Learning Hierarchical Skills for Game Agents from Video of Human Behavior

Nan Li and **David J. Stracuzzi** and **Gary Cleveland** and **Pat Langley**
School of Computing and Informatics, Arizona State University

Tolga Könik and **Dan Shapiro** and **Kamal Ali**
Computational Learning Laboratory, Stanford University

Matthew Molineaux
Knexus Research, Springfield, VA

David W. Aha
Naval Research Laboratory, Washington, DC

Abstract

Developing autonomous agents for computer games is often a lengthy and expensive undertaking that requires manual encoding of detailed and complex knowledge. In this paper we show how to acquire hierarchical skills for controlling a team of simulated football players by observing video of college football play. We then demonstrate the results in the Rush 2008 football simulator, showing that the learned skills have high fidelity with respect to the observed video and are robust to changes in the environment. Finally, we conclude with discussions of this work and of possible improvements.

1 Introduction

Constructing intelligent agents for computer games is an important aspect of game development. However, traditional methods are expensive, and the resulting agents only function in narrowly defined circumstances. Agent architectures [Newell, 1990], which are designed to model human-level intelligence, can help simplify this task by reducing the need for developers to explicitly provide agents with intelligent behaviors. Developers need only specify static knowledge about the domain, while the architecture constructs the necessary behavior specifications, and handles the reasoning and decision making required for their execution.

Recent research [Pearson and Laird, 2004; Nejati *et al.*, 2006] aims to simplify the knowledge acquisition process. We expand upon this work by learning from video data, by representing and acquiring temporal knowledge, and by reducing the amount of expert input required by the system. In the following, we present a system that learns hierarchical skills for football players by observing videos of college football plays, and then executes those skills in a simulated football environment. The system takes in discrete perceptual traces generated from the video, analyzes these traces with existing knowledge, and learns new skills that can be applied when controlling agents in a football simulator. The skills are acquired in a cumulative manner, with new skills building on those previously learned, and are based on conceptual knowledge that includes temporal constraints. The learned skills also reproduce the observed behavior faithfully with comparable efficacy in spite of differences between the observation and demonstration environments.

We begin by introducing the Rush 2008 football simulator, which we use to demonstrate our approach. We then briefly review the ICARUS agent architecture, and present our work on acquiring structured domain knowledge from observed human behavior within this framework. Next, we outline the video preprocessing steps, the application of ICARUS to the processed video, and the application of the acquired skills in Rush. Finally, we conclude with a summary of related work and directions for future development.

2 The Rush Football Simulator

Rush 2008, a research extension of Rush 2005[1], simulates play in an eight player variant of American football. For the remainder of the paper, we assume a basic familiarity with the rules and objectives of American football. The simulator encodes several types of offensive and defensive formations, along with several distinct plays (coordinated strategies) for each formation. Each player is assigned a role, such as quarterback (QB), running back (RB), or left wide receiver (LWR). Players are controlled either by assigning them a high-level goal such as *pass route cross out at yard 15*, which instructs a receiver to run 15 yards downfield, make a hard right turn, and then keep running to try to catch a pass, or by assigning specific instructions such as *stride forward* on each clock tick.

We instrumented Rush so that ICARUS can perceive all players and objects on the field and control the actions of each offensive player on a tick-by-tick basis. Each offensive player shares perception and knowledge with the other offensive players, and carries out actions instructed by ICARUS. Examples of the types of actions available to ICARUS include

[1] http://rush2005.sourceforge.net/

(throwTo <receiver>), which instructs the QB to pass to a specific teammate, and *(stride <direction>)*, which tells a player to run in one of eight directions for one clock tick. Defensive players are controlled by the simulator, which selects one of several available strategies for each play.

3 The ICARUS Architecture

Our framework for play observation, execution and learning is the ICARUS architecture, which is an instance of a unified cognitive architecture [Newell, 1990]. ICARUS shares many features with other architectures like Soar [Laird *et al.*, 1986] and ACT-R [Anderson, 1993], including a distinction between short-term and long-term memories, and goal-driven but reactive execution. It also has many novel features including a commitment to separate storage for conceptual and skill knowledge, the indexing of skills by the goals they achieve, and architectural support for temporal knowledge [Stracuzzi *et al.*, 2009]. In this section, we summarize the basic assumptions of the framework along with its operational procedures to provide background for the new learning methods. We begin by describing the conceptual and belief representations along with the inference process, and then describe skill structures and the execution mechanism.

