Eucalyptus:
An Integrated Spoken Language/Graphical Interface for Human-Computer Dialogue

Kenneth Wauchope

Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Code 5512
4555 Overlook Avenue, S.W.
Washington, DC 20375-5320
202/767-9004

Introduction

As more and more machine intelligence is built into the interactive software tools of the future, the more the human-computer "dialogue" may come to resemble a true human-human dialogue, each party anticipating information needed by the other and avoiding rigid, repetitive or overly detailed exchanges by assuming the existence of a body of shared contextual knowledge. Although to humans dialogue means primarily natural language (NL) communication, friendly and effective human-computer dialogue should be able to take full advantage of an integrated mix of several different interaction modes including keyboard, speech, graphics, and body gestures. Context tracking would allow each input or output transaction to be minimally specific, deriving its full interpretation from background information relevant to the current topic of the interchange.

The U.S. Navy has a particular interest in developing advanced user interfaces to such interactive knowledge-based tools as decision support systems, expert systems, and training aids. As a testbed for an initial investigation of integrated NL/graphical interfaces to such systems, we have been working for about a year now with a simulation-based test planning tool developed by Los Alamos National Laboratory for the Naval Air Systems Command. The tool simulates scenario-generated air engagements between friendly and hostile forces as viewed on an AWACS-style radar monitor, and was developed as a concept demonstration of the KOALAS (Knowledgeable, Observation Analysis-Linked Advisory System) process architecture for the design of intelligent control systems [1]. The tool's graphical display consists of a synthetic radar screen accompanied by a set of continuously updated text windows presenting current fighter and threat status, communications transactions, sensor readings, and the latest tactical orders being offered by the internal advice generator (Figure 1). Graphical input from the operator (for hypothesizing threats and ordering fighters to respond to them) is via pop-up menus and dialogue boxes, with data entry either by clicking on

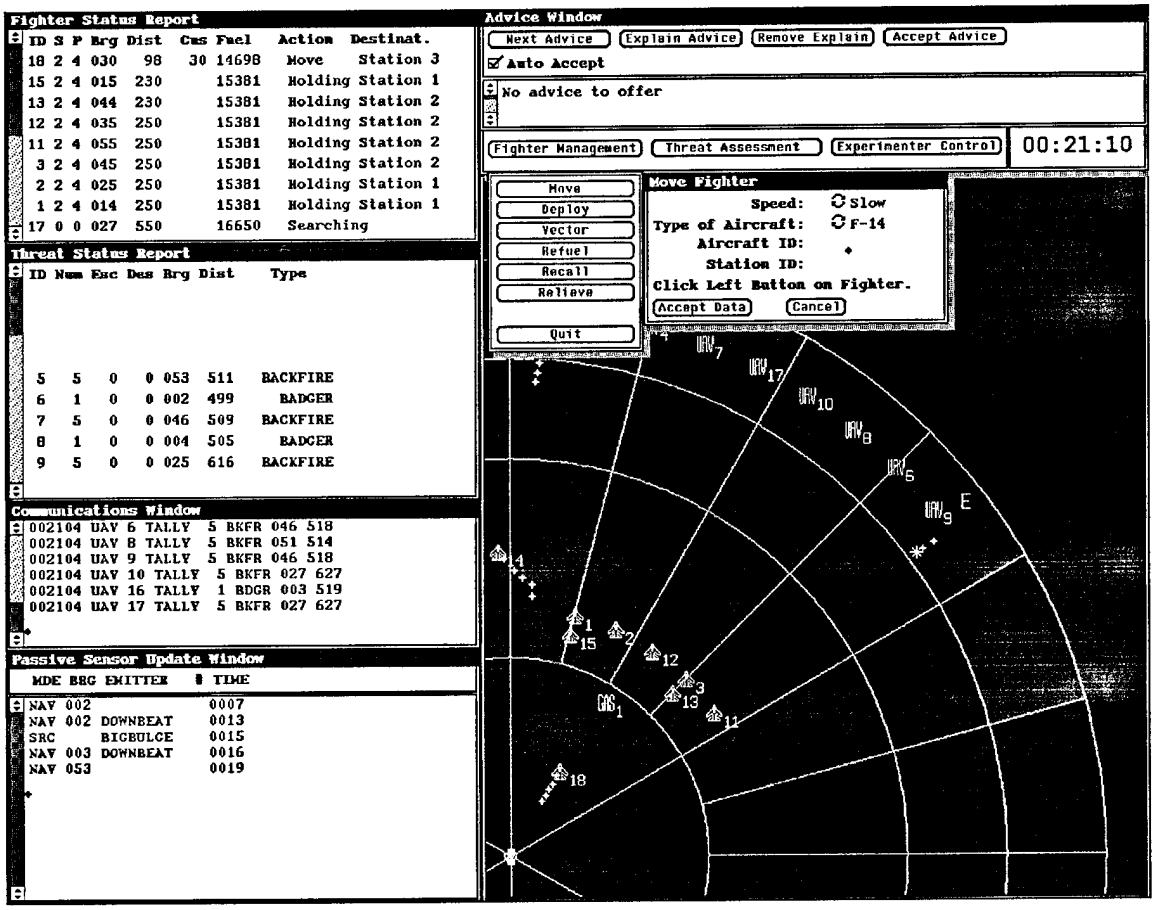


Figure 1. KOALAS interface after selecting Move option from Fighter Management menu

locations or objects on the radar screen, or typing at the keyboard.

