Effects of Experience Bias When Seeding With Prior Results

Mitchell A. Potter, R. Paul Wiegand, H. Joseph Blumenthal and Donald A. Sofge
Naval Research Laboratory
Washington, DC 20375
{mpotter,wiegand,blumenth, sofge}@aic.nrl.navy.mil

Abstract- Seeding the population of an evolutionary algorithm with solutions from previous runs has proved to be useful when learning control strategies for agents operating in a complex, changing environment. It has generally been assumed that initializing a learning algorithm with previously learned solutions will be helpful if the new problem is similar to the old. We will show that this assumption sometimes does not hold for many reasonable similarity metrics. Using a more traditional machine learning perspective, we explain why seeding is sometimes not helpful by looking at the learning-experience bias produced by the previously evolved solutions.

1 Introduction

Learning tasks for agents in complex, changing environments can be quite challenging. Indeed, the real-time adaptive requirements of agents such as autonomous robots operating in the physical world often make learning extremely difficult. To deal with this, several contemporary learning systems employ methods that use prior knowledge when learning behaviors in new, but similar tasks. This is often done by *seeding* the learning algorithm with prior knowledge. For evolutionary algorithms, previously evolved solutions are injected into initial populations, competing to survive in the changed world.

One proposed framework for dealing with situations requiring adaptive learning methods is anytime learning (Grefenstette and Ramsey 1992), more recently referred to as continuous and embedded learning (CEL) (Schultz and Grefenstette 2000). CEL involves monitoring the environment for changes, updating a simulated world with the detected changes, then learning new behaviors in simulation to cope with the new circumstances before posting the adapted behaviors to the physical robot. CEL relies on the ability of the underlying learning algorithm to use prior knowledge to assist in learning new situations. One hopes similar situations will require similar solutions, and that biasing the algorithm in this way will improve the learning time.

This seeding technique is not unique to CEL. It is also used when employing methods like shaping—where gradually more challenging problems are presented to a learning algorithm so that an explicit gradient is established and complex tasks can be learned in stages (Dorigo and Colombetti 1998). Again, here seeding carries with it the hope that a similarity bias will improve learning performance on complex tasks.

Despite the many uses of seeding, it should be clear that such methods will not always be helpful. When situations are sufficiently different, it can easily be the case that a bias toward prior behaviors will trap algorithms in new suboptima (Louis and Johnson 1997). Unfortunately, it is unclear what "sufficiently different" means for a given task; yet for those who employ methods like CEL and shaping, it is nevertheless important to understand when seeding is helpful, and when it can be harmful.

One way to answer such a question is to define a variety of useful distance metrics and study how such measures affect the performance of seeding methods. Rather than having to rely on arbitrary notions of distance, though, we adopt a different perspective in which we concentrate on how experiences in the environment can help or hinder the establishment of an appropriate learning gradient. Complex behavioral tasks often require different sets of behaviors (skills) for different sets of experiences resulting when problem characteristics differ, what one might call different *aspects* of the problem. In some sense, these aspects correspond to different objectives; however, the degree to which they are relevant changes depending on the circumstances of the problem itself. The ability to learn skills that address these aspects, and the ability to balance them appropriately in a given setting, relies on the capability of the learning algorithm to gain sufficient *experience* with them. Our view is that exposure to proper experiences, appropriately weighted, is important to establish a learning gradient for the larger problem.

In this paper, we examine the utility of seeding a learning algorithm using prior knowledge in terms of the algorithm's ability to collect sufficient experiences. We show that seeding is helpful when it maximizes experiences of important elements of the new problem; however, when prior learning runs generate behaviors that prevent the algorithm from collecting appropriate experiences in the new environment, seeding with those behaviors may not help improve learning performance on the new problem. In fact, seeding may be considered harmful in the latter case because it impedes the algorithm from learning new behaviors that are potentially more appropriate for the new environment. The result of this work is a pragmatic, more traditional machine learning view of seeding, in that seeding is a producer of a bias that may or may not be helpful.

The next section describes some of the related work with techniques involving the incorporation of prior knowledge to bias current learning. Section 3 describes a covert tracking task, the various objectives and behaviors required to accomplish the task, and how they can be objectively measured. Section 4 includes details regarding our robot control architecture, our evolutionary algorithm, and our seeding mechanism. Section 5 includes experimental results that illustrate the effect seeding with prior solutions has on the tracker's exposure to relevant experiences, which in turn affects the ability of the algorithm to learn certain behaviors. We conclude in section 6 with a discussion about how these findings relate to our work, and what future steps to take.

