Model-Based and Model-Free Learning in Markovian and Non-Markovian Environments

Kevin Sikorski and Tucker Balch

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

ks || trb@cs.cmu.edu

ABSTRACT

Model-based reinforcement learning methods such as Dyna and Prioritized Sweeping are known to find solutions for deterministic Markov Decision Problems (MDPs) with fewer training experiences than model-free methods such as Q-learning. Do model-based methods provide an advantage in stochastic MDPs and non-Markovian domains as well? To examine this question, we evaluate the performance of Dyna and Prioritized Sweeping, both model-based algorithms, and Q-learning, a model-free technique, in three domains: *(1)* an MDP with deterministic transitions, *(2)* an MDP with stochastic transitions, and *(3)* simulated foraging, a non-Markovian mobile robotics domain. We also introduce *sequence compression*, a novel technique that improves the performance of Q-learning in a non-Markovian domain.

Keywords

Reinforcement Learning, Markovian, non-Markovian.

1. INTRODUCTION AND BACKGROUND

Most reinforcement learning problems can fit in the same general framework, depicted in Figure 1 (Kaelbling, Littman, & Moore, 1996). In this model the learner's goal is to move through its environment, maximizing its reward over its lifespan. At particular points in time, the agent senses all or part of the state of the environment, and then performs the action it considers to be the most appropriate in its current state. These actions often cause a transition from one state to another, effectively moving the agent around the state-space. After performing an action, the agent receives a reward, indicating how appropriate the previous action was. The cycle continues, with the agent sensing the state of the environment, performing a new action, and being rewarded accordingly. This kind of learning problem is a Markov Decision Problem (MDP) if the environment changes state according to the Markov Property (see below).

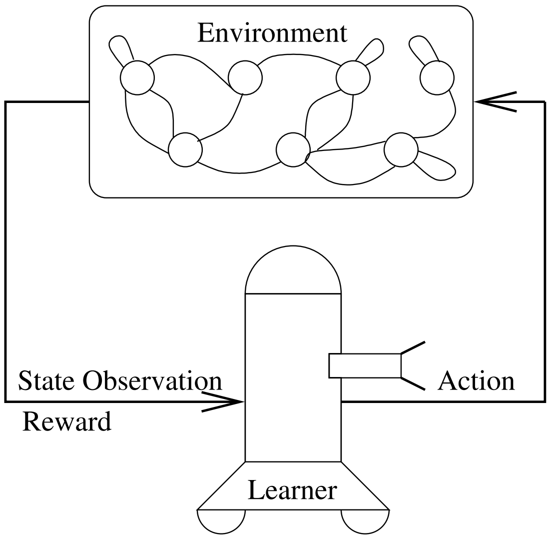

Figure 1: The standard reinforcement learning model (Kaelbling, Littman, & Moore, 1996).

Typically, learners do not maximize their immediate reward; rather, they act to maximize a total discounted reward over an extended period. Because the agent does not know when its life will end (i.e., when the computer simulation comes to a halt, or the robot's battery is drained, etc), or that taking a particular action in a particular state will actually bring about the large reward it has previously experienced (due to stochastic state transitions or rewards), potential future rewards are discounted, as they may not be realized.

Some reinforcement learning techniques explicitly model some aspect of the environment in which they are working (Sutton, 1991; Moore & Atkeson, 1993). For example, Dyna and Prioritized Sweeping explicitly learn a model of the environment's transitions based on the frequency of transitions the agent experiences (Sutton, 1991). They also model how reward is provided in response to actions the learning agent takes as it moves through the state space. These learning techniques are referred to as "Model-Based". Q-learning is an example of a "Model-Free" learning technique: although it does build a table of Q-values based on its trajectory and

AGENTS'01 Workshop on Learning Agents, May 28-June 1, 2001, Montréal, Quebec, Canada.

the rewards it receives, it does not create an explicit model of state transitions or rewards, nor can any such model be recovered from the Q-table.

Machine learning problems are classified as "Markovian" if they possess the Markov Property, and "non-Markovian" otherwise. For a reinforcement learning problem this means the state transition probabilities and rewards for performing a particular action in a particular state must depend only on the agent's current state. For example, if the agent is currently in state S_1, it does not matter if it arrived there by a transition coming from S_2 or S_3; the state transition probabilities and rewards associated with each action available to the agent in S_1 must be the same no matter what states the agent was in before it arrived at S_1. We say that problems that possess the Markov Property are "Markovian."

