Knowledge-Directed Theory Revision

Kamal Ali, Kevin Leung, Tolga Konik, Dongkyu Choi, and Dan Shapiro

Cognitive Systems Laboratory
Center for the Study of Language and Information
Stanford University, Stanford, CA 94305
kamal3@yahoo.com, kkleung@stanford.edu, konik@stanford.edu,
dongkyuc@stanford.edu, dgs@csli.stanford.edu

Abstract. Using domain knowledge to speed up learning is widely accepted but theory revision of such knowledge continues to use general syntactic operators. Using such operators for theory revision of teleoreactive logic programs is especially expensive in which proof of a top-level goal involves playing a game. In such contexts, one should have the option to complement general theory revision with domain-specific knowledge. Using American football as an example, we use Icarus' multi-agent teleoreactive logic programming ability to encode a coach agent whose concepts correspond to faults recognized in execution of the play and whose skills correspond to making repairs in the goals of the player agents. Our results show effective learning using as few as twenty examples. We also show that structural changes made by such revision can produce performance gains that cannot be matched by doing only numeric optimization.

1 Introduction

Teleoreactive logic programs (TLPs) hosted in systems such as Icarus [1] are programs that are goal-oriented yet able to react when an external factor may cause already achieved subgoals to become `false`. A TLP consists of a graph (hierarchy) of first-order rules and a corresponding graph of skills. Proof of a top-level goal starts by forward chaining from a set of facts - the *perceptual buffer* - to compute its transitive closure. Next, proof of the top-level goal is attempted using backward chaining. When reaching a leaf in the proof tree which is currently `false`, the teleoreactive framework will switch to a skill tree, back-chaining until it reaches a leaf skill which can be executed.

Theory revision of skills is different than revision of concepts in that many candidate revisions cannot be repeatedly evaluated against a static training set. As a skill is changed, it forces the environment to react. Evaluation of a candidate revision of a skill requires the game to be played again - several times for stochastic domains. Thus it becomes imperative to reduce the number of training examples by orders of magnitude. This leads to the central thesis of this paper: that a skill revision engine be capable of reading and using domain-specific revision rules when available, before falling back to general syntactic revision operators.

This paper does not present a sophisticated theory revision algorithm - the whole point is to show that for revision of skills, domain-specific theory revision rules are *necessary* in order to keep down the number of required training examples, and that only a simple revision engine is needed as long it can take advantage of such powerful revision rules. By its sophistication, a skill revision algorithm cannot make up for its lack of domain knowledge if it is to lead to revision in a small number of training examples. The situation is analogous to the transition in AI from general learning algorithms to expert systems, where it became accepted that to speed up learning, the learner should have the capacity to use and learn with domain-specific expert-provided rules before falling back to general learning methods.

For skill revision in American football, for example, it is easy to elicit domain-specific theory revision operators because coaches are well aware of domain-specific ways in which they can perturb their play designs to improve performance. As a very simple example, in a skill where a ball receiver fumbled reception of the ball, the domain-specific revision knowledge base could exploit the knowledge that there are usually multiple receivers to try another receiver. The revision knowledge embodied here is that there is a subset of players that can be thrown to, and that one need not only throw to the player currently specified in the logic program.

To implement domain-specific theory revision, we take advantage of a recent extension to Icarus [1] - multi-agent capability. A natural and elegant way to exploit the multi-agent ability for theory revision is to create a "coach agent" whose concepts correspond to faults observed while the program is in play and whose skills correspond to fixes. As the game proceeds, each agent uses its inferences to decide which skills to execute. At each time tick, the coach agent also makes inferences too - some of them corresponding directly to faults, others used in support of faults. After the game is over, skills of the coach agent are applied - these skills are implemented by rules whose preconditions are arbitrary combinations of faults and whose actions involve modifying programs of the other agents.

Since our thesis is that for skill revision which needs expensive execution of plays, the revision engine should be able to accommodate domain-specific revision rules, we chose American football as a domain because its plays are intricate coordinations chosen before the play by the coach and communicated to all the players. In addition, it has the property that once play begins, the plan may get modified due to efforts of the opposing team, so it also requires the agents to be reactive. These properties are a good fit for the teleoreactive capabilities of Icarus. Our empirical validation in section 5 show that using a proof-of-concept small revision theory using about a dozen faults and ten fixes, domain-specific revisions can produce significant performance improvement using only twenty training examples.

With regard to prior work, these "debugging rules" are similar in sprit to declarative theory refinement operators other ILP systems use, but unlike theory refinement operators that are not particularly designed for temporal events, the debugging rules of our system detect and accumulate problems over to the end of the game. Other authors [2] have something equivalent to debugging knowledge in the form of a *critic agent* but this agent does not rely on execution to find faults - it does its work in the planning phase prior to execution. Gardenfors [3] notes a similarity between belief revision and nonmonotonic logic - this is analogous to our work except at a “meta” level: we are revising theories (not beliefs) so the nonmonotonicity applies between revisions.

