An Evolutionary Approach to Learning in Robots

John Grefenstette and Alan Schultz
Navy Center for Artificial Intelligence
Naval Research Laboratory
Washington, DC 20375
{gref, schultz}@aic.nrl.navy.mil

Abstract

Evolutionary learning methods have been found to be useful in several areas in the development of intelligent robots. In the approach described here, evolutionary algorithms are used to explore alternative robot behaviors within a simulation model as a way of reducing the overall knowledge engineering effort. This paper presents some initial results of applying the SAMUEL genetic learning system to a collision avoidance and navigation task for mobile robots.

1 INTRODUCTION

This is a progress report on our efforts to design intelligent robots for complex environments. The sort of applications we have in mind include sentry robots, autonomous delivery vehicles, undersea surveillance vehicles, and automated warehouse robots. In particular, we are investigating issues relating to machine learning, using multiple mobile robots to perform tasks such as playing hide-and-seek, tag, or competing to find hidden objects.

Given the wide range of tasks in the area of robotics and learning, it may be helpful to briefly describe the flavor of our research. We begin by stating the orientation and motivation for our work, the kinds of tasks we chose to focus on, and our assumptions about what background knowledge is available to the learning system. While we make no claims that this research focus is the only reasonable approach, we are confident that this approach will provide a significant contribution to the development of robust, autonomous robots.

1.1 RESEARCH ORIENTATION

At one extreme, traditional robotics research assumes an engineering approach in which all aspects of the robot's behavior are pre-programmed. This approach suffers from the requirement of intensive human analysis of the robot and its environment, and is primarily useful in completely controlled and static task environments. At another extreme, researchers interested in Artificial Life seek to develop intelligent robots with minimal dependence on models provided by human designers. This is a very interesting approach with great promise for the long term. We take a middle ground in our research. From a practical perspective, machine learning can be justified by reducing the overall development cost for intelligent systems. This suggests an approach that treats knowledge acquisition for an intelligent autonomous system as a cooperative effort between the human knowledge engineer and the robot itself. We seek machine learning approaches that shift more of the burden to the robot and thereby reduce the human knowledge engineering costs. In contrast to research paradigms that stipulate that the system must learn its behaviors *tabula rasa*, we believe that the optimal trade-off between human engineering effort and machine learning effort will usually require that the learning system be given whatever level of knowledge can be easily provided by the designer. However, the cost of manual knowledge engineering and the inherent uncertainties in the task environment both imply that some knowledge acquisition is better left to the robot itself.

1.2 TASK ENVIRONMENTS AND KNOWLEDGE REPRESENTATION

Much traditional research on robot learning assumes a closed world model, in which only changes made by the learning agent affect the environment. For robots that need to interact with other agents, this is clearly an unwarranted assumption. Therefore, we typically select learning tasks that involve other agents operating in the learning agent's environment. We generally assume that the learning agent has no access to a predictive model of other agents in the environment. The need to interact with other competitive or cooperating agents motivates our preference for learning reactive behaviors over deliberative planning. Planning generally requires a good domain model, and given our presumed lack of knowledge about the other agents in the environment, we have chosen to focus on learning reactive rules. Our initial experiments, described below, involve learning rules that map current sensor readings to actions.

However, it should be emphasized that learning reactive rules does not necessarily imply a commitment to a simple sensor-to-action level of reasoning. For example, one might learn "reactive" rules that embody high-level control knowledge, such as:

IF situation = danger AND mission = critical
THEN perform high-risk-maneuver
IF situation = danger AND mission < critical
THEN perform low-risk-maneuver
IF...

Our approach is expected to scale up quite naturally to these forms of rules. One could easily imagine a hierarchy of such rule sets, so that the actions such as "perform high-risk-maneuver" would itself be implements as a set of learned reactive rules.

1.3 BACKGROUND KNOWLEDGE

Different learning methods assume different levels of background knowledge. For example, explanation-based learning methods have many advantages if a strong domain model is available. Our research generally focuses on what can be learned in the absence of such a model, motivated by the recognition that an autonomous robot will always face at least some aspects of its environment for which a strong model is not yet available.

