Acquiring Visibly Intelligent Behavior with Example-Guided Neuroevolution

Bobby D. Bryant
Department of Computer Science & Engineering
University of Nevada, Reno
Reno, NV 89557-0148
bdbryant@cse.unr.edu

Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-0233
risto@cs.utexas.edu

Abstract

Much of artificial intelligence research is focused on devising optimal solutions for challenging and well-defined but highly constrained problems. However, as we begin creating autonomous agents to operate in the rich environments of modern videogames and computer simulations, it becomes important to devise agent behaviors that display the visible attributes of intelligence, rather than simply performing optimally. Such visibly intelligent behavior is difficult to specify with rules or characterize in terms of quantifiable objective functions, but it is possible to utilize human intuitions to directly guide a learning system toward the desired sorts of behavior. Policy induction from human-generated examples is a promising approach to training such agents. In this paper, such a method is developed and tested using Lamarckian neuroevolution. Artificial neural networks are evolved to control autonomous agents in a strategy game. The evolution is guided by human-generated examples of play, and the system effectively learns the policies that were used by the player to generate the examples. I.e., the agents learn visibly intelligent behavior. In the future, such methods are likely to play a central role in creating autonomous agents for complex environments, making it possible to generate rich behaviors derived from nothing more formal than the intuitively generated examples of designers, players, or subject-matter experts.

Introduction

In the field of artificial intelligence we ordinarily seek optimal solutions to difficult problems. For example, we search for the shortest paths, lowest-cost plans, most accurate predictions, and so forth. For game applications that requirement may vary somewhat; for example, a game-playing AI may need to offer various levels of competence, i.e. provide several levels of difficulty that challenge a player by just the appropriate amount, or it may need to be somewhat unpredictable in order to maintain players' interest.

In videogames and computer simulations, AI controllers face a special requirement: agent behavior has to be *visibly intelligent*. That is, the behavior of the agents must *look* right to observers, sometimes even at the expense of optimal performance. For example, the agents should be able to adopt different behaviors at different times, and different roles in a team; they should be able to generalize their actions to new contexts, and create novel actions in old contexts (Bryant 2006).

In such visible contexts it is especially important for agents to observe some specific behavioral policy, rather than simply optimizing their actions: Game players expect different types of creatures to behave differently, and appropriately for their kind. Nor is the target design for such behaviors always simply "smart"; games and simulators must sometimes model agents that behave suboptimally or even outright foolishly, though still believably.

Experience with the AI agents in commercial videogames suggest that directly programming such behaviors is not an easy task. Nor is it cheap; the US Army spent millions of dollars to acquire AI automation for the *OPFOR* opponent in its *OneSAF* military simulator. Even when delivered and debugged, a similar cost and delay will be faced whenever it becomes necessary to model a new potential enemy that follows a different operational doctrine.

Thus it is natural to consider using machine learning methods to train the necessary controllers. However, machine learning research – like most of the rest of our AI efforts – has traditionally focused on obtaining solutions that are optimal, i.e. result in maximum values for a formally specified reward function. However, when the goal is to generate visibly intelligent behavior in a rich environment, such a function is very difficult to specify. The state and action spaces are extremely large, and the goal of the learning is inherently subjective. It may be as difficult to specify the necessary reward function as it would be to program the target behavior directly.

Therefore it is useful to consider mechanisms for guiding learning by means of human-generated examples of the target behavior (figure 1). If learning algorithms can be devised to exploit such examples in order to induce the behavioral policies that created them, it will become possible to create AI controllers for sophisticated behaviors without the need for knowledge engineers or AI programmers. Subject-matter experts such as game designers or intelligence officers will be able to generate appropriate examples based on their intuitions, and the learning system will be able to distill the target behavioral policy from those examples. Such a system would offer substantial improvements over the methods now used to produce AI controllers for games and simulators, and potentially provide improved behavioral mod-

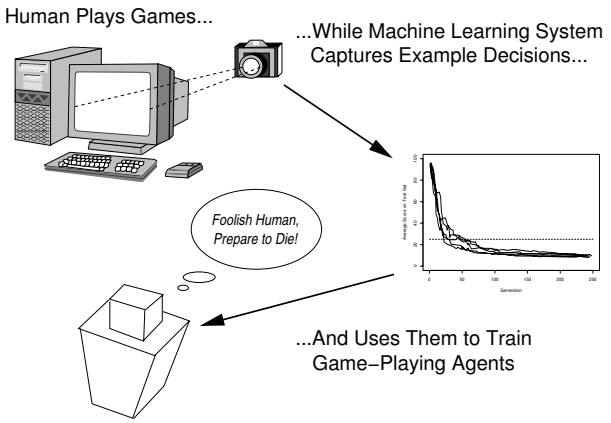

Figure 1: **Policy induction in a game context.** Human examples of play can be captured by writing an observer into the game engine, and the examples collected by the observer can be used to help train a controller for a game agent.

elling as a result.

