Neuroevolution for Adaptive Teams

Bobby D. Bryant and **Risto Miikkulainen**
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712 USA
{bdbryant,risto}@cs.utexas.edu

Abstract- We introduce the *Adaptive Team of Agents* (ATA), a system of homogeneous agents with identical control policies which nevertheless adopt heterogeneous roles appropriate to their environment. ATAs have applications in domains such as games, and can be evolved through neuroevolution. In this paper we show how ATAs can be evolved to solve the problem posed by a simple strategy game and discuss their application to richer environments.

1 Introduction

Multi-agent systems are a commonplace in social, political, and economic enterprises. Each of these domains consists of multiple autonomous parties cooperating or competing at some task. Multi-agent systems are often formalized for entertainment as well, with instances ranging from team sports to computer games. Games have previously been identified as a possible "killer application" for artificial intelligence (Laird and van Lent 2000), and a game involving multiple autonomous agents is a suitable platform for research into multi-agent systems as well.

Multi-agent systems can be composed either of homogeneous agents or of heterogeneous agents. Heterogeneous teams are often used for complex tasks (e.g., Balch 1998; Haynes and Sen 1997; Yong and Miikkulainen 2001). However, heterogeneous teams of sub-task specialists are brittle: if one specialist fails then the whole team may fail at its task. Moreover, when the agents in a team are programmed or trained for a pre-specified division of labor the team may perform inefficiently if the size of the team changes – for example, if more agents are added to speed up the task – or if the scope of the task changes dynamically.

We therefore propose teams constructed of *homogeneous* agents, each capable of adopting any role required by the team's task and capable of switching roles to optimize the team's performance in its current context. We call such a team an *Adaptive Team of Agents* (ATA). An ATA is a homogeneous team that self-organizes a division of labor so that it behaves like a heterogeneous team. It performs this division of labor without direction from a human operator, and can change the division dynamically as conditions change.

An ATA is robust because there are no critical task specialists that cannot be replaced by other members of the team; an ATA is flexible because individual agents can switch roles whenever they observe that a sub-task is not receiving sufficient attention. If necessary, an agent can alternate between roles continuously in order to ensure that sufficient progress is made on all sub-tasks. For many kinds of task an ATA could be successful even if there were fewer agents than the number of roles demanded by the task.

The optimal implementation for an ATA architecture depends on the task domain and the nature of the agents. Artificial neural networks (ANNs) are a good implementation choice in general, for several reasons: they are a universal computing architecture (Cybenko 1989; Siegelmann and Sontag 1994), they are fast, they generalize and tolerate noisy inputs, and they are robust in the face of damage or incomplete inputs.

For many multi-agent tasks the correct input-output mappings for the agents' controllers are not known, so it is not possible to program them or train them with supervised learning methods. However, networks can be evolved to perform a task in its actual context, discovering optimal mappings in the process (Yao 1995; Potter and De Jong 1995; Moriarty 1997).

In this paper we demonstrate that this approach is feasible: it is possible to evolve an ATA with ANN controllers for a non-trivial strategy game. In section 2 we describe the game designed for the experiment. In section 3 we describe the neuroevolutionary algorithm used to train the controllers, and in section 4 we describe the experimental design and results. In section 5 we discuss how ATAs could be evolved for richer environments such as commercial computer games.

2 The *Legion-I* Game

In order to test the feasibility of the ATA approach to multi-agent systems we designed a simple discrete-state strategy game that we call *Legion-I*. The game requires several legions to work together to minimize the economic depredation done to their province by an influx of barbarians.

The legionary and barbarian 'units' are autonomous agents in the game, each unit representing either a legion or a barbarian warband. The legions are the learning agents; the warbands follow pre-programmed behavior to pose a challenge for the legions. We enforce identical control poli-

