performance in Noise-Free Environments

Figure 6 shows the performance of systems evolved using the three different evaluation functions. All three evaluation functions performed adequately in this instance, though D outperformed both P and G. The convergence properties of this system demonstrate the different properties of the rover evaluation functions. After initial improvements, the system with the G evaluation function improves slowly. This is because the G evaluation function has low sensitivity. Because the fitness of each rover depends on the state of all other rovers, the noise in the system overwhelms the evaluation function.

Evaluation P, on the other hand, has a different problem: After an initial improvement, the performance of the systems using this evaluation function declines. This decline happens since even though P has high rover-selectivity, it is not fully factored with the global evaluation function. This means that rovers selected to improve P do not necessarily improve G, and may end up performing many actions that do not contribute to the overall goals of the system. In contrast, D is both fully factored and has high sensitivity. As a consequence, rovers using D continue to improve well into the simulation as the fitness signals the rovers receive are not swamped by the states of the other rovers in the system. This simulation highlights the need for having evaluation functions that are both factored with the global evaluation function and have high sensitivity. Having one or the other is not sufficient.

5.2 Scaling in Noise-Free Environments

Figure 7 shows the scaling properties of the three evaluation functions in a dynamic noise-free environment. The performance of each evaluation function at $t = 3{,}000$ for

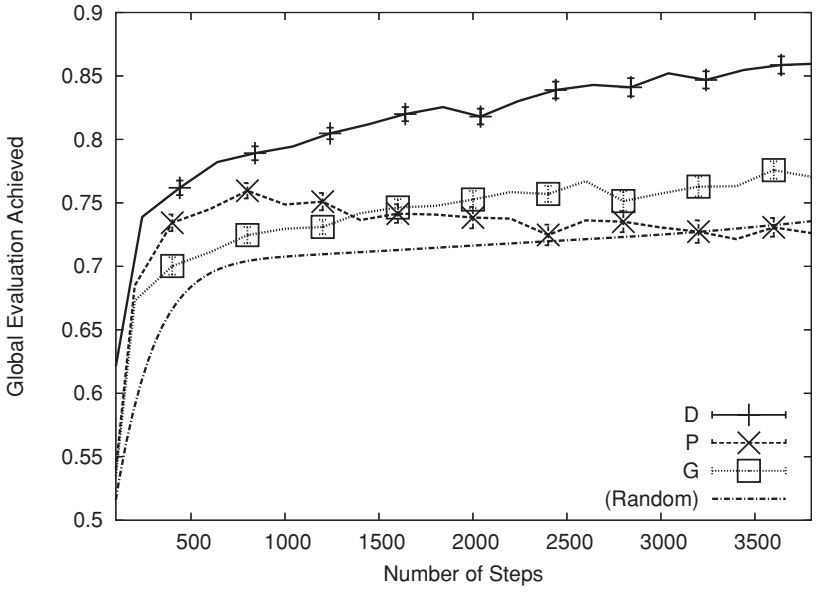

Figure 6: Performance of the 30-rover system for all three evaluation functions in a noise-free environment. The difference evaluation function provides the best system performance because it is both factored and rover-sensitive.

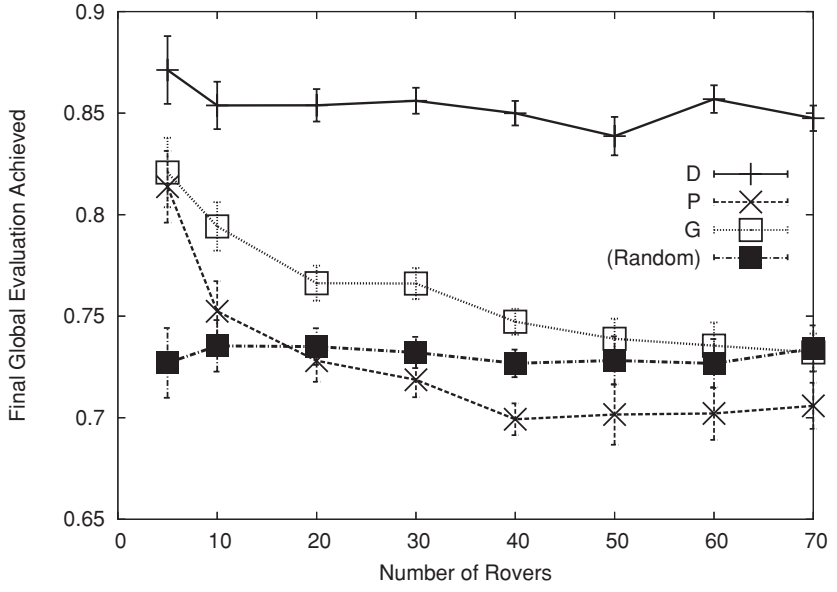

Figure 7: Scaling properties of the three evaluation functions. The D evaluation function not only outperforms the alternatives, but the margin by which it outperforms them increases as the size of the system goes up.

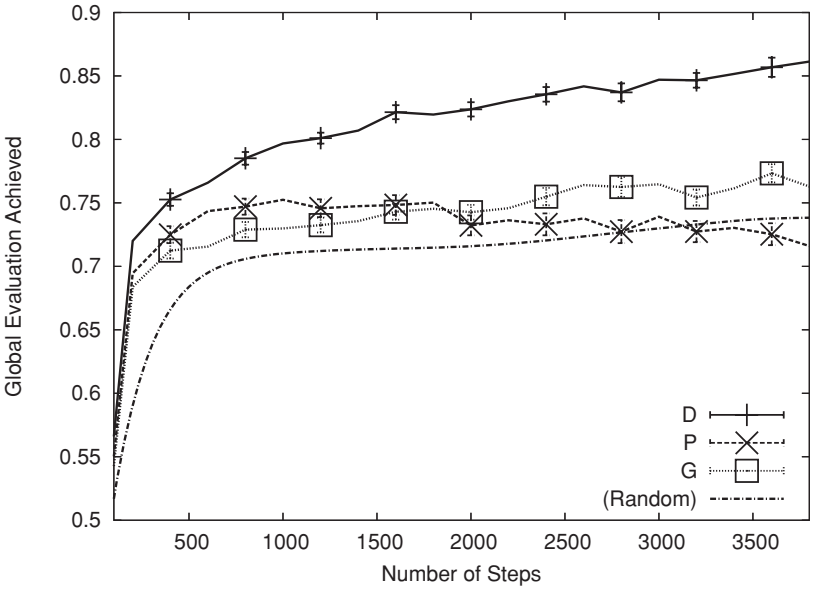

Figure 8: Performance of the 30-rover system for all three evaluation functions when the rover sensors and outputs have 20% noise. Performance of D evaluation is still high with moderate amounts of noise.

