Heuristic Value Function Revision

Mitchell Keith Bloch and John Edwin Laird

University of Michigan
2260 Hayward Street
Ann Arbor, MI. 48109-2121
bazald@umich.edu and laird@umich.edu

June 20, 2012

Motivation

- Possible to specify an arbitrary value function in Soar
- No way to revise an existing value function because reinforcement learning always make a decision
- **Given the opportunity, it may be possible to improve a value function as specified by RL-rules**

Reinforcement Learning

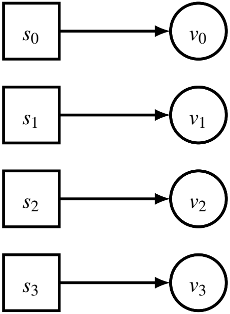
- Prefer actions leading to positive rewards to actions leading to negative rewards
- Outcomes are characterized as a discounted return, $\sum_{t=0}^{\infty} \gamma^t r_t$
- Deriving correct estimates of these returns is integral to many RL algorithms
  - What is essential, however, is learning an optimal policy
- Q-learning and Sarsa in the simplest case map $\mathcal{S} \times \mathcal{A} \Rightarrow \mathcal{Q}$ in a one-to-one fashion

Soar-RL

- Conditions on RL-rules encode which features to test and how to discretizate continuous state, defining the mapping $\mathcal{S} \times \mathcal{A} \Rightarrow \mathcal{Q}$

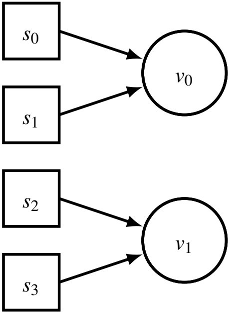
Soar-RL

- Conditions on RL-rules encode which features to test and how to discretizate continuous state, defining the mapping $\mathcal{S} \times \mathcal{A} \Rightarrow \mathcal{Q}$
  - Can be one-to-one (if no continuous space)



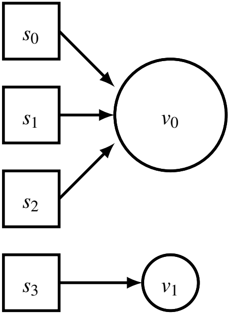
Soar-RL

- Conditions on RL-rules encode which features to test and how to discretizate continuous state, defining the mapping $\mathcal{S} \times \mathcal{A} \Rightarrow \mathcal{Q}$
  - Can be one-to-one (if no continuous space)
  - Can use coarse coding



Soar-RL

- Conditions on RL-rules encode which features to test and how to discretizate continuous state, defining the mapping $\mathcal{S} \times \mathcal{A} \Rightarrow \mathcal{Q}$
  - Can be one-to-one (if no continuous space)
  - Can use coarse coding
  - Potentially arbitrary, non-uniform abstraction



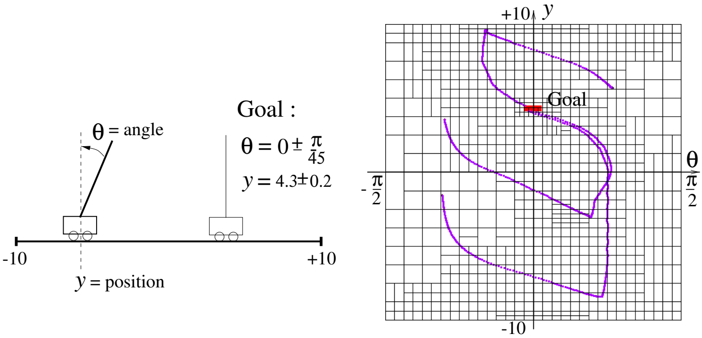
Soar-RL

- Conditions on RL-rules encode which features to test and how to discretizate continuous state, defining the mapping $\mathcal{S} \times \mathcal{A} \Rightarrow \mathcal{Q}$
  - Can be one-to-one (if no continuous space)
  - Can use coarse coding
  - Potentially arbitrary, non-uniform abstraction
- Traditionally bootstrapped from values set before execution, e.g. 0
  - Can be done simply with GPs or templates
  - Work in John's talk uses chunking to take advantage of background knowledge instead, deciding ...
    - The mapping $\mathcal{S} \times \mathcal{A} \Rightarrow \mathcal{Q}$
    - Initial Q-values

Decide

1. Reduce candidate set using non-numeric preferences
   - Possible to impasse here
2. Decide using numeric preferences (RL-rules)
   - Always results in a decision (will never impasse)
   - Cannot chunk new RL-rules to modify $\mathcal{S} \times \mathcal{A} \Rightarrow \mathcal{Q}$
     - Prevents using overgeneral conditions early on to promote quick learning
     - Prevents adding conditions on relevant features which were previously believed to be irrelevant

