Center for Cognitive Architecture
University of Michigan
2260 Hayward St
Ann Arbor, Michigan 48109-2121

TECHNICAL REPORT
CCA-TR-2009-02

---

1. Introduction

This work is intended to assess whether Soar agents employing hierarchical reinforcement learning perform better at the taxicab problem than their flat reinforcement learning counterparts. This serves as a test of the reproducibility of the original work [Dietterich 1998] and as an evaluation of the implementation of hierarchical reinforcement learning in Soar.

2. Related Work

Soar is a cognitive architecture that has been under active development for approximately 25 years. At the most basic level, it provides mechanisms for the manipulation of symbolic representations of data. Relatively recently, it has been extended to support reinforcement learning in tandem with its symbolic reasoning systems.

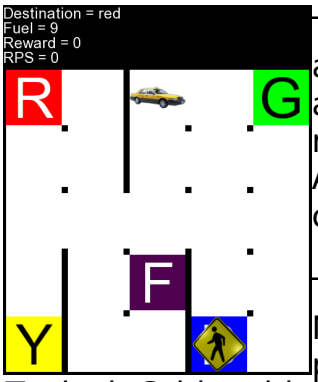
Reinforcement learning necessitates the formulation of both a set of discrete states which correspond to states in the environment and a set of actions which can be performed in the environment. A value function over all state-action pairs represents the agent's beliefs about the expected reward for taking an action from any given state. Given a reward signal from the environment, the value function will converge to the true value function as an agent explores the environment[1]. If the agent chooses to explore less and less over time, the agent's policy will converge to the optimal policy[2].

The taxicab problem domain is well known in the area of reinforcement learning. Simply put, a taxicab driver is tasked with the problem of picking up a passenger and delivering him to his destination in as few steps as possible. Typically, the taxi is constrained by a limit on the amount of fuel that can be carried.

Thomas Dietterich [1998] explored the taxicab problem in his introduction of the MAXQ decomposition for hierarchical reinforcement learning. He introduced a variant of the informed-finite task that will be presented in section 3.2. He designed the MAXQ hierarchy which will be presented in section 4.3.2. Finally, he demonstrated the performance of both agents, showing that the hierarchical agent learned significantly more quickly that the flat agent.

1 This presumes that each state-action pair can be taken an infinite number of times.
2 This holds true only if the exploration rate is decreased sufficiently slowly for the value function to converge to the true value function.

3. Formal Specification of the Problem



Taxicab Gridworld

The canonical taxicab problem is a 5x5 gridworld. There are four cells which serve as possible starting locations and possible destinations for the passenger. There is a refueling station near the middle of the map. Additionally, there are six impassable walls (or 26 counting the walls surrounding the map).

The seven actions available to the taxi are moving North, South, East and West, picking up the passenger, putting down the passenger, and refueling.

An attempt to move North, South, East, or West automatically results in the taxi moving one cell in that direction unless there is a wall in the way, in which case the move action is ignored and the taxi remains in place. Fuel decreases by 1 unless the move action is ignored. Pickup always results in the passenger being picked up if the taxi does not have the passenger and is at the passenger's starting location. Putdown always results in the passenger being put down if the taxi has the passenger and is at the destination. Refuel always sets the amount of fuel to 14 if the taxi is at the refueling station.

Each of the seven actions takes 1 unit of time. Move, pickup, putdown, and refuel actions each yield a reward of -1 except in the following cases. Refuel, pickup, and putdown each yield a reward of -10 instead if the action is impossible when attempted. Move yields an additional reward of -20 if it causes fuel to drop below 0, resulting in failure of the trial. Putdown yields an additional reward of 20 if it causes the passenger to arrive at his destination, resulting in successful termination of the trial.

The passenger has a 25% chance of starting at any of the four starting locations and a 25% chance of wishing to visit any of the possible destinations. The taxi has a 4% chance of starting at any given location in the gridworld. The taxi starts with an amount of fuel between 5 and 12 (inclusive), again with an equal probability of any given value being selected. Given these initial conditions, the task can always be solved by a competent agent.

There are two possible ways of calculating an average reward over many trials. The first is intuitive because it has a linear correlation with reward received. The second is less intuitive because it has a non-linear correlation with reward received, but it is important for understanding the plots in

section 5. Note that these equations are insufficient for calculating reward if agents are allowed to attempt illegal actions.

$$\frac{\sum_{trial=1}^{n} 20 - count_{trial}}{\sum_{trial=1}^{n} count_{trial}} \qquad \sum_{trial=1}^{n} \frac{20 - count_{trial}}{count_{trial}}$$

There are four different variants of the problem that we will consider:

3.1 Informed-Infinite

The informed-infinite variant of the problem presents the agent with infinite fuel and complete knowledge of the passenger's starting cell and destination cell. The optimal policy in this case is as follows:

1. Take the shortest sequence of move actions to get to the passenger.
2. Pickup the passenger.
3. Take the shortest sequence of move actions to get to the destination.
4. Putdown the passenger.