In an MDP no state is "hidden" from the agent, that is, the agent is aware of all states that it could conceivably experience in the environment. This has important implications for the formulation, or representation, of the problem. For example, consider an agent presented with a maze consisting of a start state S, a goal state G, and several states on the path between S and G. Suppose also that in any state, the agent can switch to operating mode A or B as an action. The agent will receive a reward in the goal state if and only if it is in operating mode B. If the problem is formulated such that the agent only keeps track of its position within the maze, but ignores its operating mode, then the domain is non-Markovian. If the problem is formulated such that two states are maintained for each possible position within the maze, one signifying that the agent is in operating mode A and the other operating mode B, then it is Markovian. Two important classes of non-Markovian representations include the Partially Observable Markov Decision Problems (POMDPs) and Hidden Markov Models (HMMs) (Meuleau, et al, 1999; Rabiner, 1989).

The remainder of this paper is organized as follows. First we introduce the learning methods used in the work. These are the standard formulations of Q-learning, Dyna, and Prioritized Sweeping. Next we examine their performance in each of three domains. Finally, we introduce and demonstrate *sequence compression* as a means for improving learning performance in non-Markovian domains.

2. THE LEARNING METHODS

There are a number of model-free and model-based reinforcement learning methods. For this work we have chosen Dyna and Prioritized Sweeping as a representative model-based approach and Q-learning as a representative model-free approach (Sutton, 1991; Watkins, 1989).

2.1 Q-learning

Q-learning is a model-free learning technique that learns a function $Q(s,a)$, the expected discounted reward resulting from taking an action a in a state s, and then acting optimally afterwards. On each step, the learner receives a tuple (s,a,s',r), where s is the previous state, a is the action, s' is the state the agent transitioned to after taking the action, and r is the resulting reward for that step. The learner uses this tuple to execute the following learning rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)).$$

γ is referred to as the discount rate, and it serves to control how much less future rewards will be valued than rewards on the very next step. α is referred to as the learning rate, and determines how strongly the learner will weight this new experience relative to all its previous experiences. Q-learning can be used with a number of exploration strategies. However, the implementation in this paper uses a decaying ϵ-greedy exploration strategy, taking a random action starting with probability $p = \epsilon$, and with ϵ decaying by a factor of f each time the learner receives a tuple. The values of parameters for Q-learning that we used experimentally are summarized in Table 1.

2.2 Dyna

Q-learning is an effective and well-known learning method, however it often requires many experience tuples before converging to an optimal policy. Another learning algorithm, Dyna, attempts to speed convergence by performing multiple backups (applications of a learning update rule) each time it receives an experience tuple. Dyna is model-based, keeping track of the transition probabilities $T(s,a,s')$ from any state s to any other state s' via an action a, and the average reward $R(s,a)$ gained by performing action a in state s. When it receives a new experience tuple (s,a,s',r), Dyna performs its update rule on the states and actions referenced in that tuple:

$$Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a').$$

After the algorithm has updated $Q(s,a)$, it selects k state-action pairs at random, and applies the update rule:

$$Q(s_k,a_k) \leftarrow R(s_k,a_k) + \gamma \sum_{s'} T(s_k,a_k,s') \max_{a'} Q(s',a').$$

This approach requires a factor of $k+1$ more computational work for each experience with the environment than Q-learning. However, the additional backups serve to propagate new information from poorly-explored states to other, more established states. For example, if a Q-learning agent suddenly found a very high reward state, it would require a large number of additional experience tuples before this information affects the policy in "far" states. For Dyna, depending on the value of k, the extra backups propagate this information to distant states without requiring another experience tuple. In the implementation of Dyna used in this paper, exploration is handled as it is in Q-learning. The values of parameters for Dyna that we used experimentally are summarized in Table 1.

2.3 Prioritized Sweeping

Prioritized Sweeping also attempts to speed convergence by executing extra backups for each experience tuple it receives. However, instead of selecting state-action pairs at random like Dyna, Prioritized Sweeping uses a more directed approach to select states that are in need of updates.

Instead of operating on a table of Q-values, Prioritized Sweeping learns a function $V(s)$, representing the expected discounted future reward of acting optimally in a state s. This method also builds a model of the transition probabilities $T(s,a,s')$ and rewards $R(s,a)$, and uses them to update $V(s)$ whenever it receives an experience tuple. The

Table 1: Summary of parameters used for Q-learning and Dyna in the various domains.

