Model-Based Learning with Hierarchical Relational Skills

Pat Langley LANGLEY@CSLI.STANFORD.EDU
Sachiyo Arai SACHIYO@KUIS.KYOTO-U.AC.JP
Daniel Shapiro DGS@STANFORD.EDU
Computational Learning Laboratory, Center for the Study of Language & Information, Stanford University, Stanford, CA 94305 USA

Abstract

In this paper we describe Icarus, an architecture for physical agents that uses hierarchical skills to support reactive execution. We review an earlier version of the system, then present an extended framework that associates reward with stored concepts and utilizes a model-based approach to select among instantiated skills. Learning involves estimating expected the durations and success probabilities from execution traces. We conclude with comments on related work and plans for further extensions.

1. Introduction

Icarus is a cognitive architecture for physical agents that unifies ideas from a number of traditions. The framework includes a commitment to the relational representation of knowledge elements, the association of numeric value functions with logical descriptions, and the hierarchical organization of these elements in long-term memory. The system also invokes cognitive mechanisms to modulate the reactive execution of behavior and incorporates incremental learning processes that are interleaved with performance. Our framework borrows ideas from a number of traditions but combines them in a unified way.

In the next section, we review an early version of the architecture, Icarus/3, that introduced a model-free method for learning value functions associated with hierarchical relational skills. After this, we describe the Icarus/4 formalism, which incorporates a number of extensions. These include a model-based approach to the execution of hierarchical relational skills and associated learning methods. We clarify the framework's assumptions about representation and the nature of reward, then present the mathematics underlying its decision making. In closing, we discuss related work and outline directions for additional research.

2. Previous Results with Icarus/3

Our previous implementation, which we will call Icarus/3, is an architecture for reactive control that encodes knowledge in distinct skills that are organized in a hierarchy. As Shapiro et al. (2001) describe, each skill has a set of objectives, a set of requirements, and a set of alternative means that may achieve the objectives when the requirements are met. Each field contains relational literals that can share generalized arguments, much as in a logic program, and that may refer to primitive percepts or to other skills. Terminal nodes in the hierarchy are skills in which primitive percepts appear in the requirements and objectives, and in which executable actions appear in the means field. Each skill also has a value field that specifies expected reward as a linear function of sensory variables.

Like other cognitive architectures, Icarus/3 operates in cycles. In each case, processing starts from a high-level skill provided by the user and descends through the skill hierarchy to find allowable paths. A path is allowable only if each skill instance it contains has its requirements met and its objectives unmet. The system calculates the expected value of each path by combining its skill's functions with current sensory values, then selects the best-scoring path for execution. The resulting actions influence the environment, which may also change on its own, and the system repeats this process on successive cycles. The architecture uses its hierarchical structures to decide which actions are allowed, but, within these constraints, is fully reactive.

Icarus/3 learns the value function associated with each skill from delayed reward. To this end, it utilizes a variant of the SARSA algorithm (Singh et al., 2000) that operates on a path though the hierarchy rather than state-action pairs. Reward is propagated not only backward through time but also upward through skills along the path. The implementation utilizes eligibility lists, normalizes sensor values at run time, and learns linear approximations for value functions rather than tabular forms. Experiments with a simulated highway driving task showed this approach learned effective policies more than 100 times faster than a non-hierarchical method. However, although the system supported relational representations, the highway driving domain did not exercise this capability.

3. The Icarus/4 Framework

Although our work with Icarus/3 introduced important ideas, it also overlooked some others that we are incorporating into its successor. Icarus/4 extends the representation of knowledge by adding logical concepts, which let the system encode more abstract beliefs about the environment. Like skills, concepts are organized into a hierarchy, but in this case the terminal nodes correspond to primitive percepts. Skills can still invoke other skills, but a skill's requirements and objectives can refer only to concepts.

More important, the extended framework includes explicit support for describing relations among physical objects, and it utilizes a model-based approach to skill selection and learning, rather than the model-free method used by its predecessor. In addition, the new architecture factors the reward function into different components, each associated with a concept in long-term memory. Thus, Icarus/4 has a considerably richer scheme for representing and reasoning about states and activities. In this section, we discuss the key aspects of the new architecture.

