Environmental Robustness in Multi-agent Teams

Terence Soule
University of Idaho
Moscow, ID
83844
toule@cs.uidaho.edu

Robert B. Heckendorn
University of Idaho
Moscow, ID
83844
heckendo@uidaho.edu

ABSTRACT
Evolution has proven to be an effective method of training heterogeneous multi-agent teams of autonomous agents to explore unknown environments. Autonomous, heterogeneous agents are able to go places where humans are unable to go and perform tasks that would be otherwise dangerous or impossible to complete. However, a serious problem for practical applications of multi-agent teams is how to move from training environments to real-world environments. In particular, if the training environment cannot be made identical to the real-world environment how much will performance suffer? In this research we investigate how differences in training and testing environments affect performance. We find that while in general performance degrades from training to testing, for difficult training environments performance improves in the test environment. Further, we find distinct differences between the performance of different training algorithms with Orthogonal Evolution of Teams (OET) producing the best overall performance.

Categories and Subject Descriptors
I.2.9 [Artificial Intelligence]: Robotics - Autonomous vehicles; I.2.2 [Artificial Intelligence]: Automatic Programming

General Terms
Algorithms, Design

Keywords
genetic programming, multi-agent systems, teams, cooperation, OET

1. INTRODUCTION
Teams of millions of nearly disposable robots will have a revolutionary effect on a host of human endeavors, such as clearing landmines, search and rescue in large natural disasters, environmental cleanup, mining and resource discovery, aircraft debris recovery, and even individual guides in urban disaster evacuation and assessment. While the hardware for this paradigm shift is rapidly becoming a reality, there remain several major software hurdles to be overcome.

Various evolutionary algorithms, and especially genetic programming [12], have been successfully applied to many problems in the area of multi-agent systems. Previous approaches include teams composed of neural networks (NNs) [13], oblique decision trees [2], stack based predictors [17], and systems of induced functions [20]. Genetic programming has also been successfully applied to a wide range of multi-agent problem domains including foraging [3], robot navigation [7], sporting strategies [18], predator/prey strategies [5, 14], forest fire detection [6], and mine detection [19]. In their 2005 review of cooperative and multi-agent learning, Panait and Luke conclude that evolutionary computation (as compared to reinforcement learning) is well suited to team learning [16]. Thus, there is good reason to approach the problem of multi-agent systems using genetic programming. However, there remains a number of distinct issues that must be addressed before genetic programming can be reliably used as a technique for evolving large scale multi-agent systems to act in the real world.

An issue of particular importance for the machine learning community in general, and for evolutionary techniques in particular, is the difficulty in moving from a training environment to the real world application environment. For most applications it is unlikely that these environments will be identical and, given the extensive time requirements of evolutionary learning, it may even be the case that agents must be trained primarily, or even entirely, in a simulated environment that is significantly different from the true application environment. Thus, it is critical to understand the relationship between training and real world environments, to determine if particular features of the training environment make the transition to other environments easier, and to develop algorithms that produce multi-agent teams that can make this transition smoothly.

In this paper we begin to address these questions, albeit still with fairly small (6 member) teams. We test two common and one fairly novel multi-agent team training algorithms and compare their relative performance on a variety of training and testing environments. The results show, perhaps not surprisingly, that in general performance degrades from training to testing environments. This degradation in performance is greater when there is a greater qualitative difference between the training and testing environments. More surprisingly this degradation is never catastrophic, the
worst decrease we observed for the best teams was 11.6% when going from the training to the testing environment. We also found that for some difficult training environments performance actually improves in the test environment, suggesting that increasing the difficulty of the training problem may be beneficial, even if it decreases training performance. Further, we find distinct differences between the performance of the different multi-agent training algorithms in terms of how well the evolved teams make the transition to the test environment.

2. THE PROBLEM

Our problem is modeled on search and investigation problems. The environment is a two dimensional grid (45x45) containing some percentage of “interesting” cells distributed in different ways. The differences in distributions are the different environments.

