Soar-RL: Reinforcement Learning and Soar

Shelley Nason

Reinforcement Learning

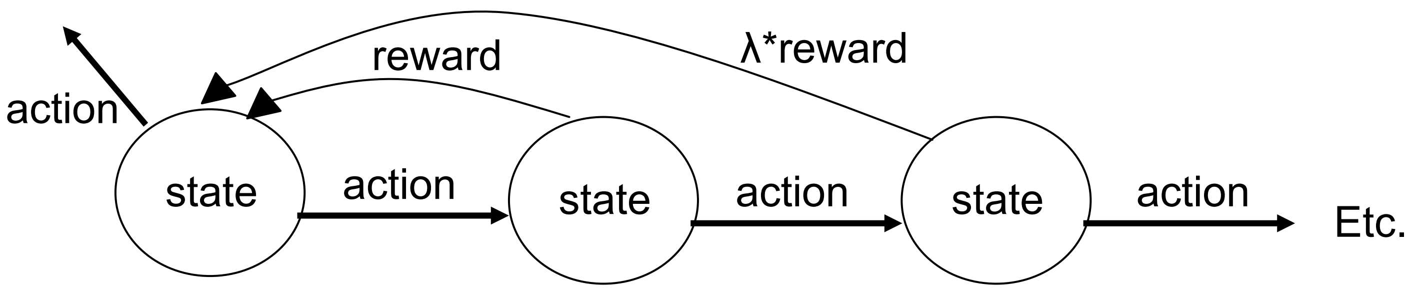
- Reinforcement learning:
  Learning how to act so as to maximize the expected cumulative value of a (numeric) reward signal
- In Soar terminology, RL learns operator comparison knowledge

A learning method for low-knowledge situations

- Non-explanation-based, trial and error learning – RL does not require any model of operator effects to improve action choice.
- Additional requirement – rewards.
- Therefore RL component should be automatic and general-purpose.
- Ultimately avoid
  - Task-specific hand-coding of features
  - Hand-decomposed task or reward structure
  - Programmer tweaking of learning parameters
  - And so on

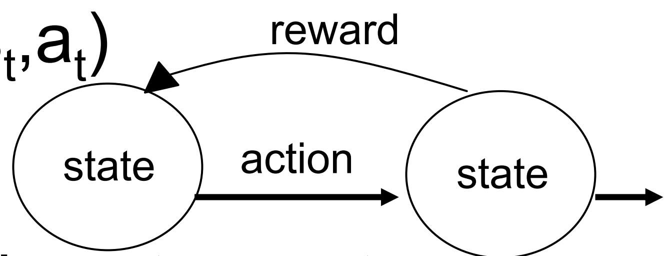
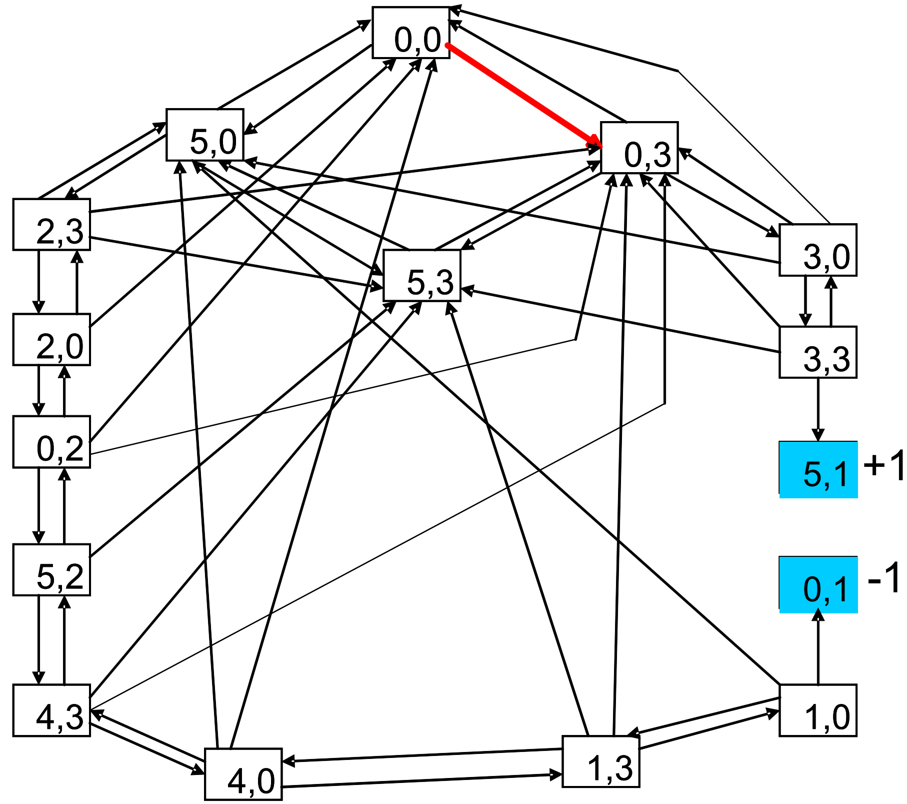
Q-values

- Q(s,a): the expected discounted sum of future rewards, given that the agent takes action a from state s, and follows a particular policy thereafter