The rest of the paper is organized as follows: section 2 presents representation, inference and execution of teleoreactive programs in Icarus. Section 3 explains how we modeled American football in Icarus. Section 4 presents `DOM-TR`: our theory revision algorithm that inputs domain-specific rules. Section 5 gives results showing how domain-specific theory revision allows effective learning using just a few tens of examples.

2 Teleoreactive Logic in Icarus

Representation - The Icarus architecture (details in [1]) represents conceptual and procedural knowledge using first-order logic. Concepts consist of a generalized head, perceptual matching conditions in a `:percepts` field, tests for constraints in a `:tests` field, and references to other concepts in a `:relations` field. For example, the concept *cch-near* shown in table 1 matches against two objects of type *agent*, and binds values to variables (indicated with `?`). Skills (right column in table 1) are also represented with generalized heads and bodies - the body can refer directly to executable procedural attachments in an `actions` field or to other skills in a `subgoals` field.

Table 1. Some sample concepts and skills in the football domain

```
((cch-near ?a1 ?a2)
 :percepts  ((agent ?a1 x ?x1 y ?y1)
             (agent ?a2 x ?x2 y ?y2))
 :tests     ((<= (sqrt (+ (expt (- ?x1 ?x2) 2)
                          (expt (- ?y1 ?y2) 2)))
                 *threshold*)))

((cch-covered-recipient ?a1 ?a2)
 :percepts  ((agent ?a1 team offense recipient t)
             (agent ?a2 team defense))
 :relations ((cch-near ?a1 ?a2)))

((cch-open-recipient ?agent)
 :percepts  ((agent ?agent team offense recipient t))
 :relations ((not
              (cch-covered-recipient ?agent ?a2))))
```

```
(RWR-play 11-1-013-1)
 :percepts ((agent ?N role RWR
             startx ?X starty ?Y))
 :actions  ((*startAt RWR ?X ?Y)
            (*PassRouteCornerLeftAtYard
             RWR *RWR-ydsb4diagLeft*)
            (*finish ?N)))

((QB-play 11-1-013-1)
 :percepts ((playState 11-1-013-1)
            (agent ?N role QB
             startx ?X starty ?Y closestRcvr ?R))
 :subgoals ((startAt QB)
            (QBFallback 11-1-013-1)
            (waitForReceivers 11-1-013-1)
            (pass ?R)))
```

Interpretation - Icarus accepts a user-provided top-level goal per agent, the set of agents being pre-defined by a user. It operates in distinct cycles, and it infers the current state of the world at the beginning of each cycle. Based on the inferred state, the system finds an executable path through its skill hierarchy, which starts with a currently executable skill for the top-level goal and terminates with a primitive skill with procedural attachments. ICARUS repeats the process for all the agents, and executes all implied actions simultaneously in a single cycle.

For example, to achieve a goal, `(QB-play 11-1-013-1)`, the `QB` agent must execute a corresponding skill with the same head – like the second skill in the right column on table 1 – that entails making true *in sequence* the four sub-goals specified in its `subgoals` clause. Teleoreactivity in the system mainly results from disjuntive skills it has for a particular goal, which provide alternatives to the system based on the current state of the world.

3 American Football – An Example Domain

For the American football domain, Icarus controls the offensive team. We define one agent per player on the offensive team and an additional agent for the coach. The agents execute their skills against an "environment" that is managed by the RUSH football simulator [4]. RUSH controls the defense, physics of the ball and stochasticity of the environment.

We use the system in [5] to transform video of real games into a sequence of segments per player. Each segment consists of a symbolic label and numeric parameters - for example, `slantLeft(253,283,RWR,10)` indicates the Right Wide-Receiver should run diagonally for ten yards from time ticks 253 to 283. To translate this into a logic program for Icarus, each sequence is translated into an Icarus skill as in the right column of table 1. Constants in the skills are replaced by variables (parameters) that have those constants as their default values. For example, 10 becomes the variable (parameter) `*RWR-yardsb4diagLeft*`.

