Using a Genetic Algorithm to Learn Strategies for Collision Avoidance and Local Navigation†

Alan C. Schultz

Navy Center for Applied Research in Artificial Intelligence (Code 5514),
Naval Research Laboratory, Washington, DC 20375-5000, U.S.A.
EMAIL: schultz@aic.nrl.navy.mil
(202) 767-2684

Abstract

Navigation through obstacles such as mine fields is an important capability for autonomous underwater vehicles. One way to produce robust behavior is to perform projective planning. However, real-time performance is a critical requirement in navigation. What is needed for a truly autonomous vehicle are robust reactive rules that perform well in a wide variety of situations, and that also achieve real-time performance. In this work, SAMUEL, a learning system based on genetic algorithms, is used to learn high-performance reactive strategies for navigation and collision avoidance.

1. Introduction

Navigation through obstacles such as mine fields is one of many important capabilities or behaviors for autonomous underwater vehicles (AUV). One way to produce robust behavior is to perform projective planning. However, real-time performance is often a critical requirement for many of the capabilities needed in autonomous vehicles. Reactive systems, where stimulus-response rules drive the behavior of the vehicle, can achieve real-time performance and can perform well in a wide variety of situations. An interesting and hard problem is how to develop or obtain the rules for these reactive systems. In previous work, SAMUEL, a learning system based on genetic algorithms, was used to learn reactive behavior to solve tracking and evasion problems. In this work, SAMUEL is used to learn high-performance reactive strategies for navigation and collision avoidance. This task domain requires an AUV to navigate through a randomly generated, dense mine field and then rendezvous with a stationary object. The AUV has a limited set of sensors, including sonar, and must set its speed and direction each decision cycle. The strategy, or plan, that is learned is expressed as a set of reactive rules (i.e. stimulus-response rules) that map sensor readings to actions to be performed at each decision time-step.

Genetic algorithms are powerful, adaptive search techniques that can learn high performance knowledge structures. The genetic algorithm's strength comes from the implicitly parallel search of the solution space that it performs, via a population of candidate solutions. In SAMUEL, the population is composed of candidate reactive strategies for solving the navigation problem. SAMUEL evaluates the candidate strategies by testing them in a simulated "AUV environment." Based on the strategies' overall performance in this environment, SAMUEL applies genetic and other operators to improve the performance of the population of strategies.

Simulation results demonstrate that an initial, human-designed strategy which has an average success rate of only eight percent on randomly generated mine fields can be improved by this system so that the final strategy can achieve a success rate of 96 percent.

† Reprinted from the Proceedings of the Seventh International Symposium on Unmanned Untethered Submersible Technology, University of New Hampshire Marine Systems Engineering Laboratory, September 23-25, 1991, pp. 213-215.

Section 2 will describe our approach to learning behaviors for autonomous vehicles. In Section 3, the SAMUEL system and the AUV domain will be described in detail. Descriptions of experiments and empirical results can be found in section 4, and finally, concluding remarks and future research will be described in section 5.

2. An Approach to Learning Strategies

In response to the knowledge acquisition bottleneck associated with the design of expert systems, research in machine learning attempts to automate the knowledge acquisition process and to broaden the base of accessible sources of knowledge. The choice of an appropriate learning technique depends on the nature of the performance task and the form of available knowledge. If the performance task is classification, and a large number of training examples are available, then inductive learning techniques (Michalski, 1983) can be used to learn classification rules. If there exists an extensive domain theory and a source of expert behavior, then explanation-based methods may be applied (Mitchell, Mahadevan & Steinberg, 1985). Many interesting practical problems that may be amenable to automated learning do not fit either of these models. One such class of problems is the class of sequential decision tasks, such as the AUV task described in this work. For many interesting sequential decision tasks, there exists neither a database of examples nor a complete and tractable domain theory that might support traditional machine learning methods. In these cases, one method for manually developing a set of decision rules is to test a hypothetical set of rules against a simulation model of the task environment, and to incrementally modify the decision rules on the basis of the simulated experience.

The approach described here reflects a particular methodology for learning via a simulation model. The motivation behind the methodology is that making mistakes on real systems may be costly or dangerous. Since learning may require experimenting with tactical plans that might occasionally produce unacceptable results if applied to the real world, we assume that hypothetical plans will be evaluated in a simulation model (see Figure 1).