Despite the lack of a strong model, we do assume that some useful knowledge is usually available in the form of heuristic rules or advice that can be used in initialize the learning system. Consequently, we have adopted a rule-based knowledge representation that facilitates the specification of heuristic strategies, as well as the interpretation and automatic refinement of learned strategies.

In many practical cases it is necessary to constrain the exploration to avoid physical harm to the robot or to the environment. Our learning systems permit the user to specify fixed rules that either require or forbid the robot to perform certain actions in specified situations.

It is somewhat fashionable to express misgivings about learning under simulations, since simulation models used in AI studies often fail to reflect the complexities, noise, and errors that arise in real sensors and actuators operating in the real world. Furthermore, traditional AI research usually assumes that the inputs have already been correctly translated from analog signals (e.g., sonar, infra-red, etc.) to symbols (e.g., "door-is-open"). In response to such shortcomings, some researchers argue for the development of adaptive robots that evolve behaviors without using a pre-specified model of the world (Brooks, 1992). Here again, we prefer a middle ground, assuming the existence of a limited fidelity simulation model of the robot and its environment. This is consistent with the view that the knowledge acquisition task for autonomous robots be viewed as a cooperative effort between the robot designers and the robot itself. Some relevant knowledge will be known in great detail to the designer, for example, the size and weight of the robot, the characteristics of its sensors and effectors, and at least some of the physics of the task environment. The robot should have access to any such knowledge that the designer can easily provide. This is likely to include a quantitative simulation model of the robot and its environment. It should be noted that some robot manufacturers already provide sophisticated simulation models of the robot in order to aid the development of control software. We see no reason to deny this resource to the learning robot itself.

2 APPROACH

The approach to learning behaviors for robots described here reflects a particular methodology for learning via a simulation model. The motivation is that making mistakes on real systems may be costly or dangerous. In addition, time constraints might limit the number of experiences during learning in the real world, while in many cases, the simulation model can be made to run faster than real time. Since learning may require experimenting with behaviors that might occasionally produce unacceptable results if applied to the real world, or might require too much time in the real environment, we assume that hypothetical behaviors will be evaluated in a simulation model (the off-line system). As illustrated in Fig. 1, the current best behavior can be placed in the on-line execution system, while learning continues in the off-line system (Grefenstette and Ramsey, 1992).

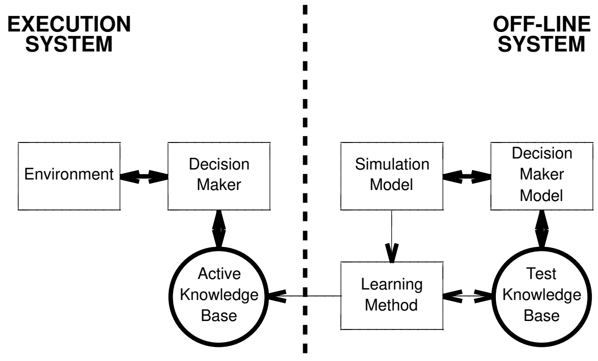

Figure 1: Learning from a Simulation Model

The expectation is that behaviors learned in limited fidelity simulations will be useful in the operational environment. Previous studies have illustrated that knowledge learned under simulation is robust and might be applicable to the real world if the simulation is more *general* (i.e. has more noise, more varied conditions, etc.) than the real world environment (Ramsey, Schultz and Grefenstette, 1990). Where this is not possible, it is important to identify the differences between the simulation and the world and measure the effect upon the learning process.

The next section very briefly explains the learning algorithm (and gives pointers to where more extensive documentation can be found). After that, the actual robot is described. Then we describe the simulation of the robot. Our current learning task is explained, followed by a description of some early experimental results.