 In *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, Vol. 3, pp. 2194-2201

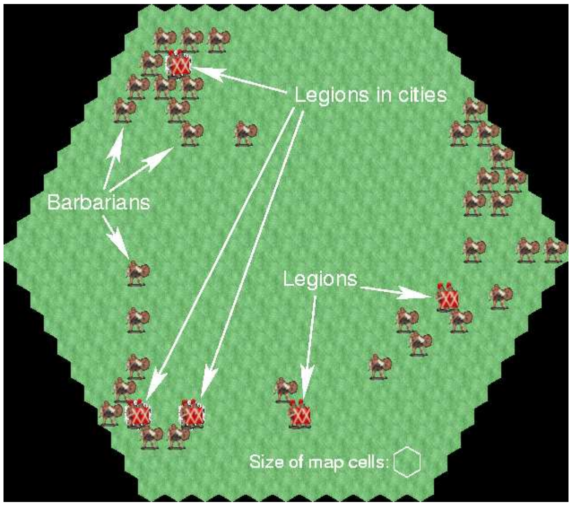

Figure 1: **The *Legion-I* game.** The large hexagonal playing area is tiled with small hexagons in order to quantize the positions of the game objects. Legions are shown iconically as close pairs of men ranked behind large rectangular shields, and warbands as individuals bearing an axe and a smaller round shield; each icon represents a large body of men, i.e. a legion or a warband. The cities are shown in white, mostly obscured by the legions that occupy them. All non-city hexes are farmland, shown with a mottled pattern. We use the game as a test bed for multi-agent learning methods, training the legions to contest possession of the playing area with the barbarians.

cies for all the legions and induce an ATA by creating a game environment that requires them to pursue two separate goals simultaneously, leading to a division of labor among them.

2.1 Structure of the game

The game is played on a map that is divided into cells by tiling the plane with hexagons (figure 1). Several randomly selected cells are designated as cities and the remaining are left to represent the surrounding farmland. The cells are also used to quantize the locations of the legions and warbands during play.

Play is conducted in turns. During a turn each unit can either remain stationary or move to an adjacent cell. Within a turn each legion selects its move and then each warband selects its move; each unit's move is implemented immediately, before play proceeds to the next unit.

Only one unit may be in any map cell after a move. The warbands are not able to eliminate or displace other units, so a warband may only remain stationary or move into an unoccupied adjacent cell. However, a legion eliminates a warband from play if it moves into the warband's cell, so a legion may either remain stationary or move into any adjacent cell that is not occupied by another legion.

A game begins with several legions placed in randomly selected cells on the map. These cells are selected independently of the choice of cells for the cities. There are no warbands in play at the beginning of the game; instead, the warbands enter play at random times and random map locations.

At the end of each turn a pillage rate is calculated. The barbarians are awarded 100 points of plunder for each city they occupy and one point for each cell of farmland. These amounts are accumulated over the course of the game to give the final game score. From the legions' perspective, lower game scores are better.

For the experiment reported in section 4 we used a hexagonal map with 21 horizontal rows of cells and 21 cells from corner to corner on the diagonals, giving a total of 331 cells. Three of the cells were designated as cities and five legions were provided for the defense of the province. The barbarians entered at an average rate of one warband per turn, and the games lasted for 200 turns.

2.2 Game scores

The scoring was designed to require the legions to garrison the cities: after a few barbarians are in play each ungarrisoned city can be expected to result in the loss of 100 points of plunder each turn. The length of the game and barbarian appearance rate were chosen so that the number of warbands in play would ramp up from zero to 200, giving an *average* of about 100 in play over the course of the game. Thus the barbarians can accumulate an average of about 400 points of plunder per turn, 300 for the three cities and about 100 more for the other barbarians milling about the countryside. (The expected value is 397 points per turn, but the randomized arrivals make the number vary slightly from game to game.)

If the legions are content to garrison the cities and take no other action against the barbarians then their province will still suffer an average of 100 points of pillage per turn over the course of the game, a reduction of the pillage rate to 25% of the worst case. Since our goal was to train the legions as an ATA we provided five legions for a game with three cities: if the legions partition their behavior so that three garrison the cities and the rest eliminate barbarians in the countryside then the team can reduce the pillage rate to *less than* 25% of the worst case. This is the target division of labor that the game was designed to elicit.

Notice that there is no *a priori* expectation that the legions will be able to completely suppress the pillaging. Since the warbands appear at random locations and move at the same speed as the legions it becomes increasingly difficult for the legions to eliminate them as their density on the map decreases. Thus the performance of the legions is ultimately limited to an equilibrium condition between the

