Reinforcement learning and Soar

Shelley Nason

Motivation

- Allow Soar to learn about statistical regularities in the environment
- Use rewards from inner motivation, likes/dislikes, emotion to bias behavior
- Fine-tune behavior
  - Learn preferences the programmer didn’t bother to write or didn’t realize were important

The goal

- Automatic and general-purpose learning (like chunking)
- Ultimately avoid task-specific hand-coding of features
- Currently requires some care in writing rules for proper learning

Introducing numeric preferences

- Productions of the form-

```
sp {random*production
    (state <s> ^operator <o> +)
    … (other conditions)
    →
    (<s> ^operator <o> = -0.7)}
```

- New decision phase:
  - Process all reject/better/best/etc. preferences
  - Compute value for remaining candidate operators by summing numeric preferences
  - Choose operator by softmax (Boltzmann)

Rewards

- Rewards are numeric values created at specified place in WM. The architecture watches this location and collects its rewards.
- Source of rewards
  - productions included in agent code
  - written directly to io-link by environment
  - Future – generated by emotion or physiology system

Fitting within RL framework

- The sum over numeric preferences has a natural interpretation as an action value Q(s,a), the expected discounted sum of future rewards, given that the agent takes action a from state s.
- Here, action a is operator
- What is state s?

Updating operator values

O1 O2

conditions

r = reward

Q(s,O1) =

sum of numeric prefs.

Q(s′,O2) =

sum of numeric prefs.

- Sarsa update-
  $Q(s,O1) \leftarrow Q(s,O1) + \underline{\beta[r + \lambda Q(s',O2) - Q(s,O1)]}$
- new numeric preference has value corresponding to underlined portion

Rudimentary condition collection

- This assumes tabular state representation.
- For instance, waterjug.
- Learn rules directly from operator proposals-

```
sp {waterjug*propose*fill
   (state <s> ^jug <i>)
   (<i> ^contents 0)
   -->
   (<s> ^operator <o> + =)
   (<o> ^name fill
       ^jug <i>)}
```

```
sp {|RL-13|
   (state <s1> ^jug <i1>
              ^operator <o1> +)
   (<i1> ^contents 0)
   (<o1> ^name fill ^jug <i1>)
   -->
   (<s1> ^operator <o1> = -0.25)}
```

