A Multistrategy Learning Scheme for Assimilating Advice in Embedded Agents

Diana F. Gordon
Naval Research Laboratory, Code 5514
Washington, D.C. 20375
gordon@aic.nrl.navy.mil

Devika Subramanian
Cornell University
Ithaca, N.Y. 14853
devika@cs.cornell.edu

Abstract

The problem of designing and refining task-level strategies in an embedded multiagent setting is an important unsolved question. To address this problem, we have developed a multistrategy system that combines two learning methods: operationalization of high-level advice provided by a human and incremental refinement by a genetic algorithm. The first method generates seed rules for finer-grained refinements by the genetic algorithm. Our multistrategy learning system is evaluated on two complex simulated domains as well as with a Nomad 200 robot.

Key words: advice, operationalize, genetic algorithms

1 Introduction

The problem of designing and refining task-level strategies in an embedded multi-agent setting is an important unsolved question. To address this problem, we have developed a multistrategy learning system that combines two learning methods: operationalization of high-level advice provided by a human, and incremental refinement by a genetic algorithm (GA). We define advice as a recommendation to achieve a goal under certain conditions. Advice is considered to be operationalized when it is translated into stimulus-response rules in a language directly usable by the agent. Operationalization generates seed rules for finer-grained refinements by a GA.

The long term goal of the work proposed here is to develop task-directed agents capable of acting, planning, and learning in worlds about which they have incomplete information. These agents refine factual knowledge of the world they inhabit, as well as strategic knowledge for achieving their tasks, by interacting with the world. *Agent knowledge acquisition* is very difficult for the same reasons that knowledge acquisition for expert systems is. It is preferable to assimilate high level knowledge because the process of entering low level domain-specific knowledge for an agent is a costly, tedious, and error-prone process. The additional challenge for agent knowledge acquisition comes from the fact that the agent must dynamically update its knowledge through interactions with its environment.

There are two basic approaches to constructing agents for dynamic environments. The first decomposes the design into stages: a parametric design followed by refinement of the parameter values using feedback from the world in the context of the task. Several refinement strategies have been studied in the literature: GAs (Odetayo and McGregor, 1989), neural-net learning (Clouse and Utgoff, 1992), statistical learning (Maes and Brooks, 1990), and reinforcement learning (Mahadevan and Connell, 1991). The second, more ambitious, approach (Grefenstette *et al.*, 1990; Tesauro, 1992) is to acquire the agent knowledge directly from example interactions with the environment. The success of this approach is tied to the efficacy of the credit assignment procedures, and whether or not it is possible to obtain good training runs with a knowledge-impoverished agent.

We have adopted the first approach. The direction we pursue is to compile an initial parametric agent using high-level strategic knowledge (e.g., advice) input by the user, as well as a body of general (not domain-specific) spatial knowledge in the form of a *Spatial Knowledge Base* (SKB). The SKB contains qualitative rules about movement in space. Example rules in our SKB are "If something is on my side, and I turn to the other side, I will not be facing it" and "If I move toward something it will get closer". This SKB is portable because it is applicable to a variety of domains where qualitative spatial knowledge is important. A similar qualitative knowledge base was constructed by (Mitchell, 1987) for the task of pushing objects in a plane. Since the knowledge provided to our agent will often be imperfect (incomplete and incorrect), this knowledge is refined by a GA.

First we describe our deductive operationalization process and the nature of the parameterization adopted for our agent. Then we describe the inductive (GA) refinement stage and compare our multistrategy approach with one that is purely inductive. Before we present the details of the method, we characterize the class of environments and tasks for which we have found this decomposition (of an agent design into an initial parametric stage and subsequent refinement stage) to be effective.

- Environment characteristics: Complete models of the dynamics of the environment in the form of differential equations or difference equations, or discrete models like STRIPS operators, are unavailable. An analytical design that maps the percepts of an agent to its actions (e.g., using differential game theory or control theory) in these domains is not possible without a complete model. Even if a model were available, standard methods for deriving agents are extensional and involve exploration of the entire state space. They fail because the domains considered in this paper have of the order of a 100 million states.
- Task characteristics: Task are sequential decision problems: payoff is obtained at the end of a sequence of actions and not after individual actions. Examples are pursuit-evasion in a single or multi-pursuer setting and navigating in a world with moving obstacles. The tasks are typically multi-objective in nature: for instance, minimize energy consumption while maximizing the time till capture by the pursuer.
- Agent characteristics: The agent has imperfect sensors. Imperfections occur in the form of noise, as well as incompleteness (all aspects of the state of the world cannot be sensed by our agent, a problem called *perceptual aliasing* in Whitehead and Ballard, 1990). Stochastic differential game theory has methods for deriving

agents with noisy sensors, but it requires detailed models of the noise as well as a detailed model of the environment and agent dynamics.

The action set of the agents and the values taken on by sensors are discrete and can be grouped into qualitative equivalence classes. This is the basis for the design of the parametric agent. A similar intuition underlies the design of fuzzy controllers that divide the space of sensor values into a small set of classes described by linguistic variables.

In such domains, human designers derive an initial solution by hand and use numerical methods (typically very dependent on the initial solution) to refine their solution. Our ultimate objective is to automate the derivation of good initial solutions by using general knowledge about the environment, task, and agent characteristics and thus provide a better starting point for the refinement process. We begin with the SKB and advice.