3.1 Beliefs, Concepts and Inference

Reasoning about the environment is a principal task performed by intelligent agents, and determines which actions the agent must carry out to achieve its goals. ICARUS performs the inference task by matching the generalized conceptual structures in long-term memory against percepts and beliefs stored in short-term memory. On each cycle, an agent receives low-level perceptual information about its environment. Each percept describes the attributes of a single object in the world, and is short lived. For example, the percepts derived from the football video last only for the duration of one video frame ($1/30^{th}$ of a second) before being replaced with new information.

Intelligent behavior requires more than low-level perceptual information. ICARUS therefore produces higher-level beliefs about the environment based on percepts. Beliefs represent relations among objects in the environment, and have two associated timestamps that indicate the beginning and end of the period in which the belief is known to be true. All of the inferred beliefs for the current episode are retained in a simple episodic belief memory, so that the agent can reason about events over time.

ICARUS beliefs are instances of generalized concepts stated in a hierarchically organized, long-term conceptual memory. Each concept describes a class of environmental situations using a relational language; it includes a head, which consists of a predicate with arguments, and a body, which defines the situations under which the concept is true. The body has a relations field as well as a constraints field. The relations field specifies the subconcepts on which the concept depends along with their associated timestamps, which correspond to the timestamps on beliefs. The constraints field uses these timestamps to describe the temporal relationships among the subconcepts. For example, in Table 1 the constraints field of *dropped-back* states that *?passer* must have possession of the ball until he has finished dropping back.

Table 1: Sample concepts for the football domain.

```
; ?passer dropped back ?n-steps after receiving the snap
((dropped-back ?passer ?n-steps)
  :relations    (((snap-completed ?passer ?ball) ?snap-start NOW)
                 ((possession ?passer ?ball) ?poss-start ?poss-end)
                 ((moved-distance ?passer ?n-steps S)
                   ?mov-start ?mov-end))
  :constraints ((≤ ?snap-start ?poss-start)
                (≤ ?mov-end ?poss-end)
                (= ?mov-end NOW)))

; learned start condition for skill dropped-back
((scdropped-back-c1 ?passer)
  :relations    (((possession ?passer ?ball) ?pos-start ?pos-end)
                 ((snap-completed ?passer ?ball) ?snap-start NOW))
  :constraints ((≤ ?snap-start ?pos-start)))
```

ICARUS updates belief memory by matching the generalized concept definitions to percepts and existing beliefs in a bottom-up manner. Concepts that depend only on percepts are matched against the perceptual buffer first, and the results are placed in belief memory. This triggers matching against higher-level concept definitions. The process continues until the deductive closure of perceptual, belief and conceptual memory has been computed.

3.2 Goals, Skills and Execution

After inferring a set of beliefs about its environment, ICARUS next evaluates its skill knowledge to determine which actions to take in the environment. For this, the architecture uses a goal memory, which stores goals that the agent wants to achieve. It then retrieves skills that are able to achieve its goals from long-term skill memory.

Skill memory contains a set of hierarchical skills indexed by concepts defined in the conceptual memory. Each skill describes a method that the agent can execute to achieve the condition described by the associated concept. Each skill controls only a single agent (player). Coordination among agents is achieved by relying on specific timing events that are observed by multiple agents. Skills consist of a head, which specifies the goal that the agent achieves by carrying out the skill, a set of start conditions that must be satisfied to initiate the skill, and a body that states the ordered subgoals that the agent must achieve to reach the skill's goal. A primitive skill is one that refers only to actions that are directly executable in the environment, while a non-primitive skill refers to other skills as subgoals.

