Adaptive Interfaces for Value-Driven Agents

Daniel Shapiro, Melinda Gervasio

Institute for the Study of Learning and Expertise
2164 Staunton Court, Palo Alto, CA 94306
{shapiro,gervasio}@isle.org

Abstract

As autonomous agents perform increasingly important and complex tasks, they have the potential to greatly expand human capabilities. For this collaboration to be successful, a human must be able to effectively communicate goals and preferences to the agent, so the agent's choices will be consistent with those of the human user. We present our plans for integrating two lines of research towards this end: an architecture for value-based agents, called Icarus, which employs a numeric utility model to resolve choices in skills; and research on adaptive user interfaces that acquire a numerical model of user preferences from experience. Taken together, these capabilities offer a means of tailoring autonomous agent behavior in complex domains.

Introduction

Autonomous agents are technological artifacts that perform tasks for humans. They sense the environment, relate perceptions to tasks, and apply an appropriate response. Because they carry out complex functions outside direct supervision, they can considerably expand human capabilities. However, they must also interact with humans enough to ensure they pursue the tasks required of them. In particular, they must be able to accept task statements at different levels of specificity and, ideally, learn to carry out those tasks in ways that their human overseers find acceptable. In this paper, we examine two lines of earlier research that address these challenges and our plans to merge them in future work.

These two approaches share a key assumption that we have stated elsewhere (Shapiro & Langley, 2002) as the *separation* hypothesis:

> An effective way to construct intelligent agents is to encode legal alternatives separately from preferences over those alternatives.

A clear benefit of this framework is that it lets one assume that legal behaviors are fixed, so that other influences, such as user advice and learning, can focus on a constrained set of preferences. This paradigm also seems appropriate when one wants to create a number of agents that exhibit distinctive behaviors within the same domain.

Some research on architectures for intelligent agents relies on the separation hypothesis. The strongest example is Prodigy (Minton, 1990), which encodes legal knowledge as Strips-like operators and inference rules, but which stores preferences as a separate set of rules. An even more familiar example comes from work on game playing, in which systems typically specify knowledge about legal moves as all-or-none rules but encode preferences about them as numeric evaluation functions. Our research has taken a similar approach to partitioning behavior into legal skills and numeric values associated with them.

In the pages that follow, we propose an approach to human interaction with autonomous agents that relies centrally on the separation hypothesis. We begin by reviewing Icarus, an architecture for autonomous agents that represents hierarchical reactive skills, uses numeric value functions to select among them, and utilizes reinforcement learning to estimate these functions from reward signals. After this, we describe work on adaptive interfaces that assist humans in making decisions, collect traces of user choices, and from these traces learn user profiles - cast as numeric functions - for utilization on future problems. Next we consider ways to incorporate ideas from such adaptive interfaces into Icarus, illustrating the approach with examples from a driving domain. In closing, we discuss related research and summarize our contributions and plans.

Icarus

Icarus is a language for specifying the behavior of artificial agents that learn. Its structure is dually motivated by the desire to create practical agent applications for dynamic environments, and the desire to support policy learning in a computationally efficient way. As a result, we have cast Icarus as a reactive language. It supplies a representation for expressing plans, together with an interpreter for evaluating plans that employs a repetitive sense-think-act loop. This repetition provides adaptive response; it lets an agent retrieve a relevant action even if the world changes from one cycle of the interpreter to the next.

Icarus is an instance of an *extremely* reactive language, as its interpreter will retrieve a relevant action even if the world to changes from one state to any other recognized by the plan in exactly one time step. It shares the logical orientation of teleoreactive trees (Nilsson, 1994) and universal plans (Schoppers, 1987), but adds vocabulary for expressing hierarchical intent, as well as tools for problem decomposition found in more general-purpose languages. For example, Icarus supports function call, Prolog-like

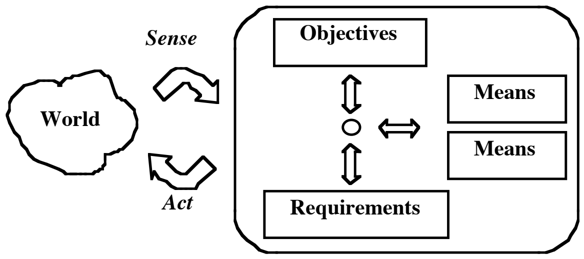

Figure 1. The structure of Icarus plans.

parameter passing, pattern matching on facts, and recursion.