2.1 EVOLUTIONARY LEARNING

The field of robotics offers an endless supply of difficult problems, requiring an equally impressive array of methods for their solution. One class of methods that has shown its utility on a number of relevant problems is called *Evolutionary Computation* (De Jong and Spears, 1993). This term applies to computational methods that incorporate principles from biological population genetics to perform search, optimization, and machine learning, and includes a variety of specific formulations with names such as genetic algorithms, evolutionary programming, evolution strategies, and genetic programming. Evolutionary methods have found applications that span the range of architectures for intelligent robotics. For example, evolutionary algorithms have been used to learn rule sets for rule-based autonomous agents (Grefenstette, Ramsey and Schultz, 1990), topologies and weights for neural nets for robotic control (Whitley et al, 1993; Yamauchi, 1994), fuzzy logic control systems (Karr, 1991), programs for LISP-controlled robots (Koza, 1992), and rules for behavior-based robots (Dorigo and Schnepf, 1993).

In our approach evolutionary algorithms are used to explore alternative robot behaviors within a simulation model as a way of reducing the overall knowledge engineering effort. The remainder of this paper will focus on the SAMUEL evolutionary learning system being developed at NRL (Grefenstette et al., 1990) and will present initial results of applying the SAMUEL system to a collision avoidance and navigation task for mobile robots. In SAMUEL, the population is composed of candidate behaviors for solving the task. SAMUEL evaluates the candidate behaviors by testing them in a simulated environment, here a simulation of the robot and its environment. Based on the behaviors' overall performance in this environment, genetic and other operators are applied to improve the performance of the population of behaviors. One cycle of testing all of the competing behaviors is referred to as a *generation*, and is repeated until a good behavior is evolved.

3 EXPERIMENTS

We have been developing this approach at NRL for the past several years, but previous studies have been limited to simulated environments. This section describes some initial results with real robots. These initial results are encouraging, and support the expectation that we will able to scale up our approach to more challenging, multi-agent tasks.

3.1 ROBOT PLATFORM

In these experiments, a Nomadic Technologies, Inc. Nomad 200 robot was used. The robot's drive system, which is housed in the base, uses three synchronized wheels controlled by two motors, one driving the rotation of all three wheels (translation of the robot) and the other controlling the steering of all three wheels. A third motor in the base controls the rotation of a turret that sits on top of the base, although in these experiments, the turret always pointed in the direction of steering.

Twenty tactile sensors are arranged in two offset rings around the base. The turret contains three sensor systems. A set of 16 sonar sensors, equally spaced in a ring around the turret, provide range data from 6 inches to a maximum of 240 inches. Each sonar cell has a beam width of approximately 22.5 degrees. The turret also contains a ring of 16 active infrared sensors. Using a reflective intensity based system, these sensors give an approximate measurement of range from 0 to 24 inches.

The robot has a two-dimensional structured-light range-finding system, but this sensor is not used in these experiments and is not described further. The robot contains an 80486 processor running at 66 megahertz, a separate microprocessor for the sensors and a separate motor controller. Currently, the processor runs the DOS operating system.

The robot can be controlled either from software running on-board the robot's processor, or from a program running on a host computer via radio modem. In both cases, the programmer uses an identical set of library routines which enable the programmer to both access the sensors and give commands to control the robot. In these experiments, the robot is controlled by giving it velocity mode commands; that is, at each decision step, translation and rotation rates are specified for the wheels, and a rotation rate is specified for the turret. These commands are given via a program running on a Unix workstation.

3.2 ROBOT SIMULATION

Learning is accomplished off-line using a simulation of the robot. The robot simulation uses the same C language library interface that is used to control the actual robot, so the simulation is the same as the real robot from a programming point of view. However, there are (as expected) significant differences between the robot simulation and the real world robot. Here, we point out some of the significant differences that affect the learning process.

In the robot simulation, both the sonar and the infrared sensors can be set to properly model beam width (the width of the beam is adjustable in the simulation) or they can be set to do simple ray tracing (i.e., an object is only detected if it intersects with the ray drawn along the direction the sensor is pointing). This has the advantage of being significantly faster to simulate. In these experiments, the ray tracing method is used for both the sonar and the infrared. The effect on learning seems to be that the learned behaviors are more cautious since, during learning, the sonar using ray tracing appears very noisy.