2 Related Work

In the discipline of machine learning, researchers strive to find a solution that is both accurate and computationally inexpensive; however, there is often a tradeoff between these two characteristics. One way to reduce training time is to incorporate domain specific knowledge into the search, though frequently little or no *a priori* knowledge is available. In the absence of domain specific knowledge, a common approach is to randomly choose starting points in the search space.

In a standard evolutionary algorithm (EA), for example, the population of candidate solutions are randomly initialized, and this strategy suffers from the risk of inadequately covering the search space and biasing the search toward suboptimal solutions. In order to circumvent this risk, there have been a number of proposed methods that select a set of points in the search space as evenly distributed as possible. Morrison (2003) proposed a solution to this common problem in evolutionary algorithms named the heuristic sentinel placement algorithm. Morrison's initialization uses heuristics, guided by a discrepancy measure, to generate a sequence of well distributed points in the search space. Of course, techniques for evenly covering the search space tend to be representation dependent. For example, a common initialization algorithm for a two layer neural network is the Nguyen-Widrow algorithm (Nguyen and Widrow 1990). This approach generates random values for weights and biases for all layers of a neural network and then adjusts these values such that the input space is parsed into different sections covered by a particular node.

When knowledge of the problem domain is available, it can be used to bias the search strategy and reduce the computation time required to find a quality solution. An early example of a mechanism that seeds an EA is Eschelman's CHC partial restart (Eshelman 1991). A restart occurs when the algorithm determines that a population has stagnated or converged. Eschelman's system preserves an identical copy of the best individual and reinitializes the rest of the population with individuals that are highly mutated copies of the best. While restarting due to stagnation or convergence has proved to be useful, our focus here is on a quite different situation in which the algorithm is restarted specifically to deal with a change in the environment.

Of particular interest is learning robot controllers. Robots typically operate in unstructured, uncertain, and changing environments where it is critical to find quality solutions quickly. In particular, if a robot is slow to adapt to changes in its environment, it may suffer any number of unfortunate consequences, including physical harm. Yet evaluating candidate solutions can be extremely time consuming when learning is done directly on the robot or in a high-fidelity simulation. A possible solution is to bias a search by including domain knowledge previously learned from similar problems. Some consider case-based reasoning to be the first application of this principle (Riesbeck and Schank 1989).

An early evolutionary computation system for learning rule-based robot controllers in a dynamic environment is continuous and embedded learning (CEL)—originally called anytime learning (Grefenstette and Ramsey 1992). The CEL architecture includes an execution system that maintains an active controller for a robot operating in a real-world environment, a learning system that includes a simulation model of the environment and is capable of producing new controllers, and a monitor that detects when significant changes in the real-world environment occur. When such a change occurs, it triggers an update to the simulation model to make it more closely match the real world, and the learning system initiates a restart and performs a round of evolutionary adaptation to this new environment. The learning system seeds a predefined percentage of its population with the best previously-learned controllers, while the remaining members of the population are initialized randomly. Later versions of CEL included a case-base of previously-evolved controllers for this purpose (Ramsey and Grefenstette 1993). As better control systems are evolved, they are transferred to the execution system for use by the robot in the real-world environment.

Louis and Johnson (1997) developed a system quite similar to continuous and embedded learning called Case Injected Genetic Algorithms (CIGAR), which employs case-based memory and genetic algorithms to reuse previously discovered information from a similar problem to bias the search of an unseen space. When CIGAR is faced with a new problem it searches the case-base containing solutions to previous problems to find whole or partial solutions to a similar problem that can be used to seed a genetic algorithm. In these experiments, it was shown that this method decreases the learning time to solve the problems. CIGAR also uses a boosting technique independent of problem similarity that injects previously evolved cases most closely resembling the best members of the currently evolving population.

Floreano and Mondada (1996) seeded an EA to adapt a neural network based robot controller to changes in the environment. Neural networks provide a compact, efficient, and highly flexible representation for robotic controllers. The neural network receives sensory input from the robot's sensors (as well as possible state information), and produces the appropriate control commands as its outputs. Floreano and Mondada used a small Khepera robot whose task was to navigate a square area for as long as its battery life would sustain. The robot was equipped with a battery sensor, and a battery recharge corner of the square was painted black and illuminated by lights. Once the Khepera could successfully navigate the area and recharge its battery, the lights were moved to the opposite corner of the square area from the black painted recharging station. After this environmental change, Floreano and Mondada did not restart the learning process, they simply continued the genetic algorithm with the final population. This strategy yielded a highly fit solution in fewer evaluations than reinitializing the genetic algorithm randomly.

