UNIVERSITY OF MICHIGAN

Carli—Efficient Value Function Specialization

Mitchell Keith Bloch and John Edwin Laird
The Soar Group at the University of Michigan

Reinforcement Learning

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

Relational RL

- Each state is described by a set of relations, such as `(<stack> ^top <block>)`
- Each feature in $\phi(\boldsymbol{i})$ 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

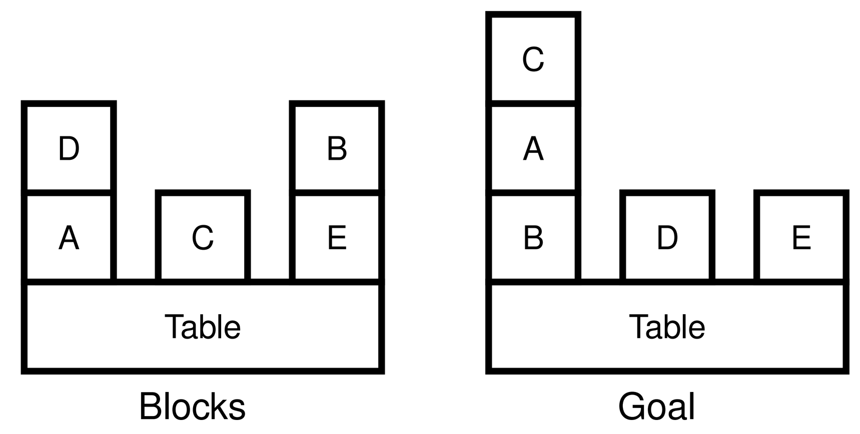
Dynamic Specialization

- Given $\phi(\boldsymbol{i})$, $\theta(\boldsymbol{i})$, 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 $\pi(\boldsymbol{s}, \boldsymbol{a})$
  - Value – Maximal change in $\theta(\boldsymbol{i})$

Rule Grammar

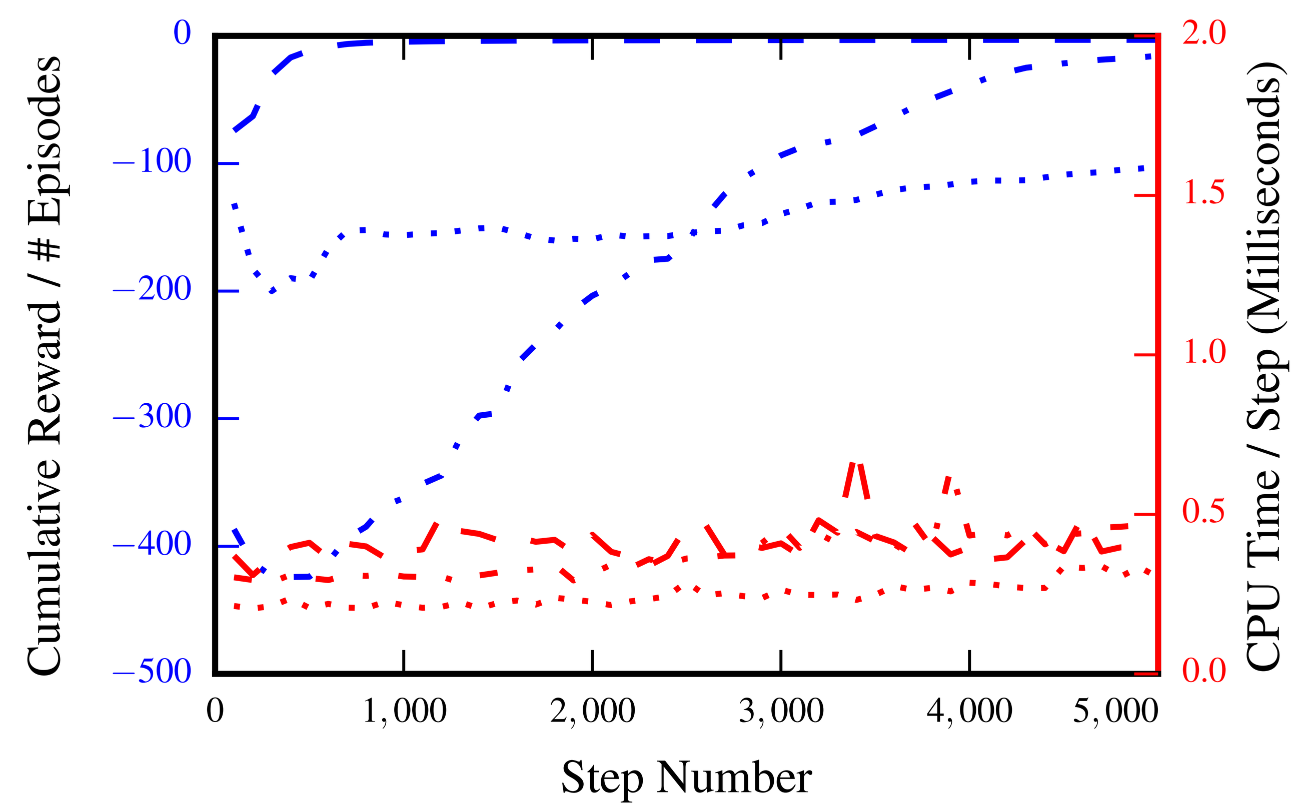
- In supporting dynamic specialization, we've constructed and implemented a rule grammar that maps onto both $\phi(\boldsymbol{i})$ and the Rete
- Carli can load $\phi(\boldsymbol{i})$ and $\theta(\boldsymbol{i})$ from rules,
- Modify the Rete as $\phi(\boldsymbol{i})$ grows, and
- Write $\phi(\boldsymbol{i})$ and $\theta(\boldsymbol{i})$ back out to disk
- The grammar is specified using flex and bison
  - It additionally supports continuous features
  - And refinement of continuous features
  - Despecialization is implemented but unused