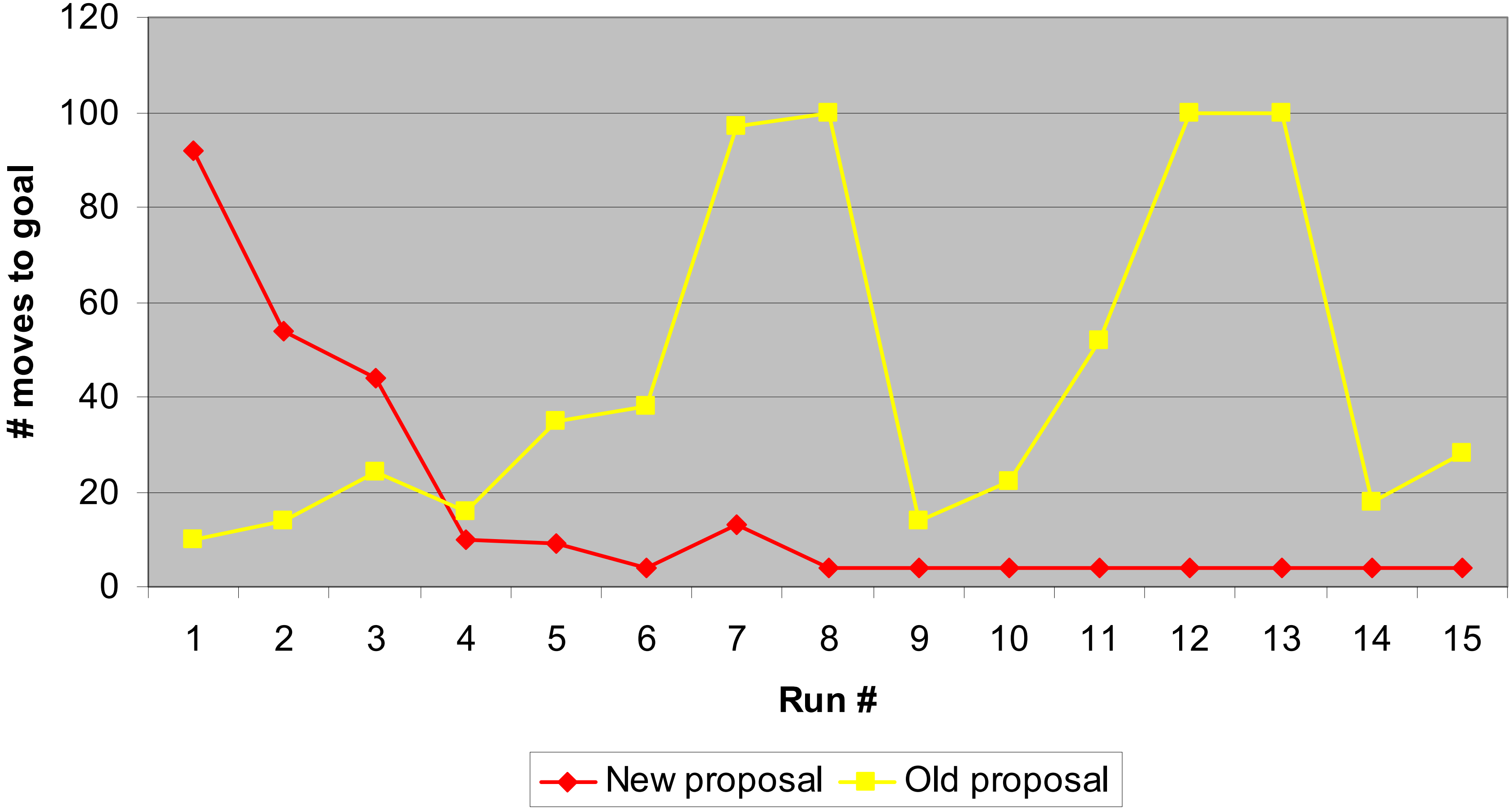
Rudimentary condition collection

- This doesn’t work without rewriting operator proposals to include complete state description.
- But writing proposals this way confuses applicability with desirability.