Dorigo and Colombetti (1998) address a related problem in getting a robotic system to learn to perform a complex task based upon interaction with an external trainer (e.g., a human operator or reinforcement program). This approach focuses heavily upon the use of reinforcement learning techniques, and relies upon the trainer to provide appropriate fitness functions in order to shape the behaviors of the robotic system. The term *shaping* is borrowed from experimental psychology (Skinner 1938) and is based upon the notion that complex behaviors can be decomposed into simpler parts. The parts are learned separately and then integrated together. One of the arguments implicit to this approach is that complex behavior is learned more easily through decomposition and learning of simpler behaviors.

Other researchers have adapted the robot shaping paradigm to use other forms of machine learning. A specific type of shaping dubbed *incremental evolution* is used to shape behaviors by manipulating the complexity of the task and the fitness function. An interesting application of this method at the University of Sussex involved a robot with a camera that is taught incrementally to distinguish between a rectangle and a triangle (Harvey et al. 1997). Learning begins with the robot given the simple task of forward motion, then moving toward targets both big and small, and finally learning to approach a triangle instead of a square.

There exists a common underlying assumption in most the previous work described here on initializing a learning algorithm with previously learned solutions. The assumption is that seeding the learning algorithm with prior results will be helpful if the new problem is similar to the old. We will show that this assumption sometimes does not hold for many reasonable similarity metrics, and explain this seeding failure by looking at the learning-experience bias produced by the prior results.

3 Covert Tracking

Many seemingly simple multiagent problem domains contain surprising complexity, often requiring agents to adapt behaviors to suit even modest changes in the problem characteristics. Consider, for example, a covert tracking problem. Here there are two agents, a target and a tracker. In our case, the behavior of the target is fixed, while we are attempting to learn behaviors for the tracker; however, the target's vision capabilities may vary.

The target moves around the environment, perhaps performing various tasks, but will react to any unknown agent it sees. Such a reaction may be to attack the tracker in some way, or perhaps to run away from the tracker, etc. In testing and training runs of our system, we assume that if the target sees the tracker it will inflict some form of damage that does not necessarily impair the tracker physically, but nonetheless is undesirable.

The tracker's task is to track the target as closely as possible while staying out of the target's field of view, as shown in Figure 1. The tracker will be rewarded the closer it is to the target; however, if seen then it will be penalized. Moreover, since the field of view of the target may be different at different times, the tracker may need to alter its basic behavior to adapt to changed target capabilities.

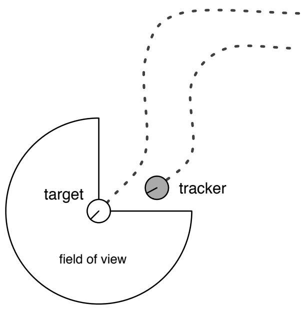

Figure 1: The task is to learn to track the target as closely as possible while not being seen.

Though the basic task appears straightforward, a closer inspection will reveal that it may require a variety of skills on the part of the tracker, depending on the vision capabilities of the target. We have identified at least three different skills observed to be helpful in high performance tracker solutions, depending upon the target's capabilities:

following: The tracker attempts to get as close as possible to the target.

avoiding: The tracker attempts to remain outside the target's range of vision.

hiding: The tracker attempts to remain outside the target's angle of view.

These three skills reflect different means of addressing the two competing objectives of tracking and not being seen. The *follow* behavior concentrates exclusively on the first objective, while the *avoid* and *hide* behaviors concentrate on different means of achieving the second objective.

Obviously these skills are interrelated, but reflect very different priorities of the overall, compound behavior. Moreover, they vary in importance in different contexts. For example, if the angle of view of the target is $360°$ then there is no need for the *hiding* skill, while the *avoid* skill is of particular importance. When the target's angle of view is $180°$, *avoiding* is less important, and *hiding* becomes a valuable skill for good performance.

It is important to understand that, while it may be reasonable to expect these skills to be important parts of the behaviors of the final solutions, they are observed phenomena. Since we do not wish to *engineer* particular solutions, they will not be used directly during learning. Instead, learning performance will concentrate on the larger task objectives: Track the target as closely as possible without being seen. These two objectives are captured by the simple minimization function:

$$f(\text{tracker}) = \sum_{i=1}^{s} \begin{cases} 3v & \text{if tracker seen} \\ r_i & \text{otherwise,} \end{cases} \qquad (1)$$

where s represents the number of steps in a simulation, v is the vision range of the target, and r_i is the range of the target from the tracker at step i. We developed separate, external measures for the three skills described above, but these measures will not be used by the EA.

3.1 Experiences and Skills

Learning the three covert tracking skills clearly requires different kinds of experiences during training. These different sets of experiences help define different aspects of the problem that relate in various ways to the two underlying objectives of the compound problem. Different problem aspects will determine the degree to which the tracker is capable of learning skills addressing those aspects. In some situations certain experiences will be rare, and the learning algorithm is unlikely to have a sufficient gradient to learn skills addressing a particular problem aspect.