2 Compiling Advice

Our operationalization method compiles high-level domain-specific knowledge (e.g., advice) and spatial knowledge (SKB) into low-level reactive rules directly usable by the agent. The compilation performs *deductive concretion* (Michalski, 1991) because it deductively converts abstract goals and other knowledge into concrete actions. An important question is why we adopt a deductive procedure for operationalization of advice. At this time, we are able to achieve operationalization without resorting to any form of non-deductive inference. This is because for the domains studied in this paper, the SKB is complete enough to yield good parametric designs with deductive inference alone. We expect that as we expand our experimental studies to cover more domains, the incompleteness of the SKB will force us to adopt more powerful operationalization methods.

We assume all the knowledge provided is operationalized immediately; however, it need not be applied immediately. We precompile all high-level knowledge because the agent will apply it in a time-critical situation. The learning cost prior to execution is not a concern, but the reaction time of the agent is critical. Therefore it is best to have all knowledge in a quickly usable (operational) form. Compiled rules are fully operational, whereas advice and SKB rules have at least some nonoperational elements. The user specifies what is operational for the agent.

Compilation uses two stacks: a GoalStack and an (operational condition) OpCondStack. Three types of knowledge are initially given to the compiler: facts, nonoperational rules (abbreviated *nonop rules*), and advice. A user can provide any of the three. The SKB has only facts and nonop rules. The output from compilation is a set of *op rules* directly usable (i.e., operational) by the agent. Facts have the form:

Predicate(X_1,...,X_n).

Nonop rules have the form:

IF cond AND ... AND cond <AND action> THEN goal.

Anything in angle brackets is optional. The portion preceding the "THEN" is the rule *antecedent*, and the portion following the "THEN" is the rule *consequent*. A nonop rule consequent is a single goal. The syntax for a goal is "function(X_1,...,X_n) = value" or "predicate(X_1,...,X_n)". Each X_i is an object (e.g., an agent). The syntax for a "cond" (condition) or "action" in the rule antecedent is the same as for goals. Advice has the form:

<IF cond AND ... AND cond THEN> ACHIEVE goal.

Although advice has a similar syntax to nonop rules, its interpretation differs. Advice recommends achieving the given "goal" under the given "conds". A nonop rule, on the other hand, states that the given "goal" will be achieved if the given "conds" (and "action") occur. Compilation results in stimulus-response op rules of the form:

IF cond AND ... AND cond THEN action.

The conditions of an op rule are sensor values detectable by the agent. The action can be performed by the agent. Our compilation algorithm is in Figure 1.

```
Push advice on GoalStack: goal followed by conditions.
Initialize OpCondStack to be empty and invoke Compile(GoalStack,OpCondStack).

Procedure Compile (GoalStack, OpCondStack)
IF GoalStack is not empty THEN
  g ← pop(GoalStack);
  CASE g:
    1. g is an operational condition:
      Push(g, OpCondStack);
      Compile(GoalStack, OpCondStack)
    2. g matches a fact:
      Compile(GoalStack,OpCondStack)
    3. g is nonoperational:
      FOREACH nonop rule R_i in
        knowledge base whose consequent
        matches g DO
          Push(antecedent(R_i), GoalStack);
          Compile(GoalStack,OpCondStack)[1]
    4. g is an operational action
      Form a new op rule from the contents
        of OpCondStack and g;
      Clear OpCondStack
ELSE Clear OpCondStack
```

FIG. 1. Algorithm for operationalizing advice.

[1] To prevent cycles, the last nonop rule used in step 3 is marked as "used" so that it will not be used again.

This algorithm takes advice and backchains through the SKB and user-provided nonop rules until an operational action is found. Once an operational action is found, it pops back up the levels of recursion, attaching conditions along the way, to form a new reactive agent op rule.

Let us examine a simple example of how this algorithm operates, as shown in Figures 2 and 3. *Heading* (X,Y) refers to the direction of motion of Y relative to X, and *bearing* (X,Y) refers to the direction of Y relative to X. Suppose the advice is "IF speed(adversary) = low THEN ACHIEVE heading(adversary, agent) = not(head-on)" (i.e., avoid adversary). Figure 2 shows how SKB nonop rules match this advice for backchaining, thereby creating an "and" tree. Anything preceded by a "*" is operational.

Figure 3 shows this algorithm in operation. Note that stacks grow downward. The algorithm begins by pushing the advice goal, followed by the advice condition, on the GoalStack. It then calls procedure Compile, which moves the advice condition to the OpCondStack because it is operational. The advice goal is not operational. In our example, the advice goal can be unified with the goal of SKB RULE1, which states "IF bearing(agent, adversary) = right AND turn(agent) = left THEN heading(adversary, agent) = not(head-on)". The condition and action of RULE1 are pushed on the GoalStack. Because the condition of RULE1 is operational, it is moved to the OpCondStack.

At this point, the action of RULE1 is at the top of the GoalStack, and it is operational, so we can create an op rule. The conditions from the OpCondStack are added to the action. This creates an op rule that states "IF speed(adversary) = low AND bearing(agent,

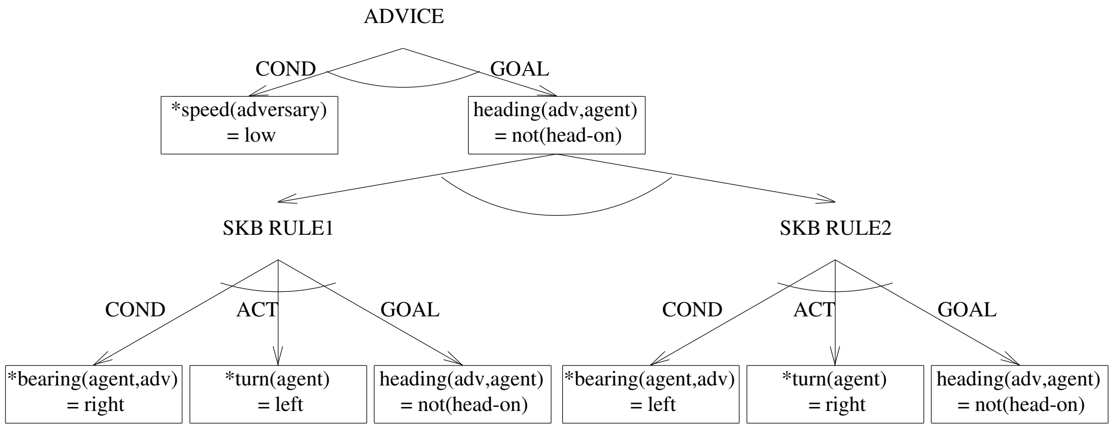

FIG. 2. Graph of example.

