Soar-RL
A Year of “Learning”

Nate Derbinsky
University of Michigan
Outline

- The Big Picture
- Developing Soar-RL Agents
- Controlling the Soar-RL Algorithm
- Debugging Soar-RL
- Soar-RL Performance
- Nuggets & Coal
- Additional Resources
Outline

- **The Big Picture**
  - The Path to Release
  - How Soar-RL Affects Agent Behavior
- Developing Soar-RL Agents
- Controlling the Soar-RL Algorithm
- Debugging Soar-RL
- Soar-RL Performance
- Nuggets & Coal
- Additional Resources
The Path to Release

- Credit for most system functionality and all research to make Soar-RL possible should go to Shelley Nason and John Laird

- The work being presented today deals with the engineering efforts to effectively and efficiently integrate Soar-RL with the Soar trunk
  - Nate Derbinsky, Nick Gorski, John Laird, Bob Marinier, Jonathan Voigt, Yongjia Wang
The RL Agent-Environment Interface

Sutton, R.S., and Barto, A.G., Reinforcement Learning: An Introduction.
Soar-RL Agent-Environment Interface

Soar Agent

Environment

input-link $S_t$

reward-link $r_t$

output-link $a_t$

$r_{t+1}$

$r_{t+1}$
Numeric Indifferent Preferences

- $(\text{state} \ ^\text{operator} \ <\text{operator}> = \text{number})$
  - \text{number}, the value of the preference, is a numeric constant

- The value of the numeric indifferent preference may bias selection of the \text{operator} from amongst indifferent preferences
  - \text{numeric-indifferent-mode} determines how values combine
  - \text{indifferent-selection} sets the policy for deciding amongst indifferent preferences
How Soar-RL Affects Agent Behavior

- Over time, Soar-RL modifies numeric indifferent preference values such as to maximize the expected receipt of future reward.

- Altering preference values in procedural memory allows Soar-RL to modify the outcome of operator selection, and thus affect agent behavior.
Water Jug Demonstration
Outline

- The Big Picture
- Developing Soar-RL Agents
  - Soar-RL Rules
  - Templates
  - Reward
- Controlling the Soar-RL Algorithm
- Debugging Soar-RL
- Soar-RL Performance
- Nuggets & Coal
- Additional Resources
Soar-RL Rules

- LHS can be anything
- RHS must be single numeric indifferent preference
- Soar-RL rules form a representation of a value function
  - $Q(s, o) = 2.3$

```
sp {my*rl*rule
   (state <s> ^operator <o> + ^attrib-a alpha ^attrib-b beta)
       (<o> ^name my-op)
   -->
       (<s> ^operator <o> = 2.3)
}
```
Water-Jug Agent Example

```
sp {water-jug*empty*small*0*0
  (state <s> ^name water-jug ^operator <op> +
   ^jug <j1> <j2>)
  (<op> ^name empty ^empty-jug.volume 3)
  (<j1> ^volume 3 ^contents 0)
  (<j2> ^volume 5 ^contents 0)
  -->
  (<s> ^operator <op> = 0)
}
```
Soar-RL Rule Usage

- In order for Soar-RL to affect selection of an operator in a particular state, a Soar-RL rule must exist whose LHS matches the state-operator pair.

- With complex agents, the requirement of manually representing the Q-function with Soar-RL rules is unreasonable.
  - Solutions: scripting or templates.
Soar-RL Templates

- Must have :template flag
- LHS can be anything
- RHS must be single numeric indifferent preference
- A Soar-RL template is a representation of the initial value function of a set of state-operator pairs

```sp {my*rl*template :template (state <s> ^operator <o> + ^attrib-a <a> ^attrib-b <b>) (<o> ^name my-op) --> (<s> ^operator <o> = 2.3) }
```
sp {water-jug*empty
  :template
  (state <s> ^name water-jug ^operator <op> +
    ^jug <j1> <j2>)
  (<op> ^name empty ^empty-jug.volume <evol>)
  (<j1> ^volume 3 ^contents <c1>)
  (<j2> ^volume 5 ^contents <c2>)
}