Blocks World

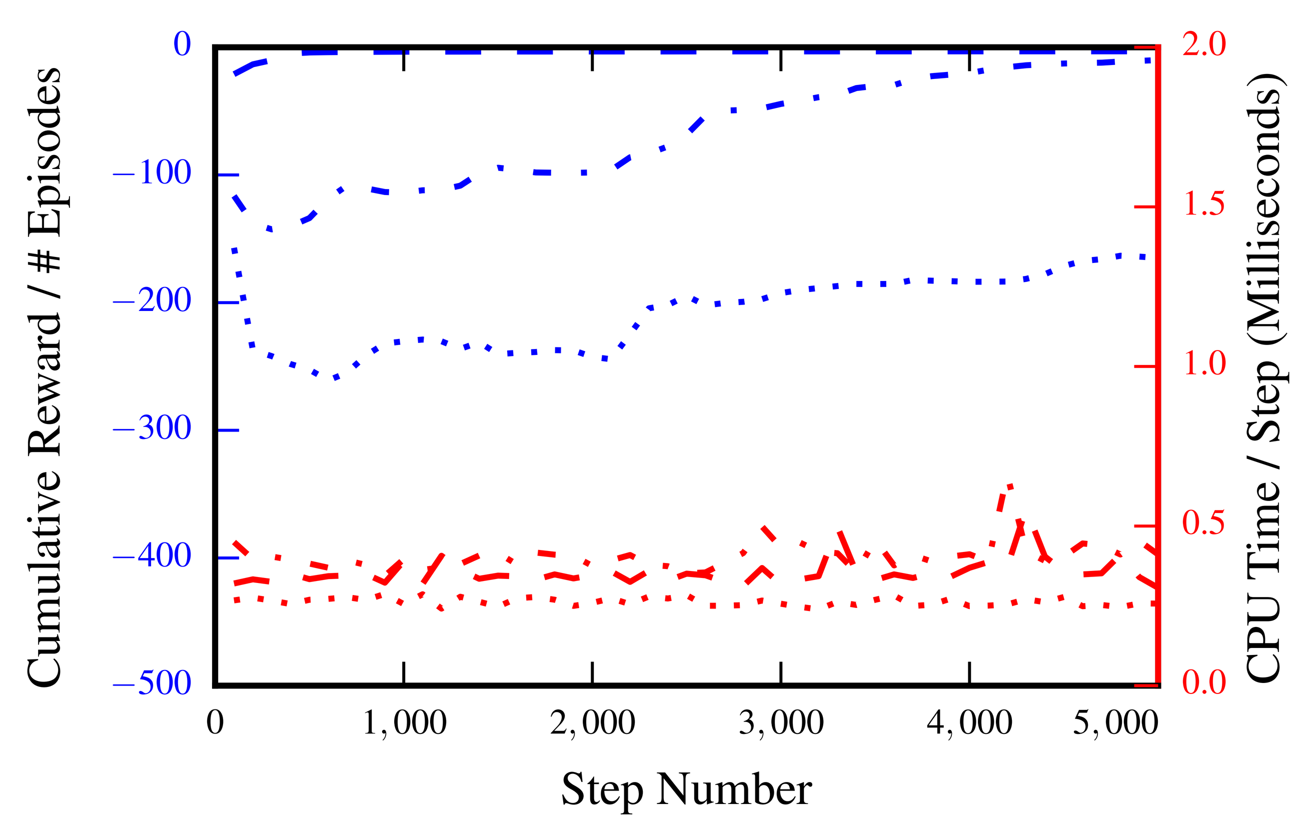


- Given a goal configuration, rearrange the blocks to match it
- Often best treated as a planning problem
- Relational RL can manage with unbounded table space
  - Do stacks match goal stacks (up to their height)?
  - Do moves cause or interfere with matches?
  - Is the destination the table?

