Adaptive Agent Tracking in Real-world Multi-Agent Domains: A Preliminary Report

Milind Tambe, Lewis Johnson and Wei-Min Shen
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
{tambe,johnson,shen}@isi.edu

August 26, 1996

Abstract

Intelligent interaction in multi-agent domains frequently requires an agent to track other agents' mental states: their current goals, beliefs, and intentions. Accuracy in this *agent tracking* task is critically dependent on the accuracy of the tracker's (tracking agent's) model of the trackee (tracked agent). Unfortunately, in real-world situations, model imperfections arise due to the tracker's resource and information constraints, as well as due to trackees' dynamic behavior modification. While such model imperfections are unavoidable, a tracker must nonetheless attempt to be adaptive in its agent tracking. This article identifies key issues in adaptive agent tracking and presents an approach called DEFT. At its core, DEFT is based on discrimination-based learning. The main idea is to identify the deficiency of a model based on tracking failures, and revise the model by using features that are critical in discriminating successful and failed tracking episodes. Because in real-world situations the set of candidate discriminating features is very large, DEFT relies on knowledge-based focusing to limit the discrimination to those features that it determines were relevant in successful tracking episodes — with an autonomous explanation capability as a major source of this knowledge. This article reports on experiments with an implementation of key aspects of DEFT in a complex synthetic air-to-air combat domain.

1 Introduction

In multi-agent environments, intelligent agents must interact — collaborate or compete — to achieve their goals. Many of these multi-agent domains are dynamic and real-time, requiring the interaction to be flexible and reactive. For instance, in the arena of training, there is a recent thrust on dynamic, real-time interactive simulations — e.g., realistic traffic environments [5], or realistic combat environments[37] — where intelligent agents may interact with tens or hundreds of collaborative and non-collaborative participants (agents and humans). In the arena of education, intelligent tutors, whether in the form of "standard" intelligent tutoring systems[1] or as participants in virtual environments (e.g., a virtual guide in a virtual historical setting[18] or virtual instructor in a training environment [8, 22]), must interact with students in real-time. Similarly, in the arena of entertainment, recent work has focused on real-time, dynamic interactivity among multiple agents within virtual reality environments [3, 6]. Such real-time interaction is also seen in robotic environments [10].

In all these environments, *agent tracking* is a key capability required for intelligent interaction [38, 36, 40, 21, 1]. It involves monitoring other agents' observable actions and inferring their mental state — their goals, beliefs, intentions — and tracking this over time. This capability is closely related to plan recognition [9, 31], which involves recognizing agents' plans based on observations of their actions. One key difference is that plan-recognition efforts generally assume that agents are executing plans that rigidly prescribe the actions to be performed. Agent tracking, in contrast, involves recognizing a broader mix of goal-driven and reactive behaviors. It is appropriate in domains where agents exhibit dynamic behaviors in response to the changing environment and the actions of other agents.

This paper focuses on *adaptive agent tracking*, an important requirement to scale up agent tracking (and plan recognition) systems to real-world domains. In particular, successful agent tracking requires that a tracker (tracking agent) use a specified model of the trackee (tracked agent), and the trackee's currently observed actions, to accurately infer and track its mental state. For instance, one typical agent tracking approach is *model tracing*[1], where the tracker relies on an executable specification of the trackee's model. By executing this model, and matching the model's predictions with actual observations, the trackee's current goals, beliefs and intentions are tracked. However, in real-world domains, the tracker's model of the trackee is often imperfect, i.e., incomplete or incorrect. This is a natural consequence of the tracker's limited resources in a real-world domain — acquiring and processing all of the required knowledge is highly problematic. Furthermore, the trackee itself contributes to model imperfections, since the trackee's behaviors are not static, but often alter over time.

The end result of such model imperfections is inaccuracy in agent tracking. Adaptive agent tracking attempts to alleviate this inaccuracy by adapting the trackee's model. Of course, in complex, real-world domains, it is difficult (if not impossible) to aim to perfect the trackee model, i.e., the tracker must in the end cope with many of the imperfections. Nonetheless, this article focuses on enabling the tracker to remedy at least some of the more critical imperfections. To this end, the article introduces an approach called *DEFT*[1], that is based on discrimination-based learning. The main idea is to identify the deficiency of a model based on tracking failures, and revise the model by using features that discriminate successful and failed tracking episodes.

[1]Discrimination-based Explanation-Focused adaptive Tracking.

Real-world domains, however, present some key challenges for a pure discrimination-based approach. First, these domains are complex, with thousands of features available as candidates for discrimination, whose values may change over time. Second, in these domains, identifying successful and failed tracking episodes for discrimination — which requires observations of or interactions with other agents — can be costly, risky, or otherwise difficult; and hence unavailable in abundant measure. To address these difficulties, DEFT adopts a knowledge-based approach to focus discrimination on features that it determines were critical in the successful tracking episodes — while relying on an autonomous explanation capability[7] to play a major role in this focusing. In short, DEFT brings together three separate threads of research: agent tracking[38], autonomous explanation[7] and discrimination-based learning[29].

DEFT is applied in a real-world synthetic air-combat environment, to enable pilot agents to adapt models of their adversaries. While our analysis focuses on this one environment, given its real-world character, we expect that its lessons will generalize to some of the other multi-agent environments mentioned earlier. Indeed, one key contribution of this article is the identification of some important issues — for instance, the types of knowledge required for focusing discrimination — as the multi-agent community scales up from simpler testbeds to complex real-world domains.

All of the agents discussed in this article are based on the Soar integrated architecture[16]. We will assume some familiarity with Soar's problem-solving, which involves applying an operator hierarchy to a state to reach a desired state. For learning, Soar relies on chunking[11], a form of explanation-based learning (EBL)[15], and its operation will be explained in a following section. The initial implementation of key elements of DEFT in Soar has shown some promising results; although many issues remain open for future research.

The rest of this article is organized as follows: Section 2 concretely motivates the need for adaptive agent tracking via a real-world example from the synthetic air-to-air combat domain. Sections 3 and 4 present the necessary background on agent tracking, discrimination-based learning and autonomous explanation, and motivate the DEFT approach. Section 5 builds on this background material to provide a detailed description of DEFT. Section 6 provides experimental results, followed by a discussion of related work in Section 7. Finally, Section 8 concludes.

2 Adaptive Agent Tracking in Real-world Domains

The domain of our work on adaptive agent tracking is one of virtual battlefields based on Distributed Interactive Simulation environments (DIS)[4, 32]. These are synthetic, yet real-world environments, and they have already been used in large-scale synthetic military exercises. These environments promise to provide cost-effective and realistic environments for training and rehearsal, as well as for testing new doctrine and tactics.[2] The realization of this promise is critically dependent on intelligent automated agents that can act as effective human surrogates — interacting intelligently with humans as well as other agents. Agent tracking is of course one key aspect of such an intelligent interaction [38]. Certainly, an adversary will not communicate information regarding its goals and plans to an agent voluntarily — such information must be inferred via tracking. Furthermore, even in collaborative situations, tracking often assumes importance due to

[2]This basic simulation technology, once proven promising for military applications, is leading to other possible applications ranging from training for disaster relief to interactive entertainment.

communication difficulties.