systems ranging from 10 to 70 rovers is plotted. For each size of system, the number of POIs equals the number of rovers and the domain is scaled to have the same ratio of surface area to number of rovers. For each of these different system sizes, the results are qualitatively similar to those reported above, except when there are only five rovers, in which case P performs as well as G. This is not surprising since with so few rovers, there are almost no interactions among the rovers.

As the size of the system increases, an interesting pattern emerges. The performance of both P and G drop at a faster rate than that of D. Again, this is because G has low sensitivity and thus the problem becomes more pronounced as the number of rovers increases. Similarly, as the number of rovers increases, P becomes less and less factored. In fact, for more than 20 rovers P performs worse than random, creating a "tragedy of the commons" situation where each rover trying to maximize its own evaluation function leads to a poor global solution (Hardin, 1968). D, on the other hand, handles the increasing number of rovers quite effectively. Because the second term in Equation 3 removes the impact of other rovers from rover i, increasing the number of rovers does very little to limit the effectiveness of this rover evaluation function. This is a powerful result suggesting that D is well suited to evolve large systems in this and similar domains where the interaction among the rovers prevents both G and P from performing well.

5.3 Performance with Noisy Sensors and Actuators

Figure 8 shows the performance of the three evaluation functions in a dynamic environment for 30 rovers with noisy sensors. In these experiments, both the input sensors and the output values of the rovers have 20% noise added. All three evaluation functions

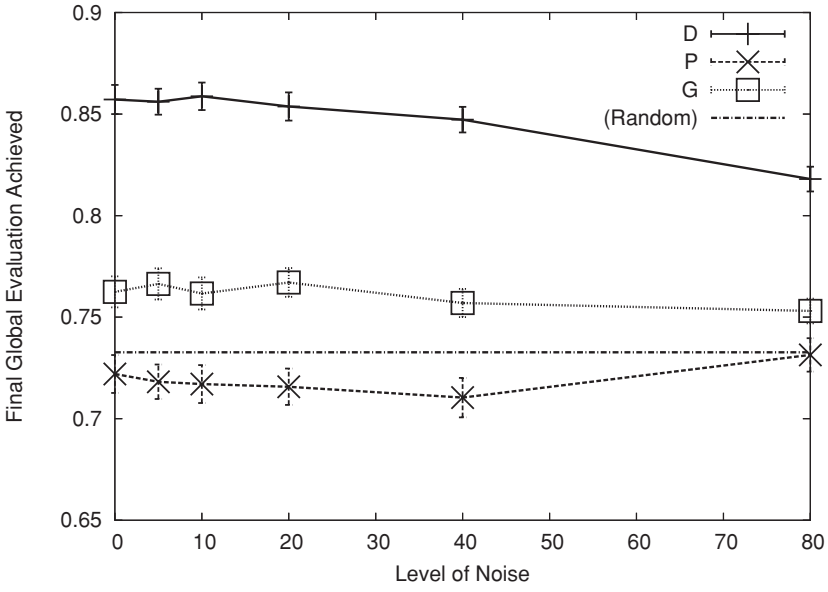

Figure 9: Sensitivity of the three evaluation functions to the noise in their sensors. All evaluations can handle large amounts of noise in this domain.

handle the noise well. This result is encouraging in that it shows that not only simple evaluation functions such as P can handle moderate amounts of noise in their sensors and outputs, but so can D. In other words, considering the impact of other rovers to yield a factored evaluation function does not compound moderate noise in the system and overwhelm the rover evaluation. Indeed, the similarity between the experiments with and without noise leads us to explore the effect of noisy sensors and actuators in more detail.

Figure 9 shows the noise sensitivity of the three different evaluation functions. The performance is reported as a function of additive noise to sensors as the percentage shown on the x-axis (e.g., 50 means the magnitude of the added noise is 50% of the sensor value.) The results show that the D evaluation is the most sensitive to high levels of noise, though even at 80% noise it still far outperforms both G and P. This is an encouraging result in the power of the D evaluation function in that it "cleans up" the evaluation function for a rover (e.g., it has high sensitivity). Adding noise starts to cancel this property of D, but even when nearly half the signal is noise, this does not prevent D from far outperforming G and P. Interestingly, rovers using P perform marginally better as noise increases. Adding noise to the system actually hindered these rovers from evolving the wrong policies, in effect decreasing the tragedy of the commons by preventing the rovers from evolving "good" selfish control policies.

5.4 Evolution with Communication Limitations

All the experiments presented so far are based on rovers receiving the required information to compute all three evaluation functions. In the presence of communication or sensing limitations, however, it is not always possible for a rover to compute its exact D, nor is it possible for it to compute G. In such cases, D can be computed based on