The average number of steps required for this task is approximately 11.45. If rewards are calculated over many trials, the average return of this policy is approximately 0.75 reward per step. If rewards are calculated on a per trial basis, the average return of this policy is approximately 1.09 reward per step.

3.2 Informed-Finite

The informed-finite variant of the problem presents the agent with finite fuel with complete knowledge of the passenger's starting cell destination cell. The optimal policy in this case is more difficult to describe, as it may be necessary to refuel once or not at all.

1. Refuel as needed.
2. Take the shortest sequence of move actions to get to the passenger.
3. Pickup the passenger.
4. Refuel as needed.
5. Take the shortest sequence of move actions to get to the destination.
6. Putdown the passenger.

The complication is that cases exist where either step 1 or 4 would be satisfactory, and it is necessary to determine which is more efficient in order to behave optimally.

The average number of steps required for this task is approximately 13.20. If rewards are calculated over many trials, the average return of this policy is approximately 0.52 reward per step. If rewards are calculated on a per trial basis, the average return of this policy is approximately 0.93 reward per step.

3.3 Uninformed-Infinite

The uninformed-infinite variant of the problem presents the agent with infinite fuel but no information regarding the passenger's starting cell or destination cell. A nearly optimal policy in this case is as follows:

1. Take the shortest sequence of move actions to get to the nearest of the four possible starting cells for the passenger.
2. If the passenger is present, pickup the passenger.
   Otherwise, ignore the cell and go back to step 1.
3. Pickup the passenger.
4. Take the shortest sequence of move actions to get to the destination.
5. Putdown the passenger.

The average number of steps required for this task is approximately 17.01. If rewards are calculated over many trials, the average return of this policy is approximately 0.18 reward per step. If rewards are calculated on a per trial basis, the average return of this policy is approximately 0.59 reward per step.

3.4 Uninformed-Finite

The uninformed-finite variant of the problems presents the agent with finite fuel and no information regarding the passenger's starting cell or destination cell. A nearly optimal policy in this case is as follows:

1. Take the shortest sequence of move actions to get to the refueling station.
2. Refuel.
3. Take the shortest sequence of move actions to get to the blue cell.
4. If the passenger is present, go to step 12.
5. Take the shortest sequence of move actions to get to the green cell.
6. If the passenger is present, go to step 12.
7. Take the shortest sequence of move actions to get to the refueling station.
8. Refuel.
9. Take the shortest sequence of move actions to get to the red cell.
10. If the passenger is present, go to step 12.

11. Take the shortest sequence of move actions to get to the yellow cell.
12. Pickup the passenger.
13. If fuel is sufficient for getting to the destination cell, go to step 16.
14. Take the shortest sequence of move actions to get to the refueling station.
15. Refuel.
16. Take the shortest sequence of move actions to get to the destination.
17. Putdown the passenger.

It is obvious that the initial refueling steps may not always be necessary. In fact, the optimal policy depends on both the starting location for the taxi and the starting fuel. However, all plans follow this basic pattern. Only the order in which the cell types is visited changes.

The average number of steps required for this task is approximately 23.87. If rewards are calculated over many trials, the average return of this policy is approximately -0.16 reward per step. If rewards are calculated on a per trial basis, the average return of this policy is approximately -0.09 reward per step.

4. Agent Construction

4.1 Taxicab SML

The taxicab environment for the Soar group was originally implemented in Java by Jon Voigt, a research computer specialist in the Soar group at the University of Michigan. I reimplemented the environment (in C++) in order to correct some discrepancies between the Soar2D environment and Dietterich's environment and, more importantly, to allow me to implement certain tricks for the agents that could not be implemented easily within Soar. Both versions of the environment connect to Soar through the use of Soar Markup Language. The important details of the final implementation are described above.

4.2 Soar-RL

There are three specially designated parts of a Soar agent's symbolic memory structure that are important for this task. The input-link is a conduit for Soar2D to provide a Soar agent with the information it needs to perform its task. The output-link is a conduit for a Soar agent to manipulate its environment. Finally, reward-links provide a mechanism for a Soar agent to give the reinforcement learning system rewards. These rewards may come from the input-link or be internally generated.

Each of the agents I built for these tasks are provided with the following information on the input-link. They know the current global coordinates of the taxi on the map. They know the type of cell currently occupied by the taxi. They know how much fuel is left in the tank. They know whether the passenger has been picked up or whether the passenger is at the current cell. Omniscient agents know the type of cell for both the source and destination of the passenger. Uninformed agents know the destination of the passenger only after the passenger has been picked up.

The seven primitive actions (move North, move South, move East, move West, pickup, putdown, and refuel) can be placed directly on the output-link by Soar operators.