Given skill memory and one or more goals, the execution module first selects an unsatisfied, top-level goal. Since skills can refer to other skills, the system next finds an applicable path through the skill hierarchy, which begins with a skill that can achieve the selected goal and terminates with one that refers to actions the agent can execute in the environment. ICARUS continues executing along this path until the goal is achieved or the selected skills no longer apply, in which case it must determine a new skill path from the current state

Table 2: Sample skills from football.

```
; learned skill for running a short receiver pattern
((short-pattern-completed ?agent ?drop-steps ?dir)
  :start      ((scshort-pattern-completed-c1 ?dir))
  :subgoals ((moved ?agent ?dir)
             (pass-blocked-until-dropped-back ?agent ?drop-steps)
             (moved-until-ball-caught ?agent ?dir)))

; learned skill for receiving a pass after running a short pattern
((short-reception-completed ?receiver ?drop-steps ?dir)
  :start      ((scshort-reception-completed-c1 ?ball))
  :subgoals ((short-pattern-completed ?receiver ?drop-steps ?dir)
             (ran-with-ball-until-tackled ?receiver ?ball)))
```

to the selected goal. This skill selection mechanism enables ICARUS to be persistent to its previous choice, while being reactive to the world if unexpected events occur.

Consider the skills shown in Table 2. The first, *short-pattern-completed*, produces a situation in which a player initially blocks while the passer completes his drop-back, and then runs in direction *?dir* until the ball is caught. The receiver pattern ends if and when the ball is caught. Thus, a goal *(short-reception-completed FB 7 E)* indicates that the fullback (FB) is the intended receiver and runs a short pattern to the right (east) after allowing the quarterback to make a seven-step drop.

4 Learning Skills from Video

The goal of this work is to learn hierarchically structured skills from preprocessed videos in a cumulative manner and in the context of temporal constraints. In this section, we outline our method for acquiring such skills by observing other actors as they achieve similar goals. The input to our learning algorithm consists of a goal, a set of concepts sufficient for interpreting the observed agent's behavior, the set of primitive skills available to the ICARUS agent plus any known non-primitive skills, and the sequence of observed perceptual states.

The algorithm runs in three steps. First, the system observes the entire video-based perceptual sequence on actors achieving the intended goal and infers beliefs about each state (frame). Next, the agent explains how the goal was achieved using existing knowledge. Finally, the algorithm constructs the needed skills along with any supporting knowledge, such as specialized start conditions, based on the explanation generated in the previous step. In the following, we describe the explanation and learning tasks in detail.

4.1 Explaining Observed Behavior

Given a goal and the associated belief memory, the agent must explain how the goal was achieved by the observed actor. Recall that belief memory is episodic and based on hierarchical concepts, so it contains sufficient information to describe the entire sequence at several levels of abstraction. Explanations are generated by either matching inferred beliefs against existing skill knowledge, or by matching inferred beliefs against known concept definitions.

The agent first tries to explain its observations with existing skills by retrieving all skills that can achieve the top-level goal. It then checks that the start conditions and subgoals are consistent with the agent's observations and inferred beliefs. Skills not consistent with these are ignored. The agent then selects a candidate skill and parses the observation sequence into subsequences based on the times when the start conditions and subgoals were achieved. The explanation process then recurses on each subsequence (corresponding to a start condition or subgoal) until the entire observation sequence is explained by a sequence of primitive skills.

If ICARUS fails to explain the observation sequence using skills, it attempts to explain it using concepts. First, it retrieves the belief corresponding to the goal from the belief memory, along with lower-level beliefs that support it. The agent then divides the observed sequence into subsequences based on these supporting lower-level beliefs. As with skill-based explanations, the agent recursively attempts to explain each subgoal. The observation sequence is considered successfully explained if either (1) there exists a skill for the goal that is applicable at the beginning of the trace, in which case there is no need to learn, or (2) all the subgoals of the current trace have already been successfully explained.

4.2 Constructing New Skills

After successfully explaining an observation sequence, the agent learns skills for the achieved goal based on the explanation generated in the previous step. The constructed skills then allow the agent to achieve these goals in the future under similar conditions without additional help from human experts. New skills are generated in two ways, depending on how the explanation was constructed.