Domain	Learner	α	γ	k	ϵ	t_{bored}	decay-rate
Deterministic	**Q-learning**	0.2	0.9	N/A	0.1	N/A	0.99
	Dyna	N/A	0.9	200	0.1	N/A	0.99
	Pri. Sweeping	N/A	0.9	200	N/A	1	0.99
Stochastic	**Q-learning**	0.2	0.9	N/A	0.1	N/A	0.99999
	Dyna	N/A	0.9	200	0.1	N/A	0.99999
	Pri. Sweeping	N/A	0.9	200	N/A	50	0.99999
Foraging	**Q-learning**	0.2	0.9	N/A	0.1	N/A	0.999
	Dyna	N/A	0.9	200	0.1	N/A	0.9995
Foraging with Sequence Compression	**Q-learning**	0.2	0.9	N/A	0.1	N/A	0.999

algorithm also associates a priority $Pri(s)$ with each state, which is useful in describing which state is most deserving of being the next updated. When receiving a tuple, Prioritized Sweeping performs the following steps:

1. stores the old value $V_{old}(s)$ of state s.
2. updates $V(s)$ according to the rule:

$$V(S) \leftarrow \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s') \right).$$

3. resets the priority of state s to zero.
4. finds how much the value of state s changed due to the update ($\Delta = |V_{old}(s) - V(s)|$).
5. for each predecessor p of state s, updates its priority to $\Delta T(p,a,s)$ if it is not already larger.

Prioritized Sweeping performs the above actions $k+1$ times for each tuple - the first time is to update state s, and the remaining k times to update the state with the largest priority. The end result is that the k extra backups performed are not randomly scattered about the state-action space as done by Dyna. Instead, they are used to propagate the results of "surprising" transitions from the frontiers of exploration to the more established states.

Exploration is handled somewhat differently in Prioritized Sweeping than Q-learning and Dyna. Instead of a decaying ϵ-greedy strategy, the implementation used in this paper employed the "t_{bored}" approach described in (Moore & Atkeson, 1993). Briefly, this approach assumes that if a particular state-action pair has not been tried t_{bored} times it is assumed to lead to an absorbing state with an extremely large reward, otherwise its value is what would normally be computed. This optimistic strategy guides the learner toward unexplored states, eventually sampling each action in each state t_{bored} times.

3. LEARNING IN A DETERMINISTIC MDP

The first environment used to compare the learning methods is a 169-state, deterministic maze, depicted in figure 2. Here, each state corresponds to a square in the maze, where the agent can choose one of four actions: North, South, East or West. If the agent attempts an action that would move it into a wall, it remains in its current state. The exception to this is when the agent is in the goal state: in this case, any action will teleport the agent back to the start state. All state-action pairs yield a reward of -1, except for all actions

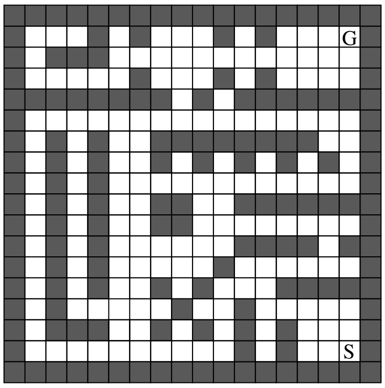

Figure 2: The maze problem used to evaluate the learners. There are 169 possible states in the deterministic MDP. The start and goal states are denoted with an 'S' and a 'G', respectively.

that are taken while in the goal state, which yield a reward of +1. This task is similar to the one investigated by Moore in (Kaelbling, et al, 1996). The size of the problem (number of states) was selected to approximate the size of the other two domains we examine below to enable comparisons between the results.

In experimental evaluations both learners were initialized to zero. The learners were trained over a 1000 trial epoch, with each trial consisting of 1000 steps in the maze. The number of steps, total reward and number of backups were recorded for each 1000 step trial. The agents were allowed to retain training data from trial to trial, but they were cleared for each epoch. Each learner was tested 10 times in the maze (10 epochs).