Finally, rewards from the environment are presented by the input-link and must be placed on a reward-link by the Soar agent. In flat agents, this is trivial. In hierarchical agents, care must be taken to assign credit appropriately. Additionally, due to limitations of Soar RL, it is necessary to internally generate rewards for subgoals which do not directly receive credit from the environment.

4.3 Soar Agents

All four agents do not discount reward[3]. They employ SARSA without eligibility traces. They use a learning rate of 0.3 and Boltzmann indifferent-selection with an initial temperature of 1.0[4]. Finally, the temperature is decayed at a rate of 0.9999 per time step to a minimum of 0.05. This lower bound prevents Boltzmann indifferent-selection, $e^{\frac{Q(s,a)}{\tau}}$, from failing due to floating point arithmetic overflow. This exponential reduction rate was chosen to decay the temperature to a value near the minimum in the final episodes of a run. Additionally, the hierarchical agents have a different temperature at each Max node in order to match Dietterich's work [1998].

The combination of Boltzmann indifferent-selection and SARSA strongly discourages the agents from exploring states from which they have received large negative rewards in the past. However, they still give agents the flexibility to explore states which are believed to be only slightly suboptimal.

Using a learning rate that is low (considering the deterministic nature of the

3 It is possible for agents to run forever in both the infinite and finite-fuel tasks, but there is no incentive in the environment to encourage this behavior. Values can become sufficiently negative to break Boltzmann indifferent-selection in badly designed agents.

4 This initial temperature is considerably lower than that of Dietterich's agents because his initial temperature seemed to cause an unacceptable delay in learning.

informed tasks) and disabling eligibility traces prevents the agents from learning an overly negative view of all possible actions very early on. This gives agents more time to learn before giving up on certain avenues of exploration. This serves a similar purpose to Dietterich's hierarchical agent's performing “an update for a Q node [only] if that node completed its subtask with an average absolute Bellman error per step of less than 0.2 .”

4.3.1 Flat Omniscient Agent

4.3.1.1 Move

Move actions decide over position (25 possibilities), direction (4 possibilities), fuel (15 possibilities), source (4 possibilities until the passenger has been picked up, ignored afterward), destination (4 possibilities), and whether the passenger has been picked up or not (2 possibilities). These factors yield 30000 Q-values for all move actions or 7500 Q-values per primitive move action.

4.3.1.2 Pickup

Refuel actions decide over position (25 possibilities), source (4 possibilities), and whether the passenger has been picked up or not (2 possibilities). These factors yield 200 Q-values.

4.3.1.2 Putdown

Refuel actions decide over position (25 possibilities), destination (4 possibilities), and whether the passenger has been picked up or not (2 possibilities). These factors yield 200 Q-values.