adversary) = right THEN turn(agent) = left". Both stacks are cleared. The algorithm continues similarly to generate a second op rule that states "IF speed(adversary) = low AND bearing(agent, adversary) = left THEN turn(agent) = right" from SKB RULE2 (see Figure 2).

Next, we apply a conversion from qualitative to quantitative op rules. The rules are given default quantitative ranges. For example, if "speed" has two values, "slow" and "fast", we bisect the range of all possible values into two subranges. Then, we allow the system to improve this initial choice of quantitative

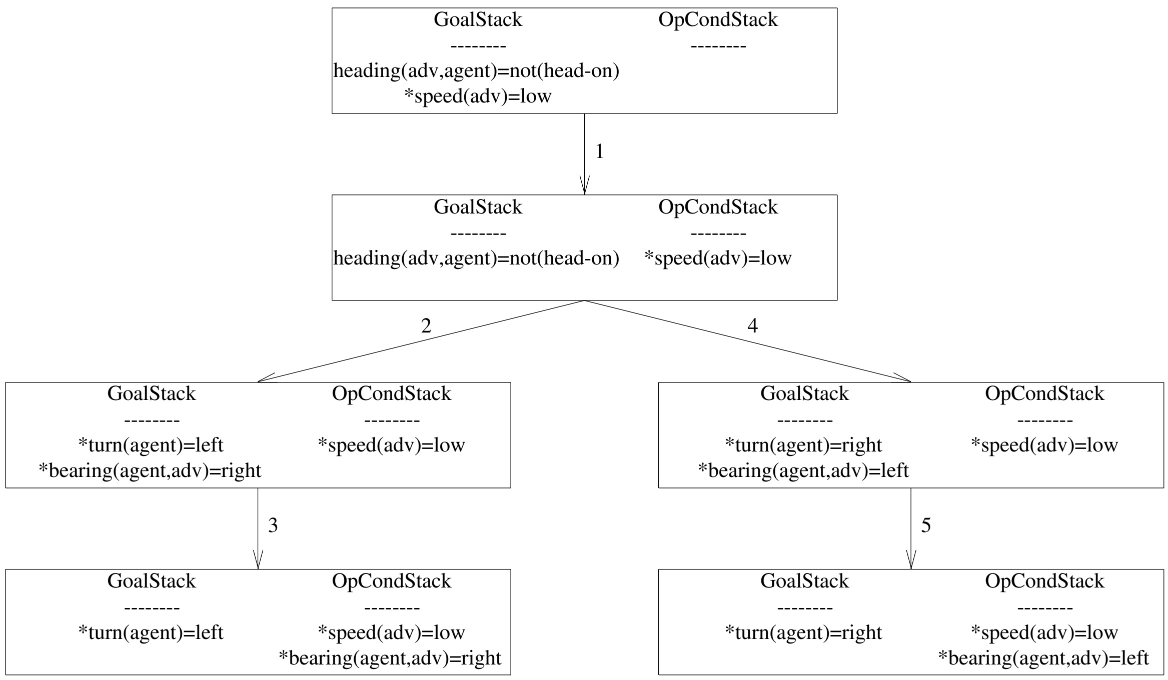

FIG. 3. Example of the compilation algorithm.

ranges by using a GA to refine the initial ranges while interacting with the environment.

3 Executing and Refining Advice

The system we use to refine and apply the op rules derived from our compiled advice is the SAMUEL reactive planner (Grefenstette *et al.*, 1990). We have chosen SAMUEL because this system has already proven to be highly effective for refining rules on complex domains (Grefenstette *et al.*, 1990; Schultz and Grefenstette, 1990). SAMUEL adopts the role of an agent in a multiagent environment in which it senses and acts. This system has two major components: a performance module and a learning module. Section 4.2 explains how performance interleaves with learning in our experiments.

The performance module, called the Competitive Production System (CPS), interacts with a simulated or real world by reading sensors, setting effector values, and receiving payoff from a critic. CPS performs matching and conflict resolution on the set of op rules. This performance module follows the match/conflict-resolution/act cycle of traditional production systems. Time is divided into *episodes*: the choice of what constitutes an episode is domain-specific. Episodes begin with random initialization and end when a critic provides payoff. At each time step within an episode, CPS selects an action using a probabilistic voting scheme based on rule strengths. All rules that match (or partially match - see Grefenstette *et al.*, 1990) the current state bid to have their actions fire. The actions of rules with higher strengths are more likely to fire. If the world is being simulated, then after an action fires, the world model is advanced one simulation step and sensor readings are updated.

CPS assigns credit to individual rules based on feedback from the critic. At the end of each episode, all rules that suggested actions taken during this episode have their strengths incrementally adjusted to reflect the current payoff. Over time, rule strengths reflect the degree of usefulness of the rules.