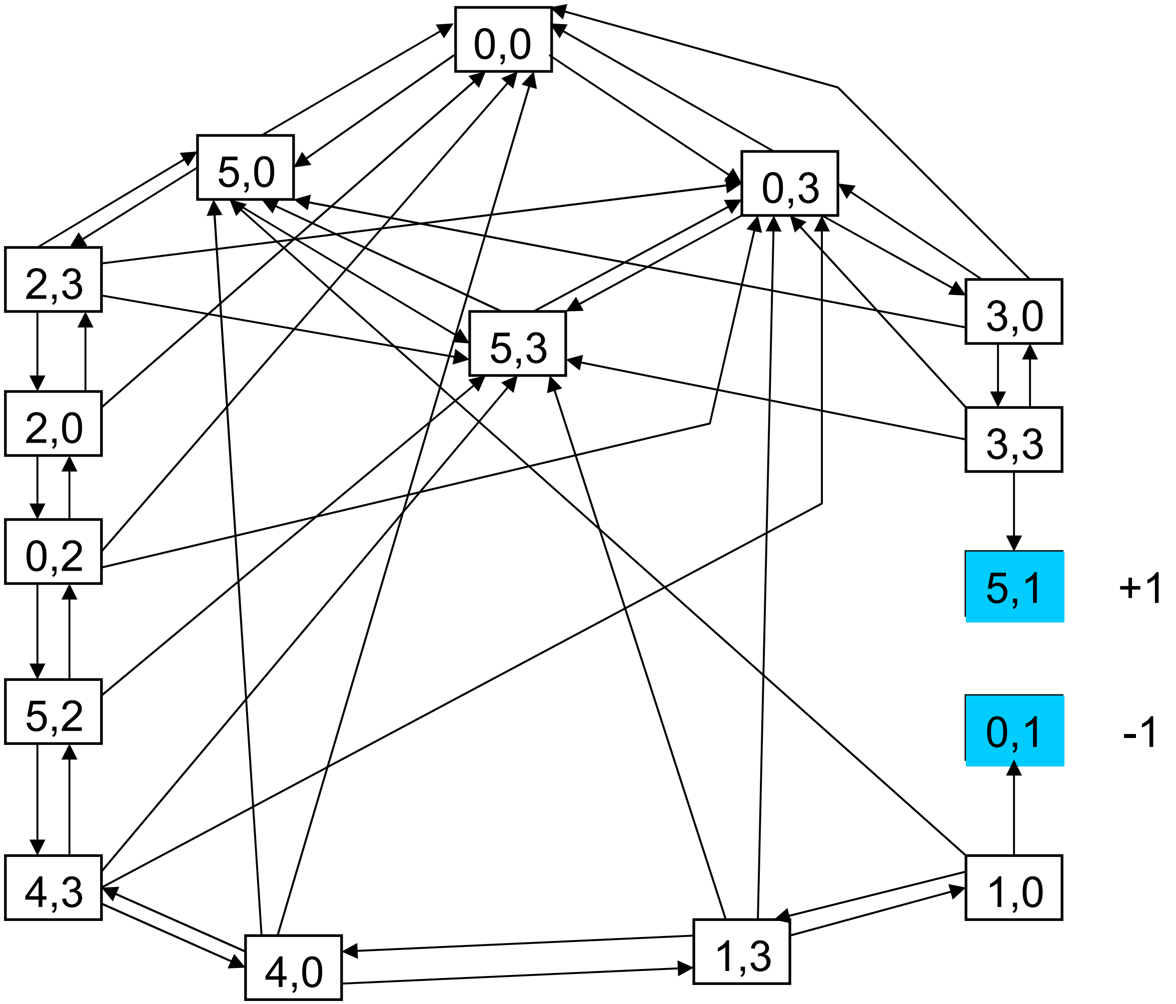


- Given optimal Q-function, selecting action with highest Q-value at each state yields optimal policy