As mentioned earlier, for agent tracking, the tracker executes a runnable model of the trackee, matching the model's predictions with actual observations. One key reason that tracking in this fashion remains a challenging problem is ambiguity. For instance, in air-combat simulations, a tracker cannot directly observe a missile fired by its opponent (the trackee). It needs to infer a missile firing from the trackee's observable maneuvers, even though those are often ambiguous. Nonetheless, given a reasonably accurate model of the trackee, the tracker agent can hope to address such ambiguity in real-time [38].

Given adaptiveness on part of the trackee, however, the problem becomes much more difficult. The tracker cannot necessarily assume its model of the trackee is accurate, or that it will stay accurate over time. Such a situation does arise in the synthetic battlefield environment, given the adaptive nature of intelligent adversaries. Human adversaries will very likely adapt and evolve their tactics to exploit weaknesses in an intelligent agent's behaviors. For instance, in the simulated theater of war exercise (STOW-E) held in November of 1994, human pilots deliberately changed their missile firing tactic — instead of pointing their aircraft nose straight at the target before firing a missile (0 - 5 degrees "nose-off" as shown in Figure 1), they began firing missiles while maintaining a 25-degree nose-off from the target (as shown in Figure 2). This was intended to confuse the participating intelligent pilot agents, and indeed it did [37]. Unable to track this changed missile firing tactic, intelligent pilot agents got shot down. Of course, human pilots are bound to come up with novel variations on known maneuvers, and intelligent agents cannot be expected to anticipate them. Yet, at the same time, intelligent agents cannot remain in a state of permanent vulnerability — for instance, getting shot down each time the 25-degree nose-off variation gets used — otherwise they would be unable to continue to provide challenging and appropriate training for human pilots.

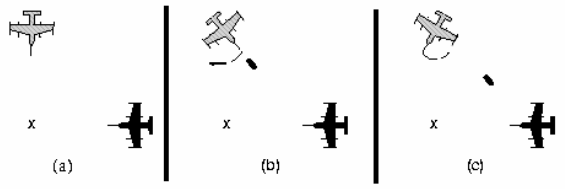


Figure 1: A simulated air-combat scenario illustrating the "normal" missile firing maneuvers: (a) aircraft approach each other; (b) the trackee (in light-shaded aircraft) points straight at the tracker to fire a missile; (c) the trackee executes an Fpole turn to continue supporting the missile without flying right behind the missile. An arc on an aircraft's nose shows its turn direction.

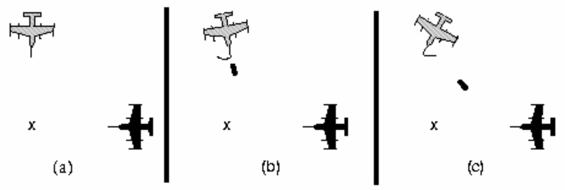


Figure 2: A simulated air-combat scenario illustrating a change in the missile firing maneuver.

To deal with such imperfections in the trackee's model, the tracker must:

- Recognize the deficiency of its model, and
- Adapt the model by either revising its assumptions regarding known agent actions or postulating the existence of heretofore unknown actions.

Unfortunately, characteristics of the combat simulation environment conspire to make this difficult. In particular, given the domain complexity, it is difficult for the tracker to pinpoint the deficiencies in its model of the trackee. For instance, in the above STOW-E example, the tracker failed to recognize that 25-degree "nose off" is also a legitimate condition for missile firing (for its model is that missile firing is possible only if the nose-off angle is within 0-5 degrees). Yet, since it cannot observe the missile, and thus cannot know when it was fired, the tracker cannot easily pinpoint this precise deficiency. In particular, there are at least three problems it faces here:

1. There are thousands of features on the state, many with continuously varying values.
2. Encounters with the trackee are costly.
3. The trackee, being non-collaborative, does not allow for any controlled experimentation.

Thus, the tracker is faced with the difficult task of determining why it was unexpectedly shot down. (One simplifying assumption here is that the tracker assumes that it is the trackee's missile that shot it down).

While addressing the above model deficiency is critical, as mentioned earlier, it is indeed difficult for the tracker to learn a perfectly accurate model of the trackee, and access all of the relevant information for accurate tracking. For instance, in some situations, to predict whether an opponent pilot (trackee) will engage in an offensive or defensive maneuver, it is useful to know whether or not the trackee is willing to enter into risky situations in pursuit of its goals. However, it is difficult to obtain such information in advance or during a real-time air-combat simulation — in fact, the tracker may end up jeopardizing its own survival if it indeed attempts to acquire this "risk" information.

Therefore, we believe that attempting to learn an exact model of the trackee will not be a very fruitful enterprise for the tracker. Instead, it should focus its learning effort on situations involving catastrophic failures, such as the unexpected missile firing in the STOW-E example. With respect to other less harmful imperfections, a flexible tracking strategy — one that can work with an imperfect model of the trackee — would appear to be a more fruitful approach. Such a strategy would need the capability to switch inferences dynamically in real-time [38, 36]. For instance, if the tracker is not sure if the trackee is performing an offensive maneuver or a defensive maneuver, it may first assume that the maneuver is offensive (worst-case scenario), and then flexibly modify this assumption as soon as warranted by further observations. Thus, the tracker need not depend on acquiring information regarding risk.

3 Motivating DEFT

The characteristics of adaptive agent tracking pose some obvious constraints or demands on the learning methods that can be applied here. On the one hand, since the tracker may not have access to a large number of training instances when a model adaptation is needed, a learning method must rely on deep analysis of the instances available instead of statistical analyses of large numbers of instances. This requirement is very likely to undermine most of the "standard" learning methods, for example, decision tree induction [20] or relational concept learning [19]. On the other hand, unlike most knowledge-rich speed-up learning, such as the traditional explanation-based learning [15], model adaptation in agent tracking must be at the knowledge level — the tracker agent must take some inductive leap to mend the model which may not be entirely explanable by the knowledge that is available.

Given these constraints, a discrimination-based method offers itself as one viable choice. The advantage of using discrimination-based learning is that it can inductively generate plausible hypotheses based on a small number of training instances, yet readily enable deep analysis when domain knowledge is available (by applying knowledge to identify a small number of critical differences in the discrimination process). Indeed, as we will see in this paper, domain knowledge and other theory-driven techniques can be used to focus the search space for critical discriminating features.

To ground the following discussion, and to motivate the need for focused discrimination-based learning, this section describes our RESC approach to agent tracking in more detail. This is followed by a discussion of the straightforward application of discrimination-based learning. The limitations of the straightforward approach are identified, followed by the augmentations in DEFT that address these limitations.