Computational intelligence has recently shown promise for creating artificial intelligence in games, and neuroevolution in particular has been used to create agents that adapt their behavior in complex simulated environments (Miikkulainen *et al.* 2006). This paper presents a mechanism for using human-generated examples to guide neuroevolution to visibly intelligent behavioral policies. The method is evaluated in a formal strategy game where its success can be measured quantitatively. The method is shown to not simply copy the examples, but to induce the policies behind them, providing a promising starting point for creating visibly intelligent behavior for games and simulators more generally.

Background

This section provides background for the policy induction experiments. The first subsection describes the strategy game in which the autonomous agents operate. The neuroevolutionary algorithm that is normally used to train them is then described, followed by the modification to that algorithm for the policy induction experiments.

The Legion II Game/Simulator

Legion II is a discrete-state strategy game designed to test adaptive behavior in teams of autonomous intelligent agents. The game pits legions against barbarian warbands on a map that represents a province of the Roman empire at an abstract level (figure 2). The legions, each represented as a single agent with an egocentric view of the world state, must learn to protect the cities and farms of the province against a steady influx of barbarian pillagers. Since the cities are by far the most lucrative source of pillage it is essential to protect each with a garrison. Any legions beyond the number required for the garrisons are free to roam the countryside to disperse the barbarians pillaging the farms. An *in situ* division of labor is necessary in order to minimize the pillaging: if the legions merely garrison the cities, the number of barbarians in the countryside builds up over the course of the

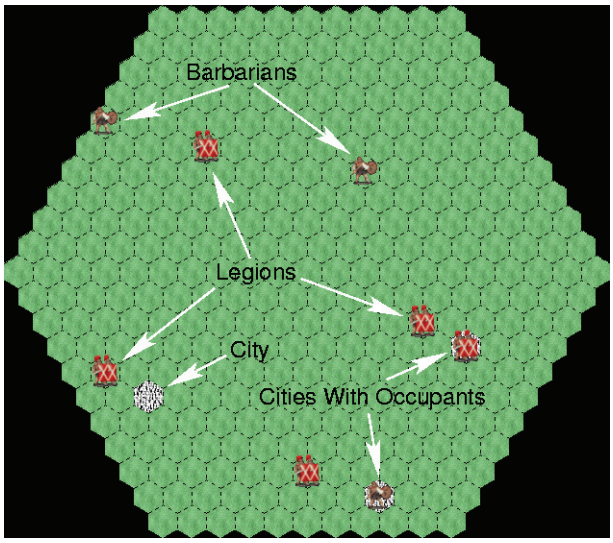

Figure 2: **The *Legion II* game.** A large hexagonal playing area is tiled with smaller hexagons in order to quantize the positions of the game objects. Legions are shown iconically as close pairs of men ranked behind large rectangular shields, and barbarians as individuals bearing an axe and a smaller round shield. Each icon represents a large body of men, i.e. a legion or a warband. Cities are shown in white, with any occupant superimposed. All non-city hexes are farmland, shown with a mottled pattern. The game is a test bed for multi-agent learning methods, wherein the legions must learn to contest possession of the playing area with the barbarians. Animations of the *Legion II* game can be viewed at `http://nn.cs.utexas.edu/keyword?ATA`.

game, and an unacceptable amount of pillage is inflicted on the province.

The legions sense the game state via radial "pie-slice" radar sensors. Separate sensor arrays are used to detect the various game objects: cities, barbarians, and other legions. Each sensor returns $\sum_i \frac{1}{d_i}$, where d is the hex-grid generalization of Manhattan distance and i indexes all the objects of the relevant type in the sensor's field of view. This sensor architecture gives only a fuzzy, alias-prone view of the game state. However, the system works well as a threat/opportunity indicator: a high sensor activation indicates either a few objects nearby or else a greater number at a greater distance. In order to improve the legion's local decision-making ability, additional point sensors are provided to indicate the presence of objects colocated with the sensing legion, or in a directly adjacent map cell (figure 3).

In order to control the legions during a game, their scalar sensor values, 39 in all, are fed through a feed-forward neural network with a single hidden layer of ten neurons and an output layer of seven neurons (figure 4). A fixed-value bias unit is also provided for each neuron. The size of the hidden layer was chosen by means of exploratory experiments, but has not been formally optimized. The output neurons are associated with the seven possible actions a legion can take in its turn: remain stationary, or move into one of the six adjacent map cells. This localist *action unit coding* is decoded by selecting the action associated with the output

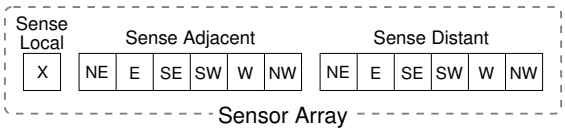

Figure 3: **A legion's sensor architecture.** Each sensor array for a legion consists of three sub-arrays. A single-element sub-array (left) detects objects colocated in the map cell that the legion occupies. Two six-element sub-arrays detect objects in the six radial fields of view; one only detects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

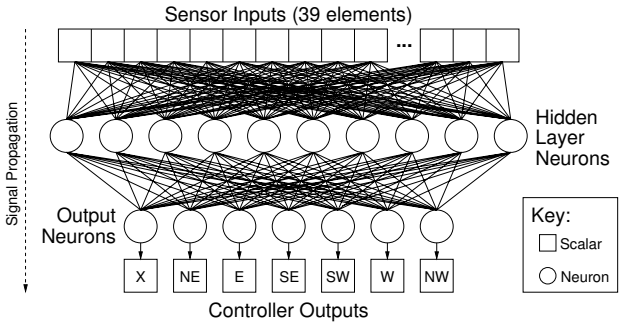

Figure 4: **A legion's controller network.** During play the values obtained by a legion's sensors are propagated through an artificial neural network to create an activation pattern at the network's output. This pattern is then interpreted as a choice of one of the discrete actions available to the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

neuron that has the highest activation level after the sensor signals have been propagated through the network (Bryant & Miikkulainen 2006).