Figure 2 helps make it clear what circumstances provide the different kinds of experiences for aspects of the problem relating the three aforementioned skills. The top segment of the figure shows that tracker positions closer to the target receive higher quality *following* experiences; the left segment shows how the tracker receives positive *avoiding* experiences by staying outside the target's vision range; and the right segment illustrates that positive *hiding* experiences are found in the target's blind-spot, regardless of distance.

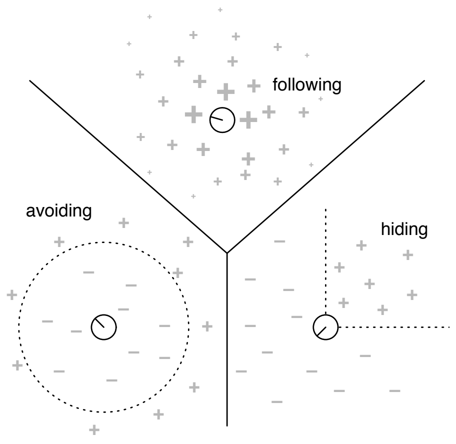

Figure 2: The tracking agent receives different types of experiences for different skills, based on its position relative to the target.

As we will see, when circumstances prevent the agent from obtaining different kinds of experiences, learning skills that distinguish such experiences becomes difficult or impossible. In some sense, this is a traditional machine learning perspective: In order to learn a concept, the learning algorithm must have appropriate examples necessary to establish a gradient.

3.2 Measuring Agent Skills

We could visually examine behaviors learned in different circumstances to see how well the tracker was able to learn to solve different aspects of the task. But to get a more objective picture of what aspects of the task are addressed by various solutions, it is necessary to quantify the performance of the tracker for each of the three covert tracking skills. Once quantified, we can then use these measures to examine how different environmental parameters affect the different kinds of tracking behaviors learned by the system.

To compute these measures for a particular solution, we run it in simulation many times. In the case of the *following* behavior, we compute a simple linear distance from the tracker to the target, averaged across all time steps. The lower the number, the more the tracker is *following* the target. Keep in mind that orientation to the target is irrelevant. Staying two meters in front of the target is just as good (or bad) as staying two meters behind it. For the *avoid* behavior, we simply compute a ratio of the number of time steps the tracker is within the vision range of the target out of all time steps, regardless of the target's angle of view. The final behavior, *hiding*, is slightly more complicated. For this we compute the absolute value of the relative angle of orientation between the tracker and the target in each step, centered directly behind the target. In other words, a tracker that is exactly behind the target receives a hiding score for that step of 0, but one that is directly in front receives a score of π. The final measure is the average of such angles over all the steps in a simulation. So, again, lower is better. We should again emphasize that these measure are used for post-analysis only. The learning process is driven only by the objective function in equation 1.

4 Learning Methodology

4.1 Control Architecture

The tracker is controlled by a combination of motor schema (Arkin 1989) and a two-layer feed-forward neural network. The neural network takes the range, bearing, and heading of the target as inputs and produces the range and bearing of a goal point at the rate of 10 Hz. A linear attraction vector is computed from the goal point, and summed with repulsive vectors for any sensed obstacles and a small amount of random noise to produce a control vector that sets the forward speed and turning rate of the tracker. The neural network is the learnable component of the tracker's control system. Networks with different connections weights will produce trackers with different high-level behaviors. In contrast, the target is controlled only by motor schema that are hand-coded to produced a smooth random walk. These actions are performed in simulation using TeamBots (Balch 1998).

The architecture of the tracker's neural network is shown in Figure 3. Five hidden and two output nodes are implemented using a sigmoid activation function that produces an output in the range $(-0.5, 0.5)$. Three real-valued inputs have weighted connections to all the hidden nodes, and each hidden node has a weighted connection to both of the output nodes. In addition, in order to provide a learnable bias, the hidden and output nodes have a weighted connection from an input clamped to the value 1.0. This topology produces a network with a total of 32 weighted connections. The network output representing the range of the goal point is converted to a value between 0.0 and 20.0 meters, and the output representing the goal bearing is converted to a value in the range $(-\pi, \pi)$.