3.1 RESC: An Approach to Real-time Dynamic Agent Tracking

The RESC (REal-time Situated Commitments) approach to agent tracking[38] builds on *model tracing*[1, 40]. Here, a tracker executes a model of the trackee (the agent being tracked), matching the model's predictions with observations of the trackee's actions. One key innovation in RESC is the use of commitments. In particular, due to ambiguity in trackee's actions, there are often multiple matching execution paths through the model. Given real-time constraints and resource-bounds, it is difficult to execute all paths, or wait so the trackee may disambiguate its actions. Therefore, RESC commits to one, heuristically selected, execution path through the model, which provides a constraining context for its continued interpretations. Should this commitment lead to a tracking error, a real-time repair mechanism is invoked. RESC is thus a repair-based approach to tracking (like repair-based approaches to constraint satisfaction[14] and natural language understanding[12]).

A second key technique in RESC leads to its situatedness, i.e., responsiveness to the present. To track the trackee's dynamic behaviors, it is necessary to execute the trackee's model so it is responsive to the changing world situation. A key assumption here is that the tracker is itself capable of the flexible and reactive behaviors required in this environment. That is, the tracker's architecture can execute such behaviors. Therefore, this architecture is reused to execute the trackee's model to allow dynamic model execution. There is thus uniformity in the tracker's generation of its own behaviors, and its tracking of the trackee's behaviors.

To present a concrete example of RESC, consider its application in the situation in Figure 1-b. To illustrate the uniformity in acting and tracking, we first describe tracker's generation of its own behaviors. Figure 3-a illustrates tracker's operator hierarchy. The top operator, *execute-mission* indicates that tracker is executing its mission (e.g., defend against intruders). Since the mission is not complete, a subgoal is generated. Different operators are available in this subgoal, such as *fly-flight-plan* and *intercept*. The tracker selects the *intercept* operator to combat its opponents. In service of *intercept*, the tracker applies *employ-missile* in the next subgoal. However, since this pilot agent has not reached its missile firing range and position, it selects *get-firing-position* in the next subgoal. Skipping to the final subgoal, *maintain-heading* enables the tracker to maintain its heading, as seen in Figure 1-b.

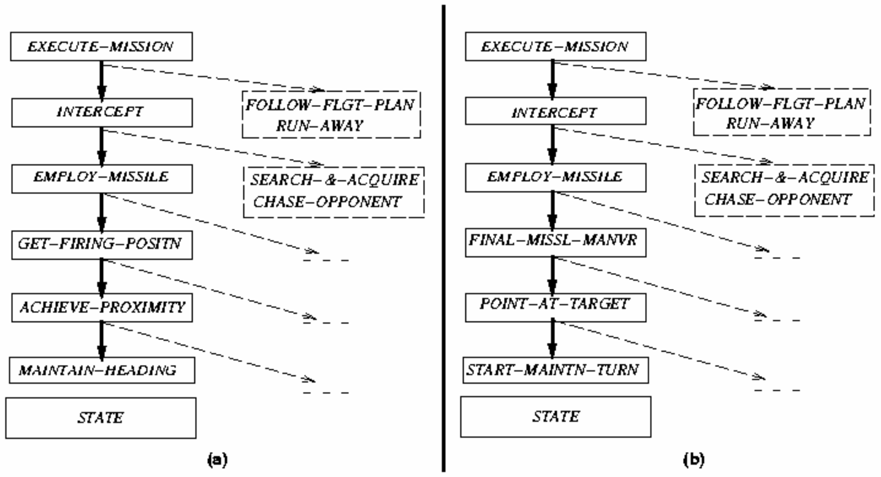


Figure 3: (a) Tracker's own operator hierarchy; (b) Operator hierarchy the tracker believes the trackee to be currently executing. Dashed lines are unselected alternative operators.

The tracker's dynamic behavior in this environment is supported by Soar's architectural mechanisms for flexible operator selection and reactive termination[24]. The tracker reuses this architecture in tracking. Thus, the tracker uses a hierarchy such as the one in Figure 3-b in tracking. Here, the operator hierarchy, the state, plus the unselected alternatives (shown by dashed lines in Figure 3-b), together constitute the tracker's model of the trackee. The selected operator hierarchy and the state constitute the tracker's current hypothesis of the trackee's mental state. By executing this operator hierarchy, and matching predictions with observation, this mental state is tracked. For instance, the tracker believes that the trackee is currently engaged in *execute-mission* — the top-most operator in Figure 3-b. Furthermore, the tracker believes that in service of *execute-mission* the trackee is executing the *intercept* operator. Skipping down to the final subgoal, the *start-&-maintain-turn* operator predicts the trackee's action. If this prediction matches the trackee's actual action, i.e., the trackee starts turning to point at tracker, then the tracker believes that the trackee is indeed attempting to point at its target in preparation for a missile launch, as indicated by higher-level operators in the hierarchy.

Such architectural reuse provides situatedness in RESC, enabling the tracking of dynamic behaviors. As for RESC's commitments, notice that from the tracker's perspective, there is some ambiguity in the trackee's actions Figure 1-b, e.g., it could possibly be part of a 150^o turn to run away. Yet, tracker commits to just one operator hierarchy in Figure 3-b. This commitment may be inaccurate, resulting in a *match failure*, i.e., a difference between prediction and the actual observed action. For example, if the trackee were to actually turn 150^o — rather than pointing straight at the target (i.e., the tracker) — there would be a match failure. RESC's primary repair mechanism to recover from such failures is "current-state backtracking", a repair-based approach that attempts the generation a new matching operator hierarchy without re-examining past states (see [38] for more details).

3.2 Applying Discrimination-based Learning to RESC

To apply discrimination-based learning in service of adaptiveness in agent tracking, we have chosen the approach of autonomous learning from the environment [27, 29]. At the center of this approach is the LIVE framework of predict-surprise-revise. In simpler domains, the approach works as follows. The prediction made by a model — about some other agents, or the environment — is compared and is verified or falsified by the actual observations. If a prediction fails in the current situation, then the current state is compared with an earlier state where the prediction was successful. Comparing the features of the two states yields differences, which are incorporated into the model, so that its prediction capability is improved. In the event that no known features are effective in discriminating the states, this approach will expand the comparison to two episodes that involve the relevant states and postulate new features that must be hidden in the environment.

The LIVE framework has been applied successfully to learning many different types of concepts and action models, including Boolean concepts [25], decision lists [26], first-order action and prediction rules [27], recursive theoretical terms (hidden variables) [30], and finite-state machines [28]. In addition, the LIVE style of learning is incremental, and the training examples can be either perfect or noisy. Compared to other incremental learning algorithms, LIVE assumes less biases in the learning process. Although it may prefer some hypotheses over others when the same amount of training is given, it can efficiently learn any concept in the hypothesis space regardless of whether the concept is conjunctive, disjunctive, or mixed. Furthermore, since LIVE uses feedback from the environment (or other agents), it can be easily integrated with problem solving systems and learning from experimentation.