SAMUEL's learning module is a genetic algorithm. GAs are motivated by simplified models of heredity and evolution in the field of population genetics (Holland, 1975). GAs evolve a population of individuals over a sequence of *generations*. Each individual acts as an alternative solution to the problem at hand, and its *fitness* (i.e., potential worth as a solution) is regularly evaluated. During a generation, individuals create *offspring* (new individuals). The fitness of an individual probabilistically determines how many offspring it can have. Genetic operators, such as *crossover* and *mutation*, are applied to the offspring. Crossover combines elements of two individuals to form new individuals; mutation randomly alters elements of a single individual. In SAMUEL, an individual is a set of op rules. In addition to genetic operators, this system also applies non-genetic knowledge refinement operators, such as "generalize" and "specialize", to op rules within a rule set.

The interface between our compilation algorithm and the SAMUEL system is straightforward. The output of our compilation algorithm is a set of op rules for the SAMUEL agent. Because the op rules may be incomplete, a *random rule* is added to this rule set. The random rule recommends performing a random action under any conditions. This rule set, along with CPS and the GA learning component for improving the rules, is our initial agent.

4 Evaluation

We have not yet analyzed the cost of our compilation algorithm. The worst case cost appears to be exponential, because the STRIPS planning problem (which is P-space complete) can be reduced to it. In the future, we plan to investigate methods to reduce this cost for complex realistic problems. Potential methods include: (1) attaching a likelihood of occurrence onto advice, which enables the agent to prioritize which advice to compile first if time is limited, (2) tailoring the levels of generality and abstraction of the advice to suit the time available for compilation (e.g., less abstract advice is closer to being operational and therefore requires less compilation time), and (3) generating a parallel version of the algorithm.

We *have* evaluated our multistrategy approach empirically. We focus on answering the following questions:

- Will our advice compilation method be effective for a reactive agent on complex domains?
- Will the coordination of multiple learning techniques lead to improved performance over using any one learning method? In particular, we want the GA to improve the success rate of the compiled advice, and the advice to improve the convergence rate of the GA. An improved convergence rate is useful when learning time is limited.
- Can we construct a portable SKB?

4.1 Domain characteristics

To address our questions, we have run experiments on two complex problems: Evasion and Navigation. Our choice of domains is motivated by the results of Schultz and Grefenstette (1990), who have obtained large performance improvements by initializing the GA component of SAMUEL with hand-coded op rules in these domains. Their success has inspired the work described here. Our objective is to automate their tedious manual task, and the work described here is one step toward our goal.

Both problems are two-dimensional simulations of realistic tactical problems. However, our simulations include several features that make these two problems sufficiently complex to cause difficulties for more traditional control theoretic or game theoretic approaches (Grefenstette *et al.*, 1990):

- A weak domain model. The learner has no initial model of other agents or objects in the domain. Most control theoretic and game theoretic models make worst case assumptions about adversaries. This yields poor designs in the worlds we consider because we have statistical rather than worst case adversaries.
- Incomplete state information. The sensors are discrete, which causes a many-to-one mapping and perceptual aliasing.
- A large state space. The discretization of state space makes the learning problem combinatorial. In the Evasion domain, for instance, over 25 million distinct feature vectors are observed, each requiring one of nine possible actions, giving a total of over 225 million maximally specific condition-action pairs.
- Delayed payoff. The critic only provides payoff at the end of an episode. Therefore a credit assignment scheme is required.
- Noisy sensors. Gaussian noise is added to all sensor readings. Noise consists of a random draw from a normal distribution with mean 0.0 and standard deviation equal to 5% of the legal range for the corresponding sensor. The value that results is discretized according to the

defined granularity of the sensor. A 5% noise level is sufficient to degrade SAMUEL's performance.

4.2 Experimental design

Two sets of experiments are performed on each of the two domains. Perception is noise-free for the first set, but noisy for the second. The primary purpose of the first set is to address our question about the effectiveness of our advice compilation method alone, without GA refinement. Facts, nonop rules, advice, and the random rule are given the compiler and the output is a set of op rules. This rule set is given to SAMUEL's CPS module and applied within the simulated world model. The baseline performance with which these rules are compared is the random rule alone. These experiments measure how the average (over 1000 episodes) success rate of the compiled rules compares with that of the baseline as problem complexity increases. Statistical significance of the differences between the curves with and without advice are presented. Significance is measured using the large-sample test for the differences between two means.

The primary purpose of the second set of experiments is to address our question about the effectiveness of the multistrategy approach (compilation followed by GA refinement). Facts, nonop rules, and advice are given to the compiler and the output is a set of op rules. This rule set, plus the random rule, becomes every individual in SAMUEL's initial GA population, i.e., it seeds the GA with initial knowledge. The baseline performance with which these rules are compared is SAMUEL initialized with every individual equal to just the random rule. In either case, GA learning evolves this initial population. In other words, we compare the performance of advice seeding the GA with GA learning alone (i.e., random seeding). *Random seeding produces an initially unbiased GA search; advice initially biases the GA search - hopefully into favorable regions of the search space.*

In this second set of experiments, performance interleaves with GA refinement. SAMUEL runs for 100 generations using a population size of 100 rule sets. Every 5 generations, the "best" (in terms of success rate) 10% of the current population are evaluated over 100 episodes to choose a single plan to represent the population. This plan is evaluated on 1000 randomly chosen episodes and the average success is calculated. This entire process is repeated 10 times and the average success rate over all 10 trials is found. The curves in our graphs plot these averages. For this set of experiments, statistical significance is measured using the two-sample t-test, with adjustments as required whenever the F statistic indicates unequal variances.

