Relational Blocks World Experiments in Carli

Mitchell Keith Bloch

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

June 4, 2015

Carli $\neq$ Soar – https://github.com/bazald/carli

What's offered:

- A Soar-like execution cycle

Carli $\neq$ Soar – https://github.com/bazald/carli

What's offered:

- A Soar-like execution cycle

Meaning:

1. `^io.input-link`
2. ~~elaboration cycle~~
3. numeric preferences (and implicit operator proposal)
4. decide
    - ~~impasses~~
5. act

Carli $\neq$ Soar – https://github.com/bazald/carli

What's offered:

- A Soar-like execution cycle
- Soar-RL-like reinforcement learning support
- Architectural support for efficiently creating more specific RL-rules over time – a generative model for a value function

What's missing or different:

- Manipulating WMEs from the RHS has not been tested yet
- Operators (as you know them) and impasses do not exist
- SMem, EpMem, and SVS do not exist

Carli $\neq$ Soar – https://github.com/bazald/carli

What's offered:

- A Soar-like execution cycle
- Soar-RL-like reinforcement learning support
- Architectural support for efficiently creating more specific RL-rules over time – a generative model for a value function

What's missing or different:

- Manipulating WMEs from the RHS has not been tested yet
- Operators (as you know them) and impasses do not exist
- SMem, EpMem, and SVS do not exist

Reinforcement Learning, Part I of II

- Must learn how to act, given experience perceiving states, trying actions, and receiving rewards
- Explore with an ϵ-greedy exploration strategy
- At the most abstract:
  - $\pi(s, a)$ represents the target policy
  - $\phi(i)$ represents the set of possible features
  - $\theta(i)$ stores weights which sum to provide value estimates for different actions

Reinforcement Learning, Part II of II

- Learn using Sarsa(λ), Q(λ), or GQ(λ)
  - On-policy: Can maximize over the exploration policy
  - Off-policy: Or over the target—typically greedy—policy
- Actions can be compared using estimates of discounted return

$$\sum_{t=0}^{\infty} \texttt{discount\_rate}^{t} \cdot \texttt{reward}_{t}$$

Temporal Difference Methods

Briefly:

- On-policy—Sarsa: $Q(s,a) \xleftarrow{\alpha} r + \gamma Q(s',a')$
- Off-policy—Q-learning: $Q(s,a) \xleftarrow{\alpha} r + \gamma \max_{a^*} Q(s',a^*)$
- Modern—GQ(λ): More elaborate

Listen to my next talk!

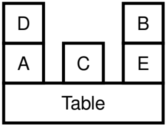
Relational Reinforcement Learning

- Each state is described by a set of relations, such as `(<stack> ^top <block>)`
- Each feature in $\phi(i)$ represents a conjunction of any number of such relations
- Value function computation could dominate CPU time since variable bindings are expensive
- The Rete algorithm can be used
  - It was designed for expert system rules
  - Handles variable bindings very efficiently
  - CPU time proportional to changes in environment rather than the total size of the environment
  - Shares work between similar rules

Dynamic Specialization

- Given $\phi(i)$, $\theta(i)$, and other metadata, which features are most likely to improve the value function?
- Many approaches have been explored
- We've explored the following criteria:
  - Cumulative Absolute Temporal Difference Error
  - Policy –Maximal change in $\pi(s, a)$
  - Value – Maximal change in $\theta(i)$

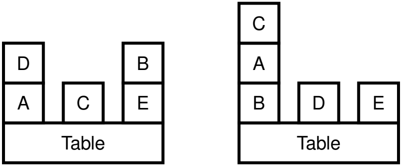
Typical Relational Blocks World



Typical state description visual

- No direct knowledge of the goal presented by the environment
- All knowledge comes from the reward function
- Only simple training goals possible for variable configurations
  - Place all blocks on the table
  - Place one specific block on one other block
  - Create a tower of a certain height

My Relational Blocks World



Blocks Goal

Complete state description visual

- Full representation of the goal presented by the environment
- Significantly more complex training goal
  - Must test more than one relation