At a high level, this approach matches well the needs of adaptive agent tracking. However, in the initial applications of this approach to our real-world situation, we encounter some practical difficulties. Since real-time agent tracking involves episodes that have many states and each state typically has a large number of features, directly discriminating two episodes is impractical. For example, in the synthetic air-combat domain, one-on-one synthetic air-combat typically has an average of 50 tracking decisions, each of which occurs in a state with an average of about 3000 attributes. Figure 4 presents a portion of this state. In the figure, S1 is the root, arcs are labeled with attributes and nodes are values, e.g., **BOGEY**(S1, B1) and **ID**(B1, BANDIT). Thus, in principle, there are over 150,000 feature-values per episode that need to be discriminated. Even if we can isolate the comparison to individual snapshots of the state, comparing two entire states will typically involve 3000 comparisons, which are unlikely to generate a small number of critical differences.

4 Adding Knowledge via Explanation in DEFT: Debrief

Because of the large number of features needed to describe each episode of agent tracking, discrimination-based learning would be impractical without some way of focusing on features that are most likely to be relevant. The following focusing heuristic is applied in our work:

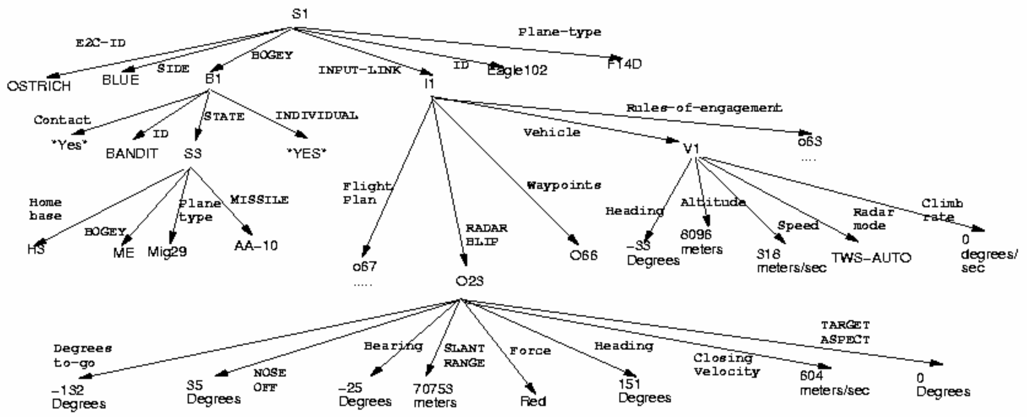


Figure 4: A small portion of a pilot agent's state during a synthetic combat episode.

> If tracking fails in a given situation, find other similar situations in which tracking succeeded, and analyze those situations to determine what features were relevant to the tracking decisions. Then focus discrimination-based learning on the features that were found relevant.

Thus, in order to understand why tracking failed in the case where nose-off angle was 25^o, the tracker should examine other cases where tracking succeeded (e.g., where the nose-off angle was 0^o). In reflecting upon the reasoning that takes place in the successful tracking cases, it is evident that the nose-off angle is one of the factors that the tracker takes into account when concluding that a missile was fired. Therefore, nose-off angle should be considered as one of the relevant features for discrimination-based learning.

Debrief [7] provides a means for analyzing successful tracking decisions in order to determine relevant features. Debrief is a package implemented in Soar which, when incorporated with Soar-based agents, enables them to explain their reasoning to people. Debrief has been used in synthetic forces and in the pedagogical agent called STEVE being developed by the Virtual Environments for Training Project [8, 22].

Debrief comprises three main capabilities.

- An episodic memory enables the agent to recall previous decisions and the circumstances in which they were made. It executes continually while the agent is performing the task, recording which operators are applied by the Soar agent, and noting when attributes in the agent's working memory change values.

- A decision analysis capability determines which features are relevant to the agent's decisions. It recalls the working memory state in which decision was made, using the episodic memory capability, and then establishes which attributes in the working memory state played a role in the decision.

- The user interface accepts questions from users, and generates answers using a combination of natural language and graphical presentations.

The episodic memory and decision analysis capability are employed by DEFT to recall agent tracking decisions and analyze them. The question input part of the user interface is employed to initiate the analysis process; the presentation capability is not required for DEFT, and typically is not used.

Debrief's episodic memory has two parts: a memory of decisions (i.e., operator applications), and a memory of working memory states. For a given type of problem solving agent, Debrief is provided with a specification of the agent that enumerates the set of Soar operators that represent interesting decisions, and the set of working memory attributes that one may wish to recall at a later date. Whenever one of the interesting operators is selected, a token identifying the name of the operator and its parameters is created and added to a log of decisions. If in the mean time the values of some of the interesting attributes have changed, the changes are recorded as well. The choice of what constitutes an "interesting" decision is up to the agent developer to decide. In the DEFT case, all agent tracking decisions are considered to be interesting.

Debrief's episodic memory relies on Soar's learning mechanism called *chunking*. Chunks in Soar are learned rules that are created as a side effect of problem solving. As described above, Soar agents maintain one or more hierarchies of operators during problem solving. If an operator that is lower in the hierarchy makes a change to global problem solving state, Soar can automatically generate a learned rule whose right hand side performs the state change, and whose left hand side summarizes the features of the state that held prior to the state change and which led to the state change. If the left hand side matches at some point in the future, the rule fires immediately, doing the work that otherwise would have been applied by the subgoal operators and making reapplication of the operators unnecessary.

In the case of episodic memory, the problem solving being chunked over is the activity of recalling problem solving states. Chunking enables Debrief to recall immediately the state associated with a decision. If Debrief did not make use of chunking, then whenever the agent attempted to recall the state associated with a decision it would have to scan through the scenario from the beginning up to the decision point, applying the incremental stage changes. Instead, processing proceeds as follows. Debrief has an operator called *recall-state*, which is applied whenever the agent wishes to recall a state. *Recall-state* takes one parameter, the token identifying a past decision. When applied for the first time to an operator token, Soar builds an operator hierarchy, in which the lower operators recall the values of individual state attributes based upon the recorded state information. To recall an attribute, Debrief scans backward in time, attempting to recall the attribute's value in previous states, until it reaches a state that it has previously recalled; at this point the chunk that was built during the previous recall will fire, recalling the attribute value. Debrief then updates the attribute value if necessary taking into account any state changes that occurred after the previously recalled state. When all of the attributes are recalled, a description of the recalled state is added to the global state, causing more chunks to be built. In the future whenever *recall-state* is applied to the same event token, the learned chunk applies immediately, making construction of the operator hierarchy unnecessary.

One should note that the RESC and Debrief both make use of Soar's operator hierarchies, but in very different ways. In RESC, operator hierarchies are a continually updated model of the goals and subgoals ascribed to the tracked agent. The same operator hierarchy is created each time a similar opponent action is modeled. In Debrief, each operator represents a problem to be solved, and operator hierarchies represent the problem decomposition. Once problems are solved, chunks are built that enable Debrief to avoid computing the same problem solutions in the future. Soar supports both types of operator hierarchies, and the choice of type of operator hierarchy determines whether or not automatic chunking takes place.[3]

Decision analysis identifies the features in the state that were critical a given decision (i.e., operator application); a feature is considered critical if the absence of that feature value would prevent the decision from being made. Decision analysis is performed by an operator *evaluate-decision* that takes two inputs: a token describing a decision, and the state in which the decision was made. It generates a list of features that are relevant to decision.