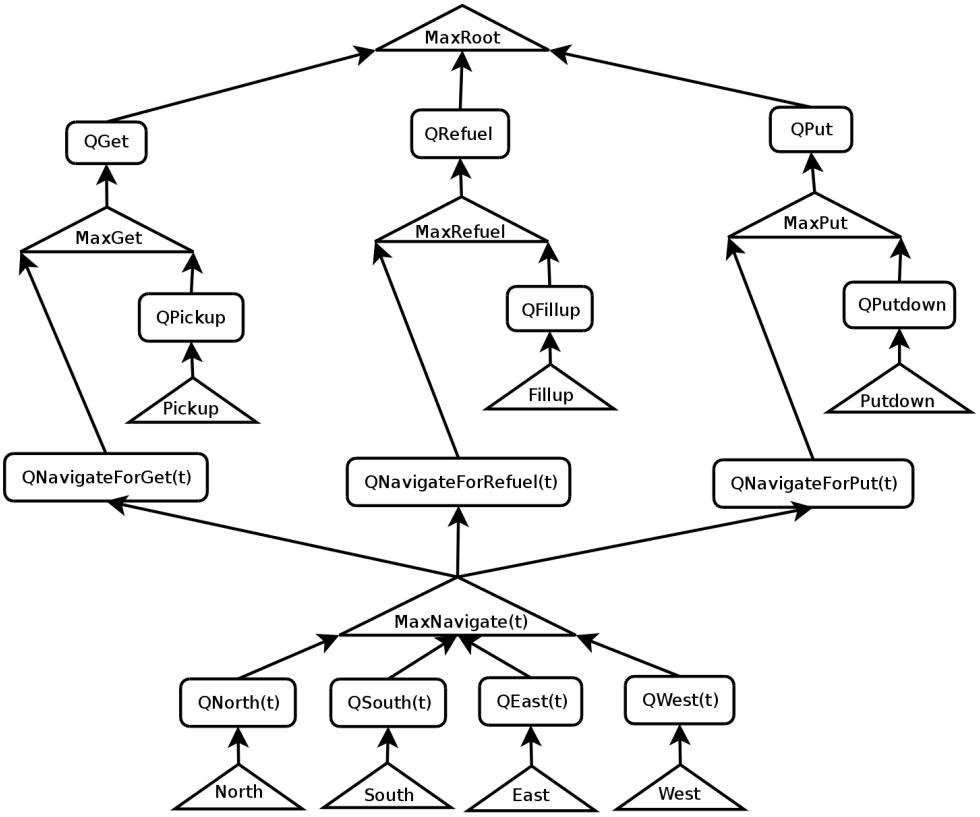
4.3.1.2 Refuel

Refuel actions decide over position (25 possibilities), fuel (15 possibilities), source (4 possibilities until the passenger has been picked up, ignored afterward), destination (4 possibilities), and whether the passenger has been picked up or not (2 possibilities). These factors yield 7500 Q-values.

4.3.1.3 Rewards

All rewards from the environment are passed directly to the reward link.

4.3.2 Hierarchical Omniscient Agent



Dieterrich's Hierarchy for the Taxicab Domain

4.3.2.1 MaxRoot

When referring to position and fuel, I am referring specifically to the position and fuel at the time of the decision. These values are not updated while the agent attempts to perform the action.

QGet decides over position (25 possibilities), fuel (15 possibilities), the source (4 possibilities), and the destination (4 possibilities) yielding 6000 Q-values.

QPut decides over position (25 possibilities), fuel (15 possibilities), and the destination (4 possibilities) yielding 1500 Q-values

Additionally, MaxRefuel is an option in each of the 7500 states in which MaxGet or MaxPut is an option, yielding an addition 7500 Q-values for QRefuel.

4.3.2.2 MaxGet

QNavigate(t) decides over nothing (1 possibility).

Pickup decides over position (25 possibilities) and the source (4 possibilities), yielding 100 Q-values.

4.3.2.2 MaxPut

QNavigate(t) decides over nothing (1 possibility).

Pickup decides over position (25 possibilities) and the destination (4 possibilities), yielding 100 Q-values.

4.3.2.2 MaxRefuel

QNavigate(t) decides over nothing (1 possibility).

Refuel decides over position (25 possibilities) only, yielding 25 Q-values.

4.3.2.3 MaxNavigate(t)

Q[North/South/East/West](t) decides over position (25 possibilities), direction (4 possibilities), and choice of destination (predetermined 1 of 5) yielding 100 Q-values for each of the 5 values of 't'.

4.3.2.4 Rewards

Rewards of ±20 are passed from the environment to the top-level reward-link, affecting values QGet, QPut, and QRefuel. Rewards of -10 are passed to MaxGet, MaxPut, and MaxRefuel, affecting values for QNavigate(t), QPickup, QPutdown, and QRefuel. Rewards of -1 are passed from the environment directly to all layers of the hierarchy. This is necessary in order to encourage all subtasks to complete in as short a time as possible.

An internal reward of 10 is generated for successful completion of MaxGet(t). An internal reward of 10 is generated for successful completion of MaxRefuel(t). Finally, an internal reward of 5 is generated for successful completion of MaxNavigate(t).

4.3.3 Flat Uninformed Agent

4.3.3.1 Move

Move actions decide over position (25 possibilities), direction (4 possibilities), and fuel (8 possibilities [0 1 2 3 4 medium=[5,9] high=[10,13] full=14]). Which of the four possible starting locations have been searched (16 possibilities) is a factor until the passenger is picked up. Afterwards, destination (4 possibilities) is a factor. These factors yield 16000 Q-values.

Keeping track of which locations have been searched turns out to be critically important. Otherwise, it is virtually impossible for an agent to tell if it just searched the green cell and is heading to search the blue cell or vice versa. This ambiguity results in a pattern of motion resembling a random walk, virtually preventing learning from taking place.

Additionally, after relaxing a restriction that pickup be performed whenever possible and never when impossible, the task became more or less impossible to learn without fuel abstraction. Medium guarantees that an agent can get to the fuel source from anywhere on the map. High guarantees that an agent can get from the fuel source to anywhere on the map, and then return to the fuel source.

4.3.3.2 Pickup

Pickup actions decide over position (25 possibilities) and whether the passenger is known to be present at the current type of cell (2 possibilties), yielding 50 Q-values.

4.3.3.2 Putdown

Pickup actions decide over position (25 possibilities) and destination (4 possibilties), yielding 100 Q-values.