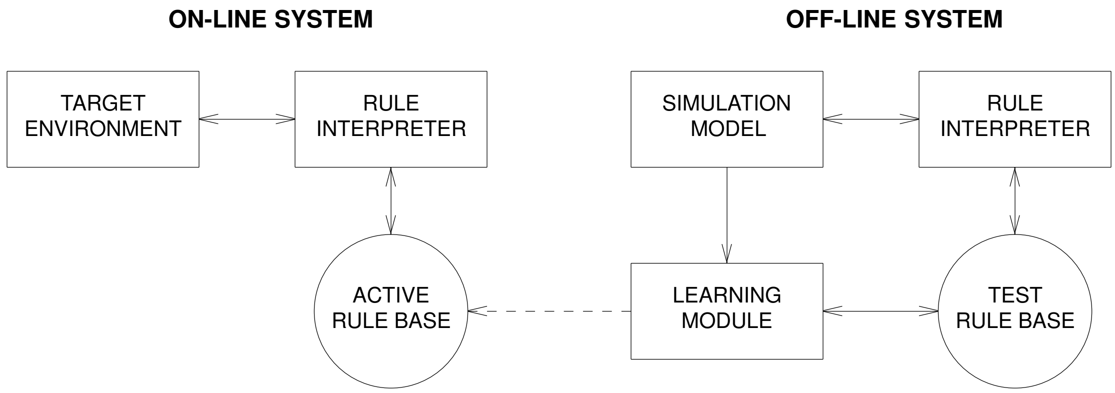

Fig. 1. A Model for Learning from a Simulation Model.

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).

3. Description of the SAMUEL System

SAMUEL is a system designed to learn reactive strategies, or behaviors, for solving sequential decision problems.[1] SAMUEL consists of three major components: a problem specific module, a

[1] SAMUEL stands for Strategy Acquisition Method Using Empirical Learning. The name also honors Art Samuel, one of the pioneers in machine learning.

performance module, and a learning module. Figure 2 shows the architecture of the system. The problem specific module consists of the task environment simulation, or world model, and its interfaces. In these experiments, the world model is a simulation of an AUV and a mine field. The performance module is called CPS (Competitive Production System), a production system that interacts with the world model by reading sensors, setting control variables, and obtaining payoff from a critic. Like traditional production system interpreters, CPS performs matching and conflict resolution. In addition, CPS performs rule-level assignment of credit based on the intermittent feedback from the critic. The learning module uses a genetic algorithm to develop reactive tactical plans, expressed as a set of condition-action rules. Each plan is evaluated by testing its performance on a number of tasks in the world model. As a result of these evaluations, plans are selected for replication and modification. Genetic operators, such as crossover and mutation, and other operators, such as generalization and specialization, produce plausible new plans from high performance precursors.