Decision analysis is accomplished by repeatedly deleting attribute values from the state description, installing the modified state description as Soar's problem solving state, and replaying the decision, by reinitiating problem solving starting from the modified problem solving state. If after problem solving is initiated the same operator is selected as before, then the deleted attribute value did not have an effect on the outcome of the decision, and is therefore considered to be irrelevant to the decision. If on the other hand a different operator is selected, or no operator is found to be applicable, then the attribute value is relevant to decision outcome. In most cases the attribute value is itself a complex object having attributes and values, in which case each of those component attribute values is analyzed in a similar fashion.

Throughout this process Debrief is building chunks; these chunks fire when analyzing similar decisions in the future, greatly reducing the work required to analyze other decisions and other states. They also fire during the analysis of a given state, whenever Debrief tries to delete an attribute has has been previously determined to be irrelevant. In essence, the analysis process is simply elaborating on the analysis that chunking performs automatically when explaining problem solving. Decision analysis adds to the analysis performed by chunking by determining *why* an attribute is relevant: it allows Debrief to determine what alternative decisions would have been made had the attribute not been present.

The use of chunking when analyzing decisions is particularly relevant to DEFT. Once Debrief has been used to analyze an agent tracking decision within one episode, the chunks that are created can be applied to a similar agent tracking decision in other episode, making it easier to compare agent tracking decisions across episodes.

5 DEFT: A Detailed Description

Given the relevant background provided in the previous sections, we can now describe DEFT in detail. Figure 5 describes DEFT as a 10 step approach. Step 1 illustrates that DEFT is a failure-driven approach, with similarities to other failure-driven learning approaches. What constitutes a critical failure worth analyzing would appear to be domain dependent. In Step 2, the blame for the failure is assigned to agent tracking — without that, there is no sense in adapting a model of the other agent. This transition from step 1 to step 2 requires generalized methods for blame assignment.

In standard discrimination-based learning, the next step would be essentially step 8, that is to identify a previously successful agent tracking episode and compare with the current failure episode.

[3]For interested readers, [39] provides details; specifically, RESC is based on a modified version of the Soar architecture, where these architectural modifications are included in the form of Soar rules.

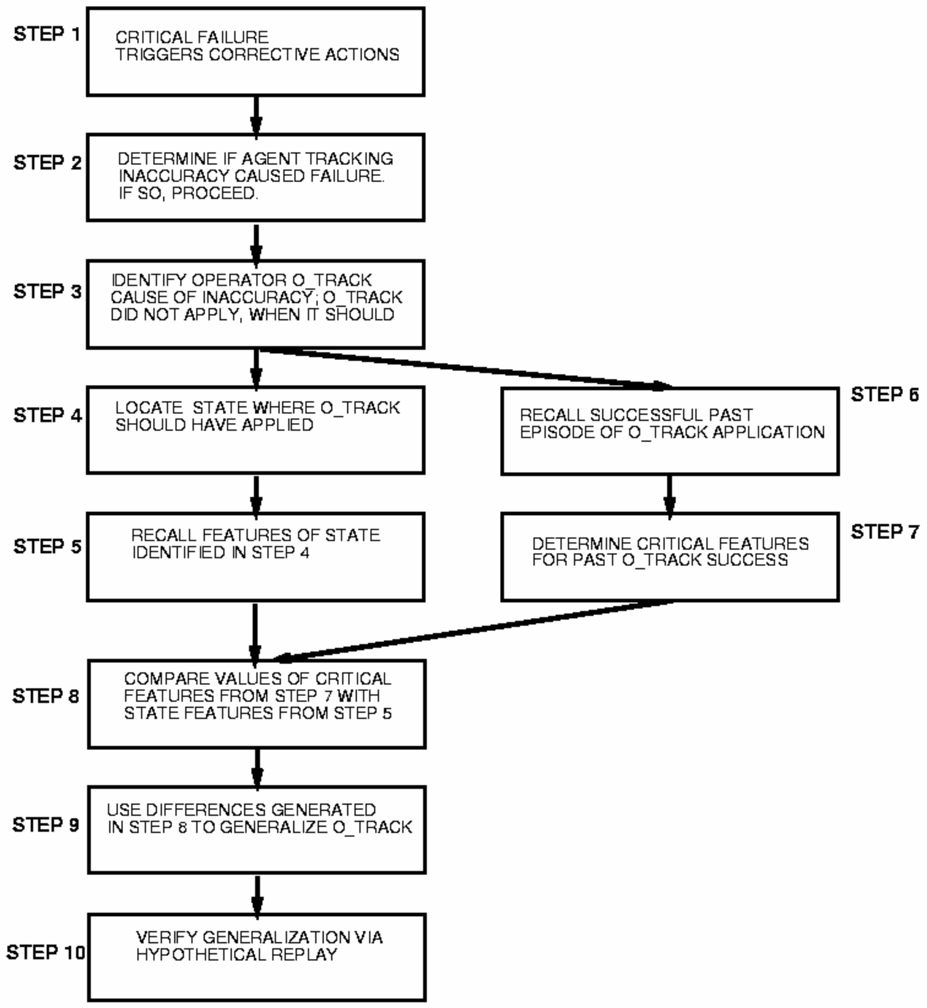

STEP 1
CRITICAL FAILURE TRIGGERS CORRECTIVE ACTIONS
STEP 2
DETERMINE IF AGENT TRACKING INACCURACY CAUSED FAILURE. IF SO, PROCEED.
STEP 3
IDENTIFY OPERATOR O_TRACK CAUSE OF INACCURACY; O_TRACK DID NOT APPLY, WHEN IT SHOULD
STEP 4
LOCATE STATE WHERE O_TRACK SHOULD HAVE APPLIED
STEP 6
RECALL SUCCESSFUL PAST EPISODE OF O_TRACK APPLICATION
STEP 5
RECALL FEATURES OF STATE IDENTIFIED IN STEP 4
STEP 7
DETERMINE CRITICAL FEATURES FOR PAST O_TRACK SUCCESS
STEP 8
COMPARE VALUES OF CRITICAL FEATURES FROM STEP 7 WITH STATE FEATURES FROM STEP 5
STEP 9
USE DIFFERENCES GENERATED IN STEP 8 TO GENERALIZE O_TRACK
STEP 10
VERIFY GENERALIZATION VIA HYPOTHETICAL REPLAY

Figure 5: Outline of the DEFT Approach.

DEFT's key additions are steps 3 through 7, which help focus the comparison. In particular, Step 3 identifies a specific operator (O_TRACK) responsible for the agent tracking failure. This operator should have applied, because it could have correctly predicted the other agent's actions, but here it failed to do so. Pinning the agent tracking problem down on a single operator is a simplification; in general, DEFT could and should extend to multiple points of failure. Step 4 involves identifying the situation in which O_TRACK failed to apply. Step 5 involves recalling the details of that situation, e.g., the agents' positions, the world situation, the tracker's current assessment of the opponent's goals, etc.