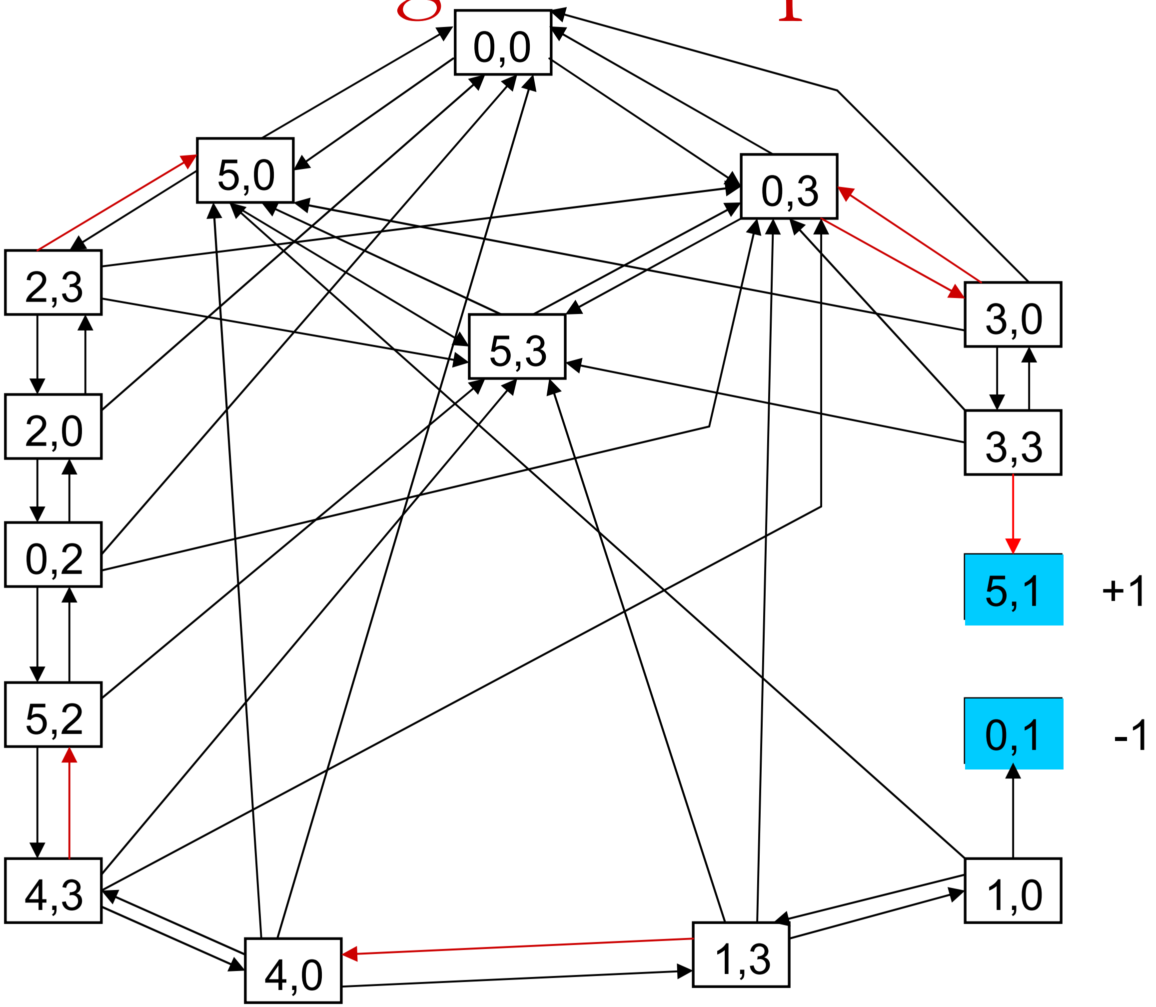
Representing the Q-function

- In Soar-RL, Q-function stored as productions, testing state and operator, and asserting numeric preferences.
- ```
  Sp{RL-rule
  (state <s> ^operator <o> +)
     …
    →
  (<s> ^operator <o> = 0.33231)}
  ```
- During decision phase, the Q-value of an operator **O** is taken to be the sum of all numeric preferences asserted for **O**.
  ```

Learning

- Q-value represented as concatenation:
  StateXAction → Set of Features → Value

Automatic Feature Generation

Q-learning (updating parameters of linear function)

- Q-learning: Move the value of $Q(s_t, a_t)$ toward $r_t + \gamma_* \max_a Q(s_{t+1}, a)$.



- Bootstrapping: Update the prediction at one step using the prediction at the next step.

Current Work-
Automatic Feature generation

- Constructing rule conditions with which to associate values
- Since values stored with RL rules, some RL rule must fire for every state-action pair for bootstrapping to work
- Sufficient distinctions required that agent will not confuse state-action pairs with significantly different Q-values
- Want rules that take advantage of opportunities for generalization

Waterjug Task-
A reasonable set of rules

- One for each state-action pair, for instance-
- ```
  Sp {RL-0003
   :rl
   (state <s> ^jug <j1>
              ^jug <j2>
              ^operator <o> +)
   (<j1> ^volume 5 ^contents 5)
   (<j2> ^volume 3 ^contents 0)
   (<o> ^name fill ^jug <j2>)
  →
   (<s> ^operator <o> = 0)}
  ```



- 46 RL rules
  ```

How to generate rule automatically?