3.1. Representational Assumptions

Our framework makes a number of assumptions about how the agent encodes short-term perceptions and beliefs, as well as long-term knowledge. These differ substantially from those made in traditional treatments of reinforcement learning, which emphasize Markov decision processes. Instead, our formulation combines probabilities with ideas from traditional AI planning.

1. The agent observes a set of percepts, each of which specifies an object in its environment, along with attribute values (continuous or discrete) that are associated with that object. For example, in the blocks world, a percept would be encoded as *(block A xpos 10 ypos 4 height 1 width 1)*. Objects are the only legitimate arguments of concepts and skills.

2. The agent has a set of primitive concepts, each of which specifies one or more arguments and which is defined as a conjunction of logical or arithmetic tests among attribute values of those arguments. Table 1 shows the primitive concept *on*, which describes a spatial relation between two blocks.

Table 1. Some Icarus concepts for the blocks world, with variables indicated by question marks.

```
(on (?block1 ?block2)
 :percepts  ((block ?block1 xpos ?x1 ypos ?y1)
             (block ?block2 xpos ?x2 ypos ?y2
                    height ?h2))
 :tests     ((equal ?x1 ?x2) (>= ?y1 ?y2)
             (<= ?y1 (+ ?y2 ?h2))))
(clear (?block)
 :percepts  ((block ?block))
 :negatives ((on ?other ?block)))
(three-tower (?x ?y ?z)
 :percepts  ((block ?x) (block ?y) (block ?z))
 :positives ((on ?x ?y) (on ?y ?z))
 :reward    10)
```

3. The agent has a set of high-level concepts, each of which specifies one or more arguments and which is defined as a logical conjunction of primitive concepts, other high-level concepts, or their negations. Table 1 shows the high-level concept *clear*, which describes a single block but also states that no other block may be on it. The concept *three-tower* describes a more complex relation among three blocks.

4. The agent has a set of durative skills, each of which specifies one or more arguments and a set of effects, separated into concepts that become satisfied and ones that become unsatisfied as a result of its execution. Table 2 presents four such skills from the blocks world and their effects.

5. Every skill has one or more decompositions, each of which specifies preconditions, stated as a conjunction of concepts, that must hold for it to be executed. The table also shows that *puton* has two such decompositions, whereas the primitive skills have only one.

6. Each decomposition of a high-level skill specifies one or more primitive or high-level subskills, and the order in which they should be invoked; each decomposition of a primitive skill specifies a single action that it should execute. Table 2 gives the subskills for each decomposition of *puton* and the actions associated with *pickup*, *unstack*, and *stack*.

7. Each primitive skill decomposition has an associated probability of achieving the skill's effects and an expected time to completion, provided it is executed when its preconditions are satisfied. The table indicates that, for this variant of the blocks world, *pickup*, *unstack*, and *stack* have high but not guaranteed probabilities of success, along with their expected execution time in cycles.

Taken together, the last four items define an implicit AND/OR tree in which each skill corresponds to a set of OR branches, each decomposition corresponds to a set of AND branches, and primitive decompositions constitute terminal nodes.

3.2. Reward Assumptions

The framework also makes some important assumptions about the nature of reward and its calculation.

1. Rewards are associated with concepts in the agent's long-term memory, and thus are factored into separate components. Here we assume scalar rewards, but, more generally, a concept may specify a reward function as some arithmetic combination of its argument's attribute values.
2. The global reward on each time step is the sum of rewards produced by the agent's satisfied concepts. Because a given concept may match with more than one set of objects as arguments, each such instance contributes to the total reward.
3. Some components of the global reward function may hold across a domain, whereas other components may be specified anew for each problem the agent encounters, corresponding to different goals and constraints. For instance, the reward of 10 associated with concept *three-tower* in Table 1 may hold only for a particular problem.

Note that this formulation does not view reward as coming from the environment, as in most treatments. Rather, the environment contains objects, which the agent perceives and uses to calculate its reward internally. This seems more natural than assuming that a single number arrives from an external source.

3.3. Performance Assumptions

We can now turn to how the agent utilizes these knowledge structures to respond to complex problems. We assume the agent operates in discrete cycles which take some fixed time that corresponds to its rate of sensing the environment and executing actions. On each cycle, the agent finds all concept instances (i.e., concepts with specific objects as arguments) that match against the perceived state of the environment. Moreover, for each concept instance, it calculates the reward produced by that instance.