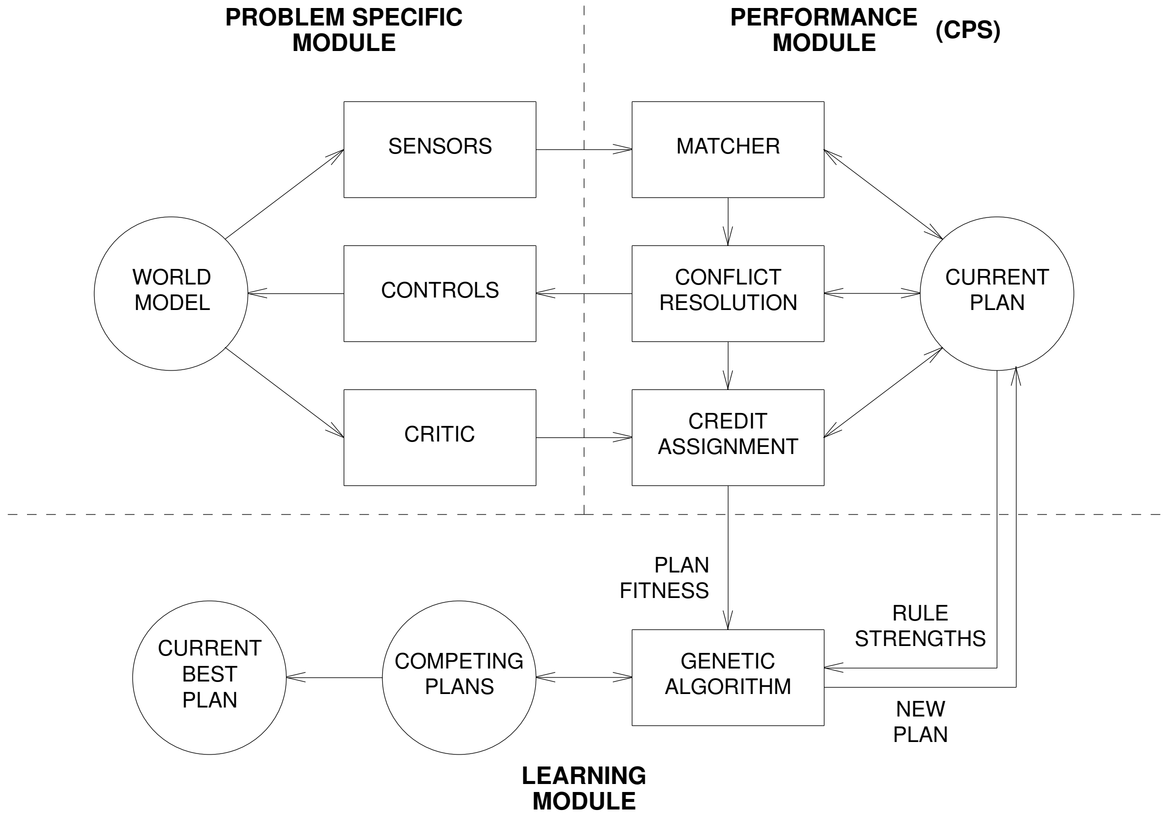

Fig. 2. SAMUEL: A Learning System for Tactical Plans

3.1. Problem Specific Module: Description of AUV simulation domain

The problem domain in this study is a simple two-dimensional simulation of an AUV that must navigate through a dense mine field towards a stationary object with which it must rendezvous.[2] The AUV has a limited set of sensors, including sonar, and can set its speed and direction each decision cycle. We wish to learn a strategy that is expressed as a set of reactive rules, (i.e. stimulus-response rules) that map sensor readings to actions to be performed at each decision time-step. Note that the system does *not*

[2] Future work will use a higher fidelity, three-dimensional model.

learn a specific path, but a set of rules that reactively decide a move at each time step allowing the vehicle to reach its goal and avoid the mines.

3.1.1. Sensors

We assume that the AUV knows its own position with some margin of error, and that the position of the stationary target is known. The AUV is equipped with an active sonar for detecting obstacles in its path. The sonar is composed of seven cells, each with a resolution of 10 degrees, giving the AUV a total coverage of 70 degrees. The AUV also has some internal, or virtual, sensors that give the AUV certain information about its own state. The sensors are:

1) *last-turn*: the current turning rate of the AUV. This sensor can assume 13 values, ranging from -60 degrees to 60 degrees in 10 degree increments.

2) *time*: the current elapsed time, an integer.

3) *range*: the range to the rendezvous area. Assumes 16 values from 0 to 1500.

4) *bearing*: the bearing to the rendezvous area. Assumes integer values from 1 to 12. The bearing is expressed in ‘‘clock terminology’’, in which *12 o'clock* denotes dead ahead of the AUV, and *6 o'clock* denotes directly behind the AUV.

5) *speed*: the current speed of the AUV. Assumes 9 values from 0 to 40. Note that the AUV can come to a stop.

6-12) *asonar_n*: One of *n* active sonar cells used for collision avoidance. Each cell takes on values from 0 to 200 in increments of 10 and represents the distance to an object within that sonar cell's view. If no object is seen, a special value, 220, indicates infinity.

Each sensor can have noise added to simulate a more realistic environment. In particular, the sonar readings have both a gaussian noise added, and a small random probability of "missing" an object, or of reading a "ghost" object that is not really there.

3.1.2. Actions (effectors)

There is a discrete set of actions available to control the AUV. In this study, we consider actions that specify discrete turning rates and discrete speeds for the AUV. The control variable *turn* has 13 possible settings, between -60 and 60 degrees in 10 degree increments. Note that the sonar can only cover -35 to +35 degrees, so it is possible for the AUV to turn into an object that is not in sonar range. The control variable *speed* has 9 possible settings between 0 and 40 (the units are arbitrary) with an increment of five. The learning objective is to develop a *reactive plan*, i.e., set of decision rules that map current sensor readings into actions, that successfully allows the AUV to rendezvous with the stationary target within a time limit while avoiding mines. The time limit loosely represents a limit of fuel, and future studies will model energy conservation more accurately. In this simulation, we assume that the AUV can change speed a maximum of 10 units and direction a maximum of 10 degrees within one decision time step.

The AUV simulation is divided into *episodes* that begin with the placement of the AUV centered in front of a randomly generated mine field with a specified density. The episodes end with either a successful rendezvous at the target location, or a loss of the AUV due to time running out (no fuel) or a collision with a mine. The rendezvous is successful if the AUV approaches within 50 units of the target location. It is assumed that the only feedback provided is a numeric payoff, supplied at the end of each episode, that reflects the success of the episode with respect to the goal of reaching the rendezvous point. The payoff is defined by the formula:

$$
\begin{aligned}
payoff &= 1.0 \quad \text{if AUV reaches rendezvous area.} \\
&= \frac{50}{50+range} \quad \text{if AUV is destroyed or runs out of fuel.}
\end{aligned}
$$

The payoff returned by the critic is 1.0 for a success and a value between 0 and 0.5 (non-inclusive) depending on the AUV's distance from the goal when the AUV is lost; this gives partial credit for failures.

3.2. Performance Component

The performance module of SAMUEL, CPS, has some similarities to both traditional production system interpreters and to classifier systems. The primary features of CPS are:

- A restricted but high level rule language;
- Partial matching;
- Competition-driven conflict resolution; and
- Incremental credit assignment methods.