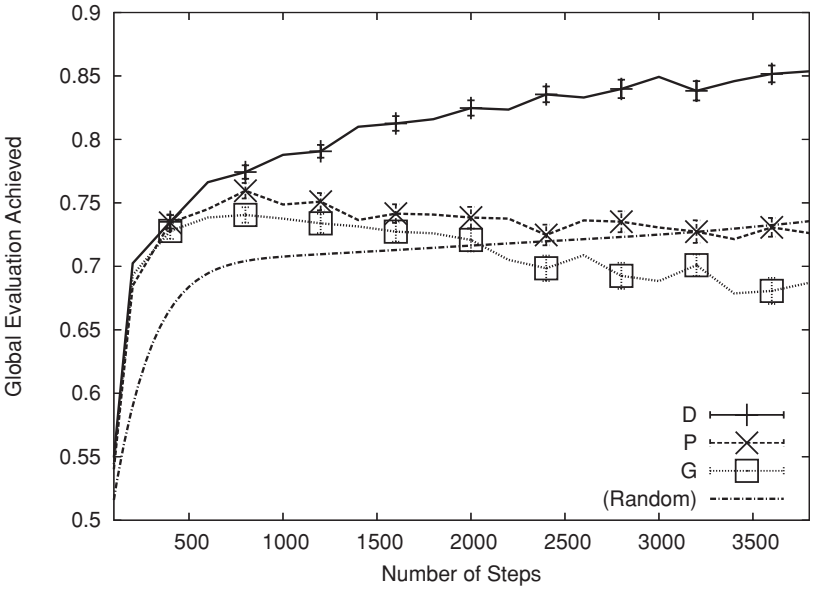

Figure 10: Results for noisy domain under communication limitations. Rovers can only see other rovers covering an area of 1% of the domain. Difference evaluation is superior since it is both factored and rover-sensitive.

local information with minor modifications, such as limiting the radius of observing other rovers in the system. This has the net effect of reducing the factoredness of the evaluation function while increasing its sensitivity. In this set of experiments we tested the performance of the three evaluation functions in a dynamic environment where not only were the rover sensors noisy, but the rovers were subject to two types of communication limitations.

5.4.1 Restrictions on Rover-to-Rover Communications

Figure 10 shows the performance of all three evaluation functions in a 30 rover system when the rovers are only aware of other rovers when they are within a radius of 4 units from their current location. This amounts to the rovers being able to communicate with only 1% of the grid. (Because P is not affected by communication restrictions, its performance is the same as that of figure 8.)

The performance of D is almost identical to that of full communication D. G on the other hand suffers significantly. The most important observation is that communication limited G is no longer factored with respect to the global evaluation function (e.g., as rovers do better with respect to communication limited G, the system performs worse). Though the sensitivity of G goes up in this case, the drop in factoredness is more significant and as a consequence, systems evolved using G cannot handle the limited communication domain (we analyze the connection between sensitivity and factoredness in more detail in Section 6).

Figure 11 expands on this issue by showing the dependence of all three evaluation functions on the communication radius for the rovers (P is flat since rovers using P ignore all other rovers). Using D provides better performance across the board and the

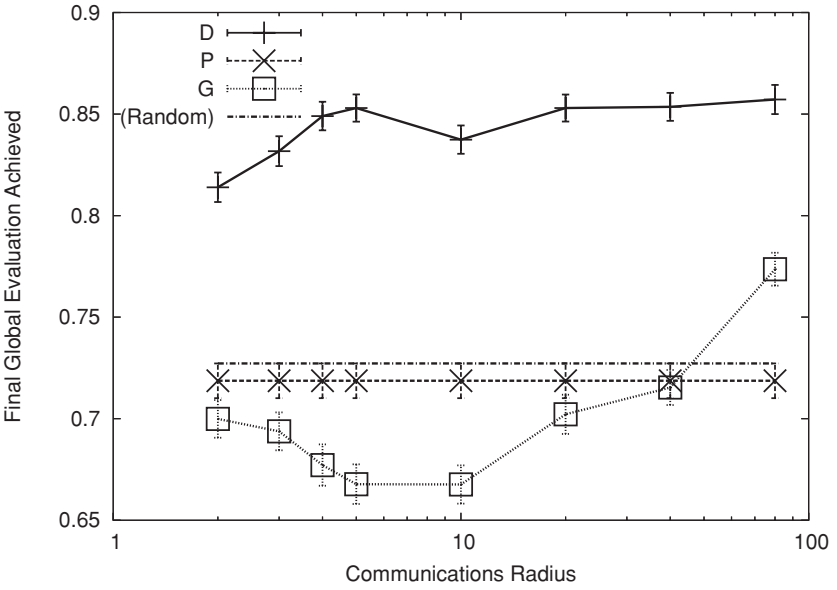

Figure 11: Sensitivity of the three evaluation functions to the degree of communication limitations. Difference evaluation is not affected by communication limitations as much as by global evaluation.

performance of D does not degrade until the communication radius is dropped to 5 units. This is a severe restriction that practically cuts off the rover from other rovers in the system. G on the other hand needs a rather large communication radius (over 40) to outperform the systems evolved using P. This result is significant in that it shows that D can be effectively used in many practical information-poor domains where neither G nor "full" D as given in Equation 3 can be accurately computed.

An interesting phenomenon appears in the results presented in figure 11, where there is a dip in the performance of the system when the communication radius is at 10 units for both D and G (the "bowl" is wider for G than D, but it is the same effect). This phenomenon is caused by the interaction between the degree of factoredness of the evaluation functions and their rover-specificity. At the maximum communication radius (no limitations) D is highly factored and has high sensitivity. Reducing the communication radius starts to reduce the factoredness, while increasing the sensitivity. However, the rate at which these two properties change is not identical. At a communication radius of 10, the drop in factoredness has outpaced the gains in sensitivity and the performance of the system suffers. When the communication radius drops to 5, the increase in sensitivity compensates for the drop in factoredness. This interaction between the sensitivity and factoredness is domain dependent (which has also been observed in other domains, e.g., Tumer and Agogino, 2004; Tumer and Wolpert, 2004b) and is discussed in more detail in Section 6.