Eucalyptus ("natural input to koalas") is the spoken NL interface we have developed and integrated into the graphical interface of the KOALAS test planning tool. As a first step toward our long-term goal of investigating the role of discourse tracking in advanced user interfaces for human-computer dialogue, Eucalyptus provides speech equivalents to each of the graphical interface constructs and maintains a local focus history of each domain object "mentioned" either verbally or graphically. It also provides an NL query facility that extends the data display capabilities of the system and provides an additional mode of dialogue interaction.

System Architecture

Speech input is processed by a Speech Systems Incorporated Phonetic Engine 200 continuous speech recognizer. The resulting utterance string is echoed in a text display window and then passed to a general-purpose NL processor for syntactic analysis, lexical/thematic semantic interpretation, reference resolution, and conversion to quantified predicate-logic expression. An interface module translates the result into a sequence of application-specific commands for execution by the target program.

The syntax for the speech recognizer consists of about 140 context-free productions and 18 word class definitions with a terminal vocabulary of just under 200 words. Whereas the speech recognition syntax is a non-iterative, non-recursive semantic grammar (to improve recognition accuracy as well as to keep the syntax down to a size that will successfully compile), the NL parser syntax (coincidentally also containing about 140 context-free rules) is a phrase structure grammar with a fully recursive treatment of embedded clauses and noun phrases (NPs) as well as generalized iteration of compounds, phrase modifiers, and verb complements. The syntactic ambiguities that inevitably arise during parsing are resolved by syntactic restriction rules augmenting the context-free grammar, by selectional restrictions enforced by the semantic interpreter, and by failures in the reference resolution module.

Referring Expressions

Assuming that the user of a tool simulating a military air control system will be comfortable adhering to a constrained vocabulary and regular syntax, we have limited the clausal paraphrase in Eucalyptus to slight syntactic variations and some synonymy of verbs and prepositions (*Send fighter 1 to threat 1, Have fighter 1 vector out to threat 1*). Where we do permit wide variety of usage, however, is in the referring expressions (pronouns and NPs), since that is where discourse coherence expresses itself so strongly. Hence the operator can use any of a variety of phrases (*fighter 1, the F14, the friendly moving to station 5, it*, etc.) to refer to the same object depending on the current discourse context. Similarly, a variety of indefinite NPs (*an F14, any fighter not holding station, one of them*) can be used when the operator does not care to identify a specific individual.

For the most part Eucalyptus has a closed-world semantics in which each definite or indefinite NP has as its referent a set of one or more objects already known to the system. Some of these (the friendly fighters and actual threat aircraft) are dynamically instantiated at runtime from the particular input scenario on which the simulation is being run; others (the aircraft carrier, the six aircraft assignment stations, the system itself, the operator, and the various on/off switches of the graphical interface) are scenario-independent and defined at compile time. The exceptions to the closed-world semantics are the hypothesized threats, whose number and identity can change from second to second and so are represented as intensional formulae evaluated on an as-referenced basis by consulting the internal run state of the simulation. The values of aircraft attributes like course, bearing and distance also change dynamically and so are represented intensionally as well.

Aircraft Control

Clicking the "Fighter Management" button in the graphical interface brings up a menu of order types with labels like "Move", "Vector" and "Recall". Selecting an order type brings up a dialogue box requesting additional information needed to carry out the order. The dialogue box's text input fields can be filled either by clicking on objects or locations in the radar screen, or by keying in their ID numbers. Once all fields have been filled, a final mouse click accepts

or rejects the data and dismisses the popup.

Users of the graphical interface have complained about the number of these low-level mechanical operations it takes to carry out an aircraft order. For example, just to have a fighter move at top speed to a new assignment requires six or seven key clicks taking 15-20 seconds. The popup menus also obscure one particular region of the radar display so much of the time that users playing adversarial scenarios against each other will sometimes have their enemy raids attack from that bearing so that they can penetrate their opponent's defenses unnoticed! With the addition of speech input, however, the user can simply say *Move fighter 1 to station 5 fast* and in a second or two Eucalyptus brings up a completely data-filled dialogue box ready for final confirmation with either a single mouse click or a verbal OK. The reduced response time, ease, naturalness, avoidance of repetitive mechanical operations, and reduced screen clutter makes spoken language an attractive alternative input modality.

An underspecific utterance like *Move fighter 1* brings up a partially filled dialogue box whose remaining empty fields respond to the usual graphical operations. For example, the "Station" field can then be filled by a quick mouse click on the desired area of the radar screen, which is more natural than having to know and remember the station ID of that location. Since we are taking an NL-based rather than token-based approach, we do not allow the user to bring up a completely empty dialogue box by simply saying "Move," since that is not a grammatically appropriate way to tell someone to move something (rather, it seems to be telling the system itself to move).

For the most part each fighter order corresponds to a verb and the fields of its dialogue box either to the verb's "inner cases" (thematic role arguments) or "outer cases" (verb phrase modifiers), making it particularly easy to integrate NL input with these graphical operators. For example, the Fighter and Station fields of the "Move" dialogue box correspond respectively to the "patient" (affected object) and "to-location" (destination) thematic roles of the verb *move*, while the Speed field corresponds to an "at-speed" outer case (*Move it at top speed*). The exception is the "Relieve" order, whose dialogue box has three fields labelled Forward Fighter, Backup Fighter and Tanker, the first two corresponding to thematic roles of the verb (*Relieve forward fighter 1 with a backup*) but not the last. In effect, the order compresses the two actions *relieve* and *refuel* into a single operation, making an embedded clause necessary for a fully specific natural language command (*Have a backup relieve fighter 1 to refuel at the tanker*).

