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 Q(λ)
- At the most abstract:
  - π(s, a) represents the target policy
  - φ(λ) represents the set of possible features
  - θ(λ) stores weights which sum to provide value estimates for different actions

Relational RL

- Each state is described by a set of relations, such as (state, top < block)
- Each feature in φ(λ) 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 φ(λ), θ(λ), 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 π(s, a)
  - Value — Maximal change in θ(λ)

Rule Grammar

- In supporting dynamic specialization, we've constructed and implemented a rule grammar that maps onto both φ(λ) and the Rete
- Carli can load φ(λ) and θ(λ) from rules,
- Modify the Rete as φ(λ) grows, and
- Write φ(λ) and θ(λ) 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 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

- Dynamically specializing a value function has a degree, but all recover
- Rete enables RRL agents to solve tasks quickly
- We test scalability of the Rete when reasoning dynamically

Dynamically Specialized Hierarchy, Policy Criterion

- 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

Dynamically Specialized Hierarchy, Value Criterion

- 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

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

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

Evaluating Agents

- 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

Contributions

- Rete enables RRL agents to solve tasks quickly
- Dynamically specializing a value function has a negligible 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

Flat / Non-Hierarchical RRL

- 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

Dynamically Specialized Hierarchy, Policy Criterion

- 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

Dynamically Specialized Hierarchy, Value Criterion

- 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

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

---


{@umich.edu}