Investigating the Soar-RL Implementation of the MAXQ Method for Hierarchical Reinforcement Learning

Nate Derbinsky
nlderbin@umich.edu
EECS 592 Term Project

December 11, 2007

1 Introduction

At a high level, this project has two driving forces: (I) analyzing the design and implementation of the Soar-RL system, and (II) investigating the effects of feature abstraction in a specific reinforcement learning (RL) problem. Before delving into the core of the research, let us examine these motivations.

1.1 Research Motivations

Discussed in greater detail below, Soar-RL is the integration of the reinforcement learning method of machine learning into Soar, a generalized architecture. The MAXQ method for hierarchical reinforcement learning [1] greatly influenced the design for the hierarchical reinforcement learning components of Soar-RL [2]. In its pre-release form, it is prudent to question the merits of this union: what, conceptually and computationally, have we gained and lost by implementing a highly optimized algorithm in a general architecture? Intuitively, abstracting a problem implementation carries a computational cost, in the form of increased time/space requirements. Additionally, when moving from the low-level control of a custom solution to an architectural paradigm, we may suffer from reduced ability to direct program behavior. However, modern programs are not typically written in assembler: abstraction has its benefits. Most pertinent, with abstraction comes the ability to quickly generate, tune, and explore relatively large numbers of problem instances. We dedicate a large portion of this project effort to comparing these tradeoffs in context of a complex, hierarchical reinforcement learning domain.

Another motivation in this project is to extend previous research investigating the effects of feature abstraction on learning performance in a hierarchical reinforcement learning domain. The fundamental theory here is that agents learning behavioral strategies from their environment will learn faster when afforded early feedback, even if not completely accurate. For purposes of intuition, consider the case of a young child learning how to interact with other human beings. From a young age, parents instill a simple wisdom: "don't talk to strangers." Taken literally, the lesson is that "interaction with unknown individuals is harmful." As adults we recognize that living by this creed is nigh impossible, and arguably not optimal. However, at an age of rough motor control, limited cognitive capabilities, and limited experience, it is difficult for a child to

absorb, assimilate, and act upon more complex directives. When the child has had time to tune its mental and physical capabilities, parents can refer to more nuanced, arguably more effective behavioral guidelines. This example roughly corresponds to what we believe should be the learning style of a hierarchical reinforcement learning agent. We provide learning structures that are initially easily accessible, by virtue of feature abstraction. This accessibility leads to frequent feedback and thus development of a roughly beneficial behavioral strategy. These initial, generalized strategies enable the agent to develop lower-level capabilities, which, once matured, assist in training more feature-specific behavior. This approach seems plausible and has some promising initial support [4]. This project provides a new, complex domain in which to test this theory.

1.2 Paper Outline

The remainder of this paper deals with the core of the project. Section 2 covers background material, including related literature, necessary to understand this project. Section 3 details the domain of study. Given this information, section 4 explicitly defines the research questions. Section 5 covers the research process, including experimental results and analysis. Section 6 contains some concluding remarks, including discussion of future work. The paper finishes with referenced works and appendices.

2 Background

This section covers the background material necessary to understand the research project. Section 2.1 briefly discusses the reinforcement learning (RL) problem. Section 2.2 summarizes relevant portions of the MAXQ method for hierarchical reinforcement learning (HRL). Section 2.3 discusses Soar-RL. Finally, section 2.4 covers relevant work done by Yongjia Wang and John Laird on the effect of action history on HRL performance [4].

2.1 The Reinforcement Learning Problem

The reinforcement learning (RL) problem is a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision maker are referred to as the agent. The interaction is done with the environment. The environment responds to agent actions by issuing reward, or numerical values, of which the agent tries to maximize over time [3].

More formally, at a particular time instant t, we can consider the state of the environment as S_t, the action chosen and performed by the agent as O_t, and the reward raised by the environment as r_t. If we introduce the concept of discounting the value reward received at future time steps, then we can consider a discount factor γ and thus the agent attempts to maximize the following quantity:

$$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$$

In the frequent case that the state and action spaces are finite, the RL problem is a special case of a finite Markov Decision Process (MDP).

In an effort to maximize the expected future reward, RL agents will typically estimate value functions. This function represents, with respect to a given behavior policy π, either the value of an agent being in a particular state (denoted the state-value function, V^{π}) or being in a particular state and performing an action (denoted the action-value function Q^{π}). Unless otherwise specified, we will be interested in the latter function type.