In the real robot, the tactile sensors require a certain amount of force to activate them. However, in the simulation, a collision with an object always results in an activation of the tactile sensor. This difference has no direct effect on learning, but it does effect the testing of the final behaviors on the real robot.

In the real world, wheel slippage and irregularities of the floor result in errors in dead reckoning (i.e., determining the robot's position). Although the simulator has parameters to model simple slippage of the wheels, we set the simulation to not model slippage. The resulting simulation has perfect location information from the integrating of its velocity over time. This does not effect these experiments due to the relatively short distances traversed.

Another difference lies in the time lags associated with the real robot being controlled by a remote host. Getting the sensor values from the robot to the host, and the action from the host back to the robot, takes a certain amount of time that is not accurately simulated. This time delay generally results in poorer performance of the behaviors in the real world since the time from the stimulus to the response differs from that experienced in the simulation. In future experiments, the time lag will be more accurately modeled.

3.3 PERFORMANCE TASK

The task to be performed by the robot, for which it must learn a suitable reactive behavior, is to navigate through a room from a given start position to within a certain radius of a given goal location, while avoiding collisions with obstacles randomly placed in the room. Since the robot does not have a global map of the room, this is *not* a path planning problem, but one of local navigation and collision avoidance. The learning problem is to evolve a behavior represented as a set of stimulus-response rules that map current sensor state into velocity mode commands for the robot to execute. The decision rate is approximately 1 hertz. The robot is given a limited amount of time to reach the goal.

For this task, the learning algorithm is not directly presented with the 52 individual sensor readings or the integrated robot position, but the algorithm is instead given the following *virtual* sensor information:

Forward_sonar: The minimum value returned by the three forward facing sonars.

Rear_sonar: The minimum value of the three rear facing sonars.

Left_sonar: The minimum value returned by the five left facing sonars.

Right_sonar: The minimum value returned by the five right facing sonars.

The infrared is handled is an identical fashion, with four sensors defined, *forward_ir, rear_ir, left_ir,* and *right_ir*. In addition, the following virtual sensors are defined for the learning algorithm to use:

Time: The current decision time step.

Speed: The current translation rate of the robot.

Range: The range in inches to the goal position.

Bearing: The relative bearing in degrees to the goal position.

Given these sensors, the resulting behavior must, at each decision step, produce two actions: a translation rate for the robot which is between -1 and 5 inches per second, and a turning rate between -40 and 40 degrees per second.

For these experiments, the starting position and the goal position do not change. However, with each trial presented to the robot during learning, the obstacles are placed in different positions. There are two sizes of obstacles placed in the room, one slightly smaller than the diameter of the robot, and the other larger than the diameter of the robot.

3.4 RESULTS

For the experiment reported here, the simulated 24 by 30 foot room contained between 5 and 10 obstacles, each obstacle randomly chosen to be either 1.5 or 2.5 feet on a side. The obstacles were placed somewhat randomly: Five slightly overlapping regions in the room were defined roughly corresponding to the North-East, South-East, North-West and South-West sections of the room, with another region defined for the center of the room. Each of the five regions had either one or two of the obstacles placed randomly within its boundaries. This arrangement guarantees that a solution exists, yet forces the robot to have to navigate around several boxes. Each trial begins with a different configuration of the obstacles in the room. The robot was given 80 decision time steps to cross the room to the goal position.

The initial *heterogeneous population* (Schultz and Grefenstette, 1990) consisted of a variety of rule sets from different sources. This included a combination of manually (human) generated rules sets and automatically generated variants of those rules. The best of these initial rule sets could successfully reach the goal 72.5 percent of the time in simulation, while the worst of them would never reach the goal. The population size was 50 rule sets. Each rule set was evaluated on 20 trials to determine its fitness, which was defined to be the average of the performance measure over the 20 trials. The system was run for 50 generations.

After every five generations, the best 5 rules sets (based on the average performance measure) were tested on 50 random trials to see which could complete the task the greatest number of times. The best performing rule set was then evaluated on another 200 trials and this value is plotted in Fig. 2.