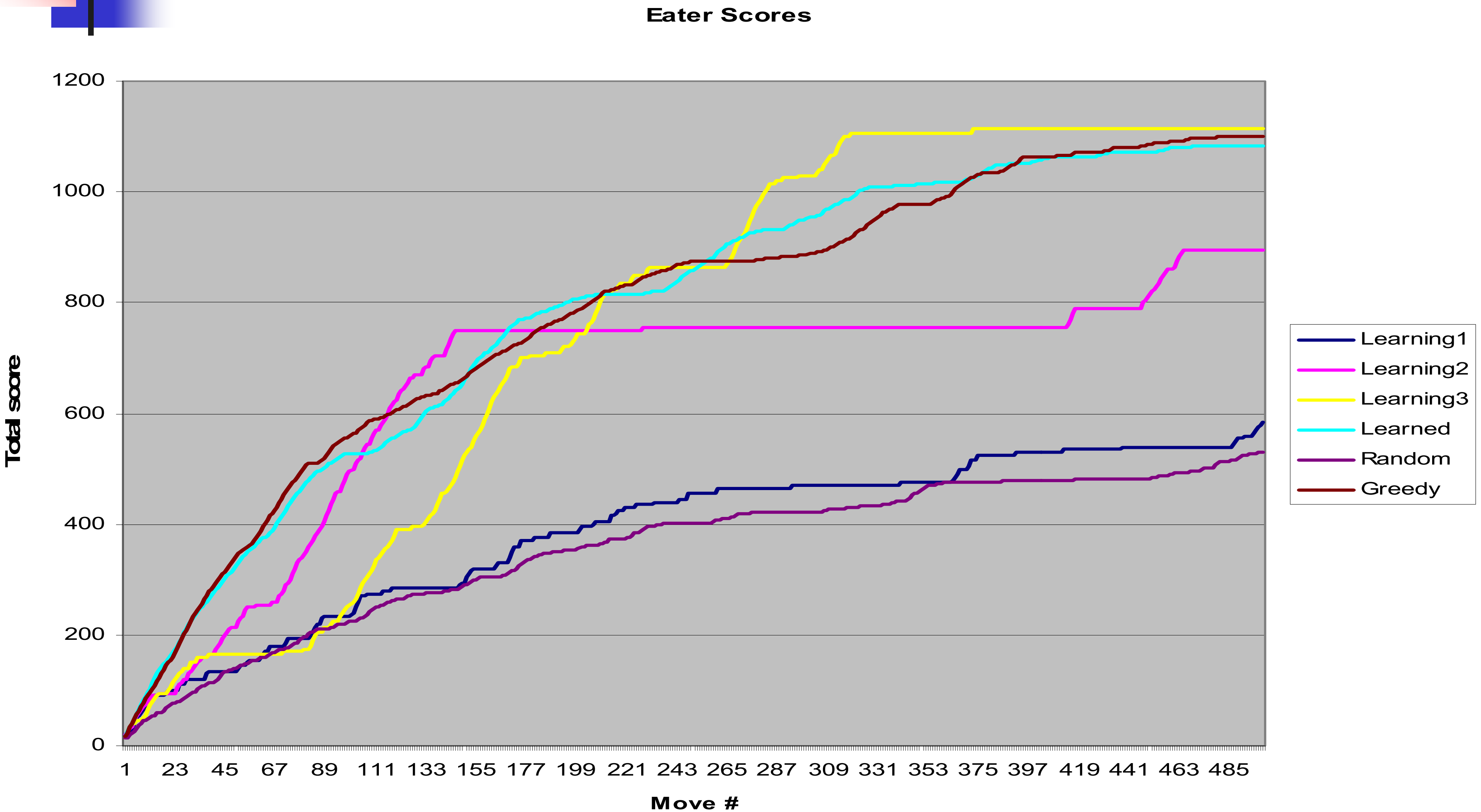
```
sp {waterjug*propose*fill
   (state <s> ^jug <i>
              { <> <i> <j> } )
   (<i> ^contents 0
       ^volume <v1>)
   (<j> ^contents <c>
       ^volume <v2>)
   -->
   (<s> ^operator <o> + =)
   (<o> ^name fill
       ^jug <i>)}
```

```
sp {|RL-31|
   (state <s1> ^jug <i1>
             { <> <i1> <j1> }
            ^operator <o1> +)
   (<i1> ^volume 3 ^contents 3)
   (<j1> ^volume 5 ^contents 0)
   (<o1> ^name fill ^jug <j1>)
   -->
   (<s1> ^operator <o1> = -0.225)}
```

Waterjug results



Eaters Results



Improved condition collection

- The charming thing about doing reinforcement learning in Soar is that we can invent new features and conditions to associate values with.
- The less charming part is lack of theory for arbitrarily adding features.

Improved condition collection: State generalization

- To generalize Q-values over states:
  - Consider LHS's of numeric preferences as set of (perhaps binary) features
    - `{if energy low and <o> = shields-on, <o> = -5}`
  - How to combine features into a numeric value?
    - linear functions
    - neural nets
    - memory-based methods
    - etc.