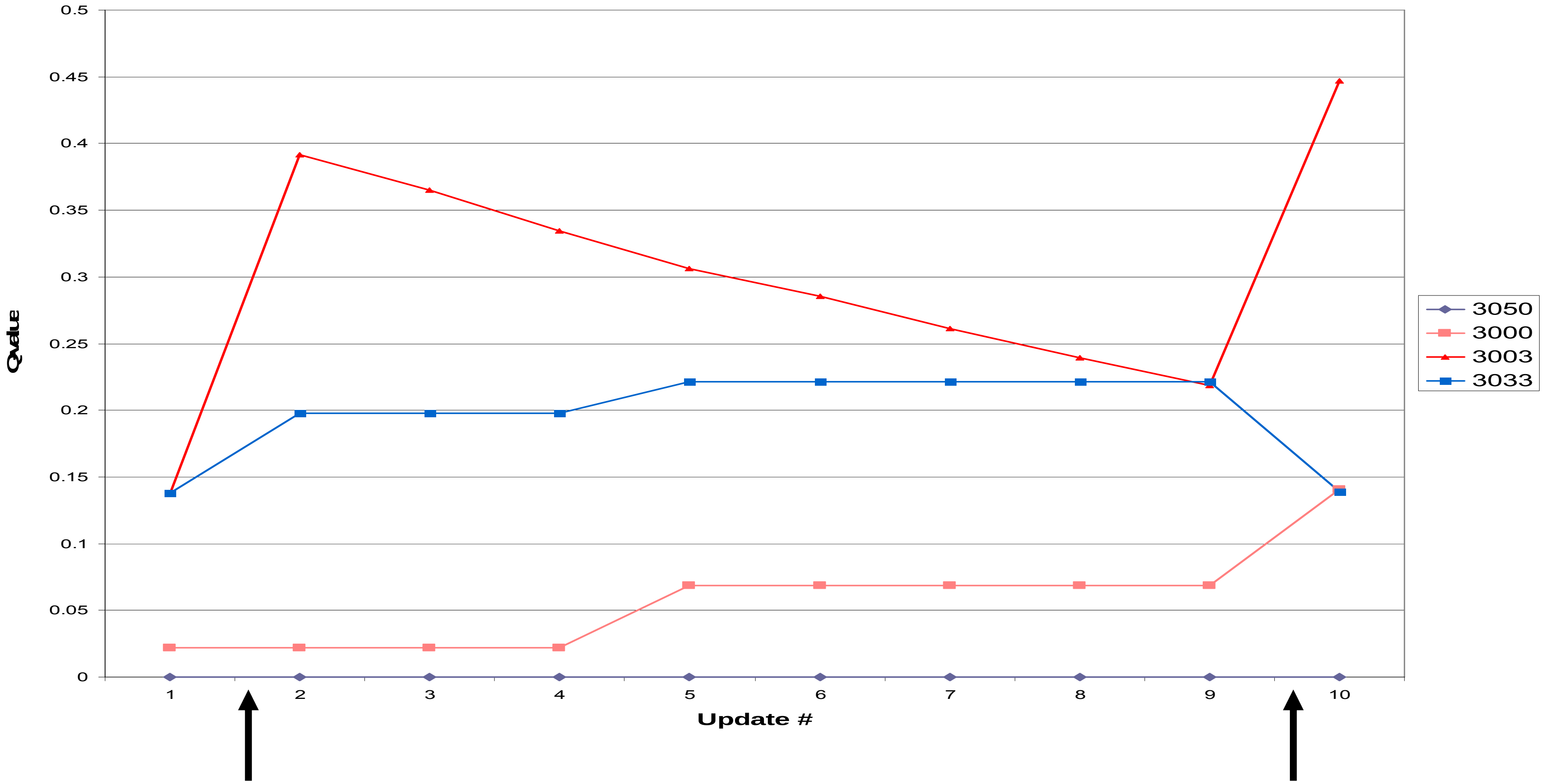
- Rule could be built from WM, for instance-
- ```
  sp {RL-1
  :rl
  (state <s> ^name waterjug ^jug <j1>
    ^jug <j2> ^operator <o4> +
    ^superstate nil ^type state)
  (<j1> ^contents 0 ^free 5 ^volume 5)
  (<j2> ^contents 0 ^free 3 ^volume 3)
  (<o4> ^name fill ^jug <j2>)
  →
  (<s> ^operator <o4> = 0)}
  ```
  ```

But we want generalization…



Adaptive representations

- System constructs feature set so that more distinctions in parts of state-action space requiring more distinctions.
- Specific-to-general:
  Collect instances and cluster according to similar values.
- General-to-specific:
  Add distinctions when area with single representation appears to contain multiple values.

General-to-specific
Our most general rules

- Rules made from operator proposals
- ```
  Sp {RL-1
  :rl
  (state <s1> ^name waterjug ^jug <j1>
                ^operator <o1> +)
  (<j1> ^free 3)
  (<o1> ^jug <j1> ^name fill)
  -->
  (<s1> ^operator <o1> = 0)}
  ```
- Only generated when no rule fires for the selected operator
  ```

Specialization – Example of overgeneral representation



If jug has contents 3, then pour from jug into other jug.

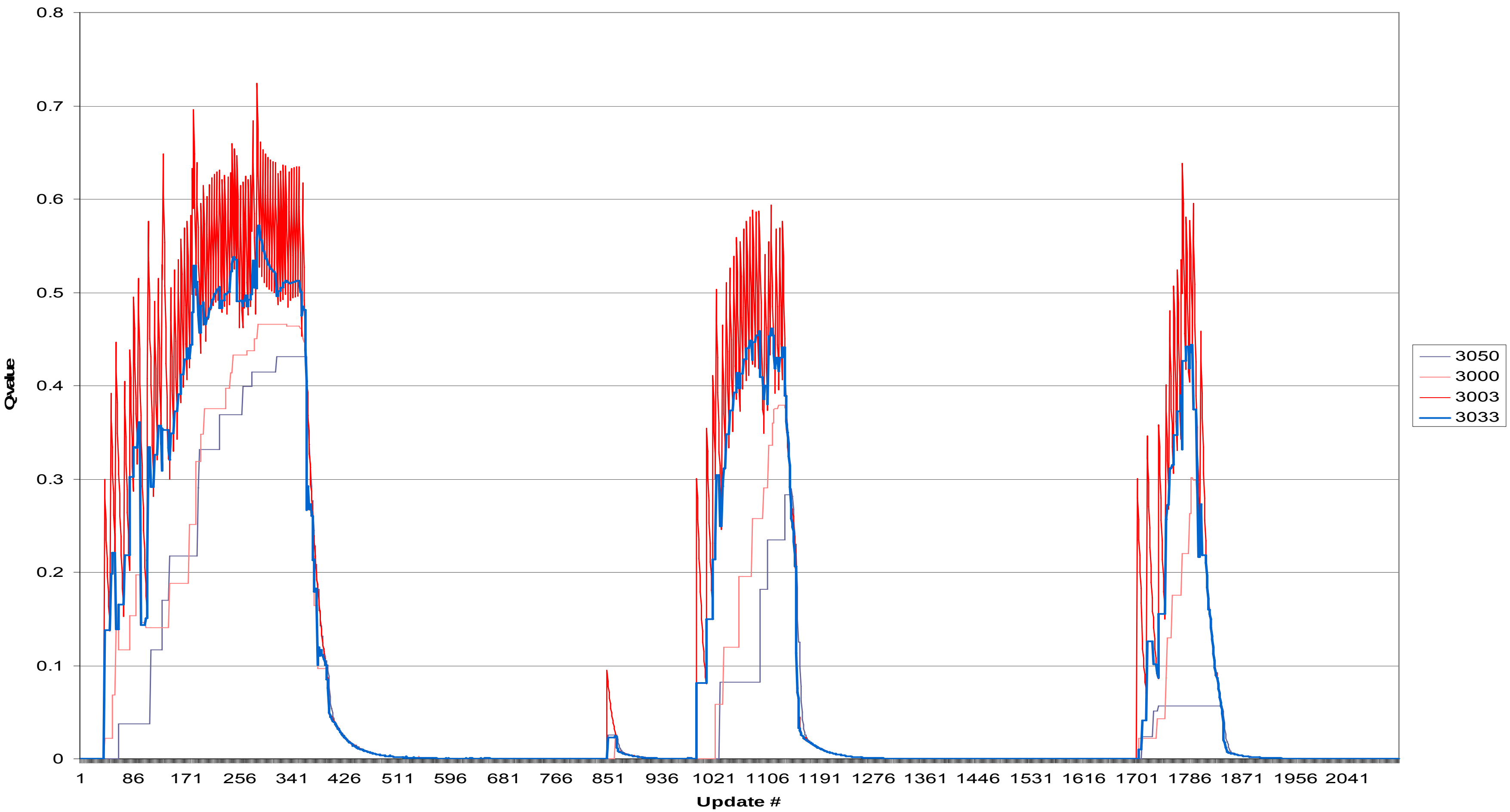
Predicted Q-values at the state (3,0)