Clicking the "Threat Assessment" button brings up a similar menu of operators for creating, altering and deleting hypothetical threat aircraft. Again, the "Create" and "Alter" dialogue boxes contain fields (Number, Bearing, Distance, Speed, and Course) that do not correspond to arguments of the verbs, so Eucalyptus only accepts inputs of the form *Create a hypothetical badger* or *Change that badger to a backfire* and requires that the remaining fields be filled using graphical operations. Since a single mouse click on the radar screen fills the Bearing, Distance and Course fields simultaneously and the remaining fields are choice items offering default values, this integration of speech and graphical input is by far the most natural to the user.

Interface Control

The Experimenter Control button brings up a menu of choices and check boxes (on/off switches) that serve primarily to control various aspects of the graphical display. Each check box can be turned on or off verbally using any of the verb pairs *turn on/off, show/don't show* and *start/stop showing (Turn on real threats, Don't show radar trails, Start showing aircraft trails*). Each item is explicitly modeled as a domain object, which allows a command like *Show all the trails* to turn on both the Aircraft Trails and Radar Trails switches simultaneously. A pragmatic constraint returns an error message if the user tries to verbally turn on or off a switch that is already in the desired state.

Advice Acceptance

The KOALAS tool contains a rule-based tactical advice generator whose recommendations the operator can either accept manually one at a time, or have automatically accepted by switching the system into "autoaccept" mode. Each piece of advice is displayed in an Advice Window as a simple imperative sentence ("Vector fighter 10 to track 1") accompanied by an English-like explanation of the antecedent conditions that caused the rule to fire. Since the explanation portion of the window takes up considerable screen space, two buttons are provided to expose and hide it. Only

one piece of advice is displayed (and can be manually accepted) at a time, so another button is provided to cycle through each piece of advice that is currently active.

The verbal commands added to Eucalyptus closely resemble the labels on the corresponding Advice Window buttons: *Show the (next) advice, Explain the (current) advice, Remove the (advice) explanation, Accept the (current) advice,* and *Autoaccept advice* (also *Turn on autoaccept, Start auto acceptance,* etc.). Since "next advice" and "current advice" are modeled as domain objects, they can be referenced anaphorically (*Explain it*) or underspecifically (*Explain the advice*). Constraints in the semantics knowledge base ensure that NL references conform to the semantics of the graphical interface, rejecting inputs like *#Show the current advice* and *#Accept the next advice.*

Query Facility

The graphical interface displays the current state of each friendly and threat aircraft in two continuously updated text display windows, the Fighter and Threat Status Reports. A line of information for each aircraft gives its ID number, bearing, distance from the carrier, and other pertinent data, and the operator must direct his/her attention to the appropriate window and scroll around in it in search of desired information. A natural language interface provides an attractive alternative in the form of a query facility, allowing the operator to ask for the particular information of interest and see it displayed separately. Eucalyptus answers queries in six different ways depending on the type and scoping of determiners, for example

Is fighter 1 moving to station 5?
Yes.
Which station is fighter 1 moving to?
Station 5.
Is fighter 1 moving to a station?
Yes: station 5.
Are all the fighters holding station?
No: not fighter 1.
How many fighters are moving to station 5?
One.
Which fighters are moving where?
Fighter 1/station 5.

Discourse Tracking

Discourse relations are linguistic co-occurrence phenomena found in multiple-utterance NL transactions. One of the simplest discourse relations is that of antecedent/anaphor: once an entity has been mentioned, references to the same object can be abbreviated in subsequent utterances (using underspecific NPs or pronouns) until the topic of the discourse changes. Anaphoric reference in Eucalyptus can considerably improve the naturalness of the discourse of operator inputs to the system:

Create a threat and lock ***it*** *on track. Now vector an F14 holding station 2* ***there.***
How much fuel does ***that fighter*** *have?*

and, in conjunction with the query facility, the discourse of the human-computer dialogue as well:

Are there any F14s vectoring to threat 2?
Yes: fighter 1.
Have ***it*** *return to the carrier.*

The reference resolution module (developed for us by visiting MIT student Gina-Anne Levow) employs a focusing algorithm based on syntax, semantics and recency to select the most likely antecedents for anaphoric references. Each referring expression in the natural language input generates a data structure called a discourse entity containing infor-

mation about the phrase's referent set, semantic class, number, and the thematic role it played in the utterance. The discourse entities are stored in data structures (focus stack, alternate focus list, current focus, and actor focus) searched by the focusing routine when trying to resolve possible anaphora.

Eucalyptus also generates a discourse entity for each referent set produced in response to a user query, thus providing the antecedent for *it* in the sample dialogue just illustrated. Finally, to fully integrate NL input with the tool's graphical interface, Eucalyptus has each graphical "reference" to an object or location (either by mouse click or typed input to a dialogue box) generate a discourse entity as well, allowing subsequent NL utterances to refer anaphorically to the graphically denoted entities.