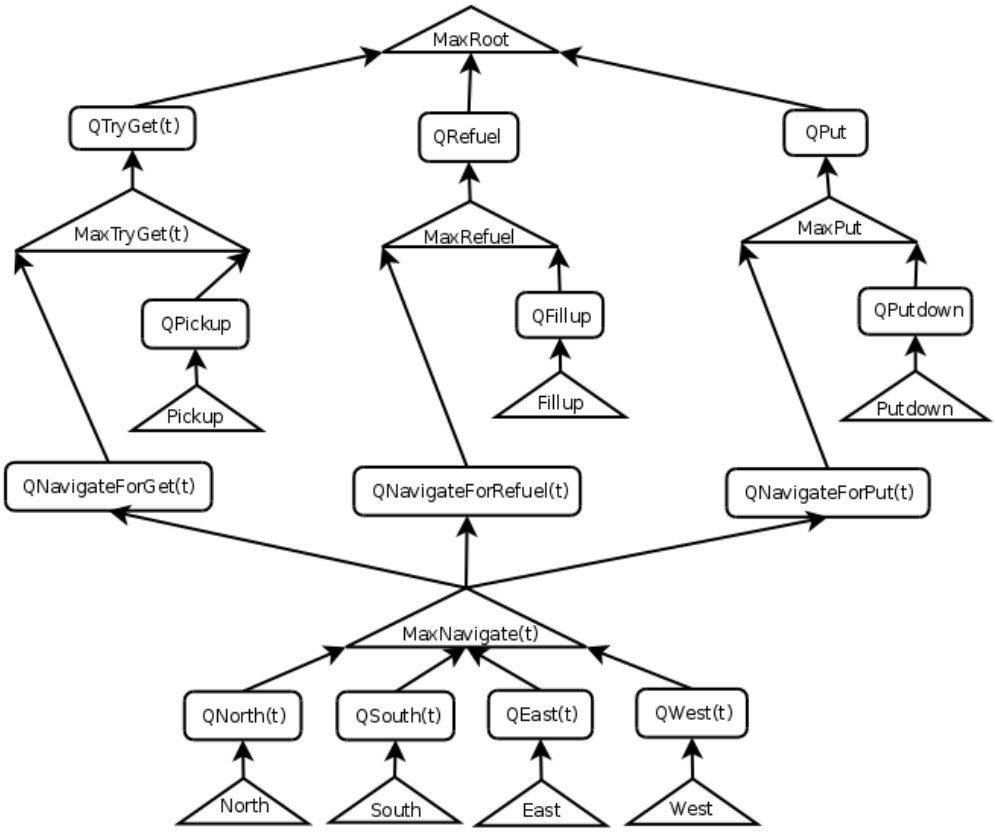
4.3.3.2 Refuel

Refuel actions decide over position (25 possibilities) and fuel (8 possibilties), yielding 200 Q-values.

4.3.3.3 Rewards

All rewards from the environment are passed directly to the reward link.

4.3.4 Hierarchical Uninformed Agent



A Modification of Dietterich's Hierarchy

4.3.4.1 MaxRoot

QTryGet decides over position (25 possibilities), fuel (15 possibilities), and which of the sources has been visited (16 possibilities) yielding 6000 Q-values.

QPut decides over position (25 possibilities), fuel (15 possibilities), and the destination (4 possibilities) yielding 1500 Q-values

Additionally, MaxRefuel is an option in each of the 7500 states in which MaxGet or MaxPut is an option, yielding an addition 7500 Q-values for QRefuel.

4.3.2.2 MaxTryGet(t)

QNavigate(t) decides over nothing (1 possibility).

Pickup decides over position (25 possibilities) and the source being tried

(predetermined 1 of 4), yielding 25 Q-values for each of the 4 values of 't'.

4.3.2.2 MaxPut

QNavigate(t) decides over nothing (1 possibility).

Pickup decides over position (25 possibilities) and the destination (4 possibilities), yielding 100 Q-values.

4.3.4.3 MaxRefuel

QNavigate(t) decides over nothing (1 possibility).

Refuel decides over position (25 possibilities) only, yielding 25 Q-values.