Goal achieved

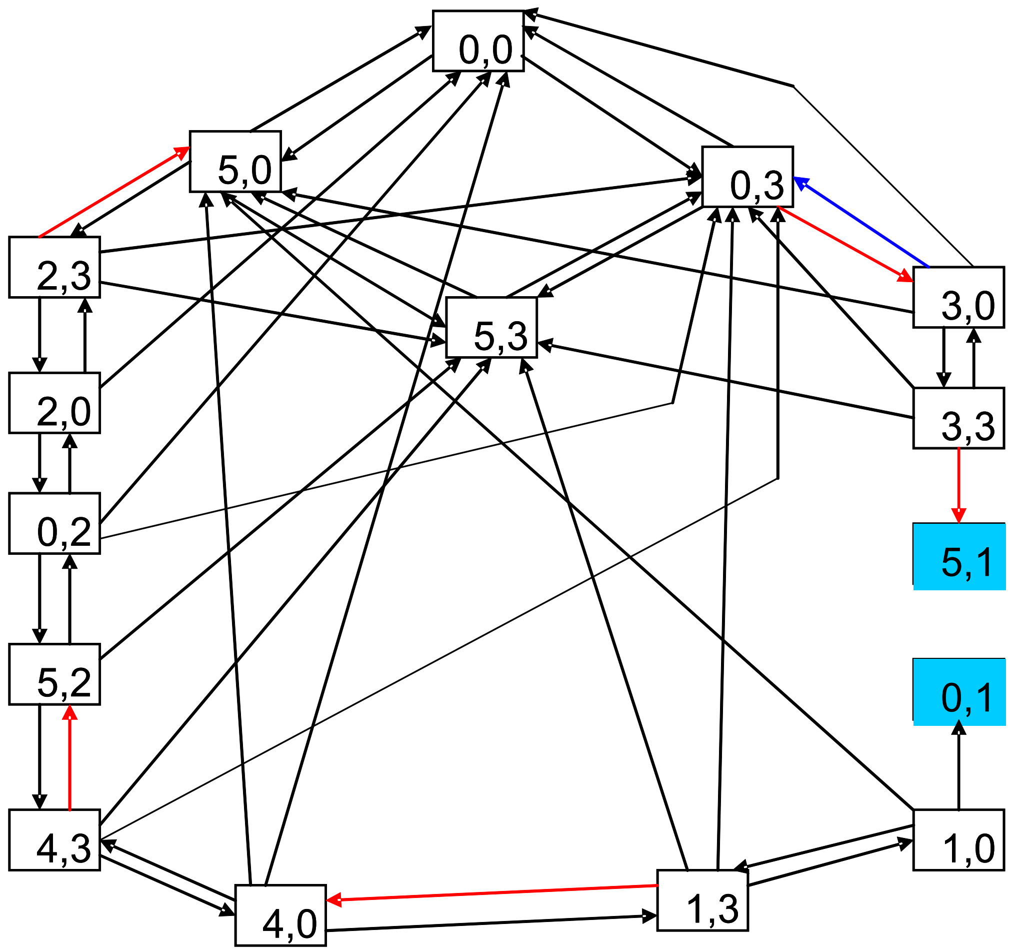
Goal achieved

Predicted Q-value for state (3,0)