An Icarus program contains up to three elements: an objective, a set of requirements, and a set of alternate means (or methods for achieving objectives), as illustrated in Figure 1. Each of these can be instantiated by further Icarus plans, creating a logical hierarchy that terminates with calls to primitive actions or sensors. Icarus evaluates these fields in a situation-dependent order, beginning with the objective field. If the objective is already true in the world, evaluation succeeds and nothing further needs to be done. If the objective is false, the interpreter examines the requirements field to determine if the preconditions for action have been met. If so, evaluation progresses to the means field, which is the locus of all choice in Icarus. Over time, Icarus will learn to select the alternative in the means field that promises the largest expected reward.

We illustrate this reactive interpretation process using a simple program for freeway driving shown in Table 1. It contains an ordered set of objectives implemented as further subplans. Icarus repetitively processes this plan, starting with its first statement every execution cycle. The first clause in Table 1 defines a reaction to an impending collision. If this context applies, Icarus returns the slam-on-the-brakes action for application in the world. If emergency braking is not required, evaluation proceeds to the second clause, which specifies a plan for reacting to trouble ahead, defined as a car traveling slower than the agent in the agent's own lane. This subplan contains options, as shown in Table 2. Here, the agent can move one lane to the left, move right, slow down, or cruise at its current speed and lane, although it does not allow the agent to speed up. Icarus makes a selection based on the long-term expected reward of each alternative.
If there is no imminent collision or trouble in front, Icarus will select an action from a later clause in Table 1. Get-to-target-speed is goal-driven subplan that causes the agent to speed up if it is traveling too slow or slow down if it is moving too fast. Avoid-trouble-behind specifies several alternate reactions to a faster car approaching from behind. If none of these contexts match, the fifth clause encodes the default case: a primitive action that causes the agent to Cruise in its current lane, at its current speed.

Table 1. The top level of an Icarus freeway-driving plan.

```
Drive ()
  :objective
   [ *not* (Emergency-brake())
     *not* (Avoid-trouble-ahead())
      Get-to-target-speed()
      *not* (Avoid-trouble-behind())
      Cruise()]
```

Since Icarus plans contain choice points, the interpreter needs a method of selecting the right option to pursue. Icarus provides this capability by associating a value estimate with each Icarus plan. This number represents the expected future discounted reward stream for choosing a primitive action (or subplan) on the current execution cycle and following the policy being learned thereafter. Icarus computes this expected value using a linear function of current observations.

The estimation architecture addresses an interesting tension in information needs. On one hand, the value of a plan clearly depends upon its context; the future of 'decelerate' is very different if the car in front is close or far. On the other hand, the cardinal rule of good programming is "hide information". We should not force Icarus programmers to define subplans with a suite of value-laden parameters that are irrelevant to performing the task at hand. Our solution is to inherit context-setting parameters down the calling tree. Thus, Avoid-trouble-ahead (Table 2) measures the distance to the car in front, and Icarus implicitly passes that parameter to the decelerate action several levels deeper in the calling tree. The programmer writes Icarus code in the usual fashion, without concern for this implicit data.

Table 2. An Icarus plan with alternate subplans.

```
Avoid-trouble-ahead ()
 :requires
  [  bind (?c, car-ahead-center())
     velocity() > velocity(?c)
     bind (?tti,  time-to-impact())
     bind (?rd, distance-ahead())
     bind (?rt, target-speed() - velocity())
     bind (?art, abs(?rt)) ]
  :means
   [ safe-cruise(?tti, ?rd, ?art)
     safe-slow-down (?tti, ?rd, ?rt)
     move-left (?art)
     move-right (?art) ]
```

Learning with SHARSHA

SHARSHA is a reinforcement learning algorithm mated to Icarus plans. It is a model-free, on-line technique that determines an optimal control policy by exploring a single, infinitely long trajectory of states and actions. SHARSHA (for State Hierarchy, Action, Reward, State Hierarchy, Action) adds hierarchical intent to the well-known SARSA algorithm (for State, Action, Reward, State, Action).