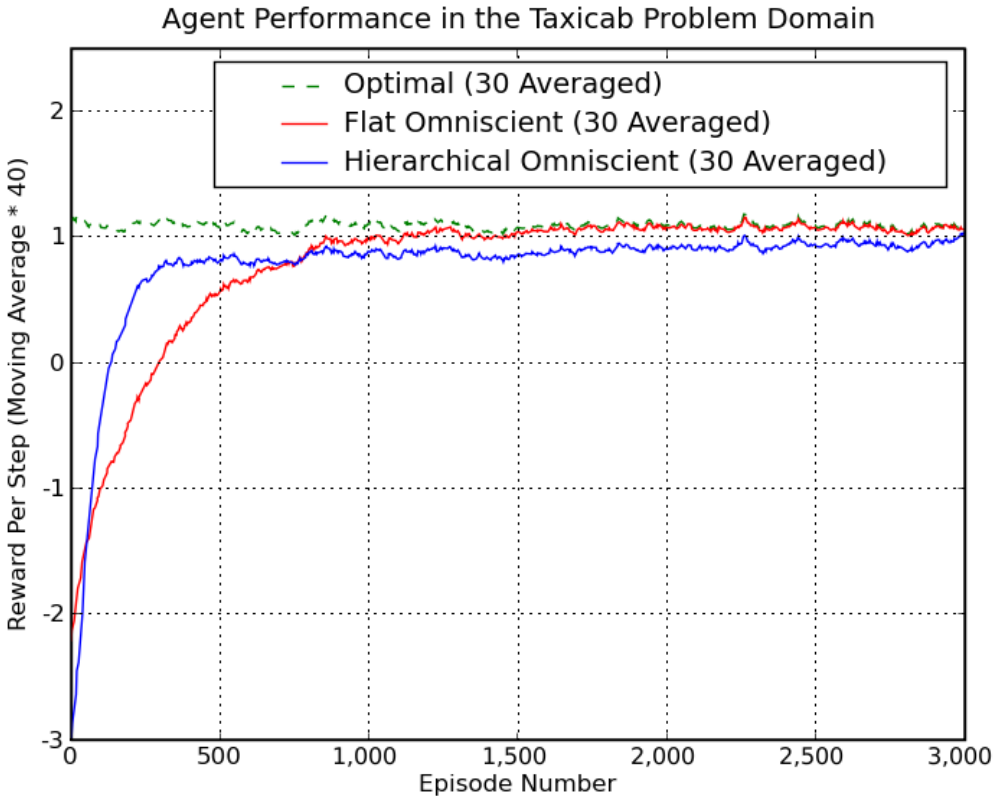
4.3.4.4 MaxNavigate(t)

Q[North/South/East/West](t) again decides over position (25 possibilities), direction (4 possibilities), and choice of destination (predetermined 1 of 5) yielding 100 Q-values for each of the 5 values of 't'.

5. Methodology and Results

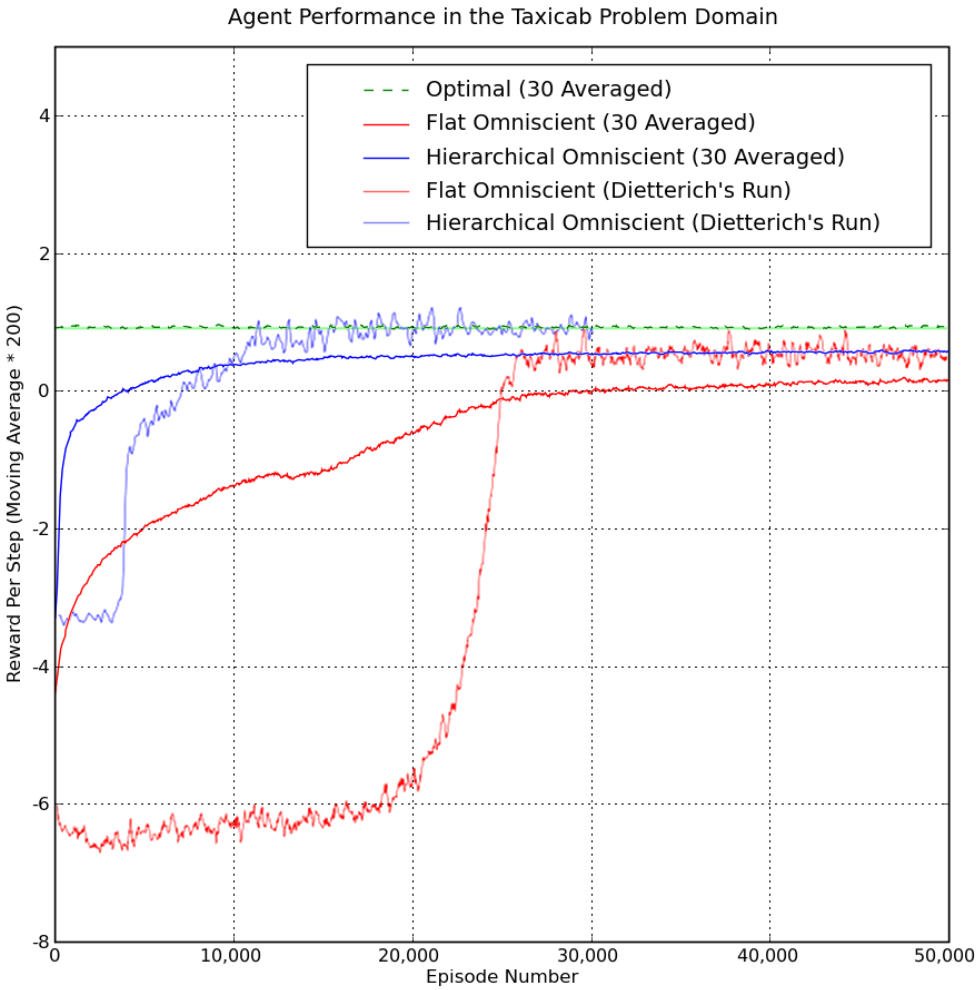
Running these flat and hierarchical agents on the tasks to which they are suited should yield results which demonstrate the superiority of hierarchical reinforcement learning over flat reinforcement learning in the taxicab problem.