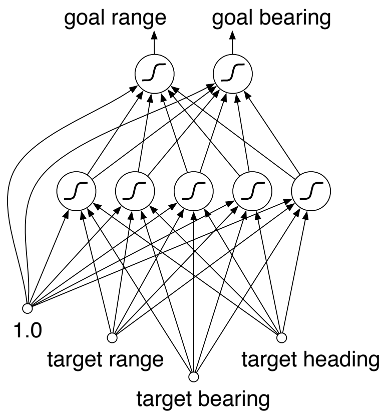

Figure 3: Neural network with learnable connection weights for producing high-level tracker behaviors

4.2 Evolving Behaviors

Although the tracker's neural network topology is fixed, we learn each of the 32 real-valued connection weights with a $(\mu + \lambda)$ evolution strategy (ES) as described by Bäck and Schwefel (1993). Specifically, we use an $\mathrm{ES}(10+70)$. That is, we begin with 10 parents, create 70 children by selecting parents uniformly and mutating them, evaluate each of the individuals in the combined population of 80 parents and children, and apply truncation selection to choose the best 10 individuals for the next generation.

In addition to a real-valued vector of connection weights, each individual consists of a companion vector of standard deviations used by a Gaussian mutation operator that is applied to each connection weight. The weights are initialized randomly in the range $(-5.0, 5.0)$ unless they are initialized using prior results as described in the next section. Regardless, they are constrained to the range $(-10.0, 10.0)$. The standard deviations are initialized to 1.0, and are themselves adapted within the range $(0.01, 1.0)$. Mutation is the only operator used by our EA.

To evaluate an individual, we construct a neural network from its connection-weight vector and run 25 four-minute TeamBots simulations of the network-controlled agent tracking the target. In each of the 25 runs, the tracker will be given a different random starting position 10 meters from the target. Given a simulation resolution of 10 Hz, the minimization function described in equation 1 will be summed over 2400 steps, and will be averaged over the 25 runs to produce the final evaluation value used by the EA.

4.3 Seeding the EA

In the next section we will describe experiments in which the population of neural networks are initialized randomly, and other experiments in which the networks are initialized from previously evolved solutions. When initializing a population from a prior run of the EA, we take the best individual from its final generation and reset its standard deviation values to 1.0. A single copy of this individual is then inserted into the new population, while the remaining individuals are initialized randomly.

By resetting the standard deviation values of the seed individual, we encourage the mutation operator to explore a larger region of the space around the previous best solution. However, given that our EA is an $\mathrm{ES}(\mu + \lambda)$, the original unmodified seed individual will continue to be copied into future generations for as long as it is not superseded by a better solution. Therefore, the algorithm may continue to exploit this prior knowledge for many generations.

5 Experimental Results

At the most basic level, the question of whether or not it is advisable to seed a current learning situation with prior knowledge seems straightforward: If the older situation is like the current one then presumably a learning algorithm should be able to leverage useful elements of the behavior in order to assist in learning the new situation. Unfortunately, finding a *measure* for similarity that is commensurate with such learning properties is difficult in general, and the most obvious similarity metrics, such as some kind of parameter-space distance, may not provide this facility. In the case of the covert tracking problem, for example, attempting to learn behaviors when the angle of view is $270°$ turns out to be easy when the $180°$ case is used as a seed, but very difficult when the $360°$ case is used, as we will show in the next section.

To try to understand why this is the case, it is necessary to better understand the relationship between problem characteristics and learning experiences. In order to do this, we examine the skills learned in five separate experimental groups, each corresponding to a covert tracking learning problem with the target having a vision range of 5 meters, but a different angle of view ($0°$, $90°$, $180°$, $270°$, and $360°$). Each group was evolved for 100 generations in 50 independent trials, then the best control system from each group was considered for external measurement. Figure 4 below shows the relative performance for each group and measure. The points are mean values of 100 sample test runs, the wings represent the 95% confidence intervals for each group. Recall that lower values are better.

Pair-wise t-tests (95% confidence) with Bonferoni adjustment shows that, in the case of the *following* measure, statistically significant differences are maintained between all groups except the $0°$ and $90°$ cases. In the case of *avoiding*, the $0°$ and $90°$ cases are not statistically different, nor are $90°$ and $180°$, but all other groups are different from one another. With *hiding*, only the $180°$ and $270°$ cannot be statistically distinguished from each other; the others are different.

These results show the effects that problem characteristics have on the ability of an agent to learn to address certain problem aspects. Situations in which the target's angle of view is very limited present no (or few) experiences for learning to *avoid* the target—as a result, the tracker *does not* learn that behavior. By the same token, when there is no