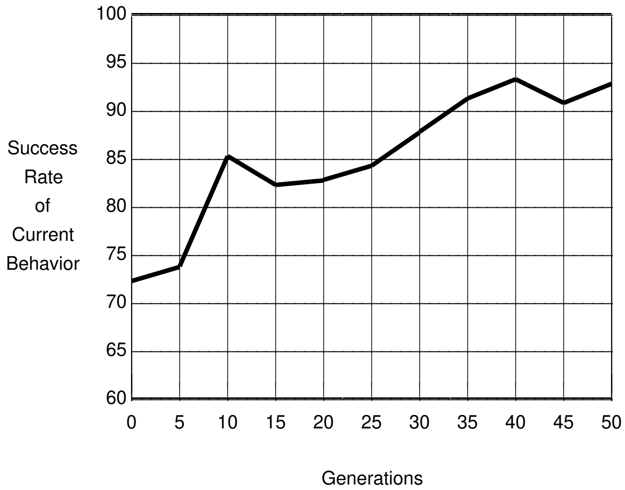

Figure 2: Learning the Navigation Task.

As seen in this figure, the best performance behavior could complete the task 93.5 percent of the time in the simulated environment. After the experiment was completed, the best rule set generated during the experiment (obtained in generation 40) was tested on the real robot. In 28 runs, the robot succeeded 24 times, a performance level of 86 percent.

4 FUTURE DIRECTIONS

We expect to scale up this method to multi-robot tasks, including competitive tasks such as playing tag, pursuit and evasion games. In addition, we will continue to explore ways on speeding the learning process by including domain-specific biases in the evolutionary learning method (Grefenstette, 1987). Here, we outline specific extensions used for our robot navigation problem.

• *Choice of Search Space and Representation.* The first choice the user of an evolutionary algorithm needs to make is the choice of the search space, along with an appropriate representation. This choice clearly reflects the trade-offs between human and machine effort. For example, if the choice is to search the space of all mappings between the set of all possible sensor readings (digitized to greatest possible resolution) to the set of all possible actuator commands, this clearly places the entire burden of knowledge acquisition on the learning system. A more reasonable trade-off might be to process the raw sensor readings into a set of virtual sensors and to provide a set of behaviors for the robot, leaving the task of learning the mappings between these much more constrained sets. If the designer can specify the desired behavior to within a set of numeric parameters, then evolutionary algorithms can be used to search the space of parameters for the most desirable values.

In the SAMUEL learning system, the representation of a behavior is a set of stimulus-response rules. The left hand sides of the rules are matched against the sensor state of the robot, and the right hand sides of the rules specify the action the robot is to take. Behaviors are evaluated in a production system interpreter that interacts with the robot (either real or simulated). Here is an example of part of a rule set:

```
IF   front_sonar < 30   AND   bearing > 10   THEN turn = 20
IF   front_ir < 5   THEN   speed = -10
IF   ...
```

Each such rule plays the role of a *gene*, with a set of such rules, or *behavior*, treated as a candidate solution to the problem (navigation in this case).

In each decision cycle, the interpreter reads the current state of the sensors to find rules that match, resolves conflicts to determine one rule to fire based on previously observed utilities of rules, and then fires the rule by passing back to the robot the action to be performed (for example a velocity mode command). This cycle is repeated until the task is accomplished or failure occurs.

In SAMUEL, the user can further limit the search space by defining a set of *constraints* in the form of rules that specify conditions under which certain actions are either forbidden or required (Grefenstette and Cobb, 1994). Constraints are intended to limit the robot's actions within physically safe parameters, but still allow freedom to explore a large set of alternative strategies (Grefenstette, 1992).

• *Initial Population.* Evolutionary algorithms often begin with candidate solutions selected at random from the search space. Often, the approach can be sped up by the use of heuristics to select the starting population. This must be done with care, however, since a lack of sufficient diversity in the initial population is almost guaranteed to produced premature convergence to suboptimal solutions.