-->  
  (<s> ^operator <op> = 0)
}
Soar-RL Template Behavior

- **During proposal** phase, the template rule is supplied to the matcher
  - Matches are used to create new Soar-RL productions that contribute to the current cycle and future decisions

- The new production has naming pattern `rl*template-name*id`
  - template-name – original template rule
  - id – auto incrementing counter
Water-Jug Agent Example

sp {rl*water-jug*empty*1
  (state <s> ^name water-jug ^operator <op> +
    ^jug <j1> <j2>)
  (<op> ^name empty ^empty-jug.volume 3)
  (<j1> ^volume 3 ^contents 0)
  (<j2> ^volume 5 ^contents 0)
-->
  (<s> ^operator <op> = 0)
}

Reward

- The agent programmer must supply reward information to guide the reinforcement learning process.

- Location of reward is a new structure, a state’s **reward-link**
  - state.reward-link.reward.value
    - state ^reward-link.reward.value 1.2
    - state ^reward-link.reward.value -2

- The **reward-link** is not part of the **io-link** and is not modified directly by the environment.
sp {water-jug*detect*goal*achieved
   (state <s> ^name water-jug
           ^jug <j> ^reward-link <r>)
   (<j> ^volume 3 ^contents 1)
--> (write (crlf) |The problem has been solved.|)
   (<r> ^reward.value 10)
   (halt)
Outline

- The Big Picture
- Developing Soar-RL Agents
- **Controlling the Soar-RL Algorithm**
  - Operator Selection
  - Reinforcement Learning
  - Manipulating Soar-RL Parameters
- Debugging Soar-RL
- Soar-RL Performance
- Nuggets & Coal
- Additional Resources
Operator Selection

- The purpose of learning a Q-function is that the agent can act optimally by selecting the operator with the highest Q-value.
- In Soar preference semantics, symbolic preferences take precedence over numeric preferences.
  - Only if there would be a tie are numeric preferences considered.
Exploration vs. Exploitation

- For reinforcement learning to discover the optimal policy, it is necessary that the agent sometimes choose an action that does not have the maximum predicted value
  - Often occurs during initial learning and as a result of a change in the task
- Control of the exploration policy takes place via the `indifferent-selec**tion**` command
Preference Updates

- Soar-RL does Temporal Difference (TD) learning:
  \[ \text{update} = \alpha( \text{target} - \text{current} ) \]

- Current estimate = \( Q(s_t, o_t) \)

- \( \alpha \) = Learning rate

- Target estimate and application of update are affected by a number of Soar-RL parameters

- Updates are applied at the beginning of the next decision phase
Gaps in Rule Coverage

- Since TD updates are transmitted backwards through the stored Q-function, it would seem necessary that the function be well-represented by Soar-RL rules at each decision cycle.

- To address this practical issue, Soar-RL provides preliminary support for automatic propagation of updates over “gaps”.

- By default, Soar-RL will automatically propagate updates over gaps, discounted exponentially with respect to the length of the gap.

- This behavior can be enabled/disabled by manipulating the `temporal-extension` parameter.
Gaps Example

go1 → no2 → go3

temporal-extension

reward
Hierarchical Reinforcement Learning

- HRL is RL done over a hierarchically decomposed structure.
  - Learning can be done to improve subtask performance, as well as selection amongst subtasks.
- Hierarchical Soar-RL is built on Soar’s impasse structure.
Op No-Change Example

- Rewards at S1 after O1 are attributed to O1, discounted with respect to the number of decision cycles.
- Rewards at S2 are attributed to the respective operator.
- After O13, reward is checked at S2 and, if present, attributed directly to O13.
Other Soar-RL Features

