Deciding to Specialize and Respecialize a Value Function for Relational Reinforcement Learning

Mitchell Keith Bloch
University of Michigan
Ann Arbor, MI, USA
bazald@umich.edu

John Edwin Laird
University of Michigan
Ann Arbor, MI, USA
laird@umich.edu

Abstract

We investigate the matter of feature selection in the context of relational reinforcement learning. We had previously hypothesized that it is more efficient to specialize a value function quickly, making specializations that are potentially suboptimal as a result, and to later modify that value function in the event that the agent gets it "wrong." Here we introduce agents with the ability to adjust their generalization through respecialization criteria. These agents continuously reevaluate the feature selection problem to see if they should change how they have structured their value functions as they gain more experience. We present performance and computational cost data for these agents and demonstrate that they can do better than the agents with no ability to revisit the feature selection problem.

Keywords: relational reinforcement learning, feature selection

1 Introduction

Bloch and Laird [2015] demonstrated the computational and learning efficiency of the Rete algorithm in the context of relational reinforcement learning (RRL) and explored the quality of learning under several feature selection criteria. This prior work involved feature selection criteria that allowed agents to decide when and how to specialize the value function. It provided no mechanisms for evaluating the quality of these value function specializations after the fact, so any decisions that were made were then final.

They hypothesized that it is more efficient to specialize the value function quickly, making specializations that are potentially suboptimal as a result, and to later modify the value function in the event that the agent gets it "wrong." We now provide our agents with criteria to allow them to respecialize the value function using different features at a given level of generality. We present these results in section 3.

2 Background

We're concerned with the version of the reinforcement learning problem defined as a minimization of cumulative regret over the course of learning. The execution cycle for one of our agents is similar to many reinforcement learning agents learning with temporal difference methods. The agent chooses an exploratory or greedy action, gets a reward from the resulting state transition, and then learns from the complete $<s, a, r, s', a'>$ tuple. Our agents learn online using the modern temporal difference method, GQ(λ) [Maei and Sutton, 2010]. GQ(λ) is a good choice because it supports online, incremental learning and provides convergence guarantees when using linear function approximation. Any other algorithm with the same properties would suffice.

2.1 Relational Blocks World with Goal Configuration

There are many formulations of the classic Blocks World domain. Blocks World typically presents a configuration of blocks, each with a unique label. The agent is tasked with moving one block at a time from the top of a stack of blocks to either the top of a different stack of blocks or to the table. The agent completes the task when the blocks' configuration matches the goal configuration.

The Blocks World task that interests us exposes the goal configuration to the agent, as depicted in figure 1. Distractor features are included to ensure that the best answer to the feature selection problem is not simply to include all features of the environment. Agents using a standard, fixed propositional representation of this formulation of the Blocks World task cannot learn a near optimal strategy for solving new goal configurations because they must ultimately capture some details of the task specific to a particular goal configuration in order to function at all. Agents using relational representations can capture connections between different features of the environment in a more general way, allowing their policies to generalize between different initial conditions, goal configurations, and even numbers of blocks. See section 2.2 for details.

The relational features available to our agents are: whether the source stack of blocks matches a goal stack or not; whether the destination stack matches a goal stack or not; whether the top block of the source stack belongs on top of the destination stack or not (a top block being considered to match the top of the destination stack if and only if the destination stack matches a goal stack and the additional block will allow it to continue to match); and whether the destination (stack) is the table. The distrator features available to our agents include the names of blocks; the heights both of the source stack and of the destination stack; a randomized brightness or luminosity value for the blocks on top of both the source stack and the destination stack; and a related value as to whether that block is considered to be glowing or not.

2.2 Relational Reinforcement Learning

Relational reinforcement learning is an approach to support generalization from experience [Tadepalli *et al.*, 2004; Džeroski *et al.*, 2001]. Given relations and conjunctions thereof, it is possible to condense both the state-space and action-space for certain tasks.

For a minimal one-feature example, if the agent is concerned that a destination stack of blocks in the environment matches a stack of blocks in the goal configuration, that feature can expressed relationally as: $(dest\text{-}stack,$ `matches`, $goal\text{-}stack)$. Given this formulation, the agent can learn about all actions in which that match is present, regardless of the details of the specific labels of any of the blocks in either the $dest\text{-}stack$ or the $goal\text{-}stack$.

Figure 1: Visual representation of both the blocks the agent sees in the world and the goal configuration provided to the agent.