5.4.2 Restrictions on POI Sensing and Rover-to-Rover Communications

In addition to exploring the effect of communication limitations between rovers, we also explore the effect of communication limitations (rover to rover) coupled with sensing

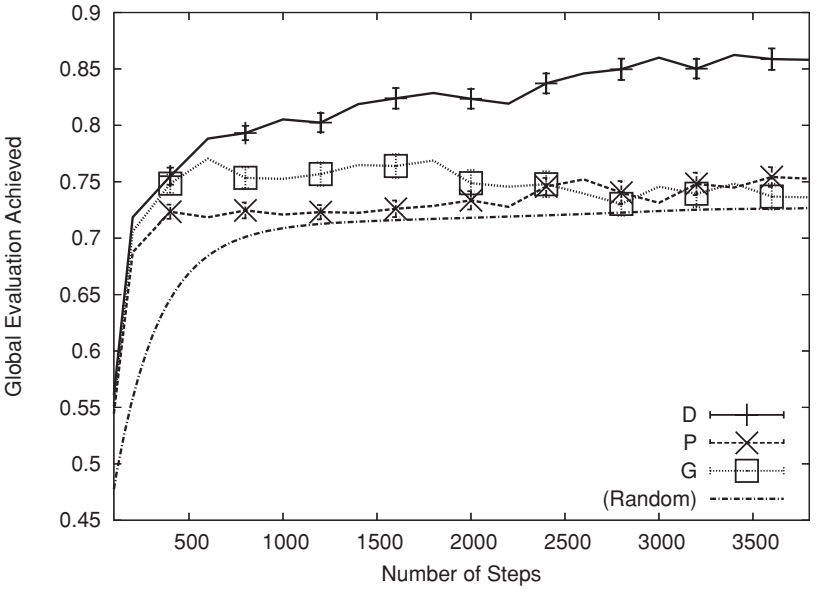

Figure 12: Results for noisy domain under increased communication limitations. Rovers can only see other rovers or POIs covering an area of 1% of the domain. Difference evaluation is superior since it is both factored and rover-sensitive.

limitations (POI detection). In these experiments, the evaluation functions only use information within a fixed radius from the rover. The results in figure 12 show that this additional limitation does not significantly affect the performance of rovers using the D evaluation. However, the decrease in information available actually increases the performance of rovers using both the P and G evaluation functions. This is not surprising in the case of G since the increase in sensitivity caused by the communication limitation is shown to allow for an increase in performance as previously shown in figure 11. In contrast, the P evaluation already has infinite sensitivity. Instead, this decrease in sensing causes the evaluation to become more factored. Figure 13 explores this issue further. The results are qualitatively similar to those in the previous section (except for P's dependence on sensing radius), though it is worth noting that in cases with severe communication restrictions, not detecting POIs proves beneficial. This shows that if the rovers are not aware of where the other rovers are, too much information hinders them more than it helps them.

5.5 Rovers Evolving Shared Population

In this paper we are primarily concerned with distributed systems where each rover is autonomous, each with its own population of control policies. However, it is illustrative to compare the distributed approach to a more conventional approach where there is a single population of control policies. This single shared population may be maintained in a central location, or it could be distributed among rovers, though distributing shared control policies could lead to significant communication overhead when the size of the control policies is large. To analyze this population sharing approach, we perform

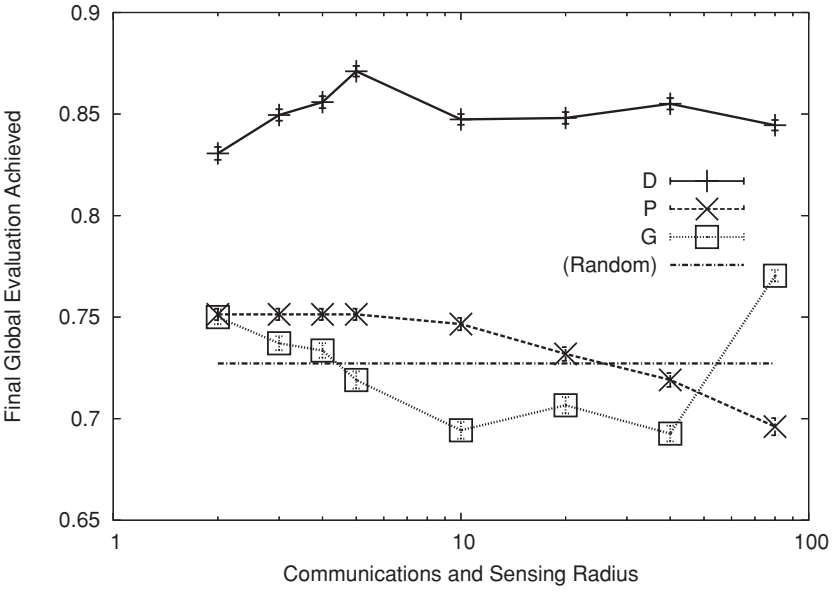

Figure 13: Sensitivity of the three evaluation functions to the degree of communication limitations. Difference evaluation is not affected by communication limitations as much as by global evaluation.

experiments where all thirty rovers share a single population of control policies. At the beginning of a trial each rover selects a control policy from the population using an epsilon-greedy selector as before. At $t = \tau$ (and subsequently at intervals of τ) the shared population is updated with a control policy used by a single rover chosen arbitrarily. Because different rovers are selected at different τ steps, all rovers' "experiences" are part of the population. Note that for relatively small values of epsilon, most of the rovers use the same control policy during the trial.

In this experiment we test the performance of a shared population using the global evaluation against the performance of individual populations using the global evaluation and the difference evaluation. Note that the difference evaluation cannot be used with the shared population, since it is specific to a particular rover. Figure 14 shows that when the G evaluation function is used, rovers that share a single population initially perform worse, but then reach a similar performance level as rovers that have individual populations. The performance of the rovers using the shared population is initially low since rovers using the same control policies scatter themselves less uniformly across the domain. However, these rovers are able to learn to improve this low level of performance relatively quickly. This is not a surprising result since when using the global evaluation, having a shared population reduces many of the signal-to-noise issues. Since at any given time most of the rovers use the same control policy, the impact of that control policy on the global evaluation is very large. Also there are no specialized rovers since all the rovers can perform the same task and are interchangeable. This means that different controllers are not strictly needed for different rovers (although the domain does have emergent specialization, where initially identical rovers tend to specialize what they do over time). However, despite this advantage, rovers using a