5.1 Informed-Infinite Task



After disabling exploration after 3000 episodes, the optimal reward possible over 500 episodes was 1.10 reward per step. The flat omniscient agent averaged 1.09 reward per step and the hierarchical omniscient agent averaged 1.10 reward per step. The hierarchical omniscient agent matched the optimal for all 500 episodes in 29 runs out of the 30.

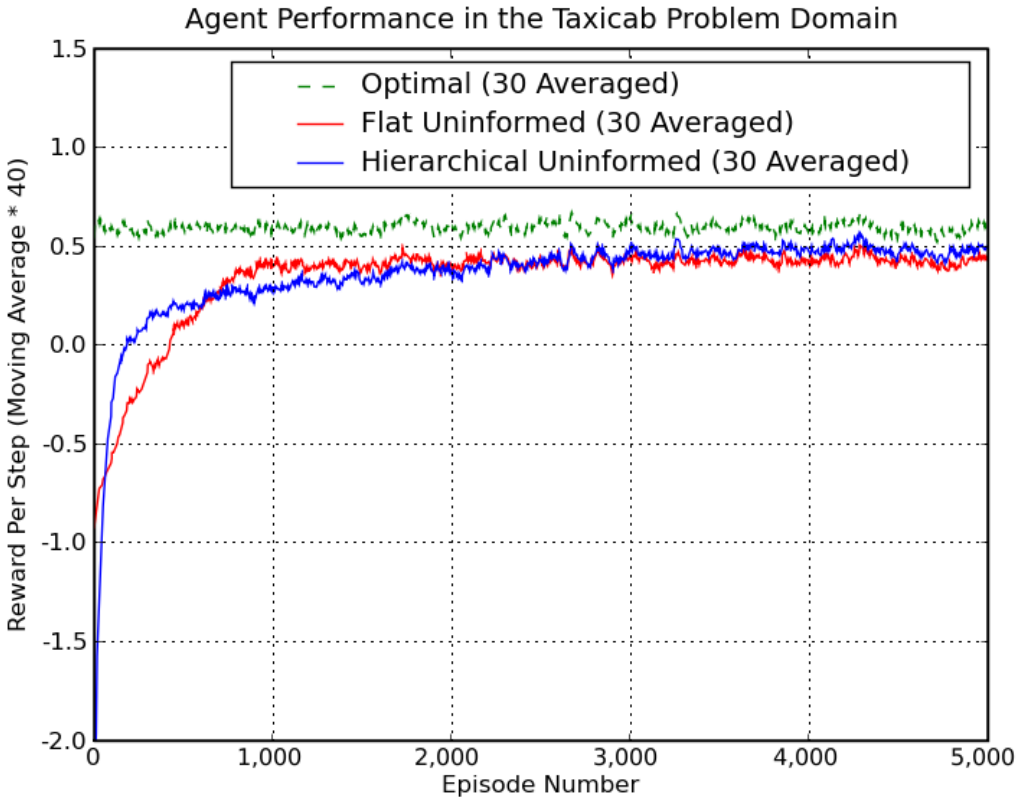
5.2 Informed-Finite Task



After disabling exploration after 50000 episodes, the optimal reward possible over 500 episodes was 0.93 reward per step. The flat omniscient agent averaged 0.16 reward per step and the hierarchical omniscient agent averaged 0.58 reward per step.