We also assume that the agent starts with the intention of executing some high-level skill or that it is given a choice among a set of such skills. At the outset, the system generates, for each top-level candidate, all ways to expand the skill hierarchy, producing a set of AND trees with primitive skills as the terminal nodes. For each such AND tree, it generates all possible instances that replace variables with specific objects in the environment. The agent checks each tree instance and retains only those that contain some path along which the preconditions of each skill instance match against the current environmental state. On each cycle, it repeats this process to produce a set of instantiated skill trees from which to select.

Table 2. Primitive and high-level skills for the blocks world.

```
(pickup (?block ?from)
 :percepts ((block ?block xpos ?x)
            (table ?from height ?h))
 :conds    ((ontable ?block ?from)
            (clear ?block)
            (hand-empty))
 :action   ((*move ?block ?x (+ ?h 10)))
 :adds     ((holding ?block))
 :deletes  ((ontable ?block ?from)
            (clear ?block))
 :success  0.98
 :duration 1.6)
(unstack (?block ?from)
 :percepts ((block ?block xpos ?x)
            (block ?from height ?height))
 :conds    ((on ?block ?from)
            (clear ?block)
            (hand-empty))
 :action   ((*move ?block ?x (+ ?height 10)))
 :adds     ((clear ?from) (holding ?block))
 :deletes  ((on ?block ?from) (clear ?block)
            (hand-empty))
 :success  0.95
 :duration 1.8)
(stack (?block ?to)
 :percepts ((block ?block)
            (block ?to xpos ?x ypos ?y height ?h))
 :conds    ((clear ?to) (holding ?block))
 :action   ((*move ?block ?x (+ ?y ?h)))
 :adds     ((on ?block ?to) (hand-empty))
 :deletes  ((clear ?to) (holding ?block))
 :success  0.93
 :duration 2.1)
(puton (?block ?from ?to)
 :percepts ((block ?block) (table ?from)
            (block ?to))
 :conds    ((ontable ?block ?from) (clear ?block)
            (hand-empty) (clear ?to))
 :skills   ((pickup ?block ?from)
            (stack ?block ?to)))
(puton (?block ?from ?to)
 :percepts ((block ?block) (block ?from)
            (block ?to))
 :conds    ((on ?block ?from) (clear ?block)
            (hand-empty) (clear ?to))
 :skills   ((unstack ?block ?from)
            (stack ?block ?to)))
```

For each such candidate, the agent calculates the expected average change in reward if it were executed and selects the one with the highest score. This calculation and selection occurs on every cycle, letting the agent shift among the top-level skill or its subskills if new information makes an alternative look better than the tree selected on the previous cycle. However, in many domains this will occur rarely, if at all, and the agent will continue to execute the expanded skill tree selected on the first cycle until completion. Expanding the skill hierarchy fully on each cycle and selecting among entire instantiated AND trees is not the most efficient way to make decisions, but it simplifies our analysis considerably.

Let I be an instance of an expanded skill tree in which S_I is the instantiated top-level skill. Furthermore, let A_I be the set of literals (concept instances) achieved by S_I that are not currently satisfied and let D_I be the literals made untrue by S_I that are currently satisfied. Finally, let $r(x)$ be the reward produced by the literal x when it is satisfied.

We are interested in two reward-related aspects of skill trees. One involves the cumulative change in reward $C(I)$ that results from I's successful completion. We can calculate this as

$$C(I) = \sum_{a \in A_I} r(a) - \sum_{d \in D_I} r(d) + \sum_{j=1}^{J} C(j) \ ,$$

where $C(j)$ refers to the change that results from completion of one of the J component skill trees of I. This recursive expression terminates with primitive skills, for which the third term is zero because they have no components. Summing over the effects of components makes sense because we care about the cumulative change in reward.

The other quantity concerns the cumulative change in reward $B(I)$ that occurs during the skill's execution. We can compute this as

$$B(I) = \sum_{j=1}^{J-1} \left[C(j) \cdot \sum_{k=j+1}^{J} d(k) \right] + C(J) \ ,$$

where $d(k)$ is the number of cycles expected to execute the component skill k successfully. This produces a weighted sum in which the reward contributions of earlier subskills are greater because they occur earlier in the behavioral trajectory. The final term represents the change produced by the final subskill, which holds only for the last time step. Because primitive skills have no components, their value for B is zero.