Temporal-Difference (TD) learning is an approach to the RL problem that combines simulation (Monte Carlo) and dynamic programming (DP) ideas, allowing for frequent updates to the value function without need for an environmental model. With respect to the terms and functions defined above, the TD update takes form as follows:

$$Q(s_t, o_t) = Q(s_t, o_t) + \alpha[r_t + \gamma \cdot Q(s_{t+1}, o_{t+1}) - Q(s_t, o_t)]$$

The new term here is the learning rate (α), used to tweak the number of updates needed for policy convergence (denoted as learning performance). Specific implementations of TD may be *on-policy* or *off-policy*. The difference being whether (on) or not (off) the policy used to select actions is the policy whose value function is being modeled.

For RL to discover the optimal policy, it is necessary that the agent sometimes choose an action that does not have the maximum predicted value. This situation may occur early in the learning (when the agent does not have an accurate prediction of the value of actions in each state) or if the reward function for a task changes (such as if the dynamics/rules of a task change). This problem is one of exploration vs. exploitation: the value trade-off of potentially improving the policy from predicted inferior strategy versus bringing the agent's current knowledge fully to bear. State-action exploration is typically defined by an exploration policy and, if applicable, one or more tuning parameters. For example, the epsilon-greedy exploration policy will choose to exploit with probability (1-ε) and explore with probability ε, where ε is a tuning parameter.

A time-consuming and important aspect of working within the RL paradigm is proper initial selection and time-dependent modification of learning and exploration parameters. While it is generally accepted that improper parameter selection will not prevent convergence of a policy (if it exists), the difference in learning performance can be dramatic. Unfortunately, there are not well-established guidelines for domain-specific parameter selection. Indeed, a significant amount of time, experimentation, and resulting doubt can be expended in proper parameter selection.

2.2 The MAXQ Method for Hierarchical Reinforcement Learning

Many RL problems can benefit from framing the domain as a hierarchical task. Learning performance can increase due to improved exploration, learning from fewer trials (because subtasks require knowledge of fewer parameters), and faster learning in task changes (because subtasks may remain constant). The MAXQ method composes a recursively optimal computational and representational model for hierarchical reinforcement learning (HRL). In demonstrating the method, Dietterich achieves significantly better learning performance and near optimal convergence, as compared to a "flat" implementation [1].

In representing an HRL problem, Dietterich proposes a MAXQ graph (see Figure 1 below). A MAXQ graph is composed of two types of nodes: Max nodes (triangles) and Q nodes (ovals). Max nodes with no children represent primitive actions; otherwise they represent hierarchical subgoals. Q nodes are the direct children of Max nodes and represent actions that can be performed to achieve parent node goals. The node differences deal with value function representation. Max nodes learn context-independent expected cumulative reward of performing the sub-task. Q nodes learn context-dependent expected cumulative reward of performing its sub-task. Further elaboration and an example will follow in the domain explanation (section 3).

It is important to remember that, in context of research motivation, MAXQ is the highly specific, highly optimized algorithm. Strictly speaking, MAXQ is a framework for representation and computation of HRL problems. However, the MAXQ paper presents a domain (see section 3 below) and a tabular implementation against which we shall compare our architectural integration.

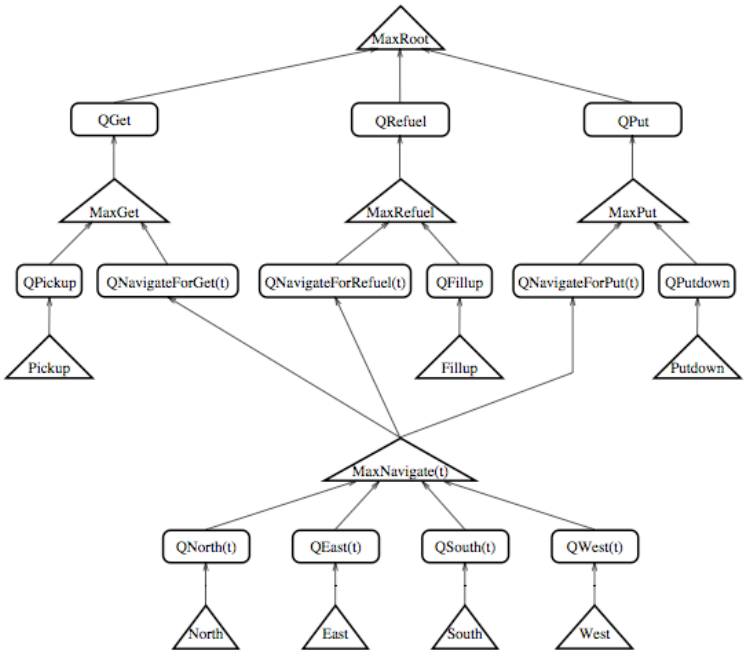

Figure 1: MAXQ Graph for the Taxi-World Domain

2.3 Soar-RL

Soar is a symbolic cognitive architecture based upon a production system. Soar has a working memory in which to store facts describing its current state and a production memory in which to store long-term knowledge (encoded as condition⇒action rules).