An agent could instead consider a feature expressing whether a block belongs on top of a destination stack of blocks in the environment: (*block*, matches-top, *dest-stack*). This relation again expresses a general concept and does not capture the details of the specific labels of either the block, *block*, nor of any of the blocks in the *goal-stack*.

If both features are important, an agent ought to consider the compound features derived from the conjunction of both (*dest-stack*, matches, *goal-stack*) and (*block*, matches-top, *dest-stack*). This is where the real power of relational reinforcement learning comes in. Not only are both features general due to the use of variables in the place of what would otherwise be specific details of the environment, but the value function ensures that identical variables in both features match. Therefore, even though specifics of the environment are not captured by the value function, it can guarantee that both conditions refer to the same *dest-stack* in this compound feature.

In the above example, given a conjunction of two Boolean features, four different possibilities will result. As features are combined, a combinatorial explosion can occur. However, as these features do not always depend on the size of the state-space and action-space, they can be particularly effective for scenarios with arbitrarily large state-spaces. As relational reinforcement learning allows an agent to reason about objects more abstractly, it additionally allows the agent to generalize from its experience to solve problems from outside its training set. If we were to propositionalize the set of features to allow it to learn this particular task, it would need complex conjunctions of many features to be filled out with every possible label in the environment, resulting in a similar combinatorial explosion but with none of the capability for generalization.

2.3 Value Functions

As in previous work, we use a value function using both linear function approximation and a dynamically specialized tile coding [Bloch and Laird, 2013; Bloch and Laird, 2015].

Linear function approximation contributes to the capability of our agents to generalize from past experience. Given n basis functions, $\phi_1, \ldots, \phi_n$, the Q-function is estimated by $Q(s,a) = \sum_{i=1}^{n} \phi_i(s,a) w_i$, where $\phi_i(s,a)$ is commonly 1 or 0 to indicate whether the feature is currently active for a given state and action, or not, and where the weight, w_i, represents the value contribution of the feature for the action under consideration, or a partial Q-value. That each feature applies to many states provides the agent with the ability to learn about states which the agent has yet to visit.

Starting with a coarse tile coding and breaking the tiles into smaller, more specific tiles over time also contributes to the capability of our agents to generalize from past experience. As the agent specializes its value function, it learns a more specialized policy, but without the full cost associated with starting with a maximally specialized value function from the beginning. We combine this with linear function approximation by keeping coarser, more general tiles in the system and allowing general learning to continue even as the smaller, more specialized tiles are added [Bloch and Laird, 2013].

3 Specializing and Respecializing

Our agents are trained on-policy with a discount rate of 0.9 and an eligibility trace decay rate of 0.3. We experimented with three value function specialization (feature selection) criteria: a cumulative absolute temporal difference error (CATDE) criterion, choosing features with the largest experienced cumulative temporal difference error; a policy criterion, choosing features that are most likely to result in changes to the policy based on estimated changes to the resulting value function; and a value criterion, choosing features that will result in the largest changes to the value estimates provided by the value function. Our policy and value criteria are directly inspired by those implemented by Whiteson *et al.* [2007]. To stress our learning algorithms, we provide them with two versions of several of the relational features: ones that provide correct information until step $5,000$ and then incorrect afterward; and ones that do the opposite. Initial block configurations and goal configurations of 4 blocks are randomly generated for each episode. Figure 2 depicts performance averages for the agents learning over 30 independent runs.

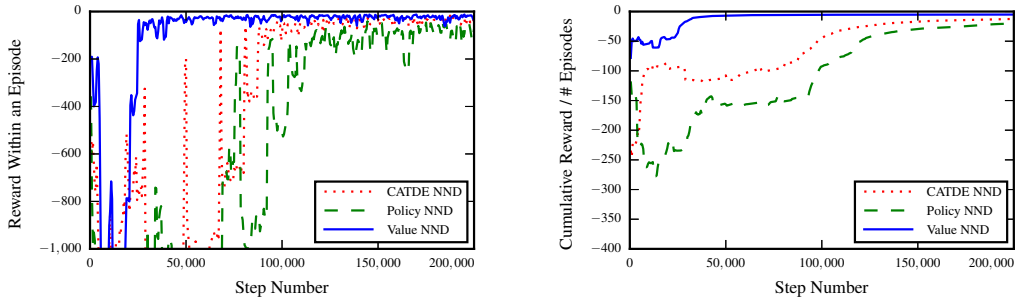
Bloch and Laird's [2015] hypothesis was that it is more efficient to specialize the value function quickly, making specializations that are potentially suboptimal as a result, and to later modify that value function in the event that the agent gets it "wrong." We expected this hypothesis to hold for all three criteria.