Performance for both learners in this task is summarized in Figure 3 and Table 2. In this particular maze, optimal performance equates to a total reward of -952 over 1000 steps. As we would expect, based on previous work, Dyna outperforms Q-learning in speed of convergence, reaching its optimum value by trial 4 compared to Q-learning's trial 153 (Kaelbling, et al, 1996). Prioritized Sweeping does marginally better, converging by trial 3. All learners eventually converge to the same level performance. Although the model-based algorithms require significantly less "real world" experience to find the optimal solution, they require 2.5 to 5 times as much work (as measured by number of backups) to

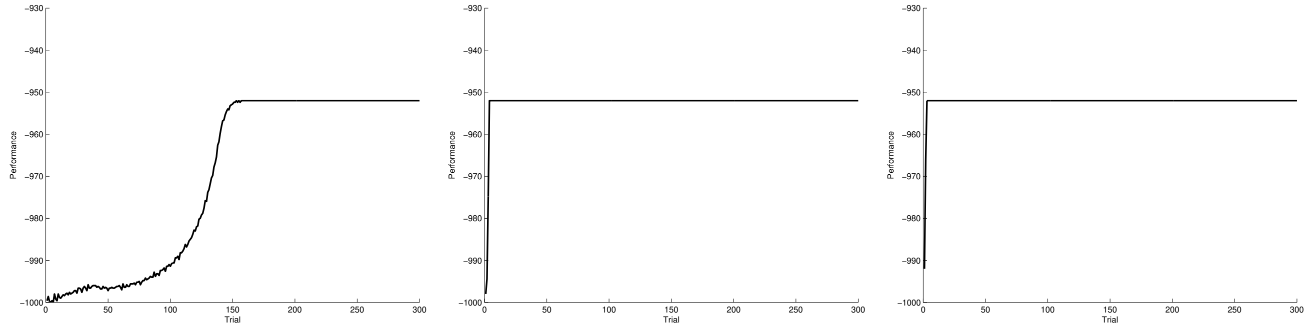

Figure 3: Performance in the deterministic MDP maze task for Q-learning (left), Dyna (center) and Prioritized Sweeping (right) over 300 trials. The results are averaged over 10 trials for each learning algorithm.

Table 2: Summary of Q-learning and Dyna performance in the Deterministic Maze.

	best perf.	on trial #	backups
Q-learning	-952	153	153,000
Dyna	-952	4	804,000
Prioritized Sweeping	-952	3	373,066

converge to this result. There was no change in performance past 300 trials, and so the remaining trials are not shown in the graphs.

4. LEARNING IN A STOCHASTIC MDP

The second environment examined is a 256-state MDP with stochastic transitions, similar to the (simplified) environment depicted in figure 4. In this domain we randomly generated 10 MDPs to test the learners. The MDPs were designed to approximate the same complexity (number of states and actions) as the maze and foraging tasks. Each MDP is composed of a single start state, 16 goal states and 239 intermediate states. For any action a taken in a state s_i, there is a 0.5 probability that the agent remains in state s_i, and a 0.1 probability that the agent is transitioned to one of 5 other states $s_{i,a,1}$ through $s_{i,a,5}$, chosen randomly when the environment is created. All state-action pairs produce a reward of -1, except for those originating in a goal state, which produce a reward of $+1$ and always transition the agent to the start state.

In experimental evaluations all learners were initialized to zero. The learners were trained over a 1000 trial epoch, with each trial consisting of 1000 steps in the environment. The number of steps, total reward and number of backups were recorded for each 1000 step trial. The agents were allowed to retain training data from trial to trial, but they were cleared for each epoch. This was repeated 10 times for both Q-learning and Dyna, and 3 times for Prioritized Sweeping, each time with different environment and random number seed for the agents.

The performance of the two learners is summarized in Figure 4 and Table 3. In early trials, Dyna improves it's performance very quickly. The worst performance possible is -1000 (no positive rewards). Dyna converges to a better policy on average (performance of -789 versus -799.0), but

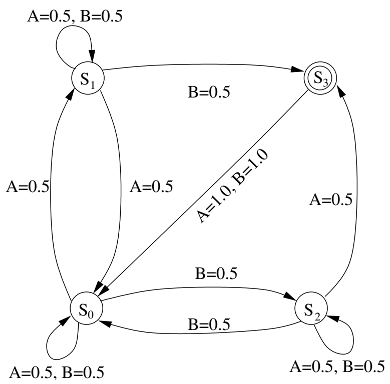

Figure 4: An example 4-state stochastic MDP with two actions and stochastic transitions. The MDPs used in experiments include 256 states with 6 possible actions at each state.