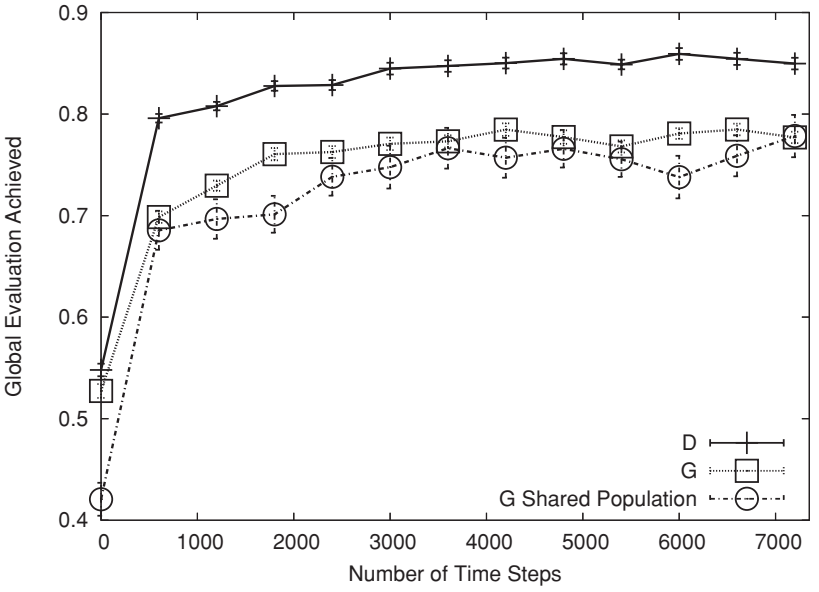

Figure 14: Performance of the 30-rover system with shared and unshared controller populations. When G evaluation is used, in this domain it is beneficial for all of the rovers to share a single population of controllers. However, rovers using the difference evaluation with individual control populations still perform best.

shared population with the G evaluation perform consistently worse than rovers using individual populations with the D evaluation. In domains where more specialized actions are needed we expect the performance difference to be even greater.

5.6 Surveillance Domain

In the previous experiments, moving rovers observed objects that did not themselves move, but where the distribution of the POIs changed slowly. However, in many surveillance domains, we need to observe objects that are moving. In these domains, a rover has to decide whether to follow a moving object, or whether that object will be better observed by another rover along the object's path.

5.7 Slow Moving POIs

To analyze how the evaluation functions handle a surveillance domain, we conduct experiments where rovers have to observe POIs that are continuously moving. In these experiments a POI changes its speed and direction every 10τ steps (150 time steps for these experiments). This is accomplished by setting the POI's x and y velocity to new random values uniformly within the range $[-d_p, d_p]$, where d_p is the maximum possible component velocity of the POI. Moving POIs are not allowed to move beyond the range of the domain where the rovers are allowed to move.

Figure 15 shows the performance of the rovers when the maximum speed the POIs can move is half the speed that the rovers can move. When the POIs are moving slowly,

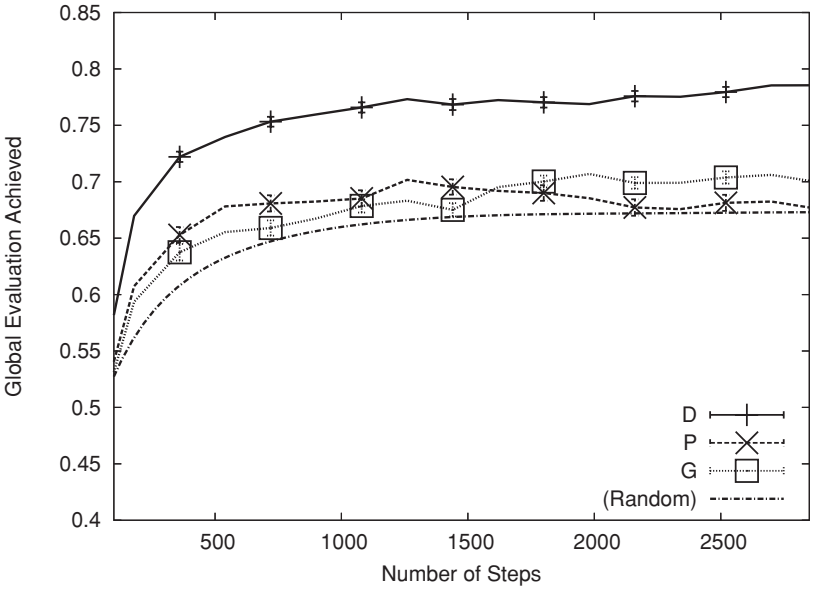

Figure 15: Results for surveillance domain. POIs move at half the rovers' maximum speed. Difference evaluation performs significantly better.

the relative results are similar to the domain where the POIs are fixed. Again rovers using D evolve the quickest, and end up with a significantly higher level of performance than the rovers using the other evaluation functions. Rovers using the G and P only perform slightly better than rovers using a random evaluation.

5.8 Fast Moving POIs

Another interesting experiment is to analyze how the evaluation functions handle POIs that move faster than the rovers. When the POIs can move at twice the maximum speed of the rovers, the rovers using P perform almost as well as the rovers using D (figure 16). When the POIs move quickly, the greedy utility, P, does well since the best strategy tends to be the short term strategy of moving toward the nearest POI. Any long term strategy tends to be defeated, since the dynamics are changing too quickly. In addition the congestion problem of all the rovers going toward the most valuable POI is not likely when the POIs move more quickly than the rovers. Instead the rovers and the POIs tend to be distributed randomly despite any different efforts by the rovers. Here only short term strategies matter. Note however that rovers using G are not even able to evolve good short term strategies and perform significantly worse than rovers using P and D.

6 Analysis of Factoredness and Sensitivity of Evaluation Functions

The results presented above underscore the need for designing evaluation functions that have both high factoredness and high sensitivity. However, in certain domains/ environments, it is not possible to satisfy both requirements (results in figures 10–13).