Of course, we are less interested in the expected cumulative change in reward than in the average change, for which we require predictions about the number of cycles taken for skills to achieve their objectives. We can define the expected duration of a high-level skill tree instance I as

$$d(I) = \sum_{j=1}^{J} d(j) \ ,$$

which is simply the sum of the durations for its component skill trees. We assume that, if a primitive skill fails, it leaves the environment unchanged, so the agent can simply execute the skill again until it succeeds. If $p(I)$ is the probability that primitive skill instance I will succeed on a given attempt and $t(I)$ is the expected number of cycles per attempt, then we have

$$d(I) = t(I)/p(I)$$

as the expected cycles required for I to have its intended effects. We assume that the probabilities and execution times for primitive skills are estimated from previous experience, as described later.

We can combine these three terms to compute the expected average change in reward $A(I)$ for a given instantiated skill tree I. A naive expression would be

$$A(I) = \left[C(I) + B(I) \right] / d(I) \ ,$$

which simply divides the sum of the expected change due to completion and the change within execution by the expected duration. However, for skills that take more than a few cycles, this gives most of the weight to within-execution changes, even though, in many domains, the effects of a top-level skill's execution will be retained and valued for some time after its completion.

To model this situation, we specify $f(I)$ as the expected number of cycles following successful completion of a top-level skill tree I that its effects will be unchanged by other activities. This suggests

$$A(I) = [C(I) \cdot f(I) + B(I)]/[d(I) + f(I)]$$

as a more reasonable expression, since $C(I)$ is multiplied by the time its changes will remain in force, giving increased weight to the long-term effects of executing I. Larger values for $f(I)$ indicate a greater willingness to amortize the results of one's actions, and thus will bias the agent to accept delayed rather than immediate gratification.

The scheme described above takes advantage of known orderings on subskills to avoid the reliance on dynamic programming required by many model-based analyses. However, expanding the skill hierarchy into all possible AND trees still seems undesirable for a reactive system. A more efficient approach would store with each skill decomposition its expected duration, probability of success, and expected change in reward during execution. This would let the agent select among alternative decompositions at the current level, without needing to expand the subskills, but would make it less sensitive to changes in the environment.

We should note that neither of these methods let the agent discriminate among alternative instantiations of an expanded skill tree, which is necessary to achieve the full benefits of the relational formalism. For this purpose, we need two extensions. One involves using functions that predict the expected duration and success probability of a skill instance from the attribute values of objects that serve as its arguments. For example, lifting a heavier block may take longer than a lighter one and be more likely to fail.

The other extension involves adding the ability to draw inferences from the effects of skills and the current state, since these may imply satisfaction or retraction of concepts not mentioned in the skills themselves. For example, executing subskills that produce *(on A B)* and *(on B C)* implies *(three-tower A B C)*, which is a source of reward, whereas executing subskills that produce *(on A B)* and *(on C D)* does not. We can mimic this ability by storing such effects with higher-level skills, but this requires more hand crafting than seems desirable.

3.4. Learning Assumptions

Our framework posits that the agent already has concepts and skills in long-term memory, including the order in which subskills should be invoked for each decomposition. However, although it has initial guesses about the expected duration and success probability of its primitive skills, we assume it can use experience to improve them. One simple approach initializes these parameters to the same values for each skill using a Dirichlet distribution. In practice, this involves storing the number of times each skill has been executed and one sum for each parameter.

In this scheme, learning simply involves incrementing the counter whenever the agent executes a skill and increasing the success and duration counts upon its completion. The agent then uses these running totals to calculate the success probability and expected duration on each cycle, which makes learning entirely incremental. For the variant in which high-level skills have similar statistics, the agent must retain multiple totals for each expansion and take the maximum.

4. Discussion

Our framework incorporates ideas from three distinct literatures in the reinforcement learning community, each involving extensions to the basic value-function framework. The first concerns the use of hierarchical structures to provide temporal abstraction and to decompose both performance and learning into simpler tasks. ICARUS has similarities to Parr and Russell's (1998) hierarchies of abstract machines, but we have been influenced more directly by Dietterich's (2000) MAXQ framework, which also makes choices among nodes in an AND/OR tree. Our focus on average reward in a hierarchical setting follows the work of Seri and Tadepalli (2002). These earlier analyses all rely on the theory of semi-Markov decision processes, which we have replaced with stronger assumptions about the effects of actions closer to those made in classical artificial intelligence research.