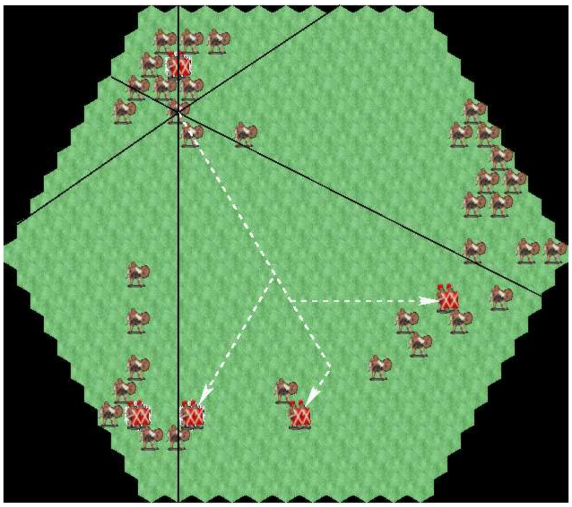

Figure 2: **Barbarian sensor geometry.** Black lines show the borders of the six sensory fields of view for one warband near the northwest corner of the map. The borders emanate from the center of the warband's cell and out through its six corners. Game objects in cells split by a border are reckoned to be in the field to the clockwise of that border. White arrows show the hexagonal Manhattan distances to each of the three legions in the warband's southeastern sensory field of view. These lines are traced from center to center of the hexagonal map cells along a shortest path between the sensing barbarian and the sensed object. All the distances are digested to a single numeric value for each field of view, giving the warband only a fuzzy sense of its world.

number of legions in play, the rate of appearance of the warbands, and the size of the map. (In practice we find that the legions can reduce the pillage rate to about 10% of the worst case, when the game parameters are set as described above.)

2.3 The warbands' controllers

The barbarian warbands are preprogrammed for a behavior that is aggressive enough to serve as a foil for the legions to learn against. Any time a warband starts its move in a city it remains stationary for the current turn, since no move will increase the amount of plunder it obtains. Otherwise it consults its sensors to decide on a direction to move.

A warband has two egocentric sensor arrays, one to detect cities and another to detect legions. Each array consists of six elements, one for each of the six "pie slice" fields of view induced by the hexagonal map grid and centered on the warband. At the start of a warband's turn its sensor elements are loaded with the values $\sum_i \frac{1}{d_i}$ for objects i of the correct type within each element's field of view, where d_i is the Manhattan distance to the object as traced on a hexagonal grid (figure 2).

The two sensor arrays are then treated as "motivation" arrays, with the city sensor attracting the warband and the legion sensor repelling it. The elements of the legion sensor are first permuted to invert their geographic sense and then the two motivation arrays are combined as $\mathcal{M}_{final} = \mathcal{M}_{cities} + 0.9\mathcal{M}_{legions}$. The 0.9 factor slightly reduces a warband's tendency to flee the legions so that it will take some risks to plunder the cities. The direction corresponding to the maximal element in $\mathcal{M}_{final}$ is the warband's choice for its current move.

The selected move is not made if the game rules forbid it. If the adjacent map cell in the chosen direction is already occupied by a legion or another warband then the warband remains stationary for the current turn instead. If the selected move would take the warband off the edge of the map then $\mathcal{M}_{final}$ is examined to see what the second choice for a move would be, and that move is made instead. If that second choice is also an illegal move then the warband remains stationary without considering further choices.

The net effect of the barbarian control algorithm is that the warbands will approach cities and flee legions, with a slight preference for approaching the cities. As a result the warbands will enter any ungarrisoned city and crowd around cities that are already occupied, but will flee the legions in the countryside so long as crowds of other warbands do not obstruct their flight.

2.4 The legions' controllers

The legions are controlled by artificial neural networks that map their sensory inputs onto a choice of actions. In order to conform to the definition of an ATA each legion must use an identical controller network. In practice we reuse the same network for each legion in turn; this method ensures that each legion has an identical control policy and any individual differences in their behavior arise purely from the differences in their sensory inputs.

Since we demand more subtle behavior from the legions than from the barbarians we provided them with more elaborate sensors (figure 3). Each legion is provided with three egocentric sensor arrays for detecting cities, barbarians, and other legions. Each of the legions' sensor arrays divides the map into six "pie slices" just as the warbands' sensors do, but the legions' sensors also divide the map into two concentric rings. The inner ring only detects the presence of game objects in the map cells immediately adjacent to the sensing legion, and the outer ring detects any game objects farther away. The sensor elements in the inner ring are set to 1.0 if a game object of the appropriate type is in the adjacent map cell, or 0.0 otherwise. The sensor elements in the outer ring are set to $\sum_i \frac{1}{d_i}$ for the appropriate game objects i, just as for the barbarians' sensors, except that any objects in the cells adjacent to the legion are excluded from the sum.