These features are described in more detail in the following sections.

3.2.1. Knowledge Representation

In a departure from several previous genetic learning systems, SAMUEL learns rules expressed in a high level rule language. The use of a high level language for rules offers several advantages over low level binary pattern languages typically adopted in genetic learning systems. First, it is easier to incorporate existing knowledge, whether acquired from experts or by symbolic learning programs. Second, it is easier to explain the knowledge learned through experience and to transfer the knowledge to human operators. Third, it is possible to combine several forms of learning in a single system (Gordon & Grefenstette, 1990). Each CPS rule has the form

```
if   (and c1  . . .  cn)
then (and a1  . . .  am)
```

where each c_i is a condition on one of the sensors and each action a_j specifies a setting for one of the control variables.

The form of the conditions depends on the type of the sensor. SAMUEL supports four types of sensors: *linear, cyclic, structured,* and *pattern.* Linear sensors take on linearly ordered numeric values. Conditions over linear sensors specify upper and lower bounds for the sensor values. For example, the *speed* sensor in AUV can take on values over the range 0 to 40, discretized into 8 equal segments. Thus, an example of a legal condition over *speed* is

```
(speed is [10 .. 15])
```

which matches if $10 \le speed \le 15$. In AUV, *last-turn, time, range, speed,* and *the sonar cells* are linear sensors.

Cyclic sensors take on cyclicly ordered numeric values. Like linear sensors, the range of each cycle sensor is divided by the user into equal segments whose endpoints constitute the legal bounds in the conditions. Unlike linear sensors, any pair of legal values can be interpreted as a valid condition for cyclic sensors. In AUV, *bearing* is a cyclic sensor, since the next ''higher'' value than *bearing* = 12 is *bearing* = 1. The following is a legal condition over *bearing*:

```
(bearing is [10 .. 3])
```

which matches if $10 \le bearing \le 12$ or if $1 \le bearing \le 3$.

The rule language of CPS also supports structured nominal sensors whose values are taken from the nodes of a tree-structured hierarchy. Conditions for structured sensors specify a list of values, and the condition matches if the sensor's current value occurs in a subtree labeled by one of the values in the list. Structured nominal sensors are not used in the AUV domain.

Finally, pattern sensors can take on binary string values. Conditions on pattern sensors specify patterns over the alphabet {0, 1, #}, as in classifier systems (Holland, 1986). The AUV domain does not use pattern sensors.

The right-hand side of each rule specifies a setting for one or more control variables. For the AUV problem, each rule specifies a setting for the variable *turn*, and a setting for the variable *speed*. In general, a given rule may specify conditions for any subset of the sensors and actions for any subset of the control variables. Each rule also has a numeric *strength,* that serves as a prediction of the rule's utility (Grefenstette, 1988). The methods used to update the rule strengths is described in the section on credit assignment below. A sample rule in the AUV system follows:

```
if   (and (bearing is [11 .. 1]) (speed is [20 .. 40])
     (asonar_4 is [0 .. 100]))
then ((turn is [30]) (speed is [10]))
strength 0.75
```

3.2.2. Production System Cycle

CPS follows the match/conflict-resolution/act cycle of traditional production systems. Since there is no guarantee that the current set of rules is in any sense complete, it is important to provide a mechanism for gracefully handling cases in which no rule matches (Booker, 1985). In CPS this is accomplished by assigning each rule a match score equal to the number of conditions it matches. The match set consists of all the rules with the highest current match score.

Once the match set is computed, an action is selected from the (possibly conflicting) actions recommended by the members of the match set. Each possible action receives a *bid* equal to the strength of the *strongest rule* in the match set that specifies that action in its right-hand side. Unlike classifier systems in which all members of the match set vote on which action to perform (Riolo, 1988), CPS selects an action using the probability distribution defined by the strength of the (single) bidder for each action. This prevents a large number of low strength rules from combining to suggest an action that is actually associated with low payoff. All rules in the match set that agree with the selected action are said to be *active* (Wilson, 1987), and will have their strength adjusted according to the credit assignment algorithm described in the next section.

After conflict resolution, the control variables are set to the values indicated by the selected actions.[3] The world model is then advanced by one simulation step. The new state is reflected in a new set of sensor readings, and the entire process repeats.

3.3. The Learning Module

Learning in SAMUEL occurs on two distinct levels: credit assignment at the rule level, and genetic competition at the plan level.