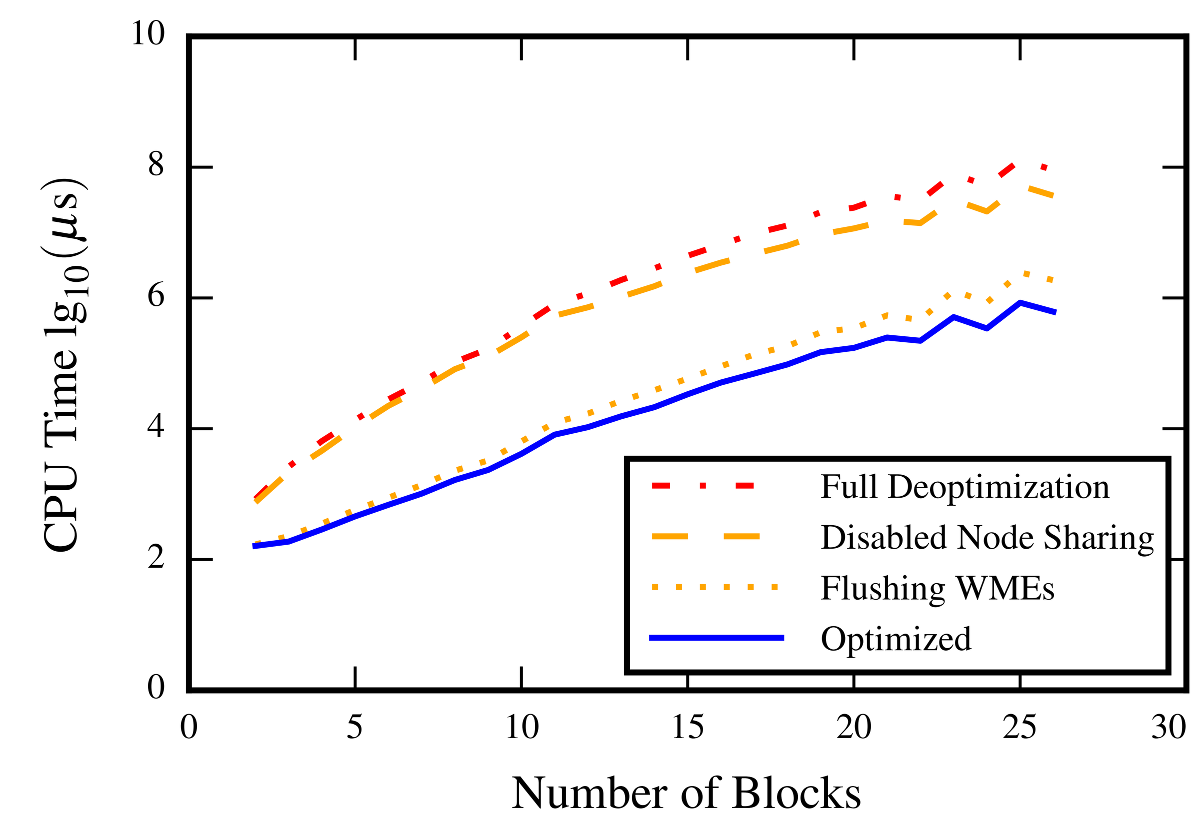
Flat / Non-Hierarchical RRL



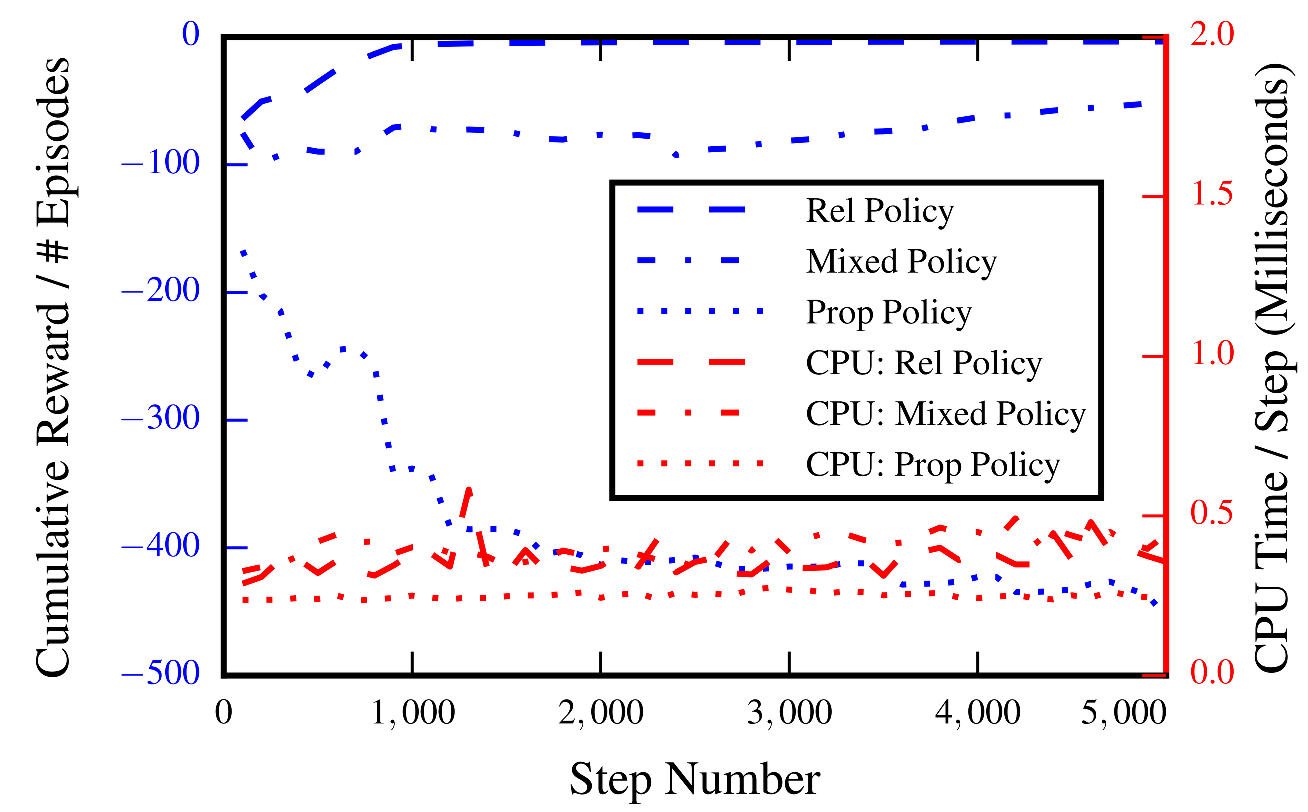
Full Hierarchy for RRL



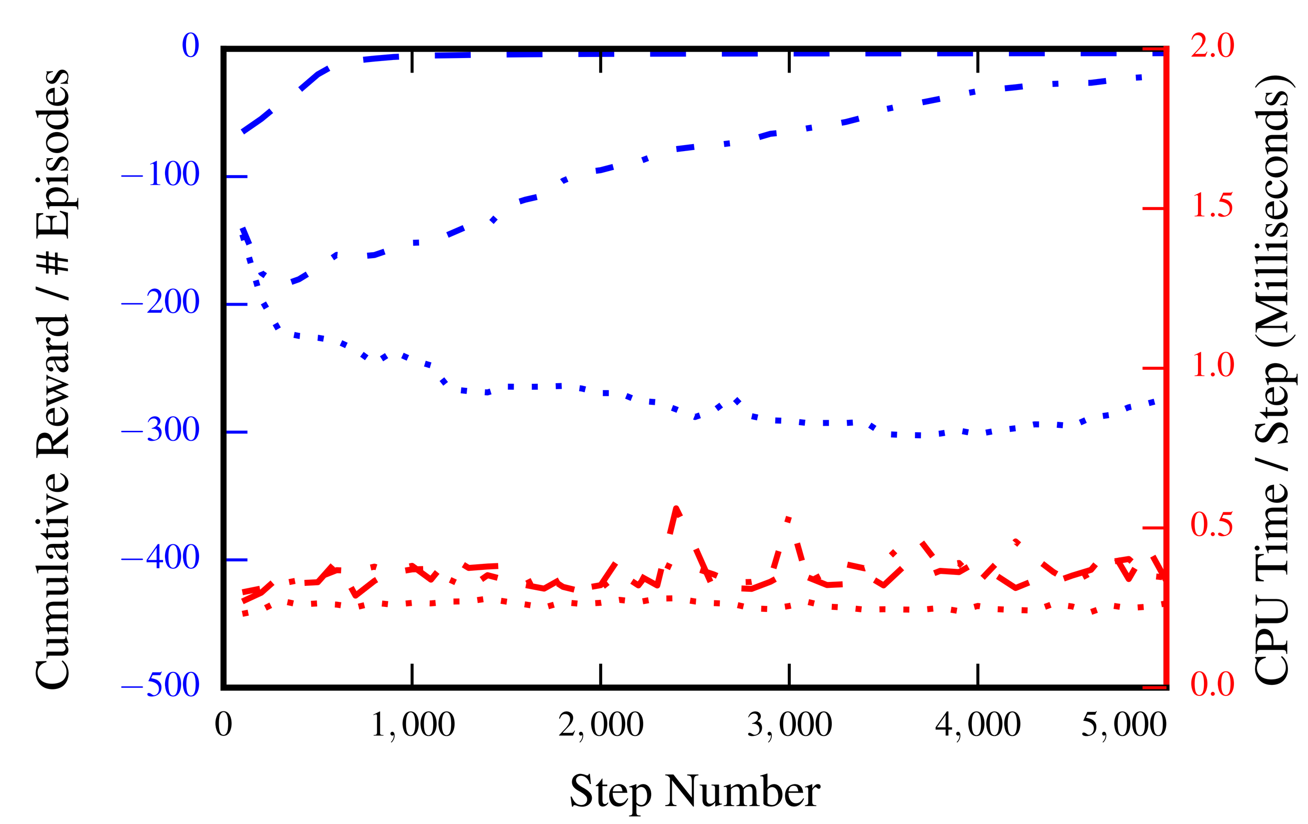
Rete Scaling for Blocks World Value Function



Dynamically Specialized Hierarchy, Policy Criterion



Dynamically Specialized Hierarchy, Value Criterion



Scalability Experiments

Using a learned policy:

- We 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

Learning Experiments

- 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

Contributions

- 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

Future Work

- Develop an effective agent for Infinite Mario
- Improve our feature selection criteria
- Sophisticated restructuring of the value function
  - Despecialization when features aren't useful
  - Swizzling the value function when learning might be more stable with ordering
- A higher order grammar for adding variables and new relations using these variables

http://bazald.com/publications/ & http://github.com/bazald/carli
{bazald,laird}@umich.edu