A 13^{th} sensor element for each of the three arrays de-

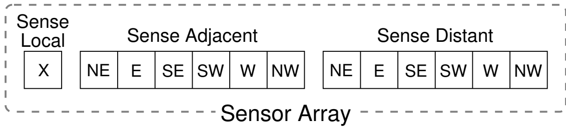

Figure 3: **Legion sensor architecture.** Each sensor array for the legions each consists of three sub-arrays. A single-element sub-array detects objects colocated in the map cell that the legion occupies. Two more six-element sub-arrays detect objects in six radial fields of view just as the barbarian's sensors do (figure 2), except that one only detects adjacent objects and the other only detects objects further away. The legions are equipped with three complete sensor arrays, one each for detecting cities, barbarians, and other legions. The three 13-element arrays are concatenated to serve as a 39-element input layer for an artificial neural network that controls the legion's behavior (figure 4).

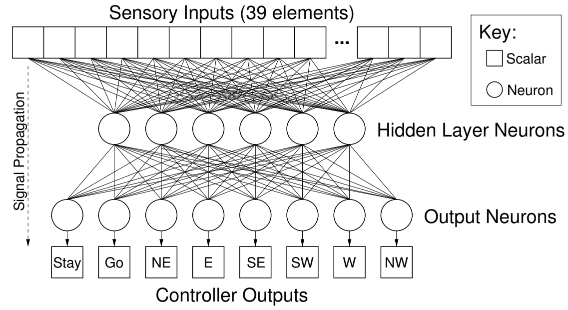

Figure 4: **Controller network.** The values obtained by a legion's sensors are propagated through an artificial neural network to create an activation pattern at the network's output. The activation pattern is then interpreted as a choice of one of the discrete moves available to the legion. When properly trained the network serves as the 'brain' for an intelligent agent.

tects the presence of a game object in the legion's own cell. This sensor proved to be necessary so that the legions could detect when they are in a city. It is not needed for other types of objects, but was provided on all three sensor arrays to keep their structures uniform.

After a legion's sensory inputs have been calculated, all the element values from the three sensor arrays are concatenated into a flat 39-element array to serve as the input for the artificial neural network that controls the legion. The network is a fully-connected feed-forward network with a single hidden layer (figure 4). Preliminary experiments showed that a hidden layer of six neurons works well on the *Legion-I* problem.

A legion's controller network has eight units in its output layer, representing the choices *Stay*, *Go*, and the six cardinal directions imposed by the hexagonal tiling of the map. After a sensory input has been propagated through the network the activations at the network's output layer are decoded by a two-step process. If the activation of the *Stay* unit is higher than the activation of the *Go* unit then the legion remains stationary for the current turn. Otherwise the six outputs associated with the cardinal directions are examined to find which has the highest activation level, and a move is taken in that direction. If the move is off the map or onto another legion then the legion remains stationary instead.

This concludes our description of the game and the agents' controllers. In the next section we describe the method used to train the legions to behave as an adaptive team.

3 The Learning Algorithm

The legions' controllers are trained using neuroevolution with enforced sub-populations (ESP), a method that has been shown to be powerful for learning control tasks (Gomez and Miikkulainen 1999; Gomez 2003; Gomez and Miikkulainen 2003). ESP is an extension of the earlier SANE algorithm, which has also been shown to work well for various discrete-state applications such as the game *Go* (Moriarty and Miikkulainen 1997; Moriarty 1997).

3.1 Neuroevolution with ESP

ESP is a direct-encoding neuroevolutionary algorithm, i.e. it specifies a network's weights directly in the chromosomes. Its major innovation is that the chromosomes only encode the weights for individual neurons rather than complete networks, and a separate sub-population is maintained for each neuron in the network (figure 5). The sub-populations are kept separate during breeding.

The evolutionary fitness of the neurons is determined by selecting a random neuron from each sub-population, combining the selected neurons to form a complete network, evaluating the performance of the network on the target task, and finally ascribing the network's score back to each neuron that participated in the evaluation. This process is repeated during a generation until all the neurons have been evaluated. As generations pass the sub-populations co-evolve to produce neurons that "cooperate" in a good solution when assembled into a network.

3.2 Application of ESP to the *Legion-I* problem

To represent the legions' controllers we used six sub-populations of chromosomes, one for each neuron in the hidden layer. We used flat arrays of floating point numbers for the chromosomes, representing the concatenation of a single neuron's input and output weights. Each sub-population consisted of 500 chromosomes. (In general larger populations produce better results, but the run time of the algorithm is approximately proportional to the population size. With 500 chromosomes our run time was just under 24 hours on a 1.0 GHz machine.)