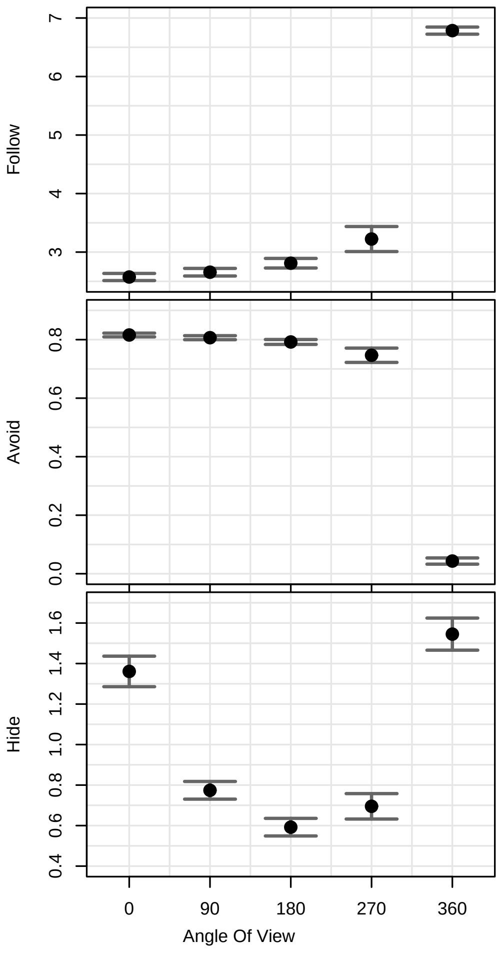

Figure 4: External measures of final learned behaviors under different angles of view. The measures illustrate the performance of the tracker as it *follows* the target, *avoids* the target, and *hides* from the target on average per time step. Lower values are better.

directional bias in the target's vision system, as is the case in the $0°$ and $360°$ groups, the tracker cannot learn to *hide* particularly well. Moreover, this graph helps illustrate the important point that parameters of the system form tradeoffs in experience sets available to the agent during learning, and thus there are natural tradeoffs in skill sets that occur as a result of such changes; in one situation it is better to learn skills A and B, while in another it is better to learn C.

Just as the problem characteristics can prevent the agent from learning skills because they preclude certain experiences, the skills themselves (once learned) restrict experiences. Behaviors that have learned to *avoid* the vision range of the target, for example, will incur few (or no) high-performance *following*-while-*hiding* experiences in the future. This being the case, it seems clear that a simple distance measure between parameter values makes an insufficient similarity metric for the purposes of determining whether or not a prior behavior makes a useful seed.

The fact that the tracker learns these skills better in some circumstances than others is no cause for alarm. There is nothing inherently wrong with this since any particular skill may be unnecessary in the context in which the tracking behavior was learned—the problem aspect may be unimportant in that circumstance. The trouble comes when we introduce an *a priori* behavior into the learning process, a behavior that will influence how the agent gathers new information.

Consider the case in which we attempt to improve the learning performance on the $270°$ angle of view problem by seeding with similar *a priori* cases. Here, the $180°$ case makes an appealing seed for a variety of reasons. First, from a parameter-space point of view, it is no more distant from the new problem than is the $360°$ case. Additionally, in our previous measures, our learning algorithm was able to evolve very good *hiding* behaviors for the $180°$ case. Still, given only distance as a rule of thumb, it is reasonable to assume that both $180°$ and $360°$ will make good seeds for the $270°$ problem. As it turns out, and has already been mentioned, using the $360°$ case as a seed does not help. Moreover, even though it is at the opposite end of the parameter space, seeding with the $0°$ case is of more value for learning the $270°$ context than the $360°$ seed.

Figure 5 illustrates these situations. The left-hand panel of the plot shows the randomly initialized evolution of the best behavior for each of the $0°$, $180°$, and $360°$ covert tracking problems, while the right shows the average best-of-generation learning performance on the $270°$ problem after seeding with the afore mentioned case. For comparison purposes, the average learning curve for the group evolving $270°$ from random initialization is also shown on the right-hand side. Statistically, the final results of the group where $270°$ is learned from random initialization is indistinguishable from that of the $360°$ seeded case; all other final performances differ significantly.

The skill sets learned in the *a priori* tracking behaviors alter the tracker's exposure to potentially necessary experiences. The $180°$ case, for example, provides a skill set that allows for most of the necessary experiences for learning high performance behaviors in the new $270°$ context; the tracker has access to positive and negative *hiding* experiences, for example. In the reverse situation, this is not the case. The $360°$ behaviors *avoid* the target entirely, and there are few experiences to learn to *hide*; the algorithm will have to rely on mutation to produce tracking behaviors that allow the agent to be exposed to the required experiences, while still performing sufficiently to survive selection. Since such a mutation is unlikely at this point, the $360°$ seed stalls out with very little improvement on the $270°$ problem, while the $180°$ makes steady improvement. In fact, when the $360°$ group was used to seed the learning algorithm to solve the $270°$ problem, only four of the 50 trials resulted in behaviors that show any significant degree of hiding.

