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 which infers faults during a game and at its conclusion applies procedural attachments to fix programs of the other 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: to use domain-specific revision knowledge when available in addition to general syntactic revision operators.

Presented at the 19th International Conference on Inductive Logic Programming (ILP'09).

In American football, for example, it is easy to elicit theory revision operators that correspond to variations that a coach may try. For instance, a rule may specify that if throwing to a particular receiver does not work, one should try another eligible 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. Our results in section 5 show that using a proof-of-concept small revision theory using about a dozen faults and ten fixes, we are able to get significant improvement using just 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 (table 1). 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 the `:relations` field. For example, the concept *cch-near* 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 to directly executable procedural attachments in the `actions` clause or to other skills (the `subgoals` clause).

Inference and Execution - Icarus accepts a user-provided top-level goal per agent, the set of agents being pre-defined by a user. Icarus performs backward chaining on the top-level goal, terminating either in `true` literals or `false` literals for which it can find a skill to execute. Next, backward chaining is performed in the skill hierarchy until it reaches a leaf skills with procedural attachments. Once Icarus finds active skill paths for all the agents, it simultaneously executes all actions in one time tick.

For example, the second skill in the right column on table 1 shows that to achieve the goal `QB-play 11-1-013-1`, the `QB` agent must execute its corresponding skill - `QB-play 11-1-013-1` - which entails making true *in sequence* the four sub-goals specified in its `subgoals` clause. Teleoreactivity is possible in this domain since, for example, the pre-condition of a skill that takes several time ticks to complete may become false while the skill is being executed, forcing Icarus to select a different skill or subgoal.

```
((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)))
```

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

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

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

Table 2. Examples of domain-specific faults and fixes.

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 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.

```
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. DOM-TR: Structural Learning Breadth-First Search

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

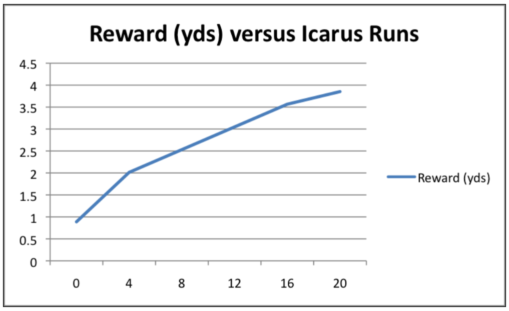

Fig. 1. Learning curve for theory revision

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 2 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 our revision theory is just a proof-of-concept) is significant in American football.

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). The results in figure 2 show that by the end of the theory revision phase, $x = 20$, `DOM-TR` has achieved a reward of 6. The numeric-only algorithm requires approximately an order of magnitude more examples ($x = 160$) to reach the same reward level. The upper learning curve in

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

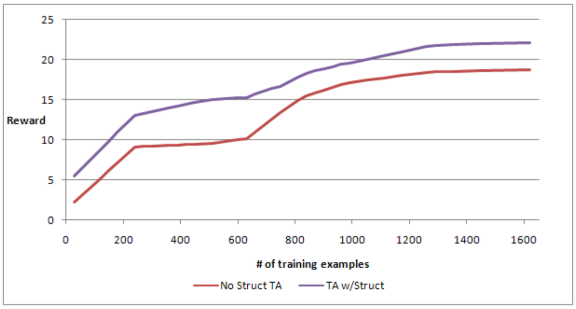

Fig. 2. Learning curves for two algorithms: upper curve does theory revision followed by parameter hill-climbing, bottom curve does only parameter optimization.

figure 2 is obtained by doing theory revision (upto the 20'th training example) and hill-climbing thereafter. Furthermore, even asymptotically, the algorithm that only does parameter optimization (lower curve) cannot match the reward obtained by the hybrid algorithm (upper curve) that combines structure learning with numeric optimization.

6 Conclusions

In domains where training examples are 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.

References

1. Asgharbeygi, N., Nejati, N., Langley, P., Arai, S.: Guiding inference through relational reinforcement learning. In: Proceedings of the International Conference on Inductive Logic Programming. (2005)
2. Wilkins, D., Myers, K., Lowrance, J., Wesley, L.: A multiagent planning architecture. In: Proceedings of AIPS-98. (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. http://sourceforge.net/projects/rush2005/ (2005)
5. Hess, R., Fern, A.: Discriminatively trained particle filters for complex multi-object tracking. In: IEEE Conference on Computer Vision and Pattern Recognition. (2009)