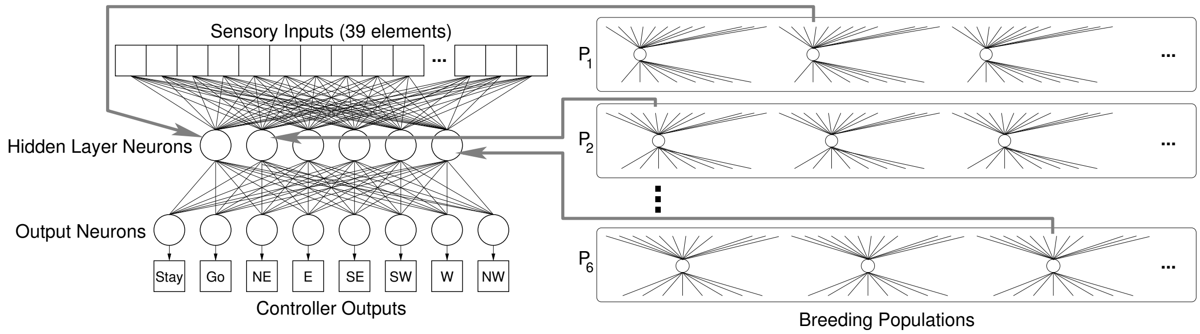

Figure 5: **Neuroevolution with ESP.** In neuroevolution with enforced sub-populations a separate breeding population is maintained for each neuron in a network, shown here as $\{P_1, P_2, \ldots, P_6\}$ for the six populations providing neurons for the hidden layer of our controller network. (Our chromosomes represent both input and output weights for the neurons in the hidden layer, so the neurons in the output layer do not require any additional genetic representations.) Networks are assembled by drawing one neuronal chromosome at random from each sub-population. The resulting network is tested and its fitness is ascribed back to each of its component neurons. The process is repeated until a fitness has been determined for every neuron in all the populations for the current evolutionary generation. As generations pass the sub-populations co-evolve to produce neurons that work well with the others in a network.

The populations were initialized with a random floating point value for each weight in each chromosome. The values were generated in an exponential distribution with a mean and standard deviation of 1.0, yielding a large number of small weights and a small number of large weights. Each random weight was inverted to a negative number on a 50% chance, to give a symmetrical distribution.

Networks were evaluated on complete 200-turn games of *Legion-I* using the game parameters described in section 2. The evaluations dominated the learning algorithm's run time, though they became faster as the legions learned the game, since progress at eliminating barbarians left fewer agents in play to be moved each turn. Near the end of training the games could be played at a rate of about three per second. Every neuron was evaluated three times per generation by playing three games with different sequences of random numbers, in order to simultaneously average out the luck of the neurons selected for the network plus any differences in the inherent difficulty of the game setups.

At the end of each generation the chromosomes were updated by breeding strictly within each sub-population. Matings were made with a preference for using the more fit neurons, but there was a slight possibility of using even the least fit neurons as well. This preference was implemented by sorting the neurons according to their fitness and then replacing each neuron *in situ* with a newly bred neuron, starting with the worst and working up to the best. Each neuron was replaced by breeding two neurons randomly selected from those that had not yet been replaced, so that the better neurons had more opportunities to contribute their genes to the next generation.

Crossovers were either 1-point or 2-point, each with a 50% chance. After crossover each weight was mutated with a 1% chance. Mutations were applied as additions to the current value, using deltas with the same distribution that was used to generate the original chromosomes' weights.

This design for the evolutionary algorithm allowed us to apply ESP to the problem of training the legions as an adaptive team. In the next section we describe the learning experiment done with these arrangements.

4 The Experiment

In this section we describe the experimental procedures and results obtained on the *Legion-I* ATA problem.

4.1 Experimental discipline

The evolutionary algorithm was run for 250 generations, resulting in 375,000 games played during a run. Auxiliary runs out to 1000 generations showed continued very slow improvements in performance, but most of the learning was done even before 250 generations, allowing us to select that as the stopping point for the formal experiment. We suspect that the legions' performance approaches the optimum equilibrium-state performance asymptotically.