The neural networks are trained by neuroevolution, as described below. The same network is used to control each legion in its turn, forcing an identical control policy upon each agent in the team. Nevertheless, it is possible to train the network so that the legions dynamically adopt distinctive roles in the team – garrison or rover – and act out those roles on the basis of a sequence of egocentric decisions that are purely local in space and time.

Neuroevolution with Enforced Sub-Populations

Genetic algorithms can be used to train artificial neural networks when annotated training examples and/or error gradient information are not available (Whitley 1995; Yao 1999). Many varieties of neuroevolutionary algorithm have been devised; this work uses *neuroevolution with enforced sub-populations* (ESP) (Gomez & Miikkulainen 1999; Gomez 2003).

The ESP algorithm works by dividing genetic representations into distinct chromosomes, one per neuron in the network, and maintaining a separate breeding population for each chromosome. Networks are constructed for fitness evaluation by drawing one random chromosome from each of those populations; after the evaluation the measured fitness of the network is ascribed back to each of the chromosomes that were used in its construction (figure 5). The evaluation is noisy, since a "good" neuron may be evaluated in a network with one or more "bad" neurons, or vice versa. Since the true fitness of a neuron is dependent on what sort of neurons are available in the other populations, an improved estimate of its fitness can be obtained by evaluating it several times per generation, in a network constructed from a different random selection of neurons each time, and averaging the fitness ratings a neuron obtained in its several evaluations. When evaluations have been obtained for all the neurons (or equivalently, the chromosomes), breeding is done separately for each population. Over time the populations co-evolve to provide neurons that work well together to perform the network's task.

Otherwise, ESP operates as ordinary direct-encoding neuroevolution, and evolves only the weights for a pre-specified network topology. For the experiments reported here, each neuron was evaluated three times per evolutionary generation. At the end of each generation a nominal best network was constructed by selecting the highest-fitness neuron from each of the populations. That nominal best network was archived iff it performed better on a validation set of games than those tested in previous generations. At the end of a fixed-length run, the most recently archived network was returned as the output of the learning algorithm.

During breeding, 1- and 2-point crossover were used with equal probability. Point mutations were applied with an independent 1% chance for each weight. Mutations were applied as deltas selected from the exponential distribution, inverted to a negative delta with a 50% chance; this distribution provides small deltas with a high probability and large deltas with a low probability, supporting both local and global search of the solution space.

With the advent of cheap desktop supercomputers and fast neuroevolutionary algorithms such as ESP, it has become feasible to apply neuroevolution to agent control in videogames and simulators. Networks in the breeding population are evaluated by using them to control an agent during a game or at some game-relevant task, and the performance of the agent in the game or task is used as the fitness measure for the network (Bryant & Miikkulainen 2003; Lucas 2004; Stanley, Bryant, & Miikkulainen 2005; Miikkulainen *et al.* 2006; Bryant 2006).

The ESP algorithm has previously proven to be a powerful approach to reinforcement learning for motor control tasks, including non-Markov decision problems. It also works well for the stateless feed-forward networks used as the agent controllers in the *Legion II* game.

Lamarckian Neuroevolution

In the early Nineteenth Century, Jean-Baptiste Lamarck proposed a mechanism for biological evolution whereby organisms passed their phenotypic adaptations along to their offspring (Lamarck 1809). The suggestion has long since been rejected, as there is no general mechanism for transferring such adaptations back into the organism's genotype