3.3.1. Credit Assignment

Systems that learn rules for sequential behavior generally face a *credit assignment problem:* If a sequence of rules fires before the system solves a particular problem, how can credit or blame be accurately assigned to early rules that set the stage for the final result? Our approach is to assign each rule a measure called *strength* that serves as a prediction of the expected level of payoff that will be

---

[3] If there is more than one control variable, as is the case in the AUV domain, the conflict resolution phase is executed independently for each control variable. As a result, the settings for different control variables may be recommended by distinct rules.

achieved if this rule fires. When payoff is obtained at the end of an episode, the strengths of all active rules (i.e., rules that suggested the actions taken during the current episode) are incrementally adjusted to reflect the current payoff. The adjustment scheme, called the *Profit Sharing Plan (PSP)*, consists of subtracting a fraction of the rule's current strength and adding the same fraction of the payoff. Rules whose strength correctly predicts the payoff retain their original levels of strength, while rules that overestimate the expected payoff lose strength and rules that underestimate payoff gain strength. It has been shown that the PSP computes a time-weighted estimate of the expected external payoff, and that this estimate is useful for conflict resolution (Grefenstette, 1988). However, conflict resolution should take into account not only the expected payoff associated with each rule, but also some measure of our confidence in that estimate. One way to measure confidence is through the variance associated with the estimated payoff. In SAMUEL, the PSP has been adapted to estimate both the mean and the variance of the payoff associated with each rule. Thus, a high strength rule must have both high mean and low variance in its estimated payoff. By biasing conflict resolution toward high strength rules, we expect to select actions for which we have high confidence of success.

3.3.2. The Genetic Algorithm

At the plan level, SAMUEL treats the learning process as a heuristic optimization problem, i.e., a search through a space of knowledge structures looking for structures that lead to high performance. A genetic algorithm is used to perform the search. Genetic algorithms are motivated by standard models of heredity and evolution in the field of population genetics, and embody abstractions of the mechanisms of adaptation present in natural systems (Holland, 1975). Briefly, a genetic algorithm simulates the dynamics of population genetics by maintaining a knowledge base of *knowledge structures* that evolves over time in response to the observed performance of its knowledge structures in their training environment. Each knowledge structure yields one point in the space of alternative solutions to the problem at hand, which can then be subjected to an *evaluation* process and assigned a measure of utility (its *fitness*) reflecting its potential worth as a solution. The search proceeds by repeatedly selecting structures from the current knowledge base on the basis of fitness, and applying idealized *genetic search operators* to these structures to produce new structures (*offspring*) for evaluation. Goldberg (1989) and Davis (1991) provide a detailed discussion of genetic algorithms. The learning level of SAMUEL is a specialized version of a standard genetic algorithm, GENESIS (Grefenstette, 1986). In SAMUEL, the knowledge structures that make up the population are plans, or sets of reactive rules, that represent a strategy for solving the problem. The remainder of this section outlines the differences between GENESIS and the genetic algorithm in SAMUEL.

3.3.2.1. Adaptive Initialization and Using Existing Knowledge

Several approaches to initializing the knowledge structures of a genetic algorithm have been reported. By far, random initialization of the first population is the most common method. This approach requires the least knowledge acquisition effort, provides a lot of diversity for the genetic algorithm to work with, and presents the maximum challenge to the learning algorithm. As a second alternative, we have developed an approach called *adaptive initialization*. Each plan starts out as a completely general rule, but its rule is specialized according to its early experiences, thus creating more rules for each plan. Specifically, each plan in the initial population consists of a maximally general rule which says:

for any sensor readings, take any action

A plan consisting of only this rule executes a random walk, since the rule matches on every cycle and specifies any possible legal action. Although each plan in the initial population executes this random walk policy, it does not follow that they all have the same performance, since the initial conditions for the episodes used to evaluate the plans are selected at random, e.g. the location of the mines and the goal location.

In order to introduce plausible new rules, (and also to specialize overly general rules in the plan later in the learning) a plan modification operator called SPECIALIZE is applied after each evaluation of a plan. The trigger in this case is the conjunction of the following conditions:

(1) There is room in the plan for at least one more rule.

(2) An episode ended with a successful rendezvous.

If these conditions hold, SPECIALIZE creates a new rule with the right hand side being set to the action that occurred during the successful episode in (2) above, and with a more specialized left-hand side. For each sensor, the condition for the sensor in the new rule covers approximately half the legal range for that sensor, splitting the difference between the extreme legal values and the sensor reading obtained in (2) above. For example, suppose the initial plan contains the maximally general rule:

```
Rule 1:  if ( ) then (turn is ANY) (speed is ANY)
```

Suppose further that the following step is recorded in the evaluation trace during the evaluation of this plan:

```
  ⋮
sensors: ... (time = 4) (range = 500) (bearing = 6) ...
action: (turn = −15) (speed = 10)     rule fired: Rule 1
  ⋮
```

Then SPECIALIZE would create the following new rule:

```
Rule 2:
if   (and ... (time is [2 .. 11]) (range is [300 .. 1000])
              (bearing is [3 .. 9]) ...)
then ((turn is [-15]) (speed is [10]))
```

The resulting rule is given a high initial strength, and added to the plan. The new rule is plausible, since its action is known to be successful in at least one situation that matches its left hand side. Of course, the new rule is likely to need further modification, and is subject to further competition with the other rules.