The following is the Soar execution cycle (as of version 8):

Input ⇒ Proposal ⇒ Selection ⇒ Application ⇒ Output

During the input phase, environmental data is populated in a special input branch of working memory. Based upon the agent's current state, rules may fire. During the proposal phase, these rules will nominate potential actions (termed "operators") that can be applied in the current state. Rules also fire to assert preferences as to the relative desirability of applying the proposed operators in the current state. During selection phase, the asserted preferences are compared, a decision is made, and a single operator is selected. In application phase, more rules may fire as a result of operator selection. Finally, a special output section of working memory is sent to the environment in the output phase.

Soar-RL is the architectural integration of reinforcement learning in Soar. Soar-RL represents two fundamental changes in Soar: numeric-indifferent preferences and rule updates. First, in addition to conditions and actions, preference rules in Soar-RL can contain a numeric "value" parameter that represents the expected future reward of applying an operator (action) under certain conditions (state). Thus, a Soar-RL rule is a representation of the action-value function, which directly affects operator selection. Second, during the operator Selection phase, Soar-RL looks to a special section of working memory for a reward signal. The architecture uses this reward to implement TD learning by changing the value of Soar-RL rules that contributed to the selected operator in the last execution cycle. To review:

1. Soar-RL preference rules fire to affect operator selection during Selection phase of cycle t
2. Between the selection phases of cycle t and $t+1$, rules fire to reflect the reward of applying the chosen operator
3. In the Selection phase of cycle $t+1$, reward is accumulated and used to perform a TD update to the Soar-RL rules in cycle t

The value parameters of Soar-RL rules affect the probability of their associated operators being selected. By changing these values, Soar-RL changes these associated probabilities. By altering the selection of actions, Soar-RL alters agent behavior. Reinforcement learning is thus achieved [2].

It is important to note that Soar simulates parallelism in the firing of its rules. Thus, in any given phase all rules that are triggered by the current agent state are fired. If rule actions trigger further production activations, another wave of rule firings commences. This process continues till production quiescence.

During the Proposal phase of an execution cycle, many Soar-RL rules may fire to affect the selection of an operator. If the conditions of these rules represent abstractions over the feature space, as opposed to exact representation of the current state, we call this feature abstraction. Since more general rules will fire more often, as compared to more specific rules, and a specific rule will always fire with the same general rule, the result is that the general rules quickly learn generalized Q values with relatively fewer training examples [4]. This is the functional evidence for our theories regarding feature abstraction.

2.4 Related Work

Wang and Laird have published work regarding the importance of action history in a Soar-RL domain [4]. The specific problem dealt with modeling rats in a T maze (see Figure 2 below). At each T juncture (numbered in the figure), the agent has to make a decision between turning north/south or east/west (relative to orientation). In this work, action history was a record of the last *n* decisions made. Wang and Laird found that they had to incorporate multiple levels of action history (i.e. differing values of *n*) in order to achieve learning behavior similar to real rat data.

In this context, action history can be considered a form of feature abstraction. Consider the case where *n*=1. The existence of such a history immediately implies junctions 2-14 (since junction 1 cannot have such historical information). Additionally, choices of north/south indicate the current junction must be 3, 5, 7… since the previous junctions were vertical. This evidence lends credit to our theories regarding feature abstraction as an important component of learning performance in HRL problems.

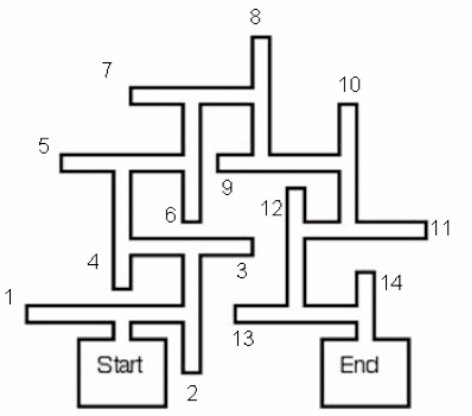

Figure 2: T Maze Domain in [4]