While a detailed description of SHARSHA is outside the scope of this paper, we can compare it to SARSA using Figure 2 as a graphical aid. Like SARSA, SHARSHA learns an estimate for the value of taking a given action in a given state by sampling its future trajectory. However, unlike SARSA, SHARSHA operates on *stacks* of state-action pairs, where each pair corresponds to an Icarus function encoding a plan to pursue a course of action in a given situation. For example, at time 1 an Icarus agent for piloting a car might accelerate to reach its target speed in order to drive, while at time 2 it might brake in order to avoid trouble as part of the same driving skill. Where SARSA observes the current state, SHARSHA observes the calling hierarchy, and where SARSA updates the current state, SHARSHA updates the estimates for each function in the calling stack (using the estimate associated with the *primitive* action of the destination stack, which contains the most informed picture of world state). Our implementation of SHARSHA employs linear value approximation functions, and it learns the coefficients of these functions from delayed reward. We have proven SHARSHA's converge properties under a common set of Markov assumptions (Shapiro, 2001).

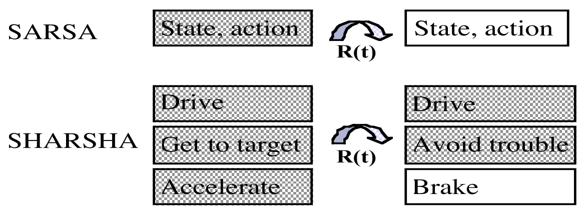

Figure 2. A comparison of SARSA and SHARSHA.

Results with Icarus

We have used Icarus to generate a number of interesting results concerning the power of the value-driven agent model. We report on three of those results here: (1) an experiment on 'programming by reward' which supports our hypothesis that values can be treated separately from skills; (2) a comparison between SHARSHA and SARSA that demonstrates a two order of magnitude speedup in learning (this speaks to the practicality of employing value learning to acquire behavior); and (3) a proof that agent Reward functions can always be aligned with human utility. This last result addresses trust, and our willingness to deploy autonomous agents in complex domains.

We conducted our experiments in a freeway-driving domain, which consists of a simulator (written in C) that manages several hundred cars on a three-lane, endless loop freeway with no entrances or exits. With the exception of one "smart" car that is capable of learning, each vehicle maneuvers according to a fixed situation-action map. An Icarus program controls the smart car. It can sense its own target speed, the presence and relative velocity of six surrounding cars, the distance to the car ahead and behind, and whether it is safe to change lanes. There are six primitive actions: speed up by two mph, slow down by two, cruise at the current speed, change lane to the left, change lane to the right, and emergency brake. The last action instantaneously slows the vehicle such that it will not collide with the car in front.

An experiment in programming by reward

A key tenet of the value driven agent model is that the structure of skills can be productively separated from the representation of preference, or values. If this separation hypothesis holds, we should be able to define a single skill for a common activity and produce a set of artificial agents with qualitatively distinct behavior simply by changing the reward function that serves as the target of learning.

We performed this experiment using the skill outlined in Tables 1 and 2, and the reward functions in Table 3. Each function is linear in the indicated features. For example, the airport driver is solely motivated by the desire to get to the airport on time, and becomes less happy as its velocity deviates from target speed. The safe driver wants to avoid collisions. Its reward function penalizes small times to impact with cars in front and cars behind - the shorter the time to impact, the larger the penalty. The goldfish driver has an imaginary fishbowl as luggage and does not want maneuvers to upend the fish. Alternatively, we can think of it as a bit queasy, since its reward function penalizes all forms of maneuver. Finally the reckless driver is out for thrills; it garners reward for near misses, and penalizes deviations from its target speed.

Table 3. Reward functions used as targets of learning.