A third approach is to seed the initial population with existing knowledge (Schultz and Grefenstette, 1990). The rule language of SAMUEL was designed to facilitate the inclusion of available knowledge. In some cases, such as the AUV domain, random behavior will never yield a success and so the adaptive initialization will not specialize the maximally general rule and create new rules. In these domains, it is essential that the initial population include heuristic knowledge to start the search. This initial knowledge does not need to lead to very good results, but simply gives the system some initial successes so that the adaptive initialization will work. Following are the set of initial rules used in this study. Note that the performance achieved with this hand-crafted plan is only eight percent, i.e. the AUV can successfully reach the rendezvous area 8 out of 100 episodes.

```
# This rule is for speed action.  It just says to randomly pick a speed.

Rule  1  if   (and)
     then (and)
     action TURN  strength 0.0
     action SPEED strength 1.0

# If the goal is somewhat in front of us, and nothing is too close in
# the sonar that looks directly ahead, then keep going straight.
```

```
Rule  2  if   (and (bearing is [11 .. 1]) (asonar4 is [180 .. 220]))
        then (and (turn is [0]))
        action TURN  strength 1.0
        action SPEED strength 0

# This rule is for collision avoidance.  If the goal is somewhat is front of us, and there
# is an object within range directly in front of us, then turn sharply. Left turn was picked out of thin air.

Rule  3  if   (and (bearing is [11 .. 1]) (asonar4 is [10 .. 190]))
        then (and (turn is [60]))
        action TURN  strength 1.0
        action SPEED strength 0

# The next group of rules say to turn towards the goal. Of course,
# our turning rate is limited, so turn as far as we can.

Rule  4  if   (and (bearing is [2 .. 6]))
        then (and (turn is [-60]))
        action TURN  strength 1.0
        action SPEED strength 0

Rule  5  if   (and (bearing is [6 .. 10]))
        then (and (turn is [60]))
        action TURN  strength 1.0
        action SPEED strength 0
```

3.3.2.2. Evaluation

Each plan is evaluated by invoking CPS, using the given plan as rule memory. CPS executes a fixed number of episodes, and returns the average payoff as the fitness for the plan. The updated strengths of the rules are also returned to the learning module. Each episode begins with randomly selected initial conditions, and thus represents a single sample of the performance of the plan on the space of all possible initial condition of the world model. (Grefenstette, Ramsey and Schultz, 1990) examined several important questions about this sampling procedure and in particular: What if the distribution used for evaluating plans differs from the true distribution of initial conditions that arise in the actual task environment? In summary, this earlier work indicated that it is important for the simulation model to include more variability and noise that the actual environment. In this study, noise is included in the sensors, and each initial environment is comprised of a random mine field.

3.3.2.3. Selection

Plans are selected for reproduction on the basis of their overall fitness scores returned by CPS. Using the notion of "survival of the fittest", plans that perform well get to produce more offspring (i.e. plans) for the next generation. The topic of reproductive selection in genetic algorithms is discussed in the literature (Goldberg, 1989; Grefenstette & Baker, 1989). In SAMUEL, the *fitness* of each plan is defined as the difference between the average payoff received by the plan and some baseline performance measure. The baseline is adjusted to track the mean payoff received by the population, minus one standard deviation. The baseline is adjusted slowly to provide a moderately consistent measure of fitness. Plans whose payoff fall below the baseline are assigned a fitness measure of 0, resulting in no offspring. This mechanism appears to provide a reasonable way to maintain consistent selective pressure toward higher performance.

3.3.2.4. Genetic Operators

Selection alone merely produces clones of high performance plans. CROSSOVER works in concert with selection to create plausible new plans. In SAMUEL, CROSSOVER treats rules as indivisible units. Since the rule ordering within a plan is irrelevant, the process of recombination can be viewed as simply selecting rules from each parent to create an offspring plan. Many genetic algorithms permit recombination within individual rules as a way of creating new rules (Smith, 1980; Schaffer, 1984; Holland, 1986). While such operators are easily defined for SAMUEL's rule language (Grefenstette, 1989), we prefer to use CROSSOVER solely to explore the space of rule combinations, and leave rule modification to other operators (i.e., SPECIALIZE, GENERALIZE, CREEP, MERGE and MUTATION).

In SAMUEL, CROSSOVER assigns each rule in two selected parent plans to one of two offspring plans. CROSSOVER attempts to cluster rules that are temporally related before assigning them to offspring. The idea is that rules that fire in sequence to achieve a successful rendezvous should be treated as a group during recombination, in order to increase the likelihood that the offspring plan will inherit some of the better behavior patterns of its parents. Of course, the offspring may not behave identically to either one of its parents, since the probability that a given rule fires depends on the context provided by all the other rules in the plan.