As a sanity check we repeated the learning algorithm five times with five different seeds for the random number generator. The seeds controlled the generation of the populations' initial weights, all randomized breeding decisions, and the generation of the game setups used for training and validation. We used independently generated training and validation games for the five runs to make the runs as independent as possible, in order to obtain five independent samples of the learning algorithm's performance on the problem.

At the end of each generation in a run a nominal best network was created by selecting the most fit neuron from each

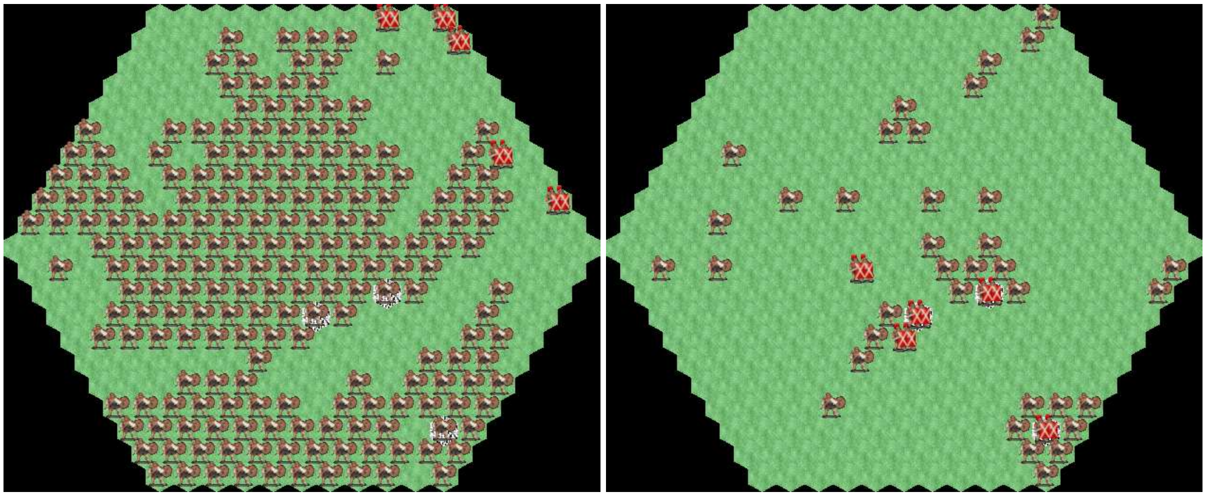

Figure 6: **Progress in learning.** Two end-of-game screenshots show the legions' performance before and after training. *Left:* Before training the legions move haphazardly, drift to an edge of the map, or sit idle throughout the game, thereby failing to garrison the cities and allowing large concentrations of barbarians to accumulate in the countryside. *Right:* After training the legions have learned to split their behavior so that three are deployed as garrisons for the three cities while the other two move to destroy most of the barbarians pillaging the countryside. The desired adaptive behavior has been induced in the team.

sub-population and assembling them together as a complete network. This *nominal* best network may not actually be the best possible network that could be assembled from the available neurons; the nominal network was used in lieu of searching the 6^{500} possible combinations of neurons to find the actual best network.

The nominal best network for each generation was tested against a 25-game validation set to measure the progress of the learning algorithm, and the network was saved if it yielded the best average score of any generation so far. At the end of a run the nominal best network that had performed best on the validation set was returned as the controller produced by that run, regardless of which generation had produced it.

The resulting controller networks can be evaluated either qualitatively or quantitatively. Either way, it is clear that all five runs evolved networks capable of a dynamic division of labor among the legions.

4.2 Qualitative results

The performance of the controller networks was evaluated qualitatively by observing the real-time animations of game play. In every case that we have examined the legions begin the game with a general rush toward the cities, but within a few turns negotiate a division of labor so that some of the legions enter the cities and remain there as garrisons while the others begin to chase down barbarian warbands in the countryside. The only time the cities are not garrisoned promptly is when two of them mask the third from the legions' low-resolution sensors. However, even in those cases the third city is garrisoned as soon as one of the roaming legions pursues a barbarian far enough to one side to have a clear view of the third city so that it can "notice" that it is ungarrisoned.

The legions also show surprisingly persistent long-term behavior for memoryless agents. It is extremely rare to see a well trained legion abandon a city that it has occupied; garrison duty almost always lasts the entire length of the game. The roaming legions also give an appearance of purposeful behavior, often moving in a straight line halfway across the map to disperse some concentration of barbarians spotted from afar. On one occasion a legion was observed to approach a city and make two smooth loops around it, then move away after having dispersed or destroyed all the barbarians crowded about the city. The roaming legions do spend a disproportionate amount of time near the cities because that is where the barbarians concentrate, but it is rare to see a non-garrison legion spend multiple turns adjacent to a garrisoned city as if hoping to get in.