	Time to impact ahead	Time to impact behind	Delta from target speed	Slow down	Speed up	Change lanes
Airport Driver			□			
Safe Driver	+	+				
Goldfish Driver				□	□	□
Reckless Driver	□	□	□			

We unleashed each of these agents to develop an action policy via SHARSHA, using the identical skill for driving. We profile the resulting behaviors in Figure 3 (see Shapiro & Langley, 2002 for details). The most obvious feature is that the agents evolve strikingly different policies as a byproduct of learning. For example, the safe driver shows the highest difference from target speed in the chart, although it was not motivated (positively or negatively) by that quantity. Apparently, it readily adjusts its velocity to avoid potential collisions. The safe driver also shows the largest following distance, which makes intuitive sense since safe drivers know that tailgating produces potential

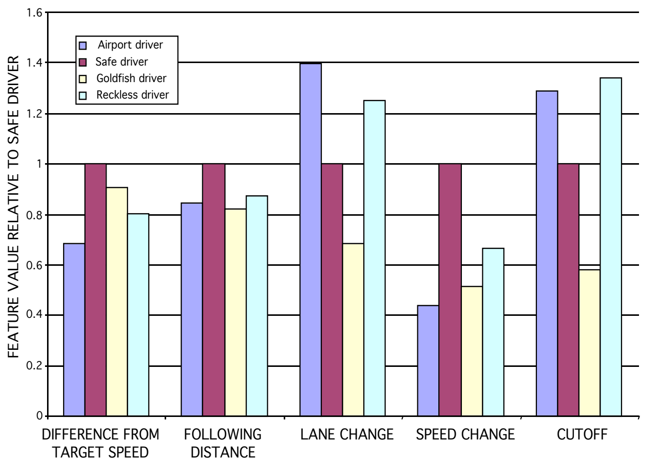

Figure 3. A profile of learned behaviors.

collisions. (If the car in front slows down, the safe driver inherits a significant penalty.) In contrast, the goldfish driver has the shortest following distance, similar to the airport driver's. We explain this observation by a cruise control effect: drivers who resist velocity changes will tend to creep up on the car in front. Finally, the goldfish driver performs the fewest cutoff actions (defined as a lane change in front of a faster vehicle), as it is motivated to avoid all maneuvers. In contrast, the reckless driver actively seeks potential collisions, as they contribute positive terms to its reward.

Note that the Icarus program for driving prevents the reckless agent from simply colliding with the car in front; its opportunity to learn is confined to an allowable realm. Said differently, the agent's skills define legal behavior. Its reward function is irrelevant whenever the behavior is determined, and relevant only when choice is allowed. This design frees us to construct reward functions for Icarus agents in an unconstrained way.

Learning with background knowledge

Our second empirical investigation examines Icarus' ability to speed learning by encoding domain knowledge into the structure of a plan. We measured the impact of plan structure on learning by comparing the behavior of two Icarus agents with the same Reward function but different reactive programs (Shapiro, et al. 2001). The first employed a standard reinforcement learning design; a "flat" plan representing all feasible actions in every situation. The second employed a "structured" plan, specifically the driving program outlined in Tables 1 and 2. Since the structured program contains fewer options (but defines a smaller space of possible policies), we expected the structured agent to learn faster, but achieve a lower level of asymptotic reward.

The results of this experiment were surprising. As expected, the structured agent learned faster than the flat agent, but by approximately two orders of magnitude. In addition, the structured agent performed better than the flat agent even after 250,000 iterations of learning in the driving domain. This leverage was largely due to reduced plan size. When we counted the number of state-action pairs in each design (somewhat after generating the learning curves), we discovered a three order of magnitude difference: the standard (flat) statement of the vehicle control problem contained over 20,000 primitive alternatives, while the hand-coded, structured plan contained exactly 24 state-action pairs. Since our learning algorithm (and reinforcement learning in general) searches the space of possible action sequences, it is not surprising that the "flat" agent failed to identify the better policy in any reasonable period of time. In contrast, our ability to obtain a two order of magnitude speed-up suggests that the separation hypothesis is viable in practice, and that we can employ value learning to refine agent behavior in non-trivial domains.

User-agent value alignment

In addition to programming an agent via its reward function, we can design its reward function to be aligned with human utility. The issue here is *communication*; since humans and artificial agents necessarily sense the world in different terms, we need a mechanism for capturing human utility in a form that agents can represent. Value alignment is a method of bridging this gap. In particular, *structural value alignment* is a conditional independence relation that ensures an agent can only impact human utility via features the agent cares about as well. Given this relationship, we can define a numerical manipulation of the agent's reward function that produces *functional value alignment*; the happy situation in which the agent will always choose the action preferred by the user. We have shown that it is always possible to establish structural alignment, and thus that we can achieve functional alignment between any user and any agent (Shapiro, 2001).