CROSSOVER is restricted so that no plan receives duplicate copies of the same rule. This restriction seems to be necessary to avoid plans containing a large number of copies of the same rule. Given the conflict resolution algorithm in CPS, duplicate copies have no effect on the performance of the plan (other than to take up space), so this restriction seems reasonable. Note that since every rule occurring in either parent is inherited by one of the offspring, any rule that occurs in both parents also occurs in both offspring. The effect is that small groups of rules that are associated with high performance propagate through the population of plans, and serve as building blocks for new plans.

While CROSSOVER operates on the entire population of plans, recombining rules among the plans, SAMUEL also includes six unary operators that modify the rules within a single strategy: MUTATION, CREEP, SPECIALIZE, GENERALIZE, MERGE, and DELETE. Unlike previous versions of the system, SAMUEL has now adopted the policy that all of these operators except DELETE are *creative*. All modifications are made on a new copy of the original rule, and the altered rule is added into the plan, where it will compete at the rule level with the rule from which it was created. A rule survives intact unless it is explicitly deleted or lost when its strategy is not selected for reproduction. This policy allows a much more aggressive application of rule modification operators with little damage if the changes are maladaptive. Each of these operators will now be discussed.

The genetic operator MUTATION introduces a new rule by making random changes to a copy of an existing rule. For example, MUTATION might alter a condition within a rule from `(asonar_4 150 200)` to `(asonar_4 10 200)`, or it might change the action from `(turn 45)` to `(turn -60)`. The operator CREEP is similar to MUTATION, except that it only makes small changes, e.g. from `(asonar_4 150 200)` to `(asonar_4 140 200)`. This operator ''creeps'' a value the smallest increment possible for a particular sensor or action.

The SPECIALIZE operator was described earlier in Section 3.3.2.1 during the discussion of population initialization. This operator is applied when general rules fire in successful episodes. The operator creates a new rule whose left hand side more closely matches the sensor values existing at the time the general rule fired, and whose right hand side more closely matches the action that was actually taken.

The GENERALIZE operator creates rules that are more general versions of overspecialized rules. GENERALIZE can be applied when a rule fires because of a partial match during a successful episode. A partial match occurs when no rule fully matches the current sensors, and the rule that most closely matches is selected. This operator creates a rule that will match when a similar situation occurs again by generalizing the conditions enough to match the sensor readings that were active when the rule fired.

The MERGE operator creates a new rule from two high-strength rules that specify the same action. The new rule matches any situation that was originally matched by either of the two original rules. Together, the MERGE operator with the DELETE operator (described next), help to eliminate overspecialized rules from the strategy.

DELETE is the only operator that can remove rules from a strategy. A rule may be deleted if the rule has not fired recently, the rule has low strength, or the rule is subsumed by another rule with higher strength.

4. Experiments and Results

The learning curve shown here reflects our assumptions about the methodology of simulation-assisted learning. At periodic intervals (10 generations in the current experiments), a single plan is extracted from the current population to represent the learning system's current hypothetical plan. This plan is tested for 100 randomly chosen problem episodes. The plot shows the sequence of results of testing, using the current plans periodically extracted from the learning system. The assumption is that the current best plan can be extracted from the learning system and used in the real world while the learning system continues looking for a better hypothesis.

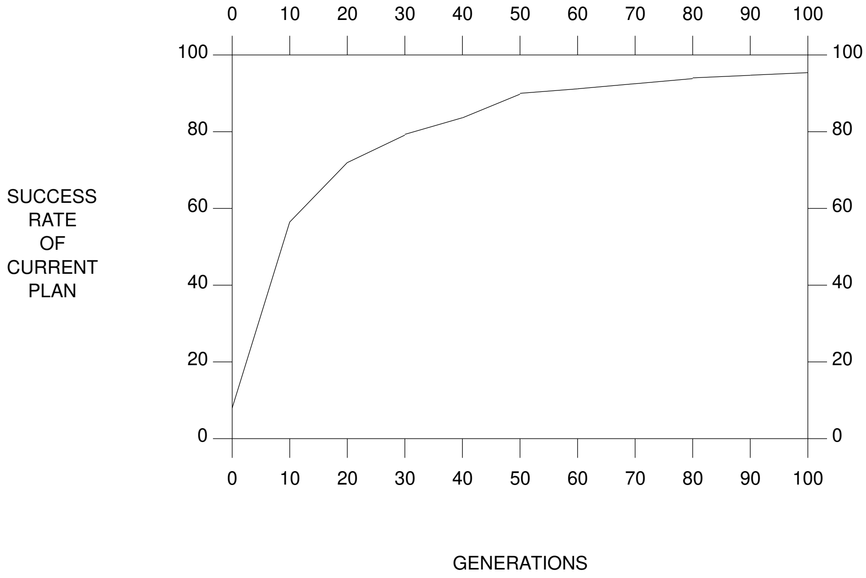

Fig. 3. Learning curve for 25 mines.