4 Theory Revision

Table 2 gives a few examples of the revision knowledge which was elicited from an expert. Note that it is generalized to first order and can either make changes that are local to an agent or changes that involve multiple agents. By execution, we may find, as shown in rule 1 in Table 2 that X manages to get more open and thus the ball should be thrown to X. The revision operators can change variables into constants and vice versa, replace predicates, introduce new variables and

Table 2. Examples of domain-specific faults and fixes

Fault: cch-farther-open-recipient(X,Y) **Description:** X is an open receiver farther downfield than Y **Fix:** Change intended receiver to X from Y
Fault: cch-crowded-recipients(X,Y) **Description:** Receivers X and Y are too close to each other **Fix:** Move the starting locations of each receivers 2 yards apart

introduce negation. Not only is the primary theory first-order, but the revisions themselves are first order. For instance, the skill in the lower right of table 1 was modified by theory revision which introduced a new variable (`?R`) whose value is picked up from the `:percepts` clause and used in the `:subgoals` clause.

Table 3. DOM-TR: Structural Learning Breadth-First Search

```
learn-structure(MaxDepth)
1  Depth = Depth + 1
2  **For** $N$ (4) times:
3        Simulate the play, record faults into Faults
4  If Faults is empty then exit
5  **For** $fix$ in fixes(Faults) do
6        Temporarily apply fix
7        Simulate $N$ (4) times, recording rewards
8  AR = reward from best fault/fix pair
9  Improvement = AR - BestReward
10 **if** Improvement $\leq 0$ then exit
11 **else** commit fix; BestReward = AR
12 If depth $\leq$ MaxDepth then go to 1
```

Table 3 presents `DOM-TR` - a breadth-first algorithm that inputs a domain-specific debugging theory to do theory revision. The domain-specificity of the debugging theory greatly reduces the breadth of the search - for our current knowledge base, the breadth factor is usually only one or two. Note that the core step of the algorithm - evaluating a fix - involves playing several ($N = 4$) games[1] because of the stochasticity of this domain.

Table 3 shows that the algorithm begins by playing N games, recording all occuring faults. The other purpose of playing these games is to establish a baseline reward value. After playing the N initial games, it iterates over fixes whose preconditions match the observed faults. For each fix, it applies the fix, plays N games to compute average reward and then undoes the fix. If the best fix so found produces an improvement compared to the baseline, it is permanently applied, otherwise the search terminates. We do not check explicitly for cycles (where a revision undoes a previous revision), opting instead to limit the depth of the revision tree to some small value such as $MaxDepth = 5$.

5 Results

For our experimental methodology, we randomly select an offense from our library of videos and a defense from RUSH's library. In each such trial, `DOM-TR` is used to produce a learning curve. The process is repeated twenty times. Figure 1 shows the results - application of `DOM-TR` to our small prototype revision theory improves performance from a baseline figure of 1 yard to 4 yards. Although this gain may appear numerically small, a repeatable three-yard gain using only 20 plays (and given that our revision theory is just a proof-of-concept) is significant in American football.

[1] Early experiments showed that values of N higher than 4 did not yield substantially greater benefits.

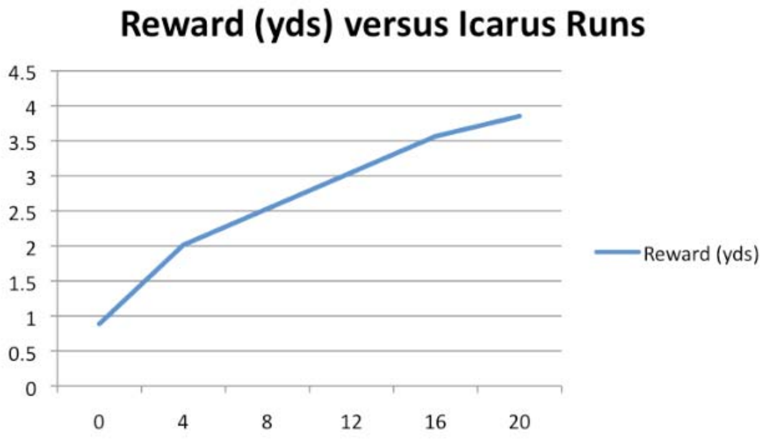

Fig. 1. Learning curve for theory revision

In a second experiment, we wanted to compare structural learning using DOM-TR to an algorithm that only did numeric optimization (by hill-climbing over all parameters mentioned in the theory). In figure 2, the leftmost points of the curve correspond to the end of the theory revision stage ($x = 20$) and show that DOM-TR achieves a performance of six yards by using the domain-specific theory revision rules. The middle curve corresponds to taking the same Icarus program but disallowing structure learning - only using parameter learning. The results show that without the benefit of the powerful domain-specific theory revision rules, to reach the six yard performance mark takes parameter optimization almost an order of magnitude more examples ($x = 160$ versus $x = 20$). Furthermore, even asymptotically, the algorithm that only does parameter optimization