Discussion

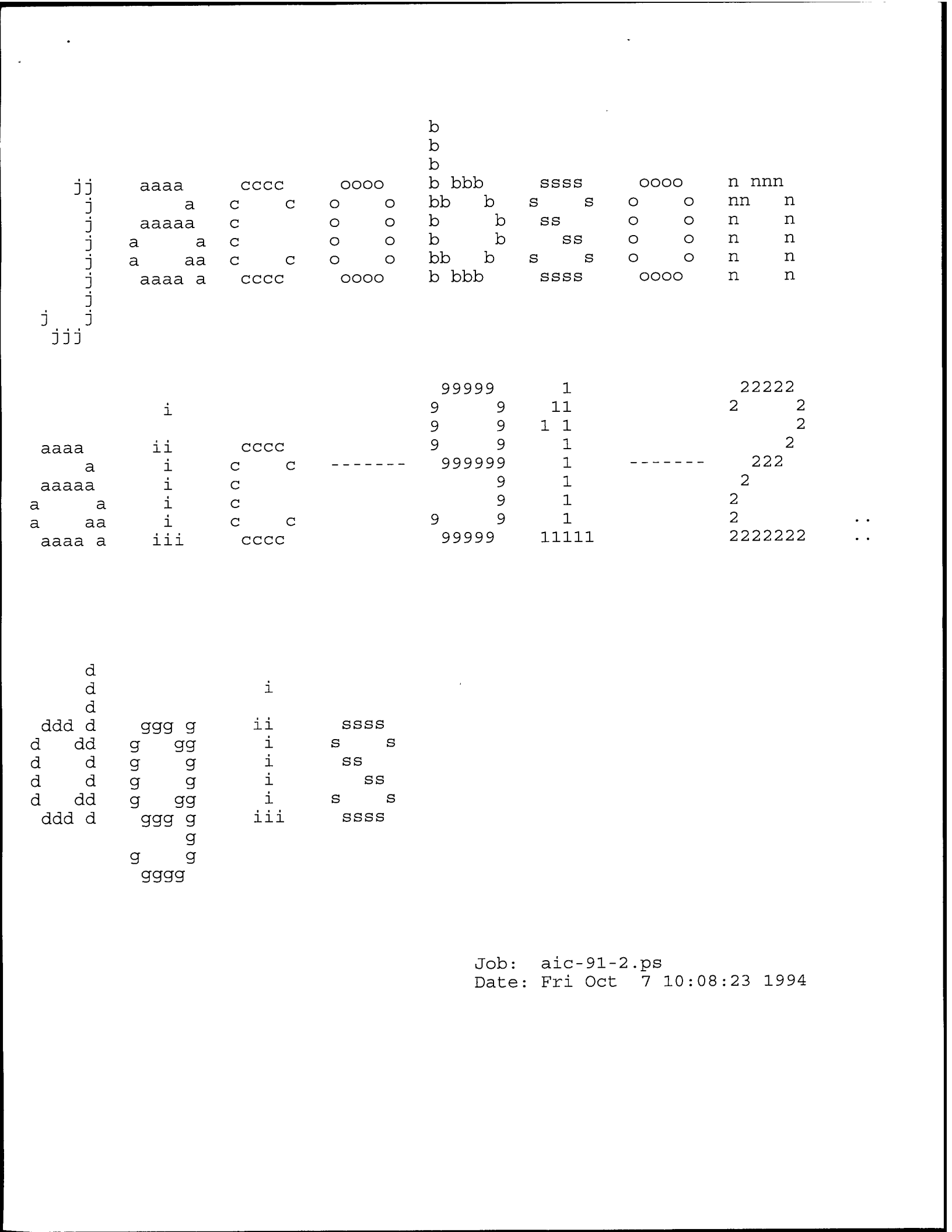
Even with the restricted syntax and vocabulary of this application domain, Eucalyptus is already starting to push against the bounds of habitable continuous speech recognition. For example, we have not yet been able to incorporate relative clauses into the speech-recognition grammar because they render it too large to compile. There is evidence that dialogue partners often split such constructions into sentence pairs, for example decomposing a sentence like *Vector the fighter holding station 4 to threat 1* into the two utterances *That fighter holding station 4? Vector it to threat 1*, but we would still like to accommodate the first construction. (Alternatively, the grammar might successfully compile if we limit it to only one relative clause per utterance, since inputs as complex as *Vector the fighter holding station 4 to the threat that fighter 1 is vectoring to* are perhaps unlikely.) Similarly, we can add iterative compounding of imperatives to the speech grammar (although doing so considerably decreases the recognition accuracy), but if we attempt to add compounding of questions as well, the grammar again blows up. Requiring users to use simple sentences is a reasonable demand, however, and it is always possible to explicitly encode as compound sentences only those pairs of operators that a user might commonly issue in sequence.

Two classes of vocabulary items are responsible for most of the speech recognition errors in this application: numbers (e.g. *two, twelve,* and *twenty* are easily confused) and determiners (*a, the, that*). The former problem can be ameliorated by allowing the user to spell out individual digits (*one niner*), but the latter is a serious problem in a system that pays such careful attention to tracking reference in discourse, where the difference between *a fighter, the fighter* and *that fighter* can be crucial. We have added a "Confirm Input" switch to the interface that allows the user to check the echoed speech input and edit it before sending it off for execution, which for minor edits like changing *a* to *the* might prove acceptable. We have also investigated providing the user with the first *n*-best transcriptions of the speech input, but have found that if the correct transcription is not the first one, more often than not it is not among the alternative transcriptions either, having fallen below threshold.