The $0°$ seed is also a very interesting case. As mentioned above, because there is no directional information in the covert tracking problem when the angle of view of the target is $0°$ (like the $360°$ case), that case does not learn

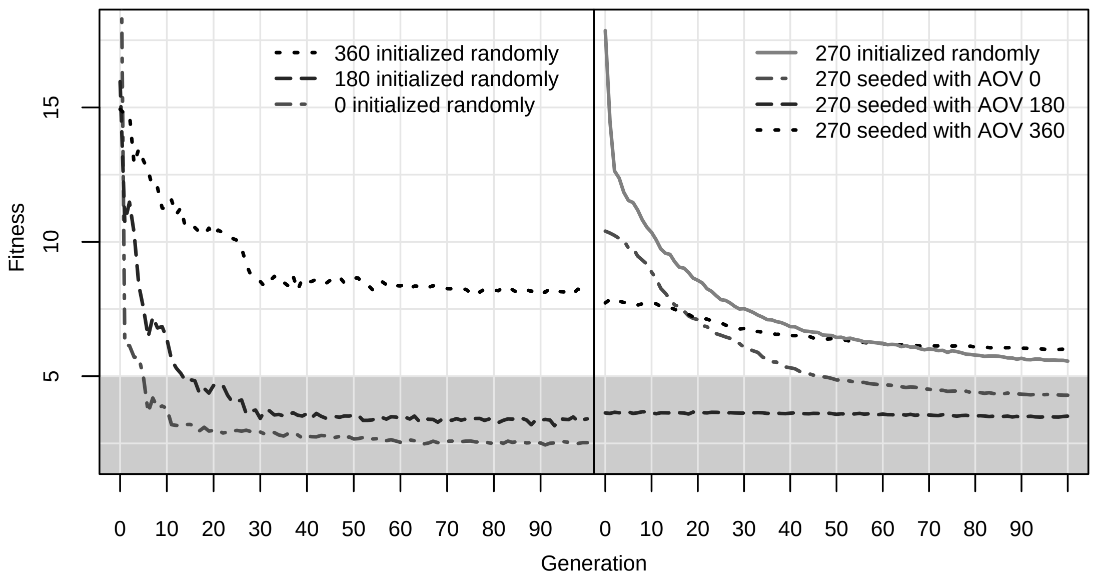

Figure 5: Using prior results from 0°, 180° and 360° angle of view (AOV) environments to seed initial population for learning in 270° angle of view environment. Grey zone represents solutions within vision range of target.

to *hide*. As such, the performance of the case on the new problem begins relatively poorly; however, because the 0° seed has access to better *hiding* and *following* experiences, it quickly overtakes the 360° seed, as well as random initialization for the 270° problem.

A more precise picture for these effects can be seen in Figure 6, where external measures are shown for the final resulting behaviors for some of the aforementioned groups. Here we average the measure over the resulting behaviors from the 50 independent runs of each experimental group. We consider the situation where the 0° and 360° cases are used to seed the 270° covert tracking problem. Additionally, we include measures for the randomly initialized 0°, 270°, and 360° cases for comparison purposes. For both *follow* and *avoid*, all groups are statistically different. In the case of the *hide* measure, the two right-most groups (360° seeding 270°, and 360° randomly initialized) do not differ, nor do the cases where 0° seeds 270°, and 270° itself. Otherwise, all the groups are different.

There are several items of note in this graph. First, though Figure 5 suggests that the 360° seed performs as well as the randomly initialized case, here we see that no new skills are learned. This bolsters our observation that we can expect very little from this seed. The 0° case, in addition to performing statistically better than the randomly initialized group, *does* learn something new; it learns to *hide*. The reason for this is quite clear: The 0° behavior allows the agent to gather relevant experiences in the 270° context, while the 360° does not.

6 Conclusions

While learning behaviors for agents differs from more traditional machine learning tasks such as concept learning, there are still many important similarities. In both cases, learning algorithms typically need to be exposed to appropriate experiences in order to learn to distinguish different concepts. When relevant experiences are missing, or weighted inappropriately, their related concepts will not be learned.

In many multiagent settings the problem domain can change, and in such cases we are tempted to leverage prior learning results to make new problems easier to solve. This paper begins to explore the question of when this so-called *seeding* is advisable by examining a particular class of problems (covert tracking) from a machine learning perspective: Changes in problem characteristics affect the algorithm's exposure to different kinds of experiences, which in turn affects the potential success of seeding. Our conclusion is that when prior learning creates behaviors that reduce or eliminate necessary experience in the new context, seeding will not help. While we concentrated on a particular learning algorithm, seeding mechanism, and problem class, we believe that this conclusion is fairly general.