Steps 6 and 7 — essentially independent of steps 4 and 5 — involve recalling a past successful application of O_TRACK, and identifying critical features responsible for the success. These steps play to the strengths of the Debrief system mentioned in the previous section. In the best case, steps 6 and 7 will identify only a handful of features relevant in successful application of O_TRACK. Step 8 then exploits these features by first extracting values of those features from the situation identified in step 5 and comparing them. Once again, in the best case, this comparison may yield just one or two differences, which are used in Step 9 to generalize O_TRACK. O_TRACK did not apply when it should have (as identified in step 3); the generalization is intended to correct this error. This correction is verified via a hypothetical replay of the failure episode in Step 10. An accurate interpretation of the tracking failures, adds to validity of the hypothesized correction.

We will now illustrate DEFT's implementation in the context of the example under discussion (Figure 2). There is clearly a catastrophic failure in this situation — the pilot agent was unexpectedly shot down (DEFT relies on domain dependent heuristics in this case for implementing Step 1). For step 2, agent tracking failure is likely here since (i) the trackee's missile unexpectedly shot down the tracker; and (ii) the tracker had serious difficulties tracking the opponent's actions at the end of the engagement. This suggests that the opponent may have performed some unexpected action which enabled it to win the fight. If the tracker is able to improve its tracking ability, it might be able to anticipate such actions in the future and plan against them.

With respect to step 3, multiple heuristics appear applicable to identify O_TRACK. One is to apply domain knowledge to reason backwards from the undesired event (being hit by a missile) to its causal antecedent (the opponent firing a missile). The other is to examine the event trace, scan backwards from the point of catastrophe to the first point where a prediction failed and hypothesize operators that follow sequentially. Here, the tracker last predicts the opponent to move to a nose-off of 0-5 degrees, to fire a missile. This never occurs, and remains the first point of significant prediction/tracking failure. The operator that typically follows is the opponent's firing a missile. Thus, in this case, either heuristic yields *push-fire-button*, to push the button actually fire the missile, as the culprit operator that was not tracked. Similar heuristics may be used in steps 4 to pinpoint the situation where O_TRACK could have been applied. Step 5 then recalls the detailed world state in the situation identified in step 4.

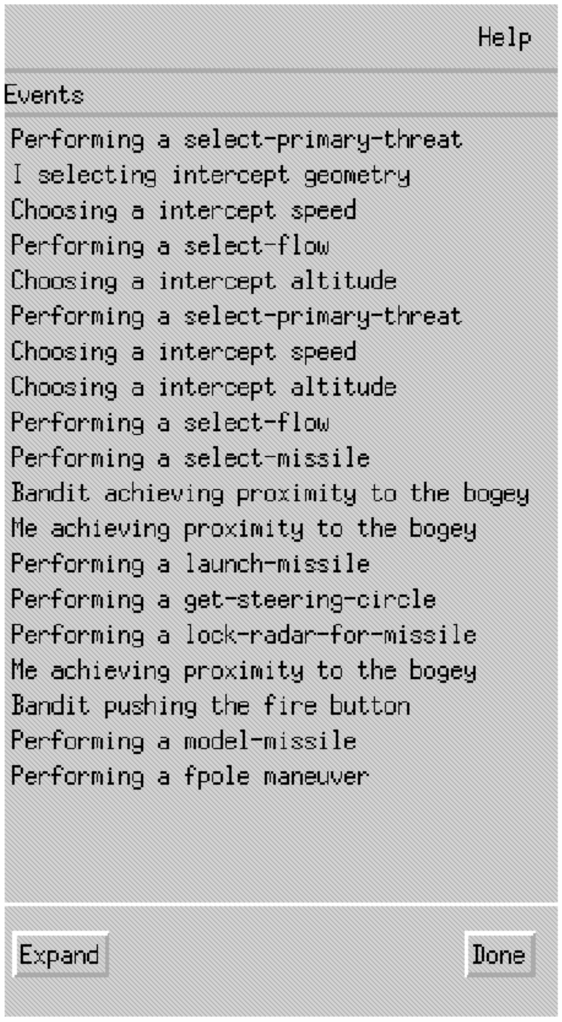
Steps 6 and 7 involve the use of Debrief to analyze a run where O_TRACK was applied successfully, and learn rules (chunks) that identify key features relevant to this success. As described in the previous section a user requests Debrief to explain a decision, it analyzes the situational factors leading to the decision, and builds chunks that recognize those factors in future decisions. Rules learned in steps 6 and 7 are used in step 8 to focus the comparison with the situation recalled in step 5.[4]

6 Experimental Results

We have implemented some key aspects of the above approach in an intelligent pilot agent for the air-combat simulation environment [37]. Specifically, we have an agent that implements steps 4-8 in Figure 5, while relying on simple heuristics in step 4. Steps 5, 6 and 7 require are critical in DEFT — they are the heart of its focusing — and mandate the application of an autonomous explanation capability. To this end, the the RESC agents developed for tracking [38, 35] were integrated with the Debrief explanation capability [7]. (As mentioned earlier, these agents are based on the Soar integrated architecture [16], and use chunking, a form of EBL, as the basis of all of their learning [11]). Figure 6 shows a snapshot of the Debrief user interface listing some of the

[4]Chunks from steps 6 and 7 were in reality hand-edited to eliminate some of their overspecificity; otherwise they would not fire appropriately in step 5. Learning chunks at the right level of generality, especially in a domain with continuously varying numerical values is an issue for future work, not only with DEFT, but with other learning systems as well.

conclusions drawn by the tracker, primarily about the opponent's actions, which may be selected and explained.



```
Performing a select-primary-threat
I selecting intercept geometry
Choosing a intercept speed
Performing a select-flow
Choosing a intercept altitude
Performing a select-primary-threat
Choosing a intercept speed
Choosing a intercept altitude
Performing a select-flow
Performing a select-missile
Bandit achieving proximity to the bogey
Me achieving proximity to the bogey
Performing a launch-missile
Performing a get-steering-circle
Performing a lock-radar-for-missile
Me achieving proximity to the bogey
Bandit pushing the fire button
Performing a model-missile
Performing a fpole maneuver
```

Figure 6: Debrief's user interface listing some of the conclusions drawn by the tracker, primarily about the opponent's actions.

We are exploring DEFT within this integrated pilot agent. The following is a discussion of the 25-degree nose-off example from Figures 1 and Figure 2. To experiment with this example, two scenarios were set up, in which the tracker and trackee's aircraft were set up in identical initial states. However, in one scenario the trackee fired its missile after attempting 0-5 degrees nose-off; and in the second scenario, the trackee attempted 25 degrees nose-off for missile firing. In the first scenario, the tracker successfully predicted a missile firing and evaded the missile; while in second scenario, the tracker, from its perspective, was unexpectedly shot down.