Future plans for Eucalyptus include the experimental addition of a discourse-based ellipsis handling capability to deal with fragmentary input, although that may again push the speech grammar size out of bounds or reduce its accuracy to unacceptable levels. Deictic reference (integrating speech input with graphical pointing and/or selection operations) is an attractive feature of integrated interfaces because it allows precise denotations through the fusion of imprecise inputs, but correctly time-stamping and correlating the spoken utterance and graphical inputs may prove difficult in Eucalyptus since the phonetic decoding of the speech signal is done offline from the host computer. An example where the time correlation is crucial is the sentence *Have that fighter relieve that fighter*, where either the first, second, or both NPs are to be interpreted as deictic if accompanied by pointing operations and as anaphoric otherwise.

References

1. Barrett, C. and Donnell, M. Real-time expert systems: Considerations and imperatives. *Information and Decision Technologies* 16. North-Holland, 1990.


Job: aic-91-2.ps
Date: Fri Oct 7 10:08:23 1994

Learning the Persistence of Actions in Reactive Control Rules

Helen G. Cobb and **John J. Grefenstette**
Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Code 5514
Washington, DC 20375-5000

Abstract

This paper explores the effect of explicitly searching for the persistence of each decision in a time-dependent sequential decision task. In prior studies, Grefenstette, *et al*, show the effectiveness of SAMUEL, a genetic algorithm-based system, in solving a simulation problem where an agent learns how to evade a predator that is in pursuit. In their work, an agent applies a control action at each time step. This paper examines a reformulation of the problem: the agent learns not only the level of response of a control action, but also how long to apply that control action. By examining this problem, the work shows that it is appropriate to choose a representation of the state space that compresses time information when solving a time-dependent sequential decision problem. By compressing time information, critical events in the decision sequence become apparent.

1 INTRODUCTION

The problem of learning to perform a decision task presents us with several learning objectives. One objective is to find an efficient representation of the search space. The representation of the system's states (i.e., state descriptors of inputs and outputs) should reduce the number of states to those that are essential for performing the task. Another learning objective is to discover the best input-output mapping (i.e., decision policy) given the state representation. Through experiences within a problem domain, a learning system develops a decision policy that maximizes reward or minimizes some penalty (Sutton, 1990).

When learning time-dependent sequential decision tasks, such as the control of dynamical systems, another important objective may be discovering the times when it is critical, or necessary, to make a decision. In some time-dependent tasks, decisions are not made on a continual basis. Instead, the control decisions persist until another decision is required. Most learning systems, however, divide a control task *a priori* into a sequence of steps having equal duration. At the end of each small time interval, the learner examines the current input state and then performs some control action which in turn generates a new input state.

In some problems, such as a maze running, it is apparent that some control actions (e.g., maintaining a direction) should persist over several steps. In other problems, the benefit of persistent actions may not be obvious. In this latter case, having the learning system explicitly search for the duration of decisions can potentially provide a better understanding of the problem domain. Discovering whether or not a particular process can be described in terms of a few persistent steps is itself interesting. Such a solution, if it is a good one, may be easier to understand than one that makes decisions at every time step. If the learning system discovers that decisions must be made at frequent, regular intervals, then the result suggests that either each step of the process is critical or the time steps are too large (i.e., the time resolution is too coarse).

Independently inspired, Booker has also considered the persistence of actions in classifier systems (Booker, 1990). Booker points out a biological motivation for the persistence of actions. Learning at every time step is complex and costly; thus, the tendency for persistence is instinctual. At the lowest instinctual level, pre-programmed behavior patterns can be seen in motor movements. Ethologists also view other more general behaviors as essentially instinctual by calling them "programmed learning" (Gould, 1982). From this perspective, learning is a context dependent way of enhancing and filling-in complex details of goal-oriented instinctual behaviors.

This paper considers the problem of an agent being pursued by a predator. An *episode* of the problem begins when the agent detects the presence of the predator; the episode ends when either the predator captures the agent or the predator becomes exhausted. During an episode, the predator continually loses speed as the predator turns

to pursue the agent. An episode may last from one to twenty seconds. If the agent successfully evades the predator, the reactive strategy receives a large summary payoff; otherwise, the strategy receives a smaller payoff that is proportional to the amount of time that the agent evades the predator.

The problem of an agent evading a predator is a rich example of a decision task where the time steps cannot be partitioned *a priori* into a number of essential decision points. If the same action-values can be applied over several time steps, i.e., if the bias of persistent action is correct, then including duration information in the rules should permit a solution requiring a fewer number of rules. With fewer rules to evaluate, the learner should develop a decision policy faster. The resulting rules should compress time information so that the critical states become clearer. However, if a bias of persistent action is inappropriate, then including duration information in the rules could increase the difficulty of the problem since encoding duration in the rules increases the size of the search space. Nevertheless, the resulting rules would be the same as those found when making a decision at each time step.

We use the SAMUEL system (Grefenstette, 1989) in this study of representational bias. In prior studies (Grefenstette, 1989, 1990, 1991; Schultz, 1990), SAMUEL learned a reactive control *strategy* for the agent, expressed as a small set of production rules, that permitted the agent to successfully evade its pursuer. At each time step, a rule in the strategy fired, thus specifying the control decision of the agent. In this paper, we modify the problem by asking SAMUEL to learn a control decision that includes both the action-value to be applied in a given circumstance and the time duration of that control action.