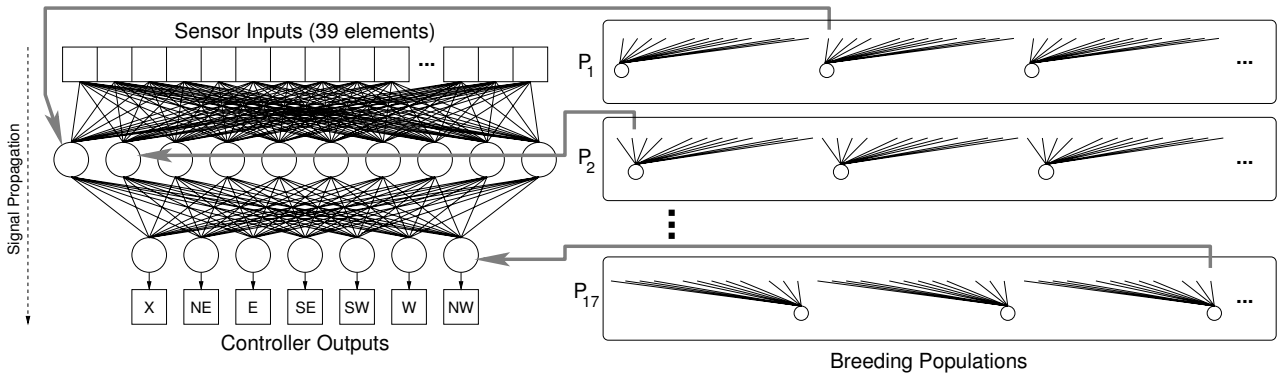

Figure 5: **Neuroevolution with ESP.** In *neuroevolution with enforced sub-populations* (ESP), a separate breeding population is maintained for each neuron in a network. For the *Legion II* controller network there are 17 such populations, shown in this figure as $\{P_1, P_2, ..., P_{17}\}$. Networks are assembled by drawing one chromosome at random from the population associated with each position in the network and instantiating the neurons the chromosomes represent (gray arrows). The resulting network is tested in the environment and its fitness is ascribed back to each of the chromosomes that specified its neurons. The process is repeated until a fitness score has been determined for every chromosome in all of the sub-populations, at which time an evolutionary generation is complete. Each population is then updated by selective breeding with random crossovers and mutations, independently of the other populations. As generations pass, the sub-populations co-evolve to produce chromosomes describing neurons that work well with the others in the network.

for propagation to its offspring. However, for evolutionary algorithms it is possible to incorporate phenotypic adaptation back into the genotype via reverse engineering. For example, if an adaptation step is added to the evolutionary life cycle in neuroevolution, any resulting modifications to a network's weights can be written back to the network's representation in the evolutionary population (Grefenstette 1991; Whitley *et al.* 1993; Ku, Mak, & Siu 2000). This Lamarckian model was used for guiding evolution with examples in the experiments reported below.

Examples were generated by having a human play the game according to some target behavioral policy (i.e., game strategy). Due to the discrete-state nature of the *Legion II* game, examples are easily recorded as $<$*state, action*$>$ tuples for each decision made over the course of a game. The states were recorded in terms of the egocentric sensor values of the legion being moved, in order to be isomorphic with the inputs that the trained legions' controller network would receive during autonomous play (figure 6). Since the legions are coerced to share a common control policy, despite the necessity of adopting roles in their team adaptively, the training examples created for all the legions in a game were pooled into a common training corpus, regardless of the individual legions' roles at any given time.

Since the training examples record game states in the same form used by the controller network, and record actions as on of the discrete values obtained by decoding the network's outputs, the Lamarckian adaptation can be implemented with backpropagation (Rumelhart, Hinton, & Williams 1986). This adaptation is orthogonal to the use of ESP: since ESP uses direct encodings for the neurons in a network, the weight modifications resulting from the backpropagations are simply written back into the chromosomes that defined the current network. Such orthogonal "plug-and-play" mechanisms are likely to play an important role in further progress at applying computational intelligence methods to the challenges provided by the rich environments of videogames and computer simulations.

Experimental Evaluation

This section describes the experimental evaluation of neuroevolutionary policy induction in the *Legion II* game. The first subsection gives details of the experimental procedures, and the second describes the overall performance at policy induction and the types of behavior learned.

Procedures

A family of distinctive operational doctrines, or control policies, was defined to constrain the legions' behavior. The doctrines are parameterized according to "level", the maximum distance $d \geq 0$ that a garrison is allowed to move away from its city. Thus a garrison following doctrine L_0 will never leave its city, a garrison following doctrine L_1 will never move more than one map cell away from its city, and so on. In addition, the doctrines require a garrison to return to its city and remain there whenever there are no barbarians within radius d of that city.

The garrisons were also required to observe a safety condition without regard to the level of doctrine in use. The safety condition specifies that a garrison may not end its turn with a barbarian as close to the city as the garrison, or closer. The legion may not select a move that violates the condition, and if a violation arises due to a barbarian's move the garrison must immediately return to the city if it cannot eliminate that barbarian with its current move. A side effect of the safety condition is that whenever there is more than one barbarian adjacent to a city, the garrison cannot leave the city to eliminate one of them. Otherwise one of the other barbarians would be able to move into the city on its own turn, and get in a turn of pillaging it before the garrison is able to return.