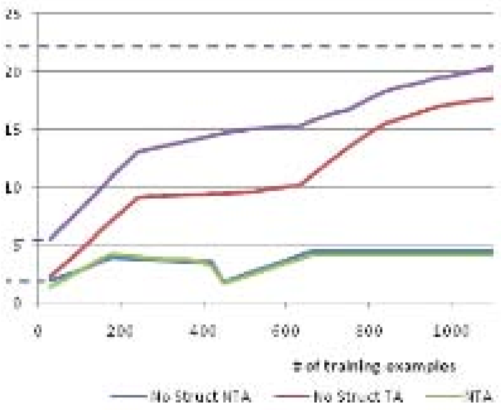

Fig. 2. Uppermost curve: structural and parameter learning; middle curve: parameter learning only; bottom curve: learning for a "knowledge-free" agent

(middle curve) cannot match the reward obtained by the hybrid algorithm (upper curve) that combines structure learning with numeric optimization.

So far, we have compared two learning algorithms, both on a rich domain-specific theory. Now we vary the program itself to answer "what is the impact of this domain-specific theory"? - can we compare it to a "knowledge-free" theory to serve as a control? It is somewhat problematic to define a knowledge-free program that achieves non-zero performance. Should the knowledge-free program be just the null program? Should it correspond to an agent that knows about team games but not American football? Should it correspond to expert-level knowledge from rugby that should serve as a starting point for adaptation to American football?

In order to achieve non-zero performance, we defined the control theory to be one that knows all the rules of American football but none of the strategies that coaches possess in their toolkits. We defined such a "naive" agent to be one that places all players that are eligible receivers on the scrimmage line under the instructions to run as fast as possible towards the touchdown line. The quarterback's behavior is defined to throw immediately (he does not have the notion of a protective fallback) to the receiver that is furthest downfield. Thus this program lacks notions of coordinated and distracting plays, notions of running downfield and then cutting across, and so forth. The Icarus program that corresponds to this definition has a non-zero set of rules so the theory-revision change-knowledge rules we used earlier can also be applied to this program. It also has parameters but they parameterize a space that consists simply of starting x and y coordinates for the players. This contrasts with the informed program whose parameters correspond to semantically rich and relevant constructs such as the length of time the quarterback should wait before throwing the ball, or how far he should fall back from the scrimmage line to protect his throw.

Figure 2 shows that after theory revision ($x = 20$) this program achieves a performance of two yards, in comparison to the six yard gain achieved when theory revision was applied to the informed theory (some theory revision rules such as switching receivers are applicable even to this control theory). Thus, applying domain-specific theory revision operators to the informed theory produce a 200% gain over application of the same revision rules to the uninformed theory.

Figure 2 also shows that the control agent is only able to take minimal advantage of the greedy hill-climbing parameter-learning algorithm - probably because its parameters correspond to low-level x and y coordinates. In American football, it is important for there to be coordination between multiple players which may not be learnable by a greedy method operating over a set of raw, low-level parameters. such as coordinates. By contrast, the greedy hill-climbing algorithm is able to significantly improve the informed theory since its parameters implicitly correspond to degrees of coordination between multiple players.

6 Conclusions

In domains where learning from training examples is orders of magnitude more expensive than usual and where skills need to be revised, general syntactic-only theory revision needs to be augmented by domain-specific revision knowledge. This paper demonstrates theory revision in the context of multi-agent teleoreactive logic programs by the implementation of a “coach” agent whose concepts correspond to faults and whose skills correspond to fixes that modify those agents’ programs. Using American football as an example, we have demonstrated that domain-specific theory revision can produce meaningful gains using as few as twenty examples and that this affords an order of magnitude faster learning that that realized by doing only parameter optimization.

Acknowledgements

This paper reports research sponsored by DARPA under agreement FA8750-05-2-0283. The U. S. Government may reproduce and distribute reprints for Governmental purposes notwithstanding any copyrights. The authors’ views and conclusions should not be interpreted as representing official policies or endorsements, expressed or implied, of DARPA or the Government.

References

1. Langley, P., Choi, D.: A unified cognitive architecture for physical agents. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence, Boston. AAAI Press, Menlo Park (2006)
2. Wilkins, D., Myers, K., Lowrance, J., Wesley, L.: A multiagent planning architecture. In: Proceedings of AIPS 1998 (1998)
3. Gardenfors, P.: Belief revision and nonomotonic logic: Two sides of the same coin? In: Proceedings of the ninth European Conference on Artificial Intelligence. Pitman Publishing (1990)
4. Sourceforge: Sourceforge.net - rush 2005 (2005), `http://sourceforge.net/projects/rush2005/`
5. Hess, R., Fern, A.: Discriminatively trained particle filters for complex multi-object tracking. In: IEEE Conference on Computer Vision and Pattern Recognition (2009)