2 THE SAMUEL SYSTEM

SAMUEL is a genetic algorithm-based system that learns to perform sequential decision tasks based on experience.[1] Learning in SAMUEL occurs at two levels: at the *reactive strategy level* and at the *production rule level*. Learning occurs at the reactive strategy level by employing an enhanced genetic algorithm (GA). A population member of the GA of SAMUEL is a set of production rules that describes a reactive strategy. Each production rule consists of an IF precondition, which specifies the range of sensor values that activate the rule, and a THEN consequence, which specifies the resulting actions. Each generation, SAMUEL selectively breeds plausible new strategies from the current strategies based on the performance of the current generation's strategies for a set of episodes of the task. These new strategies replace the current population. Since the number of strategies per generation remains fixed, the selection pressure prevents offspring of relatively poor performing strategies from entering the next generation's population. New strategies are formed by applying genetic operators[2] to copies of successful strategies. After several generations, SAMUEL discovers highly successful strategies for the task.

In the experiments reported here, SAMUEL initially selects random action values regardless of the sensor inputs. If an action sequence results in a successful episode, then SAMUEL generates rules using the *specialize* operator, based on the sensor-action values in that sequence (for details, see (Schultz, 1990)). Later during a run, any modifications to the sensor and action conditions within the rules occur due to the occasional application of the mutate and creep operators. The creep operator makes small directed changes in the conditions, increasing or decreasing an end point by an amount that depends on the granularity of the sensor or action. For example, in Rule 89 (see Section 3 below), the creep operator could change the speed sensor from (speed is [350..950]) to (speed is [350..900]) since the speed sensor has a granularity of 50.

Learning also occurs at the production rule level in SAMUEL during the evaluation phase of the GA. During this phase, the GA calls the Competitive Production System (CPS) to evaluate a strategy for several episodes in order to obtain a statistical estimate of the strategy's average performance. The CPS performs partial matching of sensor inputs to the conditions of the rules in the strategy, conflict resolution in case more than one rule matches sensor inputs, and credit assignment, which apportions a summary reward received at the end of an episode to the active rules in the decision sequence. The CPS adjusts the strengths of the rules' actions to reflect the success of the episode. Over time, a rule's strength reflects its expected payoff (Grefenstette, 1988).

3 THE PROBLEM DOMAIN

As explained in the introduction, these preliminary experiments consider the problem of an agent being pursued by a predator that loses speed as it turns. The agent detects the state of the predator at each simulation time step through the following six sensors: (1) *last-turn*, the current turning rate of the agent; (2) *time*, the clock time since the detection of the predator; (3) *range*, the predator's current distance from the agent; (4) *bearing*, the location of the predator relative to the agent; (5) *heading*, the direction that the predator points relative to the agent, and (6) *speed*, the predator's current speed measured relative to the ground. Given these six sensors, the agent has only one control action, *turn*, that sets the

[1] This research uses Version 1.4 of SAMUEL.

[2] SAMUEL uses operators such as *select*, *cluster*, *crossover*, *mutate* and *creep*. These operators and others are described in (Grefenstette *et al.*, 1990).

agent's turning rate. Rule 89 below illustrates a simplified version of the rule format used in each strategy:

```
Rule 89:
IF (and (time is [0..10]) (bearing is [6..9])
   (last-turn is [-180..90]) (last-turn-duration is [1..5])
   (range is [100..1200]) (heading is [270..250])
   (speed is [350..950]))
THEN (and (turn is [-90..-90])(turn-duration is [1..1]))
```

Figure 1: Example of a Rule Format

Associated with the last-turn condition and the turn action are a range of duration values. Even though last-turn and last-turn-duration appear to be separate conditions within Rule89, last-turn-duration is semantically linked to the last-turn sensor in the way it is interpreted, and similarly, turn-duration is semantically linked to the turn action.

During the matching cycle in the CPS, the last-turn sensor reads the last turn rate and its associated duration. The CPS attempts to match both of these quantities with the rule conditions in addition to matching other sensor readings with their respective rule conditions. When a rule fires, the turning rate and the turn's duration are read from the rule's action.

Initially, the CPS uses a rule that randomly selects the duration of a turn to be one to ten simulation steps. Later, when a rule created by the specialize operator fires, the CPS uses the duration value for turning that was once associated with a successful episode. The mutation and creep operators operate on both the conditions in the rules and any associated duration values.

4 PRELIMINARY EXPERIMENTS

We report here on two experiments: one is a *baseline* case (denoted **B**) where the turn duration information in the rules is explicitly set to one time step; the second is a *duration* case (denoted **D**) where the duration values can range from one to ten steps. For both experiments, the population consists of 100 strategies. We vary the *strategy size*, i.e., the maximum number of rules permitted within a strategy, as an experimental parameter for each experiment. In particular, we report here a total of seven *cases*: experiment **B** with strategy sizes 4, 6, and 15, and experiment **D** with strategy sizes 2, 4, 6, and 15. (We omitted the strategy size of 2 for experiment **B** due to the relatively poor results we obtained when using a larger strategy size of 4 rules.)