Improved condition collection: Feature generation

- To generate set of features (LHS's):
  - Suggested by programmer, via prototype productions:

```
sp {(state <s> ^operator <o> +
               ^energy <e>)
    (<o> ^name shields-on)
-->
    (<s> ^operator <o> = 0)}
```

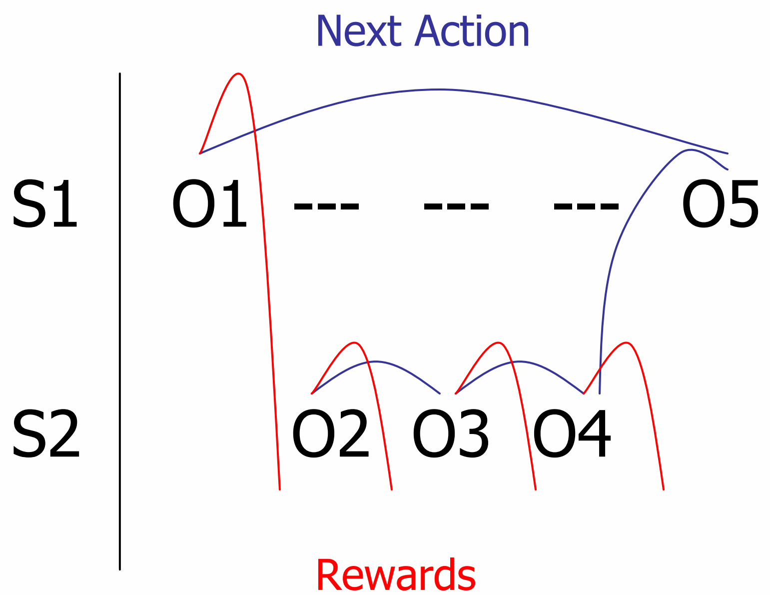
  - Activation based
  - Learned
    - perhaps utilizing episodic memory

Substates – Tie impasses

- Reintroduce tie impasses when value-based information insufficient or conflicting
- Confidence – a function of
  - # of matching numeric preferences
  - average of abs([$r + \lambda Q(s',a') - Q(s,a)$])
  - size of difference in values for proposed operators
- Tie impasses could be a place to learn additional discriminating features

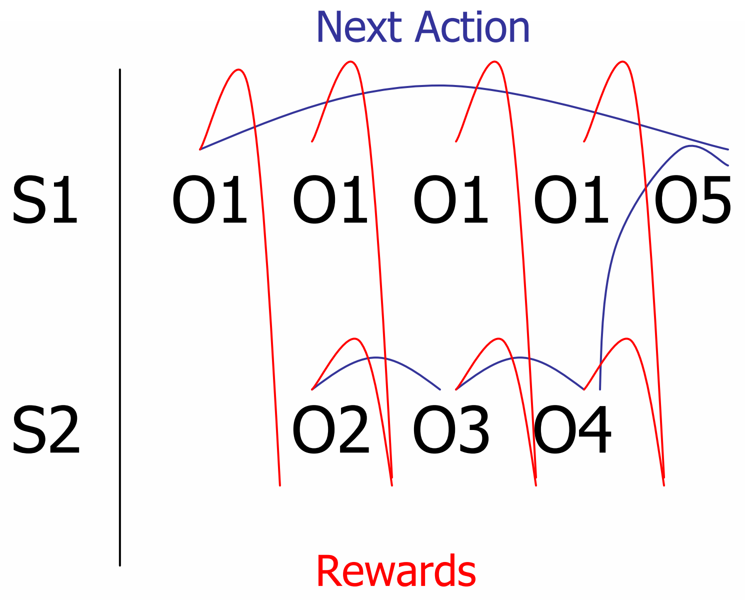
Substates-
Learning over substates

- Tie / state no-change impasses



Substates- Learning over states

- Operator no-change: possible options-like framework



Conclusion-Difficulties

- How much to adopt machine learning techniques while fitting neatly within Soar
- Haven't settled on method for generalizing Q-function
- Need to test in more domains; good empirical results to take the place of convergence proofs

Conclusion- Good Points

- Agents learned good behavior without requiring any programmer-specified control knowledge
- Could be very useful once expanded to work in harder domains