Table 3: Performance in the stochastic MDP for Q-learning and Dyna.

	best perf.	on trial #	backups
Q-learning	-799.20	781	781,000
Dyna	-789.00	985	197,985,000
Prioritized Sweeping	-774.66	524	189,774,385

does so about 200 trials later, on average than Q. Prioritized Sweeping converges even faster than Dyna (the extra choppiness in the Prioritized Sweeping graph is likely due to the fact that only 3 runs were available at the time of writting this document). Also, note that Dyna and Priorotized Sweeping require over 250 times more work (backups) to converge to its best performance. Clearly, in this domain, there is a substantial computational cost for an only slightly better policy.

5. LEARNING IN A NON-MARKOVIAN DP

We use a simulated robot foraging task to investigate learning performance in non-Markovian Decision Processes (NMDPs) (Balch, 2000). In this environment, an agent is

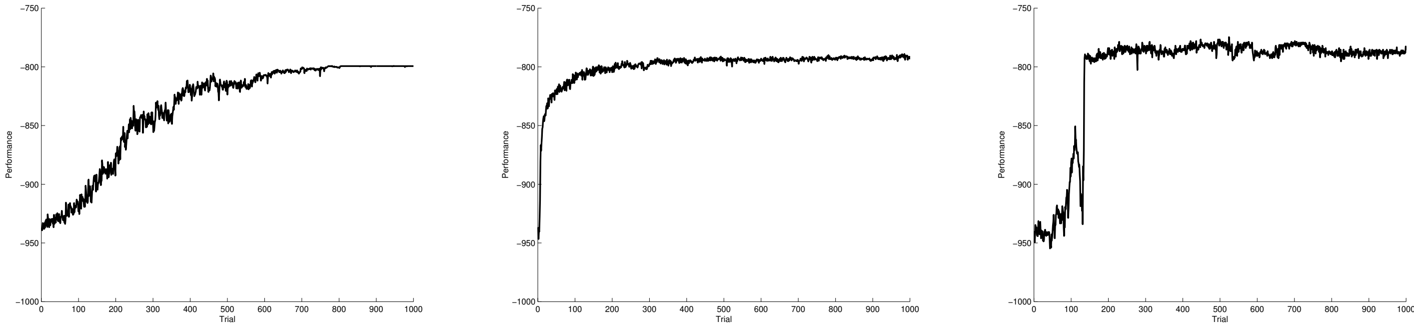

Figure 5: Performance in the stochastic MDP for Q-learning (left), Dyna (center) and Prioritized Sweeping (right).

initialized in the home area, near a blue "bin" and a red "bin". The agent must learn to wander in its environment, pick up blue and red "attractors", and return them to the appropriately colored bin. The task is complicated somewhat by the fact that the robot has a limited field of view, has slightly noisy movements, and does not know its exact position. This task has been explored extensively by one of us (Balch) in simulation and on mobile robots. Space precludes a detailed description of this domain, please see (Balch, 2000) for more information.

We use a combination of behavior-based control and reinforcement learning to solve this task. The learner is provided an abstracted description of the world state, along with six actions from which to select. The actions are behaviors that act in the continuous sensor/actuator space. The problem for learning algorithms in this task is to associate appropriate actions (behaviors) with the current abstracted sensor state.

State is represented as a nine-bit vector (i.e. 512 possible states), with each bit indicating the value of a boolean perceptual feature. Example boolean features include "red-visible," "at-red-bin," and "holding-red" indicating whether the robot sees a read attractor, if it is close to the red bin, or that it is holding a red attractor, respectively. Additional features describe the robot's state with regard to blue attractors and bins. Of the 512 states, only about 250 are reachable.

The agent always has six available actions: wander around the entire environment, wander near the home area, move towards the closest visible red attractor, move towards the closest visible blue attractor, move towards the red bin, and move towards the blue bin. In order to simplify matters, the agent is assumed to have a gripper that automatically picks up attractors when intentionally moving towards and coming close enough to one, and automatically dropping them off when intentionally moving towards and coming close enough to the appropriate color bin.