- Exploration Policies
  - Boltzmann, Epsilon Greedy, Softmax, First, Last

- Learning Policies
  - On-policy, Off-policy

- Reward Discounting

- Reward Accumulation

- Eligibility Traces
Manipulating Soar-RL Parameters

- Get a parameter
  - `rl [-g|--get] <name>`

- Set a parameter
  - `rl [-s|--set] <name> <value>`

- Get all values
  - `rl`

- Get Soar-RL statistics
  - `rl [-S|--stats] <statistic>`
Outline

- The Big Picture
- Developing Soar-RL Agents
- Controlling the Soar-RL Algorithm
- Debugging Soar-RL
- Soar-RL Performance
- Nuggets & Coal
- Additional Resources
Debugging Soar-RL

- New `watch` switches
  - `--indifferent-selection` = view numeric preferences for each operator
  - `--template` = view firing of templates
  - `--rl` = debugging information

- New `print` and `excise` switches
  - `--rl` = all Soar-RL rules
  - `--template` = all Soar-RL templates

```
rl*water-jug*empty*46  1.  0.
rl*water-jug*pour*45  1.  3.
```
New Decision Cycle Commands

- **select <id>**
  - Forces the selection of an operator

- **predict**
  - Determines which operator will be chosen during the next decision phase
  - If operator selection will require probabilistic selection predict will manipulate the random number generator to enforce its prediction (assuming no preference changes)
Outline

- The Big Picture
- Developing Soar-RL Agents
- Controlling the Soar-RL Algorithm
- Debugging Soar-RL
- **Soar-RL Performance**
  - TestSoarPerformance
  - Rules vs. Templates
- Nuggets & Coal
- Additional Resources
## TestSoarPerformance

<table>
<thead>
<tr>
<th>Platform</th>
<th>8.6.4</th>
<th>RL</th>
<th>Δ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS X (RL on)</td>
<td>8.067</td>
<td>8.231</td>
<td>2.0%</td>
</tr>
<tr>
<td>OS X (RL off)</td>
<td></td>
<td>8.201</td>
<td>1.7%</td>
</tr>
<tr>
<td>Linux (RL on)</td>
<td>3.593</td>
<td>3.660</td>
<td>1.9%</td>
</tr>
<tr>
<td>Linux (RL off)</td>
<td></td>
<td>3.637</td>
<td>1.2%</td>
</tr>
<tr>
<td>Windows XP (RL on)</td>
<td>3.703</td>
<td>3.765</td>
<td>1.7%</td>
</tr>
<tr>
<td>Windows XP (RL off)</td>
<td>3.703</td>
<td>3.725</td>
<td>0.6%</td>
</tr>
</tbody>
</table>
Rules vs. Templates

<table>
<thead>
<tr>
<th>Water Jug</th>
<th>Rules</th>
<th>Templates</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS X</td>
<td>.043</td>
<td>.065</td>
<td>51%</td>
</tr>
<tr>
<td>Linux</td>
<td>.024</td>
<td>.033</td>
<td>38%</td>
</tr>
<tr>
<td>Windows XP</td>
<td>.125</td>
<td>.140</td>
<td>12%</td>
</tr>
</tbody>
</table>
Outline

- The Big Picture
- Developing Soar-RL Agents
- Controlling the Soar-RL Algorithm
- Debugging Soar-RL
- Soar-RL Performance
- Nuggets & Coal
- Additional Resources
Nuggets & Coal

- Nuggets
  - Soar-RL is an integration of reinforcement learning with Soar
  - Soar-RL provides a highly configurable new learning mechanism with a relatively small performance cost
  - Soar-RL\textsubscript{beta} is available for download today!

- Coal
  - Current template implementation takes a heavy toll
Outline

- The Big Picture
- Developing Soar-RL Agents
- Controlling the Soar-RL Algorithm
- Debugging Soar-RL
- Soar-RL Performance
- Nuggets & Coal
- Additional Resources
Additional Resources

- http://winter.eecs.umich.edu/soar
  - Binaries
  - Tutorial
  - Manual
    - Programmer Reference