In the first experiment, the system was initialized with the rules discussed and shown in section 3.3.2.1. Each episode, the AUV was placed centered before a randomly generated mine field with a density of 25 mines. The rendezvous area was placed at a random point on the other side of the mine field. SAMUEL was then run for 100 generations. Each evaluation of a plan is the average over 20 episodes, as discussed earlier. Figure 3 shows the learning curve for the current hypothesis as discussed above. Since a stochastic process is involved, the experiment was repeated ten times, and the results averaged in the graph. Before learning, with just the initial rules by themselves, the AUV will rendezvous with the goal just eight out of 100 times. Note that after 100 generations, the performance has reached 96 percent.

In order to test the robustness of the learned strategy, the best strategy learned over the whole experiment was extracted and then used, without learning, in other environments. The results are summarized in Table 1. When presented with an environment with double the mine density (50 mines), the rules learned with 25 mines still produced 46 percent performance. The learned rule set was then tested in scenarios with 25 *drifting* mines. The mines made random walks at a speed of seven units. In this environment, the AUV achieved 91 percent performance. Next, the strategy was tested in an environment with 50 drifting mines, again with a speed of seven units. In this case, the AUV reached 41 percent performance.

	before learning (uses initial rules)	learned with 25-mine scenarios
25 mines	8%	96%
50 mines	1%	46%
25 moving mines	1%	91%
50 moving mines	0%	41%

Table 1. Trained with 25 mines; tested in various situations.

Although the system is an opportunistic learning, the results indicate that generally useful reactive rules for this task were learned by the system. Notice that the performance of the learned strategy was most affected by the mine density. One reason for this is the relatively low resolution of the sonar and particularly by the fact that the AUV had a greater turning radius than angle of view with the sonar, allowing the AUV to turn into mines just outside of its view. A newer model does not allow the AUV to turn further than its sonar coverage.

The performance of the strategy is not as affected by environments with mines that move. Only a small degradation is introduced when the mines are allowed to move, given the same density of mines. This is *not* surprising when you consider that the strategy is *reactive*, and this points out one of the drawbacks to projective planning: In this environment, a global path can almost never be constructed.

5. Conclusion and New Research

SAMUEL has demonstrated in several different domains, including mine avoidance and local navigation, tracking, and evasion, that robust reactive strategies can be learned. These strategies are expressed in a high-level language, and can be used in reactive systems where real-time response is an important capability. SAMUEL is an attempt to automate the process of learning the rules needed in a reactive system, particular since reactive rules are very difficult to construct by hand.

This research is continuing along several fronts. In the domain described in this paper, we are obtaining a high fidelity three-dimensional AUV simulator and high fidelity models of active sonar to see if the system will scale up well. In addition to the AUV domain, we are also continuing to look at strategies for other important behaviors for autonomous vehicles.

References

Booker, L. B. (1985). Improving the performance of genetic algorithms in classifier systems. *Proceedings of the International Conference Genetic Algorithms and Their Applications* (pp. 80-92). Pittsburgh, PA.

De Jong, K. A. (1975). *Analysis of the behavior of a class of genetic adaptive systems.* Doctoral dissertation, Department of Computer and Communications Sciences, University of Michigan, Ann Arbor.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading: Addison-Wesley.

Gordon, D. F. and J. J. Grefenstette (1990). Explanations of empirically derived reactive plans. *Proceedings of the Seventh International Conference on Machine Learning.* Austin, TX: Morgan Kaufmann.

Grefenstette, J. J. (1988). Credit assignment in rule discovery system based on genetic algorithms. *Machine Learning, 3(2/3),* (pp. 225-245).

Grefenstette, J. J. (1989). A system for learning control plans with genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms.* Fairfax, VA: Morgan Kaufmann. (pp. 183-190).

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

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University Michigan Press.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R.S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Michalski, R. S. (1983). A theory and methodology for inductive learning. *Artificial Intelligence, 20(2),* (pp. 111-161).

Mitchell, T. M., S. Mahadevan and L. Steinberg (1985). LEAP: A learning apprentice for VLSI design. *Proc. Ninth IJCAI, (pp. 573-580). Los Angeles: Morgan Kaufmann.*

Ramsey, C. L., Alan 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.* Austin, TX: Morgan Kaufmann (pp. 211-215).

Riolo, R. L. (1988). *Empirical studies of default hierarchies and sequences of rules in learning classifier systems,* Doctoral dissertation, Doctoral dissertation, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor.

Schaffer, J. D. (1984). *Some experiments in machine learning using vector evaluated genetic algorithms,* Doctoral dissertation, Department of Electrical and Biomedical Engineering, Vanderbilt University, Nashville.

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

Wilson, S. W. (1987). Classifier systems and the animat problem. *Machine Learning, 2(3),* (pp. 199-228).