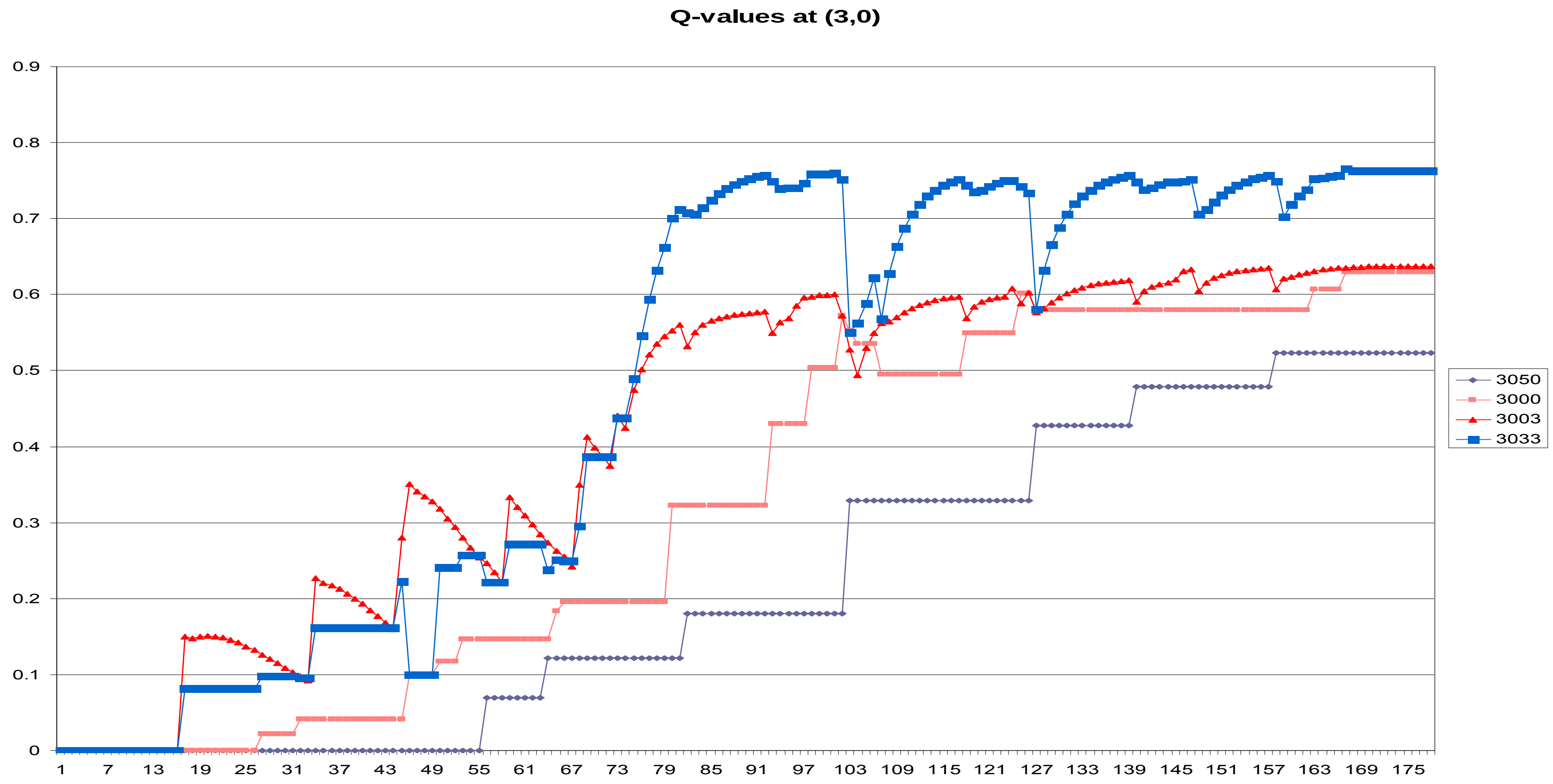
How to fix – Add following rule

- If there is a jug <u>*with volume 3*</u> and contents 3, pour this jug into the other jug.



How to fix – Add following rule

- If there is a jug ***with volume 3*** and contents 3, pour this jug into the other jug.



Designing a specialization procedure

1. How to decide whether to specialize a given rule.
2. Given that we have chosen to specialize a rule, what conditions should we add to the rule?
3. (optional) In what, if any, cases should a rule be eliminated?