Figure 2a depicts the performance of agents that can only specialize the value function but are not allowed to change their minds about specializations they have made. It is evident that given that thrashing is impossible, the agents are eventually capable of learning good policies in the limit regardless of the choice of specialization criterion.

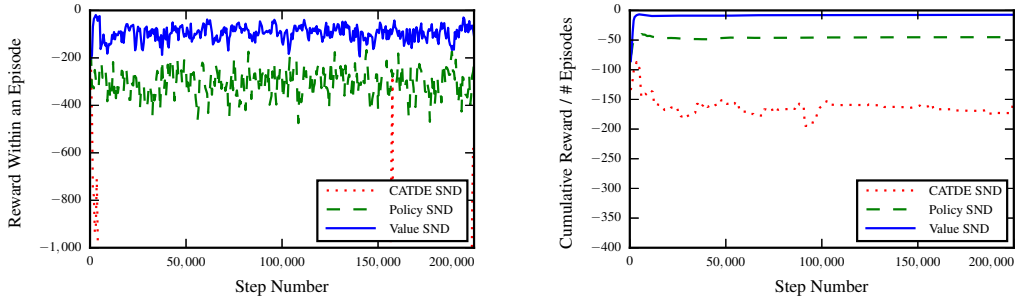
The first modification we make to our agents is to provide them with the ability to rerun their specialization criteria at each level of generalization of the value function at each time step. This allows our agents to compare metadata, policy decisions, and value estimates between the actual value function and versions of the value function in which each other specialization had been made instead, restricted to a one-step lookahead. These agents can then respecialize over a different set of features if it appears to be valuable to do so. Figure 2b depicts the performance of these agents. As these agents respecialize the value function with no mechanism in place to prevent thrashing, the agents using the policy and value criteria thrash to the extent that the value function stays very small. The agent using CATDE is comparably stable, but its computational cost is quite high as a result. It's worth noting

that the agents using the value criterion still manage to do quite well on average despite significant thrashing and get computational savings that our later efforts do not come close to matching. This result is highlighted in pink in figure 3.

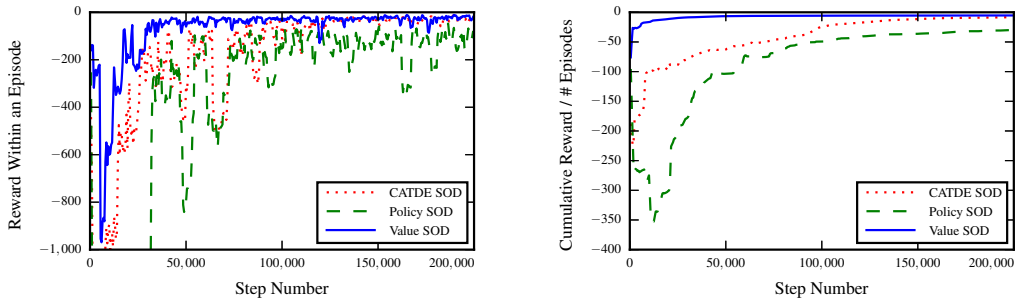
Our first attempt at a second modification of our agents is to implement a *blacklist* mechanism. This naive approach disallows an agent from choosing the same specialization twice at a given level of generalization (unless the value function is respecialized at an even more general level). The figure depicting the performance of the agents that additionally have this *blacklist* mechanism in place have been omitted because the performance is orders of magnitude worse than that of the agents with no respecialization. Thrashing likely results from the scenario in which a small number of choices are very good. Therefore, it is intuitive to expect blacklisting



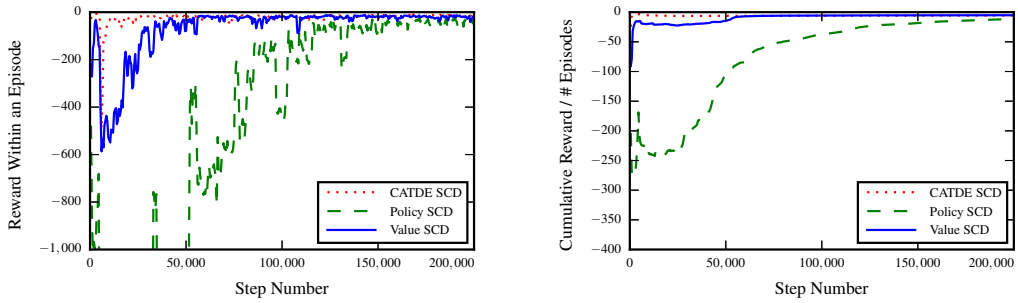
(a) No Despecialization



(b) Despecialization



(c) Despecialization + Boost



(d) Despecialization + Boost + Concrete

Figure 2: Plots of immediate reward per episode (left) and cumulative reward per episode (right). These are averages over 30 runs.