If the explanation is based on skill knowledge, then the subgoals of the new skill are the chronologically ordered start conditions and subgoals from the skills that provided the explanation that were achieved during the observed sequence. The start condition for a new skill includes the start conditions and subgoals that were true before or during the first cycle of the observed subsequence.

If the explanation is based on conceptual knowledge, then the learner retrieves the concept definition corresponding to the goal. The subgoals of the new skill correspond to the subconcepts that were achieved during the observed sequence in chronological order. The start condition of the new skill includes the generalized lower-level beliefs (subgoals from the explanation as in Section 4.1) that were true before or during first cycle of the observed subsequence.

Since the concept definition includes information about the temporal relationships among its subconcepts (and therefore the skill's subgoals), the learned skill should retain this information in its start condition. The agent therefore constructs a start condition concept by extracting the start condition subconcepts as well as their associated temporal constraints from the original concept definition. Note that the learned start condition concepts describe both what must be true in the current state and conditions that must have held in the past to ensure that the skill applies.

Examples of learned concepts and skills in the football domain are shown in Tables 1 and 2. Our learning mechanism is

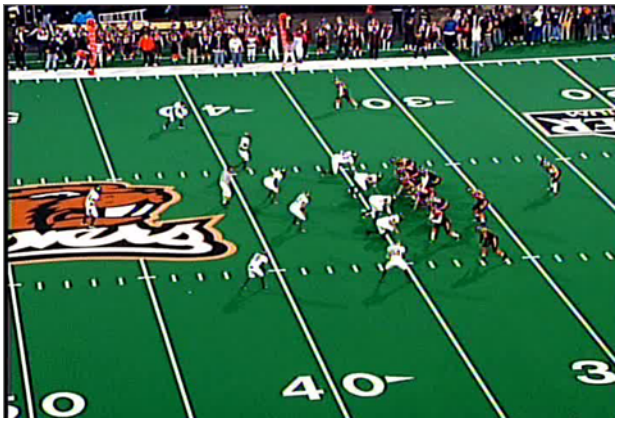

Figure 1: A typical frame from the football video.

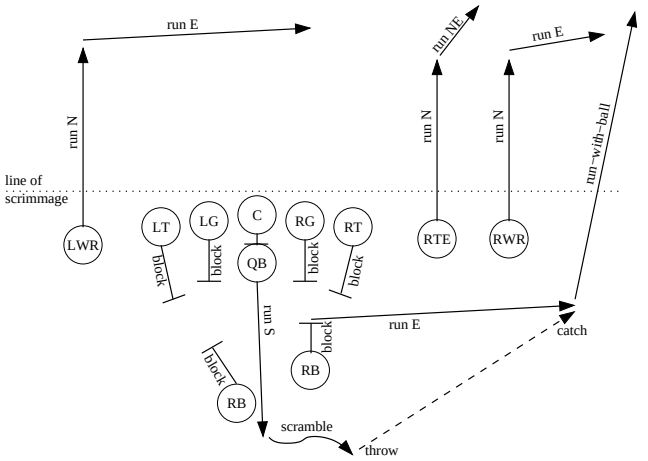

Figure 2: Diagram of the pass play observed by ICARUS with annotations indicating actions taken by individual players.

incremental in the sense that constructed concepts and skills can serve as domain knowledge for future trace explanation. This enables our mechanism to acquire increasingly complex domain knowledge over time.

5 Demonstration and Evaluation in Rush

Our evaluation objectives are to assess whether the learned skills have both high fidelity with respect to the original video, and utility similar to the observed play with respect to the game of football. To evaluate the former, we visually compared the player tracks produced by executing our learned agents in Rush to the player tracks from the source video. For the latter, we compared the yardage gained by the learned agents to yardage gained by hand-constructed agents for the same play.

Three steps were required to acquire the skills from video and then execute them in Rush. First, the raw video was preprocessed to produce a sequence of ICARUS perceptual states. Second, we applied the learning system to construct skills from the sequence. Finally, we mapped the learned eleven-player skills onto an eight-player team, and executed the resulting skills in Rush. Below we provide details for this process, along with the results.