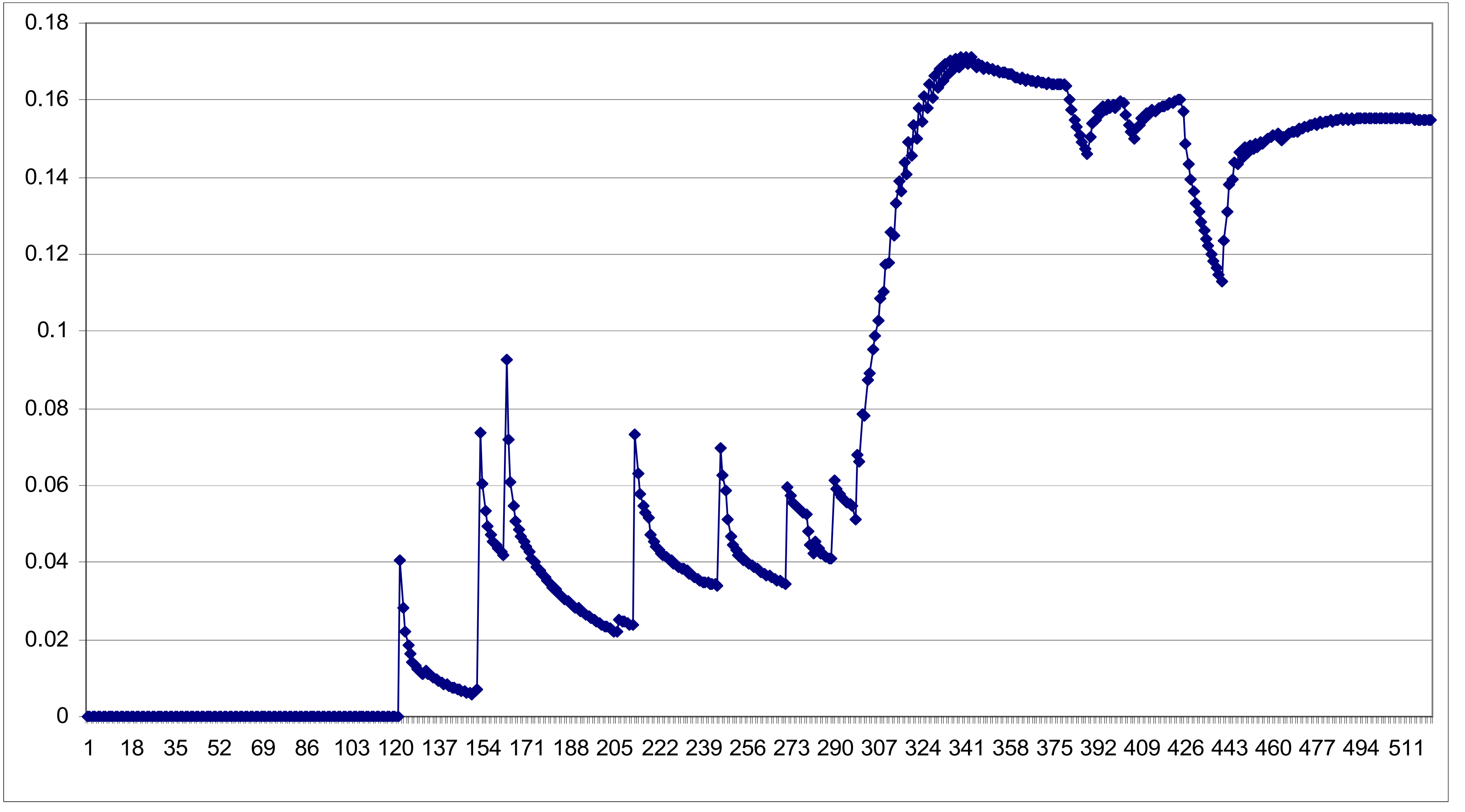
Question 2 (What conditions to add to a rule) – Proposed Answer

- Trying an activation-based scheme.
  - When an (instantiated) rule R decides to specialize, it finds the most activated WME, w = (ID ATTR VALUE).
  - Traces upward through WM, to find a shortest path from ID to some identifier in the rule's instantiation
  - w and the WMEs in the trace add themselves to the conditions in R to form a new rule R'
  - If R' is not a duplicate of some existing rule, R' is added to the Rete (without removing R).

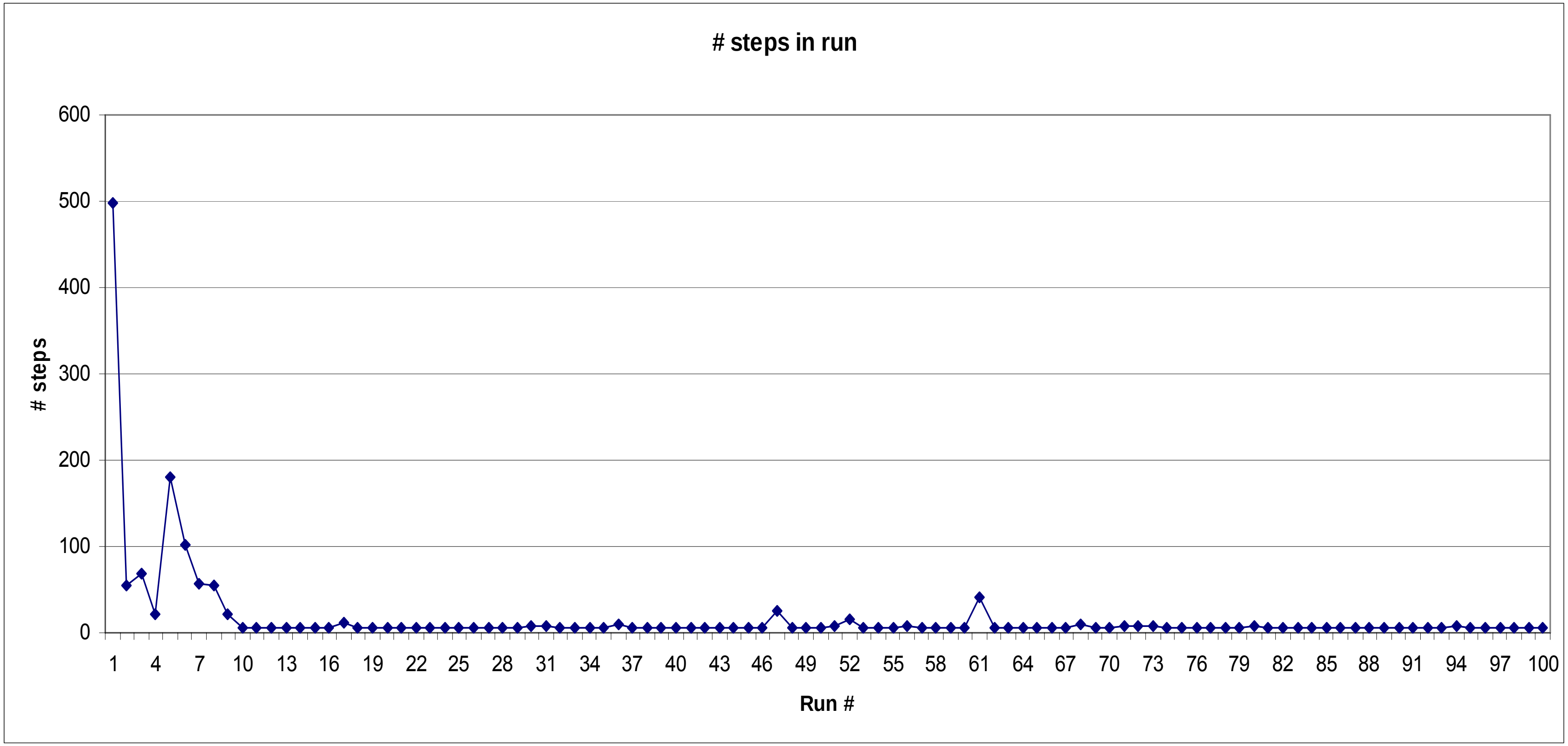
Question 1 (Should a given rule be specialized) – Proposed answer