To evaluate each strategy in the GA, the CPS computes the average performance of the strategy over ten episodes. All episodes use the following initial conditions for the sensors: last-turn = 0; last-turn-duration = 1; time = 0; range = 1000; heading = 0, and speed = 700. The initial condition for bearing is a randomly selected value between 0 and 12, inclusive.[3] In running the experiments, the same initial pseudo-random number generator seeds were used for all of the cases. A separate generator was used to introduce duration values into the simulation in order to minimize the effect of this stream on the random stream of other state variables.

For each of the seven cases, we run SAMUEL through both a *training* and a *testing phase* (for details, see Grefenstette, 1991). During training, SAMUEL runs ten repetitions which are identical except for changes in the seeds of the pseudo-random generators. In these experiments, SAMUEL runs each repetition for 400 generations. (The learning curves actually converge by the 240th generation.) At the end of each repetition, we extracts the *best strategy* over the generations.

The testing phase consists of running each best strategy for 1000 episodes to determine the percentage of the time that the agent succeeds in evading the predator. Based on the sample of the ten strategies for each case, we compute the average and the standard deviation of the percentage of wins. Table 1 (on the next page) summarizes the results for experiments **B** and **D**. For each strategy size, the *%Wins* column indicates the average percentage of wins for the ten best strategies and the associated standard deviation. The *Maximum* column indicates the largest value for the percentage of wins, and the number of strategies achieving that level of success. For example, the test results for experiment **B** with a strategy size of fifteen rules indicate that three of the strategies permit the agent to win 100% of the time.

4.1 ADDING DURATION TO ACTIONS

In reviewing the results, first let us compare the two experiments **B** and **D** for the same strategy sizes.

In examining a strategy size of 15, a two-tailed $F_{9,9,0.05}$ test leads us to reject the null hypothesis that the variances are equal. There is more variance associated with **D** in this case. As a result, we apply the Behrens-Fisher two-tailed t-test for unknown, and perhaps unequal variances to test the differences in the means. A $T_{5,0.05}$ test indicates that we cannot reject the hypothesis that the means are equal.

There appears to be no significant difference in variances or the means for the strategy size of 6 (using two-tail tests at a 0.05 level of confidence). The difference in the performance between the experiments for the strategy of size 4 is obvious (i.e., 76.1 ± 5.7 for **B** versus 96.2 ± 5.7 for **D**). It is clear that when using rules that express actions for only one time step, a strategy size of 4 is too small to permit satisfactory performance.

[3] These are the same as the "fixed" initial conditions used in (Grefenstette, 1990); however, the experiments discussed in this previous paper do not use the creep operator.

Table 1: Strategy Performances for **B** and **D**

"Best" Strategy Performances for B and D Based on Tests of 1000 Episodes					
Case	Strategy Size	%Wins Avg	%Wins SD	Maximum %Wins	Maximum No.
B	2	-	-	-	-
	4	76.1	5.7	83.2	1
	6	94.3	6.3	100.0	1
	15	98.4	1.9	100.0	3
D	2	93.3	5.6	95.3	5
	4	96.2	1.5	100.0	1
	6	95.1	6.7	100.0	4
	15	95.9	3.8	100.0	1

The strategy size in Table 1 indicates the number of rules in a strategy. Some of these rules may be redundant in that they may virtually cover the same conditions as other rules; thus, only a few of a strategy's rules may actually fire. Table 2 below summarizes for each case, the average number of rules that the ten strategies actually use. These results indicate the necessary space requirements for each case.

Table 2: Rules Used in Experiments **B** and **D**

Rules Used for Experiments B and D Based on a Test of 1000 Episodes				
Strategy Size	Rules **B** Avg	Rules **B** SD	Rules **D** Avg	Rules **D** SD
2	-	-	2	0
4	3.2	0.8	3.4	0.7
6	5.5	0.5	3.9	0.7
15	11.5	2.3	6.9	2.1

Table 2 suggests that experiment **B** requires more space than **D** for strategy sizes 6 and 15; this is as we might expect. The overall performance of **D** is comparable to that of **B** for these two cases. From the results in Tables 1 and 2, it appears that compressing time information into the rules does not present a difficult problem for SAMUEL.

4.2 EXAMINING A SIMPLE STRATEGY

The interpretation of a strategy tends not to be immediately obvious by simply examining the set of rules making up the strategy. (This observation is true, regardless whether or not the rules contain duration information.) For example, Figure 2 shows one of the strategies from experiment **D** that uses only three of its six rules to win 100% of the 1000 episodes.

```
IF (and (time is [0..8]) (bearing is [6..9])
    (last-turn is [-90..135])(last-turn-duration is [1..1])
    (range is [100..1200]) (heading is [270..250])
    (speed is [350..950]))
THEN (and (turn is [-90..-90]) (turn-duration is [8..8]))

IF (and (time is [0..11]) (bearing is [10..2])
    (last-turn is [-180..45])(last-turn-duration is [1..8])
    (range is [600..1200]) (heading is [260..10])
    (speed is [100..850]))
THEN (and (turn is [135..135]) (turn-duration is [1..1]))

IF (and (time is [0..4]) (bearing is [3..6])
    (last-turn is [0..0]) (last-turn-duration is [1..3])
    (range is [600..1500]) (heading is [280..250])
    (speed is [450..950]))
THEN (and (turn is [90..90]) (turn-duration is [9..9]))
```