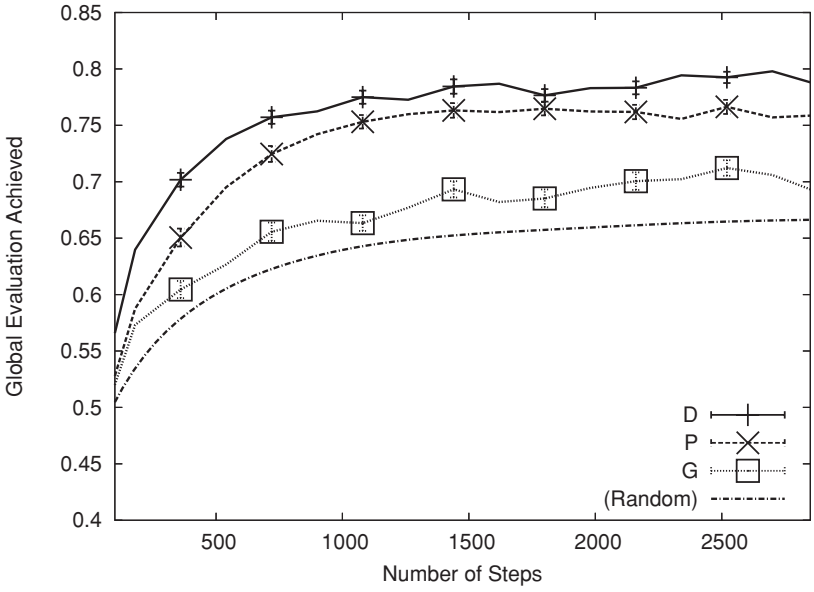

Figure 16: Results for surveillance domain. POIs move at twice the rovers' maximum speed. With fast POIs, the greedy response is adequate, allowing rovers using P evaluation to perform well.

In such cases, it is beneficial to explicitly compute the factoredness and sensitivity of an evaluation function and determine what trade-offs are needed.[4]

In this section we show how to compute the sensitivity and factoredness of evaluation functions using Monte Carlo methods. The degree of factoredness of an evaluation function as defined in Equation 1 shows how well an evaluation function is aligned with the system evaluation function (Agogino and Tumer, 2005). This value can range between one, for fully factored systems, and zero, for anti-factored systems. For an evaluation function, having a factoredness of one (e.g., difference evaluation) means that the evaluation function is always aligned with the global evaluation, that is, any change to a rover's action that causes its evaluation to go up (or down) causes the global evaluation to go up (or down). Having high factoredness means that the evaluation function is aligned with the global evaluation for a large percentage of actions. Finally, having a factoredness of 0.5 means that an action that improves a rover's evaluation function is as likely to improve the global evaluation as it is to reduce it (50-50 odds), which means the evaluation function is essentially irrelevant to the domain.

We now estimate factoredness for an evaluation function by taking the action, $z_{i,t}$ for a rover i at a random time t and replacing it with a random action $z'_{i,t}$. If n samples are taken, the factoredness for evaluation function g is estimated as:

$$\frac{1}{n_d}\sum_{j=1}^{n} I_{sgn(g(z)-g(z-z_i+r^j_{i,t}))=sgn(G(z)-G(z-z_{i,t}+r^j_{i,t}))} \tag{10}$$

[4]Visualization of these functions over the domain state-space was used to analyze reinforcement learning rewards in Agogino and Tumer (2005).

Table 1: Factoredness and Sensitivity of Evaluation Functions

Reward	Factoredness	Sensitivity
D	1.0	0.94
G	1.0	0.11
P	0.64	∞

where I is an indicator operator, $r_{i,t}^{j}$ is the jth random action for rover i, and n_d is the number of times the random action caused a change in G. This equation measures for how many of the n samples did the change in value of the evaluation function of interest align with the change in value of the global evaluation.

Along with factoredness, we are interested in the sensitivity of an evaluation function as defined previously in Equation 2: how sensitive the evaluation function is to the actions of the rover compared to the actions of all the other rovers. All other things being equal, having high sensitivity means a rover can more easily see the effects of its actions on its evaluation function and thus will be able to evolve good policies more quickly. As with factoredness, sensitivity can be approximated through sampling as:

$$\frac{1}{n}\sum_{j=1}^{n}\frac{\left\|g(z)-g\left(z-z_{i,t}+r_i^j\right)\right\|}{\left\|g(z)-g\left(z-z_{-i,t}+r_{-i,t}\right)\right\|} \tag{11}$$

where $z_{-i,t}$ equals $z_t - z_{i,t}$ and $r_{-i,t}$ is a random set of actions for all rovers other than i.

The factoredness and sensitivity were computed for the D, G, and P evaluation functions. The results are given in table 1. These estimates were obtained with a total of 90,000 samples for factoredness (1,500 time steps $\times$ 30 agents $\times$ 2 samples) and 135,000 for sensitivity (1,500 $\times$ 30 $\times$ 3).

As expected, the D evaluation function is fully factored: any action a rover takes to improve D also improves G. However, notice that the D evaluation function is more sensitive to the rover's actions. While the P evaluation has infinite sensitivity (since it is only a function of one rover), it has low factoredness. Due to the perfect sensitivity, rovers can maximize this evaluation more easily, but maximizing the P evaluation may not lead to the maximization of the system evaluation because of this low factoredness.

While D and G are fully factored when there are no communication restrictions, they are not necessarily fully factored when there are communication restrictions. Intuitively, in this case there is not enough information for a rover to compute D and G accurately enough so that they are fully aligned with the true global evaluation function. Table 2 shows the factoredness and sensitivity calculations under communication limitations (for evaluation functions used in figures 10–11). The radius in which a rover can see other rovers and POIs and include their contribution in its computation of its evaluation is given in the second column. D has superior factoredness properties even when a rover cannot communicate with most of the world. This is a striking result and explains the reasons for the performance of D in all the experiments. The factoredness of P is barely above 0.5, meaning it is mostly useless for this domain. As for G, only when communication becomes quite global does its factoredness reach good values.