A feel for these qualitative behaviors can be obtained by comparing end-of-game screenshots taken early and late during a training run, as shown in figure 6.

4.3 Quantitative results

The performance of each run was evaluated quantitatively on a 25-game test set of games similar to the validation sets, but generated with a different random seed. The same test set was used for each of the five runs.

The scores on the games in the test set show that all five runs produced controllers that allowed the legions to reduce pillaging well below the 25% rate obtainable by garrisoning

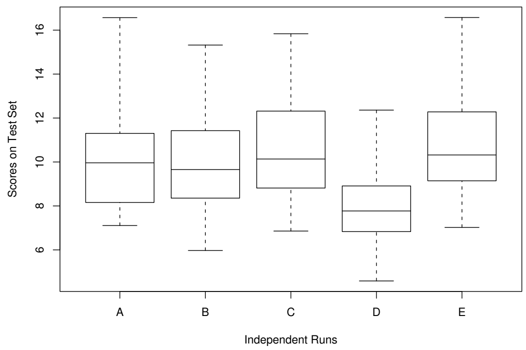

Figure 7: **Performance on the test set.** Boxplots show the distribution of scores on the 25 games of the test set for the adaptive teams produced by five independent runs of the learning algorithm, labeled A through E. (A box shows the median and quartiles of the scores and whiskers show the extremes. Half the scores lie with the bounds of the box and a quarter lie within the range of each whisker. Asymmetries indicate the degree of skew to the set of scores.) Scores are shown as a percentage of the maximum amount of pillage that the barbarians could inflict without any legions present. The mean score over all the test games was 9.96%, and even the worst score among all the test games (16.58%) was a substantial improvement over the pillage rate obtainable by simply garrisoning the cities (25%).

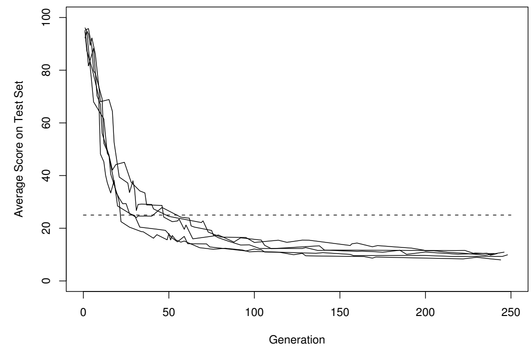

Figure 8: **Learning curves.** The plot shows learning progress for the five independent runs as measured on the common test set. Each point shows the average score on the 25 games of the set. To reduce clutter we plot points only for the generations where progress was made on the validation set. (The plots are not monotonic because progress on the validation set does not strictly imply progress on the test set.) The horizontal line shows the 25% pillage rate obtainable by garrisoning the cities only. All the runs were performing substantially better than the 25% threshold by generation 80, and continued to show slow improvements thereafter. Runs out to 1000 generations show continued very slow improvement, suggesting that the legions are refining their task asymptotically as evolution continues.

the cities and taking no further actions against the barbarians (figure 7).

We also used the common test set to observe the progress of learning in each of the five runs. The learning curves show the familiar pattern of diminishing returns seen in many machine learning experiments, with fast initial learning tapering off into slower steady progress (figure 8). There was very little variety in the speed of learning between the five runs, and no stair-step pattern to suggest that the legions' two modes of behavior were learned sequentially. (Observations confirm that the legions begin chasing barbarians even before they have learned to garrison all the cities.)

Observations and measurements both show that the legions learned to behave as an adaptive team in all five runs, allowing us to conclude that our neuroevolutionary method can endow agents with the necessary intelligence for adaptive behavior, at least for a problem of *Legion-I*'s complexity. We now turn to the prospects of inducing ATAs in more complex environments.

5 Discussion and Future Work

The experiment described above shows that Adaptive Teams of Agents are a feasible approach to multi-agent problems, and that they can be created by neuroevolutionary methods. We must now evaluate the prospects of creating and applying ATAs at the scale, complexity, and "messiness" of real-world applications.