We have also borrowed ideas from work on model-based reinforcement learning, which utilizes a mapping from states and actions onto states rather than values. Most efforts along these lines (e.g., Barto et al., 1995) suppose the agent must learn this mapping, whereas we assume it begins with knowledge about the effects of actions and must learn only durations and success probabilities. Our representation for action models is similar to that described by Pasula et al. (in press), who utilize a probabilistic variant of STRIPS operators. However, our embedding of model-based methods within a hierarchical formalism comes closer to Seri and Tadepalli's framework.

Finally, we have been influenced strongly by research on relational reinforcement learning, which incorporates relational representations to support more powerful abstractions than traditional methods. Within this paradigm, some work has adapted model-free learning techniques (e.g., Dzeroski, de Raedt, & Driessens, 2001), whereas other efforts have pursued model-based approaches (e.g., van Otterlo, in press). Our framework makes only marginal contact with these earlier efforts, since it calculates expected reward for instantiations of general skills, which it relies on pattern matching to generate. Each such instantiation is treated as an alternative course of action available to the agent, which the agent evaluates using its knowledge about the effects of component skills.

Although we have implemented the key representational ideas described above in ICARUS/4, we have not yet incorporated the model-based selection of skill trees or the methods for learning durations and success probabilities. We predict that this approach will support more effective initial behavior and more rapid learning than the model-free scheme of ICARUS/3, but this is an empirical question that can best be answered with experiments. To this end, we plan to evaluate the two variants in a simulated in-city driving environment that we have already used in other studies.

Our model-based techniques rely centrally on both accurate predictions of primitive skills' effects and on a well-structured skill hierarchy. Although the current framework assumes these are provided at the outset, we have definite ideas about how they might be learned from experience. Benson (1995) describes one method for learning action models that correspond to primitive durative skills, and Pasula et al. report another that produces probabilistic descriptions similar to those in our analysis. Reddy and Tadepalli (1997) present an approach to inducing decomposition rules from problem-solving traces, and we have developed a related method that learns ICARUS skill hierarchies in a cumulative manner (Langley & Rogers, 2004).

However, like any induction mechanism, such techniques can make errors, which would lead to skills with conditions and effects that are only partially correct. In such cases, it seems advisable to retain some form of model-free reinforcement learning to account for reward not predicted by the agent's skills. In this framework, the system would divide the reward stream into two components, one claimed by predicted effects and another handled by the model-free method from ICARUS/3. This combined scheme would let each skill maintain two estimates of future reward, one easy to attribute but dependent on accurate models and another to account for the remainder. We hypothesize that such a hybrid architecture would still support rapid learning but give higher asymptotic behavior.

Acknowledgements

This research was funded in part by Grant IIS-0335353 from the National Science Foundation, by Grant NCC 2-1220 from NASA Ames Research Center, and by Grant HR0011-04-1-0008 from Rome Labs.

References

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*, 81–138.

Benson, S. (1995). Induction learning of reactive action models. *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 47–54). San Francisco: Morgan Kaufmann.

Choi, D., Kaufman, M., Langley, P., Nejati, N., & Shapiro, D. (in press). An architecture for persistent reactive behavior. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*. New York: ACM Press.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *9*, 227–303.

Dzeroski, S., de Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*, 7–52.

Langley, P., & Rogers, S. (2004). *Cumulative learning of hierarchical skills* (Technical Report). Institute for the Study of Learning and Expertise, Palo Alto, CA.

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems 10* (pp. 1043–1049). Cambridge, MA: MIT Press.

Pasula, H. M., Zettlemoyer, L. S., & Kaelbling, L. P. (in press). Learning probabilistic relational planning rules. *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*.

Reddy, C., & Tadepalli, P. (1997). Learning goal-decomposition rules using exercises. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 278–286). Nashville, TN.

Seri, S., & Tadepalli, P. (2002). Model-based hierarchical average-reward reinforcement learning. *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 562–569). Sydney: Morgan Kaufmann.

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.

Singh, S., Jaakola, T., Littman, M. L., & Szepesvari, C. (2000). Convergence results for single-step on-policy reinforcement learning algorithms. *Machine Learning*, *38*, 287–308.

van Otterlo, M. (in press). Reinforcement learning for relational MDPs. *Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands*. Brussels, Belgium.