From the learner's perspective, this environment is non-Markovian. For example, it never knows exactly how far away it is from a bin. If it needs to return a red attractor to the red bin, the probability of transitioning to a state with the "close-to-red-bin" bit set increases the more times it performs the "move towards the red bin" action. However, it may take hundreds of steps before the bin is reached. Because the Markov assumption is violated it is not expected that either of the learners would do well in this domain.

To evaluate their performance in this domain the learners were run for 1000 trials of 3000 steps each. Each epoch was run 10 times for each learner, each time with new random number seeds for the simulator and agent, and a new random initial arrangement of attractors.

Performance results are summarized in Figure 6 and Table 4. Performance does not improve after 300 trials, so the remaining trials are not presented. As predicted, Q-learning does very poorly. Dyna does somewhat better, recovering almost 2 attractors per trial. However, hand-coded policies using the same state/action space in this domain have been found to recover an average of 6 attractors per trial (Balch, 2000). This indicates that there is significant room for improvement. (Prioritized Sweeping is not included in this comparison because results were not available at the time this paper was prepared.)

To address this poor attractor-collecting performance, we briefly introduce a novel technique for accelerating learning in this domain.

6. SEQUENCE COMPRESSION FOR NMDPS

Presumably, it is the non-Markovian nature of the foraging environment that causes the learners such difficulty. After all, none of these methods make any guarantees for non-Markovian domains. We introduce a technique referred to as *sequence compression* to simplify the problem for learners in these domains.

Rather than supply the learner with experience tuples at each and every step of interaction with the environment, sequence compression acts as a middle layer between the learner and the environment. If there is a state transition between one experience tuple and the previous one, then the tuple is passed to the learner, which then learns and acts normally. However, if there was no state transition between one experience tuple and the next, the action that the learner last attempted is repeated, and the learner is not updated in any way. This cycle continues until either a state transition is found with that action, or a pre-set number of attempts is reached, in which case the experience tuple is submitted to the learner, and the process begins anew. This approach is somewhat similar to those investigated by McGovern, and separately by Wang (McGovern, Sutton, & Fagg, 1997; Wang & Mahadevan, 1999).

As an example, the state/action sequence $< s_1, a_0 >< s_2, a_1 >< s_2, a_1 >< s_2, a_1 >< s_3, a_2 >$

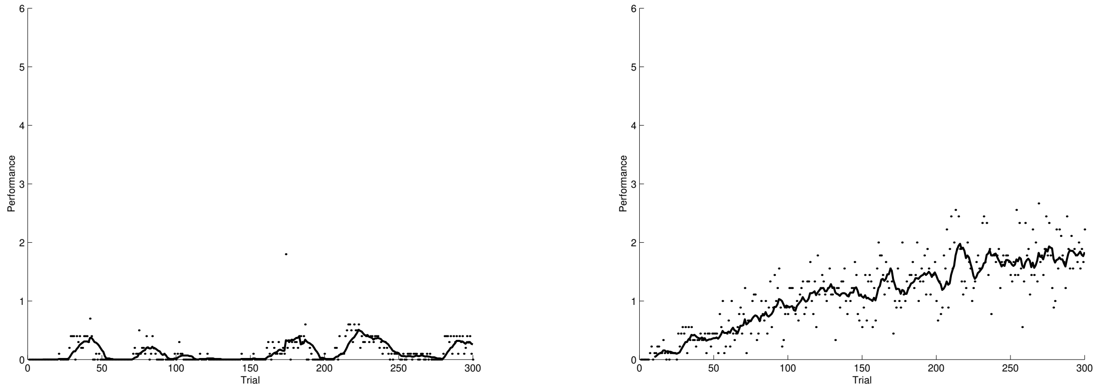

Figure 6: Performance in the foraging domain for Q-learning (left) and Dyna (right). This shows a running average over 10 trials for each learner, as well as the average score at each trial. Data was accumulated over 10 runs for each learning method.

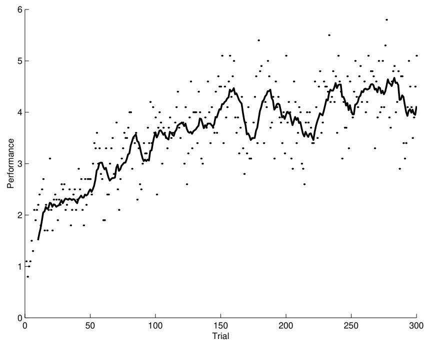

Figure 7: Performance in foraging with sequence compression for Q-learning. This shows a running average over 10 trials, as well as the average score at each trial. Data was accumulated over 5 runs.