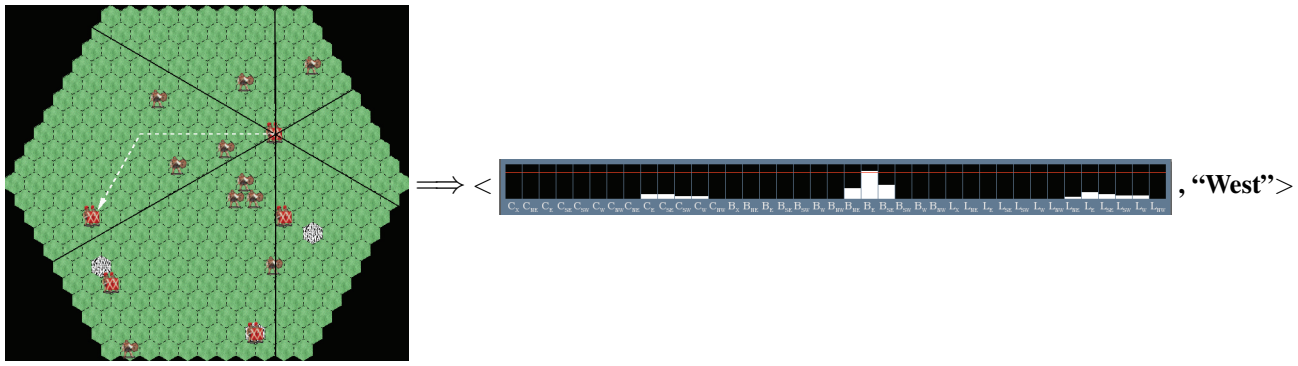

Figure 6: **Training examples for the *Legion II* game.** To train agents for the *Legion II* game, the captured <*state, action*> tuples consist of the legion's egocentric sensory perception of the game state and the human's choice of a discrete action for the legion. For example, if the human chooses to move the legion at the upper right to the west when the map is in the game state shown, the built-in observer records the 39-element pattern of the legion's sensor activations and the discrete symbolic choice "West".

A human played twelve example games using each doctrine, using different randomized starting positions for each game. Due to the low resolution of the legions' sensors at $d > 1$, only doctrines L_0 and L_1 were used in the experiments. The player controlled all five legions in each game, creating a total of 1,000 contextual example moves per 200-turn game. The examples generated by the twelfth game in each of the two sets were reserved for test data, leaving 11,000 training examples for each of the two doctrines studied.

Controller networks were trained for 5,000 generations with Lamarckian neuroevolution, using a single iteration of backpropagation on the training examples for the phenotypic adaptation. Adding backpropagation with a large number of training examples to every candidate solution's fitness evaluation can greatly increase the run time of a neuroevolutionary algorithm. Therefore the experiments used only a random subset of the 11,000 available examples each time a phenotype was trained. The general trend was that increasing the sample size increased the accuracy of the policy inductions, as well as the run times (Bryant 2006). Due to space limitations, only the results for using a sample size of 5,000 are described below.

For comparison, additional controller networks were trained using backpropagation only, using all 11,000 training examples for 20,000 epochs. The choice of 20,000 epochs was not intended to equalize some measure of the amount of training provided by the two methods; rather, the choice of 5,000 generations and 20,000 epochs put the learners far out on the "flat" of their asymptotic learning curves, to ensure that undertrained networks were not used for testing and comparison.

Results

The legions learned to perform well in the *Legion II* game in all the experimental runs. However, examining success at policy induction requires metrics for behavioral similarity rather than for success at the target task *per se*.

A high-level overview of the experimental results can be obtained by means of simple coarse-grained behavioral similarity metrics. One such metric is the game score. Autonomous agents operating under a specific doctrine (policy) can be expected to obtain game scores similar to those obtained by a human player using the same doctrine. Notice that since the goal is policy induction rather than task optimization, higher or lower game scores alone are not indicators of success; successful induction of a less-effective policy should produce worse scores than an equally successful induction of a more-effective policy.

A second coarse-grained behavioral similarity metric comes from treating the examples of play reserved from the twelfth example game for each doctrine as a classification problem. Autonomous agents operating under a specific doctrine (policy) can be expected to make local decisions similar to those made by a player using the same doctrine, and thus obtain a high success rate at guessing what the player did at any game state.

A phase space can be defined by using these two behavioral similarity metrics as the axes. The results of various methods for policy induction can then be compared visually by plotting their mean performance on the two metrics as (x, y) points in the phase space. The results for the current experiments are shown in figure 7. Unguided evolution (asterisk) produces similarity results somewhat distant from the human using either of the two doctrines, with scores intermediate between them and example classification failures running at about 50%. Example-guided neuroevolution (diamonds) pulls the results substantially toward the human-generated target behaviors along both axes.

The backpropagation-only results are shown for comparison. For both doctrines, example-guided evolution produced results nearer the targets than backpropagation did; slightly so for the L_0 experiments (grey), and substantially so for the L_1 experiments (black).

Surprisingly, the classification-based similarity metric show errors in the range of 15-20% regardless of the doctrine or learning method. That is because the "coarse similarity metric" really *is* coarse. Some of the player's decisions are arbitrary even when playing in compliance with one of the specified doctrines. For example, when moving toward a location that requires cutting across the hex-grid on the map at a diagonal, a zig-zag path must be used. In such cases