Our mechanism for establishing alignment requires human designers to (a) instrument the agent with features that serve as surrogates for a subset of the features that determine human utility, and (b) conduct a series of simultaneous assessment experiments that set the agent's Reward for a perceived situation, **x**, equal to the human's expected utility for the corresponding situation, **y**. This is always possible (as **x** can be the full set of agent actions and observations), but the assessment problem can become quite large in the absence good surrogate features. We are exploring methods for minimizing this cost.

User-agent value alignment has a profound consequence that we call the "be all you can be" guarantee: it ensures that an aligned agent will do everything it possibly can to maximize human utility, and it tells us this fact before the agent is deployed to learn from its environment. This property should increase our willingness to trust highly autonomous artificial agents, and thus to employ them in a greater variety of applications.

In summary, our work with Icarus has shown that the value-driven agent architecture provides many desirable abilities. We can: program by value without implementing new skills; employ value learning in a practical way; and elevate the problem of behavior verification into a discourse about value alignment. However, many tasks require more coordination between agents and users than Icarus currently supports, since it places all such interactions at design time. The next section discusses our work on adaptive user interfaces, which support an on-line interaction over value.

Adaptive Interfaces

Adaptive user interfaces (Langley, 1999) are assistant or advisory systems that learn to tailor their behavior to individual users by utilizing profiles that capture user preferences. Through the use of machine learning techniques, these systems can automatically acquire user profiles from traces of user decisions and thus provide increasingly personalized service with experience. Over the past several years, we have developed a variety of adaptive user interfaces, including a driving route advisor (Rogers et al., 1999), a crisis response scheduler (Gervasio et al., 1999), a destination advisor (Goker & Thompson, 2000), and a stock tracker (Yoo et al., 2003). Given a trace of user decisions, these systems apply machine learning to acquire content-based user profiles that represent user preferences in terms of item attributes. They also emphasize unobtrusive data collection techniques, which extract preference information from a user's natural interaction with the system.

For example, INCA (Gervasio et al., 1999) is a scheduling assistant for hazardous material incident response that learns a user's preferences for addressing the different hazards involved in a situation. Given an incident, it carries out an interactive search through the space of possible response schedules. On each cycle, INCA generates a set of candidate schedules and shows them to the user, who selects one for further development. This interaction repeats until INCA generates a schedule that the user finds acceptable. In evaluating candidate schedules, INCA relies on a utility function (a linear evaluation function) on schedule attributes. The user's decisions provide feedback regarding the evaluation and are translated into training data for updating the weights of the utility function using a modified perceptron algorithm. Experimental studies with INCA have shown that this approach can be used to acquire user profiles that reflect different individual preferences. Through these profiles, INCA can provide increasingly personalized assistance to support more effective crisis response.

INCA also demonstrates a variable level of autonomous behavior. In particular, a user can tell INCA how it should attempt to improve the current schedule at different levels of detail. At the highest level, a user may simply select, "improve schedule", which lets INCA explore improvements along all dimensions. At the lowest level, a user can choose to address a specific hazard by specifying particular actions, start times, durations, and resources. Experimental studies have shown that more detailed guidance leads to more rapid learning. However, by providing users with variable levels of assistance, INCA can encompass not only different preferences regarding crisis response schedules but also individual differences in the amount and type of assistance desired or required.

In designing our adaptive user interfaces, we have focused on exploiting the preference information that is implicit in a user's decisions. For example, by selecting a schedule in INCA, the user indicates a preference for that schedule over the others presented. When asking for improvement along particular dimensions, the user indicates the importance of the associated features. Our general approach to adaptive user interfaces does not exclude the use of more explicit user input such as numeric evaluations, relative rankings, or even statements about general preferences. However, by focusing on the information that is readily available through a system's interaction with the user, we can support more natural interfaces that require little or no extra effort on the human user's part to achieve personalization.

