Available online at www.sciencedirect.com

SCIENCE DIRECT®

Cognitive Systems Research 6 (2005) 51–59

Cognitive Systems RESEARCH

www.elsevier.com/locate/cogsys

Soar-RL: integrating reinforcement learning with Soar

Action editor: Christian Schunn

Shelley Nason [*], John E. Laird

University of Michigan, 1101 Beal Avenue, Ann Arbor, MI 48109-2110, USA

Received 16 September 2004; accepted 16 September 2004
Available online 28 October 2004

Abstract

In this paper, we describe an architectural modification to Soar that gives a Soar agent the opportunity to learn statistical information about the past success of its actions and utilize this information when selecting an operator. This mechanism serves the same purpose as production utilities in ACT-R, but the implementation is more directly tied to the standard definition of the reinforcement learning (RL) problem. The paper explains our implementation, gives a rationale for adding an RL capability to Soar, and shows results for Soar-RL agents' performance on two tasks.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Soar; Reinforcement learning; Cognitive architectures

1. Introduction

The Soar architecture has been used extensively, both for developing AI applications and cognitive models. One of its strengths has been the ability to efficiently represent and use large bodies of symbolic knowledge to solve a wide variety of problems using many different methods. It dynamically combines available knowledge for decision-making, and can dynamically create subgoals whenever the knowledge for a decision is incomplete or inconsistent. Soar can also compile the problem solving in subgoals into rules, using a process called *chunking*, so that over time, problem solving in subgoals is replaced by rule-driven decision making. Chunking has proved to be extremely versatile because it stores away whatever problem solving is performed in a subgoal, allowing Soar programs to learn using a wide variety of methods, including explanation-based learning, macro-operator learning, strategy acquisition, learning by instruction, and many others. In general, Soar's processing is symbolic, and although that is sufficient (and necessary) for a wide variety of cognitive activities, it is inadequate (or at the very least extremely inefficient) when encoding probabilities and numeric rewards.

[*] Corresponding author.
E-mail addresses: snason@umich.edu (S. Nason), laird@umich.edu (J.E. Laird).

1389-0417/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.cogsys.2004.09.006

While Soar has strengths in knowledge-rich symbolic reasoning and learning and weaknesses in knowledge-lean, statistical-based learning, the strengths and weaknesses of reinforcement learning (RL) techniques are the reverse. They are successful at capturing statistical regularities related to the expected reward that an agent will receive, but cannot encode and effectively use large bodies of symbolic knowledge. In this paper, we will present an initial integration of reinforcement learning with Soar, enriching the learning capabilities as well as the representation of knowledge in Soar, while at the same time developing a unique integration of reinforcement learning with symbolic, knowledge-rich reasoning. Specifically, Soar supports dynamic hierarchical task-decomposition, meta-reasoning, and the ability to enrich the state descriptions through internal abstractions. All of these capabilities both complicate and enrich reinforcement learning. This integration requires structural changes to the Soar architecture and we will refer to the unification as Soar-RL.

In the remainder of this paper, we first present a simplified description of Soar and the extensions we have made to incorporate reinforcement learning. We then demonstrate the implementation on two simple tasks, highlighting the contributions RL makes to Soar, as well as the capabilities Soar-RL provides beyond standard reinforcement learning. We also compare and contrast Soar-RL to ACT-R, which incorporates a rule-tuning mechanism, comparable to reinforcement learning. Finally, we outline future directions and offer our conclusions.

2. Soar

The structure of Soar's memories is shown in Fig. 1. Soar has a declarative working memory, which contains its representation of the current situation using labeled graph structures, organized in a hierarchy of states/goals. All long-term procedural knowledge is encoded as production rules. Whenever a rule's conditions match working memory, the rule is fired and its actions performed. Actions may involve adding or removing structures from working memory. They may also create preferences used to select operators.

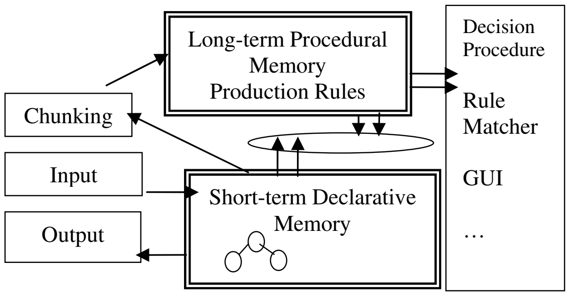
Long-term Procedural Memory Production Rules
Decision Procedure
Rule Matcher
GUI
…
Chunking
Input
Output
Short-term Declarative Memory

Fig. 1. Soar's structure.

Soar's learning mechanism, chunking, monitors problem solving and automatically creates new rules, which are added to long-term memory during execution.

Soar's basic reasoning cycle is illustrated in Fig. 2.

1. *Input:* Changes to perception are processed and Soar's perceptual buffer in working memory is updated.
2. *State elaboration:* Rules that newly match are fired in parallel to retrieve relevant information. For example, in a robotic task, a rule might test the distance to the object and the robot's available reach and determine if the object is within reach.
3. *Proposing operators:* Rules can propose operators by creating *acceptable* preferences for specific operators. In general, the rules' conditions test the situation so that an operator is proposed only when it is relevant.
4. *Comparing and evaluating operators:* Rules can test the proposed operators and other features of the situation and create preferences, which make assertions about the absolute or relative merit of the operators. Multiple preferences can be generated for a single operator.
5. *Selecting the current operator:* The preferences are evaluated to select the current operator. If the preferences are insufficient or contradictory, an impasse ensues and Soar creates a substate in which the goal is to resolve that impasse. This provides Soar with meta-reasoning so that it can reflect on its own processing.