Figure 2: A Successful Strategy From Experiment **D**

By examining how these rules are used during an episode, we gain more insight. We observe during the strategy's execution that only some of the sensors contain useful information. In particular, for this example we can eliminate the range, heading, and speed sensors. Furthermore, by examining the sensor values as each rule fires, we notice that some of the conditions cover more of a range of values than necessary. After choosing the critical actions at times zero and one, it is unimportant what the agent does, since the predator has been exhausted after the agent turns for eight or nine seconds. No other rules fire since the episode comes to an end. Figure 3 shows a handcrafted strategy that encorporates these simplifications.

```
IF (and (time is [0..1]) (bearing is [6..9])
    (last-turn is [-90..135]) (last-turn-duration is [1..1]))
THEN (and (turn is [-90..-90]) (turn-duration is [8..8]))

IF (and (time is [0..0]) (bearing is [10..2])
    (last-turn is [0..0]) (last-turn-duration is [1..1]))
THEN (and (turn is [135..135]) (turn-duration is [1..1]))

IF (and (time is [0..0]) (bearing is [3..6])
    (last-turn is [0..0]) (last-turn-duration is [1..1]))
THEN (and (turn is [90..90]) (turn-duration is [9..9]))
```

Figure 3: A Simplification of the Strategy in Figure 2

This strategy outlines three courses of action that depend upon the initial reading of the bearing sensor: (1) if the bearing sensor reads between 6 o'clock and 9 o'clock at time zero, then the agent should turn a sharp right (-90) for eight seconds; (2) if the bearing sensor reads between 10 o'clock and 2 o'clock at time zero, then the agent

should turn left 135 degrees for one second, and then make a sharp right at subsequent time steps, and (3) if the bearing sensor reads between 3 o'clock and 6 o'clock at time zero, the agent should make a sharp left (90) for nine seconds. The first rule in the strategy above handles two events: the right turn at time zero in (1), and the sharp right turn during subsequent time steps in (2). Figure 4 shows a six rule strategy using single time steps that is comparable to the strategy in Figure 3.

```
IF (and (time is [0..0])(bearing [6..9])
  (last-turn is [0..0]))
        THEN (and (turn is [-90..-90]))
IF (and (time is [1..8]) (last-turn is [-90..-90]))
        THEN (and (turn is [-90..-90]))
IF (and (time is [0..0])(bearing [10..2])
  (last-turn is [0..0])
        THEN (and (turn is [135..135]))
IF (and (time is [1..15]) (last-turn is [135..135]))
        THEN (and (turn is [-90..-90]))
IF (and (time is [0..0])(bearing is [3..6])
  (last-turn is [0..0]))
        THEN (and (turn is [90..90]))
IF (and (time is [1..9]) (last-turn [90..90]))
        THEN (and (turn is [90..90]))
```

Figure 4: A Simplified Strategy Using Single Time Steps

5 CONCLUSION

The preliminary results in Section 4 suggest some avenues for future research. First, when we compare the strategy given in Figure 2 with the strategy that it is based upon Figure 1, it is apparent that many of the sensors are irrelevant for this simple instance of the problem. It may be desirable to employ an operator that permits the elimination, or masking out, of irrelevant sensor information. We might expect SAMUEL to eliminate certain sensors from consideration as it discovers that the condition of a sensor encompass the entire range for that sensor (e.g., (speed is [0..1000]). However, since SAMUEL reduces the size of the search space that it examines as it learns (i.e., more exploitation than exploration), the boundaries of a rule's conditions may not be tested sufficiently so that SAMUEL discovers the irrelevancy of a sensor. Second, it may be desirable to develop operators that explicitly chain together rules that always execute in succession; this way, sequential patterns could be observed in the rules of the strategies themselves.

Overall, despite the significantly larger search space, a small strategy size performs well in the case **D** runs. SAMUEL finds completely successful strategies requiring only a few rules by including duration of actions in those rules. The example in Section 4.2 illustrates how compressing time in the rules can potentially offer a simpler solution strategy. Simple solutions are generally not apparent when examining execution traces that require agent to make a decision at each time step. By including duration information in the rules, SAMUEL can discover *efficient* solutions that requires fewer decisions or fewer changes in the actions of the agent. Notice that when using this approach, SAMUEL can *still discover* a solution that permits the agent to make a decision at each time step. Including duration in the search space at the very least tells us something about the difficulty of the problem. For these reasons, we believe that this approach is worthy of further consideration.

Acknowledgements

We would like to thank Diana Gordon for her useful comments.

References

Lashon B. Booker. (1990) Instinct as an Inductive Bias for Learning Behavior Sequences. In *Proceedings of the 1990 Conference Simulation of Adaptive Behavior*, MIT Press.

James J. Gould. (1982) *Ethology: The mechanisms and evolution of behavior*, New York: W. W. Norton.

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

John J. Grefenstette. (1989) A system for learning control strategies with genetic algorithms. In J. David Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*. San Mateo, CA: Morgan Kaufmann.

John J. Grefenstette, Connie Loggia Ramsey, and Alan C. Schultz. (1990) Learning Sequential Decision Rules Using Simulation Models and Competition. *Machine Learning Journal.*

John J. Grefenstette, and Helen G. Cobb. (1991) User's Guide for SAMUEL, Version 1.3. (to be published as a) *NRL Report.*

Alan C. Schultz and John J. Grefenstette. (1990) Improving Tactical Plans with Genetic Algorithms. In *Second International Conference on Tools for Artificial Intelligence*, IEEE.

Richard S. Sutton. (1990) Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*: 216-224.