We add sensor noise, as defined in Section 4.1, for this second set of experiments because GAs can learn robustly in the presence of noise (Grefenstette *et al.*, 1990). Two performance measures are used: the success rate and the convergence rate. The convergence rate is defined as the number of GA generations required to achieve and maintain an $n\%$ success rate, where n is different for each of the two domains. The value of n is set empirically.

4.3 Evaluation on the Evasion problem

Our simulation of the Evasion problem is partially inspired by (Erikson and Zytkow, 1988). This problem consists of an agent, which is controlled by SAMUEL, that moves in a two-dimensional world with a single adversary pursuing the agent. The agent's objective is to avoid contact with the adversary for a bounded length of time. Contact implies the agent is captured by the adversary. The problem is divided into episodes that begin with the adversary approaching the agent from a random direction. The adversary initially travels faster than the agent, but is less maneuverable (i.e., it has a greater turning radius). Both the agent and the adversary gradually lose speed when maneuvering, but only the adversary's loss is permanent. An episode ends when either the adversary captures the agent (failure) or the the agent evades the adversary (success). At the end of each episode, a critic provides full payoff for successful evasion and partial payoff otherwise, proportional to the amount of time before the agent is captured. The strengths of op rules that fired are updated in proportion to the payoff.

The agent has the following operational sensors: time, last agent turning rate, adversary speed, adversary range, adversary bearing, and adversary heading. The agent has one operational action: it can control its own turning rate. For further detail, see (Grefenstette *et al.*, 1990).

In our experiments, we provide the following domain-specific knowledge:

FACTS

Chased_by(agent, adversary).
Moving(agent). Moving(adversary).

NONOP RULES

IF chased_by(X, Y) AND range(X, Y) = close AND turn(X) = Z THEN turn(Y) = Z.
IF range(X, Y) = not(close) AND heading(Y, X) = not(head_on) THEN avoids(X, Y).
IF turn(adversary) = hard THEN decelerates(adversary).

ADVICE

IF speed(adversary) = high THEN ACHIEVE decelerates(adversary).
IF speed(adversary) = low THEN ACHIEVE avoids(agent, adversary).

We also include knowledge of the agent's operational sensors and actions as facts.

Ten SKB nonop rules are used (they are instantiations of the two rules described in English in the introduction of this paper). Although room does not permit listing them all, some examples are:

IF bearing(X,Y) = right AND turn(X) = left THEN heading(Y,X) = not(head_on).
IF bearing(X,Y) = left AND moving(X) AND turn(X) = left THEN range(X,Y) = close.

From our input and our SKB rules, the compilation method of Section 2 generates op rules. The sensor values of these rules are translated from qualitative values to default quantitative ranges. For example, "bearing = left" is translated into "bearing = [6..12]", where the numbers correspond to a clock, e.g., 6 means "6 o'clock". Every new rule is given a strength of 1.0 (the maximum). The final op rule set includes rules such as:[2]

IF speed(adversary) = [700..1000] AND range(agent, adversary) = [0..750] THEN turn(agent) = hard-left.

The total number of op rules generated from our advice is 16.

We begin our experiments by addressing the first question, which concerns the

[2] To generate a few of these rules, we used a variant of our compilation algorithm. We omitted a description of this variation for the sake of clarity. See (Gordon and Subramanian, 1993) for details.

effectiveness of our advice-taking method. We do not use the GA. Problem difficulty is varied by adjusting a "safety" envelope around the agent. The "safety" envelope is the distance at which the adversary can be from the agent before the agent is considered captured by the adversary.

Figure 4 shows how the performance (averaged over 1000 episodes) of these op rules compares with that of just the random rule. All of the differences between the means are statistically significant (using significance level $\alpha = 0.05$). From Figure 4 we see that from difficulty levels 80 to 120, the agent is approximately twice as successful with advice than without it. This is a 100% performance advantage. Furthermore, for levels 120 to 160, the agent is about four times more effective with advice. For levels 160 to 200, the agent is an order of magnitude more effective with advice. We conclude that as the difficulty of this problem increases, the advice becomes more helpful. These results answer our first question: our advice compilation method is effective on this domain.

We address the second question about multistrategy effectiveness by combining the compiled advice with GA refinement. Figure 5 shows the results of comparing the performance of the GA with and without advice. The "safety" envelope is fixed at 100 (chosen arbitrarily) and noise is added to the sensors. For this domain, the convergence rate is the number of GA generations required to maintain a 60% success rate.

Figure 5 shows that in a small amount of time (less than 10 generations), the GA provides a substantial (50%) improvement in success rate. However, the convergence rate with and without advice is the same. Furthermore, although prior to 50 generations the differences between the means are not statistically significant (other than the initial boost provided by the advice), after 50 generations the improvement without advice over advice is significant ($\alpha = 0.05$). Therefore, this domain fails to demonstrate the superiority of combining strategies. We conjecture that our advice is not properly biasing the GA into the most favorable regions of the search space.

Note that these results illuminate the tight coupling that exists between the two strategies in our multistrategy system. Consistently high performance depends not only on successful compilation of advice. It also depends on how the advice initially biases the GA. A ripe area for future research is to experimentally determine effective initialization methods.

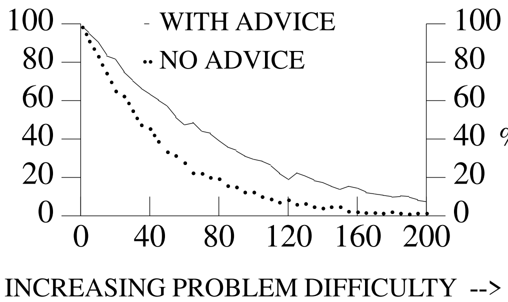

FIG. 4 Evasion domain.