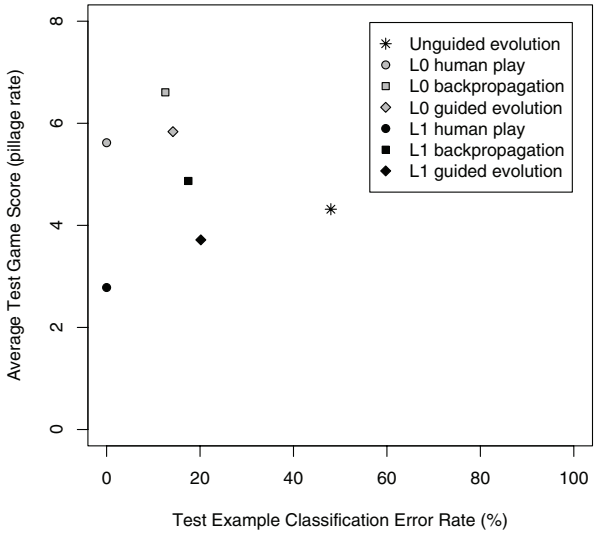

Figure 7: **Learning to imitate human behavior.** Mean game scores are plotted against the ability to imitate human decisions in a previously unseen game. Circles show the mean scores obtained by human play when using the L_0 (grey) and L_1 (black) operational doctrines. Since these scores were obtained while generating the target behaviors they have a classification error rate of 0%. The star shows the mean results obtained with unguided neuroevolution, relatively distant from both targets along both axes. Squares show the results obtained by backpropagation and diamonds the results obtained by Lamarckian neuroevolution. For both doctrines, the guided evolution provided results closer to the targets than backpropagation did.

the LRLRL... and RLRLR... alternations are equally legitimate, and since the choice is underspecified by the doctrines there is no reason to suppose that the human player preferred one over the other. And when the training examples are not consistent, the trained controller cannot guess what the human player actually did in any specific case that occurred in the test data.

To go beyond the high-level overview of the results provided by the phase plot, more precise behavioral metrics are required. The key features of the L_0 and L_1 doctrines are specified in terms of rules that can be formalized for measurement. During test games the rate of violations of the safety condition or of the maximum distance a garrison is allowed to travel from a city can be measured. The doctrines also have implications for the mean distance between the cities and their garrisons over the course of a game, and this can be measured as well.

Figure 8 shows the results for the rate of safety condition violations for legions trained for the two doctrines using unguided neuroevolution, guided neuroevolution, and backpropagation. The figure shows that both backpropagation and guided evolution provide better conformance to the safety condition than unguided evolution does, but for both doctrines the guided evolution results in controllers that enforce conformance faster at the start of a game. The conclusion is that guided evolution has learned the rules better. (Cf. figure 7, where the coarse behavioral similarity metric based on example classifications suggests that backpropagation performed slightly better.)

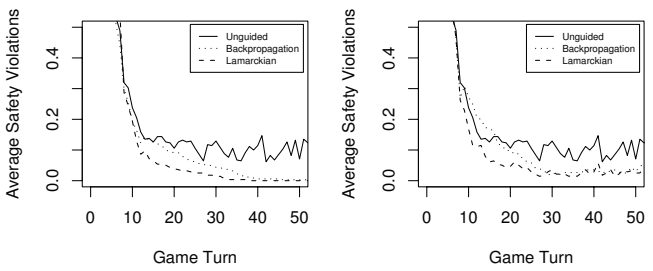

Figure 8: **Learning the rules underlying the human behavior.** *Left:* The average number of safety condition violations is plotted vs. game time for the L_0 doctrine. Safety violations are impossible when a garrison remains in its city, so successful learning of the L_0 doctrine drives the count to zero. (The initial violation rate is high due to the random placement of the legions.) The backpropagation-only solution produces a similar result, but the controllers it produces are slower to eliminate the violations. *Right:* The same information plotted for the L_1 doctrine. The violations are reduced to a very low rate. The backpropagation-only solution produces a similar result, but again with a slower response.

Similar results were obtained when measuring the average distances from garrisons to their cities and violation rates of the maximum allowed distance from city to garrison, except that guided evolution did not produce faster conformance than backpropagation in the latter case, for the L_0 doctrine only. Thus example-guided neuroevolution provided faster rule conformance than backpropagation did on five of the six rule induction tests, and faster + better conformance than unguided evolution on all six tests. The full set of plots are given by Bryant (Bryant 2006).

Discussion and Future Work

The experiments show that Lamarckian neuroevolution is a promising method for inducing visibly intelligent behavioral policies in autonomous intelligent agents. This result is seen both through coarse-grained measurements, e.g. by creating agents that score better or worse according to the dictates of the target policy, and through examining rule learning for the formal rules that were merely implicit in the examples used to guide the evolutionary process.

Future work includes the twin challenges of applying this method in environments that are continuous in space and time rather than discrete as in *Legion II*, and applying it to controllers that require memory. The continuous space and time generally require an egocentric controller to output motor control signals rather than a choice among discrete actions; training such controllers with neuroevolution is already commonplace, albeit without the Lamarckian mechanism (Lucas 2004; Stanley, Bryant, & Miikkulainen 2005). Memory can be provided by using recurrent networks; the choice of a simple recurrent network (Elman 1990) makes it straightforward to implement the Lamarckian mechanism used here with backpropagation through time (Werbos 1990). Work in these two areas is already in progress in the *NERO* game and the *LagoonCraft* naval simulator.