5.1 Preprocessing the College Football Video

Our raw video data was obtained from the Oregon State University football team and corresponds to that used by coaches during game analysis and planning. It was shot via a panning and zooming camera that is fixed at a mid-field location on top of the stadium. The video is qualitatively different than typical television footage in that the camera operator attempts to keep as much of the action in view as possible. A typical shot from our video is shown in Figure 1.

For ICARUS to learn from video, it must be processed into a sequence of ICARUS perceptions. Specifically, the representation should include: (1) player tracks, which give the 2D field coordinate of each player at each time point during the play, and (2) player labels, which describe the functional role of each player (e.g. quarterback, running back, etc), and (3) activity labels, which describe the high-level activity of each player throughout the play.

The tracking data used here is the same as used by Hess and Fern [2009] for their work on supervised learning. The authors manually provided activity labels for each player, and used an automated technique [Hess *et al.*, 2007] to provide the player labels. From these labeled tracks, we generated the perception sequence for the play.

5.2 Mapping College Plays into Rush

As noted, a team in Rush consists of eight players while the teams from the video have eleven players. We therefore map the eleven-player skills learned from the video into skills for the eight Rush players. In practice, we can accomplish this simply by dropping the goals associated with three of the players. Ideally, we would ignore the players that have the least impact on the play.

Figure 2 shows a play diagram for the specific play observed by ICARUS during testing (this information is not included in the perceptual sequence). Rush plays typically include three offensive linemen (RG, C, and LG) along with some combination of four running backs and receivers. The play shown in Figure 2 has five linemen and five backs/receivers. To map this play into Rush, we therefore dropped two linemen (RT and LT) and one running back (RB, left side), all of which had top-level goals of pass-blocking for the duration of the play. ICARUS used the goals for the eight remaining players to guide execution in Rush.

5.3 Evaluating Plays in Rush

The learning task in this work is to acquire hierarchical skills for all eleven offensive football players on the field for a given play. To evaluate the learned skills, we let ICARUS control the offensive players in a football simulator using the learned skills. Ideally, the play produced by the learned skills should look similar to play observed in the video, while advancing the ball on the field.

Figures 3 and 4 show the tracks generated by each offensive player for the observed and simulated plays respectively. Both figures are plotted on the same scale horizontally, similar scales vertically, and use the same initial ball location. The

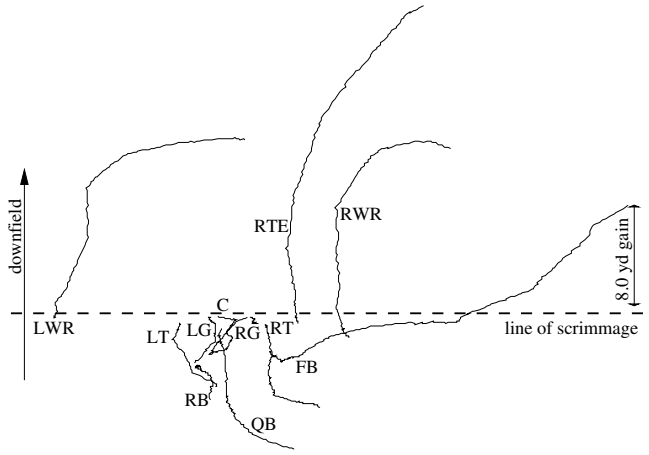

Figure 3: Tracks of players observed by ICARUS in the video.

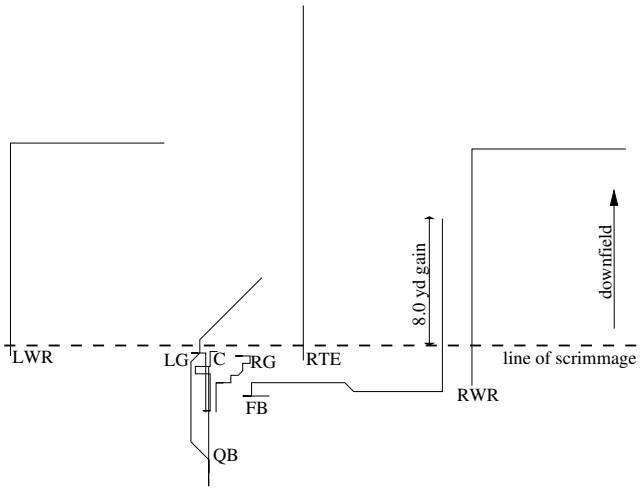

Figure 4: Tracks of players generated by ICARUS using learned skills in the Rush simulator.