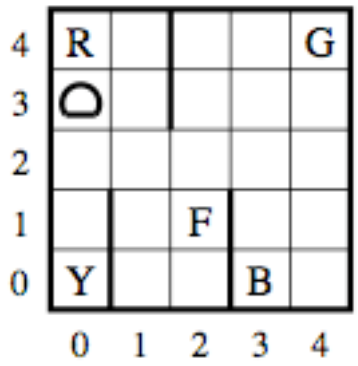

Figure 3: The Taxi-World Domain

3 The Taxi-World Domain

The Taxi-World domain (see Figure 3 above) was introduced in the MAXQ paper. In this domain, a taxi inhabits a 5x5 grid world. There are four special pickup/drop-off locations labeled (R)ed, (Y)ellow, (G)reen, and (B)lue, as well as a (F)uel point. The Taxi-World domain is episodic. In each episode, the taxi starts in a random grid position with an initial fuel level randomly chosen between 5 and 12 (inclusive). The passenger starts at one of the four, color locations (randomly selected per episode). The taxi's goal is to pickup the passenger at its source and drop it off at its destination color location (randomly selected, potentially the same as its source). The grid configuration (coordinates of walls and special locations) does not change from episode-to-episode.

There are seven primitive actions in this domain: movement (north, south, east, west, each consuming one unit of fuel), pickup, putdown, and fillup. Each action is deterministic. There is a -1 reward for each primitive action and an additional reward of +20 for successfully achieving the goal. There is a -10 reward if the taxi attempts a pickup or putdown illegally. If a movement would cause the taxi to hit a wall, the action result is negated, but still incurs a -1 reward. Finally, if the taxi runs out of fuel before achieving the goal, there is a -20 reward. In a given episode there are 3200 possible starting states (25 start positions, 8 fuel levels, 4 pickup locations, 4 drop-off locations). We computed that an optimal policy from any of these starting states will achieve an average reward per step of approximately 0.9098.

Figure 1 is the MAXQ hierarchical decomposition of the Taxi-World domain. Notice that the MaxRoot node need only learn to select at a high level from amongst three strategies (get the passenger, put the passenger, or refuel). Within each of these strategies, the Max nodes need only learn to navigate or perform a primitive action. At the navigation level, the MaxNavigate node need know only where it is and its destination (t), such to learn the effects of choosing a direction from the current state. This hierarchical decomposition reduces the number of parameters over which to apply RL. Additionally, since the navigation sub-goal is shared amongst the high-level strategies, it is able to learn efficient navigation strategies relatively quickly.

There are a number of interesting features of this domain. First, due to the initial fuel level minimum of 5 and the distribution of locations on the grid, the goal can always be achieved (i.e. you never start more than 5 moves from the Fuel location). Also, optimal path lengths are primarily symmetrical around the y=2 line (lending well to feature abstraction). Finally, the reward function, by virtue of unique values, naturally groups types of actions. This final feature lends well to the hierarchical credit assignment problem (namely, when reward is received at level *x*, determining the hierarchy level to which to apply the feedback).

4 Statement of Research Questions

Given research motivations, background, and the domain, it is now appropriate to discuss the specific research questions at hand. First:

> *Can a rule-based Taxi agent implemented in Soar-RL require relatively fewer values to represent Q-functions while still maintaining equivalent or better learning performance, as compared to the published tabular MAXQ implementation?*

This question gets to the heart of the first motivation of the research: comparing a custom, optimized algorithm to a generalized architecture. This question unfolds to form two sub-questions: (1) can Soar-RL match MAXQ in learning performance (i.e. what have we lost) and (2) can Soar-RL improve upon the means to attain this performance (i.e. what have we gained).

As mentioned previously, MAXQ has a tabular representation of the value function. This means that the algorithm stores a value for the cross product of all states with all actions that are pertinent in those states. Dietterich's paper claims their algorithm required over 18,000 values to represent the Q function. Soar-RL rules are a form of tabular representation. However, a special class of Soar-RL rules, called Soar-RL templates, is used to dynamically generate large number of Soar-RL rules as necessary. Thus, in the worst case, Soar-RL will require as many values to represent the value function. However, it is reasonable to expect that not all of the state/action pairs will be visited in the average case, and thus Soar-RL will require fewer values.

This question can be falsified in two ways. First, the Soar-RL agent is not able to achieve comparable levels of learning performance as compared to the published MAXQ implementation (convergent reward per step of 0.92 after about 12,000 trials). Second, the Soar-RL agent requires more Q function values to achieve this performance. Values are in this case are synonymous with the number of Soar-RL rules in an agent.

The second research question is as follows:

> *Does representing increasing levels of state generalization in reward signals result in faster learning when applied to a rule-based Taxi agent implemented in Soar-RL?*

This question addresses the heart of our investigation into feature abstraction (a term used interchangeably here with state generalization). Namely, will supplementing specific Soar-RL rules with generalized rules reduce the number of trials necessary to achieve policy convergence (equivalent or better than that in question 1)?