- Track weights (numeric preferences) of rules.
- Weights should converge when rules sufficient, so stop specializing when weights stop moving.

Tracking weights of rule RL-3

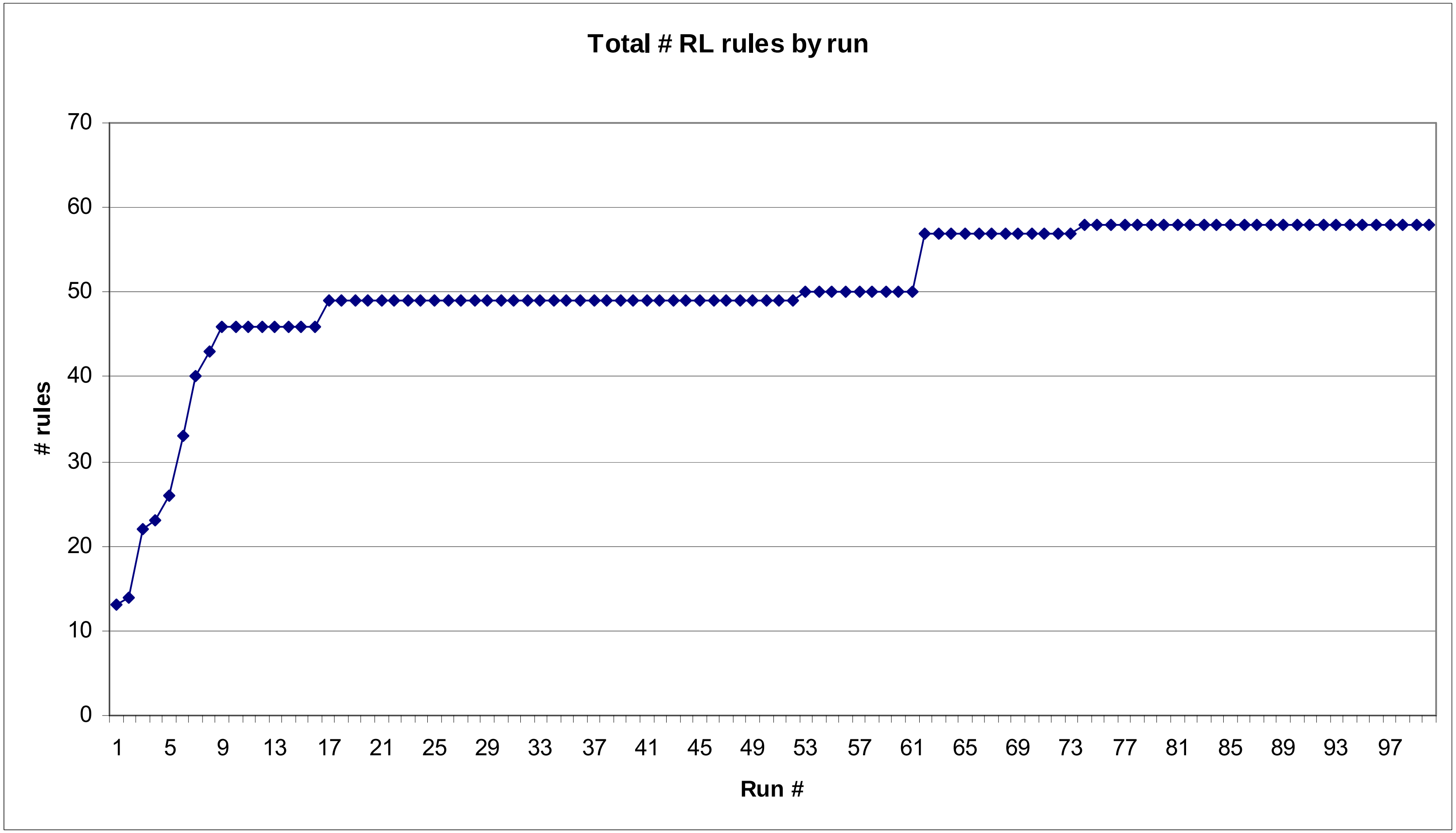


steps in run



steps in run
600
500
400
300
200
100
0
steps
1
4
7
10
13
16
19
22
25
28
31
34
37
40
43
46
49
52
55
58
61
64
67
70
73
76
79
82
85
88
91
94
97
100
Run #

rules



Total # RL rules by run
70
60
50
40
30
20
10
0
rules
1
5
9
13
17
21
25
29
33
37
41
45
49
53
57
61
65
69
73
77
81
85
89
93
97
Run #

Conclusions

- Nuggets – Did work (at least on Waterjug) That is, came to follow the optimal policy.
- Coal – Makes too many rules
  1. Specializes rules that don’t need specialization.
  2. Specializations not always useful, since chosen heuristically
  3. Will try to explain non-determinism by making rules.