While our work to date has demonstrated the ability to acquire user preferences in a non-obtrusive way, we have focused on recommendation tasks that require very little autonomy on the part of the underlying system. At most, they have involved one-step modifications of already generated alternatives, such as the schedules in INCA or the driving routes in the Adaptive Route Advisor (Rogers et al., 1999). However, the same technology should support dialogues over preference in more complex applications, such as the interaction between humans and highly autonomous artificial agents.

An adaptive interface for Icarus

We believe that value-driven architectures like Icarus and adaptive interfaces like INCA hold great potential for synergy. As we have noted, the current version of Icarus cannot learn from user advice, while adaptive interfaces support one-step decision-making tasks rather than complex control problems. However, since both frameworks view behavior as emerging from the selection of legal alternatives based on numeric value functions, their integration should be relatively straightforward. Our approach is to augment the Icarus architecture to incorporate techniques from adaptive interfaces for collecting, and learning from, user input.

We can illustrate this synthesis in the domain of freeway driving, this time assuming that there is a human passenger who can give advice and directions to an autonomous limousine driver. The Icarus agent should both follow any legal advice that it receives from the backseat driver and, over time, learn the (opinionated) passenger's preferences so that he must give fewer directions. The autonomous agent pilots the car by executing an Icarus program like that in Tables 1 and 2, while the user observes the traffic situation and provides advice of the form, “Slow down!” “Change lanes to the right!” and "Go around that guy!" whenever he/she disagrees with the agent's current choice.

This form of user input differs somewhat from that assumed by most adaptive interfaces. For example, INCA presents its user with explicit choices about revisions to a schedule, and asks the user to select one over the others. However, the literature on adaptive interfaces contains a few examples of systems that let users correct undesired behaviors. For example, Billsus and Pazzani's (1999) News Dude lets its user interrupt a news story he does not want to hear, whereas Hermens and Schlimmer's (1994) adaptive form filler lets its user overwrite entries he thinks should have different values.

This example reflects another difference from traditional adaptive interfaces, in that the system can accept advice at different levels of detail. Icarus is well suited to support this facility, because its hierarchy of skills lets users specify higher-level actions (e.g., passing) without stating the details of how the agent should execute them (e.g., on the left or right). INCA also has this capability, though it is perhaps the only adaptive interface to date that supports multi-level advice.

We must also modify Icarus to learn from traces of user corrections that it collects in this manner. Recall that the current architecture estimates the value functions associated with each skill from a reward signal it observes on each time step. In the new setting, the agent receives corrections when the user disagrees with its action and no feedback on other cycles. Thus, the user's feedback concerns preferred action rather than the numeric data assumed in learning from delayed reward. In effect, the user forces the agent to alter its policy, making the traditional concern with policy exploration irrelevant.

Fortunately, we can adapt the method of learning used in INCA almost directly to the new version of Icarus. Suppose the agent is executing Cruise when the user says "Slow down", and slowing down is a legal skill in the current situation. In addition to executing this action, we want the system to alter the weights associated with the Slow down and Cruise skills so the latter has a higher value in this context. INCA utilized a modified version of the perceptron algorithm to revise the weights on its utility function, and Icarus can utilize a similar technique. The main difference is that the system must revise the parameters on an entire set of Icarus skills rather than a single utility function that represents the user profile.

We plan to implement this variation on Icarus and evaluate its ability to learn user preferences through a controlled experiment. In particular, we will initialize the agent with some fixed set of value estimation functions, provide it with user feedback during the course of its operation, and record the user corrections that occur over time. From this trace, we will utilize a sliding window to compute the probability of user corrections as a function of driving time. This metric will serve as a measure of user disapproval. We will also evaluate user satisfaction via a questionnaire.

In this context, we can make four predictions about the behavior of the system over time. First, the probability of user corrections will decrease monotonically as the agent gains experience with the user. Second, this probability will move towards an asymptote that reflects the best policy the agent can express within its legal skills. Third, the system will reach this point of diminishing returns after an acceptably small number of user corrections. Finally, we predict that addition of further parameters to value estimation functions will decrease that asymptote but slow down the rate at which the system approaches it.