5 Research Process

This section is organized as follows: 5.1 discusses preliminary concerns including experimental components, variables, and measurements; 5.2 focuses on the navigation sub-goal; 5.3 details experiments regarding high-level decision making; and 5.4 presents results for a complete taxi agent.

5.1 Preliminary Concerns

The first stages of preparation included developing the tools with which to perform the research. These tools included the Soar-RL system, a Soar2D taxi domain, and a Soar experiment automation suite. First it was necessary to implement Soar-RL. This effort was an extension of previous work by Shelley Nason [2] and occurred primarily this past summer (with some extensions and bug fixes during this semester). Development of the Soar2D taxi domain was primarily the effort of Jonathan Voigt, research programmer for the Soar group. See Appendix A for program flow of the environment. Finally, RL experiments require averaging many runs of many sequential trials of experiments, often times changing RL or exploration parameters. Thus, it was necessary to develop a set of scripts to efficiently automate the process of collecting data en masse from Soar-RL experiments.

To answer the research questions, it was important to collect the correct data during experiments. The three major data points from a run were the average number of Soar-RL rules, the average reward per step of the convergent policy, and the number of trials till convergence. Average reward per step and trials till convergence are used to illustrate learning performance. Number of Soar-RL rules represents the number of values required to represent the Q function.

In testing feature abstraction, a simple set of location generalization rules was used in the navigation sub-task. The base rule models the value of moving in a cardinal direction, given an (x,y) coordinate. The "state generalization" rule set models the value of moving in a cardinal direction given a "side" (left or right of x=2), "hemisphere" (up or down from y=2), and quadrant (combinations of side and hemisphere). We postulated integrating abstractions of fuel levels in MaxRoot, but have yet to implement.

After developing the preliminary tools, we tested a full agent (without feature abstraction rules) in the taxi world environment. It was our [naïve] assumption that all would work and we could progress quickly to the more interesting questions of feature abstraction rules. The result, however, was the agent displayed a negative learning slope. Thus, we opted to break down the agent into sub-goals and study/debug iteratively.

5.2 Navigation

Our first experiments in the navigation sub-goal were to learn how quickly the taxi could find a labeled location. Note that the agent knows its current (x,y) position in the grid and the desired destination, but initially it does not know how to get there. Thus, the RL problem is to learn optimal path planning via local search.

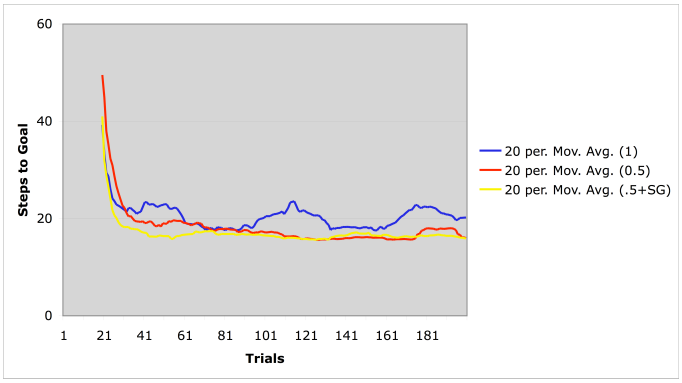

Figure 4: Navigation - Constant Start, Constant End

The agent was given a static start position and a static end location. The measurement of interest was number of movements till goal. Fuel concerns were removed from the problem. The experiment consisted of 3 averaged runs of 200 trials each, with all combinations of learning rates (1, .7, .5, .3) and state generalization (used or not). Exploration was achieved using epsilon-greedy exploration policy (ε=0.1) and the TD implementation was SARSA (on-policy).

Figure 4 illustrates smoothed moving averages (period=20) from this experiment set. The blue line had a learning rate of 1, the red line had a learning rate of 0.5, and the yellow line had a learning rate of 0.5, as well as state generalization rules. This initial result set was very informative. First it helped identify strong candidates for learning rate (0.5 was empirically the best choice in this problem). With learning rate 1 (as shown), learning performance was very fast (about 60 trials to convergence), but the average steps at convergence was relatively high and varied. A learning rate of 0.5 enhanced stability and level of convergence, but at the cost of performance rate. Combining learning rate of 0.5 with state generalization rules achieved fast, stable, optimal learning performance.

We then performed a series of experiments to complicate the problem. We achieved similar results (not shown) for constant start/random end, random start/constant end, and random start/random end. To illustrate the full navigation task, we investigated the following problem: start at a random coordinate, pick-up a passenger from a random labeled location, and drop-off the passenger at a random labeled location. Results were similar to Figure 4 and are shown in Figure 5 below. The careful reader will notice that the results for full navigation appear to be equal to those of half the task. The discrepancy here occurred because late in the experimentation process we realized that our count of primitive steps was being exaggerated by Soar, which often takes many meta-steps for a single environmental primitive. The data were corrected in Figure 5, but not Figure 4.