There are two agent types: scouts and investigators. The scouts’ role is to find interesting squares and mark them with a beacon that is detectable at a distance by the investigators. The investigators’ role is to investigate interesting squares and mark them as investigated. Scouts travel at up to twice the speed of investigators. Scouts automatically place beacons in any interesting square they are in or adjacent to (unless it already contains a beacon). If an investigator enters an interesting square, with or without a beacon, it changes the square to investigated and deactivates any beacons in the square. It is important to reinforce here that the investigators work at half the speed of the scouts, but can see the beacons at a distance. Therefore the space can be more efficiently explored by fast scouts marking interesting areas and investigators using the beacons to go directly to the areas to be investigated.

This model represents an abstraction of a number of practical problems. For example, scouts and investigators could represent two robot types exploring a minefield. Scouts fly overhead marking locations of potential mines and investigators deactivate the mines. Alternatively, they could represent an automated planetary surveying team. Scouts identify potentially interesting geological formations and investigators follow up by taking soil samples, etc.

A difficulty with learning approaches to this problem is that the specific environment a team must operate in may not be known a priori or may change between “missions”. In particular, the frequency and distribution of interesting cells is likely to both vary and have a significant effect on the optimal search strategy.

Thus, for these experiments we have developed three different environments, where environment is defined by the distribution of interesting cells. In the first environment, which we call random, each cell has a uniform 20% chance of being interesting. Figure 1 shows an example of a random environment. In the second environment, which we call clumped, exactly 20% of the cells are interesting and they are clumped, stochastically, in the four corners of the space. Figure 2 shows an example of a clumped environment. This environment corresponds to a number of real world problems. For example, if a team of robotic agents must identify the debris from an airplane crash, the debris will be found in clumps around the impact sites of the larger pieces of the plane. In the third environment, which we call linear, exactly 10% of the cells are interesting and they are distributed randomly along eleven rows in the environment - the same eleven rows are always used, but the exact placement of interesting cells within the rows is random. Figure 3 shows an example of a linear environment. This environment was chosen primarily to be significantly and qualitatively different from the other two. However, it also corresponds to a number of real world applications, including agricultural harvesting, where crops are planted in rows, and detection of undersea mines, which are typically laid behind a moving ship producing linear patterns.

For all of the environments a new random case is generated for each evaluation, so individuals cannot memorize the locations of interesting cells, but must evolve strategies that are suited to the particular environment, i.e. the particular distribution pattern of the interesting cells.

We are interested in a number of questions:

1. How specialized are the search strategies that evolve to their particular training environment? E.g. will a team well suited to one environment perform poorly in another.

2. Is there an optimal training environment that evolves teams that are more robust with respect to their environment and therefore transition to other environments with a smaller change in performance?

3. Is there an optimal training algorithm that evolves teams that are robust with respect to their environment?

2.1 The vector model for agent movement

The agents treat the environment as a two dimensional real valued space. Agent movement is determined by a vector expression, represented by an expression tree, that calculates their next move. This expression tree calculates and returns a vector, based on current input vectors, and the agent moves in that direction. Investigators are limited to moves of length one and scouts are limited to moves of length two. The expression tree is a representation of the solution.
space in the form of a program for each agent to determine its behavior. The objective of the evolutionary algorithms is to evolve a good expression tree for each team member.

Input vectors (terminal nodes in an agent’s evaluation tree):

- North - a unit vector pointing North, 0 radians.
- Constant - a vector that is initially generated randomly at the time an agent’s program is initialized. The constant remains so throughout the lifetime of an agent but can be overwritten during mutation for instance.
- Random - a vector that is randomized at each time step.
- Nearest scout - a vector to the nearest scout.
- Nearest investigator - a vector to the nearest investigator.
- Nearest beacon - a vector to the nearest beacon.
- Last move - a vector representing the agent’s last move.
- Nearest edge - a vector to the nearest boundary of the search space.

In the current implementation there is no limit on an agent’s vision e.g., an agent can “see” the nearest beacon regardless of its distance. If an input is meaningless, e.g. nearest beacon when no beacons are present, the zero vector (direction = 0, magnitude = 0) is returned.

Vector operations (non-terminal nodes in an agent’s evaluation tree):

- Invert - takes a single vector argument and inverts it (rotates π radians).
- Sum - takes two vector arguments, returns the vector sum.

Clearly the choice of inputs and operations has a significant influence on how the agents can evolve. We chose a fairly extensive set of input vectors. These vectors represent some distance sensors but the agents do not share information. Future research will look at the effect of changing this set: reducing the range of agent’s vision, allowing agents to sense each other by name or tag, etc.