These hypotheses describe the behavior of a successful merger between adaptive interfaces and value-driven agent architectures, as they depend upon the abilities to (a) learn numeric preferences from advice about action obtained in an unobtrusive way, and (b) to rapidly acquire action policies in the presence of background knowledge. In addition, we hope to compare this approach to a reinforcement learning design that induces action policies from numeric user feedback. We would expect the first approach to learn faster, since it relies on advice about action that obviates the attribution problem. We also anticipate that our questionnaires will reveal that users find it more natural to interact with the driving agent by stating preferred action. If these hypotheses are confirmed, we will have made significant progress toward developing a practical mechanism for tailoring the behavior of autonomous agents in complex environments.

Related work on Control Learning

As we have noted, our approach to human interaction with autonomous systems draws on two distinct research traditions. The first involves architectures for intelligent agents, which typically represent knowledge about problem solving and action as condition-action rules, and which chain this knowledge together to produce complex behavior. Well-developed examples of this general approach include SOAR (Laird & Rosenbloom, 1990), Prodigy (Minton, 1990), and ACT-R (Anderson, 1993). Icarus diverges from these efforts by grouping knowledge into larger skills, but it shares ACT-R's decision-theoretic approach to selecting among alternative knowledge structures. Our architecture comes even closer to Sun et al.'s (2001) CLARION, which combines symbolic structures with numeric functions that it learns from delayed reward. However, systems cast within these architectures are usually designed to be autonomous, rather than to achieve a user's goals.

The second research tradition concerns adaptive user interfaces (Langley, 1999), which learn user profiles from traces of interactions with those users. These systems are highly interactive and devote themselves to serving the user's interests, but they have typically focused on classification tasks like filing email (e.g., Segal & Kephart, 1999), recommending news stories (e.g., Billsus & Pazzani, 1999), and suggesting stock trading actions (e.g., Yoo et al., 2003). In contrast, our own work in this area has concentrated on adaptive systems that help users solve complex problems like scheduling (Gervasio et al., 1999) and route finding (Rogers et al., 1999). However, we cast user profiles in terms of numeric utility functions and our systems infer the weights in these functions from data about user choices, which simplifies unification of this approach with the Icarus architecture.

We should also mention some additional research paradigms that are related to our approach, but which have influenced our thinking less directly. One approach, known as *advisable planning*, adapts planning methods to accept high-level advice from humans in the form of goals or constraints. However, work in this area (e.g., Myers, 1996) differs from ours by emphasizing advisory input at the outset of planning, rather than during execution. Moreover, work on advisable planning generally assumes new advice for each problem, rather than learning user preferences from each interaction that carry across sessions.

Another framework, known as *behavioral cloning*, aims to learn control policies by observing another agent's behavior. For example, Sammut (1996) and Anderson et al. (2000) describe systems that record traces of human's actions when flying a simulated aircraft, transform these traces into supervised training cases, and induce reactive policies for controlling the aircraft autonomously. This framework differs from ours in its reliance on detailed traces of human behavior rather than occasional and possibly high-level advice, and in its emphasis on learning unstructured policies rather than incorporating human preferences into a hierarchy of skills. Ichise et al. (2002) have used behavioral cloning methods to learn the structure of a skill hierarchy, but they have not combined this idea with value functions for selecting skills.

An older approach relies heavily on domain knowledge to interpret observed behaviors and thus to learn more complex controllers. This paradigm includes some, but not all, approaches to explanation-based learning (e.g., Segre, 1987), learning apprentices (e.g., Mitchell, 1985), and programming by demonstration (e.g., Cypher, 1993). However, these methods are concerned primarily with learning the qualitative structure of skills, rather than learning to select among alternative choices within an existing structure. Moreover, they emphasize nondynamic domains that do not require the type of reactivity we designed Icarus to support.

Summary

We have developed an approach that promises to support human interaction with autonomous systems in complex environments. The key tenet is that we can productively separate the concepts of value and skills. This allows us to express user preferences as value functions within an architecture for value-driven agents, and to employ adaptive interface technology to acquire appropriate value functions through user feedback about action. Our results to date suggest that this approach can express a wide variety of preferences and policies, that it offers a non-invasive user interaction model, and that it phrases the underlying learning problems in a computationally tractable form.

Acknowledgements

We thank Pat Langley for his many contributions to this work.

References

Anderson, C., Draper, B., & Peterson, D. (2000). Behavioral cloning of student pilots with modular neural networks. *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 25--32). Stanford: Morgan Kaufmann.