	No Respecialization	Respecialization	With Boost	With Boost+Concrete
CATDE	-12.5 | 13.8s	-158.3 | 27.4s	-8.3s | 44.8s	-5.5 | 11.0s
Policy	-20.2 | 18.0s	-45.1 | 1.2s	-30.0s | 96.6s	-11.5 | 11.0s
Value	-5.1 | 19.2s	-7.4 | 1.6s	-5.3s | 48.8s	-5.2 | 14.1s

Figure 3: Final cumulative reward vs average runtimes of our agents.

those choices to result in pathologically poor performance. However, should an agent run out of specializations to attempt, it can cease reevaluating other options, reducing computational overhead.

Our next attempt at a second modification of our agents is to implement a mechanism to boost the likelihood of an agent reselecting specializations that it has selected before. Each time it selects a specialization, it is given a boost that increases quadratically over time: $1 + 2 + \ldots$. This should cause an agent to eventually settle on one of its top choices. Figure 2c depicts the performance of agents that additionally have this *boost* mechanism in place. Given our intuition about why the *blacklist* mechanism was poor, we expected this to have better results, and this is clearly the case. The agents using the CATDE criterion converge better than the agents with no despecialization but at significant computational cost, as highlighted in cyan in figure 3. The *boost* mechanism is generally computationally inefficient since it successfully specializes the value function but never stops reevaluating its decisions even after it has effectively settled on them.

Our final modification is to address the computational overhead of the *boost* mechanism by disabling further attempts to reevaluate whether respecialization should occur at a given level of generality after 300 steps of stability. Figure 2d depicts the performance of agents that additionally have this *concrete* mechanism in place. The performance of these agents more consistently dominates that of the agents with no despecialization. We take this result as evidence supporting our original hypothesis. Additionally, as it reduces computational costs, it's of obvious practical value. These positive results are highlighted in green in figure 3. It's worth noting that while the agents using the value criterion benefit computationally and still do the best of the three criteria, the performance is nonetheless slightly worse than that of the original agents using the value criterion with no respecialization. The robust trend of effective learning using the value criterion would seem to indicate that specialization in order to achieve maximal spread of weights at each level of generality is reliably more effective than either attending to the error experienced by weights or by estimating the number of different actions in which a different policy decision would be made. Our best guess as to why the policy criterion performs poorly is that it is likely effective only when one-step lookahead provides a significant change to the capabilities of the agent.

4 Conclusion

This work implemented much of the future work suggested by Bloch and Laird [2015]. We provide evidence to support Bloch and Laird's [2015] hypothesis that it is more efficient to specialize the value function quickly, making specializations that are potentially suboptimal as a result, and to later modify that value function in the event that the agent gets it "wrong."

The number of distinct variables, however, remains fixed by the features initially provided to the system. Allowing future agents to reason about multiple objects of some type in the environment without needing to enumerate a fixed number of them ahead of time could still be a useful extension for other domains. It would however add another whole dimension to the feature selection problem, effectively resulting in the worst case value function complexity potentially matching environment complexity.

References

[Bloch and Laird, 2013] Mitchell Keith Bloch and John Edwin Laird. Online value function improvement. In *Reinforcement Learning and Decision Making 2013: Extended Abstracts*, pages 111–115, 2013.

[Bloch and Laird, 2015] Mitchell Keith Bloch and John Edwin Laird. The carli architecture–efficient value function specialization for relational reinforcement learning. In *Reinforcement Learning and Decision Making 2015: Extended Abstracts*, pages 44–47, 2015.

[Džeroski *et al.*, 2001] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine learning*, 43(1-2):7–52, 2001.

[Maei and Sutton, 2010] Hamid Reza Maei and Richard S Sutton. Gq (λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the Third Conference on Artificial General Intelligence*, volume 1, pages 91–96, 2010.

[Tadepalli *et al.*, 2004] Prasad Tadepalli, Robert Givan, and Kurt Driessens. Relational reinforcement learning: An overview. In *Proceedings of the ICML-2004 Workshop on Relational Reinforcement Learning*, pages 1–9, 2004.

[Whiteson *et al.*, 2007] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Adaptive tile coding for value function approximation, 2007.