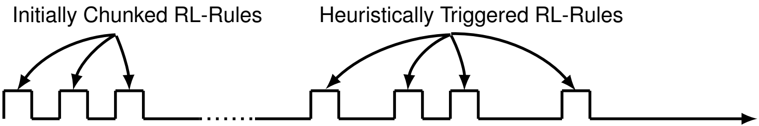
cart pole



(a) The ''Cart-pole'' (b) The projection of the state space

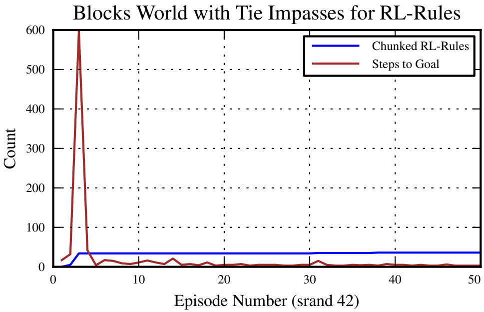
- [Munos and Moore, 2001] developed metadata to decide which Q-values ...
  - Might be important to split (influence)
  - Are good candidates for changing values (variance)

Design Goals



- Specify initial value function
  - Condition on features of clear importance
  - Err on side of overgenerality to speed learning
- Track metadata until they indicate an opportunity to improve the value function
- Generate additional RL-rules in tie impasses until metadata indicate improvement
  - Generally condition RL-rules on a smaller part of the state space

blocks world (preliminary)



- Start with creating one RL-rule per move (e.g. A onto B)
- Tie impasse when variance is above a low threshold, 0.002
- Add RL-rules testing features (in-place, on-top)
- Achieved optimal consistently by 50 episodes, ignoring exploration

When Tie Impasses Occur

- Operators without numeric preferences can tie
  - Only acceptable preferences $\rightarrow$ tie impasse
  - Multiple best, no better or worse preferences $\rightarrow$ tie impasse

  ⋮
- Operators with numeric preferences (RL-rules) never tie
  - A somewhat random choice is always made
  - Of course, we can change this

Enabling Tie Impasses for RL-Rules

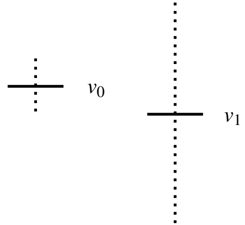


Figure: Depiction of Q-values, v_1 having high variance.

- Must track metadata which summarize experience on which a decision can be based
  - Values have high variance
  - Values have high influence
  - *Other metrics...?*

Build a Tie Impasse for RL-Rules

- Add subset of `^numeric (^tied <o> ^improve <o>)` parallel to `^item <o>` in the impasse state
  - `^tied` indicates that the operator is involved in the tie
  - `^improve` indicates that the operator needs a new preference to resolve the tie
  - Metadata may be exposed under `^numeric` in future work, allowing the agent to reason about which preferences could resolve the impasse

Resolve Tie Impasse

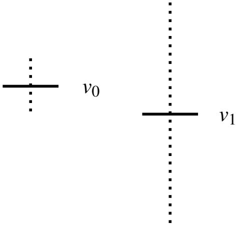


Figure: Depiction of Q-values, v_1 having high variance.

- Determine which preference(s) will resolve the impasse
  - The expected case is one RL-rule per operator
  - Current work just adds RL-rules with the value 0

Resolve Tie Impasse

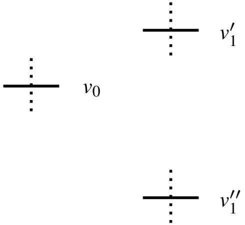


Figure: Depiction of Q-values, v_1 now separated in different states

- Rely on chunking to allow improvement over time
  - Test a more complete set of features in `blocks world`
  - Test a smaller region of continuous state in `cart pole`

Nuggets and Coal

Nuggets:

- Tie impasses for RL-rules are happening (in a branch)
- Using a *simple* tie-detection procedure, `blocks world` can converge
- Code can be written fairly generally using an extended problem space description

Coal:

- Not currently achieving good performance in `cart pole`
- Open questions about general tie-detection procedure
  - Must balance need for improved discretization with need for experience
  - Must be feasible to resolve ties with RL-rules, including $= 0$

Rémi Munos and Andrew Moore. Variable resolution discretization in optimal control. In *Machine Learning*, pages 291–323, 2001.