The sensitivity of D is always near one, though for a small communication radius, G has a sensitivity larger than one. Intuitively this means that the G evaluation with a

Table 2: Factoredness and Sensitivity of Evaluation Functions with Limited Communication and Sensing Capabilities

Reward	Communication radius	Factoredness	Sensitivity
D	10	0.79	1.0
G	10	0.66	3.4
P	10	0.56	∞
D	20	0.93	0.92
G	20	0.64	1.4
P	20	0.54	∞
D	50	1.0	0.95
G	50	0.66	0.74
P	50	0.55	∞

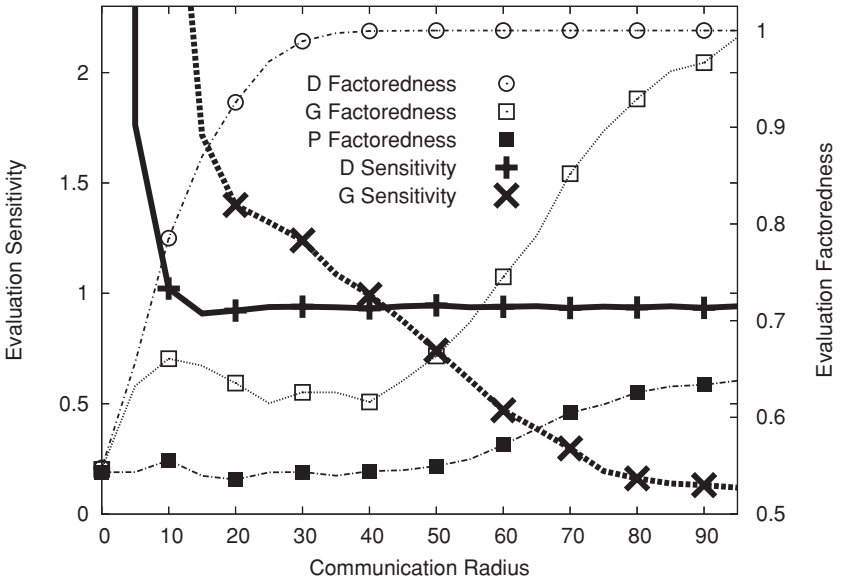

Figure 17: Factoredness and sensitivity of evaluation functions under communication restrictions.

communication radius of 10 depends more on a particular rover's actions while the D evaluation under similar restrictions depends as much on a particular rover's actions as on the actions of other rovers. The sensitivity of D is steady across a large range of communication settings. The sensitivity of G on the other hand drops nearly linearly with increasing communication radius. The conclusion we draw from this experiment is that while low sensitivity hurts the system, it is the interaction between factoredness and sensitivity that is critical to good performance. Having high values of sensitivity that are not accompanied by high values of factoredness (e.g., small communication radius for G) does not provide good system behavior.

Figure 17 expands on these results and shows the factoredness and sensitivity of the evaluation functions under a communication radius ranging from 0 (rovers cannot see any other rovers or POIs) to 100 (rovers can see everything). This figure illustrates that the difference evaluation has both good sensitivity and factoredness across a wide

range of communication limitations. In contrast, the communication restricted global evaluation declines in factoredness dramatically as the communication radius shrinks, explaining its poor performance in Section 5.4.2. The communication limited evaluation P maintains a poor level of factoredness for all communication radiuses. Note that as the communication radius approaches zero, all the evaluations approach infinite sensitivity, but this is counterbalanced by their factoredness being low.

7 Discussion

Extending the success of evolutionary algorithms in continuous single-controller domains to large, distributed multi-controller domains has been a challenging endeavor. Unfortunately the direct approach of having a population of entire systems and applying the evolutionary algorithm to that population results in a prohibitively large search space in most cases. As an alternative, this paper presents a method for providing local evaluation functions to directly evolve individual components in the system. The fundamental research question addressed in this work is the method by which those individual evaluation functions are selected so that the components evolved to optimize those evaluation functions display coordinated behavior.

The solution to this problem is to ensure that the individual evaluation functions satisfy two important system level properties, alignment and sensitivity. In a wide range of dynamic environments, in the presence or absence of noise in the rover sensors/actuators and in the presence or absence of communication restrictions, systems evolved using the difference evaluation function D, provided good solutions to the distributed control problem of rover coordination. Because the D evaluation function provided evaluations that were both factored and highly rover-sensitive, rovers evolved using D significantly outperformed rovers using P (nonfactored, perfectly rover sensitive) and G (fully factored, low sensitivity). This result also supports the intuition expressed in Section 2 that approach 3(a) (i.e., evolving rovers based on the fitness of the full system) is ill-suited to evolving effective control policies in all but the smallest systems. In addition, results highlight that approach 3(b) only works when the rover evaluation functions are selected in a principled manner.

An important observation is that in all experiments the gains due to D increased as the system became more realistic/complex (e.g., increased size, reduced observation). This is a powerful result suggesting that D is well suited to evolve large and communication limited systems in this and similar domains where the interaction among the components restricts the applicability of either G or P or both. In summary, this paper demonstrated the promise of using fitness evaluations with high factoredness and sensitivity to allow evolutionary computation methods to successfully evolve control policies for large distributed systems. The results show that coordination can evolve as a by-product of selecting the individual evaluation functions.

While this paper has shown how evolutionary methods using the difference evaluation function can provide robust results in difficult domains, there are still a number of challenging questions. One of those challenges is ensuring system robustness, or determining how to evolve a large complex system so that it operates when a certain number of components break down. This is crucial for applying this method to large complex systems operating in unknown and dangerous environments. The second challenge is transferability, or determining how a system can be first evolved in one environment and then continue to evolve new policies when deployed in a new environment. This is essential for systems that will operate for long periods of time and where the conditions

may change during the operation of the system. The results shown in this paper give promise that both those challenges can be met by evolving individual system components using the difference evaluation function, and we are currently investigating both lines of research.

Acknowledgments

The authors would like to thank the reviewers for their detailed and constructive comments that have significantly improved this paper.

References

Agah, A., and Bekey, G. A. (1996). A genetic algorithm-based controller for decentralized multi-agent robotic systems. *Proceedings of the IEEE International Conference of Evolutionary Computing* (pp. 431–436).