In further preliminary experiments we have found that adding additional modes of behavior to the *Legion-I* environment, such as road-building, makes the task much harder to learn. Promising approaches to the more difficult problem include the use of shaping (Randløv 2000; Gomez and Miikkulainen 1997) or simulated annealing (Lozano et al. 1999) to boost learning power as the demands on the agents' behavior are increased.

Another challenge is the construction of adequate sensory representations for richer environments. The ATAs in *Legion-I* learned their task with only a very fuzzy representation of their environment, but other domains can be expected to require more detailed representations. Moreover, some domains will require representations for recursive symbolic relationships such as "in" and "on". Processing such relationships may require us to draw on techniques developed for natural language processing in artificial neural networks (Elman 1990; Miikkulainen 1996).

We therefore expect the development of ATAs for richer environments to lead to productive basic research, and to practical applications in computer gaming as well. In commercial computer games there is a constant need for improved intelligent agents. Many games include sets of agents that would work well as adaptive teams. For example, games in the popular civilization-building genre usually offer a settler or pioneer unit type, deployed in a number that varies over the course of the game, each functionally identical but capable of many disparate activities that must be pursued in building a competitive civilization. Application of our methodology to such domains can be expected to uncover challenges and solutions that extend the state of the art for learning-based multi-agent systems, and the improved ability to implement robust game intelligences through machine learning methods will help game companies meet increased consumer expectations economically.

We have shown that the Adaptive Team of Agents is a feasible concept, and that neuroevolution is a practical way of training adaptive agents for some environments. We anticipate that ATAs will soon find real-world applications in the gaming industry, and that scaling this new agent architecture to the needs of commercial applications will stimulate the invention of powerful new techniques for inducing sophisticated behavior in autonomous intelligent agents.

Acknowledgments

This research was supported in part by the National Science Foundation under grant IIS-0083776 and the Texas Higher Education Coordinating Board under grant ARP-003658-476-2001.

The images used in *Legion-I*'s animated display are derived from graphics supplied with the game *Freeciv*. See `http://www.freeciv.org/` for credits.

Bibliography

Balch, T. (1998). *Behavioral Diversity in Learning Robot Teams*. PhD thesis, Georgia Institute of Technology. Technical Report GIT-CC-98-25.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14:179–211.

Gomez, F. (2003). *Learning Robust Nonlinear Control with Neuroevolution*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin.

Gomez, F., and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342.

Gomez, F., and Miikkulainen, R. (1999). Solving non-Markovian control tasks with neuroevolution. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. Denver, CO: Morgan Kaufmann.

Gomez, F., and Miikkulainen, R. (2003). Active guidance for a finless rocket using neuroevolution. In *Proceeedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*. San Francisco, CA: Morgan Kaufmann.

Haynes, T. D., and Sen, S. (1997). Co-adaptation in a team. *International Journal of Computational Intelligence and Organizations*.

Laird, J. E., and van Lent, M. (2000). Human-level AI's killer application: Interactive computer games. In *Proceedings of the 17th National Conference on Artificial Intelligence*. Cambridge, MA: MIT Press.

Lozano, J. A., Larrañaga, P., Graña, M., and Albizuri, F. X. (1999). Genetic algorithms: bridging the convergence gap. *Theoretical Computer Science*, 229(1–2):11–22.

Miikkulainen, R. (1996). Subsymbolic case-role analysis of sentences with embedded clauses. *Cognitive Science*, 20:47–73.

Moriarty, D. E. (1997). *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin. Technical Report UT-AI97-257.

Moriarty, D. E., and Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5:373–399.

Potter, M. A., and De Jong, K. A. (1995). Evolving neural networks with collaborative species. In *Proceedings of the 1995 Summer Computer Simulation Conference*.

Randløv, J. (2000). Shaping in reinforcement learning by changing the physics of the problem. In *Proc. 17th International Conf. on Machine Learning*, 767–774. Morgan Kaufmann, San Francisco, CA.

Siegelmann, H. T., and Sontag, E. D. (1994). Analog computation via neural networks. *Theoretical Computer Science*, 131(2):331–360.

Yao, X. (1995). Evolutionary artificial neural networks. In Kent, A., and Williams, J. G., editors, *Encyclopedia of Computer Science and Technology*, vol. 33, 137–170. Marcel Dekker Inc.

Yong, C. H., and Miikkulainen, R. (2001). Cooperative coevolution of multi-agent systems. Technical Report AI01-287, Department of Computer Sciences, The University of Texas at Austin.