In order to evaluate this approach to policy induction in a wider variety of applications, more rigorous and precise metrics for success need to be developed. For example, the error classification metric used in the phase diagram (figure 7) suggests that backpropagation learns one aspect of the target behaviors slightly better than example-guided evolution does, but the rule-based metrics (figure 8) suggest the opposite. In order to understand that the latter is more meaningful, a substantial knowledge of the details of the application is necessary. Moreover, even though it is possible to quantify performance differences using rule-based metrics – e.g. by integrating the area between the curves for backpropagation and example-guided evolution in figure 8 – it is not clear how such quantities should be interpreted. If the area between the curves is, say, 5.23, *how much* better or worse is that, and in what way? It is possible that the usual quantitative measures of performance in AI and machine learning need to be replaced by more qualitative ones like those used for behavior analysis in psychology.

When the Lamarckian mechanism of adaptation was added to the ESP algorithm for neuroevolution, the other conventions of neuroevolution remained unchanged. In particular, game scores were still used to measure fitness. In some cases it may be advantageous to replace game scores with a fitness function that rewards at least some of the higher-level aspects of a target behavior, even if it is not feasible to reward all the details of the desired behavior in all contexts. For example, providing the maximum reward for game scores that are closest to the scores obtained during the human play rather than for the absolutely best game scores is a simple modification that may reduce a tendency for the optimizing effect of the evolutionary algorithm to pull the population away from the target behavior.

Additionally, the difference in point-of-view between the human player and the embedded agents needs to be taken into account, and eliminated to the extent that it is possible. When generating examples for these experiments a deliberate effort was made to avoid basing decisions on the "God's-eye view" that the graphical interface provided the player. However, it is likely that whenever moves were not fully constrained by the doctrines, the human player made intuitive or subconscious use of information not available to the trained agents. Such decisions can result in aliased sensory states with unaccountable – and thus unlearnable – differences in the human behavior. In order to make learning as effective as possible, the human view needs to be as similar as possible to the agents' views.

Even beyond such technical issues, policy induction in rich environments is a difficult challenge. Learning methods must be powerful enough, and examples of behavior copious enough, so that the correct inductions can be made, despite spurious features of the environment's state at the time an example decision was made.

On the other hand, policy induction is itself an intelligent capability. If an autonomous agent can use policy induction to create an accurate model of its opponents (or allies), it can then predict their actions in hypothetical environments, and use those predictions to modify its own behavior. Thus policy induction can lead to visibly intelligent behavior not only by allowing direct imitation of such behavior, but also by supporting the kind of *in situ* adaptive behavior expected of genuinely intelligent creatures.

Related Work

This work lies within a cluster of related applications that includes user modelling, opponent modelling, social robotics, and the use of examples simply to help machine learning on a difficult task. Space allows only the sparsest sampling of that work here.

Once a decision has been made to use examples for agent training and the examples have been collected, new learning options arise. A discrete-option controller like the one used to implement the legions' control policy can be seen as a simple pattern classifier. That opens up the possibility of bringing the full weight of the field of learning classifier systems to bear on the problem. For example, Sammut et al. use *behavioral cloning* to induce a decision tree for controlling an airplane in a flight simulator, by applying C4.5 to 90,000 <*state,action*> tuples extracted from traces of human control (Sammut *et al.* 1992; Bain & Sammut 1999; Quinlan 1993). The same task is addressed by van Lent and Laird using their rule-based *KnoMic* "Knowledge Mimic" system, which applies a modified version of the *Find-S* algorithm to a trace of <*state,action*> tuples (van Lent & Laird 2001). Both sets of authors found it necessary for a human expert to split the traces into segments according to changing goals over the course of a flight. It is not clear whether that requirement arose from an inherent difficulty of the problem or from weaknesses in the power of the methods used. However, since both operate on the same sort of data required by example-guided evolution, future comparisons will be straightforward.

Ng and Russell take an oblique route to the problem of instilling behavior. Their *inverse reinforcement learning* methods induce the reward function implicit in example behavior (Ng & Russell 2000). From the reward function ordinary reinforcement learning methods can then be used for agent training, including methods that do not readily incorporate examples directly. Discovering an implicit reward function may also prove useful as a method for analyzing the tacit motivations for examples of behavior.

An extended overview of methods for learning from demonstration in the robotics domain is given by Nicolescu, and additional methods and applications for policy induction are surveyed by Bryant (Nicolescu 2003; Bryant 2006).

Conclusions

In games and simulators it is not always sufficient for autonomous agents to perform optimally; they also need to display behavior that "looks right" to observers. The requirement for such visibly intelligent behavior varies with the application, but often includes agents that conform to some specific strategy, doctrine, or policy. With the increasingly rich state and action spaces in modern games, it has become difficult to specify such policies directly. Example-guided neuroevolution offers a possible alternative approach. It can induce behavioral policy from examples, learning rules that

are merely implicit in those examples. In the future it may be possible to induce rich patterns of behavior directly from the intuitively generated examples of game designers and simulation subject-matter experts, making games easier to develop and more satisfying to play, and allowing serious games and simulators to be more accurate and realistic.

Acknowledgments

This work was supported in part by the NSF under grants IIS-0083776 and EIA-0303609, and a fellowship from the Digital Media Collaboratory at the IC^2 Institute at the University of Texas at Austin. The images used in *Legion II*'s animated display are derived from graphics supplied with the game *Freeciv*, `http://www.freeciv.org/`.