In SAMUEL, the rule representation was designed to encourage the user to include heuristic strategies in the initial population (Schultz and Grefenstette, 1990). In fact, for many complex robotic tasks, it is unlikely that the system will be able to evolve solutions without some initial heuristics. For example, if the task is to track a moving target, the robot is unlikely to perform the task at all given a set of random rules of the form shown above. We are exploring several approaches for exploiting heuristic knowledge, including the automatic generation of several variants from a user-generated rule set. Again, this raises the issue of trade-offs between the effort of specifying good initial rules and the effort of engineering the search space to include only plausible solutions to the task. The optimal trade-off will vary from task to task.

• *Fitness Function.* The fitness function is the main mechanism for expressing bias in the genetic learning approach. Two important issues must be addressed. First, the fitness function should accurately reflect the desired performance on the task. Evolutionary algorithms are highly opportunistic optimizers and may produce surprising results if the fitness function rewards some behavior that the system designer does not want. Evolutionary algorithms may also exploit unexpected features of the simulation model to maximize performance. Of course, this implies that the model should accurately reflect the conditions in the target environment. To the extent that this is not possible, our studies (Ramsey et al., 1990) have shown that it is still possible to learn from limited-fidelity simulations that err on the side of difficulty (e.g., have more noisy sensors that the real robots). In such cases, the learning time increases, but so does the robustness of the learned rules. Second, the fitness function should provide differential payoff so that alternative candidate solutions can be ranked. It should not present a "needle-in-a-haystack" problem in which only the final solution is assigned a positive fitness. In the task described here, the performance measure is based on differential payoff for different behaviors of the robot. In the navigation experiments, getting to the goal quickly yields the highest payoff while behaviors that result in collisions with an obstacle receive the lowest payoff.

• *Genetic Operators.* Many early studies of genetic algorithms employed simple syntactic operators to alter candidate solutions, such as simple crossover and random mutation. However, some recent studies have used more heuristic operators that make more directed changes based on the learning agent's experience. For example, some genetic classifier systems use *triggered operators* such a creating a new rule to cover a novel situation (Booker, 1988). In SAMUEL, we use *generalization* and *specialization* operators that are triggered by specific conditions relating the measured utilities of individual rules and the outcome of the task. These may be viewed as Lamarckian forms of evolution (Grefenstette, 1991), and show that artificial evolution need not proceed along purely Darwinian lines.

• *Hybrid Approaches.* Finally, it is often useful to use evolutionary methods in concert with other methods that have complementary strengths. The strength of an evolutionary algorithm is in rapidly identifying the most promising regions of a complex search space. Evolutionary methods are less efficient at fine-tuning candidate solutions. Therefore, a natural hybrid is to use an efficient local optimization method to improve the final solutions found by the evolutionary system (Grefenstette, 1987).

Another promising hybrid approach is to combine the SAMUEL learning method with a case-based module, to provide a way to dynamically modify the simulation model based on the robot's experience with the external environment. The next section covers this approach in more detail.

4.1 ANYTIME LEARNING

The basic characteristics of anytime algorithms are: (1) the algorithm can be suspended and resumed with negligible overhead, (2) the algorithm can be terminated at any time and will return some answer, and (3) the answers returned improve over time. However, our use of the term *anytime learning* is meant to denote a particular way of integrating execution and learning. The basic idea is to integrate two continuously running modules: an execution module and a learning module (see Figure 3). The agent's learning module contains a simulation model with parameterized aspects of the domain. It continuously tests new strategies against the simulation model to develop improved strategies, and updates the knowledge base used by the agent (in the execution system) on the basis of the best available results. The execution module controls the agent's interaction with the environment, and includes a monitor that can detect changes in the environment, and dynamically modify the parameter ranges of the simulation model (used by the learning module) to reflect these changes. When the simulation model is modified, the learning process is restarted on the modified model. The learning system is assumed to operate indefinitely, and the execution system uses the results of learning as they become available.