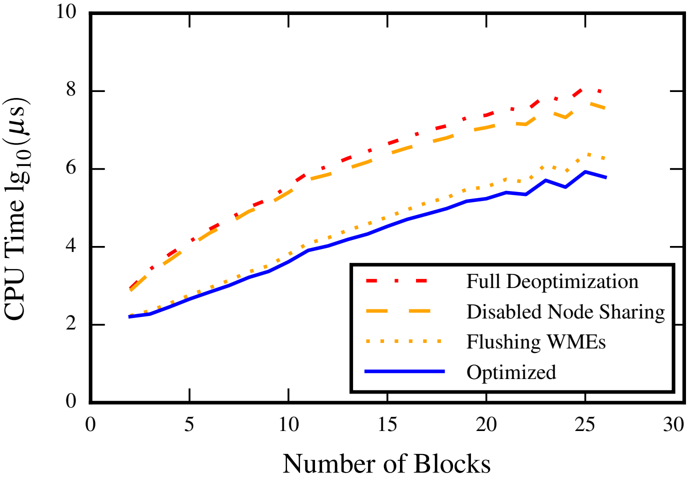
A Carli Agent Rule – https://github.com/bazald/carli

```
sp {blocks-world*rl-fringe*s38
  :feature 3 split blocks-world*rl-fringe*s16
  (<s> ^blocks <blocks>)
  (<s> ^goal <goal>)
  ⋮                                # Rule abbreviated
  -{(<goal> ^stack <goal-stack>)
    (<stack> ^matches <goal-stack>)}
  +{(<goal> ^stack <goal-stack>)
    (<dest-stack> ^matches <goal-stack>)}
-->
  = 0.32900469057018422217
}
```

A Carli Agent

- Executes quickly, using a rete implementation for its value function
- Learns using the TD methods we described earlier
- Tackles the problem of feature selection
  - Which conditions to add to new RL-rules, i.e.
    ```
    +{(<goal> ^stack <goal-stack>)
      (<dest-stack> ^matches <goal-stack>)}
    ```
- Efficiently adds new rules to the rete using the chosen conditions

Results – Rete Scaling for a Value Function

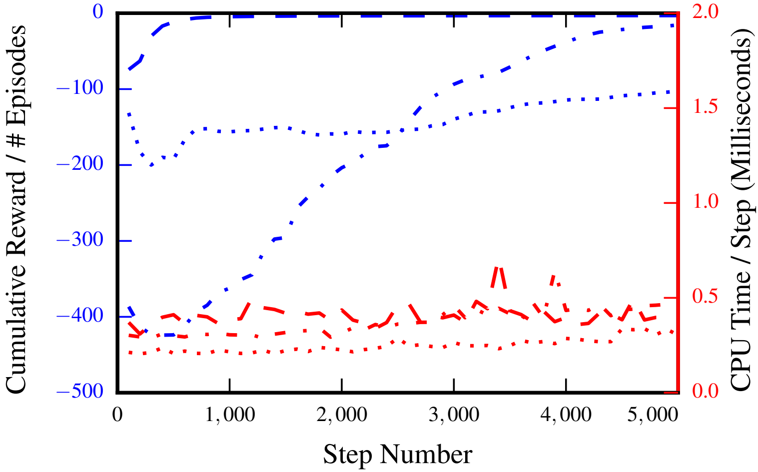


Results – Scalability Discussion

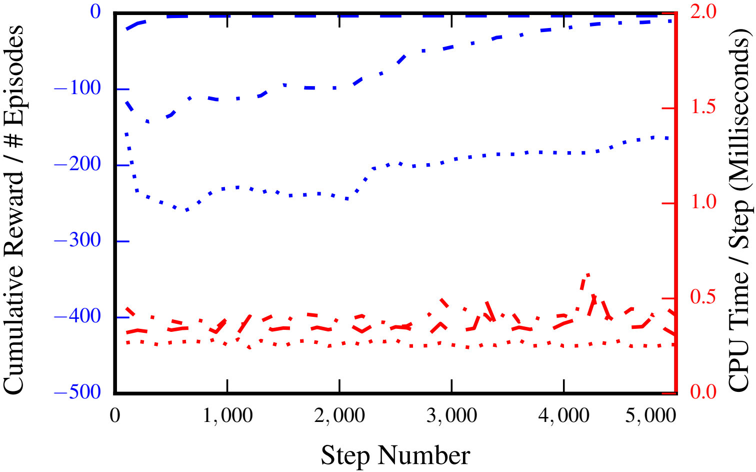
Using a learned policy:

- Test scalability of the Rete when reasoning over complex relations
- The deoptimized Rete takes 100 seconds per move at 26 blocks
- The optimized Rete takes only 1 second per move at 26 blocks
  - 16 blocks is the cutoff for reasoning in 50 ms
  - With 10 blocks, 100 moves take half a second
- This is quite fast, and it's actually a degenerate, bad case for Rete
  - Multivalued block and stack attributes cause exponential explosions

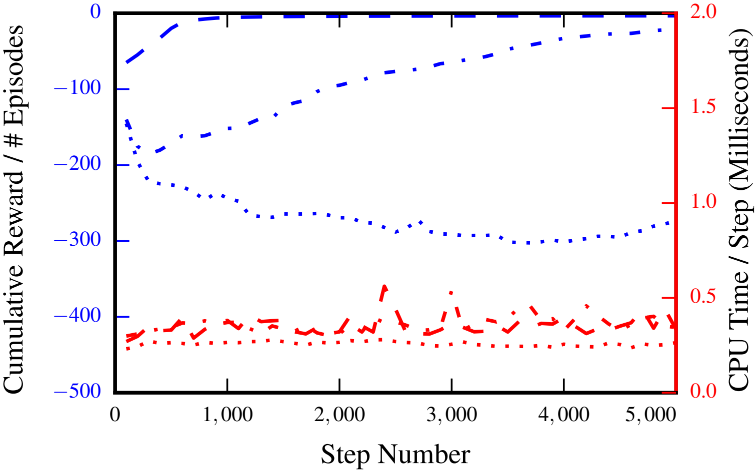
Results – Flat / Non-Hierarchical



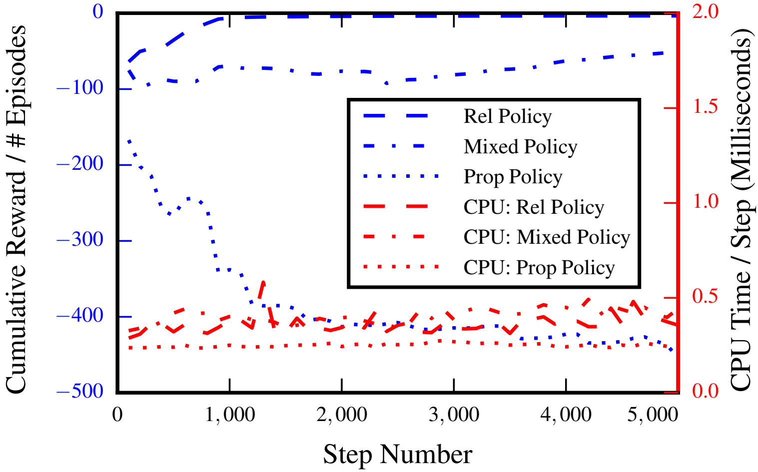
Results – Full Hierarchy



Results – Value Criterion



Results – Policy Criterion



Results – Learning Discussion

- We test the learning ability of our system
  - With only inadequate propositional features
  - With only good relational features
  - With a mix of both
- With only good relational features, all agents succeed quickly
- Propositional features distract the agents to a degree, but all recover
  - The flat agent handles the distractors the least well

Nuggets and Coal

Nuggets:

- Rete enables RRL agents to solve tasks quickly
- Dynamically specializing a value function has a neglible CPU cost, and the resulting suboptimality in the policy is temporary
- We have developed and implemented a rule grammar to specify dynamically specializable relational reinforcement learning agents

Coal:

- Could still improve our feature selection criteria
- Haven't yet implemented sophisticated restructuring of the value function
- A higher order grammar for adding variables and new relations using these variables would be helpful
- Not a part of Soar