tracks generated by ICARUS represent an idealized version of the tracks from the video. Some of the differences, such as round versus square turns, derive from the simple physics used in Rush. Others, such as differences in the tracks of the quarterback (QB) and blockers (C, LG, RG), stem from the behaviors of the defensive players. For example, after completing a 7-yard drop (same in both figures), the quarterback simply "scrambles" in any direction necessary to avoid the oncoming defense while waiting for the intended receiver (FB) to become available. This is equally true in collegiate games and in the simulator.

One difference between the observed and simulated plays relates to the fullback (FB), who does not run as far to the right either before or after catching the ball in the simulator. Prior to the catch, this is partly due to differences in the relative speed of the players, and partly due to timing differences between the video (one tick equals $1/30^{th}$ of a second) and Rush (one tick equals $1/10^{th}$ of a second). Further investigation into these timing differences may yield a higher fidelity response. After the catch, the difference is due to insufficient parameterization of the *run-with-ball* primitive skill. Currently, the skill causes the receiver to run directly downfield regardless of oncoming defenders. To reproduce the play more faithfully, this skill should cause the player to run to the most open part of the field.

A second important divergence involves the tight end (RTE), who does not turn northeast in his simulated route. A closer look at the trace data shows that the ball is caught before he makes the turn, which ends the skill governing his route (run north for 17 yards, then northeast until the ball is caught). This is again an artifact of timing differences between the simulator and the video, which highlights the need to fine-tune plays after they are learned. We return to this issue in the next section.

Finally, both plays produced similar yardage results, but this will not always be the case. Evaluating differences in performance is problematic. Although not done here, we can run the play repeatedly in Rush while varying the characteristics of both the offensive and defensive players (such as speed and agility), along with the defensive strategy (such as formation and patterns) to establish an average. However, we do not have such data for the observed video. Doing so would require us to collect and analyze video based on the same play from several different offensive teams against several different defensive teams. In practice, this type of play is usually aimed at gaining relatively small distances (up to 10 yards), suggesting that our result is reasonable.

6 Related and Future Work

Acquiring domain knowledge is an important task in building intelligent agents. Pearson and Laird [2004] acquire domain knowledge guided by subject matter experts, while Nejati et al. [2006] learn hierarchical task networks by observing preprocessed expert traces. Likewise, Hogg et al. [2008] describe an algorithm that acquires hierarchical task networks from a set of traces and semantically-annotated tasks. Taking a different tack, Forbus and Hinrichs [2004] demonstrate skill learning by analogy. None of these approaches currently support learning from observed behavior or learning coordinated multi-agent strategies. Moreover, the start conditions of the constructed methods only check the current state of the world, while our approach enables the start conditions to describe the past states of the world.

Other game-related efforts have aimed at reducing the effort required to construct game-agents. For example, Kelly et al. [2008] use offline planning with hierarchical task networks to generate scripts automatically. Changes in the environment do not affect the behavior of these constructed agents, whereas the agents built by our system will perform reactively. A second approach uses reinforcement learning algorithms [Bradley and Hayes, 2005; Nason and Laird, 2005] to allow an agent to learn through experience. Our approach differs because learning is analytical and based on observed video rather than agent experience, which makes our approach much more computationally efficient. Our approach also differs from these in that ICARUS is an integrated architecture that combines observation, inference, learning and

execution in a single system.

There are many possible avenues for future development. For example, we need to conduct detailed experiments across many different plays to examine the strengths and weaknesses of our approach. We are also extending our approach to acquire skills for coordinating among multiple agents. Ideally the system would learn knowledge about timing and cooperation among multiple agents. The main differences between the multi-agent system and the current system, would be that (1) beliefs, explanations and skills would be generated for all players in a single episode, rather than multiple single-player episodes, and (2) additional, high-level multi-player coordination skills would also be acquired. This would require only minor extensions to the knowledge representation, inference and execution modules.