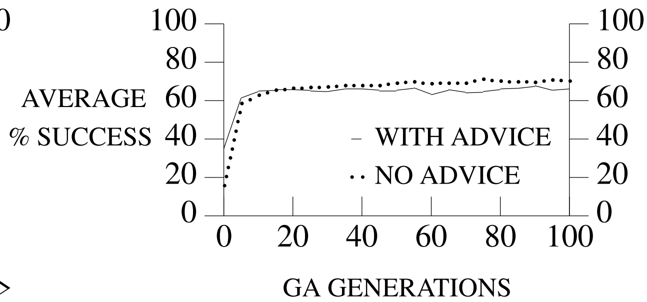

FIG. 5. Evasion domain with GA.

4.4 Evaluation on the Navigation problem

In the Navigation domain, our agent is again controlled by SAMUEL in a two-dimensional simulated world. The agent's objective is to avoid obstacles and navigate to a stationary target with which it must rendezvous before exhausting its fuel (implemented as a bounded length of time for motion). Each episode begins with the agent centered in front of a randomly generated field of obstacles with a specified density. An episode ends with either a rendezvous at the target location (success) or the exhaustion of the agent's fuel or a collision with an obstacle (failure). At the end of an episode, a critic provides full payoff if the agent reaches the target, and partial payoff otherwise, depending on the agent's distance to the goal.

The agent has the following operational sensors: time, the bearing of the target, the bearing and range of an obstacle, and the range of the target. The agent has two operational actions: it can control its own turning rate and its speed. For further detail, see Schultz and Grefensette (1992).

We provide the following domain-specific knowledge (in addition to a list of operational sensors and actions):

FACTS

Moving(agent).

NONOP RULES

IF range(*X*, *Y*) = not(close) AND heading(*Y*, *X*) = not(head_on) THEN avoids(*X*, *Y*).

ADVICE

IF range(agent, obstacle) = not(close) THEN ACHIEVE range(agent, target) = close.

IF range(agent, obstacle) = close THEN ACHIEVE avoids(agent, obstacle).

ACHIEVE speed(agent) = high.

The *same* 10 SKB nonop rules used for Evasion are again used for this domain, which confirms the portability of our SKB, thus addressing the third question. A total of 42 op rules are generated.[3]

Again, we address the first question by using SAMUEL without the GA. The success rate is averaged over 1000 episodes. Without advice, the average success rate is 0% because this is a very difficult domain. Figure 6 shows how we improve the success rate to as much as 90% by using our advice on this domain. At all but the last few points, the differences between the means are statistically significant (α = 0.05). When we vary the number of obstacles, performance follows a different trend than for the Evasion domain. By far the greatest benefit of the advice occurs when there are few obstacles. Performance is 10 times better when there is only one obstacle, for example. The advantage drops as the problem complexity increases. After difficulty level 80, advice no longer offers any benefit.

Our experiments on both domains confirm that our advice compiler can be effective, however, they also indicate that the usefulness of advice may be restricted to a particular range of situations. Another learning task, which we are currently exploring, would be to identify this range and add additional conditions to the advice.

We address the second question by comparing the performance of the GA with and without advice. Noise is added. Figure 7 shows the results. All differences between the means are statistically significant (α = 0.05). Here, the number of obstacles is fixed at five (chosen arbitrarily). For this domain,

[3] We were able to decrease the number of op rules to 9 by making one careful qualitative to quantitative mapping choice.

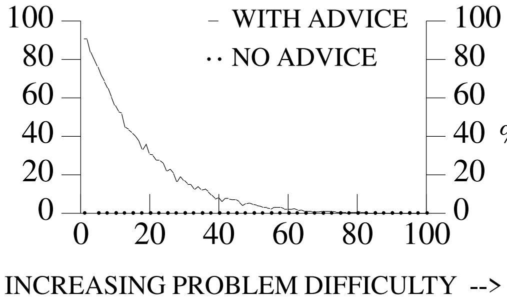

FIG. 6. Navigation domain.

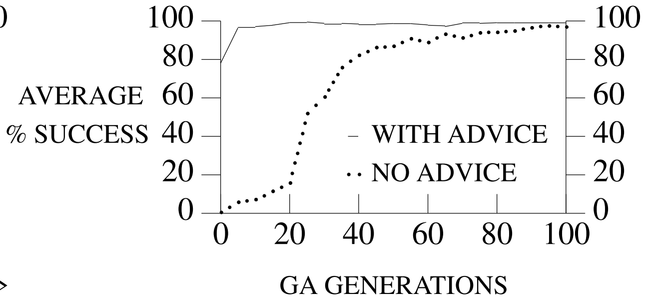

FIG. 7. Navigation domain with GA.

the convergence rate is the number of GA generations required to maintain a 95% success rate.

Figure 7 shows that the addition of advice yields an enormous performance advantage on this domain. Figure 7 also shows that given a moderate amount of time (10 generations), the GA provides a 10% increase in the success rate. Furthermore, the addition of advice produces an 18-fold improvement in the convergence rate over using GAs alone. Not only does advice improve the convergence *rate*, but it also improves the *level* of convergence: after 80 generations, the GA with advice holds a 99% or above success rate whereas after all 100 generations the GA without advice still cannot get above a 97% success rate. For this problem, the advice appears to be biasing the GA into a very favorable region of the search space.

To further test our compilation method, we have recompiled our Navigation advice into op rules for a Nomad 200 mobile robot that is equipped with very noisy sonar and infrared sensors and can adjust its turning rate and speed. The sensors are so noisy that the robot sometimes mistakes two boxes four feet apart for a wall. The op rules that result from compilation have not been refined by the GA to develop a tolerance to noise; therefore, this noise poses a severe challenge.