3. FITNESS EVALUATION

A significant difficulty in many multi-agent systems is how to assign credit (fitness) to individual team members. In this research, we chose to begin with a simple, basically greedy, approach. Each evaluation of fitness involves a new random problem space of interesting cells. The agents begin at random location within the center cell and pointing North. They have a fixed amount of time to explore.

The fitnesses for the scouts and investigators are as follows:

\[
\text{fitness}_s = 3\beta - .1b \\
\text{fitness}_i = 3I - .1b
\]

Where \(\beta\) is the number of beacons placed, \(b\) is the number of time steps outside of the bounded problem area, \(I\) is the number of interesting areas investigated. Thus, scouts and investigators are rewarded for finding interesting squares and investigating them respectively and penalized for leaving the boundaries. Interestingly we have observed that if the boundary penalty is too large agents will evolve that simply sit still to avoid incurring a penalty. (Note: for the linear environment all reported fitness are doubled, this reflects that there are only half as many interesting cells present.)

The fitness of a team is the sum of the team members’ fitnesses. The fitness of a particular member or team will vary somewhat between evaluations because for each evaluation the environment is randomly generated.

4. TEAM TRAINING ALGORITHMS

We compare three team based training algorithm to determine both which algorithm produces the best results in the
training environment and which algorithm produces teams that are robust with respect to the environment. E.g. which 
algorithm produces teams who’s performance is least de-
graded in moving to a new environment.

There are two fundamental requirements to creating a suc-
 cessful multi-agent team: the individual agents must be rel-
avely successful and the agents must cooperate in a way 
that improves the performance of the whole team. Typically 
this means that the bots specialize to solve particular sub-
problems. Common EC approaches for multi-agent teams 
can be roughly divided into two categories: island (or con-
current [16]) approaches and team approaches. Thus, in this 
research we implement one team, one island, and one hybrid 
(Orthogonal Evolution of Teams or OET) approach.

All three algorithms are based on a steady-state GP. Selec-
tion, be it on teams or on team members, is always done via 
a three member tournament. A three member tournament is 
also used to select teams or individuals for replacement. In 
the case of ties the size of the teams or individuals is used as 
a tie breaker, which helps to limit the size of the programs. 
Other parameters are given in Table 1.

4.1 Team

In team approaches, a single population evolves, each in-
dividual in the population represents a team of $m$ individu-
als, and fitness and selection are based purely on the whole 
team’s performance [5, 14, 20, 1, 17]. Thus, in our imple-
mentation of the team algorithm all selection is done at the 
level of complete teams (3 scouts and 3 investigators). That 
is on the rows in Figure 4. The performance of individual 
members is not used.

In general, team approaches lead to Team members that 
cooperate well. However, research shows that even in suc-
cessful teams the individual team members can become “lazy”, 
letting others on the team cover for their poor performance 
and producing poor overall performance [21, 1].

4.2 Island

In general in island approaches, concurrent, independent, 
evolutionary processes are used to train specific members of 
the team [11, 4, 25, 16]. Thus, in our implementation of the 
 island algorithm all selection is done on individuals, regard-
less of the performance of their team. Members and their 
offspring always remain in the same column in the popula-
tion (see Figure 4). Thus, each column acts as an indepen-
dent island. (It would also be possible to allow individuals to 
move between columns, but research shows that this hurts 
specialization and degrades performance [14].)

In each iteration one cycle of selection, crossover, muta-
tion, and replacement is done per column. This, introduces 
the major drawback of the island approach - it requires many 
more evaluations per iteration. Consider an iteration of the 
island approach. In column 1 two individuals are selected as 
parents, they undergo crossover and mutation, then replace 
two individuals selected (via three member tournaments) for 
their low fitness, say individuals $I_{1,i}$ and $I_{1,j}$ (the $i^{th}$ and $j^{th}$ 
in column 1). Now to evaluate those individuals the teams 
in rows $i$ and $j$ must be evaluated, because even though se-
lection and reproduction applies to individuals, evaluation 
still requires testing an entire team (testing a single scout 
or investigator in isolation isn’t an accurate measure of its 
fitness). Then the same thing must be done in each of the 
succeeding columns. Thus, instead of evaluation 2 teams per 
iteration (as in the team and OET approaches), it’s neces-
sary to evaluate 12 teams per iteration: one evaluation for 
each of the two replaced offspring per column. Thus, to 
balance the number of evaluations the island algorithm is 
run for one sixth as many iterations as the team and OET 
approaches. (Team and OET require two evaluations per 
iteration, one for each offspring team, island requires 12.)