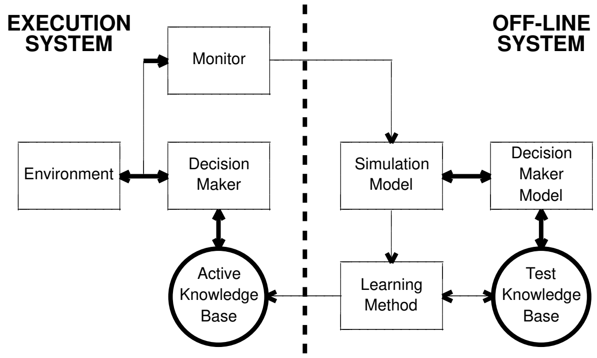

Figure 3: Anytime Learning System

In short, this approach uses an existing simulation model as a base on which strategies are learned, and updates the parameters of the simulation model as new information is observed in the external environment in order to improve long-term performance.

One important aspect of the anytime learning model concerns the criteria for deciding when the external environment has changed. In our current implementation, the monitor compares measurable aspects of the environment with the parameters provided by the simulation design. In addition, the monitor might also detect differences between the expected and actual performance of the current strategy in the environment. For example, if the performance level degrades in the environment, that is a sign that the current strategy is no longer applicable. If the performance of the current strategy improves unexpectedly, it may indicate that the environment has changed, and that another strategy may perform even better.

While the anytime learning approach could be used merely to fill in particular values for parameters that are initially unknown, it is especially useful in changing environments. Each observed set of values for the parameters can be treated as an environmental *case*. The learning system can store strategies for different cases, indexed by the environmental parameters that characterize that case. When the environment changes, the set of previous cases can be searched for similar cases, and the corresponding best strategies can be used as a starting point for the new case.

Our particular instantiation of anytime learning uses SAMUEL (Grefenstette and Cobb, 1994; Grefenstette et al., 1990; Schultz, 1991). While the basic ideas of anytime learning could be applied using a number of other learning methods, especially other reinforcement learning methods, SAMUEL has some important advantages for this approach. In particular, SAMUEL can learn rapidly from partially correct strategies and with limited fidelity simulation models (Schultz and Grefenstette, 1990; Ramsey et al., 1990). More importantly for this discussion, the genetic algorithms provide an effective mechanism for modifying previously learned cases. When an environmental change is detected, the genetic algorithm is restarted with a new initial population. This initial population can be "seeded" with the best strategies found for previous similar cases. We call this *case-based initialization* of the genetic algorithm. Our studies have shown that by using good strategies from several different past cases as well as exploratory strategies, default strategies and members of the previous population, the genetic algorithm can respond effectively to environmental changes, and can also recover gracefully from spurious false alarms (i.e., when the monitor mistakenly reports that the environment has changed).

A case-based anytime learning system can be viewed as one that starts with an underspecified model of its world (the original parameterized quantitative model), and then learns, on the basis of experience, a set of fully specified models that correspond to the environmental cases it actually encounters. The power of this approach derives from the cooperative effort between designer and robot. The designer provides a rich sets of models that can be used for learning, and the robot selects the appropriate models from that (possibly infinite) set of models. Each partner makes a significant contribution to the knowledge acquisition process.

Preliminary experiments have shown the effectiveness of anytime learning in a changing environment (Grefenstette and Ramsey, 1992; Ramsey and Grefenstette, 1993). The task used in these experiments was a cat-and-mouse game in which one robot had to track another without being detected. The changing environmental parameters included the speed distribution of the target agent and the maneuverability of the target agent, as well as environmental variables that were in fact irrelevant to the performance of the task. The most promising aspect of these results is that, within each time period (epoch) after an environmental change, the case-based anytime learning system consistently showed a much more rapid improvement in the performance of the execution system than a baseline learning system (with its monitor disabled). Case-based initialization allows the system to automatically bias the search of the genetic algorithm toward relevant areas of the search space. More recent experiments show that the longer the epochs last and the longer the system runs and gathers a base of experiences of different environmental cases, the greater the expected benefit of case-based anytime learning is.