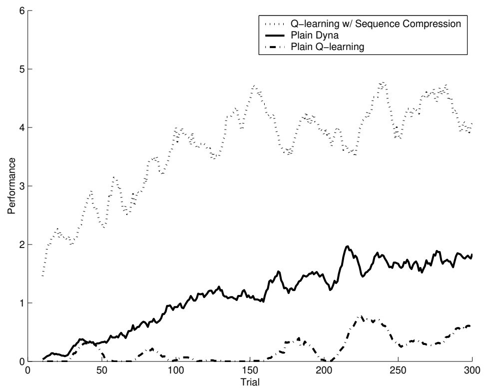

Figure 8: Performance in foraging environment with sequence compression for Q-learning, and without sequence compression for Dyna and Q-learning.

would be perceived by the learner as:
$< s_1, a_0 >< s_2, a_1 >< s_3, a_2 >$

The sequence compressor is thus able to reduce the non-Markovian nature of the environment, so that it can be better modeled as a stochastic MDP. Of course, with sequence compression, the learner does not learn the optimal policy for its environment; instead, it learns the optimal policy for the environment to which sequence compression is exposes the learner. If the two are the similar, we learn a successful policy.

The impact on performance of the Q-learning when provided with sequence compression is significant. Figure 7 shows the average performance of the learners with sequence compression. These tests were run just as in the standard foraging environment, with the sequence compressor set for a maximum of 100 attempts to transition to a new state. Clearly, Q-learning augmented with sequence compression can outperform traditional Dyna. The effects of sequence compression are even more obvious in figure 8.

7. CONCLUSION

We evaluate model-based and model-free reinforcement learning algorithms in deterministic Markov Decision Problems (MDPs), stochastic MDPs and non-Markovian Decision Problems (NMDPs). Model-free and model-based learning approaches (Q-learning, Dyna, and Prioritized Sweeping) were demonstrated to be effective in the deterministic MDP and a stochastic MDP. However, Q-learning and Dyna were shown to be less effective in a non-Markovian foraging environment. To address this, we introduced *sequence compression*, a novel technique for abstracting the environment so that the learner is presented a "more Markovian" problem to deal with. With the use of sequence compression, Q-learning performs far better in the non-Markovian environment.

In future work, we plan to continue the examination of Prioritized Sweeping in these environments, and to thoroughly explore the benefits and limitations of sequence compression. We will also examine the theoretical implications of this new approach.

Acknowledgements

The authors would like to thank Manuela Veloso, Andrew Moore, Hans Moravec, and Will Uther for their invaluable contributions to this paper. This research was sponsored by the United States Army under Grant No DABT63-99-1-0013. The content of this publication does not necessarily reflect the position or the policy of the sponsors and no official endorsement should be inferred.

References

Balch, T. (2000). Hierarchic Social Entropy: An Information Theoretic Measure of Robot Group Diversity. *Autonomous Robots 8* (pp. 209-237). Kluwer Academic Publishers, The Netherlands

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research 4* (pp. 237-285). AI Access Foundation and Morgan Kaufmann Publishers.

McGovern, A., Sutton, R. S., Fagg, A. H. (1997). Roles of macro-actions in accelerating reinforcement learning. *Proceedings of the 1997 Grace Hopper Celebration of Women in Computing* (pp. 13-17).

Meuleau, N., Peshkin, L., Kim, K. E., & Kaelbling, L. P. (1999). Learning Finite-State Controllers for Partially Observable Environments. *Proceedings of the Fifteenth International Conference on Uncertainty in Artificial Intelligence.*

Watkins, C. J. C. H. (1989). Learning from Delayed Rewards. PhD thesis, King's College, Cambridge, UK.

Sutton, R. S. (1991). Planning by incremental dynamic programming. *Proceedings of the Eighth International Workshop on Machine Learning* (pp.353-357). Morgan Kaufmann.

Moore, A. W., Atkeson, C. G. (1993). Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning, Volume 13.*

Wang, G., Mahadevan, S. (1999). Hierarchical Optimization of Policy-Coupled Semi-Markov Decision Processes. *Proceedings of the 16th International Conference on Machine Learning.*

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE, Vol. 77* (pp. 257-286).