Anderson, J.R. (1993). *Rules of the mind.* Hillsdale, NJ: Lawrence Erlbaum.

Billsus, D., & Pazzani, M. (1999). A personal news agent that talks, learns and explains. *Proceedings of the Fifth International Conference on Autonomous Agents* (pp. 268--275). Seattle: ACM Press.

Cypher, A. (Ed.). (1993). *Watch what I do: Programming by demonstration.* Cambridge, MA: MIT Press.

Engelmore, R., and Morgan, A. eds. (1986). *Blackboard Systems*. Reading, Mass.: Addison-Wesley.

Gervasio, M. T., Iba, W., & Langley, P. (1999). Learning user evaluation functions for adaptive scheduling assistance. *Proceedings of the Sixteenth International Conference on Machine Learning* (pp. 152-161). Bled, Slovenia: Morgan Kaufmann.

Goker, M. & Thompson, C. (2000). Personalized conversational case-based recommendation. *Proceedings of the Fifth European Workshop on Case Based Reasoning*. Trento, Italy

Hermens, L. A., & Schlimmer, J. C. (1994). A machine-learning apprentice for the completion of repetitive forms. *IEEE Expert*, *9*, 28--33.

Ichise, R., Shapiro, D. G., & Langley, P. (2002). Learning hierarchical skills from observation. *Proceedings of the Fifth International Conference on Discovery Science*. Lubeck, Germany: Springer.

Laird, J.E., & Rosenbloom, P.S. (1990). Integrating execution, planning, and learning in SOAR for external environments. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 1022--1029). Boston, MA: AAAI Press.

Langley, P. (1999). User modeling in adaptive interfaces. *Proceedings of the Seventh International Conference on User Modeling* (pp.357-370). Banff, Alberta: Springer.

Minton, S.N. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, *42*, 363--391.

Mitchell, T.M., Mahadevan, S., & Steinberg, L. (1985). Leap: A learning apprentice for VLSI design. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 573--580). Los Angeles: Morgan Kaufmann.

Myers, K.L. (1996). Strategic advice for hierarchical planners. *Principles of Knowledge Representation and Reasoning Proceedings of the Fifth International Conference* (pp. 112-123). San Francisco: Morgan Kaufmann.

Nilsson, N. (1994). Teleoreactive programs for agent control. *Journal of Artificial Intelligence Research, 1,* 139-158.

Rogers, S., Fiechter, C., & Langley, P. (1999). An adaptive interactive agent for route advice. *Proceedings of the Third International Conference on Autonomous Agents* (pp. 198--205). Seattle: ACM Press.

Sammut, C. (1996). Automatic construction of reactive control systems using symbolic machine learning. *Knowledge Engineering Review*, *11*, 27--42.

Segal, R., & Kephart, J. (1999). MailCat: An intelligent assistant for organizing e-mail. *Proceedings of the Third International Conference on Autonomous Agents* (pp. 276--282). Seattle: ACM Press.

Segre, A. (1987). A learning apprentice system for mechanical assembly. *Proceedings of the Third IEEE Conference on AI for Applications* (pp. 112--117).

Shapiro, D. (2001). *Value-driven agents*. PhD thesis, Department of Management Science and Engineering, Stanford University, Stanford, CA.

Shapiro, D., Langley, P., & Shachter, R. (2001). Using background knowledge to speed reinforcement learning in physical agents. *Proceedings of the Fifth International Conference on Autonomous Agents* (pp. 254--261). Montreal: ACM Press.

Shapiro, D., & Langley, P. (2002). Separating skills from preference: Using learning to program by reward. *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 570-577). Sydney: Morgan Kaufmann.

Schoppers, M. (1987). Universal Plans for reactive robots in unpredictable environments. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 1039-1046). Morgan Kaufmann.

Sun, R., Merrill, E., & Peterson, T. (2001). From implicit skills to explicit knowledge: A bottom-up model of skill learning. *Cognitive Science*, *25*, 203--244.

Yoo, J., Gervasio, M., & Langley, P. (2003). An adaptive stock tracker for personalized trading recommendations. *Proceedings of the 2003 International Conference on Intelligent User Interfaces.* New Orleans, LA.