Our examination was promulgated by our needs: Our lab conducts research in evolutionary robotics, and continuous and embedded learning is one of the tools we use. However, it is clear to us that the results are helpful in many contexts, such as shaping. Indeed, another study currently underway relates to the order in which learning cases should presented to an algorithm employing shaping. Using the same perspective we've shown here, we hypothesize that one should

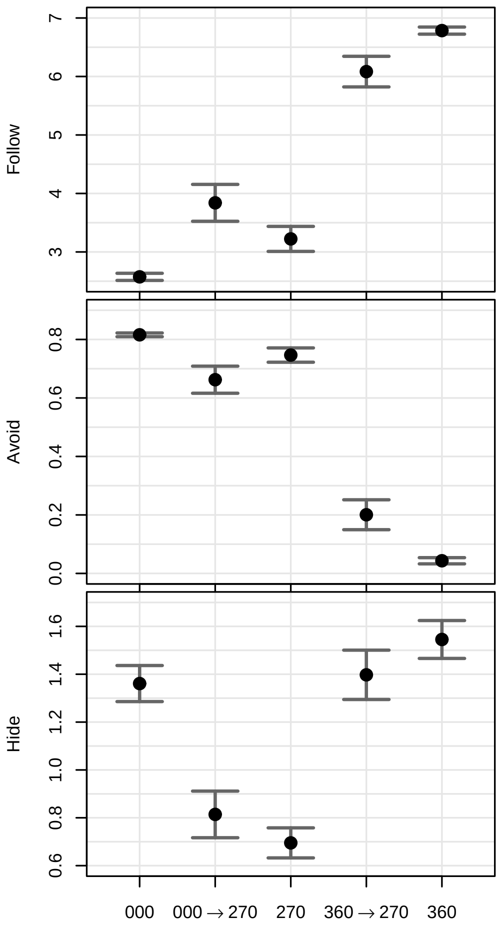

Figure 6: External measures of final learned behaviors under various initialization conditions.

first learn sub-problems that constitute rare but vital experiences in the global problem. In a tracking and docking task, for example, one should learn docking first, then the complete problem.

Next we turn our attention to the use of CEL as a means of co-adaptively learning behaviors for cooperative multi-agent teams. To do this, it will be important to understand what types of prior-learned behaviors will be useful for different team configurations and environmental parameters when configurations or parameters change. We believe this work is a step toward answering such questions.

Acknowledgments

This work was performed under ONR Work Orders N0001405WX20057 and N0001405WX30001.

Bibliography

Arkin, R. C. (1989). Motor schema-based mobile robot navigation. *The International Journal of Robotics Research 8*(4), 92–112.

Bäck, T. and H.-P. Schwefel (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation 1*(1), 1–23.

Balch, T. (1998). Integrating robotics research with javabots. In *Working Notes of the AAAI-98 Spring Symposium, Stanford, CA.*

Dorigo, M. and M. Colombetti (1998). *Robot Shaping: An Experiment in Behavior Engineering*. Intelligent Robotics and Autonomous Agents series, vol. 2. MIT Press.

Eshelman, L. (1991). The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of Genetic Algorithms I*, pp. 265–283. Morgan Kaufmann.

Floreano, D. and F. Mondada (1996). Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man and Cybernetics 26*(3), 396–407.

Grefenstette, J. J. and C. L. Ramsey (1992). An approach to anytime learning. In *Proceedings of the Ninth International Conference on Machine Learning*, pp. 189–195. Morgan Kaufmann.

Harvey, I., P. Husbands, D. Cliff, A. Thompson, and N. Jakobi (1997). Evolutionary robotics: the sussex approach. *Robotics and Autonomous Systems 20*, 205–224.

Louis, S. J. and J. Johnson (1997). Solving similar problems using genetic algorithms and case-based memory. In *Proceedings of the International Conference on Genetic Algorithms*, pp. 283–290. Morgan Kauffman.

Morrison, R. W. (2003). Dispersion based population initialization. In *Proceedings from the 2003 Genetic and Evolutionary Computation Conference*, pp. 1210–1221.

Nguyen, D. and B. Widrow (1990). Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *Proceedings of the International Joint Conference of Neural Networks*, Volume 3, pp. 21–26. IEEE.

Ramsey, C. and J. Grefenstette (1993). Case-based initialization of genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, California, pp. 84–91. Morgan Kauffman.

Riesbeck, C. K. and R. C. Schank (1989). *Inside Case-Based Reasoning*. Cambridge, MA: Lawrence Erlbaum Associates.

Schultz, A. C. and J. J. Grefenstette (2000). Continuous and embedded learning in autonomous vehicles: Adapting to sensor failures. In *Unmanned Ground Vehicle Technology II: Proceedings of SPIE*, Volume 4024, pp. 55–62.

Skinner, B. F. (1938). *The Behavior of Organisms: An Experimental Analysis*. D. Appleton-Century, New York.