The tracker's failure in the second scenario triggers DEFT. We have simplified DEFT's steps 1 through 3, assuming they are capable of identifying *push-fire-button* as the culprit tracking operator (O_TRACK in Figure 5). Step 4 involves pin-pointing the spot when *push-fire-button* should have applied but did not. Among the heuristics discussed in the previous section, DEFT at present relies on the simplest — searching for the first tracking failure. Step 5 involves recalling and reconstructing the state at the point of this failure; we have implemented this step by exploiting Debrief's recall capability.

Steps 6 and 7 require recalling a past episode of successfully tracking *push-fire-button*, and identification of the features critical to this success. DEFT's results are shown in Figure 7-a. Once again, in the figure, S1 is the root, arcs are labeled with attributes and nodes are values, e.g., **BOGEY**(S1, B1) and **ID**(B1, BANDIT). Thus, DEFT has determined that only a handful of attributes are really critical for explaining why *push-fire-button* is the appropriate operator to apply when tracking B1. (Figure 7 should be contrasted with Figure 4).

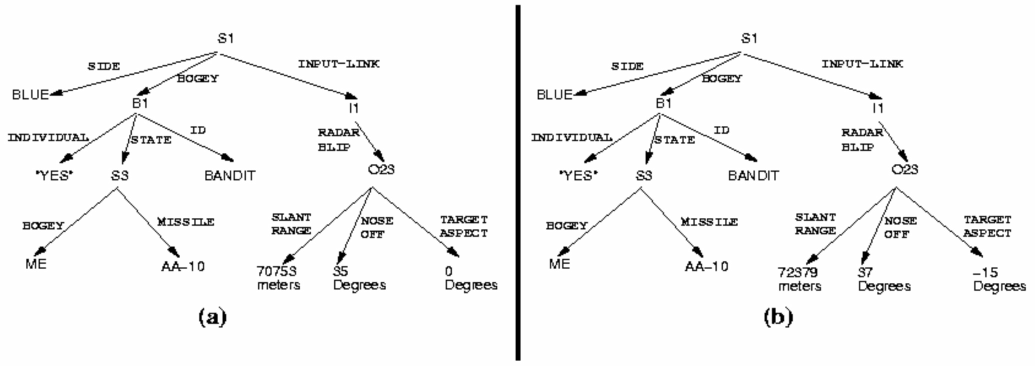


Figure 7: (a) Critical attributes identified by DEFT in case of tracking success (labels on arcs are attributes, while the nodes are values); (b) Values of those critical attributes in the case involving tracking failure.

Figure 7-b presents a portion of the results of step 8. From the state recalled in step 5, DEFT has retrieved values of the attributes found critical in step 7. DEFT's comparison yields three differences: (i) Slant-range (i.e., 3-D range between the two aircraft) is 70753 meters in the success case, but 72379 meters in the case of tracking failure; (ii) Nose-off has changed from 35 to 37 degrees; and (iii) Target-aspect has changed from 0 to -15 degrees. The key point here is that the deviation in nose-off 2/35 is just about 5% , while the deviation in slant-range 1626/70753 is just about 2%. In contrast, the deviation in target-aspect is really large. Indeed, assuming that target aspect values cases will fall withing 0-5 degree range, the deviation is really 200%.

Thus, target aspect is seen to be the key culprit. In step 9 the application of *push-fire-button* for tracking should be generalized, so that it is applicable even if target-aspect is at least as high as 15 degrees (or as low as -15 degrees); rather than restricting it to 0-5 degrees. Step 10 would test this generalization in a hypothetical replay. (There is an apparent discrepancy between Figure 7 and Figure 2 in that in Figure 2 the nose-off is seen to deviate, rather than target aspect. The key to resolving this discrepancy is to recognize that the nose-off and target-aspect are symmetric relationships, i.e., the bogey's nose-off is identical to a pilot-agent's own target-aspect and vice versa.)

While in these experiments, DEFT has narrowed down the critical differences to just three features, and then further to just one feature, obviously, DEFT may not necessarily meet similar success in other situations. In such cases, analysis of additional tracking success and failure episodes may be essential to narrow down the differences — an important issue for future work.

7 Related Work

Research in multi-agent systems has experienced tremendous growth in the recent years. In a survey paper by Stone and Veloso [33], multi-agent systems are classified along the dimensions of agent heterogeneity and communications. In this taxonomy, our adaptive tracking agents can in principle be viewed as "heterogeneous and communicating" although in this particular implementation we only used homogeneous agent models and ignored the communication between agents. In general, the DEFT approach can be applied easily to heterogeneous agents as long as the tracker has a good model of the the trackee agents. Likewise, communication does not introduce difficulties, just the opposite — the more communication there is, the easier it is to form an accurate model. For example, there is less need for a detailed, runnable trackee model in domains where the trackee is perfectly willing to tell you what his/her goals and intentions are.

Similar to the approach described in this paper, Pearson and Laird [17] also studied the problem of correcting an existing model using discriminations. In their work, errors are assumed when the system reaches a point where no operators can be proposed for the current goal, and they compare the failed attempt's operator sequence with a successful solution and use the differences to identify the operators in error. Although their work and ours share the same philosophy, we do not assume that two sequences share the same states for detecting the faulty operators. Instead, our discrimination process applies to any two episodes where there are states that share the same tracking decisions. In addition, we use explanation for focusing on features that are critical for the purpose of discrimination.

The work described in this paper is also related to the work of learning finite state machines [23, 2, 28] and reinforcement learning [34, 41]. However, the states in the Soar agents have a great number of internal structures and features, while state-machine-based learning assumes states are atomic. This difference has two implications. First, the number of states is so extremely large that it is beyond the scale of most state-machine-based learning approaches. Second, the structured states offer advantages when discriminating instances and thus learning can proceed rapidly.

In terms of using knowledge to guide the learning process, theory revision (e.g. FOCL [13]) is another large body of research that is related to our work. Given a set of positive and negative examples and an approximate theory T, a theory revision system can produce a revised theory RT of T such that RT will correctly classify the given examples. However, most theory revision methods are based on calculations of information gains of a set of given literals with respect to the examples, while our approach relies on the explanation and discrimination in order to select the critical theory elements (features) to revise our operators. This difference enables our approach to apply well to a very small number of instances.

8 Summary and Future Work

Agent tracking, a central requirement in many multi-agent worlds, can run into difficulties if the tracker possesses an imperfect model of the trackee. This problem is especially important in real-world situations where a tracker faces resource constraints and imperfect information, and a trackee may itself modify its behavior dynamically. To address this problem, this article proposes an approach called DEFT, for adaptive agent tracking. DEFT is a hybrid approach, relying on discrimination-based learning for necessary inductive leaps, and on its self-explanation to focus the discrimination to relevant features of the environment. This focusing is critical to meet the constraints posed by real-world multi-agent domains — thousands of features are possible as candidates for discrimination among a small number of training instances. Key aspects of DEFT have been implemented in a real-world synthetic air-to-air combat domain, with promising initial results.