References

[Bain & Sammut 1999] Bain, M., and Sammut, C. 1999. A framework for behavioural cloning. In Furukawa, K.; Michie, D.; and Muggleton, S., eds., *Machine Intelligence 15: Intelligent Agents*. Oxford University Press. chapter 6, 103–129.

[Bryant & Miikkulainen 2003] Bryant, B. D., and Miikkulainen, R. 2003. Neuroevolution for adaptive teams. In *Proceeedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, volume 3, 2194–2201. Piscataway, NJ: IEEE.

[Bryant & Miikkulainen 2006] Bryant, B. D., and Miikkulainen, R. 2006. Exploiting sensor symmetries in example-based training for intelligent agents. In Louis, S. J., and Kendall, G., eds., *Proceeedings of the 2006 IEEE Symposium on Computational Intelligence and Games (CIG'06)*, 90–97. Piscataway, NJ: IEEE.

[Bryant 2006] Bryant, B. D. 2006. *Evolving Visibly Intelligent Behavior for Embedded Game Agents*. Ph.D. Dissertation, Department of Computer Sciences, The University of Texas at Austin, Austin, TX.

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

[Gomez & Miikkulainen 1999] Gomez, F., and Miikkulainen, R. 1999. Solving non-Markovian control tasks with neuroevolution. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1356–1361. San Francisco, CA: Morgan Kaufmann.

[Gomez 2003] Gomez, F. 2003. *Robust Non-Linear Control Through Neuroevolution*. Ph.D. Dissertation, Department of Computer Sciences, The University of Texas at Austin.

[Grefenstette 1991] Grefenstette, J. J. 1991. Lamarckian learning in multi-agent environments. In Belew, R., and Booker, L., eds., *Proceedings of the Fourth International Conference on Genetic Algorithms*, 303–310. San Mateo, CA: Morgan Kaufman.

[Ku, Mak, & Siu 2000] Ku, K. W. C.; Mak, M. W.; and Siu, W. C. 2000. A study of the Lamarckian evolution of recurrent neural networks. *IEEE Transactions on Evolutionary Computation* 4:31–42.

[Lamarck 1809] Lamarck, J.-B. 1809. *Philosophie zoologique*.

[Lucas 2004] Lucas, S. M. 2004. Cellz: a simple dynamic game for testing evolutionary algorithms. In *Proceeedings of the 2004 Congress on Evolutionary Computation (CEC 2004)*, 1007–1014. Piscataway, NJ: IEEE.

[Miikkulainen *et al.* 2006] Miikkulainen, R.; Bryant, B. D.; Cornelius, R.; Karpov, I. V.; Stanley, K. O.; and Yong, C. H. 2006. Computational intelligence in games. In Yen, G. Y., and Fogel, D. B., eds., *Computational Intelligence: Principles and Practice*. Piscataway, NJ: IEEE Computational Intelligence Society.

[Ng & Russell 2000] Ng, A. Y., and Russell, S. 2000. Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, 663–670. San Francisco: Morgan Kaufmann.

[Nicolescu 2003] Nicolescu, M. N. 2003. *A Fremework for Learning From Demonstration, Generalization and Practiced in Human-Robot Domains*. Ph.D. Dissertation, University of Southern California.

[Quinlan 1993] Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. San Francisco, CA: Morgan Kaufmann.

[Rumelhart, Hinton, & Williams 1986] Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning internal representations by error propagation. In Rumelhart, D. E., and McClelland, J. L., eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*. Cambridge, MA: MIT Press. 318–362.

[Sammut *et al.* 1992] Sammut, C.; Hurst, S.; Kedzier, D.; and Michie, D. 1992. Learning to fly. In *Proceedings of the Ninth International Conference on Machine Learning*, 385–393. Aberdeen, UK: Morgan Kaufmann.

[Stanley, Bryant, & Miikkulainen 2005] Stanley, K. O.; Bryant, B. D.; and Miikkulainen, R. 2005. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation* 9(6):653–668.

[van Lent & Laird 2001] van Lent, M., and Laird, J. E. 2001. Learning procedural knowledge through observation. In *Proceedings of the International Conference on Knowledge Capture*, 179–186. New York: ACM.

[Werbos 1990] Werbos, P. J. 1990. Backpropagation through time: What it does and how to do it. In *Proceedings of the IEEE*, volume 78, 1550–1560. Piscataway, NJ: IEEE.

[Whitley *et al.* 1993] Whitley, D.; Dominic, S.; Das, R.; and Anderson, C. W. 1993. Genetic reinforcement learning for neurocontrol problems. *Machine Learning* 13:259–284.

[Whitley 1995] Whitley, D. 1995. Genetic algorithms and neural networks. In Periaux, J.; Galan, M.; and Cuesta, P., eds., *Genetic Algorithms in Engineering and Computer Science*. New York: Wiley. 203–216.

[Yao 1999] Yao, X. 1999. Evolving artificial neural networks. *Proceedings of the IEEE* 87(9):1423–1447.