Agogino, A., Stanley, K., and Miikkulainen, R. (2000). Online interactive neuro-evolution. *Neural Processing Letters*, 11:29–38.

Agogino, A., and Tumer, K. (2004a). Efficient evaluation functions for multi-rover systems. *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 1–12).

Agogino, A., and Tumer, K. (2004b). Unifying temporal and structural credit assignment problems. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems* (pp. 980–987).

Agogino, A., and Tumer, K. (2005). Multi agent reward analysis for learning in noisy domains. *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems* (pp. 81–88).

Agogino, A., and Tumer, K. (2006). Distributed evaluation functions for fault tolerant multi rover systems. *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 1079–1086).

Agogino, A., Tumer, K., and Miikkulainen, R. (2005). Efficient credit assignment through evaluation function decomposition. *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 1309–1316).

Dorigo, M., and Stützle, T. (2004). *Ant Colony Optimization*. Bradford Books, Denver.

Farritor, S., and Dubowsky, S. (2002). Planning methodology for planetary robotic exploration. *ASME Journal of Dynamic Systems, Measurement and Control*, volume 124 (pp. 698–701).

Floreano, D., and Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. *Proceedings of the Third International Conference on Simulation of Adaptive Behavior* (pp. 421–430). MIT Press, Cambridge, MA.

Globus, A., Crawford, J., Lohn, J., and Pryor, A. (2003). Scheduling earth observing satellites with evolutionary algorithms. *Proceedings of International Conference on Space Mission Challenges for Information Technology (SMC-IT)* (pp 836–843).

Gomez, F., and Miikkulainen, R. (1999). Solving non-Markovian control tasks with neuroevolution. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)* (pp. 1356–1361).

Gomez, F., and Miikkulainen, R. (2003). Active guidance for a finless rocket through neuroevolution. *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 2084–2095).

Hardin, G. (1968). The tragedy of the commons. *Science*, 162:1243–1248.

Harrald, P., and Fogel, D. (1996). Evolving continuous behaviors in the iterated prisoner's dilemma. *BioSystems: Special Issue on the Prisoner's Dilemma*, 37(1–2):135–145.

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. Macmillan, New York.

Hoen, P. J., and de Jong, E. (2004). Evolutionary multi-agent systems. *Proceedings of the Eighth International Conference on Parallel Problem Solving from Nature* (pp. 872–881).

Hoffmann, F., Koo, T.-J., and Shakernia, O. (1999). Evolutionary design of a helicopter autopilot. *Advances in Soft Computing—Engineering Design and Manufacturing, Part 3: Intelligent Control* (pp. 201–214).

Lamma, E., Riguzzi, F., and Pereira, L. M. (2001). Belief revision by multi-agent genetic search. *Proceedings of the Second International Workshop on Computational Logic for Multi-Agent Systems*.

Martinoli, A., Ijspeert, A. J., and Mondala, F. (1999). Understanding collective aggregation mechanisms: From probabilistic modeling to experiments with real robots. *Robotics and Autonomous Systems*, 29:51–63.

Mataric, M. J. (1998). Coordination and learning in multi-robot systems. *IEEE Intelligent Systems* (pp. 6–8).

Miglino, O., Lund, H. H., and Nolfi, S. (1995). Evolving mobile robots in simulated and real environments. *Artificial Life*, 2(4):417–434.

Mikami, S., Mitsuo, W., and Kakazu, Y. (1996). Combining reinforcement learning with GA to find co-ordinated control rules for multi-agent system. *Proceedings of the IEEE International Conference on Evolutionary Computation* (pp. 356–361).

Mundhe, M., and Sen, S. (2000). Evolving agent societies that avoid social dilemmas. *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 809–816).

Panait, L., Luke, S., and Wiegand, R. P. (2006). Biasing coevolutionary search for optimal multiagent behaviors. *IEEE Transactions on Evolutionary Computation*, 10:629–645.

Potter, M. A., and de Jong, K. A. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29.

Sen, S., Debnath, S., and Mundhe, M. (2003). Evolving coordinated agents. *Advances in evolutionary computing: Theory and applications* (pp. 559–577).

Stanley, K., and Miikkulainen, R. (2002). Efficient reinforcement learning through evolving neural network topologies. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)* (pp. 569–577).

Tumer, K., and Agogino, A. (2004). Overcoming communication restrictions in collectives. *Proceedings of the International Joint Conference on Neural Networks*, volume 2 (pp. 1127–1132).

Tumer, K., and Agogino, A. (2006a). Robust coordination of a large set of simple rovers. *Proceedings of the IEEE Aerospace Conference*.

Tumer, K., and Agogino, A. K. (2006b). Efficient reward functions for adaptive multi-rover systems. In K. Tuyls, P. J. Hoen, S. Sen, and K. Verbeeck (Eds.), *Learning and Adaptation in Multi Agent Systems* (pp. 177–191). Springer, Berlin.

Tumer, K., and Agogino, A. K. (2007). Coordinating multi-rover systems: Evaluation functions for dynamic and noisy environments. In S. Yang, (Ed.), *Evolutionary Computation in Dynamic and Uncertain Environments* (pp. 371–388). Springer, Berlin.

Tumer, K., and Wolpert, D. (Eds.). (2004a). *Collectives and the Design of Complex Systems*. Springer, Berlin.

Tumer, K., and Wolpert, D. (2004b). A survey of collectives. *Collectives and the Design of Complex Systems* (pp. 1–42). Springer, Berlin.

Tumer, K., and Wolpert, D. H. (2000). Collective intelligence and Braess' paradox. *Proceedings of the Seventeenth National Conference on Artificial Intelligence* (pp. 104–109).

Whitley, D., Gruau, F., and Pyeatt, L. (1995). Cellular encoding applied to neurocontrol. *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 460–469).

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

Wu, A., Schultz, A., and Agah, A. (1999). Evolving control for distributed micro air vehicles. *IEEE International Symposium on Computational Intelligence in Robotics and Automation* (pp. 174–179).

Yao, X., and Darwen, P. (1994). An experimental study of n-person iterated prisoner's dilemma games. *Informatica*, 18:435–450.