In general, island approaches are based on the assumption 
that each evolutionary process (e.g. each separate island) 
will produce agents well suited to a particular role, and that
when combined they will naturally cooperate to cover the entire problem domain. However, research has shown that the members generated tend to have significantly “overlapping” behaviors such that much of the problem domain remains unaddressed and overall performance of the team is sub-par [11, 9, 23].

Thus, the team approach produces teams with strong cooperation, but some underperforming members, whereas the island approach creates highly fit individuals which may cooperate poorly because their areas of expertise overlap unproductively. Despite these weaknesses both EC approaches to evolving teams have proven successful when compared to other forms of ensemble learning [8, 25, 10, 15].

4.3 Orthogonal Evolution of Teams

Orthogonal Evolution of Teams (OET) combines these two approaches by alternating whether the total population is treated as a islands (columns) or as teams (rows). A number of OET approaches are possible depending on whether the population is treated as rows or columns during selection and replacement [24]. In this paper we take one of the most straightforward approaches, during the selection step the population is treated as islands, that is we apply (3 member tournament) selection to each column creating a new team consisting of highly fit individuals. This is done twice to create to “superstar” teams. These teams undergo crossover, with crossover applied to individuals from the same column (e.g. scout 1 in parent 1 is crossed with scout 1 in parent 2; scout 2 in parent 1 is crossed with scout 2 in parent 2, etc.). This helps the individual members to evolve specialized roles in the population [14].

Then, during the replacement step the population is treated as teams. Two poorly fit teams are selected (again via 3 member tournament selection) and replaced by the two offspring teams. Thus, there is direct selective pressure on both teams and individuals. Individuals must perform well to be selected as parents; teams must perform well to avoid being replaced.

Previous research has confirmed that OET produces teams whose average members are better than the average member in a team algorithm, teams whose members cooperate better than in an island algorithm, and teams that perform better than teams generated via either the island or team algorithms [23]. Further preliminary results suggest that OET has several significant advantages when applied to multiagent systems. Behaviors learned via OET scale better to larger teams, allowing “training in the small” for in the large, than do behaviors learned via team or island algorithms [22]. Further, the behaviors learned via OET are more robust to changes in mission parameters and to changes in team composition than the behaviors learned via either team or island algorithms [24].

5. RESULTS

Each of the three training algorithms was tested by using each of the three environments as a training environment and the other two environments as test environments. This let us measure how performance changed between training and testing environments for each of the training algorithms.

Table 2 shows the average fitness of all individuals in all 25 trials in the final generation for each of the three algorithms (values in parentheses are standard deviations). The values in bold are for the cases where the training and test environments are the same. E.g. in the first row of data in Table 2 OET was used with the random environment for training and the average fitness was 1581.0, then those individuals were tested on the clumped and linear environments and had average fitnesses of 1546.2 and 900.4 respectively.

The next row shows the results with the OET algorithm when the training environment was clumped, etc. The final column shows the average performance across all three environments, one training and two testing.

Table 3 shows the percentage change (as a decimal) from the training to the testing environments. Tables 4 and 5 show the same data, but for the best individual from each population (averaged across the 25 trials).

Several results are clear from this data. First, all of the algorithms produce fairly robust behaviors. Among the best teams (Table 5) the largest decrease in performance is a fairly reasonable 11.6% for the island algorithm when its trained on the random environment and tested on the linear environment.

Second, training on either the random or clumped environment is good training for the other environment, but neither is as good of a training environment for the linear environment. This is particularly true of the average teams (Tables 2 and 3, but is also somewhat true of the best teams. Of the clumped and random environments, the clumped environment turns out to be slightly better for generating robust teams. The decrease in performance when tested on the other two environments is less than when the random environment is used for training.