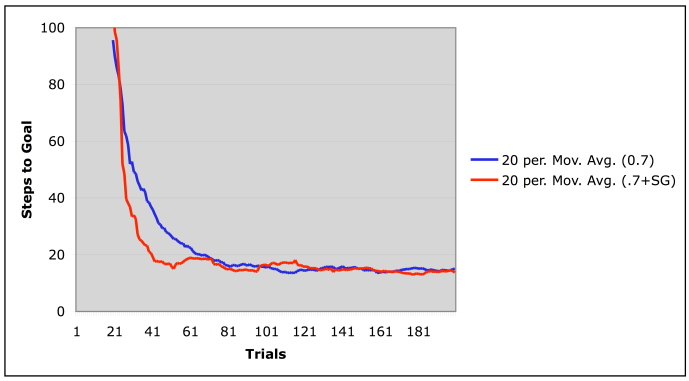

Figure 5: Full Navigation

5.3 High-Level Decisions

Once we were convinced that the Soar-RL taxi agent was achieving optimal navigation strategies quite quickly (about 50 trials), we modified the agent to learn, via HRL, all the levels of the MAXQ hierarchy except refueling (we considered this to be a complication that could be postponed). Thus, the agent was now learning *when* to pickup, putdown, and navigate. Using epsilon-greedy exploration (ε=0.1), SARSA TD, and deterministic discount rate (γ=1), we ran a similar barrage of tests on the agent. The task was to pick-up the passenger from a random location and drop-off the passenger at a random destination. In addition to varying learning rate and usage of state generalization rules, we also introduced more reasonable bounds on fuel (initial=20, 100). The measurement for evaluation in these experiments was reward per step. We ran each experiment for 1000 trials and averaged over 3 runs.

As with the full navigation experiments, a learning rate of 0.7 proved to exhibit the best combination of learning performance and convergence stability. In the case of high initial fuel level (see Figure 6, smoothing period=50), state generalization rules did slightly increase learning performance.

In the case of low initial fuel level (20), we found a significant decrease in performance. After some analysis, we believe a short-lived bug relating to hierarchical credit assignment caused this discrepancy. In this case, the navigation sub-goal, rather than the MaxRoot, was receiving -20 reward for attempting to move without remaining fuel. As a result, the high-level decision making ability was not receiving feedback by which to improve the choice of QGet vs. QSet, while navigation was receiving "confusing" reward signals that was interfering with proper path planning. Unfortunately, time restrictions prevented us from generating new data sets for this experiment after learning of the cause. The high initial fuel data set did not suffer as greatly from this problem because the -20 reward signal would come much less frequently.

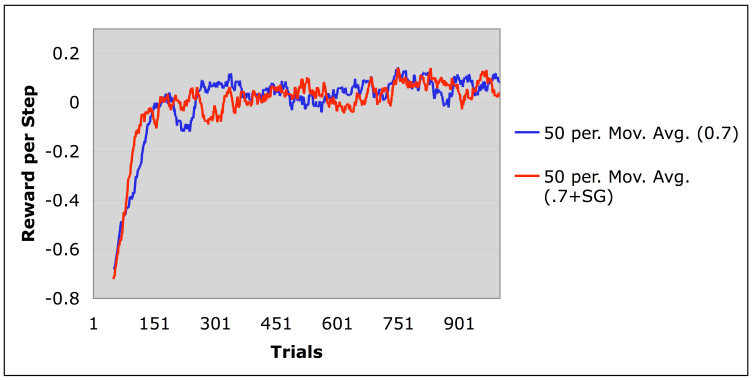

Figure 6: High-Level Decisions, Fuel=100

5.4 Full Soar-RL Taxi-World Agent

Results from the full Soar-RL taxi agent (full HRL over the entire MAXQ graph) can be seen in Figure 7 below. The graph is showing a smoothed, moving average trend line (period=150) of three averaged runs over 15,000 trials. Exploration policy was epsilon-greedy (ε=0.1), TD implementation was SARSA, discount rate was 1, learning rate was 0.5.

The results are encouraging, discouraging, and carry some caveats. First, convergence occurred quite quickly (within about 2000 trials) and took about 4000 Soar-RL rules to achieve this performance. Unfortunately, reward per step at convergence was only about 0. It is worth noting that this data was collected before we discovered two crucial pieces of information: learning rate was more efficient at 0.7 for the full problem and the difference between Soar primitive steps and environmental primitive steps. Thus, we expect that learning performance and reward per step at convergence are slightly higher than depicted here. Unfortunately, collecting one data set requires approximately two days on a 2.8GHz Core 2 Duo, and thus we have been unable to produce any more runs at this time.