Beyond DEFT, the other contributions of this article include identification of the issues in adaptive agent tracking in real-world domains, particularly, the constraints faced by automated learners, and types of knowledge required to address some of these constraints. Furthermore, this work has revealed several important future research directions. Discovering general heuristics or techniques in place of the domain-dependent heuristics currently implemented in DEFT (see Figure 5) is an important item on our agenda. Additionally, the approach focused on generalization of applicable operators; in some cases, specialization of operators may be essential. Finally, DEFT would appear to generalize to enable adaptation of different types of models, not necessarily only of other agents. Such generalization is also an important item on our agenda.

9 Acknowledgments

This research was supported under subcontract to the University of Southern California Information Sciences Institute from the University of Michigan, as part of contract N00014-92-K-2015 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Research Laboratory (NRL); and under contract N66001-95-C-6013 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Command and Ocean Surveillance Center, RDT&E division (NRAD). Critical expertise and support has been provided by BMH Inc. One of the authors (Wei-Min Shen) was supported in part under Grant No. IRI-9529615 (National Science Foundation), in part under Contract Number F30602-94-C-0210 (Rome Laboratory of the Air Force Systems Command and the Advanced Research Projects Agency), and in part under Contract Number MDA972-94-2-0010 (Technology Reinvestment Program award funded by the Advanced Research Projects Agency) and by the State of California under Contract Number C94-0031.

References

[1] J. R. Anderson, C. F. Boyle, A. T. Corbett, and M. W. Lewis. Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, 42:7–49, 1990.

[2] D. Angluin. Learning regular sets from queries and counter-examples. *Information and Computation*, 75(2):87–106, November 1987.

[3] J. Bates, A. B. Loyall, and W. S. Reilly. Integrating reactivity, goals and emotions in a broad agent. Technical Report CMU-CS-92-142, School of Computer Science, Carnegie Mellon University, May 1992.

[4] R. B. Calder, J. E. Smith, A. J. Courtemanche, J. M. F. Mar, and A. Z. Ceranowicz. Modsaf behavior simulation and control. In *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*, 1993.

[5] J. Cremer, J. Kearney, Y. Papelis, and R. Romano. The software architecture for scenario control in the Iowa driving simulator. In *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*, 1994.

[6] B. Hayes-Roth, L. Brownston, and R. V. Gen. Multiagent collaobration in directed improvisation. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*, 1995.

[7] W.L. Johnson. Agents that learn to explain themselves. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1257–1263, Seattle, WA, August 1994. AAAI, AAAI Press.

[8] W.L. Johnson. Pedagogical agents for virtual learning environments. In *Proceedings of the International Conference on Computers in Education*, pages 41–48, Singapore, 1995. AACE.

[9] A. Kautz and J. F. Allen. Generalized plan recognition. In *Proceedings of the National Conference on Artificial Intelligence*, pages 32–37. Menlo Park, Calif.: AAAI press, 1986.

[10] Y. Kuniyoshi, S. Rougeaux, M. Ishii, N. Kita, S. Sakane, and M. Kakikura. Cooperation by observation: the framework and the basic task pattern. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1994.

[11] J. E. Laird, P. S. Rosenbloom, and A. Newell. Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1(1):11–46, 1986.

[12] R. L. Lewis. An architecturally-based theory of human sentence comprehension. In *Proceedings of the Annual Conference of the Cognitive Science Society*, 1993.

[13] Pazzani M. and Kibler D. The utility of knowledge in inductive learning. *Machine Learning*, 9(1):57–94, 1991.

[14] S. Minton, M. D. Johnston, A. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the National Conference on Artificial Intelligence*, 1990.

[15] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[16] A. Newell. *Unified Theories of Cognition*. Harvard Univ. Press, Cambridge, Mass., 1990.

[17] J. D. Pearson and J. E. Laird. Toward incremental knowledge correction for agents in complex environments. *Machine Intelligence*, 15, 1996.

[18] K. Pimentel and K. Teixeira. *Virtual reality: Through the new looking glass*. Windcrest/McGraw-Hill, Blue Ridge Summit, PA, 1994.

[19] R. J. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[20] R. J. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[21] A. S. Rao. Means-end plan recognition: Towards a theory of reactive recognition. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR-94)*, 1994.

[22] J. Rickel and W.L. Johnson. Pedagogical agents for immersive training environments. Accepted for publication in the *Proceedings of the First International Conference on Autonomous Agents*, 1997.

[23] R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 1993.

[24] P. S. Rosenbloom, J. E. Laird, A. Newell, , and R. McCarl. A preliminary analysis of the soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47(1-3):289–325, 1991.

[25] W.M. Shen. Complementary discrimination learning: A duality between generalization and discrimination. In *Proceedings of Eighth National Conference on Artificial Intelligence.* MIT Press, 1990.

[26] W.M. Shen. Complementary discrimination learning with decision lists. In *Proceedings of Tenth National Conference on Artificial Intelligence.* MIT Press, 1992.

[27] W.M. Shen. Discovery as autonomous learning from the environment. *Machine Learning*, 12:143–165, 1993.

[28] W.M. Shen. Learning finite state automata using local distinguishing experiments. In *Proceedings of IJCAI-93*, Chambery, France, August 1993.

[29] W.M. Shen. *Autonomous Learning from the Environment.* W. H. Freeman, Computer Science Press, 1994.

[30] W.M. Shen and H.A. Simon. Fitness requirements for scientific theories containing recursive theoretical terms. *British Journal for the Philosophy of Science*, pages 641–652, 1993.

[31] F. Song and R. Cohen. Temporal reasoning during plan recognition. In *Proceedings of the National Conference on Artificial Intelligence.* Menlo Park, Calif.: AAAI press, 1991.

[32] The DIS steering committee. The dis vision: A map to the future of distributed simulation. Technical Report IST-SP-94-01, Institute for simulation and training, University of Central Florida, Orlando, May 1994.

[33] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *IEEE Transaction on Data and Knowledge Engineering*, (Submitted), 1997.

[34] R. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Machine Learning Conference*, 1990.

[35] M. Tambe. Recursive agent and agent-group tracking in a real-time dynamic environment. In *Proceedings of the International Conference on Multi-agent systems (ICMAS)*, 1995.

[36] M. Tambe. Tracking dynamic team activity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1996.

[37] M. Tambe, W. L. Johnson, R. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), Spring 1995.

[38] M. Tambe and P. S. Rosenbloom. RESC: An approach for real-time, dynamic agent tracking. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[39] M. Tambe and P. S. Rosenbloom. Architectures for agents that track other agents in multi-agent worlds. In *Intelligent Agents, Volume II: Lecture Notes in Artificial Intelligence 1037*. Springer-Verlag, Heidelberg, Germany, 1996.

[40] B. Ward. *ET-Soar: Toward an ITS for Theory-Based Representations.* PhD thesis, School of Computer Science, Carnegie Mellon Univ., 1991.

[41] C. Watkins and P Dayan. Technical note: q-learning. *Machine Learning*, 8(3/4), 1992.