The op rules are linked to a vendor-provided interface that translates the language of the SAMUEL rules (e.g., "IF asonar4 [17..85] THEN SET turn -400") into joint velocity and servo motor commands. From high-level advice to avoid obstacles and rendezvous with a goal point, our method has compiled rules that enable the robot to succeed approximately a third of the time in avoiding three large boxes and reaching a goal point on the other side of a room. The same SKB rules are used for compilation. With the random rule alone, it is extremely unlikely to successfully complete this task. Our next step will be to refine these robot rules using GAs within a simulation.[4]

In conclusion, our multistrategy system offers two advantages. First, it provides an initial "boost" from seeding with initial high-level knowledge. Although this boost is insubstantial on Evasion, on Navigation we see an order of magnitude in improvement in the convergence rate. Second, the multistrategy system provides the robustness and improvement gained from GA refinement.

[4] We wish to use SAMUEL both to handle the noise and because we had to manually refine the qualitative to quantitative mappings somewhat - SAMUEL could automate this.

Refinement yields a 10% increase in success rate on Navigation and a 50% increase on Evasion.

5 Related Work

This work relates most strongly to the following topics in machine learning: advice taking, combining projective and reactive planning, methods for compiling high-level goals into reactive rules, learning in fuzzy controllers, and multistrategy learning. This work also relates to research in differential game theory. We discuss each in turn.

5.1 Machine learning

Advice taking has been considered as early as 1958 (McCarthy) and later by Mostow (1983) and others. To date, research on assimilating advice in embedded agents has been limited but encouraging. Previous research has focused mainly on providing low-level knowledge. For example, Laird *et al.* (1990) and Clouse and Utgoff (1992) have had good success providing agents with information about which action to take. Chapman (1990) gives his agent high-level advice. Our advice taker differs from Chapman's because it can operationalize advice long before the advice is applied and because it refines the advice with a GA. Most important of all, our advice taking method is unique because it involves a multistrategy approach that couples a knowledge-intensive *deductive* precompilation phase with an empirical *inductive* refinement phase.

We assume that high-level knowledge is operationalized but not applied immediately. Methods for operationalizing advice that will be applied immediately include STRIPS-like planners (Nilsson, 1980) and explanation-based learning (EBL) planners (e.g., Segre, 1988). A closely related system is Mitchell's (1990). This system combines EBL projective planning with reactive planning. Our method for compiling goals is similar to that of EBL because it uses the notion of operationality. It differs because we do not assume that the advice will be applied immediately, and therefore our compilation method has no current state on which to focus plan generation. All of the above-mentioned methods create a projective plan to achieve a goal from the current state. We precompile advice for multiple possible states.

Because our method precompiles plans from possible states rather than from a current state, it is very similar to the methods of Schoppers (1987) and Kaelbling (1988) for compiling high-level goals into low-level reactive rules. Our method differs from those of Schoppers and Kaelbling because it includes the EBL notion of operationality. Also unlike Schoppers and Kaelbling, we use a refinement method following compilation.

Considerable prior work has focused on knowledge refinement. Others have used GAs to refine qualitative to quantitative mappings. For example, Karr (1991) uses GAs to select fuzzy membership functions for a fuzzy controller. Lin (1991), Mahadevan and Connell (1991), and Singh (1991) initialize their systems with modular agent architectures then refine them with reinforcement learning. Lin trains a robot by giving it advice in the form of a sequence of desired actions. Mahadevan and Connell initialize their reinforcement learner with a prespecified subsumption architecture, and Singh guides his reinforcement learner by giving it abstract actions to decompose.

One of the most similar approaches to ours is that of Towell and Shavlik (1991). They also couple rule-based input with a refinement method; however, their refinement method is neural networks. This multistrategy system converts rules into a network topology. The content of each rule is preserved; therefore, the transformation is syntactic. Our multistrategy system, on the other hand, focuses primarily on semantic transformations that use qualitative knowledge about movements in space to convert abstract goals into concrete actions. The deductive compilation scheme (but not the refinement) is in common with Mitchell's (1987) derivation of a strategy for pushing objects in a tray using a qualitative theory of the process.

5.2 Differential game theory

Differential game theory is a branch of mathematical optimal control theory. It assumes that the behavior of the controlled system can be modeled as a system of ordinary differential equations (ODEs). The evasion problem considered in this paper is a typical example of a differential game. In particular, the problem is two-person zero-sum differential game with a *constant terminal time*. Both the pursuer and the evader move in a bounded rectangle in two dimensions. The evader has to avoid getting to within a certain distance of the pursuer for a certain length of time. In the minimax formulation of the problem, the optimal strategy of the evader is one that achieves its objective under the least favorable assumptions on the motion of the pursuer.

Differential games are formulated mathematically by specifying the motion equations of the pursuer and evader, the class of admissible controls for both systems (which identifies the way in which the pursuer and evader can change their motions), and the target or goal functional. A classic reference for this is (Basar and Olsder, 1982). The focus of work in differential game theory is to identify conditions under which optimal strategies for the evader can be derived. This assumes complete knowledge of the dynamics of the evader and pursuer, both of which are unavailable to us. The theory would be more useful to us if it had a qualitative counterpart which allowed us to determine the existence of solutions to the evader's problem from partial knowledge of the evader and pursuer's dynamics.

6 Discussion

We have presented a novel multistrategy learning method for operationalizing and refining high-level advice into low-level rules to be used by a reactive agent. Operationalization uses a portable SKB. An implementation of this method has been tested on two complex domains and a Nomad 200 robot.