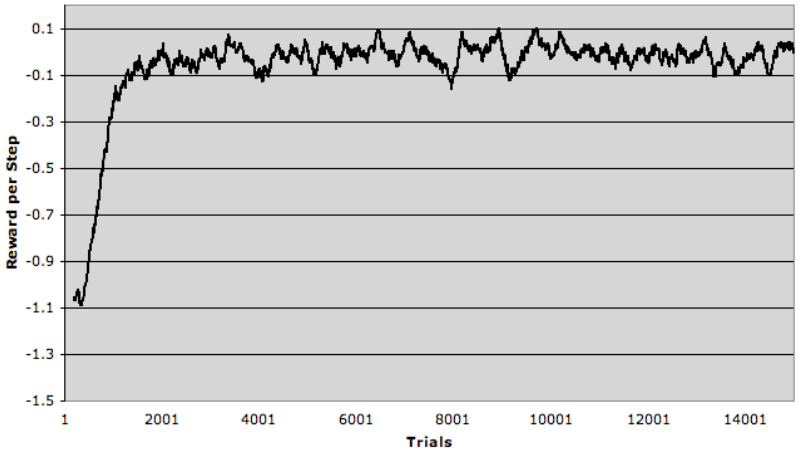

Figure 7: Full Soar-RL Taxi-World Agent

6 Conclusions

This section is organized as follows: 6.1 summarizes the results in context of the stated research questions and 6.2 discusses future work.

6.1 Revisiting the Research Questions

For review, we restate here the research questions from section 4:

Can a rule-based Taxi agent implemented in Soar-RL require relatively fewer values to represent Q-functions while still maintaining equivalent or better learning performance, as compared to the published tabular MAXQ implementation?

Does representing increasing levels of state generalization in reward signals result in faster learning when applied to a rule-based Taxi agent implemented in Soar-RL?

With regard to the first question, consider the following summary of the results of our full agent experiments, as compared to the published MAXQ implementation:

	MAXQ	**Soar-RL**
Trials to Converge	12,000	**2,000**
Values	18,000	**4,000**
Reward per Step	**0.92**	0

Unfortunately, while we are able to show extremely fast convergence with relatively few values to represent the Q function, we are not able to achieve the published average reward per step in our convergence strategy. This result is not entirely unexpected. The MAXQ implementation, as just one optimization, applies different exploration policies to different nodes in the MAXQ graph. This customization likely helps achieve a higher, stable reward per step and is just one example of a feature Soar-RL, in the spirit of architectural production systems, cannot duplicate. Thus, at this time, the answer to the first question is no.

With respect to the second question, we have some promising evidence to support the theory that feature abstraction Soar-RL rules do in fact increase learning. We have presented data to support this theory in the navigation sub-goal and high-level decision-making. Unfortunately, time constraints prevented us from generating a full agent data set with state generalization rules (the added rules increase run time a great deal). Thus, pending future verifying work, we submit a hopeful yes to the second question.

6.2 Future Work

One major avenue we intend to pursue in future experimentation is decreasing exploration rates over time. Research has shown that proper management of exploration policies over the course of a run is necessary to achieving optimal convergence.

Another point of interest has to do with in-depth comparison of TD updates as they relate to specific versus state generalization rules. One would expect, for instance, that absolute updates to specific rules would decrease over time to convergence, whereas generalized rules may remain active (due to inaccuracies). Investigating patterns in these updates may lead to insights as to autonomously tuning RL and exploration parameters during the course of an experiment.

During the course of this project, we calculated the average reward per step of the optimal policy from any starting point in the domain. This value is less than the published mean optimal value in the MAXQ paper. We have contacted the author and he has afforded us access to his implementation source code. We may be able to gain additional insights into our system by comparing internals with his work.

Assuming we are able to achieve an agent that is capable of [at least] equivalent learning performance of MAXQ, it would be interesting to experiment with having the agent learn over the entire feature space, as opposed to manually curated feature sets.

Finally, having investigated feature abstraction in the Taxi-World domain, it will be intriguing to explore how well the results generalize to similar HRL domains.

Acknowledgements

I would like to thank John Laird, my advisor, for his guidance and support in selecting this project, as well as pushing me to find simplifying approaches to complex problems.

Boundless appreciation goes to Jon Voigt, research programmer for the University of Michigan Soar Group, for developing and maintaining the Taxi-World Soar2D application (at all hours, on all days).

Thank you Michael Wellman for facilitating this research and providing guidance and feedback through its development.