5 SUMMARY

The SAMUEL system has been used to learn behaviors for controlling simulated autonomous underwater vehicles (Schultz, 1991), missile evasion, and other simulated tasks. This paper reports some early tests of the learned knowledge on a real physical system. For more details of the SAMUEL system, see (Grefenstette and Cobb, 1994).

Future work will continue examining the process of building robotic systems through evolution. We want to know how multiple behaviors that will be required for a higher level task interact, and how multiple behaviors can be evolved simultaneously. We are also examining additional ways to bias the learning both with initial rule sets, and by modifying the rule sets during evolution through human interaction. Other open problems include how to evolve hierarchies of skills and how to enable the robot to evolve new fitness functions as the need for new skills arises.

References

Booker, L. B. (1988). Classifier systems that learn internal world models. *Machine Learning 3(3),* 161-192.

Brooks, R. *Artificial Life and Real Robots.* Cambridge: MIT Press. 1992.

De Jong, K. A. and W. Spears (1993). On the state of evolutionary computation. *Proceedings of the Fifth International Conference on Genetic Algorithms,* pp. 618-626, San Mateo, CA: Morgan Kaufmann.

Dorigo, M. and U. Schnepf (1993). Genetics-based machine learning and behavior-based robotics: a new synthesis. *IEEE Transactions on Systems, Man and Cybernetics, SMC-23, 1.*

Grefenstette, J. (1987). Incorporating problem specific knowledge into genetic algorithms, in *Genetic Algorithms and Simulated Annealing,* L. Davis (Ed.), London: Pitman.

Grefenstette, J. J. (1991). Lamarckian learning in multi-agent environments. *Proceedings of the Fourth International Conference of Genetic Algorithms* pp. 303-310, San Mateo, CA: Morgan Kaufmann.

Grefenstette, J. (1992). The evolution of strategies for multi-agent environments. *Adaptive Behavior 1(1),* 65-90.

Grefenstette, J. J. and H. C. Cobb (1994). User's guide for SAMUEL. Version 4.0. NRL Report, Naval Research Lab, Washington, DC.

Grefenstette, J. J. and C. L. Ramsey (1992). An approach to anytime learning. *Proceedings of the Ninth International Conference on Machine Learning* pp. 189-195, D. Sleeman and P. Edwards (eds.), San Mateo, CA: Morgan Kaufmann.

Grefenstette, J. J., C. L. Ramsey and A. C. Schultz (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning 5(4),* 355-381.

Karr, C, L. (1991). Design of an adaptive fuzzy logic controller using a genetic algorithm. *Proceedings of the Fourth International Conference on Genetic Algorithms,* pp. 450-457, San Mateo, CA: Morgan Kaufmann.

Koza, J. R. (1992). *Genetic Programming.* Cambridge, MA: MIT Press.

Ramsey, C. L., A. C. Schultz and J. J. Grefenstette (1990). Simulation-assisted learning by competition: Effects of noise differences between training model and target environment. *Proceedings of the Seventh International Conference on Machine Learning* pp. 211-215. San Mateo, CA: Morgan Kaufmann.

Ramsey, C. L. and J. J. Grefenstette (1993). Case-based initialization of genetic algorithms. *Proc. Fifth Int. Conf. on Genetic Algorithms.* pp. 84-91, San Mateo, CA: Morgan Kaufmann.

Schultz, A. C., Using a genetic algorithm to learn strategies for collision avoidance and local navigation. *Seventh International Symposium on Unmanned, Untethered, Submersible Technology, 1991,* (pp 213-225). Durham, NH.

Schultz, A. C. and J. J. Grefenstette (1990). Improving tactical plans with genetic algorithms. *Proceedings of IEEE Conference on Tools for AI 90* (pp 328-334). Washington, DC: IEEE.

Whitley, D., S. Dominic, R. Das, C. Anderson (1993). Genetic reinforcement learning for neurocontrol problems. *Machine Learning 13(2/3),* 259-284.

Yamauchi, B. (1994). Dynamic neural networks for mobile robot control. *ISRAM 94.*