We have learned the following lessons:
(1) Our advice compiler can be effective on complex domains, and it will be important to identify the regions of greatest effectiveness for advice, (2) A portable SKB appears feasible, and (3) Coordinating a deductive learning strategy (advice compilation) with an inductive learning strategy (GA refinement) can lead to a substantial performance improvement over either method alone. This success, however, depends on the how the advice biases the GA search. Future work will focus on identifying those characteristics of advice that bias this search favorably. We will also focus on further addressing our questions about performance using different advice and alternative domains (e.g., Subramanian and Hunter, 1993).

Many other interesting directions are suggested by our experimental results. At present we do not consider the cost of incorporating advice. For larger scale problems and situations where advice is provided more frequently, the agent has to reason about the costs and benefits of compiling advice at a given point in time. Classical issues in trading off deliberation time for action time are relevant here. We have chosen the GA method for refinement because it was readily available to us. A comparison of neural network refinement schemes and reinforcement learning schemes on the problems studied here will provide valuable insights into the tradeoffs between various refinement strategies. We believe that multistrategy learning systems of the future must have a bank of operationalization and refinement methods at their disposal and have fast methods for selecting them. We have chosen a specific breakdown of effort between the advice compilation and refinement phases. How this coordinates with our choice of problem domains and refinement schemes is another question for future study.

Acknowledgements

We appreciate the helpful comments and suggestions provided by Bill Spears, Alan Schultz, Connie Ramsey, John Grefenstette, Alan Meyrowitz, and the anonymous reviewers.

References

Basar, T. and G. Olsder, Dynamic Noncooperative Game Theory. New York: Academic Press, 1982.

Chapman, D., Vision, instruction and action. Ph.D. thesis. MIT, 1990.

Clouse, J. and P. Utgoff, A teaching method for reinforcement learning. In *Proc. of the Ninth International Workshop on Machine Learning*, 1992.

Erickson, M. and J. Zytkow, Utilizing experience for improving the tactical manager. In *Proc. of the Fifth International Workshop on Machine Learning*, 1988.

Gordon, D. and D. Subramanian, Assimilating advice in embedded agents. Unpublished manuscript, 1993.

Grefenstette, J., Ramsey, C., and A. Schultz, Learning sequential decision rules using simulation models and competition. *Machine Learning,* Volume 5, Number 4, 1990.

Holland, J. *Adaptation in Natural and Artificial Systems.* University of Michigan Press: Ann Arbor, 1975.

Kaelbling, L., Goals as parallel program specifications. In *Proc. of the Seventh National Conference on Artificial Intelligence*, 1988.

Karr, C., Design of an adaptive fuzzy logic controller using a genetic algorithm. In *Proc. of the Ninth International Conference on Genetic Algorithms*, 1991.

Laird, J., Hucka, M., Yager, E., and C. Tuck, Correcting and extending domain knowledge using outside guidance. In *Proc. of the Seventh International Conference on Machine Learning*, 1990.

Lin, L., Programming robots using reinforcement learning and teaching. In *Proc. of the Ninth National Conference on Artificial Intelligence*, 1991.

Maes, P. and R. Brooks, Learning to coordinate behaviors. In *Proc. of the Eighth*

National Conference on Artificial Intelligence, 1990.

Mahadevan, S. and J. Connell, Automatic programming of behavior-based robots using reinforcement learning. In *Proc. of the Ninth National Conference on Artificial Intelligence*, 1991.

McCarthy, J., Mechanisation of thought processes. In *Proc. Symposium*, Volume 1, 1958.

Michalski, R., Inferential learning theory as a basis for multistrategy task-adaptive learning. In *Proc. of the Eighth International Workshop on Multistrategy Learning*, 1991.

Mitchell, T., Becoming increasingly reactive. In *Proc. of the Eighth National Conference on Artificial Intelligence*, 1990.

Mitchell, T., Mason, M., and A. Christiansen, Toward a learning robot. CMU Technical Report, 1987.

Mostow, D. J., Machine transformation of advice into a heuristic search procedure. In R. Michalski, J. Carbonell, and T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Vol. 1). Tioga Publishing Co., Palo Alto, CA., 1983.

Nilsson, N., *Principles of Artificial Intelligence.* Tioga: Palo Alto, 1980.

Odetayo, M. and D. McGregor, Genetic algorithm for inducing control rules for a dynamic system. In *Proc. of the Third International Conference on Genetic Algorithms*, 1989.

Schoppers, M., Universal plans for reactive robots in unpredictable environments. In *Proc. of the Sixth National Conference on Artificial Intelligence*, 1987.

Schultz, A. and J. Grefenstette, Using a genetic algorithm to learn behaviors for autonomous vehicles. In *Proc. of the Navigation and Control Conference*, 1992.

Schultz, A. and J. Grefenstette, Improving tactical plans with genetic algorithms. In *Proc. of the IEEE Conference on Tools for AI*, 1990.

Segre, A., *Machine Learning of Robot Assembly Plans.* Kluwer: Boston, 1988.

Singh, S., Scaling reinforcement learning algorithms by learning variable temporal resolution models. In *Proc. of the Ninth International Workshop on Machine Learning*, 1992.

Subramanian, D. and S. Hunter, Some preliminary studies in agent design in simulated environments. Unpublished manuscript, 1993.

Tesauro, G., Temporal difference learning of backgammon strategy. In *Proc. of the Ninth International Workshop on Machine Learning*, 1992.

Towell, G. and J. Shavlik, Refining symbolic knowledge using neural networks. In *Proc. of the Eighth International Workshop on Multistrategy Learning*, 1991.

Whitehead, S. and D. Ballard, Learning to perceive and act by trial and error. *Machine Learning,* Volume 7, Number 1, 1991.