Much gratitude goes to Jim Boerkoel and Brett Clippingdale for their feedback in development of project materials.

Thanks to Melanie Agnew for helping to verify the algorithm to compute the average reward per step of an optimal agent in the Taxi-World domain.

References

[1] Dietterich, T., The MAXQ Method for Hierarchical Reinforcement Learning, Proceedings of the Fifteenth International Conference on Machine Learning, p.118-126, July 24-27, 1998.

[2] Nason, S. and Laird, J. E., Soar-RL, Integrating Reinforcement Learning with Soar, International Conference on Cognitive Modeling, 2004.

[3] Sutton, R.S., and Barto, A.G., Reinforcement Learning: An Introduction. MIT Press, Cambridge, 1998.

[4] Wang, Y., and Laird, J.E. 2007. The Importance of Action History in Decision Making and Reinforcement Learning. Proceedings of the Eighth International Conference on Cognitive Modeling. Ann Arbor, MI.

Appendix A: Taxi-World Environment

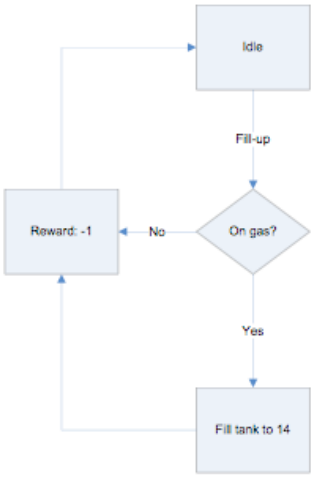

Figure 8: Fill-Up Action

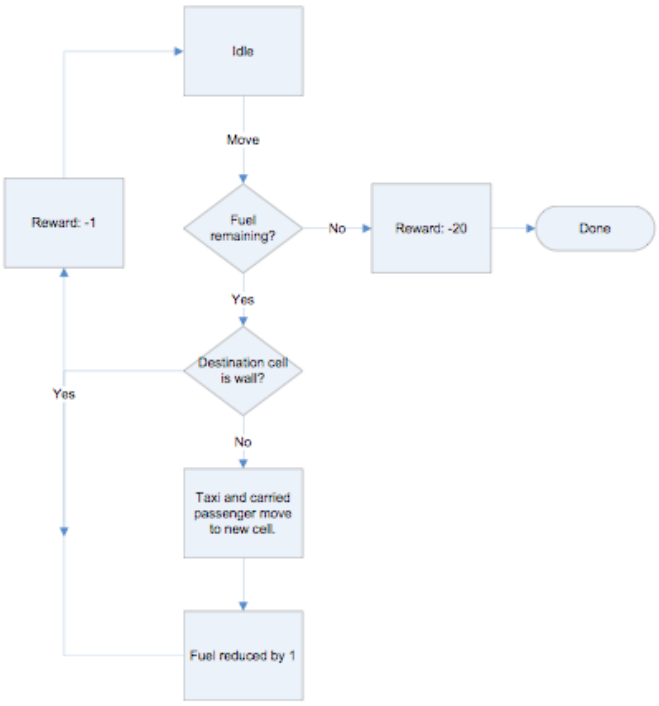

Figure 9: Movement

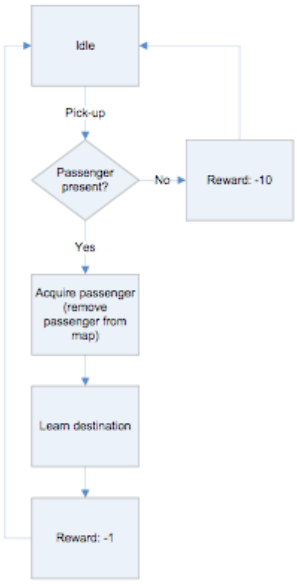

Figure 10: Pick-Up Action

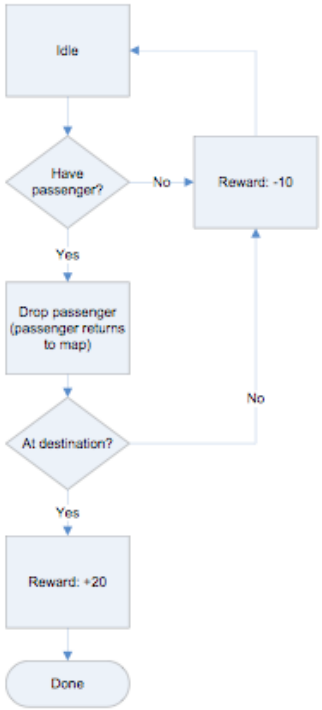

Figure 11: Putdown Action