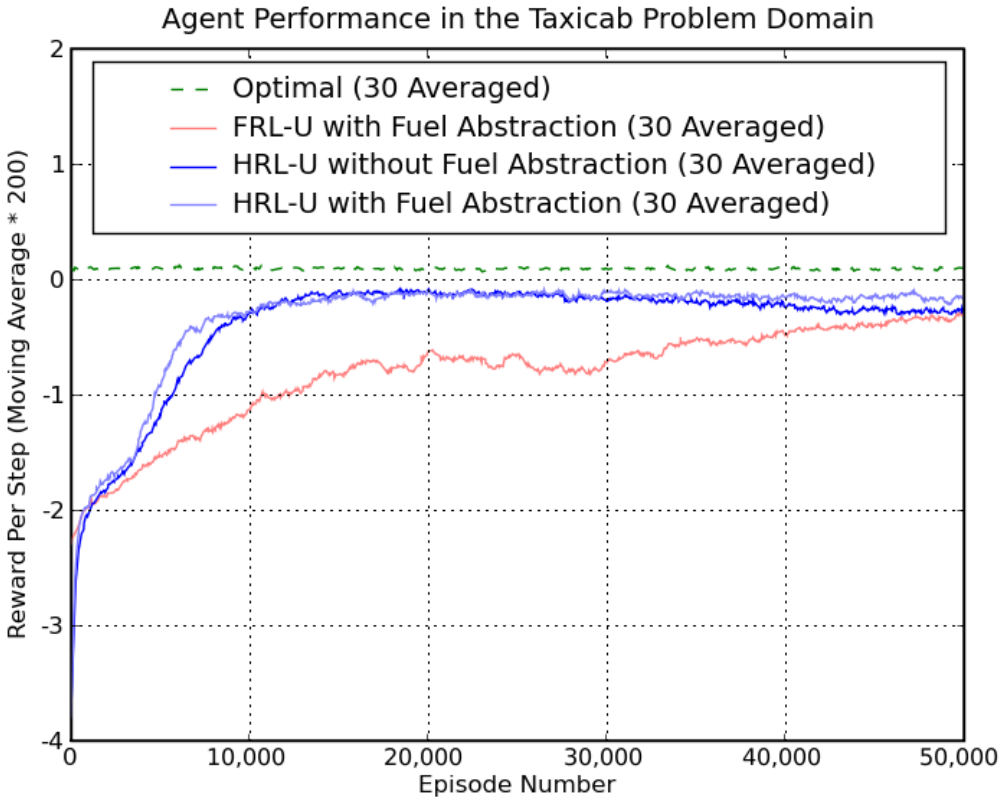
The Soar agents clearly learn faster than Dietterich's agents, though they appear to achieve a lower quality of optimal policy by the end of 50000 runs. It is unclear whether the runs Dietterich presented were exceptional for his system or the norm, but it seems more likely that they were the former rather than the latter.

5.3 Uninformed-Infinite Task



After disabling exploration after 5000 episodes, the optimal reward possible over 500 episodes was 0.58 reward per step. The flat uninformed agent averaged 0.42 reward per step and the hierarchical uninformed agent averaged 0.47 reward per step.

5.4 Uninformed-Finite Task



After disabling exploration after 50000 episodes, the optimal reward possible over 500 episodes was 0.10 reward per step. With fuel abstraction[5], the flat uninformed agent averaged -0.29 reward per step and the hierarchical uninformed agent averaged -0.16 reward per step.

Without fuel abstraction, the hierarchical uninformed agent was still able to achieve -0.27, slightly edging out the flat uninformed agent with fuel abstraction. The flat agent was unable to succeed at the task without fuel abstraction or some other aid.

It is important to note the increased difficulty of this task. The uninformed tasks are no longer deterministic from the point of view of the agent. Coupled with the incredible length of a successful episode, it is considerably more difficult for the agent to settle on paths that are both resistant to failure and near optimal.

5 This corrects an error in the presentation of this data at Soar Workshop 29.

7. Discussion

The Soar agents I designed do fairly well in the four tasks described in section 3. Dietterich's MAXQ hierarchy performs well, as expected. For the uninformed tasks, it was necessary to add information about which source cells have been searched. Without this information, behavior resembling a random walk developed in ambiguous situations. Additionally, it was necessary to change MaxGet to MaxTryGet in order to allow the agent to search for the passenger. Given these changes, both uninformed agents perform very well on the uninformed-infinite task. Both agents perform reasonably well on the uninformed-finite task given a fuel abstraction, but only the hierarchical agent is able to learn the task without some sort of aid.

I was able to reproduce the improvement in learning speed as demonstrated by Dietterich [1998], but the quality of the learned policy seems to be slightly lower. However, the learning can be reproduced reliably. The hierarchical agents perform better than the flat agents across the board.

Future work includes the development of techniques for automatically generating hierarchies for reinforcement learning agents. Additionally, might be useful to experiment with automatic hierarchy flattening. If an agent could detect when to dynamically switch from a hierarchical policy to a flat policy, it might be possible to learn a more general policy before learning more specialized policies, improving both the efficiency of learning and the quality of the end result.

Additional automation of temperature selection and temperature reduction would both decrease the amount of tinkering necessary to achieve good performance and improve robustness of reinforcement learning systems. The agents exploring these tasks seem to be have a fairly narrow window of temperatures which allow learning to progress. How this window evolves as it learns is difficult to judge. Finding the window initially required a stab in the dark.

Citations

1. Derbinsky, N., Gorski, N., Laird, J. E., Marinier, B., and Wang, Yongjia. (2009). Soar-RL Manual Version 1.0.1.
2. Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. Journal of Artificial Intelligence Research, 13, 227-303.
3. Dietterich, T. G. (1998). The MAXQ Method for Hierarchial Reinforcement Learning. In Proceedings of the Fifteenth International Conference on Machine Learning.
4. Laird, J. E., Congdon, C.B., and Coulter, K. J. (2008). The Soar User's Manual Version 9.0.
5. Sutton R. S., Barto A. G. (1998). Reinforcement Learning: An Introduction.