Surprisingly, the linear environment turns out to be fairly good training environment. First, and unsurprisingly, the teams trained on the linear environment give the best results for the linear environment. More importantly the performance of teams trained in the linear environment improves significantly when tested on the random and clumped environments, so that the overall performance across all three environments is quite good.

Of the three algorithms OET generated significantly better teams. A student’s two-tailed t-test was used to compare the performance of the best teams generated via OET to the best teams generated via island (the next best algorithm) on all three training environments. In all three cases the OET algorithm performed significantly better ($p < 0.025, p < 0.01, and p < 0.001$ on the random, clumped,

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>random</th>
<th>clumped</th>
<th>linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>OET</td>
<td>random</td>
<td>0.043</td>
<td>-0.158</td>
<td>0.022</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team</td>
<td>random</td>
<td>0.136</td>
<td>0.093</td>
<td>0.411</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td>0.112</td>
<td>-0.004</td>
<td>0.405</td>
</tr>
<tr>
<td>Island</td>
<td>random</td>
<td>0.010</td>
<td>-0.249</td>
<td>0.422</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td>-0.265</td>
<td></td>
<td>-0.412</td>
</tr>
</tbody>
</table>

Table 3: Percent decrease in team performance (negative values reflect an increase) across all population members and all trials in the final generation as a function of the training and testing environments.
Table 2: Average team performance across all population members and all trials in the final generation as a function of the training and testing environments. Bold values are for teams trained and tested in the same environment. The results show that teams trained in one environment perform more poorly in other environments. The exception is teams trained in the linear environment, which sometimes perform better when tested in alternative environments.

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>random</th>
<th>clumped</th>
<th>linear</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OET</td>
<td>random</td>
<td>1581.0(±96.68)</td>
<td>1546.2(±111.9)</td>
<td>900.4(±130.8)</td>
<td>1342.6(±333.1)</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td>1578.8(±159.3)</td>
<td>1649.2(±139.1)</td>
<td>993.33(±177.6)</td>
<td>1407.3(±334.8)</td>
</tr>
<tr>
<td></td>
<td>linear</td>
<td>1464.7(±145.1)</td>
<td>1483.1(±153.9)</td>
<td>1265.7(±132.9)</td>
<td>1404.5(±174.6)</td>
</tr>
<tr>
<td>Team</td>
<td>random</td>
<td>1382.3(±136.8)</td>
<td>1253.2(±137.0)</td>
<td>814.64(±90.7)</td>
<td>1149.8(±290.5)</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td>1249.9(±196.7)</td>
<td>1445.9(±189.6)</td>
<td>860.66(±167.2)</td>
<td>1185.3(±305.4)</td>
</tr>
<tr>
<td></td>
<td>linear</td>
<td>1054.0(±150.9)</td>
<td>1086.6(±151.9)</td>
<td>1188.3(±141.6)</td>
<td>1110.4(±158.7)</td>
</tr>
<tr>
<td>Island</td>
<td>random</td>
<td>1418.1(±148.17)</td>
<td>1411.9(±141.6)</td>
<td>819.9(±186.6)</td>
<td>1216.6(±323.0)</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td>1410.7(±146.0)</td>
<td>1425.2(±129.33)</td>
<td>838.01(±132.6)</td>
<td>1224.7(±305.4)</td>
</tr>
<tr>
<td></td>
<td>linear</td>
<td>1246.6(±102.8)</td>
<td>1262.3(±107.1)</td>
<td>997.7(±91.67)</td>
<td>1168.9(±157.6)</td>
</tr>
</tbody>
</table>

Table 4: Performance of the best member in the final generation averaged across all trials as a function of the training and testing environments. Bold values are for teams trained and tested in the same environment.