A second extension involves modifying the learned skills and the top-level goal parameters (such as number of yards that a receiver runs before changing direction) by learning within Rush. For example, systematic errors such as receivers running into heavy coverage can be detected and revised. This constitutes a structural revision to the learned skills. Similarly, ICARUS can modify the form of learned plays by adjusting the parameters (arguments) in the goals. For example, timing plays that require receivers to run n yards downfield before turning to catch the ball can be adjusted by tuning the value of n.

7 Conclusion

Constructing autonomous agents is an essential task in game development. In this paper, we outlined a system that analyzes preprocessed video footage of human behavior in a college football game, constructs hierarchical skills, and then executes them in a football simulator. The learned skills incorporate temporal constraints and provide for a variety of coordinated behavior among the players. Although the level of precision in ICARUS' control needs improvement, the results suggest that our method is a viable and efficient approach to acquiring the complex and structured behaviors required for realistic agents in modern games. Moreover, we pointed out several avenues for extending this work and improving the quality of the learned agents. Additional studies are required, but we view agent architectures as a promising approach for future development.

8 Acknowledgments

This material is based in part on research sponsored by DARPA under agreement FA8750-05-2-0283. The U. S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed on implied, of DARPA or the U. S. Government.

References

[Anderson, 1993] J. R. Anderson. *Rules of the Mind*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1993.

[Bradley and Hayes, 2005] Jay Bradley and Gillian Hayes. Group utility gunctions: Learning equilibria between groups of agents in computer games by modifying the reinforcement signal. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1914–1921, Edinburgh, UK, 2005. IEEE Press.

[Forbus and Hinrichs, 2004] Ken Forbus and Tom Hinrichs. Companion cognitive systems: A step towards human-level AI. In *Proceedings of the AAAI Fall Symposium on Achieving Human-level Intelligence through Integrated Systems and Research*, Washington, DC, 2004. AAAI Press.

[Hess and Fern, 2009] Robin Hess and Alan Fern. Discriminatively trained particle filters for complex multi-object tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, 2009. IEEE Press.

[Hess *et al.*, 2007] Robin Hess, Alan Fern, and Eric Mortenson. Mixture-of-parts pictorial structures for objects with variable part sets. In *Proceedings of the Eleventh IEEE International Conference on Computer Vision*, Rio de Janeiro, Brazil, 2007. IEEE Press.

[Hogg *et al.*, 2008] Chad Hogg, Héctor Muñoz-Avila, and Ugur Kuter. HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. In *Proceedings of the Twenty-Third Conference on Artificial Intelligence*, Chicago, 2008. AAAI Press.

[Kelley *et al.*, 2008] John Paul Kelley, Adi Botea, and Sven Koenig. Offline planning with hierarchical task networks in video games. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, Stanford, CA, 2008. AAAI Press.

[Laird *et al.*, 1986] John E. Laird, Paul S. Rosenbloom, and Alan Newell. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

[Nason and Laird, 2005] Shelley Nason and John E. Laird. Soar-RL: Integrating reinforcement learning with Soar. *Cognitive Systems Research*, 6(1):51–59, 2005.

[Nejati *et al.*, 2006] Negin Nejati, Pat Langley, and Tolga Konik. Learning hierarchical task networks by observation. In *Proceedings of the 23nd International Conference on Machine Learning*, Pittsburgh, PA, 2006. ACM.

[Newell, 1990] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA, 1990.

[Pearson and Laird, 2004] Douglas Pearson and John E. Laird. Redux: Example-driven diagrammatic tools for rapid knowledge acquisition. In *Proceedings of Behavior Representation in Modeling and Simulation*, Washington D.C., 2004.

[Stracuzzi *et al.*, 2009] David J. Stracuzzi, Nan Li, Gary Cleveland, and Pat Langley. Representing and reasoning over time in a symbolic cognitive architecture. In *Proceedings of the 31st Annual Meeting of the Cognitive Science Society*, Amsterdam, Netherlands, 2009. Cognitive Science Society, Inc.