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>random</th>
<th>clumped</th>
<th>linear</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OET</td>
<td>random</td>
<td>2354.6(±82.44)</td>
<td>2270.1(±39.09)</td>
<td>2112.9(±66.92)</td>
<td>2245.8(±119.6)</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td>2338.1(±76.58)</td>
<td>2255.6(±48.54)</td>
<td>2128.7(±101.2)</td>
<td>2240.8(±116.5)</td>
</tr>
<tr>
<td></td>
<td>linear</td>
<td>2291.5(±86.28)</td>
<td>2241.9(±47.86)</td>
<td>2137.8(±82.94)</td>
<td>2223.7(±98.2)</td>
</tr>
<tr>
<td>Team</td>
<td>random</td>
<td>1996.2(±142.3)</td>
<td>1956.5(±139.7)</td>
<td>1795.9(±139.9)</td>
<td>1916.2(±165.2)</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td>1940.3(±187.3)</td>
<td>1996.7(±162.7)</td>
<td>1816.9(±165.2)</td>
<td>1918.0(±187.7)</td>
</tr>
<tr>
<td></td>
<td>linear</td>
<td>1733.1(±155.3)</td>
<td>1780.0(±159.2)</td>
<td>1858.4(±125.3)</td>
<td>1790(±156.2)</td>
</tr>
<tr>
<td>Island</td>
<td>random</td>
<td>2250.8(±133.2)</td>
<td>2206.6(±61.9)</td>
<td>1989.1(±155.6)</td>
<td>2148.9(±168.3)</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td>2242.4(±91.4)</td>
<td>2170.4(±77.75)</td>
<td>1976.9(±132.9)</td>
<td>2129.9(±152.5)</td>
</tr>
<tr>
<td></td>
<td>linear</td>
<td>2110.4(±104.1)</td>
<td>2119.9(±67.3)</td>
<td>1981.9(±104.1)</td>
<td>2070.8(±112.7)</td>
</tr>
</tbody>
</table>
Table 5: Percent decrease in team performance (negative values reflect an increase) of the best members from each trial in the final generation as a function of the training and testing environments.

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>random</th>
<th>clumped</th>
<th>linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>OET</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>random</td>
<td>-</td>
<td>0.036</td>
<td>0.103</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td>-0.072</td>
<td>-0.049</td>
<td>0.056</td>
</tr>
<tr>
<td></td>
<td>linear</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>random</td>
<td>-</td>
<td>0.028</td>
<td>0.100</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td>0.067</td>
<td>0.042</td>
<td>0.099</td>
</tr>
<tr>
<td></td>
<td>linear</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Island</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>random</td>
<td>-</td>
<td>0.020</td>
<td>0.116</td>
</tr>
<tr>
<td></td>
<td>clumped</td>
<td>-0.033</td>
<td>-0.070</td>
<td>0.089</td>
</tr>
<tr>
<td></td>
<td>linear</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and linear environments respectively). In contrast, the performance of the best teams generated via OET on the different training environments are not significantly different (p > 0.4 for the random environment, the best, versus the linear environment, the worst).

In general, the team approach had the smallest decreases in performance, Table 5, suggesting that the evolved teams were most robust with respect to the environment. However, an alternative explanation is simply that the team approach did so poorly (relatively speaking) on the training environment that the results on the test environments represent a smaller decline in fitness.

6. CONCLUSIONS AND FUTURE WORK

First, and perhaps most importantly, this research suggests that in general evolutionary techniques do evolve team behaviors that are fairly robust with respect to the environment. Even in the case of the linear environment, which is clearly quite different from the other environments, the behaviors evolved in other environments performed reasonably well. Similarly, behaviors learned in the linear environment applied well to the other environments. Thus, even though in real world applications the training environment is likely to be different, even significantly different, from the actual application environment, evolutionary algorithms show promise as a training technique for multi-agent systems.

Second, and perhaps surprisingly, some of the best all around results were obtained when training on the linear environment, which differed significantly from the other two. This suggests that given a choice of training environments (and constraints that limit how many can be used during training) the best all around results might be achieved by choosing the "outlier" environment.

Third, the results show that OET is the best of the three approaches for this type of search problem. The best teams produced via OET were significantly better than the best teams produced via the other approaches.

This research clearly just begins to touch upon the problem of evolving cooperative, multi-agent behaviors that are robust to the problem environment. In particular, we would like to look at the influence of training on several different environments. It seems likely, but needs to be demonstrated, that training on a number of extremely different, e.g. outlier, environments is likely to create the most robust teams. However, it is also certainly the case that this would make the overall problem harder. Thus, there is a likely a trade-off between creating the most robust teams versus creating teams that are more specialized, and hence have better performance, but on a smaller set of environments.

7. REFERENCES


